INTERNATIONAL STANDARD

ISO 10303-235

First edition
2009-09-01

Industrial automation systems and integration — Product data representation and exchange —

Part 235: Application protocol: Engineering properties for product design and verification

Systèmes d'automatisation industrielle et intégration — Représentation et échange de données de produits —

Partie 235: Protocole d'application: Propriétés d'ingénierie pour la conception de produits et vérification

Reference number
ISO 10303-235:2009(E)

ISO copyright office
Case postale 56 • CH-1211 Geneva 20
Tel. + 41 22 749 01 11
Fax + 41 22 749 09 47
E-mail copyright@iso.org
Web www.iso.org

Published in Switzerland

Contents

Page

Figures

Tables

Foreword

ISO (the International Organization for Standardization) is a worldwide federation of national standards bodies (ISO member bodies). The work of preparing International Standards is normally carried out through ISO technical committees. Each member body interested in a subject for which a technical committee has been established has the right to be represented on that committee. International organizations, governmental and non-governmental, in liaison with ISO, also take part in the work. ISO collaborates closely with the International Electrotechnical Commission (IEC) on all matters of electrotechnical standardization.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 2.

The main task of technical committees is to prepare International Standards. Draft International Standards adopted by the technical committees are circulated to the member bodies for voting. Publication as an International Standard requires approval by at least 75 % of the member bodies casting a vote.

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO shall not be held responsible for identifying any or all such patent rights.

ISO 10303-235 was prepared by Technical Committee ISO/TC 184, *Automation systems and integration*, Subcommittee SC 4, *Industrial data*.

ISO 10303 is organized as a series of parts, each published separately. The structure of ISO 10303 is described in ISO 10303-1.

Each part of ISO 10303 is a member of one of the following series: descriptive methods, implementation methods, conformance testing methodology and framework, integrated generic resources, integrated application resources, application protocols, abstract test suites, application interpreted constructs and application modules. This part of ISO 10303 is a member of the application protocols series.

A complete list of the parts of ISO 10303 is available from the Internet:

http://www.tc184-sc4.org/titles/STEP_titles.htm .

Introduction

ISO 10303 is an International Standard for the computer-interpretable representation of product information and for the exchange of product data. The objective is to provide a neutral mechanism capable of describing products throughout their life cycle. This mechanism is suitable not only for neutral file exchange, but also as a basis for implementing and sharing product databases, and as a basis for archiving.

This part of ISO 10303 is a member of the application protocol series. This part of ISO 10303 specifies an application protocol (AP) for those properties of products that can be used for product design and design validation.

This application protocol defines the context, scope, and information requirements for properties of products that can be used for product design and design validation, the testing, measurement and approval processes used to determine those properties and specifies the integrated resources necessary to satisfy these requirements.

Application protocols provide the basis for developing implementations of ISO 10303 and abstract test suites for the conformance testing of AP implementations.

Clause 1 defines the scope of this part of ISO 10303 and summarizes the functionality and data covered by this part of ISO 10303. Clause 3 lists the words defined in this part of ISO 10303 and gives pointers to words defined elsewhere. An application activity model that is the basis for the definition of the scope is provided in Annex F. The information requirements for the application are specified in Clause 4, using terminology appropriate to the application. A graphical representation of the information requirements, referred to as the application reference model, is given in Annex G.

Resource constructs are interpreted to meet the information requirements of this application and produce the application interpreted model (AIM). This interpretation, given in 5.1, shows the correspondence between the information requirements and the AIM. The short-listing of the AIM specifies the interface to the integrated resources and is given in 5.2. Note that the definitions and EXPRESS provided in the integrated resources for constructs used in the AIM can include items in select lists and subtypes that are not imported into the AIM. The expanded listing given in Annex A contains the complete EXPRESS for the AIM without annotation. A graphical representation of the AIM is given in Annex H. Additional requirements for specific implementation methods are given in Annex C.

Engineering properties, which include materials properties, are not fundamental constants derived from physical or chemical laws. The value of an engineering property of a product is dependent on the process used to measure the property value and on the conditions used in that process.

If properties that are based on fundamental physical or chemical behaviour, such as latent heat or melting temperature, are measured by different methods, then the results obtained are usually sufficiently similar to be regarded as a single value. A method used for measuring an engineering property attempts to simulate the behaviour of a product in an engineering situation in the real world. Each aspect of behaviour, for example the hardness of a product, can be simulated by several different methods. The methods are usually designed to be convenient to use and to provide a consistent result from repeated measurements. However, the difference between physical or chemical properties of a substance and the engineering properties of a product is that if different methods are used to measure an engineering property, then different results are obtained. For example, the measurement of the elongation property attempts to provide a numerical value to represent the engineering concept of plastic ductility by stretching a specially shaped sample of a product by applying a uniaxial tensile load. The value of the elongation property is determined as a percentage of the original length of a portion of a sample piece of the product. Comparisons between values of the elongation property for

different products are therefore only possible if the fixed length was the same for each case. It is therefore necessary to state this length explicitly for all values of the elongation property.

An engineering property is therefore the result from operating a specific test method in a specific manner and it is necessary to associate the value of an engineering property with the conditions in which it is valid, in order for the meaning of the value to be explicitly determined. This additional information is called the data environment in ISO 10303-45. An alternative term that is often used is metadata, i.e. data about data.

In most communications of engineering data, the relationship of a property value to its data environment or metadata is often an implicit assumption and it might not be explicitly associated with the value. The purpose of this part of ISO 10303 is to provide the means to associate a property value explicitly to the conditions in which it was measured, and thus provide an audit trail to the origins of data values that can be used in product design.

In order to measure the properties of a product, it is sometimes, but not always, possible to test the whole product. Accordingly, a sample of a product can be taken to represent the bulk of the product and the procedure for taking this sample can be specified in some regulatory document, such as a quality manual, or in a standard. The operation of the testing apparatus and the measurement procedure can require that the item that is tested has a specific shape and dimensions, and it will be necessary to create this from the product sample by some manufacturing process. The result of this process can be called a test piece. The specific shape and dimensions of test pieces can also be defined in standards or other regulatory documents. The measurement of the engineering property is then carried out on the test piece by means of some measuring apparatus or testing machine, whose operation might have to be controlled to be within specified limits. Manufactured products can be assemblies or single products, but they are rarely homogenous or isotropic in their properties, and so it is necessary to know the relationships between the test piece, the sample and the original product if the results of the measurement are to be related to the original product.

Data produced by a testing or measurement process is rarely used in its original form. It is necessary to first evaluate data values by some process in order to determine if the conditions prescribed for a particular test method have been met. For many properties, such as fracture toughness as an example, the validity of a test result can only be determined by an evaluation process after all the measurements have been completed and the process is specified in the standard that describes how to make the measurement. Data values are rarely used as single values, but can be combined or processed in some way to provide a collective result that is indicative of the results of a series of measurements. It is necessary to identify the central value of the collection and to provide the uncertainty associated with this value.

The validity of a test result can be established by an approval procedure which results in the issue of a certificate. The certificate affirms that the original product from which the sample was taken conforms to a particular requirement or specification, and that the tests used to determine this were carried out in an approved manner. The data obtained from a valid test can also be subject to a further approval procedure that confirms the suitability of property values for the design of a functional product. This procedure will use criteria for the approval based on the requirements that the product has to satisfy. The approval process and the criteria can be established and administered by an independent regulatory body or authority.

Test data values are not used for design because they often represent a condition of failure of the test piece. Design values are derived from the test data to represent a condition in which it is safe to use the product, and it is also advisable to record explicitly the procedures by which a design value is derived. Further testing can be required to measure the design values.

The number of different engineering properties and test methods is too large for every property and test method to be included in this part of ISO 10303. There are also differences in test methods, and therefore differences in the engineering meaning of the properties, between different national engineering systems. Provision has therefore been made for the names of test methods and their assocation with particular properties to be defined in computer-processable dictionaries conforming to ISO 13584 Parts Libraries, or defined in a referenced document. An entry in such a dictionary can be referenced from the information model in this part of ISO 10303, in order to make use of a particular property name associated with a particular measurement method.

The benefit of this approach is that, with appropriate dictionaries to define test methods and their relevant property names, this part of ISO 10303 can be used for the representation of any engineering property measured by any method, provided that those methods and properties are defined in a computer-processable

dictionary. The application of this part of ISO 10303 therefore extends to other engineering domains and is not restricted to materials. Other applications could include the results of measurements of environmental data, for example.

Figure 1 shows a high-level view of the concept of this part of ISO 10303. Figure 2 shows the high-level view of a process. Further information on the application of product data technology to materials information, as examples of engineering properties, can be found in Reference [13] in the Bibliography.

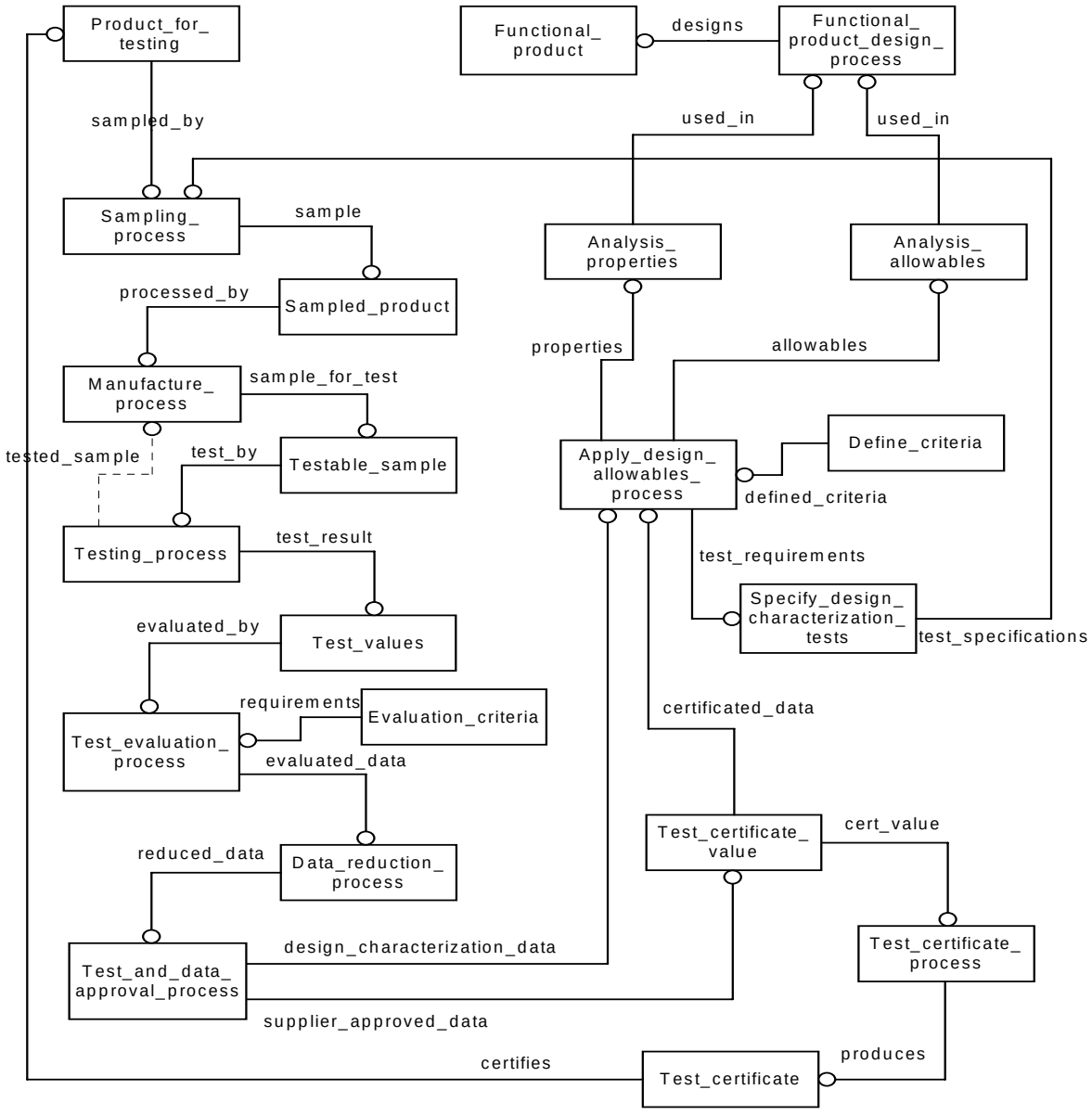


Figure 1 — Processes for the measurement and approval of engineering properties

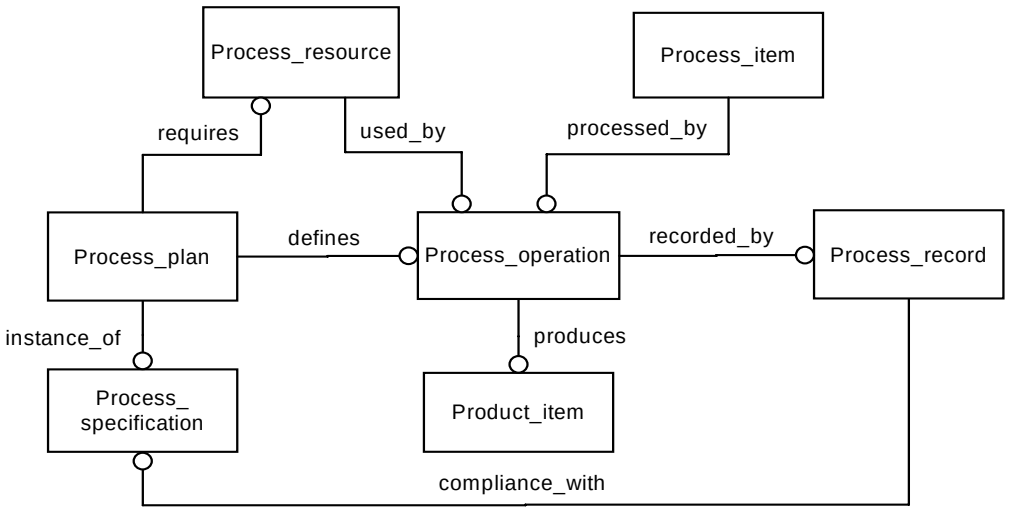


Figure 2 — Generic model for a process

Industrial automation systems and integration — Product data representation and exchange —

Part 235:
Application protocol: Engineering properties for product design and verification

1 Scope

This part of ISO 10303 specifies the use of the integrated resources necessary for the scope and information requirements for the representation of engineering property data that are used for product design and product validation.

NOTE 1 The application activity model in Annex F provides a graphical representation of the processes and information flows that are the basis for the definition of the scope of this part of ISO 10303.

The following are within the scope of this part of ISO 10303:

— descriptions and definitions of the manufactured product, the sample of the product and the testable version of the sample;

— description of the composition and substance of the product;

— description of the processes used in the measurement;

— descriptions of the data values produced by the measurement, with the specification of the conditions in which the data is valid;

— references to standards and other documents wherein sampling, measurement and other details of testing and measurement processes can be specified or described;

— descriptions and qualifications of the personnel and or organizations responsible for the measurement;

— specification of the requirements, conditions and tolerances to be satisfied in the measurement and a description of the outcome;

— descriptions of the locations of the measurement process and the effectivity of the results;

— descriptions of the approval that establishes the validity of the measurements and the use of the properties for product design and design validation.

NOTE 2 Data representations described in this part of ISO 10303 might need to be archived to meet legal and regulatory requirements and to meet quality objectives.

The following are outside the scope of this part of ISO 10303:

— data describing rules, guidelines and expert knowledge in the testing of products;

— names of properties and test methods;

— data describing why a decision was made to use a particular process;

— scheduling data for measurement processes;

— algorithms used for data evaluation and data processing.

NOTE 3 The names and definitions of properties and test methods are assumed to be provided in computer-processable dictionaries, conforming to ISO 13584 Parts Libraries, which classify measurement methods and their associated property types.

2 Normative references

The following referenced documents are indespensible for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

ISO 639-2, *Codes for the representation of names of languages - Part 2: Alpha-3 code.*

ISO 3166-1, *Codes for the representation of names of countries and their subdivisions - Part 1: Country codes.*

ISO 10303-1, *Industrial automation systems and integration — Product data representation and exchange — Part 1: Overview and fundamental principles*.

ISO 10303-21 *Industrial automation systems and integration — Product data representation and exchange — Part 21: Implementation methods: Clear text encoding of the exchange structure.*

ISO 10303-31, *Industrial automation systems and integration — Product data representation and exchange — Part 31: Conformance testing methodology and framework: General concepts*.

ISO 10303-41, *Industrial automation systems and integration — Product data representation and exchange — Part 41: Integrated generic resource: Fundamentals of product description and support.*

ISO 10303-42, *Industrial automation systems and integration — Product data representation and exchange — Part 42: Integrated generic resource: Geometric and topological representation.*

ISO 10303-43, *Industrial automation systems and integration — Product data representation and exchange — Part 43: Integrated generic resource: Representation structures.*

ISO 10303-45, *Industrial automation systems and integration — Product data representation and exchange — Part 45: Integrated generic resource: Material and other engineering properties*.

ISO 10303-47, *Industrial automation systems and integration — Product data representation and exchange — Part 47: Integrated generic resource: Shape variation tolerances*.

ISO 10303-49, *Industrial automation systems and integration —Product data representation and exchange — Part 49: Integrated generic resources: Process structure and properties*.

ISO 10303-50, *Industrial automation systems and integration — Product data representation and exchange — Part 50: Integrated generic resource: Mathematical constructs*.

ISO 10303-54 *Industrial automation systems and integration — Product data representation and exchange — Part 54: Integrated generic resource: Classification and set theory*.

ISO 10303-56 *Industrial automation systems and integration — Product data representation and exchange — Part 56: Integrated generic resource: State*.

ISO 10303-519, *Industrial automation systems and integration — Product data representation and exchange — Part 519: Application interpreted construct: Geometric tolerances*.

ISO 13584-20, *Industrial automation systems and integration — Parts library — Part 20: Logical resource: Logical model of expressions.*

ISO 13584-26, *Industrial automation systems and integration — Parts library: — Part 26: Logical resource: Information supplier identification.*

ISO 13584-42, *Industrial automation systems and integration — Parts library — Part 42: Description methodology: Methodology for structuring part families.*

3 Terms and definitions

3.1 Terms defined in ISO 10303-1

For the purposes of this document, the following terms defined in ISO 10303-1 apply:

— application;

— application activity model (AAM);

— application context;

— application interpreted model (AIM);

— application object;

— application protocol (AP);

— application reference model (ARM);

— conformance class;

— conformance requirement;

— data;

— data exchange;

— generic resource;

— information;

— information model;

— presentation;

— product;

— product data;

— product information;

— product information model;

— structure;

— unit of functionality.

3.2 Terms defined in ISO 10303-31

For the purposes of this document, the following terms defined in ISO 10303-31 apply:

— conformance.

3.3 Terms defined in ISO 10303-45

For the purposes of this document, the following terms defined in ISO 10303-45 apply:

— data environment.

3.4 Other terms and definitions

For the purposes of this document, the following terms and definitions apply:

3.4.1
material product
physical object that is manufactured to a specification and from which other objects can be manufactured

3.4.2
sample
portion of a material product that is removed by a specified procedure to ensure that the portion is representative of the whole material product

3.4.3
test method
test
procedure designed to measure characteristics of a material product in specified conditions

3.4.4
test specimen
physical object that is manufactured from a sample in order to be used in a test

NOTE The manufacture, shape and dimensions of a test specimen can be specified for the test method.

3.4.5
tested specimen
test specimen after it has been used in a test

NOTE A tested specimen can be used in further tests.

4 Information requirements

This clause specifies the information required for the representation and exchange of engineering properties and the processes and conditions in which the values were measured.

The information requirements are specified as a set of units of functionality (UoF) and application objects. A unit of functionality specifies the collection of application objects that are needed to satisfy a general information requirement. The information requirements are defined using the terminology of the subject area of this part of ISO 10303.

NOTE 1 A graphical representation of the information requirements is given in annex G.

NOTE 2 The information requirements correspond to those of the activities identified as being within the scope of this part of ISO 10303 in annex F.

NOTE 3 The mapping specification specified in 5.1 shows how the integrated resources and application interpreted constructs are used to meet the information requirements of this part of ISO 10303.

4.1 Units of functionality

This subclause specifies the units of functionality for engineering properties for product design and validation. This part of ISO 10303 specifies the following units of functionality:

— activity;

— adminstration;

— approval;

— condition;

— document management;

— effectivity;

— external reference;

— geometry;

— geometric tolerance;

— location;

— measure;

— person organisation;

— product;

— properties;

— requirement;

— state;

— substance;

— tolerance datum.

The units of functionality and a description that each UoF supports are given below. The application objects included in the UoFs are defined in Clause 4.2

4.1.1 activity UoF

The activity UoF enables the description of the operations, input items, output items and equipment needed for a process. The activity UoF uses the following application objects:

— Activity_as_planned;

— Activity_as_realised;

— Activity_relationship;

— Activity_type;

— Activity_type_defined_in_external_reference;

— Activity_type_relationship;

— Activity_type_specified;

— Individual_activity;

— Individual_activity_assignment.

4.1.2 adminstration UoF

The administration UoF enables the description of the administration of the testing and measurement processes. The adminstration UoF uses the following application objects:

— Calendar_date;

— Contract;

— Contract_assignment;

— Date_time;

— Event;

— Event_relationship;

— Local_time;

— Project;

— Specification;

— Specification_relationship;

— Specification _type;

— Time_offset.

4.1.3 approval UoF

The approval UoF enables the description of the approval and certification of the data used and generated in the testing and measurement processes. The approval UoF uses the following application objects:

— AP_qualification_type;

— Approval;

— Approval_assignment;

— Approval_relationship;

— Approval_status;

— Approving_person_organisation;

— Certificate;

— Certificate_assignment;

— Security_classification.

4.1.4 condition UoF

The condition UoF enables the description of the conditions that shall be satisfied in a testing and measurement process. The condition UoF uses the following application objects:

— Condition;

— Condition_assignment;

— Condition_evaluation;

— Condition_evaluation_assignment;

— Condition_relationship.

4.1.5 document management UoF

The document management UoF enables the description of documents used to support the testing and measurement process and store the results. The document management UoF uses the following application objects:

— Digital_document_definition;

— Digital_file;

— Digital_record;

— Document;

— Document_assignment;

— Document_definition;

— Document_definition_relationship;

— Document_location_identification;

— Document_relationship;

— Document_version;

— External_item_identification;

— External_source_identification;

— File;

— File_location_identification;

— File_relationship;

— Hard_copy;

— Physical_document_definition.

4.1.6 effectivity UoF

The effectivity UoF enables the conditions to be described in which data is effective. The effectivity UoF uses the following application objects:

— Dated_effectivity;

— Effectivity;

— Effectivity_assignment;

— Effectivity_relationship;

— Frequency_interval;

— Lot_effectivity;

— Serial_effectivity;

— Time_interval;

— Time_interval_effectivity;

— Time_interval_relationship;

— Time_interval_with_bounds.

4.1.7 external reference UoF

The external reference UoF enables references to external sources of information to be described. The external_reference UoF uses the following application objects:

— Alias_identification;

— External_library _reference;

— Language;

— Multi_language_string;

— PLIB_class_reference;

— PLIB_property_reference;

— String_with_language.

4.1.8 geometry UoF

The geometry UoF enables the shape and orientation of physical objects in the testing and measurement processes to be described. The geometry UoF uses the following application objects:

— Axis_placement;

— Axis_placement_mapping;

— Axis_placement_transformation_mapping;

— Cartesian_point;

— Cartesian_transformation_2D;

— Cartesian_transformation_3D;

— Direction;

— Geometric_coordinate_space;

— Geometric_model;

— Geometric_model_element;

— Shape;

— Shape_defined_in_external_reference;

— Shape_dimension;

— Shape_element.

4.1.9 geometric tolerance UoF

The geometric tolerance UoF enables the specification of the permitted deviation from a nominal or ideal geometrical shape to be described. The geometric tolerance UoF uses the following application objects:

— Angularity_tolerance;

— Coaxiality_tolerance;

— Concentricity_tolerance;

— Cylindricity_tolerance;

— Flatness_tolerance;

— Geometric_tolerance;

— Geometric_tolerance_relationship;

— Parallelism_tolerance;

— Perpendicularity_tolerance;

— Position_tolerance;

— Reference_datum;

— Roundness_tolerance;

— Straightness_tolerance;

— Symmetry_tolerance;

— Tolerance_zone;

— Tolerance_zone_definition;

— Total_runout_tolerance.

4.1.10 location UoF

The location UoF enables the place where an event occurred to be described. The location UoF uses the following application objects:

— Address_location_specification;

— Global_location_representation;

— Location;

— Location_representation;

— Location_representation_relationship;

— Organisation_location_representation;

— Organisational_location_identification;

— Product_location_representation;

— Regional_grid_location_representation.

4.1.11 measure UoF

The measure UoF enables the values of quantities to be described. The measure UoF uses the following application objects:

— Context_dependent_unit;

— Count_measure;

— Derived_unit;

— Derived_unit_element;

— Descriptive_measure;

— Duration;

— Maths_expression_value;

— Measure_item;

— Numerical_measure_value;

— Qualified_expression;

— Qualified_measure;

— Qualifier_precision;

— Qualifier_type;

— Qualifier_uncertainty;

— Range_measure;

— Ratio_measure;

— Uncertainty_expanded;

— Uncertainty_qualitative;

— Uncertainty_standard;

— Unit;

— Unit_relationship.

4.1.12 person organisation UoF

The person organisation UoF enables the description of persons and organisations. The person organisation UoF uses the following application objects:

— Address;

— Facsimile_address;

— Network_address;

— Organisation;

— Organisation_relationship;

— Organisation_type;

— Person;

— Person_in_organisation;

— Postal_address;

— Qualification;

— Qualification _relationship;

— Telephone_address.

4.1.13 product UoF

The product UoF enables the description of the products to which the results of the testing and measurement processes apply. The product UoF uses the following application objects:

— Individual_product;

— Individual_product_feature;

— Product_as_planned;

— Product_as_realised;

— Product_feature_relationship;

— Product_type;

— Product_type_defined_in external_reference;

— Product_type_relationship;

— Product_type_specified;

— Resource_item;

— Resource_item_assignment;

— Resource_item_characterisation;

— Resource_item_relationship.

4.1.14 properties UoF

The properties UoF enables a property of a product or an activity or a resource to be described. The properties UoF uses the following application objects:

— Activity_property;

— Activity_property_defined_in_external_reference;

— Activity_property_relationship;

— Actvity_property_representation;

— Engineering_property;

— Engineering_property_defined_in_external_reference;

— Engineering_property_representation;

— Property_environment_relationship;

— Property_environment_representation;

— Property_representation_relationship;

— Resource_property;

— Resource_property_defined_in_external_reference;

— Resource_property_representation.

4.1.15 requirement UoF

The requirement UoF enables the description of items that are needed in a testing and measurement process. The requirement UoF uses the following application objects:

— Required_resource;

— Required_resource_by_resource_item;

— Required_resource_by_specification;

— Required_resource_relationship;

— Requirement;

— Requirement_assignment;

— Requirement_source;

— Requirement_version;

— Requirement_version_relationship;

— Requirement_view_definition**.**

4.1.16 state UoF

The state UoF enables the specification of the condition of something at a particular time. The state UoF uses the following application objects:

— Derived_state_type;

— Process_defined_state_type;

— Product_type_defined_state_type;

— Property_defined_state_type;

— Quantity;

— State_Type;

— State_type_assignment;

— State_type_relationship.

4.1.17 substance UoF

The substance UoF enables the description of the physical and chemical state of a substance.
The substance UoF uses the following application objects:

— Structure_element_characterisation;

— Substance;

— Substance_assignment;

— Substance_composition_element;

— Substance_defined_in_external_reference;

— Substance_structure_element.

4.1.18 tolerance datum UoF

The tolerance datum UoF enables the description of the datum from which the permitted deviation from a nominal or ideal geometrical shape can be specified. The tolerance datum UoF uses the following application objects:

— Compound_datum;

— Datum;

— Datum_target;

— Datum_target_set;

— Placed_target;

— Reference_shape;

— Reference_shape_element;

— Single_datum;

— Target_area;

— Target_circle;

— Target_point;

— Target_rectangle;

— Target_straight_line;

— Tolerance_condition.

4.2 Application objects

This subclause specifies the application objects for the engineering properties for product design and validation application protocol. Each application object is an atomic element that embodies a unique application concept and may contain attributes specifying the data elements of the object. The application objects, their definitions and attributes are specified below. The application objects and other data types are grouped in subclauses identified with a Unit of Functionality.

4.2.1 Application objects for the activity UoF

This subclause specifies the data types and application objects associated with the Activity UoF.

4.2.1.1 assignment_select

An assignment_select is a select data type that provides a mechanism for associating an instance of Individual_activity with an instance of one of:

— Approval;

— Certificate;

— Document_version;

— Individual_product;

— Engineering_property_representation.

EXPRESS specification:

*)

```
  TYPE assignment_select = SELECT
    (Approval,
      Certificate,
      Document_version,
      Individual_product,
      Engineering_property_representation);
  END_TYPE;
(*
```

4.2.1.2 Activity_as_planned

An Activity_as_planned is a type of Individual_activity that is intended to happen.

EXPRESS specification:

*)

```
  ENTITY Activity_as_planned
    SUBTYPE OF(Individual_activity);
    plan : Activity_type;
  END_ENTITY;
(*
```

Attribute definitions:

plan : identification of the Activity_type that is intended to be used

4.2.1.3 Activity_as_realised

An Activity_as_realised is a type of Individual_activity that occurred.

EXPRESS specification:

*)

```
  ENTITY Activity_as_realised
    SUBTYPE OF(Individual_activity);
    realisation_of : Activity_as_planned;
    of_type        : Activity_type;
  END_ENTITY;
(*
```

Attribute definitions:

realisation_of : instance of the activity that was planned

of_type : class of activity of which the Activity_as_realised is a member

4.2.1.4 Activity_relationship

An Activity _relationship is an association between two instances of Individual_activity.

EXPRESS specification:

*)

```
  ENTITY Activity_relationship;
    relating_activity : Individual_activity;
    related_activity  : Individual_activity;
    description       : STRING;
    relation_type     : STRING;
  END_ENTITY;
(*
```

Attribute definitions:

relating_activity : one of the instances of Individual_activity participating in the association

related_activity : the other instance of Individual_activity participating in the association

NOTE If one element of the association is dependent on the other then this attribute is the dependent one

description : text that describes the nature of the relationship

relation_type : text that defined the meaning of the relationship

4.2.1.5 Activity_type

An Activity_type is a procedure to achieve an objective.

EXPRESS specification:

*)

```
ENTITY Activity_type;
  name        : STRING;
  description : STRING;
END_ENTITY;
```

(*

Attribute definitions:

name : label by which an Activity_type is known and can be referred to

description : text that characterises an Activity_type

4.2.1.6 Activity_type_defined_in_external_reference

An Activity_type_defined_in_external_reference is a type of Activity_type that is described in an external source.

EXPRESS specification:

*)

```
ENTITY Activity_type_defined_in_external_reference
  SUBTYPE OF(Activity_type);
  class_select              :classification_source_select;
END_ENTITY;
```

(*

Attribute definitions:

class_select : identification of the external reference

4.2.1.7 Activity_type_relationship

An Activity_type_relationship is an association between two instances of Activity_type.

EXPRESS specification:

*)

```
  ENTITY Activity_type_relationship;
    relating_process          :Activity_type;
    related_process           :Activity_type;
    description               :STRING;
    relation_type             :STRING;
  END_ENTITY;
(*
```

Attribute definitions:

relating_process : one of the instances of Activity_type participating in the relationship

related_process : the other instance of Activity_type participating in the relationship

NOTE If one element of the relationship is dependent on the other then this attribute is the dependent one.

description : text that describes the nature of the relationship

relation_type : text that defines the meaning of the relationship

4.2.1.8 Activity_type_specified

An Activity_type_specified is a type of Activity_type that is defined in a formal specification.

EXPRESS specification:

*)

```
  ENTITY Activity_type_specified
    SUBTYPE OF(Activity_type);
    activity_specification    :Specification;
  END_ENTITY;
(*
```

Attribute definitions:

activity_specification : reference to a formal definition of the Activity_type

4.2.1.9 Individual_activity

An Individual_activity is the identification of an occurence of an Activity_type.

EXPRESS specification:

*)

```
ENTITY Individual_activity
  SUPERTYPE OF (ONEOF(Activity_as_planned, Activity_as_realised));
  description                :STRING;
  activity_name              :STRING;
  activity_id                :STRING;
  activity_role              :STRING;
  activity_event             :date_or_event;
  responsible                :person_organisation;
END_ENTITY;
```

(*

Attribute definitions:

description : text that provides information about an Individual_activity

activity_name : label by which an Individual_activity is known and can be referred to

activity_id : label that distinguishes an Individual_activity

activity_role : description of the objective of the Individual_activity

activity_event : information to describe when and where an Individual_activity occurred

responsible : person and or organisation responsible for carrying out the Individual_activity

4.2.1.10 Individual_activity_assignment

An Individual_activity_assignment is a mechanism for associating an Individual_activity with an item.

EXPRESS specification:

*)

```
ENTITY Individual_activity_assignment;
  assigned_individual_activity: Individual_activity;
  assigned_role              :STRING;
  assignment_item            :assignment_select;
END_ENTITY;
```

(*

Attribute definitions:

assigned_individual_activity : reference to the Individual_activity that is to be associated

assigned_role : specification of the purpose of the Individual_Activity_assignment with product or activity data

REMARK Where applicable, the following values shall be used for assigned_role:

'input', where the concepts represented by the assigned_items are inputs to the Activity;
'output', where the concepts represented by the assigned_items result from the Activity;
'constraint', where the concepts represented by the assigned_items influence the execution of the Activity.

assignment_item : identification of the item asigned to an Individual_activity

4.2.2 Application objects for the adminstration UoF

This subclause specifies the data types and application objects associated with the administration UoF.

4.2.2.1 day_in_month

A day_in_month is the ordinal number of a specified day in a month.

EXPRESS specification:

*)

```
TYPE day_in_month = INTEGER;
WHERE
  WR1      :{1 <= SELF <= 31};
END_TYPE;
(*
```

Formal Propositions:

WR1 The value of the integer number shall be from 1 to 31

4.2.2.2 hour_in_day

An hour_in_day is the ordinal number to identify the hour in a Local_time.

EXPRESS specification:

*)

```
TYPE hour_in_day = INTEGER;
WHERE
  WR1      :{ 0 <= SELF < 24};
END_TYPE;
(*
```

Formal Propositions:

WR1 The value of the integer shall be from 0 to 24

4.2.2.3 minute_in_hour

A minute_in_hour is the identification of the ordinal number of a minute in a Local_time.

EXPRESS specification:

*)

```
  TYPE minute_in_hour = INTEGER;
  WHERE
    WR1      :{ 0 <= SELF <= 59};
  END_TYPE;
(*
```

Formal Propositions:

WR1	The value of the integer shall be from 0 to 59

4.2.2.4 month_in_year

A month_in_year is the identification of the ordinal number of a month in a year.

EXPRESS specification:

*)

```
  TYPE month_in_year = INTEGER;
  WHERE
    WR1      :{ 1 <= SELF <= 12 };
  END_TYPE;
(*
```

Formal Propositions:

WR1	The value of the integer number shall be from 1 to 12

4.2.2.5 second_in_minute

A second_in_minute is the identification of the ordinal number of a second in a Local_time.

EXPRESS specification:

*)

```
  TYPE second_in_minute = REAL;
  WHERE
    WR1      :{ 0 <= SELF <= 60.0};
  END_TYPE;
(*
```

Formal Propositions:

WR1 The value of the real number shall be between 0 and 60.0

4.2.2.6 year_number

A year_number is the identification for a year in the Gregorian calendar.

EXPRESS specification:

*)

```
TYPE year_number = INTEGER;
END_TYPE;
```

(*

4.2.2.7 offset_orientation

An offset_orientation is the position of a time zone with respect to the Coordinated Universal Time zone (UTC).

EXPRESS specification:

*)

```
TYPE offset_orientation = ENUMERATION OF
  (AHEAD,
  EXACT,
  BEHIND);
END_TYPE;
```

(*

4.2.2.8 contract_item

A contract_item is a select data type that defines an extensible list of alternate items to which a contract may be assigned.

NOTE The items in the list may be extended in an application that implements this part of ISO 10303.

An instance of Contract may be associated with an instance of one of:

— Individual_activity;

— Individual_product.

EXPRESS specification:

*)

```
TYPE contract_item = EXTENSIBLE_SELECT
  (Individual_activity,
    Individual_product);
END_TYPE;
```

(*

4.2.2.9 date_or_date_time

A date_or_date_time is a select data type that provides a mechanism to refer to an instance of Calendar_date or an instance of Date_time.

EXPRESS specification:

```
*)

  TYPE date_or_date_time = SELECT
    (Date_time,
    Calendar_date);
  END_TYPE;
(*
```

4.2.2.10 date_or_event

A date_or_event is a select data type that provides a mechanism to refer to an instance of: a Calendar_date, a Date_time, or an Event.

EXPRESS specification:

```
*)

  TYPE date_or_event = SELECT
    (Event,
    date_or_date_time);
  END_TYPE;
(*
```

4.2.2.11 Calendar_date

A Calendar_date is the identification of a day in a month of a year.

EXPRESS specification:

```
*)

  ENTITY Calendar_date;
    year                        :year_number;
    month                       :month_in_year;
    day                         :day_in_month;
  END_ENTITY;
(*
```

Attribute definitions:

year : identifier for the year in a Calendar_date

month : identifier for a month in a Calendar_date

day : identifier for a day in a Calendar_date

4.2.2.12 Contract

A Contract is a binding agreement between at least two participants.

EXPRESS specification:

*)

```
  ENTITY Contract;
    effective_date            :Calendar_date;
    contract_id               :STRING;
    contractee                :person_organisation;
    contractor                :SET [1:?] OF person_organisation;
    for_project               :Project;
  END_ENTITY;
(*
```

Attribute definitions:

effective_date : calendar date on which the agreement becomes effective

contract_id : label that identifes an instance of Contract

contractee : beneficiary of the agreement

contractor : executor of the agreement

for_project : reference to the Project for which a Contract is valid

4.2.2.13 Contract_assignment

A Contract_assignment provides a mechanism for the association of a Contract with activity or product data.

EXPRESS specification:

*)

```
  ENTITY Contract_assignment;
    items                     :SET [1:?] OF contract_item;
    assigned_contract         :Contract;
  END_ENTITY;
(*
```

Attribute definitions:

items : activity or product data to which an instance of Contract is assigned

assigned_contract : instance of Contract that is to be associated with activity or product data

4.2.2.14 Date_time

A Date_time is a moment of time on a particular day.

EXPRESS specification:

```
*)

  ENTITY Date_time;
    date                        :Calendar_date;
    time                        :Local_time;
  END_ENTITY;
(*
```

Attribute definitions:

date : reference to an instance of Calendar_date that specifies the particular day

time : reference to the instance of Local_time that specifies the moment of time on a particular day

4.2.2.15 Event

An Event is a fact of something occurring at a point in time.

EXPRESS specification:

```
*)

  ENTITY Event;
    actual_date                 :date_or_date_time;
    event_name                  :STRING;
    event_id                    :STRING;
    description                 :STRING;
    event_location              :Location;
  END_ENTITY;
(*
```

Attribute definitions:

actual_date : instance of Calendar_date or Date_time associated with an Event

event_name : label by which an instance of Event is known and can be referred to

event_id : label that distinguishes an event

description : words that further characterise an Event

event_location : particular place where the event occurred

4.2.2.16 Event_relationship

An Event_relationship is an association between two instances of Event.

EXPRESS specification:

*)

```
  ENTITY Event_relationship;
    relating_event             :Event;
    related_event              :Event;
    relation_type              :STRING;
    description                :STRING;
  END_ENTITY;
(*
```

Attribute definitions:

relating_event : one of the instances of Event that is part of the association

related_event : the other instance of Event that is part of the association

NOTE If one element of the relationship is dependent on the other then this attribute is the dependent one

relation_type : definition of the association between the events

NOTE 1 Where applicable the following values can be used:
- 'actualisation': the relationship specifies that the relating Event identifies a predicted or planned event and the related Event identifies the point in time when that event actually occurred. This relationship is valid of the actual_date attribute of the relating Event is not specified and the actual_date attribute of the related Event is specified, or, if both these attributes are specified and are equal.
- 'exclusiveness': the relationship specifies that the relating and the related Event shall not have any overlap during their duration.
- 'sequence': the relationship specifies that the relating Event shall be completed before the related Event starts.
- 'simultaneity'; the relationship specifies that both the relating and the related event are considered as occurring during the same time period or shall be performed together in order to ensure consistency and enhance efficiency.

NOTE 2 The criteria for 'simultaneity' and in particular the delay allowed between two events declared as simultaneous depend on the application.

description : information about the relationship between two instances of Event

4.2.2.17 Local_time

A Local_time is a point in time in a day represented on a 24-hour clock by hour, minute and second.

NOTE The time is expressed in the local time zone and the offset from the Coordinated Universal Time is specified.

EXPRESS specification:

```
*)

   ENTITY Local_time;
     hour                          :hour_in_day;
     minute                        :minute_in_hour;
     second                        :second_in_minute;
     zone                          :Time_offset;
   END_ENTITY;
(*
```

Attribute definitions:

hour : number of hours at the Local_time

minute : number of minutes at the Local_time

second : number of seconds at the Local_time

zone : reference to Time_offset to determine the deviation from the Coordinated Universal Time

NOTE 1 The symbol for Coordinated Universal Time is UTC.

NOTE 2 The determination of the local time is determined as follows:
- if Time_offset.sense = 'ahead' then Local_time = UTC - hour_offset - minute_offset;
- if Time_offset.sense = 'exact' then Local_time = UTC;
- if Time_offset.sense = 'behind' then Local_time = UTC + hour_offset + minute_offset.

4.2.2.18 Project

A Project is an enterprise or activity planned to achieve a particular objective.

EXPRESS specification:

```
*)

   ENTITY Project;
     project_id                    :STRING;
     project_name                  :STRING;
   END_ENTITY;
(*
```

Attribute definitions:

project_id : label by which the Project can be distinguished

project_name : label by which a Project is known and can be identified

4.2.2.19 Specification

A Specification is a formal definition of a state of a product or an activity.

NOTE A Specification can be a standard.

EXPRESS specification:

*)

```
  ENTITY Specification;
    of_type                   :Specification_type;
    id                        :STRING;
    issued_by                 :Organisation;
    date                      :Calendar_date;
    specification_document    :Document_version;
  END_ENTITY;
(*
```

Attribute definitions:

of_type : reference to the Specification _type of which the Specification is an instance

id : label by which the Specification can be identified

issued_by : identifcation of the organisation resposnible for the specification

date : calendar date on which the specification was issued

specification_document : document where the Specification is described

4.2.2.20 Specification_relationship

A Specification_relationship is an association between two instances of Specification.

EXPRESS specification:

*)

```
  ENTITY Specification_relationship;
    related                   :Specification;
    relating                  :Specification;
    relation_type             :STRING;
    description               :STRING;
  END_ENTITY;
(*
```

Attribute definitions:

related : the other Specification that is part of the relationship.

NOTE If one element of the relationship is dependent upon the other, then this attribute is the dependent one.

relating : one of the Specifications that is part of the Specification_relationship

relation_type : text that defines the meaning of the relationship

description : text that provides further information about the relationship

4.2.2.21 Specification_type

A Specification_type is the class of Specification to which an instance of a Specification may belong.

EXPRESS specification:

*)

```
ENTITY Specification_type;
  name                        :STRING;
  description                 :STRING;
END_ENTITY;
(*
```

Attribute definitions:

name : label by which a Specification type is known

description : information about the Specification type

4.2.2.22 Time_offset

A Time_offset is the oriented offset from Coordinated Universal Time.

EXPRESS specification:

*)

```
ENTITY Time_offset;
  sense                       :offset_orientation;
  hour_offset                 :INTEGER;
  minute_offset               :INTEGER;
DERIVE
  actual_minute_offset        :INTEGER  := NVL(minute_offset,0);
WHERE
  WR1      :{0 <= hour_offset <24};
  WR2      :{0 <= minute_offset <= 59};
  WR3      :NOT (((hour_offset <> 0) OR (actual_minute_offset
            <> 0)) AND (sense = exact));
END_ENTITY;
(*
```

Attribute definitions:

sense : direction of the offset from Universal Coordinated Time

hour_offset : number of hours by which a time is offset from Coordinated Universal Time

minute_offset : the number of minutes by which a time is offset from Coordinated Universal Time

actual_minute_offset : value of the minute offset used to compute a Time_offset.

NOTE The value is zero if the value of minute_offset is either not specified or is equal to zero.

Formal Propositions:

WR1 the hour_offset shall be a positive number less than 24.

WR2 the minute offset shall be a positive number less than 59

WR3 If the value of sense specifies that ther is no offset from the Coordinated Universal Time then the hour_offset and the actual_minute_offset shall both be equal to zero. If either hour_offset or the actual_minute_offset are different from zero then the value of sense shall be specified to show that there is an offset either ahead or behind from the Universal Coordinated Time.

4.2.3 Application objects for the approval UoF

This subclause specifies the data types and application objects associated with the approval UoF.

4.2.3.1 affected_item_select

An affected_item_select is a select data type that provides a mechanism for associating an instance of either Approval or Certificate with an instance of one of:

— Activity_type;

— Document_version;

— Individual_activity;

— Individual_product;

— Organisation;

— Person_in_organisation;

— Product_type.

EXPRESS specification:

*)

```
TYPE affected_item_select = SELECT
  (Activity_type,
  Document_version,
```

```
    Individual_activity,
    Individual_product,
    Product_type,
    Organisation,
    person_organisation);
  END_TYPE;
(*
```

4.2.3.2 AP_qualification_type

An AP_qualification_type is the identification of a formal capability and authority.

EXPRESS specification:

```
*)

  ENTITY AP_qualification_type;
    name                        :STRING;
    description                 :STRING;
  END_ENTITY;
(*
```

Attribute definitions:

name : label by which the qualification is known

description : further information about the nature of the qualification

4.2.3.3 Approval

An Approval is formal confirmation of an activity or item of product data

EXPRESS specification:

```
*)

  ENTITY Approval;
    defined_in                  :Certificate;
    status                      :Approval_status;
    purpose                     :STRING;
    approval_date               :Calendar_date;
    authorised_by               :Approving_person_organisation;
  END_ENTITY;
(*
```

Attribute definitions:

defined_in : document where the approval is recorded

status : user interpretable designation of the level of approval

purpose : text specifying the reason or intention of the approval

approval_date : point in time when the approval was performed

authorised_by : person and organisation responsible for granting an Approval

4.2.3.4 Approval_assignment

An Approval_assignment is a mechanism for the association of an Approval with an activity or with product data.

EXPRESS specification:

*)

```
ENTITY Approval_assignment;
  assigned_approval          :Approval;
  items                      :SET [1:?] OF affected_item_select;
  role                       :STRING;
END_ENTITY;
(*
```

Attribute definitions:

assigned_approval : approval that is assigned with activity or product data

items : activity or product data to which the approval is assigned

role : text that describes the function of the approval with respect to the items to which it is assigned

4.2.3.5 Approval_relationship

An Approval_relationship is an association between two instances of Approval.

EXPRESS specification:

*)

```
ENTITY Approval_relationship;
  relating_approval          :Approval;
  related_approval           :Approval;
  relation_type              :STRING;
  description                :STRING;
END_ENTITY;
(*
```

Attribute definitions:

relating_approval : one of the Approval objects that is part of the relationship

related_approval : One of the Approval objects that is part of the relationship.

NOTE If one element of the relationship is dependent on the other then this attibute is the dependent one.

relation_type : text that identifies the meaning of the relationship

Where applicable the following values of relation_type shall be used:
- 'decomposition': identifies a relationship where the related Approval is one of the components into which the relating Approval is broken down with no implication of 'sequence' or 'dependency'.
- 'dependency' : identifies a relationship where the issuing of the related Approval is dependent upon the relating Approval.
- 'precedence': identifies a relationship where the realted Approval has a higher priority than the relating Approval.
- 'sequence': identifies a relationship where the relating Approval shall be completed before the related Approval is given.

NOTE The value 'depencency' does not imply the meaning of the values 'decomposition' or 'sequence'

EXAMPLE 1 The Approval of an Activity may be dependent on the Approval of all of the constituent parts of the Activity.

EXAMPLE 2 The Approval of an Activity in a sequence of activities may depend on the prior approval of a preceding Activity.

description : text that provides further information about the Approval_relationship

4.2.3.6 Approval_status

An Approval_status is a particular rank or importance of Approval.

EXPRESS specification:

*)

```
ENTITY Approval_status;
  status_name                :STRING;
END_ENTITY;
(*
```

Attribute definitions:

status_name : text that describes the Approval_status

EXAMPLE 'approved' or ' not approved' are examples of Approval_status.

4.2.3.7 Approving_person_organisation

An Approving_person_organisation is an association between an Approval and the person or organisation that has granted the Approval.

EXPRESS specification:

*)

```
ENTITY Approving_person_organisation;
  role                       :STRING;
```

```
    qualification              :AP_qualification_type;
    approved_by                :person_organisation;
  END_ENTITY;
(*
```

Attribute definitions:

role : description of the function of the considered person or organisation with respect to an Approval.

qualification : reference to the formal authority for providing an Approval

approved_by : reference to the person or organisation responsible for an Approval

4.2.3.8 Certificate

A Certificate is a record of approval.

EXPRESS specification:

*)

```
  ENTITY Certificate;
    cert_id                    :STRING;
    cert_name                  :STRING;
    cert_date                  :Calendar_date;
    issued_by                  :person_organisation;
  END_ENTITY;
(*
```

Attribute definitions:

cert_id : identifier for a Certificate

cert_name : label that identifies the type of certificate

cert_date : calendar date when the certificate was issued

issued_by : identification of the person or organisation responsible for the issue of a Certificate

4.2.3.9 Certificate_assignment

A Certificate _assignment is a mechanism for specifying an association between a record of approval and the objects that are approved.

EXPRESS specification:

*)

```
  ENTITY Certificate_assignment;
    assigned_certificate       :Certificate;
```

```
    certificated_items          :SET [1:?] OF affected_item_select;
    role                        :STRING;
  END_ENTITY;
(*
```

Attribute definitions:

assigned_certificate : identification of the record of approval

certificated_items : set of objects that are approved

role : purpose of the association

4.2.3.10 Security_classification

A Security_classification is a type of Approval that defines the conditions of access.

EXPRESS specification:

```
*)

  ENTITY Security_classification
    SUBTYPE OF(Approval);
    classification_level        :STRING;
    description                 :STRING;
  END_ENTITY;
(*
```

Attribute definitions:

classification_level : text that defines the conditions of access

EXAMPLE 'confidential' is a value of classification_level that can be applied to messages

description : text to provide further information about the conditions of access

4.2.4 Application objects for the condition UoF

This subclause specifies the data types and application objects associated with the condition UoF.

4.2.4.1 condition_evaluation_item

A condition_evaluation_item is an extensible select data type for instances of objects that are to be included in the Condition_evaluation.

NOTE The list can be extended by applications that implement this part of ISO 10303.

An instance of Condition_evaluation may be associated with one of:

— Aproval;

— Individual_activity:

— Individual_product.

EXPRESS specification:

*)

```
  TYPE condition_evaluation_item = SELECT
    (Individual_product,
    Individual_activity,
    Approval);
  END_TYPE;
(*
```

4.2.4.2 conditioned_item

A conditioned_item is an extensible select data type for instances of items to which a Condition is assigned.

NOTE The list can be extended by applications that implement this part of ISO 10303.

An instance of Condition may be associated with an instance of:

— Approval;

— Individual_activity;

— Individual_product.

EXPRESS specification:

*)

```
  TYPE conditioned_item = SELECT
    (Individual_product,
    Individual_activity,
    Approval);
  END_TYPE;
(*
```

4.2.4.3 Condition

A Condition is a state that shall exist before something else is possible.

EXPRESS specification:

*)

```
  ENTITY Condition;
    description              :STRING;
    name                     :STRING;
    defined_state            :State_type;
  END_ENTITY;
(*
```

Attribute definitions:

description : text describing an instance of Condition

name : lable by which a Condition is known

defined_state : reference to the state that shall be satisfied

4.2.4.4 Condition_assignment

A Condition_assignment provides a mechanism for the association of a Condition with product or activity data.

NOTE Condition_assignment represents the consequences of Condition.

EXPRESS specification:

```
*)

  ENTITY Condition_assignment;
    assigned_condition         :Condition;
    assigned_item              :conditioned_item;
  END_ENTITY;
(*
```

Attribute definitions:

assigned_condition : the instance of Condition that is being assigned

assigned_item : product or activity data to which the Condition is being assigned

4.2.4.5 Condition_evaluation

A Condition_evaluation is the record of the evaluation of a Condition and the subsequent result.

EXPRESS specification:

```
*)

  ENTITY Condition_evaluation;
    condition                  :Condition;
    description                :STRING;
    name                       :STRING;
    evaluation_state           :State_type;
  END_ENTITY;
(*
```

Attribute definitions:

condition : instance of Condition that has been evaluated

description : text to provide information about the result of the Condition_evaluation

NOTE Possible outcomes of an evaluation and their meanings are:
'TRUE' - the condition was satified
'FALSE' - the condition was not satisfied
'NOT KNOWN' - the result of the evaluation was not measured

name : label by which the Condition_evaluation is known

evaluation_state : the state of the object at the time of the evaluation

4.2.4.6 Condition_evaluation_assignment

A Condition_evaluation_assignment provides a mechanism for the association of a Condition_evaluation with product or activity data.

NOTE A Condition_evaluation_assignment represents the consequence of the Condition_evaluation.

EXPRESS specification:

*)

```
ENTITY Condition_evaluation_assignment;
  assigned_condition_evaluation    :  Condition_evaluation;
  item                             :  condition_evaluation_item;
END_ENTITY;
```

(*

Attribute definitions:

assigned_condition_evaluation : instance of Condition_evaluation that is being assigned

item : product or activity data to which the Condition_evaluation is being assigned

4.2.4.7 Condition_relationship

A Condition_relationship is an association between two instances of Condition.

NOTE A Condition_relationship normally represents a logical combination of conditions.

EXAMPLE ' If the engine has been running for 1000 hours AND if the oil filter has a lifetime of 1000 hours then change the oil filter' is an example of two conditions related by a logical AND.

EXPRESS specification:

*)

```
ENTITY Condition_relationship;
  relating_condition         :Condition;
  related_condition          :Condition;
  relation_type              :STRING;
  description                :STRING;
END_ENTITY;
```

(*

Attribute definitions:

relating_condition : one of the instances of Condition that participates in the relationship

related_condition : the other instance of Condition that participates in the relationship

NOTE 1 If one element of the relationship is dependent on the other then this attribute is the dependent one.

relation_type : text that defines the meaning of the relationship

description : text to describe the Condition relationship

NOTE 2 The value of this attribute need not be specified

4.2.5 Application objects for the document_management UoF

This subclause specifies the data types and application objects associated with the document_management UoF.

4.2.5.1 documented_items

A documented_items is an extensible select data type that provides a list of alternatives with which a Document can be selectively associated.

Document may be associated with an instance of:

— Activity_type;

— Individual_activity;

— Individual_product;

— Product_type.

EXPRESS specification:

*)

```
TYPE documented_items = SELECT
  (Individual_activity,
  Individual_product,
  Activity_type,
  Product_type);
END_TYPE;
(*
```

4.2.5.2 external_identification_item

An external_identification_item is an extensible select data type that provides a list of alternatives from which an instance of External_source_identification can be selected.

NOTE The list may be extended by applications that implement this part of ISO 10303.

External_source_identification can be associated with an instance of:

— Document_definition;

— Document_location_identification;

— File;

— File_location_identification.

EXPRESS specification:

*)

```
  TYPE external_identification_item = EXTENSIBLE SELECT
    (File,
    Document_definition,
    Document_location_identification,
    File_location_identification);
  END_TYPE;
(*
```

4.2.5.3 Digital_document_definition

A Digital_document_definition is a type of Document_definition that defines a digital electronic form.

EXPRESS specification:

*)

```
  ENTITY Digital_document_definition
    SUBTYPE OF(Document_definition);
  END_ENTITY;
(*
```

4.2.5.4 Digital_file

A Digital_file is a type of File where the information is stored by using particular values of voltage or magnetic polarization to represent information

EXPRESS specification:

*)

```
  ENTITY Digital_file
    SUBTYPE OF(File);
  END_ENTITY;
(*
```

4.2.5.5 Digital_record

A Digital_record is a type of Digital_file that contains an identifiable item of information.

EXPRESS specification:

*)

```
  ENTITY Digital_record
    SUBTYPE OF(Digital_file);
  END_ENTITY;
(*
```

4.2.5.6 Document

A Document is written, printed or electronic matter that provides information.

EXPRESS specification:

```
*)

  ENTITY Document;
    document_name              :STRING;
  END_ENTITY;
(*
```

Attribute definitions:

document_name : label that enables a document to be identified.

4.2.5.7 Document_assignment

A Document_assignment is a mechanism for the association of a Document with product or activity data.

EXPRESS specification:

```
*)

  ENTITY Document_assignment;
    assigned_document          :SET [1:?] OF Document;
    item                       :documented_items;
    assigned_role              :STRING;
  END_ENTITY;
(*
```

Attribute definitions:

assigned_document : reference to the document or documents that are associated with product or activity data

item : reference to the product or activity data to which the document is assigned

assigned_role : text to describe the purpose for which the assignment of a document has been made

4.2.5.8 Document_definition

A Document_definition is the identification of a representation of a Document_version in a particular format.

NOTE A Document_version can have more than one representation.

EXPRESS specification:

*)

```
ENTITY Document_assignment;
  assigned_document          :SET [1:?] OF Document;
  item                       :documented_items;
  assigned_role              :STRING;
END_ENTITY;
(*
```

Attribute definitions:

assigned_document : reference to the document or documents that are associated with product or activity data

item : reference to the product or activity data to which the document is assigned

assigned_role : text to describe the purpose for which the assignment of a document has been made

4.2.5.9 Document_definition_relationship

A Document_definition_relationship is an association between two instances of Document_definition.

EXPRESS specification:

*)

```
ENTITY Document_definition_relationship;
  relating_definition        :Document_definition;
  related_definition         :Document_definition;
  relation_type              :STRING;
  description                :STRING;
END_ENTITY;
(*
```

Attribute definitions:

relating_definition : one of the instances of Document_definition that is part of the relationship

related_definition : the other instance of Document_definition that is part of the relationship

NOTE If one element of the relationship is dependent upon the other then this attribute is the dependent one.

relation_type : text that defines the meaning of the relationship

description : text that explains the nature of the relationship

4.2.5.10 Document_location_identification

A Document_location_identification is a type of External_source_identification that identifies the location of the components of a Document_definition in an external storage system.

EXAMPLE An HTML file that includes a picture may be represented as a Digital_document consisting of two components:

— the HTML file;

— the binary file that contains the picture.

If these files are located within the same directory or relatively to the same directory then the source_id attribute would convey the directory name.

EXPRESS specification:

*)

```
ENTITY Document_location_identification
  SUBTYPE OF(External_source_identification);
WHERE
  WR1      :'DOCUMENT_DEFINITION' IN
            TYPEOF(SELF\External_source_identification.item_id);;
END_ENTITY;
```

(*

Formal Propositions:

WR1 The identified external item shall be of type Document_definition.

4.2.5.11 Document_relationship

A Document_relationship is an association between two instances of Document.

EXPRESS specification:

*)

```
ENTITY Document_relationship;
  relation_type              :STRING;
  description                :STRING;
  related_document           :Document;
  relating_document          :Document;
END_ENTITY;
```

(*

Attribute definitions:

relation_type : text that defines the meaning of the association

description : text to explain the nature of the relationship

related_document : one of the instances of Document that is part of the association

NOTE If one element of the relationship is dependent on the other then this attribute is the dependent one.

relating_document : the other of the instances of Document that is part of the association

4.2.5.12 Document_version

A Document_version is the identification of a particular variant of a Document.

EXPRESS specification:

*)

```
ENTITY Document_version;
  of_product                :Document;
  document_date             :date_or_date_time;
  document_location         :Location;
  document_author           :LIST [1:?] OF person_organisation;
END_ENTITY;
```

(*

Attribute definitions:

of_product : reference to the Document of which the instance of this entity is a version

document_date : calendar date of the version of the document

document_location : place where the document version can be found

document_author : person or organisation responsible for the creation of the document

4.2.5.13 External_item_identification

An External_identification is a type of External_source_identification that identifies an item in the external source where it can be found.

EXPRESS specification:

*)

```
ENTITY External_item_identification
  SUBTYPE OF(External_source_identification);
  external_id               :STRING;
END_ENTITY;
```

(*

Attribute definitions:

external_id : label that distinguishes an item in an external source

4.2.5.14 External_source_identification

An External_source_identification is the identification of the source where an item, that is not fully represented in a population of instances of EXPRESS enitity data type but is only referred to, or the components of such an item, can be found.

EXPRESS specification:

```
*)

  ENTITY External_source_identification;
    item_id                    :External_identification_item;
    description                :STRING;
    source_id                  :STRING;
    source_type                :STRING;
  END_ENTITY;
(*
```

Attribute definitions:

item_id : identification of the item that constitutes the external source

description : text to provide further information about the item

source_id : identification of the source where the item can be found

source_type : description of the nature of the source where the item can be found

EXAMPLE Values of the source_type attribute could be:
- 'URL' for a location on the world wide web;
- 'FTP' for a file transfer protocol address;
- 'ISBN' for the International Standard Book Number of a published book.

4.2.5.15 File

A File is a collection of information stored under a single identifying name.

EXPRESS specification:

```
*)

  ENTITY File
    SUPERTYPE OF (ONEOF(Hard_copy, Digital_file));
    file_name                  :STRING;
  END_ENTITY;
(*
```

Attribute definitions:

file_name : label that identifies the file and by which it can be referred

4.2.5.16 File_location_identification

A File_location_identification is a type of External_item_identification that is a place where an instance of File can be found.

EXPRESS specification:

*)

```
  ENTITY File_location_identification
    SUBTYPE OF(External_item_identification);
  WHERE
    WR1      :'FILE' IN TYPEOF(SELF\External_source_identification.item_id);
  END_ENTITY;
(*
```

Formal Propositions:

WR1	The identified external item shall be of type File

4.2.5.17 File_relationship

A File_relationship is an association between two instances of File.

EXPRESS specification:

*)

```
  ENTITY File_relationship;
    relating_file              :File;
    related_file               :File;
    description                :STRING;
    relation_type              :STRING;
  END_ENTITY;
(*
```

Attribute definitions:

relating_file : one of the instances of File that is part of the association

related_file : the other instance of File that is part of the association.

NOTE If one element of the relationship is dependent on the other then this attribute is the dependent one.

description : text to provide further information about the relationship

relation_type : information about the nature of the relationship

4.2.5.18 Hard_copy

A Hard_copy is a type of File where the information is stored as characters or images printed on paper or

other printable media.

EXPRESS specification:

*)

```
ENTITY Hard_copy
  SUBTYPE OF(File);
END_ENTITY;
(*
```

4.2.5.19 Physical_document_definition

A Physical_document_definition is a type of Document_definition that defines a printed form.

EXPRESS specification:

*)

```
ENTITY Physical_document_definition
  SUBTYPE OF(Document_definition);
END_ENTITY;
```

4.2.6 Application objects for the effectvity UoF

This subclause specifies the data types and application objects associated with the effectivity UoF.

4.2.6.1 effectivity_item

An effectivity_item is an extensible select data type that provides a list of alternatives to which an instance of Effectivity can be selectively associated.

NOTE The list can be extended by an application that implements this part of ISO 10303.

An instance of Effectivity can be associated with one of:

— Approval;

— Contract;

— Individual_product;

— Individual_activity.

EXPRESS specification:

*)

```
TYPE effectivity_item = EXTENSIBLE SELECT
  (Approval,
    Contract,
    Individual_product,
    Individual_activity);
END_TYPE;
```

(*

4.2.6.2 interval_select

An interval_select provides the means to be able to choose between an interval defined by a number of cycles of a recurrent event or an interval measured in units of time.

EXPRESS specification:

*)

```
TYPE interval_select = SELECT
  (Frequency_interval,
  Time_interval);
END_TYPE;
```

(*

4.2.6.3 Dated_effectivity

A Dated_effectivity is a type of Effectivity for which the domain of applicability is defined as an interval of time bounded by dates or events.

NOTE The time interval can be open ended

EXAMPLE Events can be used to bound a period of Dated_effectivity at a planning phase.

REMARK Depending on whether the end_bound attribute is specified, the actual domain of time identified by an instance of Dated_effectivity is:

— the time interval between the start and the end date of an event, or

— the open time interval that starts at the start date or event.

If the end_bound is an event that actually identifies a point in time that comes before the start_bound then the actual domain of effectivity is empty.

EXPRESS specification:

*)

```
ENTITY Dated_effectivity
  SUBTYPE OF(Effectivity);
  start_bound               :date_or_event;
  end_bound                 :OPTIONAL date_or_event;
END_ENTITY;
```

(*

Attribute definitions:

start_bound : date or time or event that defines the lower bound of the interval of applicability

end_bound : date or time or event that defines the lower bound of the interval of applicability

NOTE 1 The value of the attribute need not be specified.

NOTE 2 If the value for this attribute is not specified then the interval of applicability has no upper limit.

4.2.6.4 Effectivity

An Effectivity is the domain of applicability of product or activity data.

NOTE Instances of Effectivity can be applied to any kind of product or activity data.

EXPRESS specification:

```
*)

  ENTITY Effectivity
    SUPERTYPE OF (ONEOF(Dated_effectivity, Lot_effectivity,
                        Serial_effectivity, Time_interval_effectivity));
    name                    :STRING;
    effectivity_id          :STRING;
    description             :STRING;
  END_ENTITY;
(*
```

Attribute definitions:

name : label by which the instance of Effectivity is known

effectivity_id : label that distinguishes the instance of Effectivity

description : text to provide further information on the instance of Effectivity

4.2.6.5 Effectivity_assignment

An Effectivity_assignment is a mechanism for the association of an instance of Effectivity with product or activity data.

EXPRESS specification:

```
*)

  ENTITY Effectivity_assignment;
    assigned_effectivity       :Effectivity;
    items                      :SET [1:?] OF effectivity_item;
    role                       :STRING;
  END_ENTITY;
(*
```

Attribute definitions:

assigned_effectivity : instance of the Effectivity entity that is assigned

items : set of effectivity_item whose effectivity is characterised by this entity

role : description of the purpose of the association of the Effectivity with the effectivity_item

NOTE For an Effectivity that defines a period of time, the following values can be used:
- 'actual': the assigned effectivity defines a period of time during which the associated items are or were effective;
- 'planned': the assigned effectivity defines a period of time during which the associated items are or were expected to be effective;
- 'required': the assigned effectivity defines a period of time during which the associated items are or were required to be effective.

4.2.6.6 Effectivity_relationship

An Effectivity_relationship is an association between two instances of Effectivity.

NOTE The meaning of the relationship is represented in the relation_type attribute.

EXPRESS specification:

*)

```
ENTITY Effectivity_relationship;
  relating_effectivity       :Effectivity;
  related_effectivity        :Effectivity;
  relation_type              :STRING;
  description                :STRING;
END_ENTITY;
(*
```

Attribute definitions:

relating_effectivity : one of the instances of Effectivity in the association

related_effectivity : the other instance of Effectivity in the association

NOTE If one element of the relationship is dependent on the other then this attribute is the dependent one.

relation_type : text that defines the meaning of the relationship

description : text that provides further information about the relationship

4.2.6.7 Frequency_interval

A frequency_interval is a time interval measured by the accumulated number of cycles of a recurrent event.

EXPRESS specification:

*)

```
  ENTITY Frequency_interval;
    accumulation                :Measure_item;
    rate                        :Measure_item;
  END_ENTITY;
(*
```

Attribute definitions:

accumulation : value of the number of recurrent events

rate : frequency of occurence of the recurring event

4.2.6.8 Lot_effectivity

A Lot_effectivity is a type of Effectivity for which the domain of applicability is an identified batch of items.

EXPRESS specification:

```
*)

  ENTITY Lot_effectivity
    SUBTYPE OF(Effectivity);
    lot_id                      :STRING;
    lot_size                    :INTEGER;
  END_ENTITY;
(*
```

Attribute definitions:

lot_id : label that distinguishes a lot or batch

lot_size : quantity of items in the lot or batch

4.2.6.9 Serial_effectivity

A Serial_effectivity is a type of Effectivity for which the domain of applicability is an interval of serial numbers for products or activities.

NOTE The interval can be open ended.

EXPRESS specification:

```
*)

  ENTITY Serial_effectivity
    SUBTYPE OF(Effectivity);
    start_id                    :STRING;
    end_id                      :OPTIONAL STRING;
  END_ENTITY;
(*
```

Attribute definitions:

start_id : first valid serial number to which the Effectivity is valid

end_id : last serial number to which the Effectivity is valid

NOTE 1 The value of this attribute need not be specified.

NOTE 2 If the value for this attribute is not specified then the interval of applicability has no upper bound.

4.2.6.10 Time_interval

A Time_interval is the identification of an elapse of time.

EXPRESS specification:

```
*)

  ENTITY Time_interval;
    description               :STRING;
    name                      :STRING;
    id                        :STRING;
  END_ENTITY;
(*
```

Attribute definitions:

description : text that provides further information on the Time_interval

NOTE The value of this attribute need not be specified

name : label by which the TIme_interval is known

id : label that distinguishes the Time_interval

4.2.6.11 Time_interval_effectivity

A Time_interval_effectivity is a type of Effectivity for which the domain of applicability is defined as a Time_interval.

EXPRESS specification:

```
*)

  ENTITY Time_interval_effectivity
    SUBTYPE OF(Effectivity);
    efectivity_interval       :interval_select;
  END_ENTITY;
(*
```

Attribute definitions:

effectivity_interval : period of time within which the effectivity is valid

4.2.6.12 Time_interval_relationship

A Time_interval_relationship is an association between two instances of Time_interval.

NOTE The meaning of the association is specified in the relation_type attribute.

EXPRESS specification:

*)

```
ENTITY Time_interval_relationship;
  related_interval          :Time_interval;
  relating_interval         :Time_interval;
  relation_type             :STRING;
  description               :OPTIONAL STRING;
END_ENTITY;
(*
```

Attribute definitions:

related_interval : the other of the instances of Time_interval that participates in the association

NOTE 1 If one of the instances is dependent on the other then this attribute is the dependent one.

relating_interval : one of the instances of Time_interval that participates in the association

relation_type : text that defines the meaning of the relationship

description : information about the nature of the association

NOTE 2 The value of this attribute need not be specified

4.2.6.13 Time_interval_with_bounds

A Time_interval_with_bounds is a type of Time_interval that is bounded either on one side or both sides.

NOTE If neither secondary_bound nor duration are specified then the time interval begins at the point in time identified by the primary_bound attribute and has no specified end point.

EXPRESS specification:

*)

```
ENTITY Time_interval_with_bounds
  SUBTYPE OF(Time_interval);
  primary_bound             :date_or_event;
  secondary_bound           :date_or_event;
  duration                  :Duration;
WHERE
```

```
    WR1      :NOT (EXISTS (secondary_bound) AND EXISTS (duration));
    WR2      :EXISTS (primary_bound) OR EXISTS (secondary_bound);
  END_ENTITY;
(*
```

Attribute definitions:

primary_bound : bound of the Time_interval_with_bound from which the time interval is measured.

secondary_bound : bound of the Time_interval_with_bound to which the time interval is measured

duration : measure of the extent of the interval from the primary bound

Formal Propositions:

WR1	The secondary_bound and duration shall not both be specified for an instance of Time_interval_with_bounds
WR2	Either the primary_bound or the secondary_bound or both shall be specified

4.2.7 Application objects for the external reference UoF

This subclause specifies the data types and application objects associated with the external reference UoF.

4.2.7.1 default_language_string

A default_language_string is a label that identifies the name of a natural language

EXPRESS specification:

```
TYPE default_language_string = STRING;
END_TYPE;
```

4.2.7.2 alias_select

An alias_select is an extensible select data type that provides a mechanism to associate an instance of Alias_identification with one of:

— Activity_property;

— Activity_type;

— Document;

— Engineering_property;

— Product_type;

— Resource_item;

— Resource_property;

— Substance.

EXPRESS specification:

*)

```
  TYPE alias_select = EXTENSIBLE SELECT
    (Activity_property;
      Activity_type;
      Document;
      Engineering_property;
      Product_type;
      Resource_item;
      Resource_property;
      Substance);
  END_TYPE;
(*
```

4.2.7.3 classification_source_select

A classification_souce_select is an extensible select data type that provides a list of alternatives for categorised information about products and activities.

EXPRESS specification:

*)

```
  TYPE classification_source_select = EXTENSIBLE SELECT
    (Plib_class_reference,
    External_library_reference,
    External_source_identification);
  END_TYPE;
(*
```

4.2.7.4 property_source_select

A property_source_select is an extensible select data type that provides a list of alternatives for categorised information about engineering properties.

EXPRESS specification:

*)

```
  TYPE property_source_select = SELECT
    (Plib_property_reference,
    External_library_reference,
    External_source_identification);
  END_TYPE;
(*
```

4.2.7.5 string_select

A string_select provides a mechanism to associate the description of an instance of Alias_identification with either an instance of default_language_string or and instance of Multi_language_string.

EXPRESS specification:

```
*)
TYPE string_select = SELECT
  (default_language_string,
  Multi_language_string);
END_TYPE;
(*
```

4.2.7.6 Alias_identification

An Alias_identification is a mechanism to associate an object with an additional identifier that is used to identify the object of interest in a different context, either in another organization, or in some other context.

NOTE The scope of the alias_identification is specified either by the attribute 'alias_scope' or by the attribute 'description'.

EXAMPLE A document (e.g., a book) can have a unique document id (ISBN) and an alias_identification as inventory number in the context of the inventory of a company.

EXPRESS specification:

```
*)
ENTITY alias_identification;
  alias_id          :  STRING;
  alias_scope       :  OPTIONAL organization;
  alias_version_id  :  OPTIONAL STRING;
  description       :  OPTIONAL string_select;
  is_applied_to     :  alias_select;
END_ENTITY;
(*
```

Attribute definitions:

alias_id : the identifier used in the context specified by the alias_scope, or by the description

alias_scope : the organization in which the alias_identification is valid
NOTE The alias_scope need not be specified for a particular instance of Alias_identification.

alias_version_id : the version of the object as known in the context of the alias_identification
NOTE 1 An alias_version_id can be applied even if the object of interest does not have versions.

NOTE 2 The alias_version_id is not always specified for a particular instance of Alias_identification

description : the type of the alias_identification
NOTE The description need not be specified for an particular instance of Alias_identification

is_applied_to : the object to which an instance of Alias_identification refers

4.2.7.7 External_library_reference

An External_library_reference provides a mechanism to refer to an entry in an external library that does not conform to ISO 13584 Parts library.

EXPRESS specification:

*)

```
ENTITY External_library_reference;
  library_type              :STRING;
  external_id               :STRING;
  description               :OPTIONAL STRING;
END_ENTITY;
```

(*

Attribute definitions:

library_type : The library_type specifies the type of library that is used.

external_id : The external_id specifies the unique identifier of the referenced entry in the external library.

description : The description specifies additional information about the external_library_reference.

The description need not be specified for a particular external_library_reference.

4.2.7.8 Language

A Language is the identification of the natural language in which an instance of an information object is communicated.

EXPRESS specification:

*)

```
ENTITY Language;
  country_code              :OPTIONAL STRING;
  language_code             :STRING;
END_ENTITY;
```

(*

Attribute definitions:

country_code : label to identify the country in accordance with ISO 3166-1
NOTE The country_code need not be specified.

language_code : the language of the text information in the Alpha-3 code specified in ISO 639-2.

4.2.7.9 Multi_language_string

A Multi_language_string is text, expressed in one or more languages, that is associated with information objects.

EXPRESS specification:

```
*)

ENTITY Multi_language_string;
  additional_language_string :SET [0:?] OF String_with_language;
  primary_language_string    :String_with_language;
END_ENTITY;
(*
```

Attribute definitions:

additional_language_string : the set of language objects that represent the information in a particular language

primary_language_string : the representation of the information in the original language

4.2.7.10 Plib_class_reference

A Plib_class_reference designates a class in a library compliant with ISO 13584 Parts library.

EXPRESS specification:

```
*)

ENTITY Plib_class_reference;
  supplier_bsu               :STRING;
  code                       :STRING;
  version                    :STRING;
END_ENTITY;
(*
```

Attribute definitions:

supplier_bsu : The supplier_bsu (basic semantic unit) specifies the supplier of the class in a PLIB library, in which the class is defined. The format of this specification is defined in ISO 13584-26.

code : The code specifies the class in the PLIB library. The format of this code is defined in ISO 13584-42.

version : The version specifies the identification of a particular version of a class in a PLIB library. The format of this version is defined in ISO 13584-42.

4.2.7.11 Plib_property_reference

A plib_property_reference designates a property defined in a library compliant with ISO 13584 Parts library.

EXPRESS specification:

*)

```
ENTITY Plib_property_reference;
  code                      :STRING;
  version                   :STRING;
  name_scope                :Plib_class_reference;
END_ENTITY;
(*
```

Attribute definitions:

code : The code specifies the property in the PLIB library. The format of this code is defined in ISO 13584-42.

version : The version specifies the identification of a particular version of a property in a PLIB library. The format of this version is defined in ISO 13584-42.

name_scope : The name_scope specifies the plib_class_reference in which the property is visible.

4.2.7.12 String_with_language

A String_with_language represents text information in a specific language together with an identification of the language used.

EXPRESS specification:

*)

```
ENTITY String_with_language;
  value                     :STRING;
  language_specification    :Language;
  INVERSE
  used_by                   :SET [1:?] OF multi_language_string FOR
                             primary_language_string;
END_ENTITY;
(*
```

Attribute definitions:

contents : textual information stored in the language identified by the language_specification attribute

language_specification : the natural language in which the information is communicated.

used_by : the reference to the language of the primary_language_string.

4.2.8 Application objects for the geometry UoF

This subclause specifies the data types and application objects associated with the geometry UoF.

4.2.8.1 cartesian_transformation

A cartesian_transformation is a select data type that provides a list of alternatives with which an instance of Axis_placement_transformation_mapping can be selectively associated. An instance of Axis_placement_transformation_mapping can be associated with one of:

— Cartesian_transformation_2D;

— Cartesian_transformation_3D.

EXPRESS specification:

```
*)

  TYPE cartesian_transformation = SELECT
    (Cartesian_transformation_2D,
     Cartesian_transformation_3D);
  END_TYPE;
(*
```

4.2.8.2 Axis_placement

An Axis_placement is a type of Geometrical_model_element that defines a right-handed, 2D or 3D coordinate system.

NOTE If the Axis_placement belongs to a 3D geometric space then the third direction of the coordinate system is defined by the vector product of the x_axis and the y_axis.

EXPRESS specification:

```
*)

  ENTITY Axis_placement
    SUBTYPE OF(Geometric_model_element);
    origin                    :Cartesian_point;
    x_axis                    :Direction;
    y_axis                    :Direction;
  DERIVE
```

```
    dim                          :INTEGER := dim : INTEGER :=
                                  SIZEOF(origin.point_coordinates);;
  WHERE
    WR1      :dim >1;;
    WR2      :dim = SIZEOF(x_axis.direction_coordinates);;
    WR3      :dim = SIZEOF(y_axis.direction_coordinates);;
  END_ENTITY;
(*
```

Attribute definitions:

origin : geometric position of a reference point that forms the origin of the placement coordinate system in the geometric space

x_axis : Direction that defines the first axis of the Axis_placement

y_axis : Diirection that defines the second axis of the Axis_placement

dim : dimensionality of the Axis_placement

NOTE The value of the dimensionality is equal to the number of coordinates in the origin

Formal Propositions:

WR1 the dimensions of the Axis_placement shall be greater than 1.

WR2 the number of coordinates of the reference_direction shall be equal to the number of coordinates of the location of the Axis_placement

WR3 the number of coordinates of the axis shall be equal to the number of coordinates of the origin of the Axis_placement

4.2.8.3 Axis_placement_mapping

An Axis_placement_mapping is the geometric transformation defined by a source Axis_placement and a target Axis_placement that results from the transformation of the source.

NOTE 1 Both instances of Axis_placement have the same dimensions.

NOTE 2 By construction the determinant of the transformation matrix equals one.

REMARK The transformation is computed as the isometric transformation that maps:

— the origin of the source to the origin of the target;

— the x_axis of the source to the x_axis of the target;

— the y_axis of the source to the y_axis of the target.

EXPRESS specification:

*)

```
  ENTITY Axis_placement_mapping;
```

```
    source                    :Axis_placement;
    target                    :Axis_placement;
  WHERE
    WR1      :source\Axis_placement.dim = target\Axis_placement.dim;;
  END_ENTITY;
(*
```

Attribute definitions:

source : the instance of Axis_placement that is the start point of the mapping

target : the instance of Axis_placement that is the end point of the mapping

Formal Propositions:

WR1 the source and the target have the same dimensions

4.2.8.4 Axis_placement_transformation_mapping

An Axis_placement_transformation_mapping is a geometric transformation defined by a source Axis_placement and a target Cartesian_transformation.

NOTE This kind of mapping enables the representation of non-isometric transformations including symmetry and projection transformations.

REMARK The transformation is computed as the transformation that maps:

— the origin of the source onto the translation point of the target;

— the reference_direction of the source onto the first element of the multiplication_matrix of the target;

— the axis of the source onto the second element of the multiplication_matrix of the target;

— if the transformation is a three dimensional transformation then the normalised vector product of the x_axis and the y_axis of the source is mapped onto the third element of the multiplication_matrix of the target.

EXPRESS specification:

*)

```
  ENTITY Axis_placement_transformation_mapping;
    source                    : Axis_placement;
    target                    : cartesian_transformation;
  WHERE
    WR1      : source\Axis_placement.dim = SIZEOF
               (target.translation\Cartesian_point.point_coordinates);;
  END_ENTITY;
(*
```

Attribute definitions:

source : the instance of Axis_placement that is the start point of the mapping

target : the instance of Axis_placement that is the end point of the mapping

Formal Propositions:

WR1 : the source and the target shall have the same dimensionality

4.2.8.5 Cartesian_point

A Cartesian_point is a type of Geometric_model _element that defines a point in a Cartesian coordinate system by a list of up to three numbers.

EXPRESS specification:

*)

```
ENTITY Cartesian_point
  SUBTYPE OF(Geometric_model_element);
  point_coordinates          : LIST [1:3] OF NUMBER;
END_ENTITY;
```

(*

Attribute definitions:

point_coordinates : list of up to three number values that define the Cartesian coordinates of a point

4.2.8.6 Cartesian_transformation_2D

A Cartesian_transformation is a geometric transformation defined in a two dimensional geometric space by a 2*2 matrix and a Cartesian point.

REMARK Let:

— MM be the 2*2 matrix of the Cartesian transformation;

— A be the point of origin of the Cartesian transformation;

— P be a point in the geometric space;

— Q be the result of the application of the transformation to P.

Then, the coordinates of Q are obtained by the formula: Q = MM*P + A

EXPRESS specification:

*)

```
ENTITY Cartesian_transformation_2D
  SUBTYPE OF(Geometric_model_element);
  multiplication_matrix      : ARRAY [1:2] OF Direction;
  translation                : Cartesian_point;
WHERE
  WR1: SIZEOF(multiplication_matrix[1]\Direction.direction_coordinates)=2;
```

```
    WR2: SIZEOF(multiplication_matrix[2]\Direction.direction_coordinates)=2;
    WR3: SIZEOF(translation.point_coordinates)=2;;
  END_ENTITY;
(*
```

Attribute definitions:

multiplication_matrix : array of two instances of Direction that defines the multiplication matrix of the transformation

translation : Cartesian_point that defines the position of the result of the application of the trnsformation to the origin of the geometric space

Formal Propositions:

WR1: the first element of the matrix refers to a two-dimensional Direction

WR2: the second element of the matrix refers to a three dimensional Direction

WR3: the translation point has two coordinates

4.2.8.7 Cartesian_transformation_3D

A Cartesian_transformation_3D is a geometric transformation defined in a three dimensional geometric space by a 3*3 matrix and a Cartesian point.

REMARK Let:

— MM be the 3*3 matrix of the Cartesian transformation;

— A be the point of origin of the Cartesian transformation;

— P be a point in the geometric space;

— Q be the result of the application of the transformation to P.

Then, the coordinates of Q are obtained by the formula: Q = MM*P + A.

EXPRESS specification:

*)

```
  ENTITY Cartesian_transformation_3D
    SUBTYPE OF(Geometric_model_element);
    multiplication_matrix      : ARRAY [1:3] OF Direction;
    translation                : Cartesian_point;
  WHERE
    WR1: SIZEOF(multiplication_matrix[1]\Direction.direction_coordinates)=3;
    WR2: SIZEOF(multiplication_matrix[2]\Direction.direction_coordinates)=3;
    WR3: SIZEOF(multiplication_matrix[3]\Direction.direction_coordinates)=3;
    WR4: SIZEOF(translation.point_coordinates)=3;
  END_ENTITY;
(*
```

Attribute definitions:

multiplication_matrix : the array of three instances of Direction that defines the multiplication matrix of the transformation

translation : Cartesian_point that defines the poition of the result of the application of the transformation to the origin of the geometric space.

Formal Propositions:

WR1: the first element of the matrix refers to a three-dimensional Direction.

WR2: the second element of the matrix refers to a three-dimensional Direction.

WR3: the third element of the matrix refers to a three-dimensional Direction.

WR4: the translation point has three coordinates

4.2.8.8 Direction

A Direction is a type of Geometric_model_element that defines a direction vector in either two or three dimensional space.

NOTE 1 Only the ratios of the components are significant.

NOTE 2 The components of this entity are not normalised. If a unit vector is required then it is normalised before use.

EXPRESS specification:

*)

```
  ENTITY Direction
    SUBTYPE OF(Geometric_model_element);
    direction_coordinates      : LIST [2:3] OF NUMBER;
  WHERE
    WR1: SIZEOF(QUERY(tmp <* direction_coordinates | tmp <> 0.0)) > 0;
  END_ENTITY;
(*
```

Attribute definitions:

direction_coordinates : list of two or three number values whose ratios define the direction of the vector in the Geometric_coordinate_space.

Formal Propositions:

WR1: The magnitude of the direction vector shall be greater than zero.

4.2.8.9 Geometric_coordinate_space

A Geometric_coordinate_space is a representation context that defines a coordinate space where geometric elements can be defined.

NOTE 1 There are at least two units specified for Geometrical_coordinate_space: one length unit and one plane angle unit. The length unit applies to each coordinate direction.

EXAMPLE The length unit millimetre and the angle unit radian are units that can be assigned to a Geometric_coordinate _space.

NOTE 2 The origin for values of coordinates is implicitly defined as the point whose coordinates are all zero.

EXPRESS specification:

*)

```
  ENTITY Geometric_coordinate_space;
    dimension_count            : INTEGER;
  WHERE
    WR1: dimension count > 0;
  END_ENTITY;
(*
```

Attribute definitions:

dimension_count : number of dimensions for an instance of Geometric_coordinate_space

Formal Propositions:

WR1: The dimensions of the coordinate space are greater than zero.

4.2.8.10 Geometric_model

A Geometric_model is a type of representation dedicated to the description of geometric constructs.

NOTE 1 The Geometric_model is founded in Geometric_coordinate_space.

NOTE 2 The items of a Geometric_model are instances of Geometric_model_element.

EXPRESS specification:

*)

```
  ENTITY Geometric_model;
    context_of_items           : Geometric_coordinate_space;
    items                      : SET [1:?] OF Geometric_model_element;
    description                : STRING;
    model_name                 : STRING;
  END_ENTITY;
(*
```

Attribute definitions:

context_of_items : the Geometric_coordinate_space in which a Geometric_model is defined

items : set of instances of Geometric_model_element that are included in the Geometric_model

description : words to provide information about a Geometric_model

model_name : label by which a Geometric_model can be identified.

4.2.8.11 Geometric_model_element

A Geometric_model_element is the data type for the identification of a geometric construct.

NOTE Only non-abstract specialisations of Geometric_model_element data type can be instantiated.

EXPRESS specification:

*)

```
ENTITY Geometric_model_element
  ABSTRACT SUPERTYPE OF (ONEOF(Cartesian_point, Direction, Axis_placement,
                    Cartesian_transformation_3D,
                    Cartesian_transformation_2D));
END_ENTITY;
```

(*

4.2.8.12 Shape

A Shape is the external form of something as produced by their outline.

EXPRESS specification:

*)

```
ENTITY Shape;
  has_dimension              : SET [1:?] OF Shape_dimension;
  has_representation         : Shape_representation;
  shape_name                 : STRING;
  shape_description          : STRING;
END_ENTITY;
```

(*

Attribute definitions:

has_dimension : dimensions of the Shape

has_representation : representation of the Shape

NOTE A Shape can have several representations

shape_name : word or words by which a Shape can be identified

shape_description : text to describe the Shape

4.2.8.13 Shape_defined_in_external_reference

A Shape_defined_external_reference is a type of Shape that is defined in an external source.

EXPRESS specification:

*)

```
  ENTITY Shape_defined_in_external_reference
    SUBTYPE OF(Shape);
    reference_select          :classification_source_select;
  END_ENTITY;
(*
```

Attribute definitions:

reference_select : source of the information

4.2.8.14 Shape_dimension

A Shape_dimension is a measurable extent of a Shape**.**

EXPRESS specification:

*)

```
  ENTITY Shape_dimension;
    dimension_value           : Measure_item;
    dimension_name            : STRING;
  END_ENTITY;
(*
```

Attribute definitions:

dimension_value : magnitude of a dimension

dimension_name : label by which the dimension is known

4.2.8.15 Shape_element

A Shape_element is an identifiable part of the external form or outline of something.

EXPRESS specification:

*)

```
  ENTITY Shape_element;
    of_shape                  : Shape;
    element_name              : STRING;
    description               : STRING;
  END_ENTITY;
(*
```

Attribute definitions:

of_shape : Shape of which the Shape_element is a part

element_name : label by which the Shape_element is known

description : text to describe the Shape_element

4.2.9 Application objects for the geometric_tolerance UoF

This subclause specifies the data types and application objects associated with the geometric_tolerance UoF.

4.2.9.1 Angularity_tolerance

An Angularity_tolerance is a type of Geometric_tolerance that specifies the allowed deviation from a requirement that a feature should be at a given angle to the reference datum.

EXPRESS specification:

```
*)

  ENTITY Angularity_tolerance
    SUBTYPE OF(Geometric_tolerance);
    ref_datum                  : SET [1:2] OF Reference_datum;
  END_ENTITY;
(*
```

Attribute definitions:

ref_datum : datum system that defines the reference for this type of Geometric_tolerance

4.2.9.2 Coaxiality_tolerance

A Coaxiality_tolerance is a type of Geometric_tolerance that specifies the allowed deviation from a requirement that the axis of a Shape_element and the datum should coincide.

EXPRESS specification:

```
*)

  ENTITY Coaxiality_tolerance
    SUBTYPE OF(Geometric_tolerance);
    ref_datum                  : SET [1:2] OF Reference_datum;
  END_ENTITY;
(*
```

Attribute definitions:

ref_datum : datum system that defines the reference for this type of

Geometric_tolerance

4.2.9.3 Concentricity_tolerance

A Concentricity_tolerance is a type of Geometric_tolerance that specifies the allowed deviation from a requirement that a feature shares the same centre as the reference datum.

EXPRESS specification:

```
*)

  ENTITY Concentricity_tolerance
    SUBTYPE OF(Geometric_tolerance);
    ref_datum                  : SET [1:2] OF Reference_datum;
  END_ENTITY;
(*
```

Attribute definitions:

ref_datum : datum system that defines the reference for this type of Geometric_tolerance

4.2.9.4 Cylindricity_tolerance

A Cylindricity_tolerance is a type of Geometric_tolerance that specifies the allowed deviation from a requirement that a surface should be a cylinder.

EXPRESS specification:

```
*)

  ENTITY Cylindricity_tolerance
    SUBTYPE OF(Geometric_tolerance);
  END_ENTITY;
(*
```

4.2.9.5 Flatness_tolerance

A Flatness_tolerance is a type of Geometric_tolerance that specifies the allowed deviation from a requirement that a surface should be flat.

EXPRESS specification:

```
*)

  ENTITY Flatness_tolerance
    SUBTYPE OF(Geometric_tolerance);
  END_ENTITY;
(*
```

4.2.9.6 Geometric_tolerance

A Geometric_tolerance is the allowed deviation from the shape of an object.

EXPRESS specification:

*)

```
ENTITY Geometric_tolerance;
  tolerance_value            : Measure_item;
  applied_to                 : Shape_element;
  modification               : Tolerance_condition;
END_ENTITY;
(*
```

Attribute definitions:

tolerance_value : magnitude of the width of the tolerance zone

applied_to : reference to the Shape_element to which the Geometric_tolerance is applied

modification : Tolerance_condition that is associated with the application of the Geometrical_tolerance

4.2.9.7 Geometric_tolerance_relationship

A Geometric_tolerance_relationship is an association between two instances of Geometric_tolerance.

EXPRESS specification:

*)

```
ENTITY Geometric_tolerance_relationship;
  related_tolerance          : Geometric_tolerance;
  relating_tolerance         : Geometric_tolerance;
  relation_type              : STRING;
END_ENTITY;
(*
```

Attribute definitions:

related_tolerance : instance of the Geometric_tolerance that is one part of the association.

NOTE If one element of the association is dependent on the other then this is the dependent one

relating_tolerance : one of the instances of the Geometric_tolerance that is part of the association

relation_type : text that defines the meaning of the relationship

4.2.9.8 Parallelism_tolerance

A Parallelism_tolerance is a type of Geometric_tolerance that specifies the allowed deviation from a requirement that a surface shall be parallel to a reference datum.

EXPRESS specification:

```
*)

  ENTITY Parallelism_tolerance
    SUBTYPE OF(Geometric_tolerance);
    ref_datum                  : SET [1:2] OF Reference_datum;
  END_ENTITY;
(*
```

Attribute definitions:

ref_datum : datum system that defines the reference for this type of Geometric_tolerance

4.2.9.9 Perpendicularity_tolerance

A Perpendicularity_tolerance is a type of Geometric_tolerance that specifies the allowed deviation from a requirement that a feature shall be perpendicular to a reference datum.

EXPRESS specification:

```
*)

  ENTITY Perpendicularity_tolerance
    SUBTYPE OF(Geometric_tolerance);
    ref_datum                  : SET [1:3] OF Reference_datum;
  END_ENTITY;
(*
```

Attribute definitions:

ref_datum : datum system that defines the reference for this type of Geometric_tolerance

4.2.9.10 Position_tolerance

A Position_tolerance is a type of Geometric_tolerance that specifies the allowed deviation from a requirement that the position of a shape element and the datum should coincide.

EXPRESS specification:

```
*)

  ENTITY Position_tolerance
    SUBTYPE OF(Geometric_tolerance);
    ref_datum                  : SET [1:3] OF Reference_datum;
```

```
  END_ENTITY;
(*
```

Attribute definitions:

ref_datum : datum system that defines the reference for this type of Geometric_tolerance

4.2.9.11 Reference_datum

A Reference_datum is the datum for the definition of a type of Geometric_tolerance.

EXPRESS specification:

```
*)

  ENTITY Reference_datum;
    datum_reference            : Datum;
  END_ENTITY;
(*
```

Attribute definitions:

datum_reference : reference to the datum for the type of Geometric_tolerance

4.2.9.12 Roundness_tolerance

A Roundness_tolerance is a type of Geometric_tolerance that specifies the allowed deviation from a requirement that a surface should be spherical.

EXPRESS specification:

```
*)

  ENTITY Roundness_tolerance
    SUBTYPE OF(Geometric_tolerance);
  END_ENTITY;
(*
```

4.2.9.13 Straightness_tolerance

A Straightness_tolerance is a type of Geometric_tolerance that specifies the allowed deviation from a requirement that a line or edge should be straight.

EXPRESS specification:

```
*)

  ENTITY Straightness_tolerance
    SUBTYPE OF(Geometric_tolerance);
  END_ENTITY;
(*
```

4.2.9.14 Symmetry_tolerance

A Symmetry_tolerance is a type of Geometric_tolerance that specifies the allowed deviation from a requirement that a feature should be symmetrical with respect to the datum reference.

EXPRESS specification:

*)

```
ENTITY Symmetry_tolerance
  SUBTYPE OF(Geometric_tolerance);
  ref_datum                  : SET [1:3] OF Reference_datum;
END_ENTITY;
```

(*

Attribute definitions:

ref_datum : datum system that defines the reference for this type of Geometric_tolerance

4.2.9.15 Tolerance_zone

A Tolerance_zone is a region of an object where a Geometric_tolerance applies.

EXPRESS specification:

*)

```
ENTITY Tolerance_zone;
  zone_for                   : SET [1:?] OF Geometric_tolerance;
  form_type                  : STRING;
END_ENTITY;
```

(*

Attribute definitions:

zone_for : set of Geometric_tolerance objects for which the Tolerance_zone defines the zone

form_type : description of the shape of the Tolerance_zone

EXAMPLE cylindrical, or spherical are examples of form_type

4.2.9.16 Tolerance_zone_definition

A Tolerance_zone_definition is a specification of the set of Geometric_tolerance for a zone defined by the Tolerance_zone**.**

EXPRESS specification:

*)

```
ENTITY Tolerance_zone_definition;
```

```
    defining                    : Tolerance_zone;
    first_element               : Shape_element;
    second_element              : Shape_element;
  END_ENTITY;
(*
```

Attribute definitions:

defining : Tolerance_zone that is defined by the Tolerance_zone_definition

first_element : Shape_element that represents one of the bounding elements of the Tolerance_zone

second_element : Shape_element that represents the other bounding element of the Tolerance _zone

4.2.9.17 Total_runout_tolerance

A Total_runout_tolerance is a type of Geometric_tolerance that specifies the allowed deviation from the required runout on a cylindrical or parallel-sided object.

EXPRESS specification:

```
*)

  ENTITY Total_runout_tolerance
    SUBTYPE OF(Geometric_tolerance);
    ref_datum                   : SET [1:2] OF Reference_datum;
  END_ENTITY;
(*
```

Attribute definitions:

ref_datum : datum system that defines the reference for this type of Geometric_tolerance

4.2.10 Application objects for the location UoF

This subclause specifies the data types and application objects associated with the location UoF.

4.2.10.1 Address_location_representation

An Address_location_representation is a type of Location_representation that identifes a place specified by an instance of Address.

EXPRESS specification:

```
*)

  ENTITY Address_location_representation
    SUBTYPE OF(Location_representation);
    location_address            : Address;
```

```
  END_ENTITY;
(*
```

Attribute definitions:

location_address : instance of Address that specifies the place

4.2.10.2 Global_location_representation

A Global_location_representation is a type of Location_representation that identifes a place by global coordinates.

EXPRESS specification:

*)

```
  ENTITY Global_location_representation
    SUBTYPE OF(Location_representation);
    geographical_area : STRING;
    latitude                  : Measure_item;
    longitude                 : Measure_item;
    altitude                  : OPTIONAL Measure_item;
  END_ENTITY;
(*
```

Attribute definitions:

geographical_area : name of the area with defined boundaries in which the place is located

latitude : angular displacement of a place north or south of the equator

NOTE 1 The sense of the displacement can be included in the definition of the unit of measure

longitude : angular displacement of a place from the Greenwich meridian

NOTE 2 The sense of the displacement can be included in the definition of the unit of measure

altitude : elevation of the place above some standard level

NOTE 3 This attribute need not be specified.

NOTE 4 The standard level can be defined in the unit of measure.

4.2.10.3 Location

A Location is a particular place or position.

EXPRESS specification:

*)

```
  ENTITY Location;
    representation             : Location_representation;
    location_name              : STRING;
    location_id                : STRING;
    location_description       : STRING
  END_ENTITY;
(*
```

Attribute definitions:

representation : reference to the collection of information that defines a particular place or position

location_name : label by which a Location is known and can be referred to

location_id : label that distinguishes a Location

location_description : text to provide general information about a Location

4.2.10.4 Location_representation

A Location_representation is the collection of information that defines a Location.

NOTE Location can have more than one representation.

EXPRESS specification:

*)

```
  ENTITY Location_representation;
    name                       : STRING;
    description                : STRING;
  END_ENTITY;
(*
```

Attribute definitions:

name : label by which the Location_representation is known and can be referred to

description : text to provide further information about the Location_representation

4.2.10.5 Location_representation_relationship

A Location_representation_relationship is an association between two instances of Location_representation.

EXPRESS specification:

*)

```
  ENTITY Location_representation_relationship;
    relating                   : Location_representation;
```

```
    related                    : Location_representation;
    relation_type              : STRING;
    description                 : STRING;
  END_ENTITY;
(*
```

Attribute definitions:

relating : one of the instances of Location_representation that participates in the association

related : the other instance of Location_representation that participates in the association

NOTE If one element of the association is dependent on the other then this attribute is the dependent one.

relation_type : text that defines the meaning of the association

description : text that explains the nature of the relationship

4.2.10.6 Organisation_location_representation

An Organisation_location_representation is a type of Location_representation that is the representation of the Location in the context of an organization.

EXPRESS specification:

*)

```
  ENTITY Organisation_location_representation
    SUBTYPE OF(Location_representation);
    location_identifications   : SET [1:?] OF
                                 Organisational_location_identification;
    organization               : Organisation;

  END_ENTITY;
(*
```

Attribute definitions:

location_identifications : set of information objects that provides the identification of the place

organization : Organisation in which the identification of the place is meaningful

4.2.10.7 Organisational_location_identification

An Organisational_location_identification is information that identifies a place in an organization.

EXPRESS specification:

*)

```
  ENTITY Organisational_location_identification;
    identification_type        : STRING;
    identification_value       : STRING;
  END_ENTITY;
(*
```

Attribute definitions:

identification_type : text that characterises the identification of the place

identification_value : identifier of the place in the organization

4.2.10.8 Product_location_representation

A Product_location_representation is a type of Location_representation that specifies a place in the context of a product.

EXPRESS specification:

```
*)

  ENTITY Product_location_representation
    SUBTYPE OF(Location_representation);
    referenced_product         : Individual_product;
    location_identification    : STRING;
  END_ENTITY;
(*
```

Attribute definitions:

referenced_product : product that provides the context for the specification of the place

location_identification : text that identifies a place in the context of a product

4.2.10.9 Regional_grid_location_representation

A Regional_grid_location_representation is a type of Location_representation that defines a place in the context of a regional grid of coordinates.

EXPRESS specification:

```
*)

  ENTITY Regional_grid_location_representation
    SUBTYPE OF(Location_representation);
    SELF\Location_representation.description : STRING;
    SELF\Location_representation.name        : STRING;
    regional_coordinates                     : NUMBER
  END_ENTITY;
(*
```

Attribute definitions:

description : text that describes the regional grid system

name : label by which the regional grid is known and can be referred to

regional_coordinates : values of the coordinates on the regional grid

4.2.11 Application objects for the measure UoF

This subclause specifies the data types and application objects associated with the measure UoF.

4.2.11.1 value_qualifier

A value_qualifier is a select data type that provides a list of alternatives to which an instance of Qualified_measure can be selectively associated.

An instance of Qualified_measure can be associated with an instance of one of:

— Qualifier_uncertainty;

— Qualifier_precision;

— Qualifier_type.

EXPRESS specification:

```
*)

  TYPE value_qualifier = SELECT
    (Qualifier_uncertainty,
    Qualifier_precision,
    Qualifier_type);
  END_TYPE;
(*
```

4.2.11.2 Context_dependent_unit

A Context_dependent_unit is a type of Unit that is not related to the SI system of units.

EXAMPLE The number of parts in an assembly is a physical quantity that can be measured in multiples of a unit named 'part'. Such a unit cannot be defined in the SI system of units.

EXPRESS specification:

```
*)

  ENTITY Context_dependent_unit
    SUBTYPE OF(Unit);
  WHERE
    WR1      : EXISTS(SELF\Unit.unit_name);
  END_ENTITY;
(*
```

Formal Propositions:

WR1: The Context_dependent_unit has a name

4.2.11.3 Count_measure

A Count_measure is a type of Measure_item that is the number of times that something happened.

EXPRESS specification:

*)

```
ENTITY Count_measure
  SUBTYPE OF(Measure_item);
  count_value                : NUMBER;
END_ENTITY;
(*
```

Attribute definitions:

count_value : quantity that is the magnitude of the count

4.2.11.4 Derived_unit

A Derived_unit is a type of Unit that is an expression of combinations of other Unit objects.

EXAMPLE Newtons per square millimetre is a Derived_unit.

EXPRESS specification:

*)

```
ENTITY Derived_unit
  SUBTYPE OF(Unit);
  unit_elements : SET [1:?] OF Derived_unit_element;
END_ENTITY;
(*
```

Attribute definitions:

unit_elements : set of Unit and their exponents whose product defines a Derived_unit

NOTE Each element is one term in the expression of Derived_unit

4.2.11.5 Derived_unit_element

A Derived_unit_element is an association of an exponent with a Unit.

EXPRESS specification:

*)

```
  ENTITY Derived_unit_element;
    exponent                  : REAL;
    base_unit                 : Unit;
  END_ENTITY;
(*
```

Attribute definitions:

exponent : power to which 10 is raised as the multiplier of the base_unit in the expression for Derived_unit.

base_unit : instance of Unit for a term in a Derived_unit

4.2.11.6 Descriptive_measure

A Descriptive_measure is a value in descriptive form for a quantity by comparison.

EXPRESS specification:

*)

```
  ENTITY Descriptive_measure
    SUBTYPE OF(Measure_item);
    descriptive_value : STRING;
  END_ENTITY;
(*
```

Attribute definitions:

descriptive_value : text that provides the value of a Descriptive_measure_item

EXAMPLE 'blue' can be the value of the Descriptive_measure_item that is a measure of the colour of a physical object.

4.2.11.7 Duration

A Duration is a type of Numerical_measure_item that defines the magnitude of a time interval.

EXPRESS specification:

*)

```
  ENTITY Duration
    SUBTYPE OF(Numerical_measure_value);
  WHERE
    WR1: 'TIME_UNIT' IN TYPEOF(SELF\Numerical_measure_value.unit);
  END_ENTITY;
(*
```

Formal Propositions:

WR1 : the unit of Duration is a Time_unit

4.2.11.8 Maths_expression_value

A Maths_value is a type of Measure_item where the value is represented as a mathematical expression.

EXPRESS specification:

```
*)

  ENTITY Maths_expression_value
    SUBTYPE OF(Measure_item);
    unit                        :Unit;
  END_ENTITY;
(*
```

Attribute definitions:

unit : the base quantity for the expression

4.2.11.9 Measure_item

A Measure_item is the magnitude of a quantity or characteristic.

EXPRESS specification:

```
*)

  ENTITY Measure_item
    ABSTRACT SUPERTYPE;
    measure_name                : STRING;
  END_ENTITY;
(*
```

Attribute definitions:

measure_name : label that identifies the quantity which the Measure_item represents

4.2.11.10 Numerical_measure_value

A Numerical_measure_value is a type of Measure_item that is a multiple of a numerical value and a unit of a quantity.

EXAMPLE 600 Mega Pascals is an instance of a Numerical_measure_value.

EXPRESS specification:

```
*)
```

```
  ENTITY Numerical_measure_value
    SUBTYPE OF(Measure_item);
    number_value               :NUMBER;
    unit                       :Unit;
  END_ENTITY;
(*
```

Attribute definitions:

number_value : the multiple of a unit quanitity

unit : the base quantity for the numerical measure.

4.2.11.11 Qualified_expression

A Qualifed_expression is an association of a mathematical expression with the measures of uncertainty and reliability of the expression.

EXPRESS specification:

*)

```
  ENTITY Qualified_expression;
    qualifiers                 :SET [1:?] OF value_qualifier;
    expression_qualified       :Maths_expression_value;
  END_ENTITY;
(*
```

Attribute definitions:

qualifiers : items that constitute the set of qualifiers for an expression

expression_qualified : mathematical construct for which the reliability and or uncertanty are defined

4.2.11.12 Qualified_measure

A Qualified_measure is an association of a numerical_measure_value with the measures of the uncertainty and reliability of the measurement.

EXPRESS specification:

*)

```
  ENTITY Qualified_measure;
    qualifiers                 :SET [1:?] OF value_qualifier;
    measure_qualified          :Numerical_measure_value;
  WHERE
    WR1      :SIZEOF(QUERY(temp <* qualifiers |'QUALIFIER_PRECISION' IN
              TYPEOF(temp))) <2;;
  END_ENTITY;
(*
```

Attribute definitions:

qualifiers : items that constitute the set of qualifiers for a numerical value

measure_qualified : numerical value for which the reliability and or uncertanty are defined

Formal Propositions:

WR1: At most one of the elements of the qualifier attribute can be a precision qualifier.

4.2.11.13 Qualifier_precision

A Qualifier_precision is the number of significant figures in the numerical representation of a value.

EXPRESS specification:

*)

```
ENTITY Qualifier_precision;
  value_precision            : INTEGER;
END_ENTITY;
```

(*

Attribute definitions:

value_precision : number of significant figures of the value

4.2.11.14 Qualifier_type

A Qualifier_type is an identifer for a category of the reliability of a value of a quantity.

EXPRESS specification:

*)

```
ENTITY Qualifier_type;
  name                       : STRING;
END_ENTITY;
```

(*

Attribute definitions:

name : word or group of words by which the type or reliability of the value is identified.

EXAMPLE Typical values for this attribute can include: 'measured', calculated', 'nominal', 'maximum', 'minimum', 'design allowable', 'A-basis statistical', etc.

4.2.11.15 Qualifier_uncertainty

A Qualifier_uncertainty defines the uncertainty of the value of a quantity.

EXPRESS specification:

```
*)

  ENTITY Qualifier_uncertainty
    SUPERTYPE OF (ONEOF(Uncertainty_qualitative, Uncertainty_standard));
    measure_name               : STRING;
    description                : STRING;
  END_ENTITY;
(*
```

Attribute definitions:

measure_name : kind of measure for which the entity defines the uncertainty

EXAMPLE Values for measure_name can include 'distance uncertainty', 'angular uncertainty', etc.

description : further information about the uncertainty of the value

4.2.11.16 Range_measure

A Range_measure is a type of Measure_item where the value falls between the limits of a range of values.

EXPRESS specification:

```
*)

  ENTITY Range_measure
    SUBTYPE OF(Measure_item);
    lower_limit                : Measure_item;
    upper_limit                : Measure_item;
  END_ENTITY;
(*
```

Attribute definitions:

lower_limit : Measure_item that is the lower limit of a range

upper_limit : Measure_item that is the upper limit of a range

4.2.11.17 Ratio_measure

A Ratio_measure is a type of Measure_item that is the value of the relation between two quantities that are of the same kind.

NOTE A Ratio_measure has no unit or the value of the unit is 1.

EXPRESS specification:

*)

```
ENTITY Ratio_measure
  SUBTYPE OF(Measure_item);
  ratio_value                : NUMBER;
END_ENTITY;
(*
```

Attribute definitions:

ratio_value : numerical value of the Ratio_measure

4.2.11.18 Uncertainty_expanded

An Uncertainty_expanded is a type of Uncertainty_standard that specifies the coverage factor of an uncertainty.

EXPRESS specification:

*)

```
ENTITY Uncertainty_expanded
  SUBTYPE OF(Uncertainty_standard);
  coverage_factor            : REAL;
END_ENTITY;
(*
```

Attribute definitions:

coverage_factor : multiplier for the uncertainty of a value

4.2.11.19 Uncertainty_qualitative

An Uncertainty_qualitative is a type of Qualifer_uncertainty whose value is expressed by comparison.

EXPRESS specification:

*)

```
ENTITY Uncertainty_qualitative
  SUBTYPE OF(Qualifier_uncertainty);
  uncertainty_value          : STRING;
END_ENTITY;
(*
```

Attribute definitions:

uncertainty_value : uncertainty of a value by comparison

4.2.11.20 Uncertainty_standard

An Uncertainty_standard is the standard uncertainty or the combined standard uncertainty of a value.

EXPRESS specification:

*)

```
  ENTITY Uncertainty_standard
    SUBTYPE OF(Qualifier_uncertainty);
    uncertainty_value          : REAL;
  END_ENTITY;
(*
```

Attribute definitions:

uncertainty_value : numerical uncertainty of the value

4.2.11.21 Unit

A Unit is the standard amount used as a basis for expressing the magnitude of a quantity.

EXPRESS specification:

*)

```
  ENTITY Unit
    SUPERTYPE OF (ONEOF(Derived_unit, Context_dependent_unit));
    unit_name                  : STRING;
    si_unit                    : BOOLEAN;
  END_ENTITY;
(*
```

Attribute definitions:

unit_name : label that identifies a standard quantity

si_unit : Boolean value that identifies if the standard quanitity is defined in the SI system

4.2.11.22 Unit_relationship

A Unit_relationship is the association between two instances of Unit.

EXAMPLE Two instances of Unit could be related by a conversion factor or a mathematical relationship.

EXPRESS specification:

*)

```
  ENTITY Unit_relationship;
    relating_unit              : Unit;
```

```
    related_unit                : Unit;
    conversion_factor           : OPTIONAL Numerical_measure;
    description                 : OPTIONAL STRING;
    relation_type               : STRING;
  END_ENTITY;
(*
```

Attribute definitions:

relating_unit : one of the instances of Unit in the association

related_unit : other instance of Unit in the association

NOTE 1 If one of the instances in the association is dependent on the other then this is the dependent one.

conversion_factor : numerical value that relates one Unit to another

NOTE 2 The value of this attribute need not be specified

description : text to provide information about the association

relation_type : text that defines the nature of the relationship

4.2.12 Application objects for the person_organisation UoF

This subclause specifies the data types and application objects associated with the person_organisation UoF.

4.2.12.1 person_organisation

A person_organisation is a select data type that provides a list of alternatives to which items can be selectively associated. An item can be associated with either:

— Person_in_organisation;

— Organisation.

EXPRESS specification:

*)

```
  TYPE person_organisation = SELECT
    (Person_in_organisation,
    Organisation);
  END_TYPE;
(*
```

4.2.12.2 Address

An Address is information to identify a location where a Person or an Organisation can receive a communication.

EXPRESS specification:

*)

```
ENTITY Address
  ABSTRACT SUPERTYPE;
END_ENTITY;
(*
```

4.2.12.3 Facsimile_address

A Facsimile_address is a type of Address for the receipt of communications by facsimile transfer.

EXPRESS specification:

*)

```
ENTITY Facsimile_address
  SUBTYPE OF(Address);
  facsimile_number          : STRING;
END_ENTITY;
(*
```

Attribute definitions:

facsimile_number : identifier of the location where communications by facsimile transfer can be received

4.2.12.4 Network_address

A Network_address is a type of Address for the receipt of communications by digital electronic networks.

EXPRESS specification:

*)

```
ENTITY Network_address
  SUBTYPE OF(Address);
  url                       : STRING;
END_ENTITY;
(*
```

Attribute definitions:

url : Universal Resouce Locator - identifier of the location where communications over digital electronic networks can be received

4.2.12.5 Organisation

An Organisation is an association of people with a particular purpose.

EXPRESS specification:

```
*)

   ENTITY Organisation;
     of_type                    : Organisation_type;
     org_name                   : STRING;
     org_id                     : STRING;
     org_address                : Address;
     org_qualification          : SET [0:?] OF Qualification;
   END_ENTITY;
(*
```

Attribute definitions:

of_type : Organisation_type of which an instance of Organisation is a member

org_name : label by which an instance of Organisation is known

org_id : label that distinguishes an instance of Organisation

org_address : place where messages to an Organisation can be received

org_qualification : set of Qualification possessed by an Organisation

4.2.12.6 Organisation_relationship

An Organisation_relationship is an association between two instances of Organisation.

EXPRESS specification:

```
*)

   ENTITY Organisation_relationship;
     relation_type              : STRING;
     description                : STRING;
     related_org                : Organisation;
     relating_org               : Organisation;
   END_ENTITY;
(*
```

Attribute definitions:

relation_type : text that defines the meaning of the association

description : text to provide further information about the relationship

related_org : one instance of Organisation participating in the association

NOTE If one element of the relationship is dependent on the other then this attribute is the dependent one.

relating_org : the other of the instances of Organisation participating in the relationship

4.2.12.7 Organisation_type

An Organisation_type is a class of Organisation.

EXPRESS specification:

*)

```
  ENTITY Organisation_type;
    name                        : STRING;
    description                 : STRING;
  END_ENTITY;
(*
```

Attribute definitions:

name : label by which an Organisation_type is known and can be referred to

description : text that provides information about an Organisation_type

4.2.12.8 Person

A Person is a human being regarded as an individual.

EXPRESS specification:

*)

```
  ENTITY Person;
    person_name                 : LIST [1:?] OF STRING;
    prefix_title                : OPTIONAL LIST [1:?] OF STRING;
    suffix_title                : OPTIONAL LIST [1:?] OF STRING;
    person_qualification        : SET [0:?] OF Qualification;
  END_ENTITY;
(*
```

Attribute definitions:

person_name : word or words that identify an individual human being

prefix_title : word or words that precede the identification of an individual human being

suffix_title : word or words that succeed the identification of an individual human being

4.2.12.9 Person_in_organisation

A Person_in_organisation is a human being in an Organisation.

EXPRESS specification:

*)

```
ENTITY Person_in_organisation;
  organization                : Organisation;
  person                      : Person;
  person_role                 : STRING;
  person_organisation_address :  Address;
END_ENTITY;
(*
```

Attribute definitions:

organisation : reference to the Organisation with which the Person_in_organisation is associated

person : individual human being

person_role : text to describe the purpose or function of the individual human being in the Organisation

person_organisation_address : place where messages to the person can be received

4.2.12.10 Postal_address

A Postal_address is a type of Address for the receipt of communications by postal service delivery.

EXPRESS specification:

*)

```
ENTITY Postal_address
  SUBTYPE OF(Address);
  postal_box                 : OPTIONAL STRING;
  street_number              : STRING;
  street_name                : STRING;
  postal_town                : STRING;
  post_code                  : STRING;
  region                     : OPTIONAL STRING;
  country                    : OPTIONAL STRING;
END_ENTITY;
(*
```

Attribute definitions:

postal_box : identifier for a location where items are delivered for collection by the receiver

street_number : identifier for a location on a street

street_name : label by which a street is known and can be identified

postal_town : identifier for the locality where the street_name is located

post_code : group of letters and or numbers to assist the sorting of surface mail

region : identifier for the area of a country where the postal town is located

country : nation with its own government occupying a particular territory with defined boundaries

4.2.12.11 Qualification

A Qualification defines the capability that makes a Person or Organisation suitable for carrying out a particular task or action.

EXPRESS specification:

*)

```
ENTITY Qualification;
  of_type                  : Qualification_type;
  qual_name                : STRING;
  qual_id                  : STRING;
  qual_descr               : STRING;
END_ENTITY;
(*
```

Attribute definitions:

of_type : type of capability of which Qualification is a member

qual_name : label by which an instance of Qualification is known and can be referred to

qual_id : label that distinguishes an instance of Qualification

qual_descr : text that provides further information about an instance of Qualification

4.2.12.12 Qualification_relationship

A Qualification_relationship is an association between two instances of Qualification.

EXPRESS specification:

*)

```
ENTITY Qualification _relationship;
  relating                 : Qualification_type;
  related                  : Qualification_type;
  relation_type            : STRING;
  description              : STRING;
END_ENTITY;
(*
```

Attribute definitions:

relating : one of the instances of Qualification that participates in the relationship

related : the other instance of Qualification that participates in the relationship

NOTE If one of the elements of the relationship is dependent on the other then this attribute is the dependent one.

relation_type : text that defines the meaning of the relationship

description : text to provide further information about the relationship

4.2.12.13 Telephone_address

A Telephone_address is a type of Address for the receipt of communications by telephone.

EXPRESS specification:

*)

```
ENTITY Telephone_address
  SUBTYPE OF(Address);
  telephone_number          : STRING;
END_ENTITY;
```

(*

Attribute definitions:

telephone_number : identifier of the location where communications by telephone can be received

4.2.13 Application objects for the product UoF

This subclause specifies the data types and application objects associated with the product UoF.

4.2.13.1 resourced_item

A resourced_item is a select data type that provides a mechasim for associating an instance of Resource_item with an instance of one of:

— Activity_as_planned;

— Activity_as_realised.

EXPRESS specification:

*)

```
TYPE resourced_item = SELECT
  (Activity_as_realised,
  Activity_as_planned);
END_TYPE;
```

(*

4.2.13.2 Individual_product

An Individual_product is an instance of a Product_type.

EXPRESS specification:

```
*)

  ENTITY Individual_product;
    description               :STRING;
    id                        :STRING;
    of_type                   :Product_type;
  END_ENTITY;
(*
```

Attribute definitions:

description : text to provide further information about the Individual_product

id : label that distinguishes the Individual_product

of_type : class of product of which the Individual_product is a member

4.2.13.3 Individual_product_feature

An Individual_product_feature is a characteristic of an individual product that cannot exist independently of the product.

EXPRESS specification:

```
*)

  ENTITY Individual_product_feature;
    feature_specification      :OPTIONAL Specification;
    feature_description        :STRING;
    feature_name               :STRING;
    feature_shape              :OPTIONAL Shape;
    feature_location           :OPTIONAL Product_location_representation;
    of_product                 :Individual_product;
  END_ENTITY;
(*
```

Attribute definitions:

feature_specification : formal description of the characteristics of the Individual_product_feature

feature_description : text that provides a narrative account of the charactersitics of the Individual_product_feature

feature_name : label by which an Individual_product_feature can be identified and is known

feature_shape : geometric form of the Individual_product_feature

feature_location : position of an Individual_product_feature in relation to the Individual_product

of_product : reference to the individual product of which the feature is a part

4.2.13.4 Product_as_planned

A Product_as_planned is a type of Individual_product that is intended to be made.

EXPRESS specification:

*)

```
ENTITY Product_as_planned
  SUBTYPE OF(Individual_product);
  plan                        : Product_type;
END_ENTITY;
```

(*

Attribute definitions:

plan : identification of the Product_type that is intended to be used

4.2.13.5 Product_as_realised

A Product_as_realised is a type of Individual_product that actually occurred.

EXPRESS specification:

*)

```
ENTITY Product_as_realised
  SUBTYPE OF(Individual_product);
  realisation_of              : Product_as_planned;
  of_type                     : Product_type;
END_ENTITY;
```

(*

Attribute definitions:

realisation_of : instance of the product that was planned

of_type : class of product of which the Product_as_realised is a member

4.2.13.6 Product_feature_relationship

A Product_feature_relationship defines the association between two instances of Individual_product_feature.

EXPRESS specification:

*)

```
  ENTITY Product_feature_relationship;
    relating                  : Individual__product_feature;
    related                   : Individual__product_feature;
    relation_type             : STRING;
    description               : OPTIONAL STRING;
  END_ENTITY;
(*
```

Attribute definitions:

relating : one of the instances of Individual_product_feature that participates in the association

related : the other of the instances of Individual_product_feature that participates in the association.

NOTE If one of the instances depends on the other then this is the dependent one.

relation_type : text that defines the nature of the association

description : text to provide further information about the nature of the association

NOTE This information need not be provided.

4.2.13.7 Product_type

A Product_type is a result of an action or process that is intended to achieve a particular objective or perform a particular function.

EXPRESS specification:

*)

```
  ENTITY Product_type;
    id                        : STRING;
    name                      : STRING;
    description               : STRING;
  END_ENTITY;
(*
```

Attribute definitions:

id : label that distinguishes a Product_type

name : label by which a Product_type is known

description : text to provide further information about a Product_type

4.2.13.8 Product_type_defined_in_external_reference

A Product_type_defined_in_external_reference is a type of Product_type that is defined in an external source.

EXPRESS specification:

*)

```
ENTITY Product_type_defined_in_external_reference
  SUBTYPE OF(Product_type);
  source_id                 : External_source_identification;
END_ENTITY;
(*
```

Attribute definitions:

source_id : identifier for the external source

4.2.13.9 Product_type_relationship

A Product_type_relationship is an association between two instances of Product_type.

EXPRESS specification:

*)

```
ENTITY Product_type_relationship;
  relating_type             : Product_type;
  related_type              : Product_type;
  relation_type             : STRING;
  description               : STRING;
END_ENTITY;
(*
```

Attribute definitions:

relating_type : one of the instances of Product_type that participates in the association

related_type : the other instance of Product_type participating in the association

NOTE If one of the elements of the relationship is dependent on the other then this attribute is the dependent one.

relation_type : text that defines the meaning of the relationship

description : text to provide further information about the relationship

4.2.13.10 Product_type_specified

A Product_type_specified is a type of Product_type that is defined in a formal specification.

EXPRESS specification:

*)

```
  ENTITY Product_type_specified
    SUBTYPE OF(Product_type);
    specification              : Specification;
  END_ENTITY;
(*
```

Attribute definitions:

specification : reference to the formal definition of the Product_type

4.2.13.11 Resource_item

A Resource_item is an object used in the performance of an activity.

EXAMPLE A tensile testing machine can be a Resource_item when the activity is to measure the strength of a solid product.

EXPRESS specification:

*)

```
  ENTITY Resource_item;
    resource_characterisation  :SET [1:?] OF Resource_item_characterisation;
  END_ENTITY;
(*
```

Attribute definitions:

resource_characterisation : characteristics of a Resource_item

4.2.13.12 Resource_item_assignment

A Resource_item_assignment is an association between a Resource_item and an Activity_as_planned or an Activity_as_realised.

EXPRESS specification:

*)

```
  ENTITY Resource_item_assignment;
    resource_role              :STRING;
    items                      :SET [1:?] OF Resource_item;
    assigned_to                :resourced_item;
  END_ENTITY;
(*
```

Attribute definitions:

resource_role : description of the purpose of the Resource_item in the activity

items : collection of Resource_item that is to be assigned

assigned_to : object to which the collection of Resource_item is associated

4.2.13.13 Resource_item_characterisation

A Resource_item_characterisation is information to describe a Resource_item.

EXPRESS specification:

*)

```
ENTITY Resource_item_characterisation;
  name                        :STRING;
  description                 :STRING;
  of_product                  :Individual_product;
END_ENTITY;
(*
```

Attribute definitions:

name : label by which a Resource_item is known and can be referred to

description : text that provides further information about a Resource_item

of_product : Individual_product that acts as a Resource_item

4.2.13.14 Resource_item_relationship

A Resource_item_relationship is an association between two instances of Resource_item.

EXPRESS specification:

*)

```
ENTITY Resource_item_relationship;
  relating                    :Resource_item;
  related                     :Resource_item;
  relation_type               :STRING;
  description                 :STRING;
END_ENTITY;
(*
```

Attribute definitions:

relating : one of the instances of Resource_item that participates in the relationship

related : the other instance of Resource_item that participates in the relationship

NOTE If one element of the relationship is dependent on the other then this attribute is the dependent one.

relation_type : text that defines the meaning of the relationship

description : text to provide further information about the relationship

4.2.14 Application objects for properties UoF

4.2.14.1 activity_select

An activity_select is a mechanism for proving an associaton between an instance of Activity_property and an instance of one of:

— Activity_type;

— Individual_activity.

EXPRESS specification:

*)

```
TYPE activity_select = SELECT
  (Activity_type,
  Individual_activity);
END_TYPE;
```

(*

4.2.14.2 product_assignment_select

A product_assignment_select is a mechanism that enables an instance of Engineering property to be selectively associated with one of:

— Product_type;

— Individual_product.

EXPRESS specification:

*)

```
TYPE product_assignment_select = SELECT
  (Product_type,
  Individual_product);
END_TYPE;
```

(*

4.2.14.3 Activity_property

An Activity_property is the characteristic of the behaviour, capabilities or performance of a process.

EXPRESS specification:

*)

```
  ENTITY Activity_property;
    of_activity                :activity_select;
    description                :STRING;
    name                       :STRING;
  END_ENTITY;
(*
```

Attribute definitions:

of_activity : reference to the activity of which the Activity_property is a characteristic

description : text that provides and explanation of the Activity_property

name : label by which the Activity_property can be recognised

4.2.14.4 Activity_property_defined_in_external_reference

An Activity_property_defined_in_external_reference is a type of Activity_property that is defined in an external source.

EXPRESS specification:

```
*)

  ENTITY Activity_property_defined_in_external_reference
    SUBTYPE OF(Activity_property);
    select_source              :property_source_select;
  END_ENTITY;
(*
```

Attribute definitions:

select_source : reference to the identification of a property defined in an external source

4.2.14.5 Activity_property_relationship

An Activity_property_relationship is an association between two instances of Activity_property.

EXPRESS specification:

```
*)

  ENTITY Activity_property_relationship;
    related                    :Activity_property;
    relating                   :Activity_property;
    description                :STRING;
    name                       :STRING;
  END_ENTITY;
(*
```

Attribute definitions:

related : one of the instances of Actvity_property that participates in the association

NOTE If one of the instances of Activity_property depends on the other then this is the dependent one.

relating : the other instance of Activity_property that participates in the association

description : text that provides and explanation of the association

name : label by which the association can be recognised

4.2.14.6 Activity_property_representation

An Activity_property_representation is a representation of an Activity_property.

NOTE An Activity_property can have many representations.

EXPRESS specification:

```
*)

  ENTITY Activity_property_representation;
    of_property               :Activity_property;
    name                      :STRING;
    description               :STRING;
  END_ENTITY;
(*
```

Attribute definitions:

of_property : reference to the characteristic that is being represented

name : label by which the representation can recognised

description : text provide an explanation of the representation

4.2.14.7 Engineering_property

An Engineering_property is a characteristic of a product that is either measured or assigned by a defined process.

EXPRESS specification:

```
*)

  ENTITY Engineering_property;
    of_product                :product_assignment_select;
  END_ENTITY;
(*
```

Attribute definitions:

of_product : product for which the Engineering_property is a characteristic

4.2.14.8 Engineering_property_defined_in_external_reference

An Engineering_property_defined_in_external_reference is a type of Engineering_property that is defined in an external source.

EXPRESS specification:

```
*)

  ENTITY Engineering_property_defined_in_external_reference
    SUBTYPE OF(Engineering_property);
    source_select              :property_source_select;
  END_ENTITY;
(*
```

Attribute definitions:

source_select : reference to the identification of a property defined in an external source

4.2.14.9 Engineering_property_representation

An Engineering_property_representation is a representation of an Engineering_property.

NOTE An Engineering_property can have several representations.

EXPRESS specification:

```
*)

  ENTITY Engineering_property_representation;
    property_definition        :Engineering_property;
    property_name              :STRING;
    value_elements             :SET [1:?] OF Measure_item;
  INVERSE
    dependent_on               :Property_environment_representation FOR
                                environment_elements;
  END_ENTITY;
(*
```

Attribute definitions:

property_definition : information that defines the Engineering_property

property_name : label by which the representation of the property is known

value_elements : values of the items in the Engineering_property_representation

4.2.14.10 Property_environment_relationship

A Property_environment_relatationship is an association between two instances of Property_environment_representation.

EXPRESS specification:

*)

```
  ENTITY Property_environment_relationship;
    related                    :Property_environment_representation;
    relating                   :Property_environment_representation;
    relation_type              :STRING;
    description                :OPTIONAL STRING;
  END_ENTITY;
(*
```

Attribute definitions:

related : one of the instances of Property_environment_representation that participates in the association

NOTE 1 If one of the instances of Property_environment_representation depends on the other then this is the dependent one.

relating : the other of the instances of Environment_representation that participates in the association

relation_type : text that defines the nature of the association

description : text to provide further information about the nature of the association

NOTE 2 The value of this attribute need not be specified

4.2.14.11 Property_environment_representation

A Property_environment_representation is the set of conditions in which the value of an Engineering_property is valid.

NOTE The conditions of validity are themselves instances of Engineering_property_representation.

EXPRESS specification:

*)

```
  ENTITY Property_environment_representation;
    environment_elements       :SET [1:?] OF
                                 Engineering_property_representation;
  END_ENTITY;
(*
```

Attribute definitions:

environment_elements : the set of Engineering_property_representation that constitutes the

conditions in which the value of an Engineering_property is valid

4.2.14.12 Property_representation_relationship

A Property_representation_relationship is an association between two instances of Engineering_property_representation.

EXPRESS specification:

*)

```
ENTITY Property_representation_relationship;
  related                    :Engineering_property_representation;
  relating                   :Engineering_property_representation;
  relation_type              :STRING;
  description                :OPTIONAL STRING;
END_ENTITY;
(*
```

Attribute definitions:

related : one of the instances of Material_property that participates in the association

NOTE 1 If one of the instances depends on the other then this is the dependent one.

relating : the other instance of Engineering_property that participates in the association

relation_type : text that defines the nature of the association

description : text that provides further information about the nature of the association

NOTE 2 This attribute need not be specified

4.2.14.13 Resource_property

A Resource_property is a characteristic of something needed for an activity.

EXPRESS specification:

*)

```
ENTITY Resource_property;
  name                       :STRING;
  description                :STRING;
  of_resource                :Resource_item;
END_ENTITY;
(*
```

Attribute definitions:

name : lable by which the Resource_property can be recognised

description : text that provides an explanation of the Resource_property

of_resource : reference to the Resource_item of which Resource_property is a characteristic

4.2.14.14 Resource_property_defined_in_external_reference

A Resource_property_defined_in_external_reference is a type of Resource_property that is defined in an external source.

EXPRESS specification:

*)

```
ENTITY Resource_property_defined_in_external_reference
  SUBTYPE OF(Resource_property);
  source_select              :property_source_select;
END_ENTITY;
```

(*

Attribute definitions:

source_select : reference to the identification of a property defined in an external source

4.2.14.15 Resource_property_representation

A Resource_property_representation is a representation of a Resource_property.

NOTE A Resource_property can have several representations.

EXPRESS specification:

*)

```
ENTITY Resource_property_representation;
  description               :STRING;
  name                      :STRING;
  of_property               :Resource_property;
END_ENTITY;
```

(*

Attribute definitions:

description : text that is an explanation of the representation

name : label by which the representation can be recognised

of_property : reference to the characteristic that is being represented

4.2.15 Application objects for the requirement UoF

This subclause specifies the data types and application objects associated with the requirement UoF.

4.2.15.1 characterization_of_requirement

A characterization_of_requirement is an extensible select data type that provides the means to reference items with which an instance of Requirement may be selectively associated. An instance of Requirement can be selectively associated with one of:

— Approval;

— Condition;

— Individual_activity;

— Individual_product;

— Organisation;

— Person_in_organisation;

— Qualification;

— Substance.

EXPRESS specification:

*)

```
  TYPE characterization_of_requirement = EXTENSIBLE SELECT
    (Condition,
    person_organisation,
    Approval,
    Qualification,
    Individual_product,
    Individual_activity,
    Substance);
  END_TYPE;
(*
```

4.2.15.2 requirement_assignment_item

A requirement_assignment_item is a select data type that provides a list of alternative items with which a Requirement_assignment can be selectively associated. A Requirement_assignment can be selectively associated with one of:

— Individual_activity;

— Individual_product;

— Organisation;

— Person_in_organisation;

— Substance.

EXPRESS specification:

*)

```
  TYPE requirement_assignment_item = SELECT
    (Individual_activity,
      Individual_product,
      person_organisation,
      Substance);
  END_TYPE;
(*
```

4.2.15.3 requirement_source_item

A requirement_source_item is a select data type that provides a list of alternative items with which a Requirement_source can be selectively associated. A Requirement_source can be selectively associated with one of:

— Organisation;

— Person_organisation.

EXPRESS specification:

*)

```
  TYPE requirement_source_item = SELECT
    (Organisation,
      Person_organisation);
  END_TYPE;
(*
```

4.2.15.4 Required_resource

A Required_resource is an equipment or a substance needed to achieve a specified objective.

EXPRESS specification:

*)

```
  ENTITY Required_resource
    ABSTRACT SUPERTYPE OF (ONEOF(Required_resource_by_resource_item,
                      Required_resource_by_specification));
    name                      :STRING;
    description               :STRING;
    resource_quantity         :Measure_item;
  END_ENTITY;
(*
```

Attribute definitions:

name : label that identifies the Required_resource

description : further information about the Required_resource

resource_quantity : amount of resource that is needed

4.2.15.5 Required_resource_by_resource_item

A Required_resource_by_resource_item is a type of Required_resource that can be defined within the application context.

EXPRESS specification:

*)

```
ENTITY Required_resource_by_resource_item
  SUBTYPE OF(Required_resource);
  required_item              :SET [1:?] OF Resource_item;
END_ENTITY;
```

(*

Attribute definitions:

required_item : set of objects that define the Required_resource

4.2.15.6 Required_resource_by_specification

A Required_resource_by_specification is a type of Required_resource that is defined in a formal definition.

EXPRESS specification:

*)

```
ENTITY Required_resource_by_specification
  SUBTYPE OF(Required_resource);
  resource_specification     :Specification;
END_ENTITY;
```

(*

Attribute definitions:

resource_specification : reference to the formal definition of the required resource

4.2.15.7 Required_resource_relationship

A Required_resource_relationship is an association between two instances of Required_resource.

EXPRESS specification:

*)

```
  ENTITY Required_resource_relationship;
    related                    :Required_resource;
    relating                   :Required_resource;
    relation_type              :STRING;
    description                :STRING;
  END_ENTITY;
(*
```

Attribute definitions:

related : the other instance of Required_resource that particpates in the association

NOTE If one element of the relationship is dependent on the other then this attribute is the dependent one.

relating : one of the instances of Required_resource that participates in the association

relation_type : definition of the nature of the association

description : further information on the nature of the association

4.2.15.8 Requirement

A Requirement is something specified as compulsory.

EXPRESS specification:

*)

```
  ENTITY Requirement;
    name                       :STRING;
    required                   :characterization_of_requirement;
  END_ENTITY;
(*
```

Attribute definitions:

name : label by which a Requirement is known and can be identified

required : identification of what is stipulated as compulsory

4.2.15.9 Requirement_assignment

A Requirement_assignment is a mechanism for relating an instance of a Requirement_view_definition to the data types representing the items that are affected by a Requirement.

EXPRESS specification:

*)

```
  ENTITY Requirement_assignment;
    assigned_requirement       :Requirement_view_definition;
    assigned_to                :requirement_assignment_item;
    id                         :STRING;
    description                :STRING;
    requirement_role           :STRING;
  END_ENTITY;
(*
```

Attribute definitions:

assigned_requirement : instance of Requirement_view_definition that is assigned to an item

assigned_to : identification of the item to which the instance of Requirement is assigned

id : label to distinguish an instance of Requirement_assignment

description : text to provide further information about an instance of Requirement_assignment

requirement_role : text to describe the purpose for which the assignment is made

4.2.15.10 Requirement_source

A Requirement_source is a mechanism for connecting a requirement to the data types representing the origin of a Requirement.

EXPRESS specification:

```
*)

  ENTITY Requirement_source;
    sourced_requirement        :Requirement_view_definition;
    source                     :requirement_source_item;
    id                         :STRING;
    description                :STRING;
  END_ENTITY;
(*
```

Attribute definitions:

sourced_requirement : identification of the requirement that is to be related to its source

source : item that is the source of the requirement

id : label that distinguishes a particular instance of Requirement_source

description : text to provide further information about an instance of Requirement_source

4.2.15.11 Requirement_version

A Requirement_version is the identification of a variant of a Requirement.

EXPRESS specification:

*)

```
  ENTITY Requirement_version;
    of_requirement             :Requirement;
    description                :STRING;
    version_id                 :STRING;
  UNIQUE
    UR1      :of_requirement,
              version_id;
  END_ENTITY;
(*
```

Attribute definitions:

of_requirement : stipulation of which the Requirement_version is a variant

description : text that provides further information about the Requirement_version

version_id : label that identifies the Requirement version

4.2.15.12 Requirement_version_relationship

A Requirement_version_relationship is an association between two instances of a Requirement_version.

EXPRESS specification:

*)

```
  ENTITY Requirement_version_relationship;
    relating_version           :Requirement_version;
    related_version            :Requirement_version;
    relation_type              :STRING;
    description                :STRING;
  END_ENTITY;
(*
```

Attribute definitions:

relating_version : one of the instances of Requirment_version that participates in the association

related_version : the other instance of Requirement-version that particpates in the association

NOTE If one element of the relationship is dependent on the other then this attribute is the dependent one.

relation_type : definition of the association

description : text to provide further information about the nature of the association

4.2.15.13 Requirement_view_definition

A Requirement_view_definition provides a view of a Requirement_version that is relevant for an application domain and collects requirement data for a specific engineering purpose.

NOTE If a requirement has been satisfied then the value of the attribute 'satisfied' is set to 'TRUE' otherwise it is set to 'FALSE'.

EXAMPLE An engineer can have the responsibility for specifying all requirements for a design that relate to recycling.

EXPRESS specification:

*)

```
ENTITY Requirement_view_definition;
  satisfied                 :BOOLEAN;
  defined_version           :Requirement_version;
END_ENTITY;
(*
```

Attribute definitions:

satisfied : indicator of whether the Requirement_view_definition is satisfied

defined_version : the Requirement_version that is defined by the Requirement_view_definition

4.2.16 Application objects for the state UoF

This subclause defines the data types and application objects associated with the state UoF.

4.2.16.1 state_type_of_item

A state_type_of_item is a select data type that provides a list of alternative items with which an instance of State_type_assignment may be selectively associated. An instance of State_type_assignment can be selectively associated with one of:

— Individual_activity;

— Individual_product.

EXPRESS specification:

*)

```
TYPE state_type_of_item = SELECT
  (Individual_activity,
    Individual_product);
END_TYPE;
```

(*

4.2.16.2 Derived_state_type

A Derived_state_type is a type of State_type that is the intersection of at least two instances of State_type.

EXPRESS specification:

*)

```
  ENTITY Derived_state_type
    SUBTYPE OF(State_type);
    intersection               :SET [2:?] OF State_type;
  END_ENTITY;
```

(*

Attribute definitions:

intersection : identification of the classes in the intersection

4.2.16.3 Process_defined_state_type

A Process_defined_state_type is a type of State_type that is defined in terms of a process and its properties.

EXPRESS specification:

*)

```
  ENTITY Process_defined_state_type
    SUBTYPE OF(State_type);
    defining_activity          :Individual_activity;
  END_ENTITY;
```

(*

Attribute definitions:

defining_activity : reference to the Individual_activity that defines the Process_defined_state

4.2.16.4 Product_defined_state_type

A Product_defined_state_type is a type of State_type that is defined in terms of a Product_type.

EXPRESS specification:

*)

```
  ENTITY Product_defined_state_type
    SUBTYPE OF(State_type);
    defining_product_type      :Product_type;
  END_ENTITY;
```

(*

Attribute definitions:

defining_product_type : Product_type that defines the Product_type_defined_state_type

4.2.16.5 Property_defined_state_type

A Property_defined_state_type is a type of State_type that is defined in terms of a Property_type.

EXPRESS specification:

*)

```
ENTITY Property_defined_state_type
  SUBTYPE OF(State_type);
  defining_quantity          :Quantity;
  defining_property          :Engineering_property;
END_ENTITY;
```

(*

Attribute definitions:

defining_quantity : the Quantity for the properties that define the state

defining_property : property that defines the Property_defined_state

4.2.16.6 Quantity

A Quanitity is an amount of something that is specified as a multiple of a unit amount.

EXPRESS specification:

*)

```
ENTITY Quantity
  SUBTYPE OF(State_type);
  magnitude                  :Measure_item;
END_ENTITY;
```

(*

Attribute definitions:

magnitude : the value of the amount

4.2.16.7 State_type

A State_type is a condition of something at a particular time.

EXPRESS specification:

*)

```
  ENTITY State_type
    SUPERTYPE OF (ONEOF(Derived_state_type, Product_type_defined_state_type)
                      ANDOR Property_defined_state_type ANDOR
                      Process_defined_state_type ANDOR Quantity);
    description                :STRING;
    name                       :STRING;
  END_ENTITY;
(*
```

Attribute definitions:

description : text to provide further information about the State_type

name : label by which a State_type is known and can be referred to

4.2.16.8 State_type_assignment

A State_type_assignment is a mechanism to assign an instance of State_type to an item.

EXPRESS specification:

```
*)

  ENTITY State_type_assignment;
    assigned_state_type        :State_type;
    item_state                 :state_type_of_item;
  END_ENTITY;
(*
```

Attribute definitions:

assigned_state_type : instance of State_type that is to be assigned

item_state : item to which the State_type is assigned

.

4.2.16.9 State_type_relationship

A State_type_relationship is an association between instances of State_type.

EXPRESS specification:

```
*)

  ENTITY State_type_relationship;
    relating_state_type        :State_type;
    related_state_type         :State_type;
    description                :STRING;
    relation_type              :STRING;
  END_ENTITY;
(*
```

Attribute definitions:

relating_state_type : one of the instances of State_type that is part of the association

related_state_type : the other instance of State type that is part of the association

NOTE If one element of the relationship is dependent on the other then this attribute is the dependent one.

description : text to provide further information about the association

relation_type : definition of the nature of the association between instances of State_type

4.2.17 Application objects for the substance UoF

This subclause defines the data types and application objects associated with the substance UoF.

4.2.17.1 assigned_product

An assigned_product is a select data type that provides a list of alternative items with which an instance of Substance_assignment can be selectively associated. An instance of Substance_assignment can be selectively associated with one of:

— Individual_product;

— Individual_product_feature;

— Product_type.

EXPRESS specification:

```
*)

  TYPE assigned_product = SELECT
    (Individual_product_feature,
    Individual_product,
    Product_type);
  END_TYPE;
(*
```

4.2.17.2 Structure_element_characterisation

A Structure_element_characterisation enables the description of the properties of a Substance_structure_element.

EXPRESS specification:

```
*)

  ENTITY Structure_element_characterisation;
    description                :STRING;
    characteristic_orientation :Geometric_model_element;
    characteristic_property    :Engineering_property_representation;
```

```
    characterises              :Substance_structure_element;
  END_ENTITY;
(*
```

Attribute definitions:

description : text to provide further information about the Structure_element_characterisation

characteristic_orientation : alignment of the Substance_structure_element in a geometric coordinate space

characteristic_property : property of the Substance_structure_element

characterises : the structural element whose properties are defined

4.2.17.3 Substance

A Substance is matter that possesses definite properties or characteristics.

EXPRESS specification:

```
*)

  ENTITY Substance;
    description               :STRING;
    id                        :STRING;
    substance_name            :STRING;
  END_ENTITY;
(*
```

Attribute definitions:

description : text to provide information about the Substance

id : label by which the Substance can be identified

substance_name : label by which a substance is known and can be referred to

4.2.17.4 Substance_assignment

A Substance_assignment is a mechanisam for providing the association between a Substance and a product that contains the substance.

EXPRESS specification:

```
*)

  ENTITY Substance_assignment;
    substance                 :Substance;
    of_product                :assigned_product;
    substance_role            :STRING;
```

```
  END_ENTITY;
(*
```

Attribute definitions:

substance : Substance that is to be associated

of_product : selection of the product to which the Substance is assigned

substance_role : description of the purpose for the association of a Substance

4.2.17.5 Substance_composition_element

A Substance_composition_element is an element or component in the constitution of a Substance.

EXPRESS specification:

```
*)

  ENTITY Substance_composition_element;
    name                        :STRING;
    description                 :Specification;
    composes                    :Substance;
    composition_specified       :Specification
    element_amount              :Measure_item;
  END_ENTITY;
(*
```

Attribute definitions:

name : label by which the Substance_composition_element is known and can be referred to

description : text to provide further information about the Substance_composition_element

composes : reference to the Substance of which the Substance_composition_element is a part

composition_specified : formal definition of the composition of the Substance

element_ amount : quantity of the Substance_composition_element

4.2.17.6 Substance_defined_in_external_reference

A Substance_defined_in_external_reference is a type of Substance that is defined in an external source.

EXPRESS specification:

```
*)

  ENTITY Substance_defined_in_external_reference
```

```
    SUBTYPE OF(Substance);
    source_select              :classification_source_select;
  END_ENTITY;
(*
```

Attribute definitions:

source_select : identification of the external reference

4.2.17.7 Substance_structure_element

A Substance_structure_element is the definition of a physical object that is part of the structure of the Substance.

EXPRESS specification:

*)

```
  ENTITY Substance_structure_element;
    name                       :STRING;
    description                :STRING;
    structures                 :Substance;
    structure_specified        :Specification;
  END_ENTITY;
(*
```

Attribute definitions:

name : label by which the Structure_element is known and can be referred to

description : text to provide further information about the Structure_element

structures : reference to the Substance of which the Substance_structure_element is a part

structure_specified : formal definition of the structure of the Substance

4.2.18 Application objects for the tolerance datum UoF

This subclause defined the data types and application objects associated with the tolerance datum UoF.

4.2.18.1 datum_representation_select

A datum_representation_select is a select data type that enables an instance of Single_datum to be selectively associated with one of:

— Reference_shape;

— Datum_target_set;

— Reference_shape_element.

EXPRESS specification:

*)

```
  TYPE datum_representation_select = SELECT
    (Reference_shape,
    Datum_target_set,
    Reference_shape_element);
  END_TYPE;
(*
```

4.2.18.2 Compound_datum

A Compound_datum is a type of Datum that specifies a set of two or more Single_datum of equal importance which are used to establish a single datum plane or axis.

NOTE A compound datum is a called a multiple datum feature or a common datum in some standards for tolerancing.

EXPRESS specification:

*)

```
  ENTITY Compound_datum
    SUBTYPE OF(Datum);
    made_of                    :SET [2:?] OF Single_datum;
  END_ENTITY;
(*
```

Attribute definitions:

made_of : set of Single _datum objects that form the Compound_datum

4.2.18.3 Datum

A Datum is a theoretically exact point, straight line, or plane used as a reference for locating and orienting tolerance zones.

EXPRESS specification:

*)

```
  ENTITY Datum
    ABSTRACT SUPERTYPE OF (ONEOF(Single_datum, Compound_datum));
    precedence                 :INTEGER;
  END_ENTITY;
(*
```

Attribute definitions:

precedence : identification of the order of importance of a Datum

4.2.18.4 Datum_target

A Datum_target is the identification of a portion of a feature that is used in the construction of a Datum.

EXPRESS specification:

*)

```
ENTITY Datum_target
  SUPERTYPE OF (ONEOF(Target_area, Placed_target));
  target_id                 :STRING;
END_ENTITY;
```

(*

Attribute definitions:

target_id : identifier for the Datum_target in a Datum_target_set

4.2.18.5 Datum_target_set

A Datum_target_set is a set of Datum_target objects.

NOTE Datums are established from a set of Datum_target when the use of an entire feature would introduce excessive variations or lack of repeatability in measurements.

EXPRESS specification:

*)

```
ENTITY Datum_target_set;
  datum_target              :SET [1:?] OF Datum_target;
  rule_description          :STRING;
END_ENTITY;
```

(*

Attribute definitions:

datum_target : set of instances of Datum_target that form the Datum_target_set

rule_description : type of datum that is formed by the Datum_target_set

NOTE There is exactly one object that defines the rule-description for a Datum_target_set.

EXAMPLE 'V-block' indicates that two Datum_target objects on a cylindrical element are required to form the areas of contact in a v-shaped feature

4.2.18.6 Placed_target

A Placed_target is a type of Datum_target that is represented implicitly by parameters.

EXPRESS specification:

```
*)

  ENTITY Placed_target
    ABSTRACT SUPERTYPE OF (ONEOF(Target_point, Target_straight_line,
                      Target_rectangle, Target_circle))
    SUBTYPE OF(Datum_target);
    defined_in                 :Geometric_coordinate_space;
    parameter_reference        :Axis_placement;
  END_ENTITY;
(*
```

Attribute definitions:

defined_in : Cartesian_coordinate_space in which the Axis_placement that serves as a parameter_reference is defined

parameter_reference : the Axis_placement to which the parameters of the Placed_target refers

4.2.18.7 Reference_shape

A Reference_shape is a datum that is a defined outline.

EXPRESS specification:

```
*)

  ENTITY Reference_shape;
    referenced_shape           :Shape;
  END_ENTITY;
(*
```

Attribute definitions:

referenced_shape : outline that provides the datum

4.2.18.8 Reference_shape_element

A Reference_shape_element is a datum that is part of a shape outline.

EXPRESS specification:

```
*)

  ENTITY Reference_shape_element;
    referenced_element         :Shape_element;
  END_ENTITY;
(*
```

Attribute definitions:

referenced_element : reference to the element of the outline that provides the datum

4.2.18.9 Single_datum

A Single_datum is a type of Datum derived from a single datum feature that specifies a point, line, axis, or plane used as a reference for locating and orienting tolerance zones.

EXPRESS specification:

*)

```
ENTITY Single_datum
  SUBTYPE OF(Datum);
  modification              :Tolerance_condition;
  datum_name                :STRING;
  defined_by                :datum_representation_select;
END_ENTITY;
```

(*

Attribute definitions:

datum_name : the word or group of words by which the Single_datum is known

defined_by : Datum_target_set, or Reference_shape, or Reference_shape_element that represents the Single_datum

4.2.18.10 Target_area

A Target_area is a type of Datum_target that specifies an area bounded by a closed Shape_element.

EXPRESS specification:

*)

```
ENTITY Target_area
  SUBTYPE OF(Datum_target);
END_ENTITY;
```

(*

4.2.18.11 Target_circle

A Target_circle is a type of Placed_target that specifies a Datum_target that is defined by an implicit circle whose centre is specified by the placing Axis_placement_3D and whose orientation is the x-y plane of the Axis_placement_3D.

EXPRESS specification:

*)

```
ENTITY Target_circle
  SUBTYPE OF(Placed_target);
  target_diameter           :Measure_item;
END_ENTITY;
```

(*

Attribute definitions:

target_diameter : diameter of the Target_circle

4.2.18.12 Target_point

A Target_point is a type of Placed_target that specifies a Datum_target defined by a single implicit point and whose position is defined by the placing Axis_placement.

EXPRESS specification:

*)

```
ENTITY Target_point
  SUBTYPE OF(Placed_target);
END_ENTITY;
```

(*

4.2.18.13 Target_rectangle

A Target_rectangle is a type of Placed_target that specifies a Datum_target defined by an implicit rectangle whose centre is specified by the placing Axis_placement.

NOTE The orientation of the Target_rectangle is in the x-y plane with half the length of the rectangle in the positive direction and half the length of the rectangle in the negative direction along the x-axis and with half the width of the rectangle in the postive direction and half the width of the rectangle in the negative direction along the y-axis of the Axis_placement.

EXPRESS specification:

*)

```
ENTITY Target_rectangle
  SUBTYPE OF(Placed_target);
  target_length              :Measure_item;
  target_width               :Measure_item;
END_ENTITY;
```

(*

Attribute definitions:

target_length : length of the Target_rectangle

target_width : width of the Target_rectangle

4.2.18.14 Target_straight_line

A Target_straight_line is a type of Place_target that specifies a Datum_target that is defined by an implicit straight line where the first end point of the line is specified by the placing Axis_placement and the second end point is located on the z-axis of the Axis_placement at the specified distance.

EXPRESS specification:

*)

```
ENTITY Target_straight_line
  SUBTYPE OF(Placed_target);
END_ENTITY;
```

(*

4.2.18.15 Tolerance_condition

A Tolerance_condition is the specification of the material condition of a feature of size, which is its actual size with respect to its size tolerance

NOTE 1 The set of candidate tolerance zones for tolerances with datum_reference depends on the material condition of any referenced datum features of size.

NOTE 2 The set of candidate tolerance zones of a Position_tolerance depends on the material condition of the toleranced feature of size.

EXPRESS specification:

*)

```
ENTITY Tolerance_condition;
  condition                 :STRING;
END_ENTITY;
```

(*

Attribute definitions:

condition : description of the kind of Tolerance_condition

EXAMPLE 1 Maximum material condition(MMC)
EXAMPLE 2 Least material condition (LMC)

5 Application interpreted model

5.1 Mapping specification

This clause contains the mapping specification that shows how each UoF and application object of this part of ISO 10303 (see clause 4) maps to one or more AIM constructs (see Annex A). Each mapping specifies up to five elements.

Application element: The mapping for each application element is specified in a separate sub-clause below. Application element names are given in title case. Attribute names and assertions are listed after the the application object to which they belong and are given in lower case.

AIM element: The name ofone or more AIM entity data types (see annexA), the term "IDENTICAL MAPPING", or the term "PATH". AIM entity data type names are given in lower case. Attributes of AIM entity data types are referred to as <entity name>.<attrbute name>. The mapping of an application element may involve more than one AIM element. Each of these AIM elements is presented on a separate line in the mapping specification. The term "IDENTICAL MAPPING" indicates that both applicaton objects involved in an application assertion map to the same instance of an AIM entity data type. The term "PATH" indicates that

the application assertion maps to a collection of related AIM entity instances specified by the entire reference path.

Source: For those AIM elements that are interpreted from any common resource, this is the ISO standard number and part number in which the resource is defined. For those AIM element that are created for the purpose of this part of IOS 10303, this is "ISO 10303-"followed by the number of this part.

Rules: One or more global rules may be specified that apply to the population of the AIM entity data types specified as the AIM element or in the reference path. For rules that are derived from relationships between application objects, the same rule is referred to by the mapping entries of all the involved AIM elements. A reference to a global rule may be accompanied by a reference to the subclause in which the rule is defined.

Reference path: To describe fully the mapping of an application object, it may be necessary to specify a reference path involving several related AIM elements. Each line in the reference path documents the role of an AIM element relative to the AIM element in the line following it. Two or more such related AIM elements define the interpretation of the integrated resources that satisfies the requirement specified by the application object. For each AIM element that has been created for use within this part of ISO 10303, a reference path to its supertype from an integrated resource is specified. For the expression of reference paths and of the constraints between AIM elements, the following notational conventions apply:

[] enclosed section constrains multiple AIM elements or sections of the reference path are required to satisfy an information requirement;

() enclosed section constrains multiple AIM elements or sections of the reference path are identified as alternatives within the mapping to satisfy an information requirement;

{} enclosed section constrains the reference path to satisfy an information requirement;

<> enclosed section constrains at one or more required reference path;

|| enclosed section constrains the supertype entity;

-> the attribute, whose name precedes the -> symbol, references the entity or select type whose name follows the -> symbol;

<- the entity or select type, whose name precedes the <- symbol, is referenced by the entity attribute whose name follows the <- symbol;

[i] the attribute, whose name precedes the [i] symbol, is an aggregate; any element of that aggregate is referred to;

[n] the attribute, whose name precedes the [n] symbol, is an ordered aggregate; member n of that aggregate is referred to;

=> the entity, whose name precedes the => symbol, is a supertype of the entity whose name follows the => symbol;

<= the entity, whose name precedes the <= symbol, is a subtype of the entity whose name follows the <= symbol;

= the string, select, or enumeration type is constrained to a choice or value;

\ the reference path expression continues on the next line;

* one or more instances of the relationship entity data type may be assembled in a relationship tree structure. The path between the relationship entity and the related entities, is enclosed with braces;

-- the text following is a comment or introduces a clause reference;

*> the select or enumeration type, whose name precedes the *> symbol, is extended into the select or enumeration type whose name follows the *> symbol;

<* the select or enumeration type, whose name precedes the <* symbol, is an extension of the select or enumeration type whose name follows the <* symbol.

5.1.1 Activity UoF

5.1.1.1 Activity_as_planned

AIM element: executed_action
Source: ISO 10303-41
Reference path: (action =>
executed_action<-
action_status.assigned_action
{action_status.status = 'planned'})'

5.1.1.1.1 Activity_as_planned to Activity_type (as plan)

AIM element: PATH
Reference path: (executed_action<=
action
action.chosen_method ->
action_method)

5.1.1.2 Activity_as_realised

AIM element: executed_action
Source: ISO 10303-41
Reference path: (action =>
executed_action<-
action_status.assigned_action
{action_status.status = 'realised'})

5.1.1.2.1 Activity_as_realised to Activity_type (as of_type)

AIM element: PATH
Reference path: (executed_action<=
action
action.chosen_method ->
action_method)

5.1.1.2.2 Activity_as_realised to Activity_as_planned (as realisation_of)

AIM element: PATH
Reference path: (executed_action<=
action<-
action_relationship.related_action
{action_relationship.description= 'realisation of'})

5.1.1.3 Activity_relationship

AIM element: action_relationship
Source: ISO 10303-41

5.1.1.3.1 description

AIM element: action_relationship.description
Source: ISO 10303-41

5.1.1.3.2 Activity_relationship to Individual_activity (as related_activity)

AIM element: action_relationship.related_action
Source: ISO 10303-41

5.1.1.3.3 Activity_relationship to Individual_activity (as relating_activity)

AIM element: action_relationship.relating_activity
Source: ISO 10303-41

5.1.1.3.4 relation_type

AIM element: action_relationship.name
Source: ISO 10303-41

5.1.1.4 Activity_type

AIM element: action_method
Source: ISO 10303-41

5.1.1.4.1 description

AIM element: action_method.description
Source: ISO 10303-41

5.1.1.4.2 name

AIM element: action_method.name
Source: ISO 10303-41

5.1.1.5 Activity_type_defined_in_external_reference

AIM element: externally_defined_class
Source: ISO 10303-235

5.1.1.5.1 class_select

AIM element: PATH
Reference path: (externally_defined_class <=
externally_defined_item ->
externally_defined_item.item_id)

5.1.1.6 Activity_type_relationship

AIM element: action_method_relationship
Source: ISO 10303-41

5.1.1.6.1 description

AIM element: action_method_relationship.description
Source: ISO 10303-41

5.1.1.6.2 relation_type

AIM element: action_method_relationship.name
Source: ISO 10303-41

5.1.1.6.3 Activity_type_relationship to Activity_type (as related_type)

AIM element: action_method_relationship.related_method
Source: ISO 10303-41

5.1.1.6.4 Activity_type_relationship to Activity_type (as relating_type)

AIM element: action_method_relationship.relating_method
Source: ISO 10303-41

5.1.1.7 Activity_type_specified

AIM element: action_method_with_specification_reference
Source: ISO 10303-49

5.1.1.7.1 Activity_type_specified to Specification (as activity_specification)

AIM element: PATH
Reference path: action_method_with_specification_reference
action_method_with_specification_reference.specification ->
docment)
Source: ISO 10303-49

5.1.1.8 Individual_activity

AIM element: action
Source ISO 10303-41

5.1.1.8.1 activity_id

AIM element: action.id
Source: ISO 10303-41

5.1.1.8.2 activity_name

AIM element: action.name
Source: ISO 10303-41

5.1.1.8.3 activity_role

AIM element: PATH
Source: ISO 10303-41

Reference path: (action <-
action_assignment.assigned_action
action_assignment
action_assignment.role)

5.1.1.8.4 description

AIM element: action.description
Source: ISO 10303-41

5.1.1.8.5 responsible

1. Where the responsible is an organisation

AIM element: person_organization_select ->
organization
Source: ISO 10303-41

2. Where the responsible is a person

AIM element: person_organization_select->
person
Source: ISO 10303-41

3. Where the responsible is a person in an organisation

AIM element: person_organization_select
person_and_organization
Source: ISO 10303-41

5.1.1.9 Individual_activity_assignment

AIM element: action_assignment
Source: ISO 10303-41

5.1.1.9.1 assigned_role

AIM element: action_assignment.role
Source: ISO 10303-41

5.1.1.9.2 Individual_activity_assignment to Individual_activity (as assigned_individual_activity)

AIM element: PATH
Reference path: (action_assignment
action_assignment.assigned_action->
action)
Source: ISO 10303-41

5.1.1.9.3 Individual_activity_assignment to Approval (as assignment_item)

AIM element: PATH
Source: ISO 10303-235
Reference path: (action_assignment =>
applied_action_assignment
applied_action_assignment.item ->
action_item ->
approval)

5.1.1.9.4 Individual_activity_assignment to Certificate (as assignment_item)

AIM element: PATH
Source: ISO 10303-235
Reference path: (action_assignment =>
applied_action_assignment
applied_action_assignment.item ->
action_item ->
certification)

5.1.1.9.5 Individual_activity_assignment to Document_version (as assignment_item)

AIM element: PATH
Source: ISO 10303-235
Reference path: (action_assignment =>
applied_action_assignment
applied_action_assignment.item ->
action_item ->
document)

5.1.1.9.6 Individual_activity_assignment to Engineering_property_representation (as assignment_item)

AIM element: PATH
Source: ISO 10303-235
Reference path: (action_assignment =>
applied_action_assignment
applied_action_assignment.item ->
action_item ->
material_property_representation)

5.1.1.9.7 Individual_assignment to Individual_product (as assignment_item)

AIM element: PATH
Source: ISO 10303-235
Reference path: (action_assignment =>
applied_action_assignment
applied_action_assignment.action_item ->
action_item ->
product_definition)

5.1.2 Adminstration UoF

5.1.2.1 Calendar_date

AIM element: calendar_date
Source: ISO 10303-41

5.1.2.2 Contract

AIM element: contract
Source: ISO 10303-41

5.1.2.2.1 contract_id

AIM element: contract.name
Source: ISO 10303-41

5.1.2.2.2 Contract to Calendar_date (as effective_date)

AIM element: PATH
Reference path: (date_item = contract
date_item <- applied_date_assignment.item
applied_date_assignment <=
date_assignment
{date_assignment
date_assignment.role ->
date_role
date_role.name = 'effective_date'}
date_assignment.assigned_date ->
date =>
calendar_date)

5.1.2.2.3 Contract to person_organisation (as contractee)

AIM element: PATH
Reference path: (contract<-
contract_assignment
{contract_assignment.role ->object_role
object_role.name = 'contractee'})

5.1.2.2.4 Contract to person_organisation (as contractor)

AIM element: PATH
Reference path: (contract<-
contract_assignment
{contract_assignment.role->object_role
object_role.name='contractor'})

5.1.2.2.5 Contract to Project (as for_project)

AIM element: PATH
Reference path: (contract_item = organizational_project
contract_item <- applied_contract_assignment.item
applied_contract_assignment <=
contract_assignment
{contract_assignment
contract_asignment.role = 'for_project'})

5.1.2.3 Contract_assignment

AIM element: applied_contract_assignment
Source: ISO 10303-235
Reference path: (contract_assignment =>
applied_contract_assignment)

5.1.2.3.1 items

AIM element: PATH
Reference path: (contract_assignment=>
applied_contract_assignment ->
applied_contract_assignment.items ->
(organizational_project,
property_definition,
action,
product))

5.1.2.3.2 Contract_assignment to Contract (as assigned_contract)

AIM element: PATH
Reference path: (applied_contract_assignment <=
contract_assignment ->
contract_assignment.assigned_contract)

5.1.2.4 Date_time

AIM element: date_and_time
Source: ISO 10303-41

5.1.2.5 Event

AIM element: event_occurrence
Source: ISO 10303-41

5.1.2.5.1 event_id

AIM element: event_occurrence.id
Source: ISO 10303-41

5.1.2.5.2 event_name

AIM element: event_occurrence.name
Source: ISO 10303-41

5.1.2.5.3 event_description

AIM element: event_occurrence.description
Source: ISO 10303-41

5.1.2.5.4 Event to date_or_date_time (as actual_date)

AIM element: PATH
Reference path: 1. (date_item = event_occurrence
date_item <-
applied_date_assignment.item
applied_date_assignment <=
date_assignment
{date_assignment.role->
date_role
date_role.name = 'actual_date'}
date_assignment.assigned_date ->
date =>
calendar_date)
2. (date_and_time_item = event_occurrence
date_and_time_item <-
applied_date_and_time_assignment.item
applied_date_and_time_assignment <=
date_and_time_assignment
{date_and_time_assignment
date_and_time_assignment.role ->
date_and_time_role
date_and_time_role_name = 'actual_date'}
date_and_time_assignment.assigned_date_and_time ->
date_and_time)

5.1.2.5.5 Event to Location (as event_location)

AIM element: PATH
Reference path: (location_item = event_occurrence
location_item <-
applied_location_assignment.item
applied_location_assignment <=
location_assignment
location_assignment.assigned_location ->
location)

5.1.2.6 Event_relationship

AIM element: event_occurrence_relationship
Source: ISO 10303-41

5.1.2.6.1 description

AIM element: event_occurrence_relationship.description
Source: ISO 10303-41

5.1.2.6.2 relation_type

AIM element: event_occurrence_relationship.name
Source: ISO 10303-41

5.1.2.6.3 Event_relationship to Event (as related_event)

AIM element: event_occurrence_relationship .related_event
Source: ISO 10303-41

5.1.2.6.4 Event_relationship to Event (as relating_event)

AIM element: event_occurrence_relationship.relating_event
Source: ISO 10303-41

5.1.2.7 hour_in_day

AIM element: hour_in_day
Source: ISO 10303-41

5.1.2.8 Local_time

AIM element: local_time
Source: ISO 10303-41

5.1.2.8.1 Local_time to hour_in_day (as hour)

AIM element: PATH
Reference path: (local_time
local_time.hour_component ->
hour_in_day)

5.1.2.8.2 Local_time to minute_in_hour (as minute)

AIM element: PATH
Reference path: local_time
local_time.minute_component ->
minute_in_hour

5.1.2.8.3 Local_time to second_in_minute (as second)

AIM element: PATH

Reference path: (local_time
local_time.second_component ->
second_in_minute)

5.1.2.8.4 Local_time to Time_offset (as zone)

AIM element: PATH
Reference path: (local_time
local_time.zone ->
coordinated_universal_time_offset)

5.1.2.9 minute_in_hour

AIM element: minute_in_hour
Source: ISO 10303-41

5.1.2.10 Project

AIM element: organizational_project
Source: ISO 10303-41

5.1.2.10.1 project_name

AIM element: organizational_project.name
Source: ISO 10303-41

5.1.2.11 second_in_minute

AIM element: second_in_minute
Source: ISO 10303-41

5.1.2.12 Time_offset

AIM element: coordinated_universal_time_offset
Source: ISO 10303-41

5.1.2.12.1 hour_offset

AIM element: coordinated_universal_time_offset.hour_offset
Source: ISO 10303-41

5.1.2.12.2 minute_offset

AIM element: coordinated_universal_time_offset.minute_offset
Source: ISO 10303-41

5.1.2.12.3 sense

AIM_element: coordinated_universal_time_offset.sense
Source: ISO 10303-41

5.1.2.13 Specification

AIM element: document
Source: ISO 10303-41

5.1.2.13.1 id

AIM element: document.id -> identifier
Source: ISO 10303-41

5.1.2.13.2 Specification to Calendar_date (as date)

AIM element: PATH
Reference path: date_item = date
(date_item <-
applied_date_assignment.item
applied_date_assignment <=
date_assignment
{date_assignment.role->
date_role
date_role.name = 'date'}
date_assignment.assigned_date ->
date =>
calendar_date)

5.1.2.13.3 Specification to Organisation (as issued_by)

AIM element: PATH
Reference path: (organization_item = document
organization_item <-
applied_organization_assignment.item
applied_organization_assignment <=
organization_assignment
{organization_assignment.role ->
organization_role
organization_role.name = 'issued_by'})

5.1.2.13.4 Specification to Specification_type (as of_type)

AIM element: PATH
(document
document.description ->
{document_type
document_type.product_data_type='specification'})
Source: ISO 10303-41

5.1.2.14 Specification_relationship

AIM element: document_relationship
Source: ISO 10303-41

5.1.2.14.1 relation_type

AIM element: document_relationship.name
Source: ISO 10303-41

5.1.2.14.2 description

AIM element: document_relationship.description
Source: ISO 10303-41

5.1.2.14.3 Specification_relationship to Specification (as relating)

AIM element: document_relationship.relating_document
Source: ISO 10303-41

5.1.2.14.4 Specification_relationship to Specification (as related)

AIM element: document_relationship.relating_document
Source: ISO 10303-41

5.1.2.15 Specification_type

AIM element: document_type
Source: ISO 10303-41

5.1.2.15.1 name

AIM element: document_type.product_data_type
Source: ISO 10303-41

5.1.3 Approval UoF

5.1.3.1 Approval

AIM element: approval
Source: ISO 10303-41

5.1.3.1.1 purpose

AIM element: approval_role
Source: ISO 10303-41

5.1.3.1.2 Approval to Approval_status (as status)

AIM element: approval.status
Source: ISO 10303-41

5.1.3.1.3 Approval to Certificate (as defined_in)

AIM element: PATH
Reference path: (certification_item = approval
certification_item <-
applied_certification_assignment.item
applied_certification_assignment <=
certification_assignment
{certification_assignment.role ->
certification_role
certification_role.name = 'defined_in'})

5.1.3.1.4 Approval to Approving_person_organisation (as authorised_by)

1. If the approval is given only by a person

AIM element: PATH
Reference path: (approval <-
approval_person_organization.authorised_approval
approval_person_organization
approval_person_organization.person_organization ->
person_organization_select
person_organization_select = person
person)

2. If the approval is given only by an organisation

AIM element: PATH
Reference path: (approval <-
approval_person_organization.authorised_approval
approval_person_organization
approval_person_organization.person_organization ->
person_organization_select
person_organization_select = organization
organization)

3. If the approval is given by a person in an organisation

AIM element: PATH
Reference path: (approval <-
approval_person_organization.authorised_approval
approval_person_organization
approval_person_organization.person_organization ->
person_organization_select
person_organization_select = person_and_organization
person_and_organization)

5.1.3.2 Approval_assignment

AIM element: applied_approval_assignment
Source: ISO 10303-235
Reference path: (applied_approval_assignment <=
approval_assignment)

5.1.3.2.1 items

AIM element: applied_approval_assignment.item
Source: ISO 10303-235
Reference path: applied_approval_assignment.item ->
approval_item ->
(action,
action_method,
document,
product,
property_definition,
person,
person_in_organization,
organization)

5.1.3.2.2 role

AIM element: PATH
Reference path: (applied_approval_assignment <=
approval_assignment ->
approval_assignment.role)

5.1.3.2.3 Approval_assignment to Approval (as assigned_approval)

AIM element: PATH
Reference path: (applied_approval_assignment <=
approval_assignment ->
approval_assignment.assigned_approval)

5.1.3.3 Approval_relationship

AIM element: approval_relationship
Source: ISO 10303-41

5.1.3.3.1 description

AIM element: approval_relationship.description
Source: ISO 10303-41

5.1.3.3.2 relation_type

AIM element: approval_relationship.name
Source: ISO 10303-41

5.1.3.3.3 related_approval

AIM element: approval_relationship.related_approval
Source: ISO 10303-41

5.1.3.3.4 relating_approval

AIM element: approval_relationship.relating_approval
Source: ISO 10303-41

5.1.3.4 Approving_person_organisation

AIM element: approval_person_organization
Source: ISO 10303-41

5.1.3.4.1 role

AIM element: approval_person_organization.role
Source: ISO 10303-41

5.1.3.4.2 Approving_person_organization to AP_qualification_type (as qualification)

AIM element: PATH
Reference path: (qualification_item = person_and_organization
qualification_item <-
applied_qualification_assignment.item
applied_qualification_assignment <=
qualification_type_assignment
{qualification_type_assignment.role = 'qualification'})

5.1.3.5 AP_qualification_type

AIM element: qualification_type
Source: ISO 10303-41

5.1.3.5.1 name

AIM element: qualification_type.name
Source: ISO 10303-41

5.1.3.5.2 description

AIM element: qualification_type.description
Source: ISO 10303-41

5.1.3.6 Approval_status

AIM element: approval_status
Source: ISO 10303-41

5.1.3.6.1 name

AIM element: approval_status.name
Source: ISO 10303-41

5.1.3.7 Certificate

AIM element: certification
Source: ISO 10303-41

5.1.3.7.1 cert_name

AIM element: certification.name
Source: ISO 10303-41

5.1.3.7.2 cert_id

AIM element: Path
Reference path: (identification_item = certification
identification_item <-
applied_identification_item.item
applied_identification_assignment <=
identification_assignment
{identification_assignment.role ->
identification_role
identification_role.name = 'cert_id'})

5.1.3.7.3 Certificate to Calendar_date (as cert_date)

AIM element: PATH
Reference path: (date_item = certification
date_item <-
applied_date_assignment.item
applied_date_assignment <=
date_assignment
{date_assignment.role->
date_role
date_role.name = 'cert_date'}
date_assignment.assigned_date ->
date =>
calendar_date)

5.1.3.7.4 Certificate to Organisation (as issued_by)

AIM element: PATH
Reference path: (organization_item = approval
organization_item <-
applied_organization_assignment.item
applied_organization_assignment <=
organization_assignment

{organization_assignment.role ->
organization_role
organization_role.name = 'issued_by'})

5.1.3.8 Certificate_assignment

AIM element: applied_certification_assignment
Source: ISO 10303-235 (See 5.2.2.6)

5.1.3.8.1 certificated_items

AIM element: applied_certification_assignment.item
Source: ISO 10303-235 (See 5.2.2.6)

5.1.3.8.2 role

AIM element: PATH
Reference path: (applied_certification_assignment <=
certification_assignment
certification_assignment.role)

5.1.3.9 Security_classification

AIM element: security_classification
Source: ISO 10303-41

5.1.3.9.1 classification_level

AIM element: security_classification_level.name
Source: ISO 10303-41
Reference path: security_classification
security_classification.security_level ->
security_classification_level
security_classification_level.name

5.1.3.9.2 description

AIM element: security_classification.purpose
Source: ISO 10303-41

5.1.4 Condition UoF

5.1.4.1 Condition

AIM element: state_observed
Source: ISO 10303-56

5.1.4.1.1 name

AIM element: state_observed.name
Source: ISO 10303-56

5.1.4.1.2 description

AIM element: state_observed.description
Source: ISO 10303-56

5.1.4.2 Condition_assignment

AIM element: applied_observed_type_assignment
Source: ISO 10303-235
Reference path: applied_state_observed_assignment <=
state_observed_assignment

5.1.4.2.1 Condition_assignment to Condition (as assigned_condition)

AIM element: PATH
Reference path: (applied_state_observed_assignment <=
state_observed_assignment ->
state_observed_assignment.assigned_state_observed ->
state_observed)

5.1.4.2.2 Condition_assignment toconditioned_item (as assigned_item)

AIM element: PATH
Reference path: (applied_state_observed_assignment ->
applied_state_observed_assignment.item ->
state_observed_item)

5.1.4.3 Condition_evaluation

AIM element: ascribable_state
Source: ISO 10303-56

5.1.4.3.1 description

AIM element: ascribable_state.description
Source: ISO 10303-56

5.1.4.3.2 name

AIM element: ascribable_state.name
Source: ISO 10303-56

5.1.4.3.3 Condition_evaluation to Condition (as condition)

AIM element: ascribable_state.ascribable_state_observed
Source: ISO 10303-56

5.1.4.3.4 Condition_evaluation to State_type (as evaluation_state)

AIM element: ascribable_state.pertaining_state_type
Source: ISO 10303-56

5.1.4.4 Condition_evaluation_assignment

AIM element: applied_condition_evaluation_assignment
Source: ISO 10303-235

5.1.4.5 Condition_relationship

AIM element: state_observed_relationship
Source: ISO 10303-56

5.1.4.5.1 description

AIM element: state_observed_relationship.description
Source: ISO 10303-56

5.1.4.5.2 relation_type

AIM element: state_observed_relationship.name
Source: ISO 10303-56

5.1.4.5.3 Condition_relationship to Condition (as related_condition)

AIM element: state_observed_relationship.related_state_observed
Source: ISO 10303-56

5.1.4.5.4 Condition_relationship to Condition (as relating_condition)

AIM element: state_observed_relationship.relating_state_observed
Source: ISO 10303-56

5.1.5 Document management UoF

5.1.5.1 Digital_document_definition

AIM element: product_definition
Source: ISO 10303-41
Reference path: {product_definition

product_definition.frame_of_reference->
product_definition_context<=
application_context_element
application_context_element.name ='digital document definition'}

5.1.5.2 Digital_file

AIM element: document
Source: ISO 10303-41
Reference_path: [document
{document<-document_representation_type.represented_document
document_representation_type.name='digital'}]

5.1.5.3 Digital_record

AIM element: /SUBTYPE(Digital_file)/ –(See 5.1.5.2)

5.1.5.4 Document

AIM element: product
Source: ISO 10303-41
Reference_path: product
{product <-
product_related_product_category.products
product_related_product_category <=
product_category
product_category.name='document'}

5.1.5.4.1 document_name

AIM element: product.name
Source: ISO 10303-41

5.1.5.5 Document_assignment

1. If the assignment is not a partial document reference

AIM element: applied_document_reference
Source: ISO 10303-235 (See 5.2.2.10)
Reference path: (applied_document_reference <=
document_reference)

2. If the assignment is a partial document assignment

AIM element: applied_document_usage_constraint_assignment
Source: ISO 10303-235 (See5.2.2.11)
Reference_path: (applied_document_usage_constraint <=
document_usage_constraint_assignment)

5.1.5.5.1 assigned_role

1. If the assignment is not a partial document_assignment

AIM element: object_role.name
Source: ISO 10303-41
Reference path: (applied_document_reference <=
document_reference
document_reference.role ->
object_role
object_role.name)

2. If the assignment is a partial document assignment

AIM element: document_usage_role.name
Source: ISO 10303-41
Reference path: (applied_document_usage_constraint_assignment <=
document_usage_constraint_assignment
document_usage_constraint_assignment.role ->
document_usage_role
document_usage_role.name)

5.1.5.5.2 Document_assignment to Document (as assigned_document)

1. If the assignment is not a partial document assignment

AIM element: PATH
Reference path: applied_document_reference <=
document_reference
document_reference.assigned_document ->
document <-
{document.kind -> document_type
document_type.product_data_type ='configuration_controlled_document'}
document_product_association.relating_document
document_product_association
{document_product_association => document_product_equivalence}
document_product_association.related_product ->
product_or_formation_or_definition
product_or_formation_or_definition = product
product <- product_related_product_category.products
product_related_product_category <= product_category
{product_category.name = 'document'}

2. If the assignment is to a partial document assignment

AIM element: PATH
Reference path: applied_document_usage_constraint_assignment <=
document_usage_constraint_assignment
document_useage_constraint_assignment.assigned_document_usage ->
document_usage_constraint
document_usage_constraint.source ->
document <-
{document.kind -> document_type
document_type.product_data_type ='configuration_controlled_document'}
document_product_association.relating_document
document_product_association
{document_product_association => document_product_equivalence}
document_product_association.related_product ->
product_or_formation_or_definition

product_or_formation_or_definition = product
product <- product_related_product_category.products
product_related_product_category <= product_category
{poduct_category.name = 'document'}

5.1.5.5.3 Document_assignment to documented_items (as item)

1. If assignment is not a partial document assignment

AIM element: PATH
Reference path: applied_document_reference
applied_document_reference.items[i] ->
document_item

2. If assignment is a partial document assignment

AIM element: PATH
Reference path: applied_document_usage_constraint_assignment
applied_document_usage_constraint_assignment.items[i] ->
document_item

5.1.5.6 Document_definition

AIM element: product_definition
Source: ISO 10303-41
Rules: restrict_document_definition_category
Reference path: {product_definition
product_definition.frame_of_reference ->
product_definition_context <=
application_context_element
(application_context_element.name = 'digitial document definition')
(application_context_element.name = 'physical document definition')

5.1.5.6.1 definition_id

AIM element: product_definition.id
Source: ISO 10303-41

5.1.5.6.2 description

AIM element: product_definition.description
Source: ISO 10303-41

5.1.5.6.3 Document_definition to Document_version (as associated_version)

AIM element: PATH
Reference path: product_definition
{product_definition.frame_of_reference ->
product_definition_context <=
application_context_element
(application_context_element.name = 'digitial document definition')

(application_context_element.name = 'physical document definition')}
{product_definition.formation->product_definition_formation
product_definition_formation.of_product->
product<-
product_related_product_category.products
product_related_product_category<=
product_category
product_category.name='document'}

5.1.5.6.4 Document_definition to File (as files)

AIM element: PATH
Reference path: product_definition =>
product_definition_with_associated_documents
product_definition_with_associated_documents.documentation_ids[i] ->
document
{documents <- document_representation_type.represented_document
document_representation_type.name = 'file'

5.1.5.7 Document_definition_relationship

AIM element: product_definition_relationship
Source: ISO 10303-41

5.1.5.7.1 relation_type

AIM element: product_definition_relationship.name
Source: ISO 10303-41

5.1.5.7.2 description

AIM element: product_definition_relationship.description
Source: ISO 10303-41

5.1.5.7.3 Document_definition_relationship to Document_definition (as relating_document_definition)

AIM element: PATH
Reference path: product_definition_relationship
product_definition_relationship.relating_product_definition ->
product_definition
{product_definition.frame_of_reference ->
product_definition_context <=
application_context_element
(application_context_element.name = 'physical_document_definition')
(application_context_element.name = 'digital_document_definition')}

5.1.5.7.4 Document_definition_relationship to Document_definition (as related_document_definition)

AIM element: PATH

Reference path: product_definition_relationship
product_definition_relationship.related_product_definition ->
product_definition
{product_definition.frame_of_reference ->
product_definition_context <=
application_context_element
(application_context_element.name = 'physical_document_definition')
(application_context_element.name = 'digital_document_definition')}

5.1.5.8 Document_location_identification

AIM element: /SUPERTYPE(External_source_identification)/ -- (See 4.2.5.14.)

5.1.5.9 Document_relationship

AIM element: product_relationship
Source: ISO 10303-41

5.1.5.9.1 description

AIM element: product_relationship.description
Source: ISO 10303-41

5.1.5.9.2 relation_type

AIM element: product_relationship.name
Source: ISO 10303-41

5.1.5.9.3 Document_relationship to Document (as relating_document)

AIM element: product_relationship.relating_product->product
Source: ISO 10303-41

5.1.5.9.4 Document_relationship to Document (as related_document)

AIM element: product_relationship.related_product->product
Source: ISO 10303-41

5.1.5.10 Document_version

AIM element: product_definition_formation
Source: ISO 10303-41
Reference path: product_definition_formation
{product_definition_formation.of_product ->product
product <-
product_related_product_category.products[i]
product_related_product_category <=
product_category

product_category.name='document'}

5.1.5.10.1 Document_version to Document (as SELF\Product_version.of_product)

AIM element: PATH
Reference path: product_definition_formation.of_product ->product
{product <-
product_related_product_category.products[i]
product_related_product_category <=
product_category
product_category.name='document}

5.1.5.11 External_item_identification

AIM element: externally_defined_item
Source: ISO 10303-41

5.1.5.11.1 external_id

AIM element: externally_defined_item.item_id
Source: ISO 10303-41

5.1.5.12 External_source_identification

AIM element: external_source
Source: ISO 10303-41

5.1.5.12.1 description

AIM element: external_source.description
Source: ISO 10303-41

5.1.5.12.2 source_id

AIM element: external_source.source.id
Source: ISO 10303-41

5.1.5.12.3 source_type

AIM element: identification_role.name
Source: ISO 10303-41
Reference path: (applied_external_identification_assignment<=
external_identification_assignment<=
identification_assignment
identification_assignment.role->
identification_role
identification_role.name)

5.1.5.12.4 External_source_identification to external_identification_item (as item)

AIM element: PATH
Reference path: applied_external_identification_assignment.items[i] ->
external_identification_item

5.1.5.13 File

AIM element: document
Source: ISO 10303-41
Reference path: [document
{document<-document_representation_type.represented_document
(document_representation_type.name='digital')
(document_representation_type.name='physical')}]

5.1.5.13.1 file_name

AIM element: document.name
Source: ISO 10303-41

5.1.5.14 File_location_identification

AIM element: /SUPERTYPE(External_item_identification)/

5.1.5.15 File_relationship

AIM element: document_relationship
Source: ISO 10303-41

5.1.5.15.1 description

AIM element: document_relationship.description
Source: ISO 10303-41

5.1.5.15.2 relation_type

AIM element: document_relationship.name
Source: ISO 10303-41

5.1.5.15.3 related_file

AIM element: document_relationship.related_document->document
Source: ISO 10303-41

5.1.5.15.4 relating_file

AIM element: document_relationship.relating_document->document

Source: ISO 10303-41

5.1.5.16 Hard_copy

AIM element: document
Source: ISO 10303-41
Reference path: [document
{document<-document_representation_type.represented_document
document_representation_type.name='physical'}]

5.1.5.17 Physical_document_definition

AIM element: product_definition
Source: ISO 10303-41
Reference path: {product_definition
product_definition.frame_of_reference->
product_definition_context<=
application_context_element
application_context_element.name='physical document definition'}

5.1.6 Effectivity UoF

5.1.6.1 Dated_effectivity

AIM element: dated_effectivity
Source: ISO 10303-41
Reference path: dated_effectivity <= effectivity

5.1.6.1.1 Dated_effectivity to Calendar_date (as start_bound)

AIM element: PATH
Reference path: dated_effectivity.effectivity_start_date ->
date_time_or_event_occurrence
{date_time_or-event_occurrence = date_time_select
date_time_select = date
date => calendar_date

5.1.6.1.2 Dated_effectivity to Date_time (as start_bound)

AIM element: PATH
Reference path: dated_effectivity.effectivity_start_date ->
date_time_or_event_occurrence
{date_time_or_event_occurrence = date_time_select
date_time_select = date_and_time}

5.1.6.1.3 Dated_effectivity to Event (as start_bound)

AIM element: PATH
Reference path: dated_effectivity.effectivity_start_date ->

{date_time_or_event_occurrence = event_occurrence}

5.1.6.1.4 Dated_effectivity to Calendar_date (as end_bound)

AIM element: PATH
Reference path: dated_effectivity.effectivity_end_date ->
date_time_or_event_occurrence
{date_time_or-event_occurrence = date_time_select
date_time_select = date
date => calendar_date

5.1.6.1.5 Dated_effectivity to Date_time (as end_bound)

AIM element: PATH
Reference path: dated_effectivity.effectivity_end_date ->
date_time_or_event_occurrence
{date_time_or_event_occurrence = date_time_select
date_time_select = date_and_time}

5.1.6.1.6 Dated_effectivity to Event (as end_bound)

AIM element: PATH
Reference path: dated_effectivity.effectivity_end_date ->
{date_time_or_event_occurrence = event_occurrence}

5.1.6.2 Effectivity

AIM element: effectivity
Source: ISO 10303-41

5.1.6.2.1 description

AIM element: effectivity.description
Source: ISO 10303-41

5.1.6.2.2 effectivity_id

AIM element: effectivity.id
Source: ISO 10303-41

5.1.6.2.3 name

AIM element: effectivity.name
Source: ISO 10303-41

5.1.6.3 Effectivity_assignment

AIM element: applied_effectivity_assignment

Source: ISO 10303-235
Reference path: applied_effectivity_assignment <=
effectivity_assignment

5.1.6.3.1 items

AIM element: PATH
Reference path: applied_effectivity_assignment ->
applied_effectivity_assignment.items ->
effectivity_item ->
(approval,
action_method
document,
product_definition_formation)

5.1.6.3.2 role

AIM element: PATH
Reference path: applied_effectivity_assignment <=
effectivity_assignment ->
effectivity_assignment.role

5.1.6.3.3 Effectivity_assignment to Effectivity (as assigned_effectivity)

AIM element: PATH
Reference path: applied_effectivity_assignment <=
effectivity_assignment ->
effectivity_assignment.assigned_effectivity

5.1.6.4 Effectivity_relationship

AIM element: effectivity_relationship
Source: ISO 10303-41

5.1.6.4.1 description

AIM element: effectivity_relationship.description
Source: ISO 10303-41

5.1.6.4.2 relation_type

AIM element: effectivity_relationship.name
Source: ISO 10303-41

5.1.6.4.3 Effectivity_relationship to Effectivity (as relating_effectivity)

AIM element: PATH

Reference path: effectivity_relationship.relating_effectivity ->
effectivity

5.1.6.4.4 Effectivity_relationship to Effectivity (as related_effectivity)

AIM element: PATH
Reference path: effectivity_relationship.related_effectivity ->
effectivity

5.1.6.5 Frequency_interval

AIM element: (measure_with_unit
[measure_with_unit.value_component]
[measure_with_unit.unit_component])
Source: ISO 10303-41

5.1.6.5.1 rate

AIM element: measure_with_unit.unit_component
Source: ISO 10303-41
Reference path: {measure_with_unit.unit_component->
unit->
named_unit=>
si_unit
si_unit.name->si_unit_name = 'hertz'}

5.1.6.5.2 accumulation

AIM element: measure_with_unit.value_component
Source: ISO 10303-41
Reference path: {measure_with_unit.value_component->
measure_value->
count_measure}

5.1.6.6 Lot_effectivity

AIM element: lot_effectivity
Source: ISO 10303-41
Reference path: lot_effectivity <= effectivity

5.1.6.6.1 lot_id

AIM element: lot_effectivity.effectivity_lot_id
Source: ISO 10303-41

5.1.6.6.2 lot_size

AIM element: lot_effectivity.effectivity_lot_size
Source: ISO 10303-41
Reference path; lot_effectivity.effectivity_lot_size ->
measure_with_unit

5.1.6.7 Serial_effectivity

AIM element: serial_numbered_effectivity
Source: ISO 10303-41
Reference path: serial_numbered_effectivity <= effectivity

5.1.6.7.1 start_id

AIM element: serial_numbered_effectivity.effectivity_start_id
Source: ISO 10303-41

5.1.6.7.2 end_id

AIM element: serial_numbered_effectivity.effectivity_end_id
Source: ISO 10303-41

5.1.6.8 Time_interval

AIM element: time_interval
Source: ISO 10303-41

5.1.6.8.1 description

AIM_element: time_interval.description
Source: ISO 10303-41

5.1.6.8.2 id

AIM_element: time_interval.id
Source: ISO 10303-41

5.1.6.8.3 name

AIM element: time_interval.name
Source: ISO 10303-41

5.1.6.9 Time_interval_effectivity

AIM element: time_interval_based_effectivity
Source: ISO 10303-41
Reference path: time_interval_based_effectivity <= effectivity

5.1.6.9.1 Time_interval_effectivity to Time_interval (as effectivity_period)

AIM element: PATH

Reference path: time_interval_based_effectivity.effectivity_period ->
time_interval

5.1.6.10 Time_interval_relationship

AIM element: time_interval_relationship
Source: ISO 10303-41

5.1.6.10.1 description

AIM element: time_interval_relationship.description
Source: ISO 10303-41

5.1.6.10.2 relation_type

AIM element: time_interval_relationship.name
Source: ISO 10303-41

5.1.6.10.3 Time_interval_relationship to Time_interval (as relating_interval)

AIM element: PATH
Reference path: time_interval_relationship.relating_time_interval ->
time_interval

5.1.6.10.4 Time_interval_relationship to Time_interval (as related_interval)

AIM element: PATH
Reference path: time_interval_relationship.related_time_interval ->
time_interval

5.1.6.11 Time_interval_with_bounds

AIM element: time_interval_with_bounds
Source: ISO 10303-41

5.1.6.11.1 Time_interval _with_bounds to Calendar_date (as primary_bound)

AIM element: PATH
Reference path: time_interval_with_bounds.primary_bound ->
date_time_or_event_occurrence
{date_time_or_event_occurrence = date_time_select
date_time_select = date
date => calendar_date }

5.1.6.11.2 Time_interval_with_bounds to Date_time (as primary_bound)

AIM element: PATH
Reference path: time_interval_with_bounds.primary_bound ->

date_time_or_event_occurrence
{date_time_or_event_occurrence = date_time_select
date_time_select = date_and_time}

5.1.6.11.3 Time_interval_with_bounds to Event (as primary_bound)

AIM element: PATH
Reference path: time_interval_with_bounds.primary_bound ->
date_time_or_event_occurrence
{date_time_or_event_occurrence = event_occurrence}

5.1.6.11.4 Time_interval_with_bounds to Calendar_date (as secondary_bound)

AIM element: PATH
Reference path: time_interval_with_bounds.secondary_bound ->
date_time_or_event_occurrence
{date_time_or_event_occurrence = date_time_select
date_time_select = date
date => calendar_date }

5.1.6.11.5 Time_interval_with_bounds to Date_time (as secondary_bound)

AIM element: PATH
Reference path: time_interval_with_bounds.secondary_bound ->
date_time_or_event_occurrence
{date_time_or_event_occurrence = date_time_select
date_time_select = date_and_time}

5.1.6.11.6 Time_interval_with_bounds to Event (as secondary_bound)

AIM element: PATH
Reference path: time_interval_with_bounds.secondary_bound ->
date_time_or_event_occurrence
{date_time_or_event_occurrence = event_occurrence}

5.1.6.11.7 Time_interval_with_bounds to Duration (as duration)

AIM element: PATH
Reference path: time_interval_with_bounds.derivation -> time_measure_with_unit

5.1.7 External_reference UoF

5.1.7.1 Alias_identification

AIM element: applied_identification_assignment
Source: ISO 10303-214
Reference path: applied_identification_assignment <=
identification_assignment
{identification_assignment
identification_assignment.role ->

identification_role
identification_role.name = 'alias'}

5.1.7.1.1 alias_id

AIM element: identification_assignment.assigned_id
Source: ISO 10303-41
Reference path: applied_identification_assignment <=
identification_assignment
identification_assignment.assigned_id

5.1.7.1.2 alias_version_id

AIM element: identification_assignment.assigned_id
Source: ISO 10303-41
Rules: dependent_instantiable_identification_role,
restrict_version_assignment_for_applied_identification_assignment
Reference path: applied_identification_assignment
identification_item = applied_identification_assignment
identification_item <-
applied_identification_assignment.items[i]
applied_identification_assignment <=
identification_assignment
{identification_assignment.role ->
identification_role
identification_role.name = 'version'}
identification_assignment.assigned_id

5.1.7.1.3 description

AIM element: identification_role.description
Source: ISO 10303-41
Rules: dependent_instantiable_identification_role
Reference path: applied_identification_assignment <=
identification_assignment
identification_assignment.role ->
identification_role
identification_role.description

5.1.7.1.4 Alias_identification to Organization (as alias_scope)

AIM element: PATH
Reference path: applied_identification_assignment
organization_item = applied_identification_assignment
organization_item <-
applied_organization_assignment.items[i]
applied_organization_assignment <=
organization_assignment
{organization_assignment.role ->
organization_role
organization_role.name = 'alias scope'}

organization_assignment.assigned_organization ->
organization

5.1.7.1.5 Alias_identification to Multi_language_string (as description)

1: For the primary_language_dependent_string.

2: For any additional_language_dependent_string.

AIM element: 1: ([identification_role.description]
[PATH])

2: (PATH)
Source: ISO 10303-41
Rules: restrict_multi_language_for_identification_role
Reference path: identification_role
1: (attribute_language_item = identification_role
attribute_language_item <-
attribute_language_assignment.items[i]
attribute_language_assignment
{attribute_language_assignment <=
attribute_classification_assignment
attribute_classification_assignment.attribute_name = 'description'})
2: (multi_language_attribute_item = identification_role
multi_language_attribute_item <-
multi_language_attribute_assignment.items[i]
multi_language_attribute_assignment
{multi_language_attribute_assignment <=
attribute_value_assignment
attribute_value_assignment.attribute_name = 'description'})

5.1.7.1.6 Alias_identification to Activity_property (as is_applied_to)

AIM element: PATH
Reference path: applied_identification_assignment
applied_identification_assignment.items[i] ->
identification_item
identification_item = action_property
action_property

5.1.7.1.7 Alias_identification to Activity_type (as is_applied_to)

AIM element: PATH
Reference path: applied_identification_assignment
applied_identification_assignment.items[i] ->
identification_item
identification_item = action_method
action_method

5.1.7.1.8 Alias_identification to Document (as is_applied_to)

AIM element: PATH

Reference path: applied_identification_assignment
applied_identification_assignment.items[i] ->
identification_item
identification_item = product
product
{product <-
product_related_product_category.products[i]
product_related_product_category <=
product_category
product_category.name = 'document'}

5.1.7.1.9 Alias_identification to Engineering_property (as is_applied_to)

AIM element: PATH
Reference path: applied_identification_assignment
applied_identification_assignment.items[i] ->
identification_item
identification_item = material_property
material_property

5.1.7.1.10 Alias_identification to Product_type (as is_applied_to)

AIM element: PATH
Reference path: applied_identification_assignment
applied_identification_assignment.items[i] ->
identification_item
identification_item = product_definition
product_definition

5.1.7.1.11 Alias_identification to Resource_item (as is_applied_to)

AIM element: PATH
Reference path: applied_identification_assignment
applied_identification_assignment.items[i] ->
identification_item
identification_item = action_resource
action_resource

5.1.7.1.12 Alias_identification to Resource_property (as is_applied_to)

AIM element: PATH
Reference path: applied_identification_assignment
applied_identification_assignment.items[i] ->
identification_item
identification_item = resource_property
resource_property

5.1.7.1.13 Alias_identification to Substance (as is_applied_to)

AIM element: PATH

Reference path: applied_identification_assignment
applied_identification_assignment.items[i] ->
identification_item
identification_item = product
product

5.1.7.2 External_library_reference

AIM element: externally_defined_item
Source: ISO 10303-41

5.1.7.2.1 description

AIM element: PATH
Source: ISO 10303-41
Reference path: external_source
external_source.description

5.1.7.2.2 external_id

AIM element: externally_defined_item.item_id ->
source_item
Source: ISO 10303-41

5.1.7.2.3 library_type

AIM element: externally_defined_item.source ->
external_source
Source: ISO 10303-41

5.1.7.3 Language

AIM element: language
Source: ISO 10303-214
Reference path: language <=
group

5.1.7.3.1 country_code

AIM element: group.description
Source: ISO 10303-41
Reference path: language <=
group
group.description

5.1.7.3.2 language_code

AIM element: group.name
Source: ISO 10303-41
Reference path: language <=
group
group.name

5.1.7.4 Multi_language_string

1: If additional_language_dependent_string is not populated.

2: If additional_language_string is populated.

AIM element: 1: (attribute_language_assignment)
2: ([attribute_language_assignment]
[multilanguage_attribute_assignment])
Source: ISO 10303-214
Rules: dependent_instantiable_attribute_value_role
Reference path: 1: (attribute_language_assignment <=
attribute_classification_assignment
{attribute_classification_assignment.role)
classification_role
classification_role.name = 'primary'})
2: ([attribute_language_assignment <=
attribute_classification_assignment
{attribute_classification_assignment.role ->
classification_role
classification_role.name = 'primary'}]
[multi_language_attribute_assignment <=
attribute_value_assignment
{attribute_value_assignment.role ->
attribute_value_role
attribute_value_role.name = 'alternate language'
attribute_language_item = attribute_value_assignment
attribute_language_item <-
attribute_language_assignment.items[i]
attribute_language_assignment
{[attribute_classification_assignment.role ->
classification_role
classification_role.name = 'translated'
[attribute_classificaiton_assignment.attribute_name = 'attribute_value']}])

5.1.7.4.1 Mutli_language_string to String_with_language (as additional_language_dependent_string)

AIM element: IDENTICAL MAPPING

5.1.7.4.2 Multi_language_string to String_with_language (as primary_language_string)

AIM element: IDENTICAL MAPPING

5.1.7.5 Plib_class_reference

AIM element: externally_defined_class
Source: ISO 10303-235
Rules: restrict_name_for_known_source
Reference_path: externally_defined_class <=
[class <= group]
[externally_defined_item

{externally_defined_item.source ->
external_source =>
known_source <=
pre_defined_item
pre_defined_item_name = 'ISO 13584 library'}]

5.1.7.5.1 code

AIM element: externally_defined_item.item_id
Source: ISO 10303-41
Reference path: externally_defined_class <=
externally_defined_item
{externally_defined_item
externally_defined_item.source ->
external_source =>
known_source <=
pre_defined_item
pre_defined_item.name = 'ISO 13584 library'}
externally_defined_item.item_id
externally_defined_item.item_id ->
source_item
source_item = identifier
identifier}

5.1.7.5.2 supplier_bsu

AIM element: organization.id
Source: ISO 10303-41
Rules: dependent_instantiable_organization_role,
externally_defined_class_with_known_source_requirement
Reference path: externally_defined_class <=
{externally_defined_class <=
externally_defined_item
externally_defined_item.source ->
external_source =>
known_source <=
pre_defined_item
pre_defined_item.name = 'ISO 13584 library'}
class
organization = class
organization_item <-
applied_organisation_assignment_items[i]
applied_organisaton_assignment <=
organization_assignment
{organization_assignment.role ->
organization_role
organization_role.name = 'class supplier'}
organization_assignment.assigned_organization ->
organiation.id

5.1.7.5.3 version

AIM element: idenfication_assignment.assigned_id
Source: ISO 10303-41
Rules: plib_class_reference_requires_version
Reference path: externally_defined_class <=
{externally_defined_class <=

externally_defined_item
externally_defined_item.source ->
external_source =>
known_source <=
pre_defined_item
pre_defined_item.name = 'ISO 13584 library'}
external_identification_item = externally_defined_class
external_identification_item <-
applied_external_identification_assignment.items[i]
applied_external_identification_assignment <=
{external_identification_assignment
external_identification_assignment.source ->
external_source =>
known_source <=
pre_defined_item
pre_defined_item.name = 'ISO 13584 library'}
{identification_assignment.role
identification_role
identification_role_name = 'version'}
identification_assignment.assigned_id

5.1.7.6 Plib_property_reference

AIM element: externally_defined_engineering_property
Source: ISO 10303-235
Reference path: externally_defined_engineering_property <=
[material_property]
[externally_defined_item]

5.1.7.6.1 code

AIM element: externally_defined_item.item_id
Source: ISO 10303-41
Reference_path: externally_defined_class <=
externally_defined_item
{externally_defined_item
externally_defined_item.source ->
external_source =>
known_source <=
pre_defined_item
pre_defined_item.name = 'ISO 13584 library'}
externally_defined_item.item_id
externally_defined_item.item_id ->
source_item
source_item = identifier
identifier}

5.1.7.6.2 version

AIM element: idenfication_assignment.assigned_id
Source: ISO 10303-41
Rules: plib_property_reference_requires_version
Reference path: externally_defined_class <=
{externally_defined_class <=
externally_defined_item
externally_defined_item.source ->
external_source =>

known_source <=
pre_defined_item
pre_defined_item.name = 'ISO 13584 library'}
external_identification_item = externally_defined_class
external_identification_item <-
applied_external_identification_assignment.items[i]
applied_external_identification_assignment <=
{external_identification_assignment
external_identification_assignment.source ->
external_source =>
known_source <=
pre_defined_item
pre_defined_item.name = 'ISO 13584 library'}
{identification_assignment.role
identification_role
identification_role_name = 'version'}
identification_assignment.assigned_id

5.1.7.6.3 plib_property_reference to plib_class_reference (as name_scope)

AIM element: PATH
Source: ISO 10303-31
Rules: plib_property_reference_required_name_scope,
restrict_externally_defined_item_relationship
Reference path: externally_defined_engineering_property <=
externally_defined_item
{externally_defined_item.source ->
external_source =>
known_source <=
pre_defined_item
pre_defined_item.name = 'ISO 13584 library'}
externally_defined_item_relationship.relating_item
externally_defined_item_relationship
{externally_defined_item_relationship.name = 'name scope'}
externally_defined_item_relationship.related_item ->
externally_defined_item =>
{externally_defined_item.source ->
external_source =>
known_source <=
pre_defined_item
pre_defined_item.name = 'ISO 13584 library'}
externally_defined_class

5.1.7.7 String_with_language

1: If the String_with_language does not play the role of an additional_language_dependent_string.
2: If the String_with_language does play the role of an additional_language_dependent_string.

AIM element: 1: (attribute_language_assignment)
2: ([attribute_language_assignment]
[multi_language_assignment])
Source: ISO 10303-214
Reference path: 1: (attribute_language_assignment <=
attribute_classification_assignment
{attribute_classification_assignment.role ->
classification_role

classification_role.name = 'primary'})
2: ([attribute_language_assignment <=
attribute_classification_assignment
{attribute_classification_assignment.role ->
classification_role
classification_role.name = 'primary'}]
multi_language_attribute_ssignment <=
attribute_value_assignment
{attribute_value_assignment.role ->
attribute_value_role
attribute_value_role.name = 'alternate language'
attribute_language_item <-
attribute_language.items[i]
attribute_language_assignment <=
attribute_classification_assignment
{[attribute_classification_assignment.role ->
classification_role
classification_role.name = 'translated'
[attribute_classification_assignment.attribute_name = 'attribute_value']}])

5.1.7.7.1 contents

AIM element: 1: (IDENTICAL MAPPING)
2: (attribute_value_assignment.attribute_value)
Reference path: 2. (multi_language_attribute_assignment <=
attribute_value_assignment
attribute_value_assignment.attribute_value

5.1.7.7.2 String_with_language to Language (as language_specification)

AIM element: PATH
Reference path: attribute_language_assignment <=
attrbribute_classification_assignment
attribute_classification_assignment.assigned_class ->
group =>
language

5.1.8 Geometry UoF

5.1.8.1 Axis_placement

1: if the geometric space is two dimensional

AIM element: (axis2_placement_2d)
Reference path: geometric_representation_item => placement
placement => axis2_placement_2d

2: if the geometric space is three dimensional

AIM element: (axis2_placement_3d)
Reference path: geometric_representation_item => placement
placement => axis2_placement_3d

5.1.8.1.1 Axis_placement to Cartesian_point (as origin)

AIM element: PATH
Reference path: placement.location -> cartesian.point

5.1.8.1.2 Axis_placement to Direction (as x_axis)

1: if the geometric space is two dimensional

AIM element: PATH
Reference path: axis2_placement_2d.p[1] -> direction

2: if the geometric space is three dimensional

AIM element: PATH
Reference path: axis2_placement_3d.p[1] -> direction

5.1.8.1.3 Axis_placement to Direction (as y_axis)

1: if the geometric space is two dimensional

AIM element: PATH
Reference path: axis2_placement_2d.p[2] -> direction

2: if the geometric space is three dimensional

AIM element: PATH
Reference path: axis2_placement_3d.p[2] -> direction

5.1.8.2 Axis_placement_mapping

AIM element: item_defined_transformation
Source: ISO 10303-43

5.1.8.2.1 Axis_placement_mapping to Axis_placement (as source)

AIM element: PATH
Reference path: item_defined_transformation.transform_item_1 ->
representation_item
representation_item => geometric_representation_item
(geometric_representation_item => placement
placement => axis2_placement_3d)

5.1.8.2.2 Axis_placement_mapping to Axis_placement (as target)

AIM element: PATH
Reference path: item_defined_transformation.transform_item_2 ->
representation_item
representation_item => geometric_representation_item
(geometric_representation_item => placement
placement => axis2_placement_3d

5.1.8.3 Axis_placement_transformation_mapping

AIM element: item_defined_transformation
Source: ISO 10303-43

5.1.8.3.1 Axis_placement_transformation_mapping to Axis_placement (as source)

AIM element: PATH
Reference path: item_defined_transformation.transform_item_1 ->
representation_item
representation_item => geometric_representation_item
(geometric_representation_item => placement
placement => axis2_placement_3d)

5.1.8.3.2 Axis_placement_transformation_mapping to Cartesian_transformation_2D (as target)

AIM element: PATH
Reference path: item_defined_transformation.transform_item_2 ->
representation_item
representation_item =>geometric_representation_item
geometric_representation_item => cartesian_transformation_operator
cartesian_transformation_operator => cartesian_transformation_operator_2d

5.1.8.3.3 Axis_placement_transformation_mapping to Cartesian_transformation_3D (as target)

AIM element: PATH
Reference path: item_defined_transformation.transform_item_2 ->
representation_item
representation_item =>geometric_representation_item
geometric_representation_item => cartesian_transformation_operator
cartesian_transformation_operator => cartesian_transformation_operator_3d

5.1.8.4 Cartesian_point

AIM element: cartesian_point
Source: ISO 10303-42
Reference path: geometric_representation_item => point
point => cartesian_point

5.1.8.4.1 coordinates

AIM element: cartesian_point.coordinates
Source: ISO 10303-42

5.1.8.5 Cartesian_transformation_2D

AIM element: cartesian_transformation_operator_2d

Source: ISO 10303-42
Reference path: geometric_representation_item => cartesian_transformation_operator
cartesian_transformation_operator => cartesian_transformation_operator_2d

5.1.8.5.1 Cartesian_transformation_2D to Direction (as multiplication_matrix)

AIM element: PATH
Reference path: cartesian_transformation_operator_2d <= cartesian_transformation_operator
[cartesian_transformation_operator.axis1 -> direction]
[cartesian_transformation_operator.axis2 -> direction]

5.1.8.5.2 Cartesian_transformation_2D to Cartesian_point (as translation)

AIM element: PATH
Reference path: cartesian_transformation_operator_2d <= cartesian_transformation_operator
cartesian_transformation_operator.local_origin -> cartesian_point

5.1.8.6 Cartesian_transformation_3D

AIM element: cartesian_transformation_operator_3d
Source: ISO 10303-42
Reference path: geometric_representation_item => cartesian_transformation_operator
cartesian_transformation_operator => cartesian_transformation_operator_3d

5.1.8.6.1 Cartesian_transformation_3D to Direction (as multiplication_matrix)

AIM element: PATH
Reference path: cartesian_transformation_operator_3d <= cartesian_transformation_operator
[cartesian_transformation_operator.axis1 -> direction]
[cartesian_transformation_operator.axis2 -> direction]
[cartesian_transformation_operator.axis3 -> direction]

5.1.8.6.2 Cartesian_transformation_3D to Cartesian_point (as translation)

AIM element: PATH
Reference path: cartesian_transformation_operator_3d <= cartesian_transformation_operator
cartesian_transformation_operator.local_origin -> cartesian_point

5.1.8.7 Direction

AIM element: direction
Source: ISO 10303-42
Reference path: geometric_representation_item => direction

5.1.8.7.1 coordinates

AIM element: direction.direction_ratios
Source: ISO 10303-42

5.1.8.8 Geometric_coordinate_space

AIM element: geometric_representation_context
Source: ISO 10303-42
Reference path: geometric_representation_context <= representation_context

5.1.8.8.1 dimension_count

AIM element: geometric_representation_context.coordinate_space_dimension
Source: ISO 10303-42

5.1.8.9 Geometric_model

AIM element: shape_representation
Source: ISO 10303-41

5.1.8.9.1 model_name

AIM element: PATH
Reference path: shape_representation <=
representation
representation.name

5.1.8.9.2 Geometric_model to Geometric_coordinate_space (as context_of_items)

AIM element: PATH
Reference path: shape_representation <=
representation
representation.context_of_items ->
representation_context =>
geometric_representation_context

5.1.8.9.3 Geometric_model to Geometric_model_element (as items)

AIM element: PATH
Reference path: shape_representation <=
representation
representation.items[i] ->
representation_item => geometric_representation_item

5.1.8.10 Geometric_model_element

AIM element: geometric_representation_item
Source: ISO 10303-42
Reference path: representation_item =>
geometric_representation_item

5.1.8.11 Shape

AIM element: product_definition_shape
Source: ISO 10303-41

5.1.8.11.1 shape_description

AIM element: PATH
Reference path: product_definition_shape <=
property_definition
property_definition.description

5.1.8.11.2 shape_name

AIM element: PATH
Reference path: product_definition_shape <=
property_definition
property_definition.name

5.1.8.11.3 Shape to Shape_dimension (as dimension)

AIM element: PATH
Reference path: (product_definition_shape <=
property_definition<-
property_definition_representation
property_defintion_representation->
representation =>
shape_representation=>
shape_dimension_representation)

5.1.8.12 Shape_defined_in external_reference

AIM element: externally_defined_class
Source: ISO 10303-235

5.1.8.12.1 reference_select

AIM element: PATH
Reference path: externally_defined_class <=
externally_defined_item ->
externally_defined_item.item_id

5.1.8.13 Shape_dimension

AIM element: shape_dimension_representation
Source: ISO 10303-47

5.1.8.13.1 dimension_name

AIM element: PATH
Reference path: (shape_dimension_representation<-
dimensional_characteristic_representation.representation
dimensional_characteristic_representation.dimension ->
dimensional_characteristic->
dimensional_size
dimensional_size.name -> label)

5.1.8.13.2 Shape_dimension to Measure_item (as dimension_value)

AIM element: PATH
Reference path: shape_dimension_representation<=
shape_representation<=
representation ->
representation_item =>
measure_representation_item <=
measure_with_unit

5.1.8.14 Shape_element

AIM element: shape_aspect
Source: ISO 10303-41

5.1.8.14.1 element_name

AIM element: shape_aspect.name
Source ISO 10303-41

5.1.8.14.2 description

AIM element: shape_aspect.description
Source: ISO 10303-41

5.1.8.14.3 Shape_element to Shape (as of_shape)

AIM element: shape_aspect.of_shape
Source: ISO 10303-41

5.1.9 Geometric tolerance UoF

5.1.9.1 Angularity_tolerance

AIM element: angularity_tolerance
Source: ISO 10303-519
Reference path: angularity_tolerance <=
geometric_tolerance_with_datum_reference

5.1.9.1.1 Angularity_tolerance to Reference_datum (as ref_datum)

AIM element: PATH
Reference path: angularity_tolerance <=
geometric_tolerance =>
geometric_tolerance_with_datum_reference
geometric_tolerance_with_datum_reference.datum_system[i] ->
datum_reference

5.1.9.2 Coaxiality_tolerance

AIM element: coaxiality_tolerance
Source: ISO 10303-519
Reference path: coaxiality_tolerance <=
geometric_tolerance_with_datum_reference

5.1.9.2.1 Coaxiality_tolerance to Reference_datum (as ref_datum)

AIM element: PATH
Reference path: coaxiality _tolerance <=
geometric_tolerance =>
geometric_tolerance_with_datum_reference
geometric_tolerance_with_datum_reference.datum_system[i] ->
datum_reference

5.1.9.3 Concentricity_tolerance

AIM element: concentricity_tolerance
Source: ISO 10303-519
Reference path: concentricity_tolerance <=
geometric_tolerance_with_datum_reference

5.1.9.3.1 Concentricity_tolerance to Reference_datum (as ref_datum)

AIM element: PATH
Reference path: concentricity _tolerance <=
geometric_tolerance =>
geometric_tolerance_with_datum_reference
geometric_tolerance_with_datum_reference.datum_system[i] ->
datum_reference

5.1.9.4 Cylindricity_tolerance

AIM element: cylindricity_tolerance
Source: 10303-519
Reference path: cylindricity_tolerance <=
geometric_tolerance

5.1.9.5 Flatness_tolerance

AIM element: flatness_tolerance
Source: 10303-519
Reference path: flatness_tolerance <=
geometric_tolerance

5.1.9.6 Geometric_tolerance

AIM element: geometric_tolerance
Source: ISO 10303-47
Rules: subtype_exclusiveness_geometric_tolerance
subtype_mandatory_geometric_tolerance

5.1.9.6.1 Geometric_tolerance to Measure_item (as tolerance_value)

AIM element: PATH
Reference path: geometric_tolerance =>
geometric_tolerance.magnitude ->
measure_with_unit =>
length_measure_with_unit

5.1.9.6.2 Geometric_tolerance to Shape_element (as applied_to)

AIM element: PATH
Reference path: geometric_tolerance
geometric_tolerance.toleranced_shape_aspect
shape_aspect

5.1.9.6.3 Geometric_tolerance to Tolerance_condition (as modification)

AIM element: PATH
Reference path: geometric_tolerance =>
modified_geometric_tolerance
modified_geometric_tolerance.modifier

5.1.9.7 Geometric_tolerance_relationship

AIM element: geometric_tolerance_relationship
Source: ISO 10303-47

5.1.9.7.1 Geometric_tolerance_relationship to Geometric_tolerance (as related_tolerance)

AIM element: PATH
Reference path: geometric_tolerance_relationship
geometric_tolerance_relationship.related_geometric_tolerance ->
geometric_tolerance

5.1.9.7.2 Geometric_tolerance_relationship to Geometric_tolerance (as relating_tolerance)

AIM element: PATH
Reference path: geometric_tolerance_relationship
geometric_tolerance_relationship.relating_geometric_tolerance ->
geometric_tolerance

5.1.9.7.3 relation_type

AIM element: geometric_tolerance_relationship.name
Source: ISO 10303-47

5.1.9.8 Parallelism_tolerance

AIM element: parallelism_tolerance
Source: ISO 10303-519
Reference path: parallelism_tolerance <=
geometric_tolerance_with_datum_reference

5.1.9.8.1 Parallelism_tolerance to Reference_datum (as ref_datum)

AIM element: PATH
Reference path: parallelism _tolerance <=
geometric_tolerance =>
geometric_tolerance_with_datum_reference
geometric_tolerance_with_datum_reference.datum_system[i] ->
datum_reference

5.1.9.9 Perpendicularity_tolerance

AIM element: perpendicularity_tolerance
Source: ISO 10303-519
Reference path: perpendicularity _tolerance <=
geometric_tolerance_with_datum_reference

5.1.9.9.1 Perpendicularity_tolerance to Reference_datum (as ref_datum)

AIM element: PATH
Reference path: perpendicularity _tolerance <=
geometric_tolerance =>
geometric_tolerance_with_datum_reference
geometric_tolerance_with_datum_reference.datum_system[i] ->
datum_reference

5.1.9.10 Position_tolerance

AIM element: position_tolerance
Source: ISO 10303-519
Reference path: position _tolerance <=
geometric_tolerance_with_datum_reference

5.1.9.10.1 Position_tolerance to Reference_datum (as ref_datum)

AIM element: PATH
Reference path: position _tolerance <=
geometric_tolerance =>
geometric_tolerance_with_datum_reference
geometric_tolerance_with_datum_reference.datum_system[i] ->
datum_reference

5.1.9.11 Reference_datum

AIM element: datum_reference
Source: ISO 10303-47

5.1.9.11.1 Reference_datum to Datum (as datum_reference)

AIM element: datum_reference.referenced_datum
Source: ISO 10303-47

5.1.9.12 Roundness_tolerance

AIM element: roundness_tolerance
Source: ISO 10303-519
Reference path: roundness_tolerance <=
geometric_tolerance

5.1.9.13 Straightness_tolerance

AIM element: straightness_tolerance
Source: ISO 10303-519
Reference path: straightness _tolerance <=
geometric_tolerance

5.1.9.14 Symmetry_tolerance

AIM element: symmetry_tolerance
Source: ISO 10303-519
Reference path: symmetry _tolerance <=
geometric_tolerance_with_datum_reference

5.1.9.14.1 Symmetry_tolerance to Reference_datum (as ref_datum)

AIM element: PATH
Reference path: symmetry _tolerance <=
geometric_tolerance =>
geometric_tolerance_with_datum_reference
geometric_tolerance_with_datum_reference.datum_system[i] ->

datum_reference

5.1.9.15 Tolerance_zone

AIM element: tolerance_zone
Source: ISO 10303-47

5.1.9.15.1 form_type

AIM element: tolerance_zone_form.name
Source: ISO 10303-47

5.1.9.15.2 Tolerance_zone to Geometric_tolerance (as zone_for)

AIM element: PATH
Reference path: tolerance_zone
tolerance_zone.defining_tolerance[i] ->
geometric_tolerance

5.1.9.16 Tolerance_zone_definition

AIM element: tolerance_zone_definition
Source: ISO 10303-47

5.1.9.16.1 Tolerance_zone_definition to Tolerance_zone (as defining)

AIM element: PATH
Reference path: tolerance_zone_definition
tolerance_zone_definition.zone ->
tolerance_zone

5.1.9.16.2 Tolerance_zone_definition to Shape_element (as first_element)

AIM element: PATH
Reference path: tolerance_zone_definition
tolerance_zone_definition.boundaries[i]
shape_aspect

5.1.9.16.3 Tolerance_zone_definition to Shape_element (as second_element)

AIM element: PATH
Reference path: tolerance_zone_definition
tolerance_zone_definition.boundaries[i]
shape_aspect

5.1.9.17 Total_runout_tolerance

AIM element: total_runout_tolerance
Source: ISO 10303-519
Reference path: total_runout _tolerance <=
geometric_tolerance_with_datum_reference

5.1.9.17.1 Total_runout_tolerance to Reference_datum (as ref_datum)

AIM element: PATH
Reference path: total_runout _tolerance <=
geometric_tolerance =>
geometric_tolerance_with_datum_reference
geometric_tolerance_with_datum_reference.datum_system[i] ->
datum_reference

5.1.10 Location UoF

5.1.10.1 Address_location_representation

AIM element: representation
Source: ISO 10303-43
Reference path: representation
{representation.name = 'address location'}
representation.context_of_items ->
representation_context
{representation_context.context_type = 'address location'}

5.1.10.1.1 Address_location_representation to Address (as location_address)

AIM element: PATH
Reference path: representation
representation.items ->
list_representation_item[i]
{representation_item.name = 'address'
representation_item ->
address}

5.1.10.2 Global_location_representation

AIM element: representation
Source: ISO 10303-43
Reference path: representation
{representation.name = 'global location'}
representation.context_of_items ->
representation_context
{representation_context.context_type = 'global location'}

5.1.10.2.1 geographical_area

AIM element: representation.description
Source: ISO 10303-43

5.1.10.2.2 Gobal_location_representation to Measure_item (as latitude and longitude and altitude)

AIM element: PATH
Reference path: representation
representation.items ->
list_representation_item[i]
[representation_item[1]
{representation_item[1].name = 'latitude'}
representation_item[1] -> plane_angle_measure_with_unit]
[representation_item[2]
{representation_item[2].name = 'longtitude'}
representation_item[2] -> plane_angle_measure_with_unit]
[representation_item[3]
{representation_item[3].name = 'altitude'}
representation_item[2] -> length_measure_with_unit]

5.1.10.3 Location

AIM element: location
Source: ISO 10303-41

5.1.10.3.1 location_id

AIM element: location.id
Source: ISO 10303-41

5.1.10.3.2 location_name

AIM element: location.name
Source: ISO 10303-41

5.1.10.3.3 Location to Location_representation (as representation)

AIM element: applied_location_representation
Source: ISO 10303-235 (See 5.2.2.18
Reference path: location_representation.location ->
location
location_representation =>
applied_location_representation
applied_location_representation.item = representation

5.1.10.4 Location_representation

AIM element: representation
Source: ISO 10303-43

5.1.10.4.1 description

AIM element: representation.description
Source: ISO 10303-43

5.1.10.4.2 name

AIM element: representation.name
Source: ISO 10303-43
Reference path: representation
representation.name

5.1.10.5 Location_representation_relationship

AIM element: representation_relationship
Source: ISO 10303-43

5.1.10.5.1 description

AIM element: representation_relationship.description
Source: ISO 10303-43

5.1.10.5.2 relation_type

AIM element: representation_relationship.name
Source: ISO 10303-43

5.1.10.5.3 Location_representation_relationship to Location_representation (as related)

AIM element: representation_relationship.rep_1
Source: ISO 10303-43

5.1.10.5.4 Location_representation_relationship to Location_representation (as relating)

AIM element: representation_relationship.rep_2
Source: ISO 10303-43

5.1.10.6 Organisational_location_identification

AIM element: set_representation_item
Source: ISO 10303-43

5.1.10.6.1 identification_type

AIM element: representation_type.name
Source: ISO 10303-43

5.1.10.6.2 identification_value

AIM element: PATH
Reference path: representation_type ->
label

5.1.10.7 Organisation_location_representation

AIM element: representation
Source: ISO 10303-43
Reference path: representation
{representation.name = 'organisational location'}
representation.context_of_items ->
representation_context
{representation_context.context_type = 'organisation location'}

5.1.10.7.1 Organisation_location_representation to Organisational_location_identification (as location_identifications[i])

AIM element: PATH
Reference path: representation
representation.items ->
set_representation_item[i]

5.1.10.8 Product_location_representation

AIM element: representation
Source: ISO 10303-43
Reference path: representation
{representation.name = 'product location'}
representation.context_of_items ->
representation_context
{representation_context.context_type = 'product location'}

5.1.10.8.1 location_identification

AIM element: PATH
Reference path: representation
representation.items ->
representation_item
representation_item.name -> label

5.1.10.8.2 Product_location_representation to Individual_product (as referenced_product)

AIM element: PATH
Reference path: representation
representation.items -> representation_item
representation_item -> product_definition

5.1.10.9 Regional_grid_location_representation

AIM element: representation
Source: ISO 10303-43
Reference path: representation
{representation.name = 'regional grid location'}
representation.context_of_items ->
representation_context
{representation_context.context_type = 'regional grid location'}

5.1.10.9.1 description

AIM element: representation.description
Source: ISO 10303-43

5.1.10.9.2 name

AIM element: representation.name
Source: ISO 10303-43

5.1.10.10 regional_coordinates

AIM element: PATH
Reference path: representation
representation.items ->
list_representation_item[i]
[representation_item -> numeric_measure]

5.1.11 Measure UoF

5.1.11.1 Context_dependent_unit

AIM element: context_dependent_unit
Source: ISO 10303-41

5.1.11.2 Count_measure

AIM element: count_measure
Source: ISO 10303-41

5.1.11.2.1 count_value

AIM element: PATH
Reference path: {count_measure = NUMBER}

5.1.11.3 Derived_unit

AIM element: derived_unit

Source: ISO 10303-41

5.1.11.3.1 unit_elements

AIM element: derived_unit.elements
Source: ISO 10303-41

5.1.11.4 Derived_unit_element

AIM element: derived_unit_element
Source: ISO 10303-41

5.1.11.4.1 base_unit

AIM element: derived_unit_element.unit
Source: ISO 10303-41
Reference path: derived_unit_element.unit ->named_unit

5.1.11.4.2 exponent

AIM element: derived_unit_element.exponent
Source: ISO 10303-41
Reference path: {derived_unit_element.exponent -> REAL}

5.1.11.5 Descriptive_measure

AIM element: descriptive_representation_item
Source: ISO 10303-45

5.1.11.5.1 descriptive_value

AIM element: descriptive_representation_item.description
Source: ISO 10303-45

5.1.11.6 Duration

AIM element: time_measure_with_unit
Source: ISO 10303-41
Reference path: measure_representation_item<=
[representation_item]
[measure_with_unit]
{measure_with_unit=>
time_measure_with_unit}

5.1.11.7 Maths_expression_value

AIM element: maths_value_with_unit

Source: ISO 10303-45

5.1.11.7.1 Maths_expression_value to Unit (as unit)

AIM element: PATH
Reference path: maths_value_with_unit
maths_value_with_unit.unit_component ->
unit

5.1.11.8 Measure_item

AIM element: measure_representation_item
Source: ISO 10303-45

5.1.11.8.1 measure_name

AIM element: representation_item.name
Source: ISO 10303-43
Reference path: measure_representation_item <=
[representation_item]
[measure_with_unit]
representation_item
representation_item.name

5.1.11.9 Numerical_measure

AIM element: measure_with_unit
Source: ISO 10303-41
Reference path: measure_representation_item<=
[representation_item]
[measure_with_unit]
measure_with_unit

5.1.11.9.1 number_value

AIM element: measure_with_unit.value_component
Source: ISO 10303-41
Reference path: measure_with_unit.value_component->
measure_value
{measure_value= numeric_measure
numeric_measure=NUMBER}

5.1.11.9.2 Numerical_measure to Unit (as unit)

AIM element: measure_with_unit.unit_component
Source: ISO 10303-41

5.1.11.10 Qualified_expression

AIM element: maths_value_qualification
Source: ISO 10303-45

5.1.11.10.1 qualifiers

AIM element: maths_value_qualification.qualifiers
Source: ISO 10303-45
Reference path: maths_value_qualification.qualifiers ->
(precision_qualifier)
(type_qualifier)
(uncertainty_qualifier)

5.1.11.10.2 Qualified_expression to Maths_expression_value (as expression_qualfied)

AIM element: maths_value_qualification.qualified_maths_value
Source: ISO 10303-45
Reference path: maths_value_qualification
maths_value_qualification.qualified_maths_value ->
maths_value_with_unit

5.1.11.11 Qualified_measure

AIM element: measure_qualification
Source: ISO 10303-45

5.1.11.11.1 qualifiers

AIM element: measure_qualification.qualifiers
Source: ISO 10303-45
Reference path: qualified_representation_item.qualifiers->
(precision_qualifier)
(type_qualifier)
(uncertainty_qualifier)

5.1.11.11.2 Qualified_measure to Numerical_measure_value (as measure_qualified)

AIM element: measure_qualification.qualified_measure
Source: ISO 10303-45
Reference path: measure_qualification
measure_qualification.qualified_measure ->
measure_with_unit

5.1.11.12 Qualifier_precision

AIM element: precision_qualifier
Source: ISO 10303-45

5.1.11.12.1 value_precision

AIM element: precision_qualifier.precision_value
Source: ISO 10303-45

5.1.11.13 Qualifier_type

AIM element: type_qualifier
Source: ISO 10303-45

5.1.11.13.1 name

AIM element: type_qualifier.name
Source: ISO 10303-45

5.1.11.14 Qualifier_uncertainty

AIM element: uncertainty_qualifier
Source: ISO 10303-45

5.1.11.14.1 description

AIM element: uncertainty_qualifier.description
Source: ISO 10303-45

5.1.11.14.2 measure_name

AIM element: uncertainty_qualifier.measure_name
Source: ISO 10303-45

5.1.11.15 Range_measure

AIM element: ([value_range]
[qualified_representation_item])
Source: ISO 10303-240
ISO 10303-45
Reference path: {value_range<=
compound_representation_item
compound_representation_item.item_element->
set_representation_item
set_representation_item[i]->
representation_item=>
measure_representation_item<=
measure_with_unit
measure_with_unit.unit_component->
unit}

5.1.11.15.1 Range_measure to Measure_item (as lower_limit)

AIM element: measure_with_unit.value_component
Source: ISO 10303-41
Reference path: value_range <=
compound_representation_item
compound_representation_item.item_element ->
set_representation_item
set_representation_item[i] ->
representation_item =>
{representation_item.name = 'lower limit'}
measure_representation_item <=
measure_with_unit
measure_with_unit.value_component

5.1.11.15.2 Range_measure to Measure_item (as upper_limit)

AIM element: measure_with_unit.value_component
Source: ISO 10303-41
Reference path: value_range <=
compound_representation_item
compound_representation_item.item_element ->
set_representation_item
set_representation_item[i] ->
representation_item =>
{representation_item.name = 'upper limit'}
measure_representation_item <=
measure_with_unit
measure_with_unit.value_component

5.1.11.16 Ratio_measure

AIM element: ratio_measure
Source: ISO 10303-41

5.1.11.17 Uncertainty_expanded

AIM element: expanded_uncertainty
Source: ISO 10303-45

5.1.11.17.1 coverage_factor

AIM element: expanded_uncertainty.coverage_factor
Source: ISO 10303-45

5.1.11.18 Uncertainty_qualitative

AIM element: qualitative_uncertainty
Source: ISO 10303-45

5.1.11.18.1 uncertainty_value

AIM element: qualitative_uncertainty.uncertainty_value
Source: ISO 10303-45

5.1.11.19 Uncertainty_standard

AIM element: standard_uncertainty
Source: ISO 10303-45

5.1.11.19.1 uncertainty_value

AIM element: standard_uncertainty.uncertainty_value
Source: ISO 10303-45

5.1.11.20 Unit

AIM element: named_unit
Source: ISO 10303-41

5.1.11.21 Unit_relationship

AIM element: conversion_based_unit
Source: ISO 10303-41

5.1.11.21.1 conversion_factor

AIM element: conversion_based_unit.conversion_factor
Source: ISO 10303-41

5.1.12 Person organisation UoF

5.1.12.1 Address

AIM element: address
Source: ISO 10303-41

5.1.12.2 Facsimile_address

AIM element: /SUPERTYPE(Address)/ --(See 5.1.12.1)

5.1.12.2.1 facsimile_number

AIM element: address. facsimile_number
Source: ISO 10303-41

5.1.12.3 Network_address

AIM element: /SUPERTYPE(Address)/ --(See 5.1.12.1)

5.1.12.3.1 url

AIM element: address.electronic_mail_address
Source: ISO 10303-41

5.1.12.4 Organisation

AIM element: organization
Source: ISO 10303-41

5.1.12.4.1 org_address

AIM element: organizational_address
Source: ISO 10303-41

5.1.12.4.2 org_id

AIM element: organization.id
Source: ISO 10303-41

5.1.12.4.3 org_name

AIM element: organization.name
Source: ISO 10303-41

5.1.12.4.4 Organisation to Organisation_type (as of_type)

AIM element: PATH
Reference path:

5.1.12.4.5 Organisation to Qualification (as org_qualification)

AIM element: PATH
Reference path: organization<-
{applied_qualification_assignment.item='organisation'}
applied_qualification_assignment<=
qualification_type_assignment
qualification_type_assignment.assigned_qualification_type->
qualification_type

5.1.12.5 Organisation_relationship

AIM element: organization_relationship

Source: ISO 10303-41

5.1.12.5.1 relation_type

AIM element: organization_relationship.name
Source: ISO 10303-41

5.1.12.5.2 description

AIM element: organization_relationship.description
Source: ISO 10303-41

5.1.12.5.3 Organisation_relationship to Organisation (as related_organisation)

AIM element: PATH
Reference path: organization_relationship.related_organization -> organization

5.1.12.5.4 Organisation_relationship to Organisation (as relating_organisation)

AIM element: PATH
Reference path: organization_relationship.relating_organization -> organization

5.1.12.6 Organisation_type

AIM element: organization_type
Source: ISO 10303-41

5.1.12.6.1 name

AIM element: organization_type.name
Source: ISO 10303-41

5.1.12.6.2 description

AIM element: organization_type.description
Source: ISO 13030-41

5.1.12.7 Person

AIM element: person
Source: ISO 10303-41

5.1.12.7.1 person_name

AIM element: PATH
Reference path: person_name[i]

{person_name[1]=person.last_name
person_name[2]=person.first_name
person_name[3]=person.middle_names}

5.1.12.7.2 prefix_title

AIM element: person.prefix_titles
Source: ISO 10303-41

5.1.12.7.3 suffix_title

AIM element: person.suffix_titles
Source: ISO 10303-41

5.1.12.7.4 Person to Qualification (as person_qualification)

AIM element: PATH
Reference path: person<-
{applied_qualification_assignment.item='person'}
applied_qualification_assignment<=
qualification_type_assignment
qualification_type_assignment.assigned_qualification_type->
qualification_type

5.1.12.8 Person_in_organisation

AIM element: person_and_organization
Source: ISO 10303-41

5.1.12.8.1 person_role

AIM element: person_role
Source: ISO 10303-41

5.1.12.8.2 Person_in_organisation to Address (as person_organisation_address)

AIM element: (person_and_organization_address)
Source: ISO/TS 10303-1011
Reference path: (person and organization_address <=
[personal_address]
[organizational_address]

5.1.12.8.3 Person_in_organisation to Organisation (as organisation)

AIM element: person_and_organization.the_organization
Source: ISO 10303-41

5.1.12.8.4 Person_in_organisation to Person (as person)

AIM element: person_and_organization.the_person
Source: ISO 10303-41

5.1.12.9 Postal_address

AIM element: /SUPERTYPE(Address)/ --(See 5.1.12.1)

5.1.12.9.1 country

AIM element: address.country
Source: ISO 10303-41

5.1.12.9.2 postal_town

AIM element: address.town
Source: ISO 10303-41

5.1.12.9.3 postal_box

AIM element: address.postal_box
Source: ISO 10303-41

5.1.12.9.4 post_code

AIM element: address.postal_code
Source: ISO 10303-41

5.1.12.9.5 region

AIM element: address.region
Source: ISO 10303-41

5.1.12.9.6 street_name

AIM element: address.street
Source: ISO 10303-41

5.1.12.9.7 street_number

AIM element: address.street_number
Source: ISO 10303-41

5.1.12.10 Qualification

AIM element: qualification_type
Source: ISO 10303-41

5.1.12.10.1 qual_id

AIM element: qualification_type.id
Source: ISO 10303-41

5.1.12.10.2 qual_name

AIM element: qualification_type.name
Source: ISO 10303-41

5.1.12.10.3 qual_descr

AIM element: qualification_type.description
Source: ISO 10303-41

5.1.12.11 Qualification_relationship

AIM element: qualification_type_relationship
Source: ISO 10303-41

5.1.12.11.1 description

AIM element: qualification_type_relationship.description
Source: ISO 10303-41

5.1.12.11.2 relation_type

AIM element: qualification_type_relationship.name
Source: ISO 10303-41

5.1.12.11.3 Qualification_relationship to Qualification (as related_qual)

AIM element: qualification_type_relationship.related_qualification_type
Source: ISO 10303-41
Reference path: qualification_type_relationship.related_qualification_type->
qualification

5.1.12.11.4 Qualification_relationship to Qualification (as relating_qual)

AIM element: qualification_type_relationship.relating_qualification_type
Source: ISO 10303-41
Reference path: qualification_type_relationship.relating_qualification_type->

qualification

5.1.12.12 Telephone_address

AIM element: /SUPERTYPE(Address)/ -- (See 5.1.12.1)

5.1.12.12.1 telephone_number

AIM element: address.telephone_number
Source: ISO 10303-41

5.1.13 Product UoF

5.1.13.1 Individual_product

AIM element: product_as_individual
Source: ISO 10303-235
Reference path: product_definition_formation=>
product_as_individual=>
(product_as_planned)
(product_as_realised)

5.1.13.1.1 description

AIM element: product_definition_formation.description
Source: ISO 10303-41

5.1.13.1.2 id

AIM element: product_definition_formation.id
Source: ISO 10303-41

5.1.13.1.3 of_type

AIM element: product_definition_formation.of_product
Source: ISO 10303-41

5.1.13.2 Individual_product_feature

AIM element: shape_aspect
Source: ISO 10303-41

5.1.13.2.1 feature_description

AIM element: shape_aspect.description
Source: IS 10303-41

5.1.13.2.2 feature_name

AIM element: shape_aspect.name
Source: ISO 10303-41

5.1.13.2.3 whole_or_part

AIM element: shape_aspect.product_definitional
Source: ISO 10303-41

5.1.13.2.4 Individual_product_feature to Shape (as feature_shape)

AIM element: shape_aspect.of_shape
Source: ISO 10303-41
Reference path: shape_aspect
shape_aspect.of_shape->
product_definition_shape

5.1.13.2.5 Individual_product_feature to Product_location_representation (as feature_location)

AIM element: PATH
Reference path: shape_aspect
shape_aspect.of_shape ->
product_definition_shape <=
property_definition
property_definition.characterized_definition = characterized_product_definition
characterized_product_definition = product_definition
product_definition.formation ->
product_definition_formation
product_definition_formation.of_product ->
product
(applied_location_representation_assignment
applied_location_representatin_assignment.items[i] ->
location_representation_assignment =
product

5.1.13.2.6 Individual_product_feature to Specification (as feature_specification)

AIM element: PATH
Reference path: shape_aspect
shape_aspect.of_shape ->
product_definition_shape <=
property_definition
property_definition.characterized_definition = characterized_product_definition
characterized_product_definition = product_definition
product_definition <-
document_product_association.related_product
document_product_association.relating_document ->
document
document.description = 'Specification'

5.1.13.3 Product_as_planned

AIM element: product_as_planned
Source: ISO 10303-235
Reference path: product_definition_formation=>
product_as_individual=>
product_as_planned

5.1.13.3.1 Product_as_planned to Product_type (as plan)

AIM element: PATH
Reference path: product_as_planned <=
product_as_individual <=
product_definition_formation
product_definition_formation.of_product->
product

5.1.13.4 Product_as_realised

AIM element: product_as_realised
Source: ISO 10303-235

5.1.13.4.1 Product_as_realised to Product_as_planned (as realisation_of)

AIM element: PATH
Reference path: product_as_realised<-
product_planned_to_realised.realised_product
product_planned_to_realised
product_planned_to_realised.planned_product->
planned_product

5.1.13.5 Product_feature_relationship

AIM element: shape_aspect_relationship
Source: ISO 10303-41

5.1.13.5.1 description

AIM element: shape_aspect_relationship.description
Source: ISO 10303-41

5.1.13.5.2 relation_type

AIM element: shape_aspect_relationship.name
Source: ISO 10303-41

5.1.13.5.3 Product_feature_relationship to Individual_product_feature (as related)

AIM element: shape_aspect_relationship.related_shape_aspect

Source: ISO 10303-41

5.1.13.5.4 Product_feature_relationship to Individual_product_feature (as relating)

AIM element: shape_aspect_relationship.related_shape_aspect
Source: ISO 10303-41

5.1.13.6 Product_type

AIM element: product
Source: ISO 10303-41

5.1.13.6.1 id

AIM element: product.id
Source: ISO 10303-41

5.1.13.6.2 name

AIM element: product.name
Source: ISO 10303-41

5.1.13.6.3 description

AIM element: product.description
Source: ISO 10303-41

5.1.13.7 Product_type_defined_in_external_reference

AIM element: externally_defined_class
Source: ISO 10303-235

5.1.13.7.1 Product_type_defined_in_external_class to External_source_identification (as source_id)

AIM element: PATH
Reference path: externally_defined_class <=
externally_defined_item ->
externally_defined_item.source

5.1.13.8 Product_type_relationship

AIM element: product_relationship
Source: ISO 10303-41

5.1.13.8.1 description

AIM element: product_relationship.description

Source: ISO 10303-41

5.1.13.8.2 relation_type

AIM element: product_relationship.name
Source: ISO 10303-41

5.1.13.8.3 Product_type_relationship to Product_type (as related)

AIM element: product_relationship.related_product
Source: ISO 10303-41

5.1.13.8.4 Product_type_relationship to Product_type (as relating)

AIM element: product_relationship.relating_product
Source: ISO 10303-41

5.1.13.8.5 Product_type_specialisation to Product_type (as broader)

AIM element: PATH
Reference path: product_category_relationship.category->
product_category=>
product_related_product_category
product_related_product_category.products[i]->
product

5.1.13.8.6 Product_type_specialisation to Product_type (as narrower)

AIM element: PATH
Reference path: product_category_relationship.sub_category->
product_category=>
product_related_product_category
product_related_product_category.products[i]->
product

5.1.13.9 Product_type_specified

AIM element: product_definition_with_associated_documents
Source: ISO 10303-41
Reference path: product<-
product_definition_formation.of_product
product_definition_formation<-
product_definition.formation
product_definition=>
product_definition_with_associated_documents

5.1.13.9.1 Product_type_specified to Specification (as specification)

AIM element: PATH
Reference path: product_definition_with_associated_documents.document_ids[i]

5.1.13.10 Resource_item

AIM element: action_resource
Source: ISO 10303-41

5.1.13.10.1 resource_characterisation

AIM element: resource_property
Source: ISO 10303-49

5.1.13.11 Resource_item_assignment

AIM element: applied_resource_item_assignment
Source: ISO 10303-235 (See 5.2.2.24)

5.1.13.12 Resource_item_characterisation

AIM element: resource_property
Source: ISO 10303-49

5.1.13.12.1 name

AIM element: resource_property.name
Source: ISO 10303-49

5.1.13.12.2 description

AIM element: resource_property.description
Source: ISO 10303-49

5.1.13.13 Resource_item_relationship

AIM element: action_resource_relationship
Source: ISO 10303-41

5.1.13.13.1 description

AIM element: action_resource_relationship.description
Source: ISO 10303-41

5.1.13.13.2 relation_type

AIM element: action_resource_relationship.name
Source: ISO 10303-41

5.1.13.13.3 Resource_item_relationship to Resource_item (as related)

AIM element: action_resource_relationship.related_resource
Source: ISO 10303-41

5.1.13.13.4 Resource_item_relationship to Resource_item (as relating)

AIM element: action_resource_relationship.relating_resource
Source: ISO 10303-41

5.1.14 Properties UoF

5.1.14.1 Activity_property

AIM element: action_property
Source: ISO 10303-49

5.1.14.1.1 name

AIM element: action_property.name
Source: ISO 10303-49

5.1.14.1.2 description

AIM element: action_property.description
Source: ISO 10303-49

5.1.14.1.3 of_activity

AIM element: action_property.definition
Source: ISO 10303-49
Reference path: action_property
action_property.definition ->
characterized_action_definition ->
(action_method,
action)

5.1.14.2 Action_property_defined_in_external_reference

AIM element: externally_defined_activity_property
Source: ISO 10303-235

5.1.14.2.1 select_source

AIM element: PATH
Reference path: externally_defined_activity_property <=
externally_defined_item
externally_defined_item.source ->
external_source

5.1.14.3 Activity_property_relationship

AIM element: action_property_relationship
Source: ISO 10303-49

5.1.14.3.1 name

AIM element: action_property_relationship.name
Source: ISO 10303-49

5.1.14.3.2 description

AIM element: action_property_relationship.description
Source: ISO 10303-49

5.1.14.3.3 Activity_property_relationship to Activity_property (as related)

AIM element: action_property_relationship.related_action_property
Source: ISO 10303-49

5.1.14.3.4 Action_property_relationship to Activity_property (as relating)

AIM element: action_property_relationship.relating_action_property
Source: ISO 10303-49

5.1.14.4 Activity_property_representation

AIM element: action_property_representation
Source: ISO 10303-49

5.1.14.4.1 name

AIM element: action_property_representation.name
Source: ISO 10303-49

5.1.14.4.2 description

AIM element: action_property_representation.description
Source: ISO 10303-49

5.1.14.4.3 Action_property_representation to Activity_property (as of_property)

AIM element: PATH
Source: ISO 10303-49
Reference path: action_property_representation
action_property_representation.property ->
action_property

5.1.14.5 Engineering_property

AIM element: material_property
Source: ISO 10303-45
Reference path: material_property <=
property_definition

5.1.14.5.1 Engineering_property to Individual_product or Product_type (as assigned_to)

1. If a Engineering_property is assigned to an Individual_product
AIM element: PATH
Reference path: material_property <=
property_definition
{property_definition.characterized_definition ->
characterized_product_definition ->
product_definition}

2. If a Engineering_property is assigned to a Product_type

AIM Element: IDENTICAL MAPPING

5.1.14.6 Engineering_property_defined_in_external_reference

AIM element: externally_defined_engineering_property
Source: ISO 10303-235

5.1.14.6.1 Engineering_property_defined_in_external_reference to External_source_identification (as source_id)

AIM element: PATH
Reference path: externally_defined_engineering_property <=
externally_defined_item ->
externally_defined_item.source

5.1.14.7 Engineering_property_representation

AIM element: material_property_representation
Source: ISO 10303-45
Reference path: material_property_representation<=
property_definition_representation
property_definition_representation.used_representation ->

representation

5.1.14.7.1 property_name

AIM element: PATH
Reference path: material_property_representation<=
property_definition_representation
property_definition_representation.name

5.1.14.7.2 value_elements

AIM element: PATH
Reference path: material_property_representation <=
property_definition_representation
property_definition_representation.used_representation ->
representation
representation.items[i] ->representation_item =>
measure_representation_item

5.1.14.8 Property_environment_relationship

AIM element: data_environment_relationship
Source: ISO 10303-45

5.1.14.8.1 Property_environment_relationship to Property_environment_representation (as related)

AIM element: data_environment_relationship.related_data_environment
Source: ISO 10303-45

5.1.14.8.2 Property_environment_relationship to Property_environment_representation (as relating)

AIM element: data_environment_relationship.relating_data_environment
Source: ISO 10303-45

5.1.14.9 Property_environment_representation

AIM element: data_environment
Source: ISO 10303-45

5.1.14.9.1 Property_environment_representation to Engineering_property_representation (as environment_elements)

AIM element: data_environment.elements
Source: ISO 10303-45

5.1.14.10 Property_representation_relationship

AIM element: PATH
Reference path: material_property_representation <=
property_definition_representation
property_definition_representation.used_representation ->
representation <-
representation_relationship

5.1.14.10.1 description

AIM element: PATH
Reference path: material_property_representation <=
property_definition_representation
property_definition_representation.used_representation ->
representation <-
representation_relationship
representation_relationship.description

5.1.14.10.2 relation_type

AIM element: PATH
Reference path: material_property_representation <=
property_definition_representation
property_definition_representation.used_representation ->
representation <-
representation_relationship
representation_relationship.name

5.1.14.10.3 Property_representation_relationship to Engineering_property_representaton (as related)

AIM element: PATH
Reference path: material_property_representation <=
property_definition_representation
property_definition_representation.used_representation ->
representation <-
representation_relationship
representation_relationship.rep_1

5.1.14.10.4 Property_representation_relationship to Engineering_property_representation (as relating)

AIM element: PATH
Reference path: material_property_representation <=
property_definition_representation
property_definition_representation.used_representation ->
representation <-
representation_relationship
representation_relationship.rep_2

5.1.14.10.5 Property_representation_relationship to Measure_item (as relationship_value)

AIM element: PATH
Reference path: material_property_representation <=
property_definition_representation
property_definition_representation.used_representation ->
representation <-
representation_relationship
representation ->
representation_item =>
measure_representation_item

5.1.14.11 Resource_property

AIM element: resource_property
Source: ISO 10303-49

5.1.14.11.1 name

AIM element: resource_property.name
Source: ISO 10303-49

5.1.14.11.2 description

AIM element: resource_property.description
Source: ISO 10303-49

5.1.14.11.3 of_resource

AIM element: resource_property.resource
Source: ISO 10303-49

5.1.14.12 Resource_property_defined_in_external_reference

AIM element: externally_defined_engineering_property
Source: ISO 10303-235

5.1.14.12.1 source_select

AIM element: PATH
Reference path: externally_defined_activity_property <=
externally_defined_item
externally_defined_item.source ->
external_source

5.1.14.13 Resource_property_representation

AIM element: resource_property_representation

Source: ISO 10303-49

5.1.14.13.1 name

AIM element: resource_property_representation.name
Source: ISO 10303-49

5.1.14.13.2 description

AIM element: resource_property_representation.description
Source: ISO 10303-49

5.1.14.13.3 Resource_property_representation to Resource_property (as of_resource)

AIM element: PATH
Source: ISO 10303-49
Reference path: resource_property_representation
resource_property_representation.property ->
resource_property

5.1.15 Requirements UoF

5.1.15.1 Required_resource

AIM element: |action_resource_requirement|
Source: ISO 10303-49
Reference path: action_reource_requirement
{action_resource_requirement.kind ->
resource_requirement_type
resource_requirement_type.name = 'required_resource'}
{action_resource_requirement.operations[i] ->
characterized_action_definition
characterized_action_definition = action_method
action_method
action_method.name = 'resource_management'}

5.1.15.1.1 name

AIM element: action_resource_requirement.name
Source: ISO 10303-49

5.1.15.1.2 description

AIM element: action_resource_requirement.description
Source: ISO 10303-49

5.1.15.1.3 Required_resource to Measure_item (as resource_quantity)

AIM element: PATH

Source: ISO 10303-41
Reference path: action_resource_requirement
characterized_resource_definition = action_resource_requirement
characterized_resource_definition
characterized_resource_definition <-
resource_property.resource
resource_property
{resource_property.name = 'required_quantity'}
resource_property <-
resource_property_representation.property
resource_property_representation
{resource_property_representation.name = 'required_quantity'}
resource_property_representation.representation ->
representation
[{representation.context_of_items ->
representation_context
representation_context.context_identifier = ''}
{representation.context_of_items ->
representation_context
representation_context.context_type = 'required_resource'}]
representation.items[i] ->
representation_item
representation_item =>
value_representation_item
value_representation_item.value_component ->
measure_value
measure_value <-
measure_with_unit

5.1.15.2 Required_resource_by_resource_item

AIM element: requirement_for_action_resource
Source: ISO 10303-49
Reference path: requirement_for_action_resource <=
action_resource_requirement
{action_resource_requirement.kind ->
resource_requirement_type
resource_requirement_type.name = 'required_resource_by_resource_item'}
{action_resource_requirement.operations[i] ->
characterized_action_definition
characterized_action_definition = action_method
action_method
action_method.name = 'resource_management'}

5.1.15.2.1 Required_resource_by_resource_item to Resource_item (as required_item)

AIM element: PATH
Source: ISO 10303-49
Reference path: requirement_for_action_resource.resources[i] ->
action_resource

5.1.15.3 Required_resource_by_specification

AIM element: |action_resource_requirement|

Source: ISO 10303-49
Reference path: action_resource_requirement
{action_resource_requirement.kind ->
resource_requirement_type
resource_requirement_type.name='required_resource_by_specification'}
{action_resource_requirement.operations[i] ->
characterized_action_definition
characterized_action_definition = action_method
action_method
action_method.name = 'resource_management'}

5.1.15.4 Required_resource_relationship

AIM element: action_resource_requirement_relationship
Source: ISO 10303-49

5.1.15.4.1 description

AIM element: action_resource_requirement_relationship.description
Source: ISO 10303-49

5.1.15.4.2 relation_type

AIM element: action_resource_requirement_relationship.name
Source: ISO 10303-49

5.1.15.4.3 Required_resource_relationship to Required_resource (as relating)

AIM element: PATH
Source: ISO 10303-49
Reference path: action_resource_requirement_relationship.relating_action_resource_requirement ->
action_resource_requirement

5.1.15.4.4 Required_resource_relationship to Required_resource (as related)

AIM element: PATH
Source: ISO 10303-49
Reference path: action_resource_requirement_relationship.related_action_resource_requirement ->
action_resource_requirement

5.1.15.5 Requirement

AIM element: product
Source: ISO 10303-41
Reference path: product
{product <-
product_related_product_category.products[i]
product_related_product_category <=
product_category
product_category.name='requirement'}

5.1.15.6 Requirement_assignment

AIM element: requirement_assignment
Source: ISO/TS 10303-1233
Reference path: requirement_assignment <=
[group]
[characterized_object]

5.1.15.6.1 id

AIM element: group.id
Source: ISO 10303-41
Reference path: requirement_assignment <=
group
group.id

5.1.15.6.2 description

AIM element: group.description
Source: ISO 10303-41
Reference path: requirement_assignment <=
group
group.description

5.1.15.6.3 Requirement_assignment to Required_view_definition (as assignment_requirement)

AIM element: PATH
Reference path: requirement_assignment <-
assigned_requirement.assigned_group
assigned_requirement
assigned_requirement_items ->
product_definition

5.1.15.6.4 Requirement_assignment to requirement_assignment_item (as assigned_to)

AIM element: PATH
Reference path: requirement_assignment <-
requirement_assigned_object.assigned_group
requirement_assigned_object.items ->
requirement_assigned_item

5.1.15.7 Requirement_source

AIM element: requirement_source
Source: ISO/TS 10303-1233
Reference path: requirement_source <=
group

5.1.15.7.1 id

AIM element: group.id
Source: ISO 10303-41
Reference path: requirement_source <=
group
group.id

5.1.15.7.2 description

AIM element: group.description
Source: ISO 10303-41
Reference path: requirement_source <=
group
group.description

5.1.15.7.3 Requirement_source to requirement_source_item (as source)

AIM element: PATH
Reference path: requirement_source <-
source_for_requirement.assigned_group
source_for_requirement
source_for_requirement.items
requirement_source_item

5.1.15.7.4 Requirement_source to Requirement_view_definition (as sourced_requirement)

AIM element: PATH
Reference path: requirement_source <-
sourced_requirement.assigned_group
sourced_requirement
sourced_requirement.items ->
product_definition

5.1.15.8 Requirement_version

AIM element: product_definition_formation
Source: ISO 10303-41
Reference path: product_definition_formation
{product_definition_formation
product_definition_formation.of_product ->
product <-
product_related_product_category.products[i]
product_related_product_category <=
product_category
product_category.name='requirement'}

5.1.15.8.1 description

AIM element: product_definition_formation.description
Source: ISO 10303-41

5.1.15.8.2 version_id

AIM element: product_definition_formation.id
Source: ISO 10303-41

5.1.15.8.3 Requirement_version to Requirement (as of_product)

AIM element: PATH
Reference path: product_definition_formation
product_definition_formation.of_product ->
product
{product <-
product_related_product_category.products[i]
product_related_product_category <=
product_category
product_category.name='requirement'}

5.1.15.9 Requirement_version_relationship

AIM element: product_definition_formation_relationship
Source: ISO 10303-41
Reference path: product_definition_formation_relationship
{product_definition_formation_relationship
product_definition_formation_relationship.relating_product_definition_formation ->
product_definition_formation
product_definition_formation.of_product ->
product <-
product_related_product_category.products[i]
product_related_product_category <=
product_category
product_category.name='requirement'}

5.1.15.9.1 description

AIM element: product_definition_formation_relationship.description
Source: ISO 10303-41

5.1.15.9.2 relation_type

AIM element: product_definition_formation_relationship.name
Source: ISO 10303-41

5.1.15.9.3 Requirement_version_relationship to Requirement_version (as relating_version)

AIM element: product_definition_formation_relationship.relating_product_definition_formation
Source: ISO 10303-41

5.1.15.9.4 Requirement_version_relationship to Requirement_version (as related_version)

AIM element: product_definition_formation_relationship.related_product_definition_formation
Source: ISO 10303-41

5.1.15.10 Requirement_view_definition

AIM element: product_definition
Source: ISO 10303-41
Reference path: product_definition
{product_definition
product_definition.formation ->
product_definition_formation
product_definition_formation.of_product ->
product <-
product_related_product_category.products[i]
product_related_product_category <=
product_category
product_category.name='requirement'}

5.1.15.10.1 satisfied

AIM element: property_definition.description
Source: ISO 10303-41
Reference path: product_definition <-
{product_definition
product_definition.formation ->
product_definition_formation
product_definition_formation.of_product ->
product <-
product_related_product_category.products[i]
product_related_product_category <=
product_category
product_category.name='requirement'}
property_definition.definition
property_definition
property_definition.description
{property_definition
property_definition.name='requirement view definition satisfied'}

5.1.16 State UoF

5.1.16.1 State_type

AIM element: state_type
Source: ISO 10303-56

5.1.16.1.1 description

AIM element: state_type.description
Source: ISO 10303-56

5.1.16.1.2 name

AIM element: state_type.name
Source: ISO 10303-56

5.1.16.2 State_type_assignment

AIM element: state_type_assignment
Source: ISO 10303-56

5.1.16.2.1 State_type_assignment to State_type (as assigned_state_type)

AIM element: state_type_assignment.assigned_state_observed
Source: ISO 10303-56

5.1.16.2.2 State_type_assignment to state_type_of_item (as item_state)

AIM element: PATH
Reference path: state_type_assignment =>
applied_state_assignment ->
applied_state_assignment.item

5.1.16.3 State_type_relationship

AIM element: state_type_relationship
Source: ISO 10303-56

5.1.16.3.1 description

AIM element: state_observed_relationship.description
Source: ISO 10303-56

5.1.16.3.2 relation_type

AIM element: state_type_description.name
Source: ISO 10303-56

5.1.16.3.3 State_type_relationship to State_type (as related_state_type)

AIM element: state_type_relationship.related_state_observed
Source: ISO 10303-56

5.1.16.3.4 State_type_relationship to State_type (as relating_state_type)

AIM element: state_type_relationship.relating_state_observed
Source: ISO 10303-56

5.1.16.4 Derived_state_type

AIM element: state_type_relationship
Source: ISO 10303-56

5.1.16.5 Process_defined_state_type

AIM element: state_type
Source: ISO 10303-56
Reference path: state_type ->
{state_type.name = 'process'}

5.1.16.5.1 Process_defined_state_type to Individual_activity (as defining_activity)

AIM element: action_method
Source: ISO 10303-43

5.1.16.6 Product_type_defined_state_type

AIM element: state_type
Source: ISO 10303-56
Reference path: state_type ->
{state_type.name = 'product'}

5.1.16.6.1 Product_defined_state_type to Product_type (as defining_product)

AIM element: product_definition
Source: ISO 10303-41

5.1.16.7 Property_defined_state_type

AIM element: state_type
Source: ISO 10303-56
Reference path: state_type ->
{state_type.name = 'engineering property'}

5.1.16.7.1 Property_defined_state_type to Engineering_property (as defining_property)

AIM element: material_property
Source: ISO 10303-45

5.1.16.7.2 Property_defined_state_type to Quantity (as defining_quantity)

AIM element: measure_with_unit
Source: ISO 10303-45

5.1.16.8 Quantity

AIM element: state_type
Source: ISO 10303-56
Reference path: state_type ->
{state_type.name = 'quantity'}

5.1.16.8.1 Quantity to Measure_item (as magnitude)

AIM element: measure_with_unit
Source: ISO 10303-41

5.1.17 Substance UoF

5.1.17.1 Structure_element_characterisation

AIM element: property_definition
Source: ISO 10303-41

5.1.17.1.1 description

AIM element: property_definition.description
Source: ISO 10303-41

5.1.17.1.2 Structure_element_characterisation to Geometric_model_element (as characteristic_orientation)

AIM element: PATH
Reference path: property_definition<-
property_definition_representation
property_definition_representation.used_representation ->
{representation => shape_representation}
representation
representation.items[i] ->
{representation_item
representation_item.name = 'orientation'}
representation_item =>
geometric_representation_item =>
placement

5.1.17.1.3 Structure_element_characterisation to Engineering_property_representation (as characteristic_property)

AIM element: PATH
Reference path: property_definition <-
property_definition_representation =>
material_property_representation
property_definition_ representation.used_representation->
representation

representation.item =>
measure_representation_item <=
measure_with_unit

5.1.17.1.4 Structure_element_characterisation to Substance_structure_element (as characterises)

AIM element: PATH
Reference path: property_definition
property_definition.definition ->
characterized_definition ->
shape_definition ->
shape_aspect_relationship

5.1.17.2 Substance

AIM element: product
Source: ISO 10303-41

5.1.17.2.1 id

AIM element: product.id
Source: ISO 10303-41

5.1.17.2.2 description

AIM element: product.description
Source: ISO 10303-41

5.1.17.2.3 substance_name

AIM element: product.name
Source: ISO 10303-41

5.1.17.3 Substance_assignment

AIM element: product_relationship
Source: ISO 10303-41

5.1.17.3.1 of_product

AIM element: product_relationship.relating_product
Source: ISO 10303-41

5.1.17.3.2 substance_role

AIM element: product_relationship.description
Source: ISO 10303-41

5.1.17.3.3 Substance_assignment to Substance (as substance)

AIM element: product_relationship.related_product
Source: ISO 10303-41

5.1.17.4 Substance_composition_element

AIM element: product_material_composition_element
Source: ISO 10303-45

5.1.17.4.1 name

AIM element: PATH
Reference path: product_material_composition_element <=
product_definition_relationship
product_definition_relationship.related_product_definition ->
product_definition
product_definition.name

5.1.17.4.2 description

AIM element: PATH
Reference path: product_material_composition_element <=
product_definition_relationship
product_definition_relationship.related_product_definition ->
product_definition
product_definition.description

5.1.17.4.3 Substance_composition_element to Measure_item (as element_amount)

AIM element: PATH
Reference path: product_material_composition_relationship
product_material_composition_relationship.constituent_amount ->
characterized_product_composition_value
(characterized_product_composition_value ->
maths_value_with_unit)
(characterized_product_composition_value ->
measure_with_unit)

5.1.17.4.4 Substance_composition_element to Substance (as composes)

AIM element: PATH
Reference path: product_material_composition_relationship <=
product_definition_relationship
product_definition_relationship.relating_product_definition ->
product_definition
product_definition.formation ->
product_definition_formation
product_definition_formation.of_product ->
product

5.1.17.4.5 Substance_composition_element to Specification (as composition_specified)

AIM element: PATH
Reference path: product_material_composition_relationship <=
product_definition_relationship
product_definition_relationship.relating_product_definition ->
product_definition =>
product_definition_with_associated_documents
product_definition_with_associated_documents. document_ids ->
document
document.id -> identifier
document.kind -> document_type
document_type.product_data_type = 'Specification'

5.1.17.5 Substance_defined_in_external_reference

AIM element: externally_defined_class
Source: ISO 10303-235

5.1.17.5.1 Substance_defined_in_external_reference to External_source_identification (as source_id)

AIM element: PATH
Reference path: externally_defined_class <=
externally_defined_item ->
externally_defined_item.source

5.1.17.6 Substance_structure_element

AIM element: shape_aspect_relationship
Source: ISO 10303-41

5.1.17.6.1 description

AIM element: PATH
Reference path: shape_aspect_relationship.related_shape_aspect->
shape_aspect
shape_aspect.description

5.1.17.6.2 name

AIM element: PATH
Reference path: shape_aspect_relationship.related_shape_aspect ->
shape_aspect
shape_aspect.name

5.1.17.6.3 Substance_structure_element to Specification (as structure_specified)

AIM element: PATH
Reference path: shape_aspect_relationship.related_shape_aspect ->
shape_aspect

shape_aspect.of_shape ->
product_definition_shape <-
shape_definition <-
characterized_definition ->
product_definition =>
product_definition _with_associated_documents ->
product_definition _with_associated_documents.documentation_ids ->
document
document.kind ->
document_type
{document_type.product_data_type = 'specification'}

5.1.17.6.4 Substance_structure_element to Substance (as structures)

AIM element: PATH
Reference path: shape_aspect_relationship.relating_shape_aspect
shape_aspect
shape_aspect.of_shape ->
product_definition_shape <-
shape_definition <-
characterized_definition ->
product_definition_relationship
product_definition_relationship.relating_product_definition ->
product_definition
product_definition.formation ->
product_definition_formation
product_definition_formation.of_product ->
product

5.1.18 Tolerance datum UoF

5.1.18.1 Compound_datum

1. component datums are referenced at the same Engineering condition

AIM element: (datum)
Source: ISO 10303-47
Reference path: (common_datum <=
[datum]
[composite_shape_aspect])

2. component datums are referenced at different Engineering conditions

AIM element: [referenced_datum]
[referenced_datum]
Source: ISO 10303-47

5.1.18.1.1 Compound_datum to Single_datum (as made_of)

1. component datums are referenced at the same Engineering condition

AIM element: PATH
Reference path: (common_datum <=
composite_shape_aspect

composite_shape_aspect.component_relationships[i] ->
shape_aspect_relationship
shape_aspect_relationship.related_shape_aspect ->
shape_aspect =>
datum

2. component datums are referenced at different Engineering; conditions

AIM element: IDENTICAL MAPPING

5.1.18.2 Datum

AIM element: datum_reference
Source: ISO 10303-47

5.1.18.2.1 precedence

AIM element: datum_reference.precedence
Source: ISO 10303-47

5.1.18.3 Datum_target

AIM element: datum_target
Source: ISO 10303-47

5.1.18.3.1 target_id

AIM element: datum_target.target_id
Source: ISO 10303-47

5.1.18.4 Datum_target_set

AIM element: datum
Source: ISO 10303-47

5.1.18.4.1 rule_description

AIM element: shape_aspect.description
Source: ISO 10303-41
Reference path: datum <=
shape_aspect
shape_aspect.description

5.1.18.4.2 Datum_target_set to Datum_target (as datum_target)

AIM element: PATH
Reference path: datum <=
shape_aspect <-

shape_aspect_relationship.related_shape_aspect
shape_aspect_relationship
shape_aspect_relationship.relating_shape_aspect ->
shape_aspect =>
datum_target

5.1.18.5 Placed_target

AIM element: placed_datum_target_feature
Source: ISO/TS 10303-1051
Reference path: placed_datum_target_feature <=
datum_target

5.1.18.5.1 Placed_target to Geometric_coordinate_space (as defined_in)

AIM element: PATH
Reference path: placed_datum_target_feature <=
datum_target <= shape aspect
shape_definition = shape_aspect
shape_definition
characterized_definition = shape_defininition
characterized_definition <-
property_definition.definition
property_defininition
represented_definition = property_definition
represented_definition <-
property_definition_representation.definition
{property_definition_representation =>
shape_definition_representation}
property_definition_representation
property_defintion_representation.used_representation ->
{representation => shape_representation =>
shape_representation_wih_parameters}
representation
representation_context_of_items ->
representation_context =>
geometric_representation_context

5.1.18.5.2 Placed_target to Axis_placement (as parameter_reference)

AIM element: PATH
Reference path: placed_datum_target_feature <=
datum_target <= shape aspect
shape_definition = shape_aspect
shape_definition
characterized_definition = shape_defininition
characterized_definition <-
property_definition.definition
property_definition
represented_definition = property_definition
represented_definition <-
property_definition_representation.definition
{property_definition_representation =>
shape_definition_representation}

property_definition_representation
property_defintion_representation.used_representation ->
{representation => shape_representation =>
shape_representation_with_parameters}
representation
representation.items[i] ->
representation_item
{representation_item.name = 'orientation'}
representation_item ->geometric_representation_item
geometric_representation_item => placement
placement

5.1.18.6 Reference_shape

AIM element: product_definition_shape
Source: ISO 10303-41

5.1.18.7 Reference_shape_element

AIM element: shape_aspect
Source: ISO 10303-41

5.1.18.8 Single_datum

AIM element: datum
Source: ISO 10303-47

5.1.18.8.1 datum_name

AIM element: datum.identification
Source: ISO 10303-47

5.1.18.8.2 Single_datum to Reference_shape (as defined_by)

AIM element: PATH
Reference path: datum <= shape_aspect
shape_aspect.of_shape ->
product_definition_shape

5.1.18.8.3 Single_datum to Reference_shape_element (as defined_by)

AIM element: PATH
Reference path: datum <= shape_aspect <-
shape_aspect_relationhip.related_shape_aspect
shape_aspect_relationship.relating_shape_aspect ->
shape_aspect
{shape_aspect => datum_feature}

5.1.18.8.4 Single_datum to Datum_target_set (as defined_by)

AIM element: IDENTICAL MAPPING

5.1.18.8.5 Single_datum to Tolerance_condition (as modification)

AIM element: PATH
Reference path: datum <-
datum_reference.referenced_datum
datum_reference =>
referenced_modified_datum
referenced_modified_datum.modifier

5.1.18.9 Target_area

AIM element: datum_target
Source: ISO 10303-47

5.1.18.10 Target_circle

AIM element: placed_datum_target_feature
Source: ISO/TS 10303-1051
Reference path: placed_datum_target_feature <=
datum_target
{datum_target <= shape_aspect
shape_aspect.description = 'circle'}

5.1.18.10.1 Target_circle to Measure_item (as target_diameter)

AIM element: PATH
Reference path: placed_datum_target_feature <=
datum_target <= shape_aspect
shape_definition = shape_aspect
shape_definition
characterized_definition <-
property_definition.definition
property_definition
represented_definition = property_definition
represented_definition <-
property_definiton_representation.definition
{property_definition_representation =>
shape_definition_representation}
property_definition_representation
property_definition_representation.used_representation ->
{representation =>
shape_representation =>
shape_representation_with_parameters}
representation
representation.items[i] ->
representation_item
{representation_item.name = 'target_diameter'}
representation_item => measure_representation_item

{measure_representation_item <=
measure_with_unit =>
length_measure_with_unit}

5.1.18.11 Target_point

AIM element: placed_datum_target_feature
Source: ISO/TS 10303-1051

5.1.18.12 Target_rectangle

AIM element: placed_datum_target_feature
Source: ISO/TS 10303-1051
Reference path: {datum_target <= shape_aspect
shape_aspect.description = 'rectangle'}

5.1.18.12.1 Target_rectangle to Measure_item (as target_length

AIM element: PATH
Reference path: placed_datum_target_feature <=
datum_target <= shape_aspect
shape_definition = shape_aspect
shape_definition
characterized_definition <-
property_definition.definition
property_definition
represented_definition = property_definition
represented_definition <-
property_definiton_representation.definition
{property_definition_representation =>
shape_definition_representation}
property_definition_representation
property_definition_representation.used_representation ->
{representation =>
shape_representation =>
shape_representation_with_parameters}
representation
representation.items[i] ->
representation_item
{representation_item.name = 'target_length'}
representation_item => measure_representation_item
{measure_representation_item <=
measure_with_unit =>
length_measure_with_unit}

5.1.18.12.2 Target_rectangle to Measure_item (as target_width)

AIM element: PATH
Reference path: placed_datum_target_feature <=
datum_target <= shape_aspect
shape_definition = shape_aspect
shape_definition

characterized_definition <-
property_definition.definition
property_definition
represented_definition = property_definition
represented_definition <-
property_definiton_representation.definition
{property_definition_representation =>
shape_definition_representation}
property_definition_representation
property_definition_representation.used_representation ->
{representation =>
shape_representation =>
shape_representation_with_parameters}
representation
representation.items[i] ->
representation_item
{representation_item.name = 'target_width'}
representation_item => measure_representation_item
{measure_representation_item <=
measure_with_unit =>
length_measure_with_unit}

5.1.18.13 Target_straight_line

AIM element: placed_datum_target_feature
Source: ISO/TS 10303-1051
Reference path: placed_datum_target_feature <=
datum_target <=
shape_aspect
shape_aspect.description = 'line'

5.1.18.14 Tolerance_condition

1. If the tolerance_condition is referenced by geometric_tolerance

AIM element: (modified_geometric_tolerance.modifier)
Source: ISO 10303-47

2. If the tolerance_condition is referenced by a single_datum

AIM element: (referenced_modified_datum.modifier)
Source: ISO 10303-47

5.2 AIM EXPRESS short-listing

This clause specifies the EXPRESS schema that uses elements from the integrated resources and contains the types, entity specialisation, rules and functions that are specific to this part of ISO 10303. This clause also specifies modifications to the text for constructs that are imported from the integrated resources. The definition and EXPRESS provided in the integrated resources used in the AIM may include select list items and subtypes that are not imported into the AIM. Requirements stated in the integrated resources that refer to select items and subtypes apply exclusively to those items that are imported into the AIM

EXPRESS specification:

```
SCHEMA engineering_properties_schema;

USE FROM aic_geometric_tolerances              -- ISO 10303-519
       (angularity_tolerance,
       coaxiality_tolerance,
       concentricity_tolerance,
       cylindricity_tolerance,
       flatness_tolerance,
       parallelism_tolerance,
       perpendicularity_tolerance,
       position_tolerance,
       roundness_tolerance,
       straightness_tolerance,
       symmetry_tolerance,
       total_runout_tolerance);

USE FROM action_schema                            --ISO 10303-41
       (action,
       action_method,
       action_method_relationship,
       action_relationship,
       action_resource,
       action_resource_type,
       action_status,
       executed_action,
       supported_item,
       versioned_action_request,
       action_resource_relationship);

USE FROM application_context_schema               -- ISO 10303-41
       (application_context,
       application_context_element,
       application_context_relationship,
       product_concept_context,
       product_context,
       product_definition_context);

USE FROM approval_schema                          -- ISO 10303-41
       (approval,
       approval_date_time,
       approval_person_organization,
       approval_relationship,
       approval_role,
       approval_status);

USE FROM basic_attribute_schema                   -- ISO 10303-41
       (description_attribute_select,
       id_attribute_select,
       name_attribute_select,
       role_select,
       description_attribute,
       id_attribute,
       name_attribute,
       object_role,
       role_association);
```

```
USE FROM certification_schema                    -- ISO 10303-41
       (certification,
       certification_type);

USE FROM  classification_schema                  --ISO 10303-54
       (class);

USE FROM contract_schema                         -- ISO 10303-41
   (contract,
   contract_relationship,
   contract_type);

USE FROM date_time_schema                        -- ISO 10303-41
   (ahead_or_behind,
   calendar_date,
   coordinated_universal_time_offset,
   date,
   date_time_select,
   day_in_month_number,
   day_in_year_number,
   hour_in_day,
   minute_in_hour,
   month_in_year_number,
   second_in_minute,
   week_in_year_number,
   week_of_year_and_day_date,
   year_number,
   year_month,
   date_and_time,
   date_role,
   date_and_time_role,
   date_time_or_event_occurrence,
   event_occurrence,
   event_occurrence_role,
   event_occurence_relationship,
   local_time,
   ordinal_date,
   time_interval,
   time_interval_relationship,
   time_interval_role,
   time_interval_with_bounds);

USE FROM document_schema                         --ISO 10303-41
   (document,
   document_product_association,
   document_relationship,
   document_representation_type,
   document_type,
   document_usage_constraint);

USE FROM ISO13584_generic_expressions_schema  -- ISO 13584-20
       (generic_expression,
       simple_generic_expression,
       generic_literal,
       generic_variable,
       variable_semantics,
       environment,
       unary_generic_expression,
       binary_generic_expression,
```

```
        multiple_arity_generic_expression);

USE FROM ISO13584_expressions_schema  --ISO 13584-20
        (expression,
        variable,
        defined_function,
        SQL_mappable_defined_function,
        numeric_expression,
        simple_numeric_expression,
        literal_number,
        int_literal,
        real_literal,
        numeric_variable,
        int_numeric_variable,
        real_numeric_variable,
        unary_numeric_expression,
        binary_numeric_expression,
        multiple_arity_numeric_expression,
        length_function,
        value_function,
        int_value_function,
        numeric_defined_function,
        plus_expression,
        minus_expression,
        mult_expression,
        div_expression,
        mod_expression,
        power_expression,
        unary_function_call,
        binary_function_call,
        multiple_arity_function_call,
        abs_function,
        minus_function,
        sin_function,
        cos_function,
        tan_function,
        asin_function,
        acos_function,
        exp_function,
        log_function,
        log2_function,
        log10_function,
        square_root_function,
        atan_function,
        maximum_function,
        minimum_function,
        integer_defined_function,
        real_defined_function,
        boolean_expression,
        simple_boolean_expression,
        boolean_literal,
        boolean_variable,
        unary_boolean_expression,
        not_expression,
        odd_function,
        binary_boolean_expression,
        multiple_arity_boolean_expression,
        xor_expression,
        equals_expression,
```

```
       and_expression,
       or_expression,
       comparison_expression,
       comparison_equal,
       comparison_greater,
       comparison_greater_equal,
       comparison_less,
       comparison_less_equal,
       comparison_not_equal,
       like_expression,
       interval_expression,
       boolean_defined_function,
       string_expression,
       simple_string_expression,
       string_literal,
       string_variable,
       index_expression,
       substring_expression,
       concat_expression,
       format_function,
       string_defined_function);

USE FROM external_reference_schema                -- ISO 10303-41
       (message,
       source_item,
       external_source,
       external_source_relationship,
       externally_defined_item,
       externally_defined_item_relationship,
       pre_defined_item);

USE FROM effectivity_schema                       -- ISO 10303-41
       (dated_effectivity,
       effectivity,
       effectivity_relationship,
       lot_efectivity,
       serial_numbered_effectivity,
       time_interval_based_effectivity);

USE FROM geometry_schema                          -- ISO 10303-42
       (dimension_count,
       axis1_placement,
       axis2_placement,
       axis2_placement_2d,
       axis2_placement_3d,
       curve,
       line,
       conic,
       circle,
       ellipse,
       parabola,
       hyperbola,
       vector_or_direction,
       geometric_representation_context,
       geometric_representation_item,
       point,
       cartesian_point,
       point_on_curve,
       point_on_surface,
```

```
        point_in_volume,
        cylindrical_point,
        spherical_point,
        polar_point,
        direction,
        vector,
        placement,
        cartesian_transformation_operator,
        cartesian_transformation_operator_3d,
        cartesian_tranformation_operator_2d,
        surface,
        oriented_surface,
        elementary_surface,
        plane,
        conical_surface,
        cylindrical_surface,
        spherical_surface,
        volume,
        block_volume,
        spherical_volume,
        cylindrical_volume);

USE FROM group_schema                              -- ISO 10303-41
        (group,
        group_relationship);

USE FROM location_schema                    -- ISO 10303-41
        (location,
        location_relationship);

USE FROM management_resources_schema           -- ISO 10303-41
        (action_assignment,
        action_request_assignment,
        approval_assignment,
        attribute_classification_assignment,
        attribute_value_assignment,
        attribute_value_role,
        certification_assignment,
        classification_assignment,
        classification_role,
        contract_assignment,
        date_and_time_assignment,
        date_assignment,
        document_reference,
        document_usage_constraint_assignment,
        document_usage_role,
        effectivity_assignment,
        external_identification_assignment,
        event_occurrence_assignment,
        group_assignment,
        identification_assignment,
        identification_assignment_relationship,
        identification_role,
        location_assignment,
        location_representation_assignment,
        organization_assignment,
        organizational_project_assignment,
        organizational_project_role,
        person_and_organization_assignment,
```

```
       person_assignment,
       qualification_type_assignment,
       security_classification_assignment,
       time_interval_assignment);

USE FROM material_property_definition_schema      -- ISO 10303-45
   (characterized_material_property,
   material_property,
   property_definition_relationship,
   material_designation,
   material_designation_characterization);

USE FROM material_property_representation_schema
   (material_property_representation,
   data_environment,
   data_environment_relationship);

USE FROM mathematical_functions_schema            -- ISO 10303-50
   (nonnegative_integer,
   positive_integer,
   zero_or_one,
   one_or_two,
   maths_number,
   maths_real,
   maths_integer,
   maths_logical,
   maths_boolean,
   maths_string,
   maths_binary,
   maths_simple_atom,
   maths_atom,
   atom_based_tuple,
   atom_based_value,
   maths_tuple,
   maths_value,
   maths_expression,
   maths_function_select,
   input_selector,
   elementary_space_enumerators,
   ordering_type,
   lower_upper,
   symmetry_type,
   elementary_function_enumerators,
   open_closed,
   space_constraint_type,
   repackage_options,
   extension_options,
   maths_enum_atom,
   dotted_express_identifier,
   express_identifier,
   product_space,
   tuple_space,
   maths_space_or_function,
   real_interval,
   quantifier_expression,
   dependent_variable_definition,
   bound_variable_semantics,
   free_variable_semantics,
   complex_number_literal,
```

```
logical_literal,
binary_literal,
maths_enum_literal,
real_tuple_literal,
integer_tuple_literal,
atom_based_literal,
maths_tupel_literal,
maths_variable,
maths_real_variable,
maths_integer_variable,
maths_boolean_variable,
maths_string_variable,
function_application,
maths_space,
elementary_space,
finite_integer_interval,
integer_interval_from_min,
integer_interval_to_max,
finite_real_interval,
real_interval_from_min,
real_interval_to_max,
cartesian_complex_number_region,
polar_complex_number_region,
finite_space,
uniform_product_space,
listed_product_space,
extended_tuple_space,
function_space,
maths_function,
finite_function,
constant_function,
selector_function,
elementary_function,
restriction_function,
repackaging_function,
reindexed_array_function,
series_composed_function,
parallel_composed_function,
explicit_table_function,
listed_real_data,
listed_integer_data,
listed_logical_data,
listed_string_data,
listed_complex_number_data,
listed_data,
externally_listed_data,
linearized_table_function,
standard_table_function,
regular_table_function,
triangular_matrix,
strict_triangular_matrix,
symmetric_matrix,
symmetric_banded_matrix,
banded_matrix,
basic_sparse_matrix,
homogeneous_linear_function,
general_linear_function,
b_spline_basis,
b_spline_function,
```

```
    rationalize_function,
    partial_derivative_function,
    partial_derivative_expression,
    definite_integral_function,
    definite_integral_expression,
    abstracted_expression_function,
    expression_denoted_function,
    imported_point_function,
    imported_curve_function,
    imported_surface_function,
    imported_volume_function,
    application_defined_function,
    mathematical_description);

USE FROM measure_schema                              -- ISO 10303-41
    (absorbed_dose_measure,
    absorbed_dose_unit,
    absorbed_dose_measure_with_unit,
    acceleration_measure,
    acceleration_unit,
    acceleration_measure_with_unit,
    amount_of_substance_measure,
    amount_of_substance_unit,
    amount_of_substance_measure_with_unit,
    area_measure,
    area_unit,
    area_measure_with_unit,
    capacitance_measure,
    capacitance_unit,
    capacitance_measure_with_unit,
    celsius_temperature_measure,
    celsius_temperature_measure_with_unit,
    conductance_measure,
    conductance_unit,
    conductance_measure_with_unit,
    context_dependent_measure,
    context_dependent_unit,
    conversion_based_unit,
    count_measure,
    derived_unit,
    derived_unit_element,
    descriptive_measure,
    dimensional_exponents,
    dose_equivalent_measure,
    dose_equivalent_unit,
    dose_equivalent_measure_with_unit,
    electric_charge_measure,
    electric_charge_unit,
    electric_charge_measure_with_unit,
    electric_current_measure,
    electric_current_unit,
    electric_current_measure_with_unit,
    electric_potential_measure,
    electric_potential_unit,
    electric_potential_measure_with_unit,
    energy_measure,
    energy_unit,
    energy_measure_with_unit,
    force_measure,
```

```
force_unit,
force_measure_with_unit,
frequency_measure,
frequency_unit,
frequncy_measure_with_unit,
global_unit_assigned_context,
illuminance_measure,
illuminance_unit,
illuminance_measure_with_unit,
inductance_measure,
inductance_unit,
inductance_measure_with_unit,
length_measure,
length_unit,
length_measure_with_unit,
luminous_flux_measure,
luminous_flux_unit,
luminous_flux_measure_with_unit,
luminous_intensity_measure,
luminous_intensity_unit,
luminous_intensity_measure_with_unit,
magnetic_flux_density_measure,
magnetic_flux_density_unit,
magnetic_flux_density_measure_with_unit,
magnetic_flux_measure,
magnetic_flux_unit,
magnetic_flux_measure_with_unit,
mass_measure,
mass_unit,
mass_measure_with_unit,
measure_value,
measure_with_unit,
named_unit,
non_negative_length_measure,
numeric_measure,
parameter_value,
plane_angle_measure,
plane_angle_unit,
plane_angle_measure_with_unit,
positive_length_measure,
positive_plane_angle_measure,
positive_ratio_measure,
power_measure,
power_unit,
power_measure_with_unit,
pressure_measure,
pressure_unit,
pressure_measure_with_unit,
radioactivity_measure,
radioactivity_unit,
radioactivity_measure_with_unit,
ratio_measure,
ratio_unit,
ratio_measure_with_unit,
resistance_measure,
resistance_unit,
resistance_measure_with_unit,
si_prefix,
si_unit,
```

```
    si_unit_name,
    solid_angle_measure,
    solid_angle_unit,
    solid_angle_measure_with_unit,
    thermodynamic_temperature_measure,
    thermodynamic_temperature_unit,
    thermodynamic_temperature_measure_with_unit,
    time_measure,
    time_unit,
    time_measure_with_unit,
    unit,
    velocity_measure,
    vleocity_measure,
    velocity_measure_with_unit,
    volume_measure,
    volume_unit,
    volume_measure_with_unit);

USE FROM method_definition_schema              --ISO 10303-49
    (relationship_with_condition,
    process_or_process_relationship,
    action_method_with_associated_documents,
    action_method_with_associated_documents_constrained,
    process_or_process_relationship_effectivity,
    concurrent_action_method,
    serial_action_method,
    sequential_method);

USE FROM person_organization_schema                --ISO 10303-41
    (person_organization_select,
    address,
    organization,
    organization_relationship,
    organization_role,
    organizational_address,
    organizational_project,
    organizational_project_relationship,
    person,
    person_and_organization,
    person_and_organization_role,
    person_role,
    person_type,
    person_type_definition,
    person_type_definition_formation,
    person_type_definition_relationship,
    personal_address);

USE FROM process_property_schema                   --ISO 10303-49
    (characterized_action_definition,
    characterized_resource_definition,
    property_or_shape_select,
    action_property,
    product_definition_process,
    process_product_association,
    property_process,
    process_property_association,
    resource_property,
    action_resource_requirement,
    action_property_relationship,
```

```
    requirement_for_action_resource,
    resource_property_relationship,
    action_resource_requirement_relationship,
    resource_requirement_type,
    resource_requirement_type_relationship);

USE FROM process_property_representation_schema          --ISO 10303-49
    (action_property_representation,
    resource_property_representation);

USE FROM product_definition_schema                --ISO 10303-41
    (product,
    product_category,
    product_category_relationship,
    product_definition,
    product_defintion_relationship,
    product_definition_formation,
    product_definition_formation_relationship,
    product_definition_relationship,
    product_definition_with_associated_documents,
    product_relationship);

USE FROM product_property_definition_schema    -- ISO 10303-41
    (characterized_definition,
    characterized_product_definition,
    derived_property_select,
    shape_definition,
    characterized_object,
    product_definition_shape,
    property_definition,
    shape_aspect,
    shape_aspect_relationship);

USE FROM product_property_representation_schema   -- ISO 10303-41
    (represented_definition,
    context_dependent_shape_representation,
    item_identified_representation_usage,
    property_definition_representation,
    shape_definition_representation,
    shape_representation,
    shape_representation_relationship);

USE FROM qualifications_schema                -- ISO 10303-41
    (qualification_type,
    qualification_relationship);

USE FROM qualified_measure_schema             -- ISO 10303-45
    (value_qualifier,
    type_qualifier,
    precision_qualifier,
    uncertainty_qualifier,
    qualitative_uncertainty,
    standard_uncertainty,
    expanded_uncertainty,
    measure_representation_item,
    descriptive_representation_item,
    qualified_representation_item,
    measure_qualification);
```

```
USE FROM representation_schema                  -- ISO 10303-43
   (compound_item_definition,
   founded_item_select,
   list_representation_item,
   set_representation_item,
   value_representation_item,
   transformation,
   compound_representation_item,
   functionally_defined_transformation,
   global_uncertainty_assigned_context,
   item_defined_transformation,
   mapped_item,
   parametric_representation_context,
   representation,
   representation_context,
   representation_item,
   representation_item_relationship,
   representation_map,
   representation_relationship,
   representation_relationship_with_transformation,
   uncertainty_measure_with_unit,
   uncertainty_assigned_representation);

USE FROM shape_aspect_definition_schema           -- ISO 10303-47
   (datum,
   datum_feature,
   datum_target,
   datum_reference,
   referenced_modified_datum);

USE FROM shape_dimension_schema             -- ISO 10303-47
   (angle_relator,
   dimensional_characteristic,
   angular_location,
   angular_size,
   dimensional_characteristic_representation,
   dimensional_location,
   dimensional_location_with_path,
   dimensional_size,
   dimensional_size_with_path,
   shape_dimension_representation);

USE FROM shape_tolerance_schema                   --ISO 10303-47
   (tolerance_method_definition,
   shape_tolerance_select,
   dimension_related_tolerance_zone_element,
   geometric_tolerance,
   geometric_tolerance_relationship,
   geometric_tolerance_with_datum_reference,
   geometric_tolerance_with_defined_unit,
   modified_geometric_tolerance,
   projected_zone_definition,
   runout_zone_defintion,
   runout_zone_orientation,
   runout_zone_orientation_reference_direction,
   statistical_distribution_for_tolerance,
   tolerance_with_statistical_distribution,
   tolerance_zone,
   tolerance_zone_definition,
```

```
    tolerance_zone_form,
    limits_and_fits,
    plus_minus_tolerance,
    tolerance_value);

USE FROM state_type_schema                          -- ISO 10303-56
    (state_type,
    state_type_assignment,
    state_type_relationship,
    state_type_role);

USE FROM state_observed_schema                      -- ISO 10303-56
    (ascribable_state,
    ascribable_state_relationship,
    state_observed,
    state_observed_assignment,
    state_observed_relationship,
    state_observed_role);

USE FROM support_resource_schema                    -- ISO 10303-41
    (identifier,
    label,
    text);

REFERENCE FROM cutting_tools_schema ( item_correlation); -- ISO 13399-1

REFERENCE FROM automotive_design_schema (get_multi_language);  -- ISO 10303-214

TYPE action_item = SELECT
    (approval,
    certification,
    document,
    material_property,
    material_property_representation,
    product_definition,
    property_definition,
    property_definition_representation);
END_TYPE;

TYPE action_request_item = SELECT
    (action,
    approval,
    certification,
    document,
    executed_action,
    material_property,
    product,
    product_definition,
    product_definition_formation,
    property_definition,
    organizational_project,
    security_classification,
    security_classification_level);
END_TYPE;

TYPE approval_item = SELECT
    (material_property,
    product,
    product_definition,
```

```
    product_definition_formation,
    property_definition,
    representation,
    versioned_action_request);
END_TYPE;

TYPE attribute_language_item = SELECT
    (action,
    action_method,
    action_property,
    application_context,
    certification,
    document,
    descriptive_representation_item,
    material_designation,
    material_property,
    material_property_representation,
    product,
    product_definition,
    product_definition_formation,
    property_definition,
    qualification_type,
    representation);
END_TYPE;

TYPE attribute_type = SELECT
    (label,
    text);
END_TYPE;

TYPE certification_item = SELECT
    (action,
    action_method,
    material_property,
    organization,
    product,
    product_definition,
    product_definition_formation,
    person_and_organization,
    property_definition);
END_TYPE;

TYPE characterized_product_composition_value = SELECT
    (maths_value_with_unit,
    measure_with_unit);
END_TYPE;

TYPE contract_item = SELECT
    (action,
    material_property,
    organizational_project,
    person_organization_select,
    product,
    property_definition);
END_TYPE;

TYPE date_and_time_item = SELECT
    (action,
    event_occurrence,
```

```
    representation,
    versioned_action_request);
END_TYPE;

TYPE date_item = SELECT
    (action,
    approval,
    certification,
    contract,
    event_occurrence,
    product_definition_formation,
    representation,
    versioned_action_request);
END_TYPE;

TYPE document_item = SELECT
    (action,
    action_method,
    action_resource,
    action_resource_requirement,
    contract,
    geometric_tolerance,
    material_designation,
    material_property,
    product_definition,
    product_definition_formation,
    product_definition_process,
    property_definition,
    representation);
END_TYPE;

TYPE effectivity_item = SELECT
    (action,
    document,
    product_definition_formation);
END_TYPE;


TYPE event_occurred_item = SELECT
    (action);
END_TYPE;

TYPE external_identification_item = SELECT
    (document,
    product,
    product_definition,
    externally_defined_class,
    externally_defined_engineering_property);
END_TYPE;

TYPE groupable_item = SELECT
    (action,
    action_method,
    material_property,
    property_definition,
    product,
    product_definition);
END_TYPE;
```

```
TYPE identification_item = SELECT
   (certification,
   document,
   product,
   product_definition,
   organization,
   person_and_organization);
END_TYPE;

TYPE language_item = SELECT
   (action,
   action_method,
   action_property,
   application_context,
   certification,
   document,
   descriptive_representation_item,
   material_designation,
   material_property,
   material_property_representation,
   product,
   product_definition,
   product_definition_formation,
   property_definition,
   qualification_type,
   representation);
END_TYPE;

TYPE location_item = SELECT
   (action,
   event_occurrence,
   product,
   product_definition,
   product_definition_formation);
END_TYPE;

TYPE location_representation_item = SELECT
   (representation);
END_TYPE;

TYPE multi_language_attribute_item = SELECT
   (action,
   action_method,
   action_property,
   application_context,
   certification,
   document,
   descriptive_representation_item,
   material_designation,
   material_property,
   material_property_representation,
   product,
   product_definition,
   product_definition_formation,
   property_definition,
   qualification_type,
   representation);
END_TYPE;
```

```
TYPE organization_item = SELECT
    (action,
    approval,
    certification,
    document,
    material_designation,
    versioned_action_request);
END_TYPE;

TYPE organizational_project_item =  SELECT
    (action,
    action_method,
    document,
    product,
    material_property);
END_TYPE;

TYPE person_item = SELECT
    (action,
    document,
    versioned_action_request);
END_TYPE;

TYPE person_and_organization_item = SELECT
    (action,
    certification,
    product_definition_formation,
    versioned_action_request);
END_TYPE;

TYPE qualification_item = SELECT
    (person,
    person_and_organization,
    organization);
END_TYPE;

TYPE security_classified_item = SELECT
    (action,
    action_method,
    document,
    material_property,
    representation,
    representation_item);
END_TYPE;

TYPE state_item = SELECT
    (action,
    action_method,
    product,
    product_definition,
    product_definition_formation,
    property_definition,
    material_property,
    material_property_representation);
END_TYPE;

TYPE state_observed_item = SELECT
    (action,
    action_method,
```

```
   product,
   product_definition,
   product_definition_formation,
   property_definition,
   material_property,
   material_property_representation);
END_TYPE;


TYPE time_interval_item = SELECT
   (action,
   approval,
   effectivity,
   document,
   qualification);
END_TYPE;

ENTITY applied_action_assignment
       SUBTYPE OF (action_assignment);
       item : action_item;
END_ENTITY;

ENTITY applied_action_request_assignment
       SUBTYPE OF (action_request_assignment);
       item : action_request_item;
END_ENTITY;

ENTITY applied_approval_assignment
       SUBTYPE OF (approval_assignment);
       item : approval_item;
END_ENTITY;

ENTITY applied_certification_assignment
       SUBTYPE OF (certification_assignment);
       item : certification_item;
END_ENTITY;

ENTITY applied_contract_assignment
       SUBTYPE OF (contract_assignment);
       item : contract_item;
END_ENTITY;

ENTITY applied_date_assignment
       SUBTYPE OF (date_assignment);
       item : date_item;
END_ENTITY;

ENTITY applied_date_and_time_assignment
       SUBTYPE OF (date_and_time_assignment);
       item : date_and_time_item;
END_ENTITY;

ENTITY applied_document_reference
       SUBTYPE OF (document_reference);
       item : document_item;
END_ENTITY;

ENTITY applied_document_usage_contraint_assignment
       SUBTYPE OF (document_usage_constraint_assignment);
```

```
      item : document_item;
END_ENTITY;

ENTITY applied_effectivity_assignment
      SUBTYPE OF (effectivity_assignment);
      item : effectivity_item;
END_ENTITY;

ENTITY applied_event_occurrence_assignment
      SUBTYPE OF (event_occurrence_assignment);
      item : event_occurred_item;
END_ENTITY;

ENTITY applied_external_identification_assignment
      SUBTYPE OF (external_identification_assignment);
      items : SET[1:?] OF    external_identification_item;
WHERE
   WR1:   NOT (SELF.role.name = 'version') OR
      item_correlation(SELF.items,
      ['EXTERNALLY_DEFINED_CLASS',
      'EXTERNALLY_DEFINED_ENGINEERING_PROPERTY' ]);
END_ENTITY;

ENTITY applied_group_assignment
      SUBTYPE OF (group_assignment);
      items : SET[1:?] OF  groupable_item;
END_ENTITY;

ENTITY applied_identification_assignment
      SUBTYPE OF (identification_assignment);
      item : identification_item;
END_ENTITY;

ENTITY applied_location_assignment
      SUBTYPE OF (location_assignment);
      item : location_item;
END_ENTITY;

ENTITY applied_location_representation_assignment
      SUBTYPE OF (location_representation_assignment);
      item : location_representation_item;
END_ENTITY;

ENTITY applied_organization_assignment
      SUBTYPE OF (organization_assignment);
      item : organization_item;
END_ENTITY;

ENTITY applied_organizational_project_assignment
      SUBTYPE OF (organizational_project_assignment);
      item : SET [1:?] OF organizational_project_item;
END_ENTITY;

ENTITY applied_person_assignment
      SUBTYPE OF (person_assignment);
      item : person_item;
END_ENTITY;

ENTITY applied_person_and_organization_assignment
```

```
       SUBTYPE OF (person_and_organization_assignment);
       item : person_and_organization_item;
END_ENTITY;

ENTITY applied_qualification_assignment
       SUBTYPE OF (qualification_type_assignment);
       item : qualification_item;
END_ENTITY;

ENTITY applied_security_classification_assignment
       SUBTYPE OF (security_classification_assignment);
       item : SET [1:?] OF security_classified_item;
END_ENTITY;

ENTITY applied_state_type_assignment
       SUBTYPE OF (state_type_assignment);
       item : state_item;
END_ENTITY;

ENTITY applied_state_observed_assignment
       SUBTYPE OF (state_observed_assignment);
       item : state_observed_item;
END_ENTITY;

ENTITY applied_time_interval_assignment
       SUBTYPE OF (time_interval_assignment);
       item : time_interval_item;
END_ENTITY;

ENTITY attribute_language_assignment
    SUBTYPE OF(attribute_classification_assignment);
      items    : SET [1:?] OF attribute_language_item;
    DERIVE
      language : label :=
SELF\attribute_classification_assignment.assigned_class.name;
    WHERE
      WR1 :
            SELF\attribute_classification_assignment.role.name IN ['primary',
'translated' ];
      WR2 :
            'ENGINEERING_PROPERTIES_SCHEMA.' + 'LANGUAGE' IN
          TYPEOF(SELF\attribute_classification_assignment.assigned_class);
 END_ENTITY;

ENTITY externally_defined_action_property
       SUBTYPE OF (action_property, externally_defined_item);
END_ENTITY;

ENTITY externally_defined_class
       SUBTYPE OF (class, externally_defined_item);
END_ENTITY;

ENTITY externally_defined_engineering_property
       SUBTYPE OF (material_property, externally_defined_item);
END_ENTITY;

ENTITY language
    SUBTYPE OF(group);
    WHERE
```

```
      WR1 :
            (SIZEOF(QUERY (ca <* USEDIN(SELF, 'ENGINEERING_PROPERTIES_SCHEMA.' +
          'CLASSIFICATION_ASSIGNMENT.' + 'ASSIGNED_CLASS')|
       ('ENGINEERING_PROPERTIES_SCHEMA.' + 'LANGUAGE_ASSIGNMENT' IN TYPEOF(ca))))
> 0) OR
       (SIZEOF(QUERY (aca <* USEDIN(SELF, 'ENGINEERING_PROPERTIES_SCHEMA.' +
       'ATTRIBUTE_CLASSIFICATION_ASSIGNMENT.' + 'ASSIGNED_CLASS')|
       ('ENGINEERING_PROPERTIES_SCHEMA.' + 'ATTRIBUTE_LANGUAGE_ASSIGNMENT' IN
TYPEOF(aca)))) > 0);
  END_ENTITY;

ENTITY language_assignment
    SUBTYPE OF(classification_assignment);
      items : SET [1:?] OF language_item;
    WHERE
      WR1 :
            'ENGINEERING_PROPERTIES_SCHEMA.' + 'LANGUAGE' IN
TYPEOF(SELF.assigned_class);
      WR2 :
            SELF.role.name = 'language';
      WR3 :
            SIZEOF(SELF.items) = SIZEOF(QUERY (i <* SELF.items|
          ('ENGINEERING_PROPERTIES_SCHEMA.' + 'REPRESENTATION' IN TYPEOF(i)) AND
          (i\representation.name = 'document content')));
  END_ENTITY;

ENTITY multi_language_attribute_assignment
    SUBTYPE OF(attribute_value_assignment);
      items    : SET [1:?] OF multi_language_attribute_item;
    DERIVE
      language : label := get_multi_language(SELF);
    WHERE
      WR1 :
            SELF\attribute_value_assignment.role.name = 'alternate language';
      WR2 :
            (SIZEOF(USEDIN(SELF,
'ENGINEERING_PROPERTIES_SCHEMA.ATTRIBUTE_LANGUAGE_ASSIGNMENT.ITEMS')) = 1)
          AND (SIZEOF(QUERY (ala <* USEDIN(SELF,
'ENGINEERING_PROPERTIES_SCHEMA.' +
          'ATTRIBUTE_LANGUAGE_ASSIGNMENT.' + 'ITEMS')| (ala.attribute_name =
'attribute_value'))) = 1);
  END_ENTITY;

ENTITY maths_value_qualification;
   name                : label;
   description         : text;
   qualified_maths_value      : maths_value_with_unit;
   qualifiers             : SET [1:?] OF value_qualifier;
WHERE
   WR1: SIZEOF(QUERY(temp <* qualifiers |
            'QUALIFIED_MEASURE_SCHEMA.PRECISION_QUALIFIER'
            IN TYPEOF(temp))) < 2;
END_ENTITY;

ENTITY maths_value_representation_item
   SUBTYPE OF (representation_item, maths_value_with_unit);
END_ENTITY;

ENTITY maths_value_with_unit;
```

```
  value_component      : maths_value;
  unit_component          : unit;
END_ENTITY;

ENTITY product_as_individual
       ABSTRACT SUPERTYPE OF (ONEOF (product_as_planned,
       product_as_realised))
       SUBTYPE OF (product_definition_formation);
END_ENTITY;

ENTITY product_as_planned
       SUBTYPE OF (product_as_individual);
END_ENTITY;

ENTITY product_as_realised
       SUBTYPE OF (product_as_individual);
END_ENTITY;


ENTITY product_material_composition_relationship
SUBTYPE OF (product_definition_relationship);
  class                    : label;
  constituent_amount       : SET [1:?] OF
characterized_product_composition_value;
  composition_basis        : label;
  determination_method     : text;
END_ENTITY;



RULE dependent_instantiable_attribute_value_role FOR
    (attribute_value_role);
WHERE
  WR1:  SIZEOF ( QUERY (a <* attribute_value_role |
        NOT (SIZEOF (USEDIN (a,'')) > 0 ))) = 0;

END_RULE;

RULE dependent_instantiable_classification_role FOR
    (classification_role);
WHERE
  WR1:  SIZEOF ( QUERY (c <* classification_role |
        NOT (SIZEOF (USEDIN (c,'')) > 0 ))) = 0;

END_RULE;

RULE dependent_instantiable_identification_role FOR
    (identification_role);
WHERE
  WR1:  SIZEOF ( QUERY (i <* identification_role |
        NOT (SIZEOF (USEDIN (i,'')) > 0 ))) = 0;

END_RULE;

RULE plib_class_reference_requires_version FOR
    (externally_defined_class);
WHERE
 WR1: SIZEOF ( QUERY ( edc <* externally_defined_class | (
     'ENGINEERING_PROPERTIES_SCHEMA.'+ 'EXTERNAL_SOURCE' IN TYPEOF ( edc.
```

```
     source ) ) AND ( SIZEOF ( QUERY ( aei <* USEDIN ( edc ,

'ENGINEERING_PROPERTIES_SCHEMA.APPLIED_EXTERNAL_IDENTIFICATION_ASSIGNMENT.ITEMS'
) |
    aei. role.name ='version' ) ) <>1 ) ) ) =0;
 WR2: SIZEOF ( QUERY ( edc <* externally_defined_class | (
     'ENGINEERING_PROPERTIES_SCHEMA.'+ 'EXTERNAL_SOURCE' IN TYPEOF ( edc.
     source ) ) AND ( SIZEOF ( QUERY ( aei <* USEDIN ( edc ,

'ENGINEERING_PROPERTIES_SCHEMA.APPLIED_EXTERNAL_IDENTIFICATION_ASSIGNMENT.ITEMS'
     ) | aei. role.name ='version' ) ) >0 ) ) ) =0;
END_RULE;

RULE plib_property_reference_requires_name_scope FOR
    (externally_defined_engineering_property);
WHERE
 WR1: SIZEOF ( QUERY ( edep <*
     externally_defined_engineering_property | (
     'ENGINEERING_PROPERTIES_SCHEMA.'+ 'EXTERNAL_SOURCE' IN TYPEOF ( edep.
     source ) ) AND ( SIZEOF ( QUERY ( edir <* USEDIN ( edep ,
     'ENGINEERING_PROPERTIES_SCHEMA.'+'EXTERNALLY_DEFINED_ITEM_RELATIONSHIP.'+
     'RELATING_ITEM' ) | ( edir. name = 'name scope' ) AND (
     'ENGINEERING_PROPERTIES_SCHEMA.'+ 'EXTERNALLY_DEFINED_CLASS' IN TYPEOF
     ( edir. related_item ) ) AND ( 'ENGINEERING_PROPERTIES_SCHEMA.'+
     'EXTERNAL_SOURCE' IN TYPEOF ( edir. related_item.source ) ) )
     ) <>1 ) ) ) =0;
END_RULE;

RULE plib_property_reference_requires_version FOR
    (externally_defined_engineering_property);
WHERE
 WR1: SIZEOF ( QUERY ( edep <* externally_defined_engineering_property | (
     'ENGINEERING_PROPERTIES_SCHEMA.'+ 'EXTERNAL_SOURCE' IN TYPEOF ( edep.
     source ) ) AND ( SIZEOF ( QUERY ( edir <* USEDIN ( edep ,

'ENGINEERING_PROPERTIES_SCHEMA.APPLIED_EXTERNAL_IDENTIFICATION_ASSIGNMENT.ITEMS'
) |
    (edir. role.name ='version' ) ) )<>1 ) ) ) =0;
END_RULE;


END_SCHEMA; --- engineering_properties_schema
```

5.2.1 Engineering properties for product design and validation type definitions

5.2.1.1 action_item

An action_item type identifies those objects to which an action may be assigned through an applied_action_assignment.

EXPRESS specification:

```
  *)
  TYPE action_item = SELECT
    (approval,
    certification,
    document,
```

```
    material_property,
    material_property_representation,
    product_definition,
    property_defintion,
    property_definition_representation);
END_TYPE;
 (*
```

5.2.1.2 action_request_item

An action_request_item type identifies those objects to which a versioned_action_request may be assigned though an applied_action_request_assignment.

EXPRESS specification:

```
*)
TYPE action_request_item = SELECT
  (action,
  approval,
  certification,
  contract,
  document,
  executed_action,
  material_property,
  organizational_project
  product,
  product_definition,
  product_definition_formation,
  property_defintion,
  security_classification,
security_classification_level);
END_TYPE;
(*
```

5.2.1.3 approval_item

An approval_item type identifies those objects to which an approval may be assigned through an applied_approval_assignment.

EXPRESS specification:

```
*)
TYPE approval_item = SELECT
  (material_property,
  product,
  product_definition_formation,
  property_definition,
  representation,
  versioned_action_request);
END_TYPE;
(*
```

5.2.1.4 attribute_language_item

An attribute_language_item type identifies those objects to which a language may be assigned through the use of an attribute_language_assignment.

EXPRESS specification

```
*)
TYPE attribute_language_item = SELECT
 (action,
 action_method,
 application_context,
 certification,
 descriptive_representation_item,
 document,
 material_designation,
 material_property,
 material_property_representation,
 product,
 product_definition,
 product_definition_formation,
 property_definition,
 qualification_type,
 representation);
END_TYPE;
(*
```

5.2.1.5 certification_item

A certification_item type identifies those objects to which a certificate may be assigned though an applied_certification_assignment.

EXPRESS specification:

```
*)
TYPE certification_item = SELECT
    (action,
    action_method,
    organization,
    material_property,
    product,
    product_definition,
    product_definition_formation,
    person_and_organization,
    property_definition);
END_TYPE;
(*
```

5.2.1.6 characterized_product_composition_value

A characterized_product_composition_value type provides the mechanism by which a constituent amount of composition can be associated with either a qualified numerical value or a qualified mathematical expression.

EXPRESS specification:

```
*)

TYPE characterized_product_composition_value = SELECT
   (maths_value_with_unit,
   measure_with_unit);
END_TYPE;
(*
```

5.2.1.7 contract_item

A contract_item type identifies those objects to which a contract may be assigned though an applied_contract_assignment.

EXPRESS specification:

```
*)
TYPE contract_item = SELECT
  (action,
  organizational_project,
  material_property,
  product,
  property_definition);
END_TYPE;
(*
```

5.2.1.8 date_and_time_item

A date_and_time_item type identifies those objects to which a date_and_time may be assigned though an applied_date_and_time_assignment.

EXPRESS specification:

```
*)
TYPE date_and_time_item = SELECT
  (action,
  event_occurrence,
  process_plan_version,
  representation,
  versioned_action_request);
END_TYPE;
 (*
```

5.2.1.9 date_item

A date_item type identifies those objects to which a date may be assigned through an applied_date_assignment.

EXPRESS specification:

```
*)
TYPE date_item = SELECT
  (action,
  approval,
  certification,
```

```
    contract,
    event_occurrence,
    product_definition_formation,
    process_plan_version,
    representation,
    versioned_action_request);
  END_TYPE;
   (*
```

5.2.1.10 document_item

A document_item type identifies those objects to which a document may be associated through an applied_document_assignment.

EXPRESS specification:

```
  *)
  TYPE document_item = SELECT
    (action,
    action_method,
    action_resource,
    action_resource_requirement,
    contract,
    geometric_tolerance,
    material_designation,
    material_property,
    product_definition,
    product_definition_formation,
    product_definition_process,
    property_definition,
    representation);
  END_TYPE;
   (*
```

5.2.1.11 event_occurred_item

An event_occurred_item type identifes those objects to which an event_occurrence may be associated through an applied_event_occurred_assignment.

EXPRESS specification:

```
  *)
  TYPE event_occurred_item = SELECT
    (action,
    ascribable_state,
    state_type,
    state_observed);
  END_TYPE;
  (*
```

5.2.1.12 external_identification_item

The external_identification_item type identifies those objects to which an external_identification may be associated through an applied_external_identification_assignment.

EXPRESS specification:

```
*)
TYPE external_identification_item = SELECT
 (document,
 product,
 product_definition,
 externally_defined_class,
 externally_defined_material_property);
END_TYPE;
(*
```

5.2.1.13 effectivity_item

An effectivity_item type identifies those objects to which an effectivity can be assigned through an applied_effectivity_assignment.

EXPRESS specification:

```
*)
TYPE effectivity_item = SELECT
  (action,
  document,
  product_definition_formation);
END_TYPE;
(*
```

5.2.1.14 groupable_item

A groupable_item type is an extensible list of alternate data types that may be members of a group.

EXPRESS specification:

```
*)
TYPE groupable_item = SELECT
  (action,
  action_method,
  material_property,
  property_definition,
  product,
  product_definition);
END_TYPE;
(*
```

5.2.1.15 identification _item

An identification_item type identifies those objects to which an identifier may be assigned though an applie_identification_assignment.

EXPRESS specification:

```
*)
TYPE identification_item = SELECT
  (certification,
```

```
    document,
    product,
    product_definition,
    organization,
    person_and_organization);
  END_TYPE;
  (*
```

5.2.1.16 language_item

A language_item type identifies those objects to which a classification of a language may be assigned through the use of a language_assignment.

EXPRESS specification:

```
  *)
  TYPE language_item = SELECT
   (action,
   action_method,
   application_context,
   certification,
   descriptive_representation_item,
   document,
   material_designation,
   material_property,
   material_property_representation,
   product,
   product_definition,
   product_definition_formation,
   property_definition,
   qualification_type,
   representation);
  END_TYPE;
  (*
```

5.2.1.17 location_item

A location_item type identifies those objects to which a location may be assigned trhough an applied_location_assignment.

EXPRESS specification:

```
  *)
  TYPE location_item = SELECT
    (action,
    event_occurrence,
    product,
    product_definition,
    product_definition_formation);
  END_TYPE;
  (*
```

5.2.1.18 location_representation_item

A location_representation_item type identifies a representation with which a location is associated.

EXPRESS specification:

```
*)
TYPE location_representation_item = SELECT
  (representation);
END_TYPE;
(*
```

5.2.1.19 multi_language_attribute_item

A multi_language_attribute_item type specifies those objects to which a label in an alternate language for an existing attribute may be assigned through the use of a muilti_language_attribute_assignment.

EXPRESS specification:

```
*)
TYPE multi_language_attribute_item = SELECT
 (action,
 action_method,
 application_context,
 certification,
 descriptive_representation_item,
 document,
 material_designation,
 material_property,
 material_property_representation,
 product,
 product_definition,
 product_definition_formation,
 property_definition,
 qualification_type,
 representation);
END_TYPE;
(*
```

5.2.1.20 organization_item

An organization_item identifies those objects to which an organization may be associated through an applied_organisation_assignment.

EXPRESS specification:

```
*)
TYPE organization_item = SELECT
  (action,
  approval,
  certification,
  document,
  material_designation,
  process_plan_version,
```

```
    versioned_action_request);
  END_TYPE;
  (*
```

5.2.1.21 organizational_project_item

An organizational_project_item identifies those objects which an organizational_project may be associated though an applied_organizational_project_assignment.

EXPRESS specification:

```
  *)
  TYPE organizational_project_item = SELECT
    (action,
    action_method,
    document,
    product,
    material_property);
  END_TYPE;
  (*
```

5.2.1.22 person_item

A person_item identifies those objects to which a person may be assigned tough an applied_person_assignment.

EXPRESS specification:

```
  *)
  TYPE person_item = SELECT
    (action,
    document,
    process_plan_version,
    versioned_action_request);
  END_TYPE;
  (*
```

5.2.1.23 person_and_organization_item

A person_and_organization_item identifies those objects to which a person_and_organization may be assigned though an applied_person_and_organization_assignment.

EXPRESS specification:

```
  *)
  TYPE person_and_organization_item = SELECT
    (action,
    certification,
    product_definition_formation,
    process_plan_version,
    versioned_action_request);
  END_TYPE;
  (*
```

5.2.1.24 qualification_item

A qualification_item identifies those objects to which a qualification_type can be assigned though an applied_qualification_assignment.

EXPRESS specification:

```
*)
TYPE qualification_item = SELECT
  (person,
  person_and_organization,
  organization);
END_TYPE;
(*
```

5.2.1.25 security_classified_item

A security_classified_item identifies those objects to which a security_classification may be assigned through an applied_security_classified_assignment.

EXPRESS specification:

```
*)
TYPE security_classified_item = SELECT
  (action,
  action_method,
  document,
  material_property,
  representation,
  representation_item);
END_TYPE;
(*
```

5.2.1.26 state_item

A state_item identifies an those objects to which a state_type may be assigned through an applied_state_assignment.

EXPRESS specification:

```
*)
TYPE state_item = SELECT
 (action,
  action_method,
  product,
  product_definition,
 product_definition_formation,
 property_definition,
 material_property,
 material_property_representation,
 document);
END_TYPE;
```

```
(*
```

5.2.1.27 state_observed_item

A state_observed_item identifies those objects to which a state_observed may be assigned through an applied_state_observed_assignment.

EXPRESS specification:

```
*)
TYPE state_observed_item = SELECT
 (action,
  action_method,
  product,
  product_definition,
 product_definition_formation,
 property_definition,
 material_property,
 material_property_representation,
 document);
END_TYPE;
(*
```

5.2.1.28 time_interval_item

A time_interval_assigned_item identifies those objects to which a time_interval may be assigned though a applied_time_interval_assignment.

EXPRESS specification:

```
*)
TYPE time_interval_item = SELECT
  (action,
  approval,
  effectivity,
  document,
  qualification);
END_TYPE;
(*
```

5.2.2 Engineering properties for product design and validation entity definitions

5.2.2.1 applied_action_assignment

An applied_action_assignment enables an action to be associated with a document, product, product_definition, product_definition_formation or property_definition.

EXPRESS specification:

```
 *)
  ENTITY applied_action_assignment
    SUBTYPE OF (action_assignment);
    item : action_item;
```

```
  END_ENTITY;
(*
```

Attribute definitions:

item : document, product, product_definition, product_definition_formation or property_definition

5.2.2.2 applied_action_request_assignment

An applied_action_request_assignment enables a versioned_action_request to be associated with a product_definition, product_definition_formation or property_definition.

EXPRESS specification:

```
*)
  ENTITY applied_action_request_assignment
    SUBTYPE OF (action_request_assignment);
    item : action_request_item;
  END_ENTITY;
(*
```

Attribute definitions:

item : product_definition, product_definition_formation or property_definition

5.2.2.3 applied_approval_assignment

An applied_approval_assignment enables the association of an approval with document, material_property, product, product_definition_formation, process_plan_version, property_definition, representation or versioned_action_request.

EXPRESS specification:

```
*)
  ENTITY applied_approval_assignment
    SUBTYPE OF (approval_assignment);
    item : approval_item;
  END_ENTITY;
(*
```

Attribute definitions:

item : document, material_property, product, product_definition, product_definition_formation process_plan_version, property_definition, representation, or versioned_action_request

5.2.2.4 applied_certification_assignment

An applied_certification_assignment enables the association of a certification with an action, action_method, organization, product, product_definition, product_definition_formation, person_and_organization, or property_definition.

EXPRESS specification:

```
*)
  ENTITY applied_certification_assignment
    SUBTYPE OF (certification_assignment);
    item : certification_item;
  END_ENTITY;
(*
```

Attribute definitions:

item : action, action_method, organization, product, product_definition, product_definition_formation, person_and_organization or property_definition

5.2.2.5 applied_contract_assignment

An applied_contract_assignment enables the association of a contract with an action, organizational_project, product or property_definition.

EXPRESS specification:

```
*)
  ENTITY applied_contract_assignment
    SUBTYPE OF (contract_assignment);
    item : contract_item;
  END_ENTITY;
(*
```

Attribute definitions:

item : action, product, or property_definition

5.2.2.6 applied_date_assignment

An applied_date_assignment enables the association of a with a contract, process_plan_version, representation or versioned_action_request.

EXPRESS specification:

```
*)
  ENTITY applied_date_assignment
    SUBTYPE OF (date_assignment);
    item : date_item;
  END_ENTITY;
(*
```

Attribute definitions:

item :contract, process_plan_version, representation or versioned_action_request

5.2.2.7 applied_date_and_time_assignment

An applied_date_assignment enables the association of a date_and_time with an action, event_occurrence, representation or versioned_action_request.

EXPRESS specification:

```
*)
  ENTITY applied_date_and_time_assignment
    SUBTYPE OF (date_and_time_assignment);
    item : date_and_time_item;
  END_ENTITY;
(*
```

Attribute definitions:

item : action, event_occurrence, representation or versioned_action_request

5.2.2.8 applied_document_reference

An applied_document_reference enables the association of a document with an action, action_method, action_resource, action_resource_requirement, contract, geometric_tolerance, material_designation, product_definition, product_definition_formation, product_definition_process, property_definition, or representation.

EXPRESS specification:

```
*)
  ENTITY applied_document_reference
    SUBTYPE OF (document_reference);
    item : document_item;
  END_ENTITY;
(*
```

Attribute definitions:

item : action, action_method, action_resource, action_resource_requirement, contract, geometric_tolerance, material_designation, product_definition, product_definition_formation, product_definition_process, property_definition, or representation

5.2.2.9 applied_document_usage_constraint_assignment

An applied_document_usage_constraint_assignment enables the association of part of a document with an action, action_method, action_resource, action_resource_requirement, contract, geometric_tolerance,

material_designation, product_definition, product_definition_formation, product_definition_process, property_definition, or representation.

EXPRESS specification:

```
*)
  ENTITY applied_document_usage_contraint_assignment
    SUBTYPE OF (document_usage_constraint_assignment);
    item : document_item;
  END_ENTITY;
(*
```

Attribute definitions:

item : action,action_method, action_resource, action_resource_requirement, contract, geometric_tolerance, material_designation, product_definition, product_definition_formation, product_definition_process, property_definition, or representation

5.2.2.10 applied_effectivity_assignment

An applied_effectivity_assignment enables the association of an effectivity with an action, document, or product_definition_formation.

EXPRESS specification:

```
*)
  ENTITY applied_effectivity_assignment
    SUBTYPE OF (effectivity_assignment);
    item : effectivity_item;
  END_ENTITY;
(*
```

Attribute definitions:

item : action, document or product_definition_formation

5.2.2.11 applied_event_occurrence_assignment

An applied_event_occurrence_assignment enables the association of an event_occurrence with an action, ascribable_state, state or state_observed.

EXPRESS specification:

```
*)
  ENTITY applied_event_occurrence_assignment
    SUBTYPE OF (event_occurrence_assignment);
    item : event_occurred_item;
  END_ENTITY;
(*
```

Attribute definitions:

item : action, ascribable_state, state or state_observed

5.2.2.12 applied_external_identification_assignment

An applied_external_identification_assignment is a type of external_identification_assignment that assigns an external_identification and an external_source to a set of external_identification_item.

EXPRESS specification:

```
*)
  ENTITY applied_external_identification_assignment
    SUBTYPE OF (external_identification_assignment);
    items : SET[1:?] OF   external_identification_item;
  WHERE
    WR1:  NOT (SELF.role.name = 'version') OR
          item_correlation(SELF.items,
          ['EXTERNALLY_DEFINED_CLASS',
          'EXTERNALLY_DEFINED_ENGINEERING_PROPERTY']);
  END_ENTITY;
(*
```

Attribute definitions:

items : the set of external_identification_item to which external identification is assigned

5.2.2.13 applied_group_assignment

An applied_group_assignment is a type of group_assignment that associates a group with one or more instances of groupable_item.

EXPRESS specification:

```
*)
  ENTITY applied_group_assignment
    SUBTYPE OF (group_assignment);
    items : SET[1:?] OF groupable_item;
  END_ENTITY;
(*
```

Attribute definitions:

items : the set of one or more instances that are associated with the group referred to by the inherited attribute assigned_group

5.2.2.14 applied_identification_assignment

An applied_identification_assignment enables the association of an identification_item with a certificate.

EXPRESS specification:

```
*)
  ENTITY applied_identification_assignment
    SUBTYPE OF (identification_assignment);
    item : identification_item;
  END_ENTITY;
(*
```

Attribute definitions:

item : certification

5.2.2.15 applied_location_assignment

An applied_location_assignment enables the association of a location with an action, document_version or event_occurrence.

EXPRESS specification:

```
*)
  ENTITY applied_location_assignment
    SUBTYPE OF (location_assignment);
    item : location_item;
  END_ENTITY;
(*
```

Attribute definitions:

item : action, document_version or event_occurrence

5.2.2.16 applied_location_representation_assignment

An applied_location_representation_assignment enables the association of a location with its representation.

EXPRESS specification:

```
*)
  ENTITY applied_location_representation_assignment
    SUBTYPE OF (location_representation_assignment);
    item : location_representation_item;
  END_ENTITY;
(*
```

Attribute definitions:

item : representation

5.2.2.17 applied_organization_assignment

An applied_organization_assignment enables the association of an organisation with an action, approval, certification, document, process_plan_version, or versioned_action_request.

EXPRESS specification:

```
*)
  ENTITY applied_organization_assignment
    SUBTYPE OF (organization_assignment);
    item : organization_item;
  END_ENTITY;
(*
```

Attribute definitions:

item : action, approval, certification, document, process_plan_version, or versioned_action_request

5.2.2.18 applied_organizational_project_assignment

An applied_organizational_project_assignment enables the association of an organizational_project with one or more items of product data.

EXPRESS specification:

```
*)
  ENTITY applied_organizational_project_assignment
    SUBTYPE OF (organizational_project_assignment);
    item : SET [1:?] OF organizational_project_item;
  END_ENTITY;
(*
```

Attribute definitions:

item : one or more items of product data to which an organization_project can be assigned.

5.2.2.19 applied_person_assignment

An applied_person_assignment enables the association of a person with an action, approval, document, process_plan_version or versioned_action_request.

EXPRESS specification:

```
*)
  ENTITY applied_person_assignment
    SUBTYPE OF (person_assignment);
    item : person_item;
  END_ENTITY;
(*
```

Attribute definitions:

item : action, document, process_plan_version, or versioned_action_request

5.2.2.20 applied_person_and_organization_assignment

An applied_person_and_organization_assignment enables the association of a person_and_organization with an item of product data.

EXPRESS specification:

```
*)
  ENTITY applied_person_and_organization_assignment
    SUBTYPE OF (person_and_organization_assignment);
    item : person_and_organization_item;
  END_ENTITY;
(*
```

Attribute definitions:

item: action, document, process_plan_version, or versioned_action_request

5.2.2.21 applied_qualification_assignment

An applied_qualification_assignment enables the association of a qualification_type with a person, person_and_organization or organization.

EXPRESS specification:

```
*)
  ENTITY applied_qualification_assignment
    SUBTYPE OF (qualification_type_assignment);
    item : qualification_item;
  END_ENTITY;
(*
```

Attribute definitions:

item : person, person_and_organization or organization

5.2.2.22 applied_security_classification_assignment

An applied_security_classification_assignment is a type of security_classification_assignment that enables the association of a security_classification with one or more items of product data.

EXPRESS specification:

```
*)
ENTITY applied_security_classification_assignment
    SUBTYPE OF (security_classification_assignment);
    item : SET [1:?] OF security_classified_item;
  END_ENTITY;
(*
```

Attribute definitions:

item : set of product data to which the security classification is assigned.

5.2.2.23 applied_state_type_assignment

An applied_state_type_assignment enables the association of a state_type with an action, action_method or product.

EXPRESS specification:

```
*)
ENTITY applied_state_type_assignment
    SUBTYPE OF (state_type_assignment);
    item : state_item;
  END_ENTITY;
(*
```

Attribute definitions:

item : action, action_method or product

5.2.2.24 applied_state_observed_assignment

An applied_state_observed_assignment enables the association of a state_observed with an action, action_method or product.

EXPRESS specification:

```
*)
ENTITY applied_state_observed_assignment
    SUBTYPE OF (state_observed_assignment);
    item : state_observed_item;
  END_ENTITY;
(*
```

Attribute definitions:

item : action, action_method or product

5.2.2.25 applied_time_interval_assignment

An applied_time_interval_assignment enables the association of a time_interval with an action, approval, effectivity, document or qualification.

EXPRESS specification:

```
*)
ENTITY applied_time_interval_assignment
  SUBTYPE OF (time_interval_assignment);
  item : time_interval_item;
END_ENTITY;
(*
```

Attribute definitions:

item : action, approval, effectivity, document or qualification

5.2.2.26 attribute_language_assignment

An attribute_language_assignment is a type of attribute_classification_assignment. It associates the specification of a language used for a string valued attribute to the entity instance that has this attribute. The attribute value is further classified as either being specified in the original language or as a translation thereof.

EXPRESS specification

```
*)
ENTITY attribute_language_assignment
    SUBTYPE OF(attribute_classification_assignment);
      items    : SET [1:?] OF attribute_language_item;
    DERIVE
      language : label :=
  SELF\attribute_classification_assignment.assigned_class.name;
    WHERE
      WR1 :
            SELF\attribute_classification_assignment.role.name IN
  ['primary', 'translated' ];
      WR2 :
            'ENGINEERING_PROPERTIES_SCHEMA.' + 'LANGUAGE' IN
  TYPEOF(SELF\attribute_classification_assignment.assigned_class);
  END_ENTITY;
(*
```

Attribute definitions:

items: the set of objects to which the language classification is assigned.

language: the language specifies the name of the assigned_class that provides an identification for the language in which an attribute value is specified.

Formal Propositions:

WR1: The language used to specify the string valued attributes in the set of items shall be marked either as the 'primary' language or as 'translated'.

WR2: The instance referenced as assigned_class shall be of type language.

5.2.2.27 externally_defined_action_property

An externally_defined_action_property is a type of both an action_property and an externally_defined_item that enables a reference to be made to an item in a classification that is contained within an external_source**.**

NOTE This entity could be used to reference the properties of a testing process in a dictionary of testing processes and their properties that conforms to ISO 13584.

EXPRESS specification:

```
*)
  ENTITY externally_defined_action_property
    SUBTYPE OF (action_property, externally_defined_item);
  END_ENTITY;
(*
```

5.2.2.28 externally_defined_class

An externally_defined_class is a type of both a class and an externally_defined_item that enables a reference to be made to an item in a classification that is contained within an external_source**.**

NOTE This entity could be used to reference a testing process in a dictionary of testing processes and their properties that conforms to ISO 13584.

EXPRESS specification:

```
*)
  ENTITY externally_defined_class
    SUBTYPE OF (class, externally_defined_item);
  END_ENTITY;
(*
```

5.2.2.29 externally_defined_engineering_property

An externally_defined_engineering_property is a type of both a material_property and an externally_defined_item that enables a reference to be made to property of a classification that is contained within an external_source**.**

NOTE This entity could be used to reference an engineering property in a dictionary of testing processes and their properties that conforms to ISO 13584.

EXPRESS specification:

```
*)
  ENTITY externally_defined_engineering_property
    SUBTYPE OF (material_property, externally_defined_item);
  END_ENTITY;
(*
```

5.2.2.30 language

A language is a type of group that is a specification of the natural language in which an information object is given. The group shall specify in its group.name a language code according to ISO 639-2 and, if present, in its group.description a county code conforming to ISO 3166-1.

EXPRESS specification

```
*)
  ENTITY language
      SUBTYPE OF(group);
      WHERE
        WR1 :
```

```
          (SIZEOF(QUERY (ca <* USEDIN(SELF,
  'ENGINEERING_PROPERTIES_SCHEMA.' + 'CLASSIFICATION_ASSIGNMENT.' +
  'ASSIGNED_CLASS')| ('ENGINEERING_PROPERTIES_SCHEMA.' +
  'LANGUAGE_ASSIGNMENT' IN TYPEOF(ca)))) > 0) OR (SIZEOF(QUERY (aca <*
  USEDIN(SELF, 'ENGINEERING_PROPERTIES_SCHEMA.' +
  'ATTRIBUTE_CLASSIFICATION_ASSIGNMENT.' + 'ASSIGNED_CLASS')|
  ('ENGINEERING_PROPERTIES_SCHEMA.' + 'ATTRIBUTE_LANGUAGE_ASSIGNMENT' IN
  TYPEOF(aca)))) > 0);
  END_ENTITY;
(*
```

Formal Propositions:

WR1: An instance of type language shall be referenced only by either the attribute assigned_class of a classification_assignment that is also a language_assignment, or by the attribute assigned_class of a attribute_classification_assignment that is also a attribute_language_assignment

Informal Propositions:

IP1: The **name** shall specify a language code according to ISO 639-2.

IP2: If present, the **description** shall specify a coutry code according to to ISO 3166-1.

5.2.2.31 language_assignment

A language_assignment is a type of classification_assignment. It specifes a language for the items it is assigned to. This language shall be specified by the language that is referenced in the role of assigned_class.

EXPRESS specification

```
*)
ENTITY language_assignment
    SUBTYPE OF(classification_assignment);
      items : SET [1:?] OF language_item;
    WHERE
      WR1 :
            'ENGINEERING_PROPERTIES_SCHEMA.' + 'LANGUAGE' IN
  TYPEOF(SELF.assigned_class);
      WR2 :
            SELF.role.name = 'language';
      WR3 :
            SIZEOF(SELF.items) = SIZEOF(QUERY (i <* SELF.items|
  ('ENGINEERING_PROPERTIES_SCHEMA.' + 'REPRESENTATION' IN TYPEOF(i)) AND
  (i\representation.name = 'document content')));
  END_ENTITY;
*)
```

Attribute definitions:

items: the set of items to which a language is assigned.

Formal Propositions:

WR1: The instance referenced as assigned_class shall be of type language.

WR2: The name of the classification_role referenced as role shall be 'language'.

WR3: The representations in the set of items shall be of name 'document content'.

5.2.2.32 multi_language_attribute_assignment

A multi_language_attribute_assignment is a type of attribute_value_assignment. It specifes an alternative value for the string valued attribute of an existing entity and the name of that attribute.

EXPRESS specification

```
*)
  ENTITY multi_language_attribute_assignment
      SUBTYPE OF(attribute_value_assignment);
        items    : SET [1:?] OF multi_language_attribute_item;
      DERIVE
        language : label := get_multi_language(SELF);
      WHERE
        WR1 :
              SELF\attribute_value_assignment.role.name = 'alternate
    language';
       WR2 :
              (SIZEOF(USEDIN(SELF,
    'ENGINEERING_PROPERTIES_SCHEMA.ATTRIBUTE_LANGUAGE_ASSIGNMENT.ITEMS')) =
    1) AND (SIZEOF(QUERY (ala <* USEDIN(SELF,
    'ENGINEERING_PROPERTIES_SCHEMA.' + 'ATTRIBUTE_LANGUAGE_ASSIGNMENT.' +
    'ITEMS')| (ala.attribute_name = 'attribute_value'))) = 1);

    END_ENTITY;
(*
```

Attribute definitions:

items: the set of items for which an alternative string value is specified.

language: The language specifies the attribute_language_assignment that refers as items to this object and provides the information concerning the language, in which an attribute value is specified.

Formal Propositions:

WR1: The attribute_value_role specified by the multi_language_attribute_assignment shall have a name of 'alternate language'.

WR2: The multi_language_attribute_assignment shall be used exactly one time in role of items by an attribute_language_assignment. This attribute_language_assignment shall have an attribute_name of 'attribute_value'.

Informal Propositions:

IP1: The attribute_name shall be the name of the attribute for which an alternative value is specified.

5.2.2.33 maths_value_qualification

A maths_value_qualification associates one or more qualifiers with a maths_value_with_unit.

EXPRESS specification:

*)

```
  ENTITY maths_value_qualification;
    name                 : label;
    description             : text;
    qualified_maths_value   : maths_value_with_unit;
    qualifiers           : SET [1:?] OF value_qualifier;
  WHERE
    WR1: SIZEOF(QUERY(temp <* qualifiers |
               'QUALIFIED_MEASURE_SCHEMA.PRECISION_QUALIFIER'
               IN TYPEOF(temp))) < 2;
  END_ENTITY;
(*
```

Attribute definitions:

name: the word or group of words by which the maths_value_qualification is referred to.

description: a narrative description of the maths_value _qualification.

qualified_measure: the maths_value_with_unit that is to be qualified.

qualifiers: the qualifiers of the maths_value _with_unit.

Formal propositions:

WR1: at most one of the elements of the qualifiers attribute can be a precision_qualifier.

5.2.2.34 maths_value_representation_item

A maths_value_representation_item is a type of representation_item where a particular mathematical function is represented.

EXPRESS specification:

*)

```
  ENTITY maths_value_representation_item
    SUBTYPE OF (representation_item, maths_value_with_unit);
  END_ENTITY;
(*
```

5.2.2.35 maths_value_with_unit

A maths_value_with_unit enables a quantitative value to be specified by mathematical function and to have the unit defined.

EXPRESS specification:

```
  *)
  ENTITY maths_value_with_unit;
    value_component          : maths_value;
```

```
    unit_component          : unit;
  END_ENTITY;
  (*
```

Attribute definitions:

value_component: the mathematical expression that represents the magnitude of the value

unit_component: the unit quantity that the expression multiplies.

5.2.2.36 product_as_individual

A product_as_individual is a type **of product_definition_formation** that identifies an individual artefact that has been made (see product_as_realised 5.2.2.31) or an individual artefact that has yet to be made (see product_as_planned 5.2.2.30).

NOTE 1 Where physical products are being represented, the product_as_individual represents the physical or planned physical realisation of a product.

NOTE 2 It is likely, but not essential, that the artefact was or will be made from a version of a product design which will be represented by product_defintion_formation.

NOTE 3 Many physical physical products may be produced from a given design. A single instance of product_definition_formation may lead to many product_as_individual.

EXPRESS specification:

```
   *)
    ENTITY product_as_individual
      ABSTRACT SUPERTYPE OF (ONEOF (product_as_planned,
                                    product_as_realised))
      SUBTYPE OF (product_definition_formation);
    END_ENTITY;
  (*
```

5.2.2.37 product_as_planned

A product_as_planned is a type of product_as_individual (see 5.2.2.29) that identifies an individual artefact that has yet to be made.

NOTE 1 It may be planned to make the artefact from a version of a product design (product_definition_formation). If this is the case then the relationship between the artefact (product_as_planned) and the design (product_defintion_formation) is represented by product_design_to_individual.

NOTE 2 If the planned artefact (product_as_planned) is subsequently made into an actual artefact (product_as_realised) then they are related by the relationship product_planned_to_realised.

EXPRESS specification:

```
   *)
    ENTITY product_as_planned
      SUBTYPE OF (product_as_individual);
    END_ENTITY;
```

```
(*
```

5.2.2.38 product_as_realised

A product_as_realised is a type of product_as_individual (see 5.2.2.29) that identifies an individual artefact that has been made and whose properties can only be known by observation or by derivation from observations.

NOTE 1 Where physical products are being represented, the product_as_realised represents something that can be touched.

NOTE 2 If the artefact has been made from a version of a product design (product_definition_formation), then the relationship between the artefact (product_as_realised) and the design (product_definition_formation) is represented by product_design_to_individual.

NOTE 3 If the artifact has been planned and represented by product_as_planned, then the relationship between the actual artefact (product_as_realised) and the planned artifact (product_as_planned) is represented by product_planned_to_realised.

EXPRESS specification:

```
*)
  ENTITY product_as_realised
    SUBTYPE OF (product_as_individual);
  END_ENTITY;
(*
```

5.2.2.39 product_material_composition_relationship

The product_material_composition_relationship associates a material constituent with a product. The product participates in a product_definition_relationship as a relating_product_definition. The constituent participates in a product_definition_relationship as a related_product_definition.

NOTE The spatial location and orientation of the material constituents in the product are specified by their product_definition_shape. By this means the material structure of the product is described.

EXPRESS specification:

```
*)
ENTITY product_material_composition_relationship
SUBTYPE OF (product_definition_relationship);
  class                 : label;
  constituent_amount    : SET [1:?] OF
   characterized_product_composition_value;
  composition_basis     : label;
  determination_method : text;
END_ENTITY;
(*
```

Attribute definitions:

class : the name or identifier for the kind of relationship between a constituent and a product.

EXAMPLE Possible values of a class include: 'mixture', 'chemically bonded', and 'alloyed'.

constituent amount: the quantity of the material constituent in a product and the units in which the quantity is expressed.

NOTE Examples of elements of the set could be: minimum value, maximum value, typical value. The description of the values as maximum, minimum or typical may be achieved by using the constructs in the qualified_measure_schema.

composition basis: the basis on which a product is decomposed into its constituents.

EXAMPLE Expected values for composition basis include: 'volume', 'weight', 'moles', and 'atoms'.

determination method: a description of the procedure by which the constituent amount is determined.

5.2.3 Engineering properties for product design and validation rule definitions

5.2.3.1 dependent_instantiable_attribute_value_role

The dependent_instantiable_attribute_value_role rule specifies that each instance of attribute_value_role is dependent on the usage to define another entity.

EXPRESS specification

```
*)
RULE dependent_instantiable_attribute_value_role FOR
    (attribute_value_role);
WHERE
  WR1:  SIZEOF ( QUERY (a <* attribute_value_role |
        NOT (SIZEOF (USEDIN (a,'')) > 0 ))) = 0;

END_RULE;
(*
```

Formal Propositions:

WR1: For each instance of attribute_value_role, there shall be a reference to the attribute_value_role instance from an attribute of another entity.

5.2.3.2 dependent_instantiable_classification_role

The dependent_instantiable_classification_role rule specifies that each instance of classification_role is dependent on the usage to define another entity.

EXPRESS specification

```
*)
RULE dependent_instantiable_classification_role FOR
    (classification_role);
WHERE
  WR1:  SIZEOF ( QUERY (c <* classification_role |
        NOT (SIZEOF (USEDIN (c,'')) > 0 ))) = 0;

END_RULE;
(*
```

Formal Propositions:

WR1: For each instance of classification_role, there shall be a reference to the classification_role instance from an attribute of another entity.

5.2.3.3 dependent_instantiable_identification_role

The dependent_instantiable_identification_role rule specifies that each instance of identification_role is dependent on the usage to define another entity.

EXPRESS specification

```
*)
RULE dependent_instantiable_identification_role FOR
    (identification_role);
WHERE
  WR1:  SIZEOF ( QUERY (i <* identification_role |
        NOT (SIZEOF (USEDIN (i,'')) > 0 ))) = 0;

END_RULE;
(*
```

Formal Propositions:

WR1: For each instance of identification_role, there shall be a reference to the identification_role instance from an attribute of another entity.

5.2.3.4 plib_class_reference_requires_version

The plib_class_reference_requires_version rule specifies that each instance of externally_defined_class that has as its source a known_source is contained in the set of items of exactly one applied_external_identification_assignment which references as its role an identification_role with a name of 'version'. This rule enforces the requirement for every plib_class_reference to have exactly one version.

EXPRESS specification:

```
*)
RULE plib_class_reference_requires_version FOR
    (externally_defined_class);
WHERE
 WR1: SIZEOF ( QUERY ( edc <* externally_defined_class | (
     'ENGINEERING_PROPERTY_SCHEMA.'+ 'EXTERNAL_SOURCE' IN TYPEOF ( edc.
     source ) ) AND ( SIZEOF ( QUERY ( aei <* USEDIN ( edc ,
     'ENGINEERING_PROPERTY_SCHEMA.APPLIED_EXTERNAL_IDENTIFICATION_ASSIGNMENT.
ITEMS' ) | aei. role.name ='version' ) ) <>1 ) ) ) =0;
 WR2: SIZEOF ( QUERY ( edc <* externally_defined_class | (
     'ENGINEERING_PROPERTY_SCHEMA.'+ 'EXTERNAL_SOURCE' IN TYPEOF ( edc.
     source ) ) AND ( SIZEOF ( QUERY ( aei <* USEDIN ( edc ,
     'ENGINEERING_PROPERTY_SCHEMA.APPLIED_IDENTIFICATION_ASSIGNMENT.ITEMS'
     ) | aei. role.name ='version' ) ) >0 ) ) ) =0;
END_RULE;
(*
```

Argument definitions:

externally_defined_class : the set of all instances of externally_defined_class.

Formal propositions:

WR1: Each instance of externally_defined_class that has as its source a known_source is contained in the set of items of exactly one applied_external_identification_assignment which references as its role an identification_role with a name of 'version'.

WR2: An instance of externally_defined_class that has as its source a known_source may not be contained in the set of items of an applied_identification_assignment which references as its role an identification_role with a name of 'version'.

5.2.3.5 plib_property_reference_requires_name_scope

The plib_property_reference_requires_name_scope rule specifies that each instance of externally_defined_engineering_property that has as its source a known_source is referenced as the relating_item by exactly one instance of externally_defined_item_relationship which has a name of 'name scope' and which references as the related_item an externally_defined_class that has as its source a known_source. This rule enforces the requirement for every plib_property_reference to have a name_scope.

EXPRESS specification:

```
*)
RULE plib_property_reference_requires_name_scope FOR
    (externally_defined_engineering_property);
WHERE
 WR1: SIZEOF ( QUERY ( edep <*
     externally_defined_engineering_property | (
     'ENGINEERING_PROPERTY_SCHEMA.'+ 'EXTERNAL_SOURCE' IN TYPEOF ( edep.
     source ) ) AND ( SIZEOF ( QUERY ( edir <* USEDIN ( edep ,
     'ENGINEERING_PROPERTY_SCHEMA.'+'EXTERNALLY_DEFINED_ITEM_RELATIONSHIP.'+
     'RELATING_ITEM' ) | ( edir. name = 'name scope' ) AND (
     'ENGINEERING_PROPERTY_SCHEMA.'+ 'EXTERNALLY_DEFINED_CLASS' IN TYPEOF
     ( edir. related_item ) ) AND ( 'ENGINEERING_PROPERTY_SCHEMA.'+
     'EXTERNAL_SOURCE' IN TYPEOF ( edir. related_item.source ) ) )
     ) <>1 ) ) ) =0;
END_RULE;
(*
```

Argument definitions:

externally_defined_engineering_property: the set of all instances of externally_defined_engineering_property.

Formal propositions:

WR1: Each instance of externally_defined_engineering_property that has as its source a known_source is referenced as the relating_item by exactly one instance of externally_defined_item_relationship which has a name of 'name scope' and which references as the related_item an externally_defined_class that has as its source a known_source.

5.2.3.6 plib_property_reference_requires_version

The plib_property_reference_requires_version rule specifies that each instance of externally_defined_engineering_property that has as its source a known_source is contained in the set of items of exactly one applied_external_identification_assignment which references as its role an identification_role with a name of 'version'. This rule enforces the requirement for every plib_property_reference to have a version.

EXPRESS specification:

```
*)
RULE plib_property_reference_requires_version FOR
    (externally_defined_engineering_property);
WHERE
 WR1: SIZEOF ( QUERY ( edep <*
     externally_defined_engineering_property | (
     'ENGINEERING_PROPERTY_SCHEMA.'+ 'EXTERNAL_SOURCE' IN TYPEOF ( edep.
     source ) ) AND ( SIZEOF ( QUERY ( edir <* USEDIN ( edep ,
  'ENGINEERING_PROPERTY_SCHEMA.'+'APPLIED_EXTERNAL_IDENTIFICATION_ASSIGNMENT.'
 +'ITEMS' ) | ( edir. role.name ='version' ) ) ) <>1 ) ) )
     =0;
END_RULE;
(*
```

Argument definitions:

externally_defined_engineering_property : the set of all instances of externally_defined_engineering_property.

Formal propositions:

WR1: Each instance of externally_defined_engineering_property that has as its source a known_source is contained in the set of items of exactly one applied_external_identification_assignment which references as its role an identification_role with a name of 'version'.

*)

END_SCHEMA; -- engineering_properties_schema

6 Conformance requirements

Conformance of this part of ISO 10303 includes satisfying the requirements started in this part, the requirements of the implementation method(s) supported and the relevant requirements of the normative references.

An implementation shall support at least one of the following implementation methods:

— ISO 10303-21 Industrial automation systems and integration – Product data representation and exchange. Part 21. Implementation methods: Clear text encoding of the exchange structure.

Requirements with respect to implementation method specific requirements are specified in annex C.

The Protocol Implementation Conformannce Statement (PICS) proforma lists the options or the combination of options that may be included in the implementation. The PICS proforma is provided in annex D.

This part of ISO 10303 provides for only one option that may be supported by an implementation. This option shall be supported by a single class of conformance that consists of all the units of functionality for this part of ISO 10303.

Annex A
(normative)

AIM EXPRESS expanded listing

The following EXPRESS is the expanded form of the short form schema given in 5.2. In the event of any discrepancy between the short form and this expanded listing, the expanded listing shall be used.

```
SCHEMA engineering_properties_schema;

(* ********************************
Constants in the schema engineering_properties_schema
******************************** *)

   CONSTANT
            dummy_gri : geometric_representation_item := representation_item('')
       || geometric_representation_item();
      schema_prefix : STRING := 'ENGINEERING_PROPERTIES_SCHEMA.';
      the_booleans : elementary_space := make_elementary_space(es_booleans);
      the_empty_maths_tuple : maths_tuple := [];
      the_empty_space : finite_space := make_finite_space([]);
       the_complex_numbers : elementary_space :=
       make_elementary_space(es_complex_numbers);
      the_generics : elementary_space := make_elementary_space(es_generics);
      the_integers : elementary_space := make_elementary_space(es_integers);
      the_logicals : elementary_space := make_elementary_space(es_logicals);
      the_numbers : elementary_space := make_elementary_space(es_numbers);
      the_reals : elementary_space := make_elementary_space(es_reals);
      the_strings : elementary_space := make_elementary_space(es_strings);
       the_zero_tuple_space : listed_product_space :=
       make_listed_product_space([]);
      the_complex_tuples : extended_tuple_space :=
      make_extended_tuple_space(the_zero_tuple_space, the_complex_numbers);
      the_integer_tuples : extended_tuple_space :=
      make_extended_tuple_space(the_zero_tuple_space, the_integers);
      the_neg1_one_interval : finite_real_interval := make_finite_real_interval(-
      1.00000, closed, 1.00000, closed);
      the_nonnegative_reals : real_interval_from_min :=
      make_real_interval_from_min(0.00000, closed);
      the_real_tuples : extended_tuple_space :=
      make_extended_tuple_space(the_zero_tuple_space, the_reals);
      the_binarys : elementary_space := make_elementary_space(es_binarys);
       the_maths_spaces : elementary_space :=
       make_elementary_space(es_maths_spaces);
      the_neghalfpi_halfpi_interval : finite_real_interval :=
      make_finite_real_interval(-0.500000 * 3.14159, closed, 0.500000 * 3.14159,
      closed);
      the_negpi_pi_interval : finite_real_interval := make_finite_real_interval(-
      3.14159, open, 3.14159, closed);
      the_tuples : extended_tuple_space :=
      make_extended_tuple_space(the_zero_tuple_space, the_generics);
      the_zero_pi_interval : finite_real_interval :=
      make_finite_real_interval(0.00000, closed, 3.14159, closed);
   END_CONSTANT;
```

```
(* ********************************
Entities in the schema engineering_properties_schema
******************************** *)

   ENTITY SQL_mappable_defined_function
   ABSTRACT SUPERTYPE
   SUBTYPE OF (defined_function);
   END_ENTITY;

   ENTITY abs_function
   SUBTYPE OF (unary_function_call);
   END_ENTITY;

   ENTITY absorbed_dose_measure_with_unit
   SUBTYPE OF (measure_with_unit);
   WHERE
      WR1:
          'ENGINEERING_PROPERTIES_SCHEMA.ABSORBED_DOSE_UNIT' IN
       TYPEOF(SELF\measure_with_unit.unit_component);
   END_ENTITY;

   ENTITY absorbed_dose_unit
   SUBTYPE OF (derived_unit);
   WHERE
      WR1:
          derive_dimensional_exponents(SELF) =
       dimensions_for_si_unit(si_unit_name.gray);
   END_ENTITY;

   ENTITY abstracted_expression_function
   SUBTYPE OF (maths_function, quantifier_expression);
   DERIVE
          SELF\quantifier_expression.variables : LIST [1:?] OF UNIQUE
       generic_variable :=
       remove_first(SELF\multiple_arity_generic_expression.operands);
        expr : generic_expression :=
       SELF\multiple_arity_generic_expression.operands[1];
   WHERE
      WR1:
          SIZEOF(QUERY (operand <*
       SELF\multiple_arity_generic_expression.operands| NOT
       has_values_space(operand))) = 0;
   END_ENTITY;

   ENTITY acceleration_measure_with_unit
   SUBTYPE OF (measure_with_unit);
   WHERE
      WR1:
          'ENGINEERING_PROPERTIES_SCHEMA.ACCELERATION_UNIT' IN
       TYPEOF(SELF\measure_with_unit.unit_component);
   END_ENTITY;

   ENTITY acceleration_unit
   SUBTYPE OF (derived_unit);
   WHERE
      WR1:
          derive_dimensional_exponents(SELF) = dimensional_exponents(1.00000,
       0.00000, -2.00000, 0.00000, 0.00000, 0.00000, 0.00000);
```

```
END_ENTITY;

ENTITY acos_function
SUBTYPE OF (unary_function_call);
END_ENTITY;

ENTITY action;
   name : label;
   description : OPTIONAL text;
   chosen_method : action_method;
DERIVE
   id : identifier := get_id_value(SELF);
WHERE
   WR1:
       SIZEOF(USEDIN(SELF, 'ENGINEERING_PROPERTIES_SCHEMA.' +
    'ID_ATTRIBUTE.IDENTIFIED_ITEM')) <= 1;
END_ENTITY;

ENTITY action_assignment
ABSTRACT SUPERTYPE;
   assigned_action : action;
DERIVE
   role : object_role := get_role(SELF);
WHERE
   WR1:
       SIZEOF(USEDIN(SELF, 'ENGINEERING_PROPERTIES_SCHEMA.' +
    'ROLE_ASSOCIATION.ITEM_WITH_ROLE')) <= 1;
END_ENTITY;

ENTITY action_directive;
   name : label;
   description : OPTIONAL text;
   analysis : text;
   comment : text;
   requests : SET [1:?] OF versioned_action_request;
END_ENTITY;

ENTITY action_method;
   name : label;
   description : OPTIONAL text;
   consequence : text;
   purpose : text;
END_ENTITY;

ENTITY action_method_relationship;
   name : label;
   description : OPTIONAL text;
   relating_method : action_method;
   related_method : action_method;
END_ENTITY;

ENTITY action_method_with_associated_documents
SUBTYPE OF (action_method);
   documents : SET [1:?] OF document;
END_ENTITY;

ENTITY action_method_with_associated_documents_constrained
SUBTYPE OF (action_method_with_associated_documents);
   usage_constraints : SET [1:?] OF document_usage_constraint;
```

```
WHERE
   WR1:
       SIZEOF(QUERY (item <* usage_constraints| NOT (item.source IN
    SELF\action_method_with_associated_documents.documents))) = 0;
END_ENTITY;

ENTITY action_property;
   name : label;
   description : text;
   definition : characterized_action_definition;
END_ENTITY;

ENTITY action_property_relationship;
   name : label;
   description : text;
   relating_action_property : action_property;
   related_action_property : action_property;
WHERE
   WR1:
      relating_action_property :<>: related_action_property;
END_ENTITY;

ENTITY action_property_representation;
   name : label;
   description : text;
   property : action_property;
   representation : representation;
END_ENTITY;

ENTITY action_relationship;
   name : label;
   description : OPTIONAL text;
   relating_action : action;
   related_action : action;
END_ENTITY;

ENTITY action_request_assignment
ABSTRACT SUPERTYPE;
   assigned_action_request : versioned_action_request;
DERIVE
   role : object_role := get_role(SELF);
WHERE
   WR1:
       SIZEOF(USEDIN(SELF, 'ENGINEERING_PROPERTIES_SCHEMA.' +
    'ROLE_ASSOCIATION.ITEM_WITH_ROLE')) <= 1;
END_ENTITY;

ENTITY action_request_solution;
   method : action_method;
   request : versioned_action_request;
DERIVE
   description : text := get_description_value(SELF);
   name : label := get_name_value(SELF);
WHERE
   WR1:
       SIZEOF(USEDIN(SELF, 'ENGINEERING_PROPERTIES_SCHEMA.' +
    'DESCRIPTION_ATTRIBUTE.DESCRIBED_ITEM')) <= 1;
   WR2:
```

```
       SIZEOF(USEDIN(SELF, 'ENGINEERING_PROPERTIES_SCHEMA.' +
    'NAME_ATTRIBUTE.NAMED_ITEM')) <= 1;
END_ENTITY;

ENTITY action_resource;
   name : label;
   description : OPTIONAL text;
   usage : SET [1:?] OF supported_item;
   kind : action_resource_type;
END_ENTITY;

ENTITY action_resource_relationship;
   name : label;
   description : OPTIONAL text;
   relating_resource : action_resource;
   related_resource : action_resource;
END_ENTITY;

ENTITY action_resource_requirement;
   name : label;
   description : text;
   kind : resource_requirement_type;
   OPERATIONS : SET [1:?] OF characterized_action_definition;
END_ENTITY;

ENTITY action_resource_requirement_relationship;
   name : label;
   description : text;
   relating_action_resource_requirement : action_resource_requirement;
   related_action_resource_requirement : action_resource_requirement;
WHERE
   WR1:
       relating_action_resource_requirement :<>:
    related_action_resource_requirement;
END_ENTITY;

ENTITY action_resource_type;
   name : label;
END_ENTITY;

ENTITY action_status;
   status : label;
   assigned_action : executed_action;
END_ENTITY;

ENTITY address;
   internal_location : OPTIONAL label;
   street_number : OPTIONAL label;
   street : OPTIONAL label;
   postal_box : OPTIONAL label;
   town : OPTIONAL label;
   region : OPTIONAL label;
   postal_code : OPTIONAL label;
   country : OPTIONAL label;
   facsimile_number : OPTIONAL label;
   telephone_number : OPTIONAL label;
   electronic_mail_address : OPTIONAL label;
   telex_number : OPTIONAL label;
DERIVE
```

```
   name : label := get_name_value(SELF);
   url : identifier := get_id_value(SELF);
WHERE
   WR1:
      ((((((((((EXISTS(internal_location) OR EXISTS(street_number)) OR
    EXISTS(street)) OR EXISTS(postal_box)) OR EXISTS(town)) OR EXISTS(region))
    OR EXISTS(postal_code)) OR EXISTS(country)) OR EXISTS(facsimile_number))
    OR EXISTS(telephone_number)) OR EXISTS(electronic_mail_address)) OR
    EXISTS(telex_number);
END_ENTITY;

ENTITY amount_of_substance_measure_with_unit
SUBTYPE OF (measure_with_unit);
WHERE
   WR1:
       'ENGINEERING_PROPERTIES_SCHEMA.AMOUNT_OF_SUBSTANCE_UNIT' IN
    TYPEOF(SELF\measure_with_unit.unit_component);
END_ENTITY;

ENTITY amount_of_substance_unit
SUBTYPE OF (named_unit);
WHERE
   WR1:
      ((((((SELF\named_unit.dimensions.length_exponent = 0.00000) AND
    (SELF\named_unit.dimensions.mass_exponent = 0.00000)) AND
    (SELF\named_unit.dimensions.time_exponent = 0.00000)) AND
    (SELF\named_unit.dimensions.electric_current_exponent = 0.00000)) AND
    (SELF\named_unit.dimensions.thermodynamic_temperature_exponent = 0.00000))
    AND (SELF\named_unit.dimensions.amount_of_substance_exponent = 1.00000))
    AND (SELF\named_unit.dimensions.luminous_intensity_exponent = 0.00000);
END_ENTITY;

ENTITY and_expression
SUBTYPE OF (multiple_arity_boolean_expression);
END_ENTITY;

ENTITY angular_location
SUBTYPE OF (dimensional_location);
   angle_selection : angle_relator;
END_ENTITY;

ENTITY angular_size
SUBTYPE OF (dimensional_size);
   angle_selection : angle_relator;
END_ENTITY;

ENTITY angularity_tolerance
SUBTYPE OF (geometric_tolerance_with_datum_reference);
WHERE
   WR1:
      SIZEOF(SELF\geometric_tolerance_with_datum_reference.datum_system) < 3;
END_ENTITY;

ENTITY apex
SUBTYPE OF (derived_shape_aspect);
END_ENTITY;

ENTITY application_context;
   application : label;
```

```
DERIVE
   description : text := get_description_value(SELF);
   id : identifier := get_id_value(SELF);
INVERSE
       context_elements : SET [1:?] OF application_context_element FOR
    frame_of_reference;
WHERE
   WR1:
       SIZEOF(USEDIN(SELF, 'ENGINEERING_PROPERTIES_SCHEMA.' +
    'DESCRIPTION_ATTRIBUTE.DESCRIBED_ITEM')) <= 1;
   WR2:
       SIZEOF(USEDIN(SELF, 'ENGINEERING_PROPERTIES_SCHEMA.' +
    'ID_ATTRIBUTE.IDENTIFIED_ITEM')) <= 1;
END_ENTITY;

ENTITY application_context_element
    SUPERTYPE OF (ONEOF(product_concept_context, product_context,
    product_definition_context));
   name : label;
   frame_of_reference : application_context;
END_ENTITY;

ENTITY application_context_relationship;
   name : label;
   description : OPTIONAL text;
   relating_context : application_context;
   related_context : application_context;
END_ENTITY;

ENTITY application_defined_function
SUBTYPE OF (maths_function);
   explicit_domain : tuple_space;
   explicit_range : tuple_space;
   parameters : LIST OF maths_value;
WHERE
   WR1:
      expression_is_constant(explicit_domain);
   WR2:
      expression_is_constant(explicit_range);
END_ENTITY;

ENTITY applied_action_assignment
SUBTYPE OF (action_assignment);
   item : action_item;
END_ENTITY;

ENTITY applied_action_request_assignment
SUBTYPE OF (action_request_assignment);
   item : action_request_item;
END_ENTITY;

ENTITY applied_approval_assignment
SUBTYPE OF (approval_assignment);
   item : approval_item;
END_ENTITY;

ENTITY applied_certification_assignment
SUBTYPE OF (certification_assignment);
   item : certification_item;
```

```
END_ENTITY;

ENTITY applied_contract_assignment
SUBTYPE OF (contract_assignment);
   item : contract_item;
END_ENTITY;

ENTITY applied_date_and_time_assignment
SUBTYPE OF (date_and_time_assignment);
   item : date_and_time_item;
END_ENTITY;

ENTITY applied_date_assignment
SUBTYPE OF (date_assignment);
   item : date_item;
END_ENTITY;

ENTITY applied_document_reference
SUBTYPE OF (document_reference);
   item : document_item;
END_ENTITY;

ENTITY applied_document_usage_contraint_assignment
SUBTYPE OF (document_usage_constraint_assignment);
   item : document_item;
END_ENTITY;

ENTITY applied_effectivity_assignment
SUBTYPE OF (effectivity_assignment);
   item : effectivity_item;
END_ENTITY;

ENTITY applied_event_occurrence_assignment
SUBTYPE OF (event_occurrence_assignment);
   item : event_occurred_item;
END_ENTITY;

ENTITY applied_external_identification_assignment
SUBTYPE OF (external_identification_assignment);
   items : SET [1:?] OF external_identification_item;
WHERE
   WR1:
       NOT (SELF.role.name = 'version') OR item_correlation(SELF.items,
    [ 'EXTERNALLY_DEFINED_CLASS',
    'EXTERNALLY_DEFINED_ENGINEERING_PROPERTY' ]);
END_ENTITY;

ENTITY applied_group_assignment
SUBTYPE OF (group_assignment);
   items : SET [1:?] OF groupable_item;
END_ENTITY;

ENTITY applied_identification_assignment
SUBTYPE OF (identification_assignment);
   item : identification_item;
END_ENTITY;

ENTITY applied_location_assignment
SUBTYPE OF (location_assignment);
```

```
   item : location_item;
END_ENTITY;

ENTITY applied_location_representation_assignment
SUBTYPE OF (location_representation_assignment);
   item : location_representation_item;
END_ENTITY;

ENTITY applied_organization_assignment
SUBTYPE OF (organization_assignment);
   item : organization_item;
END_ENTITY;

ENTITY applied_organizational_project_assignment
SUBTYPE OF (organizational_project_assignment);
   item : SET [1:?] OF organizational_project_item;
END_ENTITY;

ENTITY applied_person_and_organization_assignment
SUBTYPE OF (person_and_organization_assignment);
   item : person_and_organization_item;
END_ENTITY;

ENTITY applied_person_assignment
SUBTYPE OF (person_assignment);
   item : person_item;
END_ENTITY;

ENTITY applied_qualification_assignment
SUBTYPE OF (qualification_type_assignment);
   item : qualification_item;
END_ENTITY;

ENTITY applied_security_classification_assignment
SUBTYPE OF (security_classification_assignment);
   item : SET [1:?] OF security_classified_item;
END_ENTITY;

ENTITY applied_state_observed_assignment
SUBTYPE OF (state_observed_assignment);
   item : state_observed_item;
END_ENTITY;

ENTITY applied_state_type_assignment
SUBTYPE OF (state_type_assignment);
   item : state_item;
END_ENTITY;

ENTITY applied_time_interval_assignment
SUBTYPE OF (time_interval_assignment);
   item : time_interval_item;
END_ENTITY;

ENTITY approval;
   status : approval_status;
   level : label;
END_ENTITY;

ENTITY approval_assignment
```

```
ABSTRACT SUPERTYPE;
   assigned_approval : approval;
DERIVE
   role : object_role := get_role(SELF);
WHERE
   WR1:
       SIZEOF(USEDIN(SELF, 'ENGINEERING_PROPERTIES_SCHEMA.' +
    'ROLE_ASSOCIATION.ITEM_WITH_ROLE')) <= 1;
END_ENTITY;

ENTITY approval_date_time;
   date_time : date_time_select;
   dated_approval : approval;
DERIVE
   role : object_role := get_role(SELF);
WHERE
   WR1:
       SIZEOF(USEDIN(SELF, 'ENGINEERING_PROPERTIES_SCHEMA.' +
    'ROLE_ASSOCIATION.ITEM_WITH_ROLE')) <= 1;
END_ENTITY;

ENTITY approval_person_organization;
   person_organization : person_organization_select;
   authorized_approval : approval;
   role : approval_role;
END_ENTITY;

ENTITY approval_relationship;
   name : label;
   description : OPTIONAL text;
   relating_approval : approval;
   related_approval : approval;
END_ENTITY;

ENTITY approval_role;
   role : label;
DERIVE
   description : text := get_description_value(SELF);
WHERE
   WR1:
       SIZEOF(USEDIN(SELF, 'ENGINEERING_PROPERTIES_SCHEMA.' +
    'DESCRIPTION_ATTRIBUTE.DESCRIBED_ITEM')) <= 1;
END_ENTITY;

ENTITY approval_status;
   name : label;
END_ENTITY;

ENTITY area_measure_with_unit
SUBTYPE OF (measure_with_unit);
WHERE
   WR1:
       'ENGINEERING_PROPERTIES_SCHEMA.AREA_UNIT' IN
    TYPEOF(SELF\measure_with_unit.unit_component);
END_ENTITY;

ENTITY area_unit
SUBTYPE OF (derived_unit);
WHERE
```

```
   WR1:
       derive_dimensional_exponents(SELF) = dimensional_exponents(2.00000,
    0.00000, 0.00000, 0.00000, 0.00000, 0.00000, 0.00000);
END_ENTITY;

ENTITY ascribable_state;
   name : label;
   description : OPTIONAL text;
   pertaining_state_type : state_type;
   ascribed_state_observed : state_observed;
END_ENTITY;

ENTITY ascribable_state_relationship;
   name : label;
   description : OPTIONAL text;
   relating_ascribable_state : ascribable_state;
   related_ascribable_state : ascribable_state;
END_ENTITY;

ENTITY asin_function
SUBTYPE OF (unary_function_call);
END_ENTITY;

ENTITY atan_function
SUBTYPE OF (binary_function_call);
END_ENTITY;

ENTITY atom_based_literal
SUBTYPE OF (generic_literal);
   lit_value : atom_based_value;
END_ENTITY;

ENTITY attribute_classification_assignment
ABSTRACT SUPERTYPE;
   assigned_class : group;
   attribute_name : label;
   role : classification_role;
END_ENTITY;

ENTITY attribute_language_assignment
SUBTYPE OF (attribute_classification_assignment);
   items : SET [1:?] OF attribute_language_item;
DERIVE
      language : label :=
    SELF\attribute_classification_assignment.assigned_class.name;
WHERE
   WR1:
       SELF\attribute_classification_assignment.role.name IN [ 'primary',
    'translated' ];
   WR2:
       'ENGINEERING_PROPERTIES_SCHEMA.' + 'LANGUAGE' IN
    TYPEOF(SELF\attribute_classification_assignment.assigned_class);
END_ENTITY;

ENTITY attribute_value_assignment
ABSTRACT SUPERTYPE;
   attribute_name : label;
   attribute_value : attribute_type;
   role : attribute_value_role;
```

```
END_ENTITY;

ENTITY attribute_value_role;
   name : label;
   description : OPTIONAL text;
END_ENTITY;

ENTITY axis1_placement
SUBTYPE OF (placement);
   axis : OPTIONAL direction;
DERIVE
      z : direction := NVL(normalise(axis), dummy_gri || direction([ 0.00000,
    0.00000, 1.00000 ]));
WHERE
   WR1:
      SELF\geometric_representation_item.dim = 3;
END_ENTITY;

ENTITY axis2_placement_2d
SUBTYPE OF (placement);
   ref_direction : OPTIONAL direction;
DERIVE
   p : LIST [2:2] OF direction := build_2axes(ref_direction);
WHERE
   WR1:
      SELF\geometric_representation_item.dim = 2;
END_ENTITY;

ENTITY axis2_placement_3d
SUBTYPE OF (placement);
   axis : OPTIONAL direction;
   ref_direction : OPTIONAL direction;
DERIVE
   p : LIST [3:3] OF direction := build_axes(axis, ref_direction);
WHERE
   WR1:
      SELF\placement.location.dim = 3;
   WR2:
      NOT EXISTS(axis) OR (axis.dim = 3);
   WR3:
      NOT EXISTS(ref_direction) OR (ref_direction.dim = 3);
   WR4:
       (NOT EXISTS(axis) OR NOT EXISTS(ref_direction)) OR (cross_product(axis,
       ref_direction).magnitude > 0.00000);
END_ENTITY;

ENTITY b_spline_basis
SUBTYPE OF (maths_function, generic_literal);
   degree : nonnegative_integer;
   repeated_knots : LIST [2:?] OF REAL;
DERIVE
   order : positive_integer := degree + 1;
   num_basis : positive_integer := SIZEOF(repeated_knots) - order;
WHERE
   WR1:
      num_basis >= order;
   WR2:
      nondecreasing(repeated_knots);
   WR3:
```

```
      repeated_knots[order] < repeated_knots[(num_basis + 1)];
END_ENTITY;

ENTITY b_spline_function
SUBTYPE OF (maths_function, unary_generic_expression);
   SELF\unary_generic_expression.operand : maths_function;
   basis : LIST [1:?] OF b_spline_basis;
DERIVE
   coef : maths_function := SELF\unary_generic_expression.operand;
WHERE
   WR1:
      function_is_table(coef);
   WR2:
      (space_dimension(coef.range) = 1) AND
    (number_superspace_of(factor1(coef.range)) = the_reals);
   WR3:
      SIZEOF(basis) <= SIZEOF(shape_of_array(coef));
   WR4:
      compare_basis_and_coef(basis, coef);
END_ENTITY;

ENTITY banded_matrix
SUBTYPE OF (linearized_table_function);
   default_entry : maths_value;
   below : INTEGER;
   above : INTEGER;
   order : ordering_type;
WHERE
   WR1:
      SIZEOF(SELF\explicit_table_function.shape) = 2;
   WR2:
      -below <= above;
   WR3:
       member_of(default_entry,
    factor1(SELF\linearized_table_function.source.range));
END_ENTITY;

ENTITY basic_sparse_matrix
SUBTYPE OF (explicit_table_function, multiple_arity_generic_expression);
    SELF\multiple_arity_generic_expression.operands : LIST [3:3] OF
    maths_function;
   default_entry : maths_value;
   order : ordering_type;
DERIVE
     index : maths_function :=
    SELF\multiple_arity_generic_expression.operands[1];
   loc : maths_function := SELF\multiple_arity_generic_expression.operands[2];
   val : maths_function := SELF\multiple_arity_generic_expression.operands[3];
WHERE
   WR1:
      function_is_1d_table(index);
   WR2:
      function_is_1d_table(loc);
   WR3:
      function_is_1d_table(val);
   WR4:
      check_sparse_index_domain(index.domain, index_base, shape, order);
   WR5:
      check_sparse_index_to_loc(index.range, loc.domain);
```

```
   WR6:
      loc.domain = val.domain;
   WR7:
      check_sparse_loc_range(loc.range, index_base, shape, order);
   WR8:
      member_of(default_entry, val.range);
END_ENTITY;

ENTITY binary_boolean_expression
ABSTRACT SUPERTYPE OF (ONEOF(xor_expression, equals_expression))
SUBTYPE OF (boolean_expression, binary_generic_expression);
END_ENTITY;

ENTITY binary_function_call
ABSTRACT SUPERTYPE OF (atan_function)
SUBTYPE OF (binary_numeric_expression);
END_ENTITY;

ENTITY binary_generic_expression
ABSTRACT SUPERTYPE
SUBTYPE OF (generic_expression);
   operands : LIST [2:2] OF generic_expression;
END_ENTITY;

ENTITY binary_literal
SUBTYPE OF (generic_literal);
   lit_value : BINARY;
END_ENTITY;

ENTITY binary_numeric_expression
       ABSTRACT SUPERTYPE OF (ONEOF(minus_expression, div_expression,
    mod_expression, power_expression, binary_function_call))
SUBTYPE OF (numeric_expression, binary_generic_expression);
   SELF\binary_generic_expression.operands : LIST [2:2] OF numeric_expression;
END_ENTITY;

ENTITY block_volume
SUBTYPE OF (volume);
   position : axis2_placement_3d;
   x : positive_length_measure;
   y : positive_length_measure;
   z : positive_length_measure;
END_ENTITY;

ENTITY boolean_defined_function
ABSTRACT SUPERTYPE
SUBTYPE OF (defined_function, boolean_expression);
END_ENTITY;

ENTITY boolean_expression
       ABSTRACT SUPERTYPE OF (ONEOF(simple_boolean_expression,
    unary_boolean_expression, binary_boolean_expression,
    multiple_arity_boolean_expression, comparison_expression,
    interval_expression, boolean_defined_function))
SUBTYPE OF (expression);
END_ENTITY;

ENTITY boolean_literal
SUBTYPE OF (simple_boolean_expression, generic_literal);
```

```
   the_value : BOOLEAN;
END_ENTITY;

ENTITY boolean_variable
SUBTYPE OF (simple_boolean_expression, variable);
END_ENTITY;

ENTITY bound_variable_semantics
SUBTYPE OF (variable_semantics);
END_ENTITY;

ENTITY calendar_date
SUBTYPE OF (date);
   day_component : day_in_month_number;
   month_component : month_in_year_number;
WHERE
   WR1:
      valid_calendar_date(SELF);
END_ENTITY;

ENTITY capacitance_measure_with_unit
SUBTYPE OF (measure_with_unit);
WHERE
   WR1:
       'ENGINEERING_PROPERTIES_SCHEMA.CAPACITANCE_UNIT' IN
    TYPEOF(SELF\measure_with_unit.unit_component);

END_ENTITY;

ENTITY capacitance_unit
SUBTYPE OF (derived_unit);
WHERE
   WR1:
       derive_dimensional_exponents(SELF) =
    dimensions_for_si_unit(si_unit_name.farad);
END_ENTITY;

ENTITY cartesian_complex_number_region
SUBTYPE OF (maths_space, generic_literal);
   real_constraint : real_interval;
   imag_constraint : real_interval;
WHERE
   WR1:
       ((min_exists(real_constraint) OR max_exists(real_constraint)) OR
    min_exists(imag_constraint)) OR max_exists(imag_constraint);
END_ENTITY;

ENTITY cartesian_point
SUPERTYPE OF (ONEOF(cylindrical_point, polar_point, spherical_point))
SUBTYPE OF (point);
   coordinates : LIST [1:3] OF length_measure;
END_ENTITY;

ENTITY cartesian_transformation_operator
SUPERTYPE OF (cartesian_transformation_operator_3d)
    SUBTYPE OF (geometric_representation_item,
    functionally_defined_transformation);
   axis1 : OPTIONAL direction;
   axis2 : OPTIONAL direction;
```

```
   local_origin : cartesian_point;
   scale : OPTIONAL REAL;
DERIVE
   scl : REAL := NVL(scale, 1.00000);
WHERE
   WR1:
      scl > 0.00000;
END_ENTITY;

ENTITY cartesian_transformation_operator_3d
SUBTYPE OF (cartesian_transformation_operator);
   axis3 : OPTIONAL direction;
DERIVE
      u : LIST [3:3] OF direction := base_axis(3,
    SELF\cartesian_transformation_operator.axis1,
    SELF\cartesian_transformation_operator.axis2, axis3);
       WHERE
   WR1:
      SELF\geometric_representation_item.dim = 3;
END_ENTITY;

ENTITY celsius_temperature_measure_with_unit
SUBTYPE OF (measure_with_unit);
WHERE
   WR1:
       'ENGINEERING_PROPERTIES_SCHEMA.THERMODYNAMIC_TEMPERATURE_UNIT' IN
    TYPEOF(SELF\measure_with_unit.unit_component);
END_ENTITY;

ENTITY centre_of_symmetry
SUBTYPE OF (derived_shape_aspect);
WHERE
   WR1:
       SIZEOF(QUERY (sadr <* SELF\derived_shape_aspect.deriving_relationships|
    NOT ('ENGINEERING_PROPERTIES_SCHEMA.SYMMETRIC_SHAPE_ASPECT' IN
    TYPEOF(sadr\shape_aspect_relationship.related_shape_aspect)))) = 0;
END_ENTITY;

ENTITY certification;
   name : label;
   purpose : text;
   kind : certification_type;
END_ENTITY;

ENTITY certification_assignment
ABSTRACT SUPERTYPE;
   assigned_certification : certification;
DERIVE
   role : object_role := get_role(SELF);
WHERE
   WR1:
       SIZEOF(USEDIN(SELF, 'ENGINEERING_PROPERTIES_SCHEMA.' +
    'ROLE_ASSOCIATION.ITEM_WITH_ROLE')) <= 1;
END_ENTITY;

ENTITY certification_type;
   description : label;
END_ENTITY;
```

```
ENTITY characterized_object;
   name : label;
   description : OPTIONAL text;
END_ENTITY;

ENTITY circle
SUBTYPE OF (conic);
   radius : positive_length_measure;
END_ENTITY;

ENTITY class
SUBTYPE OF (group);
END_ENTITY;

ENTITY classification_assignment
ABSTRACT SUPERTYPE;
   assigned_class : group;
   role : classification_role;
END_ENTITY;

ENTITY classification_role;
   name : label;
   description : OPTIONAL text;
END_ENTITY;

ENTITY coaxiality_tolerance
SUBTYPE OF (geometric_tolerance_with_datum_reference);
WHERE
   WR1:
      SIZEOF(SELF\geometric_tolerance_with_datum_reference.datum_system) <= 2;
END_ENTITY;

ENTITY comparison_equal
SUBTYPE OF (comparison_expression);
END_ENTITY;

ENTITY comparison_expression
       ABSTRACT SUPERTYPE OF (ONEOF(comparison_equal, comparison_greater,
    comparison_greater_equal, comparison_less, comparison_less_equal,
    comparison_not_equal, like_expression))
SUBTYPE OF (boolean_expression, binary_generic_expression);
   SELF\binary_generic_expression.operands : LIST [2:2] OF expression;
WHERE
   WR1:
     (('ENGINEERING_PROPERTIES_SCHEMA.NUMERIC_EXPRESSION' IN
    TYPEOF(SELF\binary_generic_expression.operands[1])) AND
    ('ENGINEERING_PROPERTIES_SCHEMA.NUMERIC_EXPRESSION' IN
    TYPEOF(SELF\binary_generic_expression.operands[2])) OR
    ('ENGINEERING_PROPERTIES_SCHEMA.BOOLEAN_EXPRESSION' IN
    TYPEOF(SELF\binary_generic_expression.operands[1])) AND
    ('ENGINEERING_PROPERTIES_SCHEMA.BOOLEAN_EXPRESSION' IN
    TYPEOF(SELF\binary_generic_expression.operands[2]))) OR
    ('ENGINEERING_PROPERTIES_SCHEMA.STRING_EXPRESSION' IN
    TYPEOF(SELF\binary_generic_expression.operands[1])) AND
    ('ENGINEERING_PROPERTIES_SCHEMA.STRING_EXPRESSION' IN
    TYPEOF(SELF\binary_generic_expression.operands[2]));
END_ENTITY;

ENTITY comparison_greater
```

```
SUBTYPE OF (comparison_expression);
END_ENTITY;

ENTITY comparison_greater_equal
SUBTYPE OF (comparison_expression);
END_ENTITY;

ENTITY comparison_less
SUBTYPE OF (comparison_expression);
END_ENTITY;

ENTITY comparison_less_equal
SUBTYPE OF (comparison_expression);
END_ENTITY;

ENTITY comparison_not_equal
SUBTYPE OF (comparison_expression);
END_ENTITY;

ENTITY complex_number_literal
SUBTYPE OF (generic_literal);
   real_part : REAL;
   imag_part : REAL;
END_ENTITY;

ENTITY composite_shape_aspect
SUBTYPE OF (shape_aspect);
INVERSE
     component_relationships : SET [2:?] OF shape_aspect_relationship FOR
    relating_shape_aspect;
END_ENTITY;

ENTITY compound_representation_item
SUBTYPE OF (representation_item);
   item_element : compound_item_definition;
END_ENTITY;

ENTITY concat_expression
SUBTYPE OF (string_expression, multiple_arity_generic_expression);
      SELF\multiple_arity_generic_expression.operands : LIST [2:?] OF
    string_expression;
END_ENTITY;

ENTITY concentricity_tolerance
SUBTYPE OF (geometric_tolerance_with_datum_reference);
WHERE
   WR1:
      SIZEOF(SELF\geometric_tolerance_with_datum_reference.datum_system) = 1;
END_ENTITY;

ENTITY concurrent_action_method
SUBTYPE OF (action_method_relationship);
END_ENTITY;

ENTITY conductance_measure_with_unit
SUBTYPE OF (measure_with_unit);
WHERE
   WR1:
```

```
       'ENGINEERING_PROPERTIES_SCHEMA.CONDUCTANCE_UNIT' IN
    TYPEOF(SELF\measure_with_unit.unit_component);
END_ENTITY;

ENTITY conductance_unit
SUBTYPE OF (derived_unit);
WHERE
   WR1:
       derive_dimensional_exponents(SELF) =
    dimensions_for_si_unit(si_unit_name.siemens);
END_ENTITY;

ENTITY configuration_design;
   configuration : configuration_item;
   design : configuration_design_item;
DERIVE
   name : label := get_name_value(SELF);
   description : text := get_description_value(SELF);
UNIQUE
   UR1 : configuration, design;
WHERE
   WR1:
       SIZEOF(USEDIN(SELF, 'ENGINEERING_PROPERTIES_SCHEMA.' +
    'NAME_ATTRIBUTE.NAMED_ITEM')) <= 1;
   WR2:
       SIZEOF(USEDIN(SELF, 'ENGINEERING_PROPERTIES_SCHEMA.' +
    'DESCRIPTION_ATTRIBUTE.DESCRIBED_ITEM')) <= 1;
END_ENTITY;

ENTITY configuration_item;
   id : identifier;
   name : label;
   description : OPTIONAL text;
   item_concept : product_concept;
   purpose : OPTIONAL label;
END_ENTITY;

ENTITY conic
SUPERTYPE OF (ONEOF(circle, ellipse, hyperbola, parabola))
SUBTYPE OF (curve);
   position : axis2_placement;
END_ENTITY;

ENTITY conical_surface
SUBTYPE OF (elementary_surface);
   radius : length_measure;
   semi_angle : plane_angle_measure;
WHERE
   WR1:
      radius >= 0.00000;
END_ENTITY;

ENTITY constant_function
SUBTYPE OF (maths_function, generic_literal);
   sole_output : maths_value;
   source_of_domain : maths_space_or_function;
WHERE
   WR1:
      no_cyclic_domain_reference(source_of_domain, [ SELF ]);
```

```
   WR2:
      expression_is_constant(domain_from(source_of_domain));
END_ENTITY;

ENTITY context_dependent_action_method_relationship;
   name : label;
   relating_relationship : action_method_relationship;
   related_relationship : action_method_relationship;
UNIQUE
   UR1 : relating_relationship, related_relationship;
WHERE
   WR1:
       relating_relationship.relating_method :=:
    related_relationship.relating_method;
END_ENTITY;

ENTITY context_dependent_action_relationship;
   name : label;
   relating_relationship : action_relationship;
   related_relationship : action_relationship;
UNIQUE
   UR1 : relating_relationship, related_relationship;
WHERE
   WR1:
       relating_relationship.relating_action :=:
    related_relationship.relating_action;
END_ENTITY;

ENTITY context_dependent_shape_representation;
   representation_relation : shape_representation_relationship;
   represented_product_relation : product_definition_shape;
DERIVE
   description : text := get_description_value(SELF);
   name : label := get_name_value(SELF);
WHERE
   WR1:
       'ENGINEERING_PROPERTIES_SCHEMA.PRODUCT_DEFINITION_RELATIONSHIP' IN
    TYPEOF(SELF.represented_product_relation.definition);
   WR2:
       SIZEOF(USEDIN(SELF, 'ENGINEERING_PROPERTIES_SCHEMA.' +
    'DESCRIPTION_ATTRIBUTE.DESCRIBED_ITEM')) <= 1;
   WR3:
       SIZEOF(USEDIN(SELF, 'ENGINEERING_PROPERTIES_SCHEMA.' +
    'NAME_ATTRIBUTE.NAMED_ITEM')) <= 1;
END_ENTITY;

ENTITY context_dependent_unit
SUBTYPE OF (named_unit);
   name : label;
END_ENTITY;

ENTITY contract;
   name : label;
   purpose : text;
   kind : contract_type;
END_ENTITY;

ENTITY contract_assignment
ABSTRACT SUPERTYPE;
```

```
   assigned_contract : contract;
DERIVE
   role : object_role := get_role(SELF);
WHERE
   WR1:
       SIZEOF(USEDIN(SELF, 'ENGINEERING_PROPERTIES_SCHEMA.' +
    'ROLE_ASSOCIATION.ITEM_WITH_ROLE')) <= 1;
END_ENTITY;

ENTITY contract_relationship;
   id : identifier;
   name : label;
   description : OPTIONAL text;
   relating_contract : contract;
   related_contract : contract;
END_ENTITY;

ENTITY contract_type;
   description : label;
END_ENTITY;

ENTITY conversion_based_unit
SUBTYPE OF (named_unit);
   name : label;
   conversion_factor : measure_with_unit;
DERIVE
      SELF\named_unit.dimensions : dimensional_exponents :=
    derive_dimensional_exponents(conversion_factor\measure_with_unit.unit_comp
    onent);
END_ENTITY;

ENTITY coordinated_universal_time_offset;
   hour_offset : INTEGER;
   minute_offset : OPTIONAL INTEGER;
   sense : ahead_or_behind;
DERIVE
   actual_minute_offset : INTEGER := NVL(minute_offset, 0);
WHERE
   WR1:
      (0 <= hour_offset) AND (hour_offset < 24);
   WR2:
      (0 <= actual_minute_offset) AND (actual_minute_offset <= 59);
   WR3:
       NOT (((hour_offset <> 0) OR (actual_minute_offset <> 0)) AND (sense =
    exact));
END_ENTITY;

ENTITY cos_function
SUBTYPE OF (unary_function_call);
END_ENTITY;

ENTITY curve
SUPERTYPE OF (ONEOF(line, conic))
SUBTYPE OF (geometric_representation_item);
END_ENTITY;

ENTITY cylindrical_point
SUBTYPE OF (cartesian_point);
   r : length_measure;
```

```
   theta : plane_angle_measure;
   z : length_measure;
DERIVE
       SELF\cartesian_point.coordinates : LIST [1:3] OF length_measure := [ r
    * COS(theta), r * SIN(theta), z ];
WHERE
   WR1:
      r >= 0.00000;
END_ENTITY;

ENTITY cylindrical_surface
SUBTYPE OF (elementary_surface);
   radius : positive_length_measure;
END_ENTITY;

ENTITY cylindrical_volume
SUBTYPE OF (volume);
   position : axis2_placement_3d;
   radius : positive_length_measure;
   height : positive_length_measure;
END_ENTITY;

ENTITY cylindricity_tolerance
SUBTYPE OF (geometric_tolerance);
WHERE
   WR1:
        NOT ('ENGINEERING_PROPERTIES_SCHEMA.' +
    'GEOMETRIC_TOLERANCE_WITH_DATUM_REFERENCE' IN TYPEOF(SELF));
END_ENTITY;

ENTITY data_environment;
   name : label;
   description : text;
   elements : SET [1:?] OF property_definition_representation;
END_ENTITY;

ENTITY data_environment_relationship;
   name : label;
   description : text;
   relating_data_environment : data_environment;
   related_data_environment : data_environment;
END_ENTITY;

ENTITY date
    SUPERTYPE OF (ONEOF(calendar_date, ordinal_date, week_of_year_and_day_date,
    year_month));
   year_component : year_number;
END_ENTITY;

ENTITY date_and_time;
   date_component : date;
   time_component : local_time;
END_ENTITY;

ENTITY date_and_time_assignment
ABSTRACT SUPERTYPE;
   assigned_date_and_time : date_and_time;
   role : date_time_role;
END_ENTITY;
```

```
   ENTITY date_assignment
   ABSTRACT SUPERTYPE;
      assigned_date : date;
      role : date_role;
   END_ENTITY;

   ENTITY date_role;
      name : label;
   DERIVE
      description : text := get_description_value(SELF);
   WHERE
      WR1:
          SIZEOF(USEDIN(SELF, 'ENGINEERING_PROPERTIES_SCHEMA.' +
       'DESCRIPTION_ATTRIBUTE.DESCRIBED_ITEM')) <= 1;
   END_ENTITY;

   ENTITY date_time_role;
      name : label;
   DERIVE
      description : text := get_description_value(SELF);
   WHERE
      WR1:
          SIZEOF(USEDIN(SELF, 'ENGINEERING_PROPERTIES_SCHEMA.' +
       'DESCRIPTION_ATTRIBUTE.DESCRIBED_ITEM')) <= 1;
   END_ENTITY;

   ENTITY dated_effectivity
   SUBTYPE OF (effectivity);
      effectivity_end_date : OPTIONAL date_time_or_event_occurrence;
      effectivity_start_date : date_time_or_event_occurrence;
   END_ENTITY;

   ENTITY datum
   SUBTYPE OF (shape_aspect);
      identification : identifier;
   INVERSE
      established_by_relationships : SET [1:?] OF shape_aspect_relationship FOR
related_shape_aspect;
   WHERE
      WR1:
          SIZEOF(QUERY (x <* SELF\datum.established_by_relationships|
       (SIZEOF(TYPEOF(x\shape_aspect_relationship.relating_shape_aspect) *
       [ 'ENGINEERING_PROPERTIES_SCHEMA.DATUM_FEATURE',
       'ENGINEERING_PROPERTIES_SCHEMA.DATUM_TARGET' ]) <> 1))) = 0;
   END_ENTITY;

   ENTITY datum_feature
   SUBTYPE OF (shape_aspect);
   INVERSE
          feature_basis_relationship : shape_aspect_relationship FOR
       relating_shape_aspect;
   WHERE
      WR1:
          SIZEOF(QUERY (sar <* bag_to_set(USEDIN(SELF,
       'ENGINEERING_PROPERTIES_SCHEMA.SHAPE_ASPECT_RELATIONSHIP.' +
       'RELATING_SHAPE_ASPECT'))| NOT ('ENGINEERING_PROPERTIES_SCHEMA.DATUM' IN
       TYPEOF(sar\shape_aspect_relationship.related_shape_aspect)))) = 0;
      WR2:
```

```
      SELF\shape_aspect.product_definitional = TRUE;
END_ENTITY;

ENTITY datum_reference;
   precedence : INTEGER;
   referenced_datum : datum;
WHERE
   WR1:
      precedence > 0;
END_ENTITY;

ENTITY datum_target
SUBTYPE OF (shape_aspect);
   target_id : identifier;
INVERSE
      target_basis_relationship : shape_aspect_relationship FOR
    relating_shape_aspect;
WHERE
   WR1:
       SIZEOF(QUERY (sar <* bag_to_set(USEDIN(SELF,
    'ENGINEERING_PROPERTIES_SCHEMA.SHAPE_ASPECT_RELATIONSHIP.' +
    'RELATING_SHAPE_ASPECT'))| NOT ('ENGINEERING_PROPERTIES_SCHEMA.DATUM' IN
    TYPEOF(sar\shape_aspect_relationship.related_shape_aspect)))) = 0;
   WR2:
      SELF\shape_aspect.product_definitional = TRUE;
END_ENTITY;

ENTITY defined_function
       ABSTRACT SUPERTYPE OF (ONEOF(numeric_defined_function,
    string_defined_function, boolean_defined_function) ANDOR
    SQL_mappable_defined_function);
END_ENTITY;

ENTITY definite_integral_expression
SUBTYPE OF (quantifier_expression);
   lower_limit_neg_infinity : BOOLEAN;
   upper_limit_pos_infinity : BOOLEAN;
DERIVE
       integrand : generic_expression :=
    SELF\multiple_arity_generic_expression.operands[1];
          variable_of_integration : maths_variable :=
    SELF\multiple_arity_generic_expression.operands[2];
    SELF\quantifier_expression.variables : LIST [1:1] OF UNIQUE
    generic_variable := [ variable_of_integration ];
WHERE
   WR1:
      has_values_space(integrand);
   WR2:
      space_is_continuum(values_space_of(integrand));
    WR3:
    definite_integral_expr_check(SELF\multiple_arity_generic_expression.operan
    ds, lower_limit_neg_infinity, upper_limit_pos_infinity);
END_ENTITY;

ENTITY definite_integral_function
SUBTYPE OF (maths_function, unary_generic_expression);
   SELF\unary_generic_expression.operand : maths_function;
   variable_of_integration : input_selector;
   lower_limit_neg_infinity : BOOLEAN;
```

```
   upper_limit_pos_infinity : BOOLEAN;
DERIVE
   integrand : maths_function := SELF\unary_generic_expression.operand;
WHERE
   WR1:
      space_is_continuum(integrand.range);
   WR2:
      definite_integral_check(integrand.domain, variable_of_integration,
    lower_limit_neg_infinity, upper_limit_pos_infinity);
END_ENTITY;

ENTITY dependent_variable_definition
SUBTYPE OF (unary_generic_expression);
   name : label;
   description : text;
END_ENTITY;

ENTITY derived_shape_aspect
       SUPERTYPE OF (ONEOF(apex, centre_of_symmetry, geometric_alignment,
    geometric_intersection, parallel_offset, perpendicular_to, extension,
    tangent))
SUBTYPE OF (shape_aspect);
INVERSE
       deriving_relationships : SET [1:?] OF shape_aspect_relationship FOR
    relating_shape_aspect;
WHERE
   WR1:
       SIZEOF(QUERY (dr <* SELF\derived_shape_aspect.deriving_relationships|
    NOT ('ENGINEERING_PROPERTIES_SCHEMA.' +
    'SHAPE_ASPECT_DERIVING_RELATIONSHIP' IN TYPEOF(dr)))) = 0;
END_ENTITY;

ENTITY derived_unit
     SUPERTYPE OF (ONEOF(absorbed_dose_unit, acceleration_unit,
    radioactivity_unit, area_unit, capacitance_unit, dose_equivalent_unit,
    electric_charge_unit, conductance_unit, electric_potential_unit,
    energy_unit, magnetic_flux_density_unit, force_unit, frequency_unit,
    illuminance_unit, inductance_unit, magnetic_flux_unit, power_unit,
    pressure_unit, resistance_unit, volume_unit));
   elements : SET [1:?] OF derived_unit_element;
DERIVE
   name : label := get_name_value(SELF);
WHERE
   WR1:
      (SIZEOF(elements) > 1) OR (SIZEOF(elements) = 1) AND
    (elements[1].exponent <> 1.00000);
   WR2:
      SIZEOF(USEDIN(SELF, 'ENGINEERING_PROPERTIES_SCHEMA.' +
    'NAME_ATTRIBUTE.NAMED_ITEM')) <= 1;
END_ENTITY;

ENTITY derived_unit_element;
   unit : named_unit;
   exponent : REAL;
END_ENTITY;

ENTITY description_attribute;
   attribute_value : text;
   described_item : description_attribute_select;
```

```
END_ENTITY;

ENTITY descriptive_representation_item
SUBTYPE OF (representation_item);
   description : text;
END_ENTITY;

ENTITY dimension_related_tolerance_zone_element;
   related_dimension : dimensional_location;
   related_element : tolerance_zone_definition;
END_ENTITY;

ENTITY dimensional_characteristic_representation;
   dimension : dimensional_characteristic;
   representation : shape_dimension_representation;
END_ENTITY;

ENTITY dimensional_exponents;
   length_exponent : REAL;
   mass_exponent : REAL;
   time_exponent : REAL;
   electric_current_exponent : REAL;
   thermodynamic_temperature_exponent : REAL;
   amount_of_substance_exponent : REAL;
   luminous_intensity_exponent : REAL;
END_ENTITY;

ENTITY dimensional_location
SUPERTYPE OF (ONEOF(angular_location, dimensional_location_with_path))
SUBTYPE OF (shape_aspect_relationship);
END_ENTITY;

ENTITY dimensional_location_with_path
SUBTYPE OF (dimensional_location);
   path : shape_aspect;
END_ENTITY;

ENTITY dimensional_size
SUPERTYPE OF (ONEOF(angular_size, dimensional_size_with_path));
   applies_to : shape_aspect;
   name : label;
WHERE
   WR1:
      applies_to.product_definitional = TRUE;
END_ENTITY;

ENTITY dimensional_size_with_path
SUBTYPE OF (dimensional_size);
   path : shape_aspect;
END_ENTITY;

ENTITY direction
SUBTYPE OF (geometric_representation_item);
   direction_ratios : LIST [2:3] OF REAL;
WHERE
   WR1:
      SIZEOF(QUERY (tmp <* direction_ratios| (tmp <> 0.00000))) > 0;
END_ENTITY;
```

```
ENTITY div_expression
SUBTYPE OF (binary_numeric_expression);
END_ENTITY;

ENTITY document;
   id : identifier;
   name : label;
   description : OPTIONAL text;
   kind : document_type;
INVERSE
      representation_types : SET [0:?] OF document_representation_type FOR
    represented_document;
END_ENTITY;

ENTITY document_product_association;
   name : label;
   description : OPTIONAL text;
   relating_document : document;
   related_product : product_or_formation_or_definition;
END_ENTITY;

ENTITY document_reference
ABSTRACT SUPERTYPE;
   assigned_document : document;
   source : label;
DERIVE
   role : object_role := get_role(SELF);
WHERE
   WR1:
      SIZEOF(USEDIN(SELF, 'ENGINEERING_PROPERTIES_SCHEMA.' +
    'ROLE_ASSOCIATION.ITEM_WITH_ROLE')) <= 1;
END_ENTITY;

ENTITY document_relationship;
   name : label;
   description : OPTIONAL text;
   relating_document : document;
   related_document : document;
END_ENTITY;

ENTITY document_representation_type;
   name : label;
   represented_document : document;
END_ENTITY;

ENTITY document_type;
   product_data_type : label;
END_ENTITY;

ENTITY document_usage_constraint;
   source : document;
   subject_element : label;
   subject_element_value : text;
END_ENTITY;

ENTITY document_usage_constraint_assignment
ABSTRACT SUPERTYPE;
   assigned_document_usage : document_usage_constraint;
   role : document_usage_role;
```

```
END_ENTITY;

ENTITY document_usage_role;
   name : label;
   description : OPTIONAL text;
END_ENTITY;

ENTITY dose_equivalent_measure_with_unit
SUBTYPE OF (measure_with_unit);
WHERE
   WR1:
       'ENGINEERING_PROPERTIES_SCHEMA.DOSE_EQUIVALENT_UNIT' IN
    TYPEOF(SELF\measure_with_unit.unit_component);
END_ENTITY;

ENTITY dose_equivalent_unit
SUBTYPE OF (derived_unit);
WHERE
   WR1:
       derive_dimensional_exponents(SELF) =
    dimensions_for_si_unit(si_unit_name.sievert);
END_ENTITY;

ENTITY effectivity
     SUPERTYPE OF (ONEOF(serial_numbered_effectivity, dated_effectivity,
    time_interval_based_effectivity));
   id : identifier;
DERIVE
   name : label := get_name_value(SELF);
   description : text := get_description_value(SELF);
WHERE
   WR1:
       SIZEOF(USEDIN(SELF, 'ENGINEERING_PROPERTIES_SCHEMA.' +
    'NAME_ATTRIBUTE.NAMED_ITEM')) <= 1;
   WR2:
       SIZEOF(USEDIN(SELF, 'ENGINEERING_PROPERTIES_SCHEMA.' +
    'DESCRIPTION_ATTRIBUTE.DESCRIBED_ITEM')) <= 1;
END_ENTITY;

ENTITY effectivity_assignment
ABSTRACT SUPERTYPE;
   assigned_effectivity : effectivity;
DERIVE
   role : object_role := get_role(SELF);
WHERE
   WR1:
       SIZEOF(USEDIN(SELF, 'ENGINEERING_PROPERTIES_SCHEMA.' +
    'ROLE_ASSOCIATION.ITEM_WITH_ROLE')) <= 1;
END_ENTITY;

ENTITY effectivity_relationship;
   name : label;
   description : OPTIONAL text;

   related_effectivity : effectivity;
   relating_effectivity : effectivity;
END_ENTITY;

ENTITY electric_charge_measure_with_unit
```

```
SUBTYPE OF (measure_with_unit);
WHERE
   WR1:
       'ENGINEERING_PROPERTIES_SCHEMA.ELECTRIC_CHARGE_UNIT' IN
    TYPEOF(SELF\measure_with_unit.unit_component);
END_ENTITY;

ENTITY electric_charge_unit
SUBTYPE OF (derived_unit);
WHERE
   WR1:
      derive_dimensional_exponents(SELF) =
    dimensions_for_si_unit(si_unit_name.coulomb);
END_ENTITY;

ENTITY electric_current_measure_with_unit
SUBTYPE OF (measure_with_unit);
WHERE
   WR1:
       'ENGINEERING_PROPERTIES_SCHEMA.ELECTRIC_CURRENT_UNIT' IN
    TYPEOF(SELF\measure_with_unit.unit_component);
END_ENTITY;

ENTITY electric_current_unit
SUBTYPE OF (named_unit);
WHERE
   WR1:
      ((((((SELF\named_unit.dimensions.length_exponent = 0.00000) AND
    (SELF\named_unit.dimensions.mass_exponent = 0.00000)) AND
    (SELF\named_unit.dimensions.time_exponent = 0.00000)) AND
    (SELF\named_unit.dimensions.electric_current_exponent = 1.00000)) AND
    (SELF\named_unit.dimensions.thermodynamic_temperature_exponent = 0.00000))
    AND (SELF\named_unit.dimensions.amount_of_substance_exponent = 0.00000))
    AND (SELF\named_unit.dimensions.luminous_intensity_exponent = 0.00000);
END_ENTITY;

ENTITY electric_potential_measure_with_unit
SUBTYPE OF (measure_with_unit);
WHERE
   WR1:
       'ENGINEERING_PROPERTIES_SCHEMA.ELECTRIC_POTENTIAL_UNIT' IN
    TYPEOF(SELF\measure_with_unit.unit_component);
END_ENTITY;

ENTITY electric_potential_unit
SUBTYPE OF (derived_unit);
WHERE
   WR1:
      derive_dimensional_exponents(SELF) =
    dimensions_for_si_unit(si_unit_name.volt);
END_ENTITY;

ENTITY elementary_function
SUBTYPE OF (maths_function, generic_literal);
   func_id : elementary_function_enumerators;
END_ENTITY;

ENTITY elementary_space
SUBTYPE OF (maths_space, generic_literal);
```

```
   space_id : elementary_space_enumerators;
END_ENTITY;

ENTITY elementary_surface
     SUPERTYPE OF (ONEOF(plane, cylindrical_surface, conical_surface,
    spherical_surface))
SUBTYPE OF (surface);
   position : axis2_placement_3d;
END_ENTITY;

ENTITY ellipse
SUBTYPE OF (conic);
   semi_axis_1 : positive_length_measure;
   semi_axis_2 : positive_length_measure;
END_ENTITY;

ENTITY energy_measure_with_unit
SUBTYPE OF (measure_with_unit);
WHERE
   WR1:
       'ENGINEERING_PROPERTIES_SCHEMA.ENERGY_UNIT' IN
    TYPEOF(SELF\measure_with_unit.unit_component);
END_ENTITY;

ENTITY energy_unit
SUBTYPE OF (derived_unit);
WHERE
   WR1:
       derive_dimensional_exponents(SELF) =
    dimensions_for_si_unit(si_unit_name.joule);
END_ENTITY;

ENTITY environment;
   syntactic_representation : generic_variable;
   semantics : variable_semantics;
END_ENTITY;

ENTITY equals_expression
SUBTYPE OF (binary_boolean_expression);
END_ENTITY;

ENTITY event_occurrence;
   id : identifier;
   name : label;
   description : OPTIONAL text;
END_ENTITY;

ENTITY event_occurrence_assignment
ABSTRACT SUPERTYPE;
   assigned_event_occurrence : event_occurrence;
   role : event_occurrence_role;
END_ENTITY;

ENTITY event_occurrence_role;
   name : label;
   description : OPTIONAL text;
END_ENTITY;

ENTITY executed_action
```

```
SUBTYPE OF (action);
END_ENTITY;

ENTITY exp_function
SUBTYPE OF (unary_function_call);
END_ENTITY;

ENTITY expanded_uncertainty
SUBTYPE OF (standard_uncertainty);
   coverage_factor : REAL;
END_ENTITY;

ENTITY explicit_table_function
    ABSTRACT SUPERTYPE OF (ONEOF(listed_real_data, listed_integer_data,
    listed_logical_data, listed_string_data, listed_complex_number_data,
    listed_data, externally_listed_data, linearized_table_function,
    basic_sparse_matrix))
SUBTYPE OF (maths_function);
   index_base : zero_or_one;
   shape : LIST [1:?] OF positive_integer;
END_ENTITY;

ENTITY expression
   ABSTRACT SUPERTYPE OF (ONEOF(numeric_expression, boolean_expression,
string_expression))
SUBTYPE OF (generic_expression);
END_ENTITY;

ENTITY expression_denoted_function
SUBTYPE OF (maths_function, unary_generic_expression);
DERIVE
   expr : generic_expression := SELF\unary_generic_expression.operand;
WHERE
   WR1:
      schema_prefix + 'FUNCTION_SPACE' IN TYPEOF(values_space_of(expr));
END_ENTITY;

ENTITY extended_tuple_space
SUBTYPE OF (maths_space, generic_literal);
   base : product_space;
   extender : maths_space;
WHERE
   WR1:
      expression_is_constant(base) AND expression_is_constant(extender);
   WR2:
      no_cyclic_space_reference(SELF, []);
   WR3:
      extender <> the_empty_space;
END_ENTITY;

ENTITY extension
SUBTYPE OF (derived_shape_aspect);
WHERE
   WR1:
      SIZEOF(SELF\derived_shape_aspect.deriving_relationships) = 1;
END_ENTITY;

ENTITY external_identification_assignment
ABSTRACT SUPERTYPE
```

```
SUBTYPE OF (identification_assignment);
   source : external_source;
END_ENTITY;

ENTITY external_referent_assignment
ABSTRACT SUPERTYPE;
   assigned_name : label;
DERIVE
   role : object_role := get_role(SELF);
UNIQUE
   UR1 : assigned_name;
WHERE
   WR1:
       SIZEOF(USEDIN(SELF, 'ENGINEERING_PROPERTIES_SCHEMA.' +
    'ROLE_ASSOCIATION.ITEM_WITH_ROLE')) <= 1;
END_ENTITY;

ENTITY external_source;
   source_id : source_item;
DERIVE
   description : text := get_description_value(SELF);
WHERE
   WR1:
       SIZEOF(USEDIN(SELF, 'ENGINEERING_PROPERTIES_SCHEMA.' +
    'DESCRIPTION_ATTRIBUTE.DESCRIBED_ITEM')) <= 1;
END_ENTITY;

ENTITY external_source_relationship;
   name : label;
   description : OPTIONAL text;
   relating_source : external_source;
   related_source : external_source;
END_ENTITY;

ENTITY externally_defined_action_property
SUBTYPE OF (action_property, externally_defined_item);
END_ENTITY;

ENTITY externally_defined_class
SUBTYPE OF (class, externally_defined_item);
END_ENTITY;

ENTITY externally_defined_engineering_property
SUBTYPE OF (material_property, externally_defined_item);
END_ENTITY;

ENTITY externally_defined_item;
   item_id : source_item;
   source : external_source;
END_ENTITY;

ENTITY externally_defined_item_relationship;
   name : label;
   description : OPTIONAL text;
   relating_item : externally_defined_item;
   related_item : externally_defined_item;
END_ENTITY;

ENTITY externally_listed_data
```

```
    SUBTYPE OF (explicit_table_function, generic_literal,
    externally_defined_item);
   value_range : maths_space;
WHERE
   WR1:
      expression_is_constant(value_range);
END_ENTITY;

ENTITY finite_function
SUBTYPE OF (maths_function, generic_literal);
   pairs : SET [1:?] OF LIST [2:2] OF maths_value;
WHERE
   WR1:
      VALUE_UNIQUE(list_selected_components(pairs, 1));
END_ENTITY;

ENTITY finite_integer_interval
SUBTYPE OF (maths_space, generic_literal);
   min : INTEGER;
   max : INTEGER;
DERIVE
   size : positive_integer := max - min + 1;
WHERE
   WR1:
      min <= max;
END_ENTITY;

ENTITY finite_real_interval
SUBTYPE OF (maths_space, generic_literal);
   min : REAL;
   min_closure : open_closed;
   max : REAL;
   max_closure : open_closed;
WHERE
   WR1:
      min < max;
END_ENTITY;

ENTITY finite_space
SUBTYPE OF (maths_space, generic_literal);
   members : SET OF maths_value;
WHERE
   WR1:
      VALUE_UNIQUE(members);
   WR2:
        SIZEOF(QUERY (expr <* QUERY (member <* members|
    ('ENGINEERING_PROPERTIES_SCHEMA.GENERIC_EXPRESSION' IN TYPEOF(member)))|
    NOT expression_is_constant(expr))) = 0;
   WR3:
      no_cyclic_space_reference(SELF, []);
END_ENTITY;

ENTITY flatness_tolerance
SUBTYPE OF (geometric_tolerance);
WHERE
   WR1:
       NOT ('ENGINEERING_PROPERTIES_SCHEMA.' +
    'GEOMETRIC_TOLERANCE_WITH_DATUM_REFERENCE' IN TYPEOF(SELF));
END_ENTITY;
```

```
ENTITY force_measure_with_unit
SUBTYPE OF (measure_with_unit);
WHERE
   WR1:
       'ENGINEERING_PROPERTIES_SCHEMA.FORCE_UNIT' IN
    TYPEOF(SELF\measure_with_unit.unit_component);
END_ENTITY;

ENTITY force_unit
SUBTYPE OF (derived_unit);
WHERE
   WR1:
       derive_dimensional_exponents(SELF) =
    dimensions_for_si_unit(si_unit_name.newton);
END_ENTITY;

ENTITY format_function
SUBTYPE OF (string_expression, binary_generic_expression);
DERIVE
      value_to_format : generic_expression :=
    SELF\binary_generic_expression.operands[1];
          format_string : generic_expression :=
    SELF\binary_generic_expression.operands[2];
WHERE
   WR1:
      ('ENGINEERING_PROPERTIES_SCHEMA.NUMERIC_EXPRESSION' IN
    TYPEOF(value_to_format)) AND
    ('ENGINEERING_PROPERTIES_SCHEMA.STRING_EXPRESSION' IN
    TYPEOF(format_string));
END_ENTITY;

ENTITY founded_item;
END_ENTITY;

ENTITY free_variable_semantics
SUBTYPE OF (variable_semantics);
END_ENTITY;

ENTITY frequency_unit
SUBTYPE OF (derived_unit);
WHERE
   WR1:
       derive_dimensional_exponents(SELF) =
    dimensions_for_si_unit(si_unit_name.hertz);
END_ENTITY;

ENTITY function_application
SUBTYPE OF (multiple_arity_generic_expression);
   func : maths_function_select;
   arguments : LIST [1:?] OF maths_expression;
DERIVE
      SELF\multiple_arity_generic_expression.operands : LIST [2:?] OF
    generic_expression := [ convert_to_maths_function(func) ] +
    convert_to_operands(arguments);
WHERE
   WR1:
      function_applicability(func, arguments);
END_ENTITY;
```

```
ENTITY function_space
SUBTYPE OF (maths_space, generic_literal);
   domain_constraint : space_constraint_type;
   domain_argument : maths_space;
   range_constraint : space_constraint_type;
   range_argument : maths_space;
WHERE
   WR1:
      expression_is_constant(domain_argument) AND
    expression_is_constant(range_argument);
   WR2:
      (domain_argument <> the_empty_space) AND (range_argument <>
    the_empty_space);
   WR3:
      (domain_constraint <> sc_member) OR NOT member_of(the_empty_space,
    domain_argument);
   WR4:
      (range_constraint <> sc_member) OR NOT member_of(the_empty_space,
    range_argument);
   WR5:
      NOT (any_space_satisfies(domain_constraint, domain_argument) AND
    any_space_satisfies(range_constraint, range_argument));
END_ENTITY;

ENTITY functionally_defined_transformation;
   name : label;
   description : OPTIONAL text;
END_ENTITY;

ENTITY general_linear_function
SUBTYPE OF (maths_function, unary_generic_expression);
   SELF\unary_generic_expression.operand : maths_function;
   sum_index : one_or_two;
DERIVE
   mat : maths_function := SELF\unary_generic_expression.operand;
WHERE
   WR1:
      function_is_2d_table(mat);
   WR2:
      (space_dimension(mat.range) = 1) AND subspace_of_es(factor1(mat.range),
    es_numbers);
END_ENTITY;

ENTITY general_property;
   id : identifier;
   name : label;
   description : OPTIONAL text;
END_ENTITY;

ENTITY generic_expression
    ABSTRACT SUPERTYPE OF (ONEOF(simple_generic_expression,
    unary_generic_expression, binary_generic_expression,
    multiple_arity_generic_expression));
WHERE
   WR1:is_acyclic(SELF);
END_ENTITY;

ENTITY generic_literal
```

```
ABSTRACT SUPERTYPE
SUBTYPE OF (simple_generic_expression);
END_ENTITY;

ENTITY generic_variable
ABSTRACT SUPERTYPE
SUBTYPE OF (simple_generic_expression);
INVERSE
   interpretation : environment FOR syntactic_representation;
END_ENTITY;

ENTITY geometric_alignment
SUBTYPE OF (derived_shape_aspect);
WHERE
   WR1:
      SIZEOF(SELF\derived_shape_aspect.deriving_relationships) > 1;
END_ENTITY;

ENTITY geometric_intersection
SUBTYPE OF (derived_shape_aspect);
WHERE
   WR1:
      SIZEOF(SELF\derived_shape_aspect.deriving_relationships) > 1;
END_ENTITY;

ENTITY geometric_representation_context
SUBTYPE OF (representation_context);
   coordinate_space_dimension : dimension_count;
END_ENTITY;

ENTITY geometric_representation_item
    SUPERTYPE OF (ONEOF(point, direction, vector, placement,
    cartesian_transformation_operator, curve, surface, volume))
SUBTYPE OF (representation_item);
DERIVE
   dim : dimension_count := dimension_of(SELF);
WHERE
   WR1:
       SIZEOF(QUERY (using_rep <* using_representations(SELF)| NOT
    ('ENGINEERING_PROPERTIES_SCHEMA.GEOMETRIC_REPRESENTATION_CONTEXT' IN
    TYPEOF(using_rep.context_of_items)))) = 0;
END_ENTITY;

ENTITY geometric_tolerance;
   name : label;
   description : text;
   magnitude : measure_with_unit;
   toleranced_shape_aspect : shape_aspect;
WHERE
   WR1:
      ('NUMBER' IN TYPEOF(magnitude\measure_with_unit.value_component)) AND
    (magnitude\measure_with_unit.value_component >= 0.00000);
END_ENTITY;

ENTITY geometric_tolerance_relationship;
   name : label;
   description : text;
   relating_geometric_tolerance : geometric_tolerance;
   related_geometric_tolerance : geometric_tolerance;
```

```
END_ENTITY;

ENTITY geometric_tolerance_with_datum_reference
SUBTYPE OF (geometric_tolerance);
   datum_system : SET [1:?] OF datum_reference;
END_ENTITY;

ENTITY geometric_tolerance_with_defined_unit
SUBTYPE OF (geometric_tolerance);
   unit_size : measure_with_unit;
WHERE
   WR1:
       ('NUMBER' IN TYPEOF(unit_size\measure_with_unit.value_component)) AND
    unit_size\measure_with_unit.value_component > 0.00000);
END_ENTITY;

ENTITY global_uncertainty_assigned_context
SUBTYPE OF (representation_context);
   uncertainty : SET [1:?] OF uncertainty_measure_with_unit;
END_ENTITY;

ENTITY global_unit_assigned_context
SUBTYPE OF (representation_context);
   units : SET [1:?] OF unit;
END_ENTITY;

ENTITY group;
   name : label;
   description : OPTIONAL text;
DERIVE
   id : identifier := get_id_value(SELF);
WHERE
   WR1:
       SIZEOF(USEDIN(SELF, 'ENGINEERING_PROPERTIES_SCHEMA.' +
    'ID_ATTRIBUTE.IDENTIFIED_ITEM')) <= 1;
END_ENTITY;

ENTITY group_assignment
ABSTRACT SUPERTYPE;
   assigned_group : group;
DERIVE
   role : object_role := get_role(SELF);
WHERE
   WR1:
       SIZEOF(USEDIN(SELF, 'ENGINEERING_PROPERTIES_SCHEMA.' +
    'ROLE_ASSOCIATION.ITEM_WITH_ROLE')) <= 1;
END_ENTITY;

ENTITY group_relationship;
   name : label;
   description : OPTIONAL text;
   relating_group : group;
   related_group : group;
END_ENTITY;

ENTITY homogeneous_linear_function
SUBTYPE OF (maths_function, unary_generic_expression);
   SELF\unary_generic_expression.operand : maths_function;
   sum_index : one_or_two;
```

```
DERIVE
   mat : maths_function := SELF\unary_generic_expression.operand;
WHERE
   WR1:
      function_is_2d_table(mat);
   WR2:
     (space_dimension(mat.range) = 1) AND subspace_of_es(factor1(mat.range),
    es_numbers);
END_ENTITY;

ENTITY hyperbola
SUBTYPE OF (conic);
   semi_axis : positive_length_measure;
   semi_imag_axis : positive_length_measure;
END_ENTITY;

ENTITY id_attribute;
   attribute_value : identifier;
   identified_item : id_attribute_select;
END_ENTITY;

ENTITY identification_assignment
ABSTRACT SUPERTYPE;
   assigned_id : identifier;
   role : identification_role;
END_ENTITY;

ENTITY identification_assignment_relationship;
   name : label;
   description : OPTIONAL text;
   relating_identification_assignment : identification_assignment;
   related_identification_assignment : identification_assignment;
END_ENTITY;

ENTITY identification_role;
   name : label;
   description : OPTIONAL text;
END_ENTITY;

ENTITY illuminance_measure_with_unit
SUBTYPE OF (measure_with_unit);
WHERE
   WR1:
       'ENGINEERING_PROPERTIES_SCHEMA.ILLUMINANCE_UNIT' IN
    TYPEOF(SELF\measure_with_unit.unit_component);
END_ENTITY;

ENTITY illuminance_unit
SUBTYPE OF (derived_unit);
WHERE
   WR1:
       derive_dimensional_exponents(SELF) =
    dimensions_for_si_unit(si_unit_name.lux);
END_ENTITY;

ENTITY imported_curve_function
SUBTYPE OF (maths_function, generic_literal);
   geometry : curve;
   parametric_domain : tuple_space;
```

```
WHERE
   WR1:
      expression_is_constant(parametric_domain);
END_ENTITY;

ENTITY imported_point_function
SUBTYPE OF (maths_function, generic_literal);
   geometry : point;
END_ENTITY;

ENTITY imported_surface_function
SUBTYPE OF (maths_function, generic_literal);
   geometry : surface;
   parametric_domain : tuple_space;
WHERE
   WR1:
      expression_is_constant(parametric_domain);
END_ENTITY;

ENTITY imported_volume_function
SUBTYPE OF (maths_function, generic_literal);
   geometry : volume;
   parametric_domain : tuple_space;
WHERE
   WR1:
      expression_is_constant(parametric_domain);
END_ENTITY;

ENTITY index_expression
SUBTYPE OF (string_expression, binary_generic_expression);
DERIVE
   operand : generic_expression := SELF\binary_generic_expression.operands[1];
   index : generic_expression := SELF\binary_generic_expression.operands[2];
WHERE
   WR1:
       ('ENGINEERING_PROPERTIES_SCHEMA.STRING_EXPRESSION' IN TYPEOF(operand))
    AND ('ENGINEERING_PROPERTIES_SCHEMA.NUMERIC_EXPRESSION' IN TYPEOF(index));
   WR2:
      is_int_expr(index);
END_ENTITY;

ENTITY inductance_measure_with_unit
SUBTYPE OF (measure_with_unit);
WHERE
   WR1:
       'ENGINEERING_PROPERTIES_SCHEMA.INDUCTANCE_UNIT' IN
    TYPEOF(SELF\measure_with_unit.unit_component);
END_ENTITY;

ENTITY inductance_unit
SUBTYPE OF (derived_unit);
WHERE
   WR1:
       derive_dimensional_exponents(SELF) =
    dimensions_for_si_unit(si_unit_name.henry);
END_ENTITY;

ENTITY int_literal
SUBTYPE OF (literal_number);
```

```
   SELF\literal_number.the_value : INTEGER;
END_ENTITY;

ENTITY int_numeric_variable
SUBTYPE OF (numeric_variable);
END_ENTITY;

ENTITY int_value_function
SUBTYPE OF (value_function);
END_ENTITY;

ENTITY integer_defined_function
ABSTRACT SUPERTYPE
SUBTYPE OF (numeric_defined_function);
END_ENTITY;

ENTITY integer_interval_from_min
SUBTYPE OF (maths_space, generic_literal);
   min : INTEGER;
END_ENTITY;

ENTITY integer_interval_to_max
SUBTYPE OF (maths_space, generic_literal);
   max : INTEGER;
END_ENTITY;

ENTITY integer_tuple_literal
SUBTYPE OF (generic_literal);
   lit_value : LIST [1:?] OF INTEGER;
END_ENTITY;

ENTITY interval_expression
SUBTYPE OF (boolean_expression, multiple_arity_generic_expression);
DERIVE
      interval_low : generic_expression :=
    SELF\multiple_arity_generic_expression.operands[1];
         interval_item : generic_expression :=
    SELF\multiple_arity_generic_expression.operands[2];
         interval_high : generic_expression :=
    SELF\multiple_arity_generic_expression.operands[3];
WHERE
   WR1:
      (('ENGINEERING_PROPERTIES_SCHEMA.EXPRESSION' IN TYPEOF(interval_low))
    AND ('ENGINEERING_PROPERTIES_SCHEMA.EXPRESSION' IN TYPEOF(interval_item)))
    AND ('ENGINEERING_PROPERTIES_SCHEMA.EXPRESSION' IN TYPEOF(interval_high));
   WR2:
      (('ENGINEERING_PROPERTIES_SCHEMA.STRING_EXPRESSION' IN
    TYPEOF(SELF.interval_low)) AND
    ('ENGINEERING_PROPERTIES_SCHEMA.STRING_EXPRESSION' IN
    TYPEOF(SELF.interval_high))) AND
    ('ENGINEERING_PROPERTIES_SCHEMA.STRING_EXPRESSION' IN
    TYPEOF(SELF.interval_item)) OR
    (('ENGINEERING_PROPERTIES_SCHEMA.STRING_EXPRESSION' IN
    TYPEOF(SELF.interval_low)) AND
    ('ENGINEERING_PROPERTIES_SCHEMA.NUMERIC_EXPRESSION' IN
    TYPEOF(SELF.interval_item))) AND
    ('ENGINEERING_PROPERTIES_SCHEMA.NUMERIC_EXPRESSION' IN
    TYPEOF(SELF.interval_high));
END_ENTITY;
```

```
ENTITY item_defined_transformation;
   name : label;
   description : OPTIONAL text;
   transform_item_1 : representation_item;
   transform_item_2 : representation_item;
END_ENTITY;

ENTITY item_identified_representation_usage;
   name : label;
   description : OPTIONAL text;
   definition : represented_definition;
   used_representation : representation;
   identified_item : representation_item;
WHERE
   WR1:
      SELF.used_representation IN using_representations(SELF.identified_item);
END_ENTITY;

ENTITY language
SUBTYPE OF (group);
WHERE
   WR1:
       (SIZEOF(QUERY (ca <* USEDIN(SELF, 'ENGINEERING_PROPERTIES_SCHEMA.' +
    'CLASSIFICATION_ASSIGNMENT.' + 'ASSIGNED_CLASS')|
    ('ENGINEERING_PROPERTIES_SCHEMA.' + 'LANGUAGE_ASSIGNMENT' IN TYPEOF(ca))))
    > 0) OR (SIZEOF(QUERY (aca <* USEDIN(SELF,
    'ENGINEERING_PROPERTIES_SCHEMA.' + 'ATTRIBUTE_CLASSIFICATION_ASSIGNMENT.'
    + 'ASSIGNED_CLASS')| ('ENGINEERING_PROPERTIES_SCHEMA.' +
    'ATTRIBUTE_LANGUAGE_ASSIGNMENT' IN TYPEOF(aca)))) > 0);
END_ENTITY;

ENTITY language_assignment
SUBTYPE OF (classification_assignment);
   items : SET [1:?] OF language_item;
WHERE
   WR1:
       'ENGINEERING_PROPERTIES_SCHEMA.' + 'LANGUAGE' IN
    TYPEOF(SELF.assigned_class);
   WR2:
      SELF.role.name = 'language';
   WR3:
        SIZEOF(SELF.items) = SIZEOF(QUERY (i <* SELF.items|
    ('ENGINEERING_PROPERTIES_SCHEMA.' + 'REPRESENTATION' IN TYPEOF(i)) AND
    (i\representation.name = 'document content')));
END_ENTITY;

ENTITY length_function
SUBTYPE OF (numeric_expression, unary_generic_expression);
   SELF\unary_generic_expression.operand : string_expression;
END_ENTITY;

ENTITY length_measure_with_unit
SUBTYPE OF (measure_with_unit);
WHERE
   WR1:
       'ENGINEERING_PROPERTIES_SCHEMA.LENGTH_UNIT' IN
    TYPEOF(SELF\measure_with_unit.unit_component);
END_ENTITY;
```

```
ENTITY length_unit
SUBTYPE OF (named_unit);
WHERE
   WR1:
       ((((((SELF\named_unit.dimensions.length_exponent = 1.00000) AND
    (SELF\named_unit.dimensions.mass_exponent = 0.00000)) AND
    (SELF\named_unit.dimensions.time_exponent = 0.00000)) AND
    (SELF\named_unit.dimensions.electric_current_exponent = 0.00000)) AND
    (SELF\named_unit.dimensions.thermodynamic_temperature_exponent = 0.00000))
    AND (SELF\named_unit.dimensions.amount_of_substance_exponent = 0.00000))
    AND (SELF\named_unit.dimensions.luminous_intensity_exponent = 0.00000);
END_ENTITY;

ENTITY like_expression
SUBTYPE OF (comparison_expression);
WHERE
   WR1:
       ('ENGINEERING_PROPERTIES_SCHEMA.STRING_EXPRESSION' IN
    TYPEOF(SELF\comparison_expression.operands[1])) AND
    ('ENGINEERING_PROPERTIES_SCHEMA.STRING_EXPRESSION' IN
    TYPEOF(SELF\comparison_expression.operands[2]));
END_ENTITY;

ENTITY limits_and_fits;
   form_variance : label;
   zone_variance : label;
   grade : label;
   source : text;
END_ENTITY;

ENTITY line
SUBTYPE OF (curve);
   pnt : cartesian_point;
   dir : vector;
WHERE
   WR1:
      dir.dim = pnt.dim;
END_ENTITY;

ENTITY linearized_table_function
    SUPERTYPE OF (ONEOF(standard_table_function, regular_table_function,
    triangular_matrix, symmetric_matrix, banded_matrix))
SUBTYPE OF (explicit_table_function, unary_generic_expression);
   SELF\unary_generic_expression.operand : maths_function;
   first : INTEGER;
DERIVE
   source : maths_function := SELF\unary_generic_expression.operand;
WHERE
   WR1:
      function_is_1d_array(source);
   WR2:
      member_of(first, source.domain);
END_ENTITY;

ENTITY listed_complex_number_data
SUBTYPE OF (explicit_table_function, generic_literal);
   values : LIST [2:?] OF REAL;
DERIVE
```

```
       SELF\explicit_table_function.shape : LIST [1:?] OF positive_integer :=
    [ SIZEOF(values) DIV 2 ];
WHERE
   WR1:
      NOT ODD(SIZEOF(values));
END_ENTITY;

ENTITY listed_data
SUBTYPE OF (explicit_table_function, generic_literal);
   values : LIST [1:?] OF maths_value;
   value_range : maths_space;
DERIVE
     SELF\explicit_table_function.shape : LIST [1:?] OF positive_integer :=
    [ SIZEOF(values) ];
WHERE
   WR1:
      expression_is_constant(value_range);
   WR2:
      SIZEOF(QUERY (val <* values| NOT member_of(val, value_range))) = 0;
END_ENTITY;

ENTITY listed_integer_data
SUBTYPE OF (explicit_table_function, generic_literal);
   values : LIST [1:?] OF INTEGER;
DERIVE
     SELF\explicit_table_function.shape : LIST [1:?] OF positive_integer :=
    [ SIZEOF(values) ];
END_ENTITY;

ENTITY listed_logical_data
SUBTYPE OF (explicit_table_function, generic_literal);
   values : LIST [1:?] OF LOGICAL;
DERIVE
       SELF\explicit_table_function.shape : LIST [1:?] OF positive_integer :=
    [ SIZEOF(values) ];
END_ENTITY;

ENTITY listed_product_space
SUBTYPE OF (maths_space, generic_literal);
   factors : LIST OF maths_space;
WHERE
   WR1:
      SIZEOF(QUERY (space <* factors| NOT expression_is_constant(space))) = 0;
   WR2:
      no_cyclic_space_reference(SELF, []);
   WR3:
      NOT (the_empty_space IN factors);
END_ENTITY;

ENTITY listed_real_data
SUBTYPE OF (explicit_table_function, generic_literal);
   values : LIST [1:?] OF REAL;
DERIVE
       SELF\explicit_table_function.shape : LIST [1:?] OF positive_integer :=
    [ SIZEOF(values) ];
END_ENTITY;

ENTITY listed_string_data
SUBTYPE OF (explicit_table_function, generic_literal);
```

```
   values : LIST [1:?] OF STRING;
DERIVE
       SELF\explicit_table_function.shape : LIST [1:?] OF positive_integer :=
    [ SIZEOF(values) ];
END_ENTITY;

ENTITY literal_number
ABSTRACT SUPERTYPE OF (ONEOF(int_literal, real_literal))
SUBTYPE OF (simple_numeric_expression, generic_literal);
   the_value : NUMBER;
END_ENTITY;

ENTITY local_time;
   hour_component : hour_in_day;
   minute_component : OPTIONAL minute_in_hour;
   second_component : OPTIONAL second_in_minute;
   zone : coordinated_universal_time_offset;
WHERE
   WR1:
      valid_time(SELF);
END_ENTITY;

ENTITY location;
   id : identifier;
   name : label;
   description : OPTIONAL text;
END_ENTITY;

ENTITY location_assignment
ABSTRACT SUPERTYPE;
   id : identifier;
   name : label;
   description : OPTIONAL text;
   assigned_location : location;
   role : location_role;
END_ENTITY;

ENTITY location_relationship;
   id : identifier;
   name : label;
   description : OPTIONAL text;
   relating_location : location;
   related_location : location;
END_ENTITY;

ENTITY location_representation_assignment
ABSTRACT SUPERTYPE;
   id : identifier;
   name : label;
   description : OPTIONAL text;
   represented_location : location;
   role : location_representation_role;
END_ENTITY;

ENTITY location_representation_role;
   id : identifier;
   name : label;
   description : OPTIONAL text;
END_ENTITY;
```

```
ENTITY location_role;
   id : identifier;
   name : label;
   description : OPTIONAL text;
END_ENTITY;

ENTITY log10_function
SUBTYPE OF (unary_function_call);
END_ENTITY;

ENTITY log2_function
SUBTYPE OF (unary_function_call);
END_ENTITY;

ENTITY log_function
SUBTYPE OF (unary_function_call);
END_ENTITY;

ENTITY logical_literal
SUBTYPE OF (generic_literal);
   lit_value : LOGICAL;
END_ENTITY;

ENTITY luminous_flux_measure_with_unit
SUBTYPE OF (measure_with_unit);
WHERE
   WR1:
       'ENGINEERING_PROPERTIES_SCHEMA.LUMINOUS_FLUX_UNIT' IN
    TYPEOF(SELF\measure_with_unit.unit_component);
END_ENTITY;

ENTITY luminous_flux_unit
SUBTYPE OF (named_unit);
WHERE
   WR1:
       derive_dimensional_exponents(SELF) =
    dimensions_for_si_unit(si_unit_name.lumen);
END_ENTITY;

ENTITY luminous_intensity_measure_with_unit
SUBTYPE OF (measure_with_unit);
WHERE
   WR1:
       'ENGINEERING_PROPERTIES_SCHEMA.LUMINOUS_INTENSITY_UNIT' IN
    TYPEOF(SELF\measure_with_unit.unit_component);
END_ENTITY;

ENTITY luminous_intensity_unit
SUBTYPE OF (named_unit);
WHERE
   WR1:
       ((((((SELF\named_unit.dimensions.length_exponent = 0.00000) AND
    (SELF\named_unit.dimensions.mass_exponent = 0.00000)) AND
    (SELF\named_unit.dimensions.time_exponent = 0.00000)) AND
    (SELF\named_unit.dimensions.electric_current_exponent = 0.00000)) AND
    (SELF\named_unit.dimensions.thermodynamic_temperature_exponent = 0.00000))
    AND (SELF\named_unit.dimensions.amount_of_substance_exponent = 0.00000))
    AND (SELF\named_unit.dimensions.luminous_intensity_exponent = 1.00000);
```

```
END_ENTITY;

ENTITY magnetic_flux_density_measure_with_unit
SUBTYPE OF (measure_with_unit);
WHERE
   WR1:
       'ENGINEERING_PROPERTIES_SCHEMA.MAGNETIC_FLUX_DENSITY_UNIT' IN
    TYPEOF(SELF\measure_with_unit.unit_component);
END_ENTITY;

ENTITY magnetic_flux_density_unit
SUBTYPE OF (derived_unit);
WHERE
   WR1:
       derive_dimensional_exponents(SELF) =
    dimensions_for_si_unit(si_unit_name.tesla);
END_ENTITY;

ENTITY magnetic_flux_measure_with_unit
SUBTYPE OF (measure_with_unit);
WHERE
   WR1:
       'ENGINEERING_PROPERTIES_SCHEMA.MAGNETIC_FLUX_UNIT' IN
    TYPEOF(SELF\measure_with_unit.unit_component);
END_ENTITY;

ENTITY magnetic_flux_unit
SUBTYPE OF (derived_unit);
WHERE
   WR1:
       derive_dimensional_exponents(SELF) =
    dimensions_for_si_unit(si_unit_name.weber);
END_ENTITY;

ENTITY mapped_item
SUBTYPE OF (representation_item);
   mapping_source : representation_map;
   mapping_target : representation_item;
WHERE
   WR1:
      acyclic_mapped_representation(using_representations(SELF), [ SELF ]);
END_ENTITY;

ENTITY mass_measure_with_unit
SUBTYPE OF (measure_with_unit);
WHERE
   WR1:
       'ENGINEERING_PROPERTIES_SCHEMA.MASS_UNIT' IN
    TYPEOF(SELF\measure_with_unit.unit_component);
END_ENTITY;

ENTITY mass_unit
SUBTYPE OF (named_unit);
WHERE
   WR1:
       ((((((SELF\named_unit.dimensions.length_exponent = 0.00000) AND
    (SELF\named_unit.dimensions.mass_exponent = 1.00000)) AND
    (SELF\named_unit.dimensions.time_exponent = 0.00000)) AND
    (SELF\named_unit.dimensions.electric_current_exponent = 0.00000)) AND
```

```
    (SELF\named_unit.dimensions.thermodynamic_temperature_exponent = 0.00000))
    AND (SELF\named_unit.dimensions.amount_of_substance_exponent = 0.00000))
    AND (SELF\named_unit.dimensions.luminous_intensity_exponent = 0.00000);
END_ENTITY;

ENTITY material_designation;
   name : label;
   definitions : SET [1:?] OF characterized_definition;
END_ENTITY;

ENTITY material_designation_characterization;
   name : label;
   description : text;
   designation : material_designation;
   property : characterized_material_property;
END_ENTITY;

ENTITY material_property
SUBTYPE OF (property_definition);
UNIQUE
   UR1 : SELF\property_definition.name, SELF\property_definition.definition;
WHERE
   WR1:
       ('ENGINEERING_PROPERTIES_SCHEMA.CHARACTERIZED_OBJECT' IN
    TYPEOF(SELF\property_definition.definition)) OR
    (SIZEOF(bag_to_set(USEDIN(SELF, 'ENGINEERING_PROPERTIES_SCHEMA.' +
    'PROPERTY_DEFINITION_REPRESENTATION.DEFINITION')) - QUERY (temp <*
    bag_to_set(USEDIN(SELF, 'ENGINEERING_PROPERTIES_SCHEMA.' +
    'PROPERTY_DEFINITION_REPRESENTATION.DEFINITION'))|
    ('ENGINEERING_PROPERTIES_SCHEMA.' + 'MATERIAL_PROPERTY_REPRESENTATION' IN
    TYPEOF(temp)))) = 0);
END_ENTITY;

ENTITY material_property_representation
SUBTYPE OF (property_definition_representation);
   dependent_environment : data_environment;
END_ENTITY;

ENTITY mathematical_description;
   described : maths_expression;
   describing : STRING;
   encoding : label;
END_ENTITY;

ENTITY maths_boolean_variable
SUBTYPE OF (maths_variable, boolean_variable);
WHERE
   WR1:
      subspace_of_es(SELF\maths_variable.values_space, es_booleans);
END_ENTITY;

ENTITY maths_enum_literal
SUBTYPE OF (generic_literal);
   lit_value : maths_enum_atom;
END_ENTITY;

ENTITY maths_function
    ABSTRACT SUPERTYPE OF (ONEOF(finite_function, constant_function,
    selector_function, elementary_function, restriction_function,
```

```
    repackaging_function, reindexed_array_function, series_composed_function,
    parallel_composed_function, explicit_table_function,
    homogeneous_linear_function, general_linear_function, b_spline_basis,
    b_spline_function, rationalize_function, partial_derivative_function,
    definite_integral_function, abstracted_expression_function,
    expression_denoted_function, imported_point_function,
    imported_curve_function, imported_surface_function,
    imported_volume_function, application_defined_function))
SUBTYPE OF (generic_expression);
DERIVE
   domain : tuple_space := derive_function_domain(SELF);
   range : tuple_space := derive_function_range(SELF);
END_ENTITY;

ENTITY maths_integer_variable
SUBTYPE OF (maths_variable, int_numeric_variable);
WHERE
   WR1:
      subspace_of_es(SELF\maths_variable.values_space, es_integers);
END_ENTITY;

ENTITY maths_real_variable
SUBTYPE OF (maths_variable, real_numeric_variable);
WHERE
   WR1:
      subspace_of_es(SELF\maths_variable.values_space, es_reals);
END_ENTITY;

ENTITY maths_space
    ABSTRACT SUPERTYPE OF (ONEOF(elementary_space, finite_integer_interval,
    integer_interval_from_min, integer_interval_to_max, finite_real_interval,
    real_interval_from_min, real_interval_to_max,
    cartesian_complex_number_region, polar_complex_number_region, finite_space,
    uniform_product_space, listed_product_space, extended_tuple_space,
    function_space))
       SUBTYPE OF (generic_expression);
END_ENTITY;

ENTITY maths_string_variable
SUBTYPE OF (maths_variable, string_variable);
WHERE
   WR1:
      subspace_of_es(SELF\maths_variable.values_space, es_strings);
END_ENTITY;

ENTITY maths_tuple_literal
SUBTYPE OF (generic_literal);
   lit_value : LIST OF maths_value;
END_ENTITY;

ENTITY maths_value_qualification;
   name : label;
   description : text;
   qualified_maths_value : maths_value_with_unit;
   qualifiers : SET [1:?] OF value_qualifier;
WHERE
   WR1:
```

```
         SIZEOF(QUERY (temp <* qualifiers|
       ('ENGINEERING_PROPERTIES_SCHEMA.PRECISION_QUALIFIER' IN TYPEOF(temp)))) <
       2;
   END_ENTITY;

   ENTITY maths_value_representation_item
   SUBTYPE OF (representation_item, maths_value_with_unit);
   END_ENTITY;

   ENTITY maths_value_with_unit;
      value_component : maths_value;
      unit_component : unit;

   END_ENTITY;

   ENTITY maths_variable
   SUBTYPE OF (generic_variable);
      values_space : maths_space;
      name : label;
   WHERE
      WR1:
         expression_is_constant(values_space);
   END_ENTITY;

   ENTITY maximum_function
   SUBTYPE OF (multiple_arity_function_call);
   END_ENTITY;

   ENTITY measure_qualification;
      name : label;
      description : text;
      qualified_measure : measure_with_unit;
      qualifiers : SET [1:?] OF value_qualifier;
   WHERE
      WR1:
         SIZEOF(QUERY (temp <* qualifiers|
('ENGINEERING_PROPERTIES_SCHEMA.PRECISION_QUALIFIER' IN TYPEOF(temp)))) < 2;
   END_ENTITY;

   ENTITY measure_representation_item
   SUBTYPE OF (representation_item, measure_with_unit);
   END_ENTITY;

   ENTITY measure_with_unit
       SUPERTYPE OF (ONEOF(length_measure_with_unit, mass_measure_with_unit,
       time_measure_with_unit, electric_current_measure_with_unit,
       thermodynamic_temperature_measure_with_unit,
       celsius_temperature_measure_with_unit,
       amount_of_substance_measure_with_unit,
       luminous_intensity_measure_with_unit, plane_angle_measure_with_unit,
       solid_angle_measure_with_unit, area_measure_with_unit,
       volume_measure_with_unit, ratio_measure_with_unit,
       acceleration_measure_with_unit, capacitance_measure_with_unit,
       electric_charge_measure_with_unit, conductance_measure_with_unit,
       electric_potential_measure_with_unit, energy_measure_with_unit,
       magnetic_flux_density_measure_with_unit, force_measure_with_unit,
       illuminance_measure_with_unit, inductance_measure_with_unit,
       luminous_flux_measure_with_unit, magnetic_flux_measure_with_unit,
       power_measure_with_unit, pressure_measure_with_unit,
```

```
    resistance_measure_with_unit, velocity_measure_with_unit,
    absorbed_dose_measure_with_unit, radioactivity_measure_with_unit,
    dose_equivalent_measure_with_unit));
   value_component : measure_value;
   unit_component : unit;
WHERE
   WR1:
      valid_units(SELF);
END_ENTITY;

ENTITY minimum_function
SUBTYPE OF (multiple_arity_function_call);
END_ENTITY;

ENTITY minus_expression
SUBTYPE OF (binary_numeric_expression);
END_ENTITY;

ENTITY minus_function
SUBTYPE OF (unary_function_call);
END_ENTITY;

ENTITY mod_expression
SUBTYPE OF (binary_numeric_expression);
END_ENTITY;

ENTITY modified_geometric_tolerance
SUBTYPE OF (geometric_tolerance);
   modifier : limit_condition;
END_ENTITY;

ENTITY mult_expression
SUBTYPE OF (multiple_arity_numeric_expression);
END_ENTITY;

ENTITY multi_language_attribute_assignment
SUBTYPE OF (attribute_value_assignment);
   items : SET [1:?] OF multi_language_attribute_item;
DERIVE
   language : label := get_multi_language(SELF);
WHERE
   WR1:
      SELF\attribute_value_assignment.role.name = 'alternate language';
   WR2:
      (SIZEOF(USEDIN(SELF.items[1],
    'ENGINEERING_PROPERTIES_SCHEMA.ATTRIBUTE_LANGUAGE_ASSIGNMENT.ITEMS')) = 1)
    AND (SIZEOF(QUERY (ala <* USEDIN(SELF.items[1],
    'ENGINEERING_PROPERTIES_SCHEMA.' + 'ATTRIBUTE_LANGUAGE_ASSIGNMENT.' +
    'ITEMS')| (ala.attribute_name = 'attribute_value'))) = 1);
END_ENTITY;

ENTITY multiple_arity_boolean_expression
ABSTRACT SUPERTYPE OF (ONEOF(and_expression, or_expression))
SUBTYPE OF (boolean_expression, multiple_arity_generic_expression);
    SELF\multiple_arity_generic_expression.operands : LIST [2:?] OF
    boolean_expression;
END_ENTITY;

ENTITY multiple_arity_function_call
```

```
ABSTRACT SUPERTYPE OF (ONEOF(maximum_function, minimum_function))
SUBTYPE OF (multiple_arity_numeric_expression);
END_ENTITY;

ENTITY multiple_arity_generic_expression
ABSTRACT SUPERTYPE
SUBTYPE OF (generic_expression);
   operands : LIST [2:?] OF generic_expression;
END_ENTITY;

ENTITY multiple_arity_numeric_expression
    ABSTRACT SUPERTYPE OF (ONEOF(plus_expression, mult_expression,
    multiple_arity_function_call))
SUBTYPE OF (numeric_expression, multiple_arity_generic_expression);
    SELF\multiple_arity_generic_expression.operands : LIST [2:?] OF
    numeric_expression;
END_ENTITY;

ENTITY name_assignment
ABSTRACT SUPERTYPE;
   assigned_name : label;
DERIVE
   role : object_role := get_role(SELF);
WHERE
   WR1:
      SIZEOF(USEDIN(SELF, 'ENGINEERING_PROPERTIES_SCHEMA.' +
    'ROLE_ASSOCIATION.ITEM_WITH_ROLE')) <= 1;
END_ENTITY;

ENTITY name_attribute;
   attribute_value : label;
   named_item : name_attribute_select;
END_ENTITY;

ENTITY named_unit
    SUPERTYPE OF (ONEOF(si_unit, conversion_based_unit,
    context_dependent_unit) ANDOR ONEOF(length_unit, mass_unit, time_unit,
    electric_current_unit, thermodynamic_temperature_unit,
    amount_of_substance_unit, luminous_flux_unit, luminous_intensity_unit,
    plane_angle_unit, solid_angle_unit, ratio_unit));
          dimensions : dimensional_exponents;
END_ENTITY;

ENTITY not_expression
SUBTYPE OF (unary_boolean_expression);
   SELF\unary_generic_expression.operand : boolean_expression;
END_ENTITY;

ENTITY numeric_defined_function
ABSTRACT SUPERTYPE OF (ONEOF(integer_defined_function, real_defined_function))
SUBTYPE OF (numeric_expression, defined_function);
END_ENTITY;

ENTITY numeric_expression
       ABSTRACT SUPERTYPE OF (ONEOF(simple_numeric_expression,
    unary_numeric_expression, binary_numeric_expression,
    multiple_arity_numeric_expression, length_function, value_function,
    numeric_defined_function))
SUBTYPE OF (expression);
```

```
DERIVE
   is_int : BOOLEAN := is_int_expr(SELF);
   sql_mappable : BOOLEAN := is_SQL_mappable(SELF);
END_ENTITY;

ENTITY numeric_variable
SUPERTYPE OF (ONEOF(int_numeric_variable, real_numeric_variable))
SUBTYPE OF (simple_numeric_expression, variable);
WHERE
   WR1:
       ('ENGINEERING_PROPERTIES_SCHEMA.INT_NUMERIC_VARIABLE' IN TYPEOF(SELF))
    OR ('ENGINEERING_PROPERTIES_SCHEMA.REAL_NUMERIC_VARIABLE' IN
    TYPEOF(SELF));
END_ENTITY;

ENTITY object_role;
   name : label;
   description : OPTIONAL text;
END_ENTITY;

ENTITY odd_function
SUBTYPE OF (unary_boolean_expression);
   SELF\unary_generic_expression.operand : numeric_expression;
WHERE
   WR1:
      is_int_expr(SELF);
END_ENTITY;

ENTITY or_expression
SUBTYPE OF (multiple_arity_boolean_expression);
END_ENTITY;

ENTITY ordinal_date
SUBTYPE OF (date);
   day_component : day_in_year_number;
WHERE
   WR1:
       NOT leap_year(SELF.year_component) AND ((1 <= day_component) AND
    (day_component <= 365)) OR leap_year(SELF.year_component) AND ((1 <=
    day_component) AND (day_component <= 366));
END_ENTITY;

ENTITY organization;
   id : OPTIONAL identifier;
   name : label;
   description : OPTIONAL text;
END_ENTITY;

ENTITY organization_assignment
ABSTRACT SUPERTYPE;
   assigned_organization : organization;
   role : organization_role;
END_ENTITY;

ENTITY organization_relationship;
   name : label;
   description : OPTIONAL text;
   relating_organization : organization;
   related_organization : organization;
```

```
END_ENTITY;

ENTITY organization_role;
   name : label;
DERIVE
   description : text := get_description_value(SELF);
WHERE
   WR1:
       SIZEOF(USEDIN(SELF, 'ENGINEERING_PROPERTIES_SCHEMA.' +
    'DESCRIPTION_ATTRIBUTE.DESCRIBED_ITEM')) <= 1;
END_ENTITY;

ENTITY organizational_address
SUBTYPE OF (address);
   organizations : SET [1:?] OF organization;
   description : OPTIONAL text;
END_ENTITY;

ENTITY organizational_project;
   name : label;
   description : OPTIONAL text;
   responsible_organizations : SET [1:?] OF organization;
DERIVE
   id : identifier := get_id_value(SELF);
WHERE
   WR1:
       SIZEOF(USEDIN(SELF, 'ENGINEERING_PROPERTIES_SCHEMA.' +
    'ID_ATTRIBUTE.IDENTIFIED_ITEM')) <= 1;
END_ENTITY;

ENTITY organizational_project_assignment
ABSTRACT SUPERTYPE;
   assigned_organizational_project : organizational_project;
   role : organizational_project_role;
END_ENTITY;

ENTITY organizational_project_relationship;
   name : label;
   description : OPTIONAL text;
   relating_organizational_project : organizational_project;
   related_organizational_project : organizational_project;
END_ENTITY;

ENTITY organizational_project_role;
   name : label;
   description : OPTIONAL text;
END_ENTITY;

ENTITY oriented_surface
SUBTYPE OF (surface);
   orientation : BOOLEAN;
END_ENTITY;

ENTITY parabola
SUBTYPE OF (conic);
   focal_dist : length_measure;
WHERE
   WR1:
      focal_dist <> 0.00000;
```

```
END_ENTITY;

ENTITY parallel_composed_function
SUBTYPE OF (maths_function, multiple_arity_generic_expression);
   source_of_domain : maths_space_or_function;
   prep_functions : LIST [1:?] OF maths_function;
   final_function : maths_function_select;
DERIVE
      SELF\multiple_arity_generic_expression.operands : LIST [2:?] OF
    generic_expression := convert_to_operands_prcmfn(source_of_domain,
    prep_functions, final_function);
WHERE
   WR1:
      no_cyclic_domain_reference(source_of_domain, [ SELF ]);
   WR2:
      expression_is_constant(domain_from(source_of_domain));
   WR3:
       parallel_composed_function_domain_check(domain_from(source_of_domain),
    prep_functions);
   WR4:
    parallel_composed_function_composability_check(prep_functions,
final_function);
END_ENTITY;

ENTITY parallel_offset
SUBTYPE OF (derived_shape_aspect);
   offset : measure_with_unit;
WHERE
   WR1:
      SIZEOF(SELF\derived_shape_aspect.deriving_relationships) = 1;
END_ENTITY;

ENTITY parallelism_tolerance
SUBTYPE OF (geometric_tolerance_with_datum_reference);
WHERE
   WR1:
      SIZEOF(SELF\geometric_tolerance_with_datum_reference.datum_system) < 3;
END_ENTITY;

ENTITY parametric_representation_context
SUBTYPE OF (representation_context);
END_ENTITY;

ENTITY partial_derivative_expression
SUBTYPE OF (unary_generic_expression);
   d_variables : LIST [1:?] OF maths_variable;
   extension : extension_options;
DERIVE
   derivand : generic_expression := SELF\unary_generic_expression.operand;
WHERE
   WR1:
      has_values_space(derivand);
   WR2:
      space_is_continuum(values_space_of(derivand));
   WR3:
       SIZEOF(QUERY (vbl <* d_variables| NOT subspace_of(values_space_of(vbl),
    the_reals) AND NOT subspace_of(values_space_of(vbl),
    the_complex_numbers))) = 0;
END_ENTITY;
```

```
ENTITY partial_derivative_function
SUBTYPE OF (maths_function, unary_generic_expression);
   SELF\unary_generic_expression.operand : maths_function;
   d_variables : LIST [1:?] OF input_selector;
   extension : extension_options;
DERIVE
   derivand : maths_function := SELF\unary_generic_expression.operand;
WHERE
   WR1:
      space_is_continuum(derivand.range);
   WR2:
      partial_derivative_check(derivand.domain, d_variables);
END_ENTITY;

ENTITY perpendicular_to
SUBTYPE OF (derived_shape_aspect);
WHERE
   WR1:
      SIZEOF(SELF\derived_shape_aspect.deriving_relationships) = 1;
END_ENTITY;

ENTITY perpendicularity_tolerance
SUBTYPE OF (geometric_tolerance_with_datum_reference);
WHERE
   WR1:
      SIZEOF(SELF\geometric_tolerance_with_datum_reference.datum_system) <= 3;
END_ENTITY;

ENTITY person;
   id : identifier;
   last_name : OPTIONAL label;
   first_name : OPTIONAL label;
   middle_names : OPTIONAL LIST [1:?] OF label;
   prefix_titles : OPTIONAL LIST [1:?] OF label;
   suffix_titles : OPTIONAL LIST [1:?] OF label;
WHERE
   WR1:
      EXISTS(last_name) OR EXISTS(first_name);
END_ENTITY;

ENTITY person_and_organization;
   the_person : person;
   the_organization : organization;
DERIVE
   name : label := get_name_value(SELF);
   description : text := get_description_value(SELF);
WHERE
   WR1:
       SIZEOF(USEDIN(SELF, 'ENGINEERING_PROPERTIES_SCHEMA.' +
    'NAME_ATTRIBUTE.NAMED_ITEM')) <= 1;
   WR2:
       SIZEOF(USEDIN(SELF, 'ENGINEERING_PROPERTIES_SCHEMA.' +
    'DESCRIPTION_ATTRIBUTE.DESCRIBED_ITEM')) <= 1;
END_ENTITY;

ENTITY person_and_organization_assignment
ABSTRACT SUPERTYPE;
   assigned_person_and_organization : person_and_organization;
```

```
   role : person_and_organization_role;
END_ENTITY;

ENTITY person_and_organization_role;
   name : label;
DERIVE
   description : text := get_description_value(SELF);
WHERE
   WR1:
       SIZEOF(USEDIN(SELF, 'ENGINEERING_PROPERTIES_SCHEMA.' +
    'DESCRIPTION_ATTRIBUTE.DESCRIBED_ITEM')) <= 1;
END_ENTITY;

ENTITY person_assignment
ABSTRACT SUPERTYPE;
   assigned_person : person;
   role : person_role;
END_ENTITY;

ENTITY person_role;
   name : label;
DERIVE
   description : text := get_description_value(SELF);
WHERE
   WR1:
       SIZEOF(USEDIN(SELF, 'ENGINEERING_PROPERTIES_SCHEMA.' +
    'DESCRIPTION_ATTRIBUTE.DESCRIBED_ITEM')) <= 1;
END_ENTITY;

ENTITY person_type;
   id : identifier;
   name : label;
   description : OPTIONAL text;
END_ENTITY;

ENTITY person_type_definition;
   id : identifier;
   name : label;
   description : OPTIONAL text;
   formation : person_type_definition_formation;
END_ENTITY;

ENTITY person_type_definition_formation;
   id : identifier;
   name : label;
   description : OPTIONAL text;
   of_person_type : person_type;
END_ENTITY;

ENTITY person_type_definition_relationship;
   id : identifier;
   name : label;
   description : OPTIONAL text;
   relating_person_type_definition : person_type_definition;
   related_person_type_definition : person_type_definition;
END_ENTITY;

ENTITY personal_address
SUBTYPE OF (address);
```

```
   people : SET [1:?] OF person;
   description : OPTIONAL text;
END_ENTITY;

ENTITY placement
SUPERTYPE OF (ONEOF(axis1_placement, axis2_placement_2d, axis2_placement_3d))
SUBTYPE OF (geometric_representation_item);
   location : cartesian_point;
END_ENTITY;

ENTITY plane
SUBTYPE OF (elementary_surface);
END_ENTITY;

ENTITY plane_angle_measure_with_unit
SUBTYPE OF (measure_with_unit);
WHERE
   WR1:
       'ENGINEERING_PROPERTIES_SCHEMA.PLANE_ANGLE_UNIT' IN
    TYPEOF(SELF\measure_with_unit.unit_component);
END_ENTITY;

ENTITY plane_angle_unit
SUBTYPE OF (named_unit);
WHERE
   WR1:
       ((((((SELF\named_unit.dimensions.length_exponent = 0.00000) AND
    (SELF\named_unit.dimensions.mass_exponent = 0.00000)) AND
    (SELF\named_unit.dimensions.time_exponent = 0.00000)) AND
    (SELF\named_unit.dimensions.electric_current_exponent = 0.00000)) AND
    (SELF\named_unit.dimensions.thermodynamic_temperature_exponent = 0.00000))
    AND (SELF\named_unit.dimensions.amount_of_substance_exponent = 0.00000))
    AND (SELF\named_unit.dimensions.luminous_intensity_exponent = 0.00000);
END_ENTITY;

ENTITY plus_expression
SUBTYPE OF (multiple_arity_numeric_expression);
END_ENTITY;

ENTITY plus_minus_tolerance;
   range : tolerance_method_definition;
   toleranced_dimension : dimensional_characteristic;
UNIQUE
   UR1 : toleranced_dimension;
END_ENTITY;

ENTITY point
   SUPERTYPE OF (ONEOF(cartesian_point, point_on_curve, point_on_surface,
point_in_volume))
SUBTYPE OF (geometric_representation_item);
END_ENTITY;

ENTITY point_in_volume
SUBTYPE OF (point);
   basis_volume : volume;
   point_parameter_u : parameter_value;
   point_parameter_v : parameter_value;
   point_parameter_w : parameter_value;
END_ENTITY;
```

```
ENTITY point_on_curve
SUBTYPE OF (point);
   basis_curve : curve;
   point_parameter : parameter_value;
END_ENTITY;

ENTITY point_on_surface
SUBTYPE OF (point);
   basis_surface : surface;
   point_parameter_u : parameter_value;
   point_parameter_v : parameter_value;
END_ENTITY;

ENTITY polar_complex_number_region
SUBTYPE OF (maths_space, generic_literal);
   centre : complex_number_literal;
   distance_constraint : real_interval;
   direction_constraint : finite_real_interval;
WHERE
   WR1:
       min_exists(distance_constraint) AND (real_min(distance_constraint) >=
    0.00000);
   WR2:
       (-3.14159 <= direction_constraint.min) AND (direction_constraint.min <
    3.14159);
   WR3:
        direction_constraint.max - direction_constraint.min <= 2.00000 *
    3.14159;
   WR4:
        (direction_constraint.max - direction_constraint.min < 2.00000 *
    3.14159) OR (direction_constraint.min_closure = open);
   WR5:
        ((direction_constraint.max - direction_constraint.min < 2.00000 *
    3.14159) OR (direction_constraint.max_closure = open)) OR
    (direction_constraint.min = -3.14159);
   WR6:
       (((real_min(distance_constraint) > 0.00000) OR
    max_exists(distance_constraint)) OR (direction_constraint.max -
    direction_constraint.min < 2.00000 * 3.14159)) OR
    (direction_constraint.max_closure = open);
END_ENTITY;

ENTITY polar_point
SUBTYPE OF (cartesian_point);
   r : length_measure;
   theta : plane_angle_measure;
DERIVE
        SELF\cartesian_point.coordinates : LIST [1:3] OF length_measure := [ r
    * COS(theta), r * SIN(theta) ];
WHERE
   WR1:
      r >= 0.00000;
END_ENTITY;

ENTITY position_tolerance
SUBTYPE OF (geometric_tolerance);
WHERE
   WR1:
```

```
      NOT ('ENGINEERING_PROPERTIES_SCHEMA.' +
    'GEOMETRIC_TOLERANCE_WITH_DATUM_REFERENCE' IN TYPEOF(SELF)) OR
    (SIZEOF(SELF\geometric_tolerance_with_datum_reference.datum_system) <= 3);
END_ENTITY;


ENTITY power_expression
SUBTYPE OF (binary_numeric_expression);
END_ENTITY;

ENTITY power_measure_with_unit
SUBTYPE OF (measure_with_unit);
WHERE
   WR1:
      'ENGINEERING_PROPERTIES_SCHEMA.POWER_UNIT' IN
    TYPEOF(SELF\measure_with_unit.unit_component);
END_ENTITY;

ENTITY power_unit
SUBTYPE OF (derived_unit);
WHERE
   WR1:
       derive_dimensional_exponents(SELF) =
    dimensions_for_si_unit(si_unit_name.watt);
END_ENTITY;

ENTITY pre_defined_item;
   name : label;
END_ENTITY;

ENTITY precision_qualifier;
   precision_value : INTEGER;
END_ENTITY;

ENTITY pressure_measure_with_unit
SUBTYPE OF (measure_with_unit);
WHERE
   WR1:
      'ENGINEERING_PROPERTIES_SCHEMA.PRESSURE_UNIT' IN
    TYPEOF(SELF\measure_with_unit.unit_component);
END_ENTITY;

ENTITY pressure_unit
SUBTYPE OF (derived_unit);
WHERE
   WR1:
       derive_dimensional_exponents(SELF) =
    dimensions_for_si_unit(si_unit_name.pascal);
END_ENTITY;

ENTITY process_or_process_relationship_effectivity
SUBTYPE OF (effectivity);
    effective_process_or_process_relationship :
    process_or_process_relationship;
END_ENTITY;

ENTITY process_product_association;
   name : label;
   description : text;
```

```
   defined_product : characterized_product_definition;
   process : product_definition_process;
END_ENTITY;

ENTITY process_property_association;
   name : label;
   description : text;
   process : property_process;
   property_or_shape : property_or_shape_select;
END_ENTITY;

ENTITY product;
   id : identifier;
   name : label;
   description : OPTIONAL text;
   frame_of_reference : SET [1:?] OF product_context;
END_ENTITY;

ENTITY product_as_individual
ABSTRACT SUPERTYPE OF (ONEOF(product_as_planned, product_as_realised))
SUBTYPE OF (product_definition_formation);
END_ENTITY;

ENTITY product_as_planned
SUBTYPE OF (product_as_individual);
END_ENTITY;

ENTITY product_as_realised
SUBTYPE OF (product_as_individual);
END_ENTITY;

ENTITY product_category;
   name : label;
   description : OPTIONAL text;
DERIVE
   id : identifier := get_id_value(SELF);
WHERE
   WR1:
       SIZEOF(USEDIN(SELF, 'ENGINEERING_PROPERTIES_SCHEMA.' +
    'ID_ATTRIBUTE.IDENTIFIED_ITEM')) <= 1;
END_ENTITY;

ENTITY product_category_relationship;
   name : label;
   description : OPTIONAL text;
   category : product_category;
   sub_category : product_category;
WHERE
   WR1:
      acyclic_product_category_relationship(SELF, [ SELF.sub_category ]);
END_ENTITY;

ENTITY product_concept;
   id : identifier;
   name : label;
   description : OPTIONAL text;
   market_context : product_concept_context;
UNIQUE
   UR1 : id;
```

```
END_ENTITY;

ENTITY product_concept_context
SUBTYPE OF (application_context_element);
   market_segment_type : label;
END_ENTITY;

ENTITY product_context
SUBTYPE OF (application_context_element);
   discipline_type : label;
END_ENTITY;

ENTITY product_definition;
   id : identifier;
   description : OPTIONAL text;
   formation : product_definition_formation;
   frame_of_reference : product_definition_context;
DERIVE
   name : label := get_name_value(SELF);
WHERE
   WR1:
       SIZEOF(USEDIN(SELF, 'ENGINEERING_PROPERTIES_SCHEMA.' +
    'NAME_ATTRIBUTE.NAMED_ITEM')) <= 1;
END_ENTITY;

ENTITY product_definition_context
SUBTYPE OF (application_context_element);
   life_cycle_stage : label;
END_ENTITY;

ENTITY product_definition_formation;
   id : identifier;
   description : OPTIONAL text;
   of_product : product;
UNIQUE
   UR1 : id, of_product;
END_ENTITY;

ENTITY product_definition_formation_relationship;
   id : identifier;
   name : label;
   description : OPTIONAL text;
   relating_product_definition_formation : product_definition_formation;
   related_product_definition_formation : product_definition_formation;
END_ENTITY;

ENTITY product_definition_process
SUBTYPE OF (action);
   identification : identifier;
INVERSE
   product_definitions : SET [1:?] OF process_product_association FOR process;
END_ENTITY;

ENTITY product_definition_relationship;
   id : identifier;
   name : label;
   description : OPTIONAL text;
   relating_product_definition : product_definition;
   related_product_definition : product_definition;
```

```
END_ENTITY;

ENTITY product_definition_shape
SUBTYPE OF (property_definition);
UNIQUE
   UR1 : SELF\property_definition.definition;
WHERE
    WR1:
    SIZEOF([ 'ENGINEERING_PROPERTIES_SCHEMA.CHARACTERIZED_PRODUCT_DEFINITION',
    'ENGINEERING_PROPERTIES_SCHEMA.CHARACTERIZED_OBJECT' ] *
    TYPEOF(SELF\property_definition.definition)) > 0;
END_ENTITY;

ENTITY product_definition_substitute;
   description : OPTIONAL text;
   context_relationship : product_definition_relationship;
   substitute_definition : product_definition;
DERIVE
   name : label := get_name_value(SELF);
WHERE
   WR1:
       context_relationship.related_product_definition :<>:
    substitute_definition;
   WR2:
        SIZEOF(USEDIN(SELF, 'ENGINEERING_PROPERTIES_SCHEMA.' +
    'NAME_ATTRIBUTE.NAMED_ITEM')) <= 1;
END_ENTITY;

ENTITY product_definition_with_associated_documents
SUBTYPE OF (product_definition);
   documentation_ids : SET [1:?] OF document;
END_ENTITY;

ENTITY product_material_composition_relationship
SUBTYPE OF (product_definition_relationship);
   class : label;
   constituent_amount : SET [1:?] OF characterized_product_composition_value;
   composition_basis : label;
   determination_method : text;
END_ENTITY;

ENTITY product_relationship;
   id : identifier;
   name : label;
   description : OPTIONAL text;
   relating_product : product;
   related_product : product;
END_ENTITY;

ENTITY projected_zone_definition
SUBTYPE OF (tolerance_zone_definition);
   projection_end : shape_aspect;
   projected_length : measure_with_unit;
WHERE
   WR1:
       ('NUMBER' IN
    TYPEOF(projected_length\measure_with_unit.value_component)) AND
    (projected_length\measure_with_unit.value_component > 0.00000);
```

```
    WR2:
    derive_dimensional_exponents(projected_length\measure_with_unit.unit_compo
    nent) = dimensional_exponents(1, 0, 0, 0, 0, 0, 0);
END_ENTITY;

ENTITY property_definition;
   name : label;
   description : OPTIONAL text;
   definition : characterized_definition;
DERIVE
   id : identifier := get_id_value(SELF);
WHERE
   WR1:
       SIZEOF(USEDIN(SELF, 'ENGINEERING_PROPERTIES_SCHEMA.' +
    'ID_ATTRIBUTE.IDENTIFIED_ITEM')) <= 1;
END_ENTITY;

ENTITY property_definition_relationship;
   name : label;
   description : text;
   relating_property_definition : property_definition;
   related_property_definition : property_definition;
END_ENTITY;

ENTITY property_definition_representation;
   definition : represented_definition;
   used_representation : representation;
DERIVE
   description : text := get_description_value(SELF);
   name : label := get_name_value(SELF);
WHERE
   WR1:
       SIZEOF(USEDIN(SELF, 'ENGINEERING_PROPERTIES_SCHEMA.' +
    'DESCRIPTION_ATTRIBUTE.DESCRIBED_ITEM')) <= 1;
   WR2:
       SIZEOF(USEDIN(SELF, 'ENGINEERING_PROPERTIES_SCHEMA.' +
    'NAME_ATTRIBUTE.NAMED_ITEM')) <= 1;
END_ENTITY;

ENTITY property_process
SUBTYPE OF (action);
   identification : identifier;
INVERSE
   properties : SET [1:?] OF process_property_association FOR process;
END_ENTITY;

ENTITY qualification;
   id : identifier;
   name : label;
   description : OPTIONAL text;
END_ENTITY;

ENTITY qualification_relationship;
   id : identifier;
   name : label;
   description : OPTIONAL text;
   relating_qualification : qualification;
   related_qualification : qualification;
END_ENTITY;
```

```
ENTITY qualification_type;
   id : identifier;
   name : label;
   description : OPTIONAL text;
END_ENTITY;

ENTITY qualification_type_assignment
ABSTRACT SUPERTYPE;
   id : identifier;
   name : label;
   description : OPTIONAL text;
   assigned_qualification_type : qualification_type;
   role : qualification_type_role;
END_ENTITY;

ENTITY qualification_type_role;
   id : identifier;
   name : label;
   description : OPTIONAL text;
END_ENTITY;

ENTITY qualified_representation_item
SUBTYPE OF (representation_item);
   qualifiers : SET [1:?] OF value_qualifier;
WHERE
   WR1:
       SIZEOF(QUERY (temp <* qualifiers|
    ('ENGINEERING_PROPERTIES_SCHEMA.PRECISION_QUALIFIER' IN TYPEOF(temp)))) <
    2;
END_ENTITY;

ENTITY qualitative_uncertainty
SUBTYPE OF (uncertainty_qualifier);
   uncertainty_value : text;
END_ENTITY;

ENTITY quantifier_expression
ABSTRACT SUPERTYPE
SUBTYPE OF (multiple_arity_generic_expression);
   variables : LIST [1:?] OF UNIQUE generic_variable;
WHERE
   WR1:
      SIZEOF(QUERY (vrbl <* variables| NOT (vrbl IN
    SELF\multiple_arity_generic_expression.operands))) = 0;
   WR2:
       SIZEOF(QUERY (vrbl <* variables| NOT (schema_prefix +
    'BOUND_VARIABLE_SEMANTICS' IN TYPEOF(vrbl.interpretation.semantics)))) =
    0;
END_ENTITY;

ENTITY radioactivity_measure_with_unit
SUBTYPE OF (measure_with_unit);
WHERE
   WR1:
       'ENGINEERING_PROPERTIES_SCHEMA.RADIOACTIVITY_UNIT' IN
    TYPEOF(SELF\measure_with_unit.unit_component);
END_ENTITY;
```

```
ENTITY radioactivity_unit
SUBTYPE OF (derived_unit);
WHERE
   WR1:
       derive_dimensional_exponents(SELF) =
    dimensions_for_si_unit(si_unit_name.becquerel);
END_ENTITY;

ENTITY ratio_measure_with_unit
SUBTYPE OF (measure_with_unit);
WHERE
   WR1:
       'ENGINEERING_PROPERTIES_SCHEMA.RATIO_UNIT' IN
    TYPEOF(SELF\measure_with_unit.unit_component);
END_ENTITY;

ENTITY ratio_unit
SUBTYPE OF (named_unit);
WHERE
   WR1:
       ((((((SELF\named_unit.dimensions.length_exponent = 0.00000) AND
    (SELF\named_unit.dimensions.mass_exponent = 0.00000)) AND
    (SELF\named_unit.dimensions.time_exponent = 0.00000)) AND
    (SELF\named_unit.dimensions.electric_current_exponent = 0.00000)) AND
    (SELF\named_unit.dimensions.thermodynamic_temperature_exponent = 0.00000))
    AND (SELF\named_unit.dimensions.amount_of_substance_exponent = 0.00000))
    AND (SELF\named_unit.dimensions.luminous_intensity_exponent = 0.00000);
END_ENTITY;

ENTITY rationalize_function
SUBTYPE OF (maths_function, unary_generic_expression);
   SELF\unary_generic_expression.operand : maths_function;
DERIVE
   fun : maths_function := SELF\unary_generic_expression.operand;
WHERE
   WR1:
      (space_dimension(fun.domain) = 1) AND (space_dimension(fun.range) = 1);
   WR2:
      number_tuple_subspace_check(factor1(fun.range));
   WR3:
      space_dimension(factor1(fun.range)) > 1;
END_ENTITY;

ENTITY real_defined_function
ABSTRACT SUPERTYPE
SUBTYPE OF (numeric_defined_function);
END_ENTITY;

ENTITY real_interval_from_min
SUBTYPE OF (maths_space, generic_literal);
   min : REAL;
   min_closure : open_closed;
END_ENTITY;

ENTITY real_interval_to_max
SUBTYPE OF (maths_space, generic_literal);
   max : REAL;
   max_closure : open_closed;
END_ENTITY;
```

```
ENTITY real_literal
SUBTYPE OF (literal_number);
   SELF\literal_number.the_value : REAL;
END_ENTITY;

ENTITY real_numeric_variable
SUBTYPE OF (numeric_variable);
END_ENTITY;

ENTITY real_tuple_literal
SUBTYPE OF (generic_literal);
   lit_value : LIST [1:?] OF REAL;
END_ENTITY;

ENTITY referenced_modified_datum
SUBTYPE OF (datum_reference);
   modifier : limit_condition;
END_ENTITY;

ENTITY regular_table_function
SUBTYPE OF (linearized_table_function);
   increments : LIST [1:?] OF INTEGER;
WHERE
   WR1:
      SIZEOF(increments) = SIZEOF(SELF\explicit_table_function.shape);
   WR2:
      extremal_position_check(SELF);
END_ENTITY;

ENTITY reindexed_array_function
SUBTYPE OF (maths_function, unary_generic_expression);
   SELF\unary_generic_expression.operand : maths_function;
   starting_indices : LIST [1:?] OF INTEGER;
WHERE
   WR1:
      function_is_array(SELF\unary_generic_expression.operand);
   WR2:
       SIZEOF(starting_indices) =
       SIZEOF(shape_of_array(SELF\unary_generic_expression.operand));
END_ENTITY;

ENTITY repackaging_function
SUBTYPE OF (maths_function, unary_generic_expression);
   SELF\unary_generic_expression.operand : maths_function;
   input_repack : repackage_options;
   output_repack : repackage_options;
   selected_output : nonnegative_integer;
WHERE
   WR1:
       (input_repack <> ro_wrap_as_tuple) OR (space_dimension(operand.domain)
    = 1) AND (schema_prefix + 'TUPLE_SPACE' IN
    TYPEOF(factor1(operand.domain)));
   WR2:
       (output_repack <> ro_unwrap_tuple) OR (space_dimension(operand.range) =
    1) AND (schema_prefix + 'TUPLE_SPACE' IN TYPEOF(factor1(operand.range)));
   WR3:
       selected_output <= space_dimension(repackage(operand.range,
    output_repack));
```

```
END_ENTITY;

ENTITY representation;
   name : label;
   items : SET [1:?] OF representation_item;
   context_of_items : representation_context;
DERIVE
   id : identifier := get_id_value(SELF);
   description : text := get_description_value(SELF);
WHERE
   WR1:
       SIZEOF(USEDIN(SELF, 'ENGINEERING_PROPERTIES_SCHEMA.' +
    'ID_ATTRIBUTE.IDENTIFIED_ITEM')) <= 1;
   WR2:
        SIZEOF(USEDIN(SELF, 'ENGINEERING_PROPERTIES_SCHEMA.' +
    'DESCRIPTION_ATTRIBUTE.DESCRIBED_ITEM')) <= 1;
END_ENTITY;

ENTITY representation_context;
   context_identifier : identifier;
   context_type : text;
INVERSE
    representations_in_context : SET [1:?] OF representation FOR
    context_of_items;
END_ENTITY;

ENTITY representation_item;
   name : label;
WHERE
   WR1:
      SIZEOF(using_representations(SELF)) > 0;
END_ENTITY;

ENTITY representation_item_relationship;
   name : label;
   description : OPTIONAL text;
   relating_representation_item : representation_item;
   related_representation_item : representation_item;
END_ENTITY;

ENTITY representation_map;
   mapping_origin : representation_item;
   mapped_representation : representation;
INVERSE
   map_usage : SET [1:?] OF mapped_item FOR mapping_source;
WHERE
   WR1:
       item_in_context(SELF.mapping_origin,
    SELF.mapped_representation.context_of_items);
END_ENTITY;

ENTITY representation_relationship;
   name : label;
   description : OPTIONAL text;
   rep_1 : representation;
   rep_2 : representation;
END_ENTITY;

ENTITY representation_relationship_with_transformation
```

```
SUBTYPE OF (representation_relationship);
   transformation_operator : transformation;
WHERE
   WR1:
       SELF\representation_relationship.rep_1.context_of_items :<>:
    SELF\representation_relationship.rep_2.context_of_items;
END_ENTITY;

ENTITY requirement_for_action_resource
SUBTYPE OF (action_resource_requirement);
   resources : SET [1:?] OF action_resource;
END_ENTITY;

ENTITY resistance_measure_with_unit
SUBTYPE OF (measure_with_unit);
WHERE
   WR1:
       'ENGINEERING_PROPERTIES_SCHEMA.RESISTANCE_UNIT' IN
    TYPEOF(SELF\measure_with_unit.unit_component);
END_ENTITY;

ENTITY resistance_unit
SUBTYPE OF (derived_unit);
WHERE
   WR1:
       derive_dimensional_exponents(SELF) =
    dimensions_for_si_unit(si_unit_name.ohm);
END_ENTITY;

ENTITY resource_property;
   name : label;
   description : text;
   resource : characterized_resource_definition;
END_ENTITY;

ENTITY resource_property_relationship;
   name : label;
   description : text;
   relating_resource_property : resource_property;
   related_resource_property : resource_property;
WHERE
   WR1:
      relating_resource_property :<>: related_resource_property;
END_ENTITY;

ENTITY resource_property_representation;
   name : label;
   description : text;
   property : resource_property;
   representation : representation;
END_ENTITY;

ENTITY resource_requirement_type;
   name : label;
   description : text;
END_ENTITY;

ENTITY resource_requirement_type_relationship;
   name : label;
```

```
   description : text;
   relating_requirement_type : resource_requirement_type;
   related_requirement_type : resource_requirement_type;
WHERE
   WR1:
      relating_requirement_type :<>: related_requirement_type;
END_ENTITY;

ENTITY restriction_function
SUBTYPE OF (maths_function, unary_generic_expression);
   SELF\unary_generic_expression.operand : maths_space;
END_ENTITY;

ENTITY role_association;
   role : object_role;
   item_with_role : role_select;
END_ENTITY;

ENTITY roundness_tolerance
SUBTYPE OF (geometric_tolerance);
WHERE
   WR1:
       NOT ('ENGINEERING_PROPERTIES_SCHEMA.' +
    'GEOMETRIC_TOLERANCE_WITH_DATUM_REFERENCE' IN TYPEOF(SELF));
END_ENTITY;

ENTITY runout_zone_definition
SUBTYPE OF (tolerance_zone_definition);
   orientation : runout_zone_orientation;
END_ENTITY;

ENTITY runout_zone_orientation;
   angle : measure_with_unit;
END_ENTITY;

ENTITY runout_zone_orientation_reference_direction
SUBTYPE OF (runout_zone_orientation);
   orientation_defining_relationship : shape_aspect_relationship;
END_ENTITY;

ENTITY security_classification;
   name : label;
   purpose : text;
   security_level : security_classification_level;
END_ENTITY;

ENTITY security_classification_assignment
ABSTRACT SUPERTYPE;
   assigned_security_classification : security_classification;
DERIVE
   role : object_role := get_role(SELF);
WHERE
   WR1:
       SIZEOF(USEDIN(SELF, 'ENGINEERING_PROPERTIES_SCHEMA.' +
    'ROLE_ASSOCIATION.ITEM_WITH_ROLE')) <= 1;
END_ENTITY;

ENTITY security_classification_level;
   name : label;
```

```
END_ENTITY;

ENTITY selector_function
SUBTYPE OF (maths_function, generic_literal);
   selector : input_selector;
   source_of_domain : maths_space_or_function;
WHERE
   WR1:
      no_cyclic_domain_reference(source_of_domain, [ SELF ]);
   WR2:
      expression_is_constant(domain_from(source_of_domain));
END_ENTITY;

ENTITY sequential_method
SUBTYPE OF (serial_action_method);
   sequence_position : count_measure;
END_ENTITY;

ENTITY serial_action_method
SUBTYPE OF (action_method_relationship);
END_ENTITY;

ENTITY serial_numbered_effectivity
SUBTYPE OF (effectivity);
   effectivity_start_id : identifier;
   effectivity_end_id : OPTIONAL identifier;
END_ENTITY;

ENTITY series_composed_function
SUBTYPE OF (maths_function, multiple_arity_generic_expression);
    SELF\multiple_arity_generic_expression.operands : LIST [2:?] OF
    maths_function;
WHERE
   WR1:
      composable_sequence(SELF\multiple_arity_generic_expression.operands);
END_ENTITY;

ENTITY shape_aspect;
   name : label;
   description : OPTIONAL text;
   of_shape : product_definition_shape;
   product_definitional : LOGICAL;
DERIVE
   id : identifier := get_id_value(SELF);
WHERE
   WR1:
       SIZEOF(USEDIN(SELF, 'ENGINEERING_PROPERTIES_SCHEMA.' +
    'ID_ATTRIBUTE.IDENTIFIED_ITEM')) <= 1;
END_ENTITY;

ENTITY shape_aspect_deriving_relationship
SUBTYPE OF (shape_aspect_relationship);
WHERE
   WR1:
       'ENGINEERING_PROPERTIES_SCHEMA.DERIVED_SHAPE_ASPECT' IN
    TYPEOF(SELF\shape_aspect_relationship.relating_shape_aspect);
END_ENTITY;

ENTITY shape_aspect_relationship;
```

```
   name : label;
   description : OPTIONAL text;
   relating_shape_aspect : shape_aspect;
   related_shape_aspect : shape_aspect;
DERIVE
   id : identifier := get_id_value(SELF);
WHERE
   WR1:
       SIZEOF(USEDIN(SELF, 'ENGINEERING_PROPERTIES_SCHEMA.' +
    'ID_ATTRIBUTE.IDENTIFIED_ITEM')) <= 1;
END_ENTITY;

ENTITY shape_definition_representation
SUBTYPE OF (property_definition_representation);
WHERE
   WR1:
       ('ENGINEERING_PROPERTIES_SCHEMA.PRODUCT_DEFINITION_SHAPE' IN
    TYPEOF(SELF.definition)) OR
    ('ENGINEERING_PROPERTIES_SCHEMA.SHAPE_DEFINITION' IN
    TYPEOF(SELF.definition.definition));
   WR2:
       'ENGINEERING_PROPERTIES_SCHEMA.SHAPE_REPRESENTATION' IN
    TYPEOF(SELF.used_representation);
END_ENTITY;

ENTITY shape_dimension_representation
SUBTYPE OF (shape_representation);
WHERE
   WR1:
       SIZEOF(QUERY (temp <* SELF\representation.items| NOT
    ('ENGINEERING_PROPERTIES_SCHEMA.MEASURE_REPRESENTATION_ITEM' IN
    TYPEOF(temp)))) = 0;
   WR2:
      SIZEOF(SELF\representation.items) <= 3;
   WR3:
       SIZEOF(QUERY (pos_mri <* QUERY (real_mri <* SELF\representation.items|
    ('REAL' IN TYPEOF(real_mri\measure_with_unit.value_component)))| NOT
    (pos_mri\measure_with_unit.value_component > 0.00000))) = 0;
END_ENTITY;

ENTITY shape_representation
SUBTYPE OF (representation);
END_ENTITY;

ENTITY shape_representation_relationship
SUBTYPE OF (representation_relationship);
WHERE
   WR1:
       'ENGINEERING_PROPERTIES_SCHEMA.SHAPE_REPRESENTATION' IN
    TYPEOF(SELF\representation_relationship.rep_1) +
    TYPEOF(SELF\representation_relationship.rep_2);
END_ENTITY;

ENTITY si_unit
SUBTYPE OF (named_unit);
   prefix : OPTIONAL si_prefix;
   name : si_unit_name;
DERIVE
```

```
      SELF\named_unit.dimensions : dimensional_exponents :=
    dimensions_for_si_unit(name);
WHERE
   WR1:
      NOT (('ENGINEERING_PROPERTIES_SCHEMA.MASS_UNIT' IN TYPEOF(SELF)) AND
    (SIZEOF(USEDIN(SELF,
    'ENGINEERING_PROPERTIES_SCHEMA.DERIVED_UNIT_ELEMENT.UNIT')) > 0)) OR
    (prefix = si_prefix.kilo);
END_ENTITY;

ENTITY simple_boolean_expression
ABSTRACT SUPERTYPE OF (ONEOF(boolean_literal, boolean_variable))
SUBTYPE OF (boolean_expression, simple_generic_expression);
END_ENTITY;

ENTITY simple_generic_expression
ABSTRACT SUPERTYPE OF (ONEOF(generic_literal, generic_variable))
SUBTYPE OF (generic_expression);
END_ENTITY;

ENTITY simple_numeric_expression
ABSTRACT SUPERTYPE OF (ONEOF(literal_number, numeric_variable))
SUBTYPE OF (numeric_expression, simple_generic_expression);
END_ENTITY;

ENTITY simple_string_expression
ABSTRACT SUPERTYPE OF (ONEOF(string_literal, string_variable))
SUBTYPE OF (string_expression, simple_generic_expression);
END_ENTITY;

ENTITY sin_function
SUBTYPE OF (unary_function_call);
END_ENTITY;

ENTITY solid_angle_measure_with_unit
SUBTYPE OF (measure_with_unit);
WHERE
   WR1:
      'ENGINEERING_PROPERTIES_SCHEMA.SOLID_ANGLE_UNIT' IN
    TYPEOF(SELF\measure_with_unit.unit_component);
END_ENTITY;

ENTITY solid_angle_unit
SUBTYPE OF (named_unit);
WHERE
   WR1:
      ((((((SELF\named_unit.dimensions.length_exponent = 0.00000) AND
    (SELF\named_unit.dimensions.mass_exponent = 0.00000)) AND
    (SELF\named_unit.dimensions.time_exponent = 0.00000)) AND
    (SELF\named_unit.dimensions.electric_current_exponent = 0.00000)) AND
    (SELF\named_unit.dimensions.thermodynamic_temperature_exponent = 0.00000))
    AND (SELF\named_unit.dimensions.amount_of_substance_exponent = 0.00000))
    AND (SELF\named_unit.dimensions.luminous_intensity_exponent = 0.00000);
END_ENTITY;

ENTITY spherical_point
SUBTYPE OF (cartesian_point);
   r : length_measure;
   theta : plane_angle_measure;
```

```
   phi : plane_angle_measure;
DERIVE
      SELF\cartesian_point.coordinates : LIST [1:3] OF length_measure := [ r *
    SIN(theta) * COS(phi), r * SIN(theta) * SIN(phi), r * COS(theta) ];
WHERE
   WR1:
      r >= 0.00000;
END_ENTITY;

ENTITY spherical_surface
SUBTYPE OF (elementary_surface);
   radius : positive_length_measure;
END_ENTITY;

ENTITY spherical_volume
SUBTYPE OF (volume);
   position : axis2_placement_3d;
   radius : positive_length_measure;
END_ENTITY;

ENTITY square_root_function
SUBTYPE OF (unary_function_call);
END_ENTITY;

ENTITY standard_table_function
SUBTYPE OF (linearized_table_function);
   order : ordering_type;
WHERE
   WR1:
      extremal_position_check(SELF);
END_ENTITY;

ENTITY standard_uncertainty
SUPERTYPE OF (expanded_uncertainty)
SUBTYPE OF (uncertainty_qualifier);
   uncertainty_value : REAL;
END_ENTITY;

ENTITY state_observed;
   name : label;
   description : OPTIONAL text;
END_ENTITY;

ENTITY state_observed_assignment
ABSTRACT SUPERTYPE;
   assigned_state_observed : state_observed;
   role : state_observed_role;
END_ENTITY;

ENTITY state_observed_relationship;
   name : label;
   description : OPTIONAL text;
   relating_state_observed : SET [1:?] OF state_observed;
   related_state_observed : SET [1:?] OF state_observed;
END_ENTITY;

ENTITY state_observed_role;
   name : label;
   description : OPTIONAL text;
```

```
END_ENTITY;

ENTITY state_type;
   name : label;
   description : OPTIONAL text;
END_ENTITY;

ENTITY state_type_assignment
ABSTRACT SUPERTYPE;
   assigned_state_type : state_type;
   role : state_type_role;
END_ENTITY;

ENTITY state_type_relationship;
   name : label;
   description : OPTIONAL text;
   relating_state_type : SET [1:?] OF state_type;
   related_state_type : SET [1:?] OF state_type;
END_ENTITY;

ENTITY state_type_role;
   name : label;
   description : OPTIONAL text;
END_ENTITY;

ENTITY statistical_distribution_for_tolerance
SUBTYPE OF (representation);
WHERE
   WR1:
       SIZEOF(QUERY (item <* SELF\representation.items| NOT
    ('ENGINEERING_PROPERTIES_SCHEMA.MEASURE_REPRESENTATION_ITEM' IN
    TYPEOF(item)))) = 0;
END_ENTITY;

ENTITY straightness_tolerance
SUBTYPE OF (geometric_tolerance);
WHERE
   WR1:
        NOT ('ENGINEERING_PROPERTIES_SCHEMA.' +
    'GEOMETRIC_TOLERANCE_WITH_DATUM_REFERENCE' IN TYPEOF(SELF));
END_ENTITY;

ENTITY strict_triangular_matrix
SUBTYPE OF (triangular_matrix);
   main_diagonal_value : maths_value;
END_ENTITY;

ENTITY string_defined_function
ABSTRACT SUPERTYPE
SUBTYPE OF (defined_function, string_expression);
END_ENTITY;

ENTITY string_expression
    ABSTRACT SUPERTYPE OF (ONEOF(simple_string_expression, index_expression,
    substring_expression, concat_expression, format_function,
    string_defined_function))
SUBTYPE OF (expression);
END_ENTITY;
```

```
ENTITY string_literal
SUBTYPE OF (simple_string_expression, generic_literal);
   the_value : STRING;
END_ENTITY;

ENTITY string_variable
SUBTYPE OF (simple_string_expression, variable);
END_ENTITY;

ENTITY substring_expression
SUBTYPE OF (string_expression, multiple_arity_generic_expression);
DERIVE
       operand : generic_expression :=
    SELF\multiple_arity_generic_expression.operands[1];
          index1 : generic_expression :=
    SELF\multiple_arity_generic_expression.operands[2];
          index2 : generic_expression :=
    SELF\multiple_arity_generic_expression.operands[3];
WHERE
   WR1:
       (('ENGINEERING_PROPERTIES_SCHEMA.STRING_EXPRESSION' IN TYPEOF(operand))
    AND ('ENGINEERING_PROPERTIES_SCHEMA.NUMERIC_EXPRESSION' IN
    TYPEOF(index1))) AND ('ENGINEERING_PROPERTIES_SCHEMA.NUMERIC_EXPRESSION'
    IN TYPEOF(index2));
   WR2:
      SIZEOF(SELF\multiple_arity_generic_expression.operands) = 3;
   WR3:
      is_int_expr(index1);
   WR4:
      is_int_expr(index2);
END_ENTITY;

ENTITY surface
SUPERTYPE OF (elementary_surface)
SUBTYPE OF (geometric_representation_item);
END_ENTITY;

ENTITY symmetric_banded_matrix
SUBTYPE OF (symmetric_matrix);
   default_entry : maths_value;
   above : nonnegative_integer;
WHERE
   WR1:
       member_of(default_entry,
    factor1(SELF\linearized_table_function.source.range));
END_ENTITY;

ENTITY symmetric_matrix
SUBTYPE OF (linearized_table_function);
   symmetry : symmetry_type;
   triangle : lower_upper;
   order : ordering_type;
WHERE
   WR1:
      SIZEOF(SELF\explicit_table_function.shape) = 2;
   WR2:
       SELF\explicit_table_function.shape[1] =
    SELF\explicit_table_function.shape[2];
   WR3:
```

```
       NOT (symmetry = skew) OR
    (space_dimension(SELF\linearized_table_function.source.range) = 1) AND
    subspace_of_es(factor1(SELF\linearized_table_function.source.range),
    es_numbers);
   WR4:
       NOT ((symmetry = hermitian) OR (symmetry = skew_hermitian)) OR
    (space_dimension(SELF\linearized_table_function.source.range) = 1) AND
    subspace_of_es(factor1(SELF\linearized_table_function.source.range),
    es_complex_numbers);
END_ENTITY;

ENTITY symmetric_shape_aspect
SUBTYPE OF (shape_aspect);
INVERSE
      basis_relationships : SET [1:?] OF shape_aspect_relationship FOR
    relating_shape_aspect;
WHERE
   WR1:
       SIZEOF(QUERY (x <* SELF\symmetric_shape_aspect.basis_relationships|
    ('ENGINEERING_PROPERTIES_SCHEMA.CENTRE_OF_SYMMETRY' IN
    TYPEOF(x\shape_aspect_relationship.related_shape_aspect)))) >= 1;
END_ENTITY;

ENTITY symmetry_tolerance
SUBTYPE OF (geometric_tolerance_with_datum_reference);
WHERE
   WR1:
      SIZEOF(SELF\geometric_tolerance_with_datum_reference.datum_system) <= 3;
END_ENTITY;

ENTITY tan_function
SUBTYPE OF (unary_function_call);
END_ENTITY;

ENTITY tangent
SUBTYPE OF (derived_shape_aspect);
WHERE
   WR1:
      SIZEOF(SELF\derived_shape_aspect.deriving_relationships) = 1;
END_ENTITY;

ENTITY thermodynamic_temperature_measure_with_unit
SUBTYPE OF (measure_with_unit);
WHERE
   WR1:
       'ENGINEERING_PROPERTIES_SCHEMA.THERMODYNAMIC_TEMPERATURE_UNIT' IN
    TYPEOF(SELF\measure_with_unit.unit_component);
END_ENTITY;

ENTITY thermodynamic_temperature_unit
SUBTYPE OF (named_unit);
WHERE
   WR1:
       ((((((SELF\named_unit.dimensions.length_exponent = 0.00000) AND
    (SELF\named_unit.dimensions.mass_exponent = 0.00000)) AND
    (SELF\named_unit.dimensions.time_exponent = 0.00000)) AND
    (SELF\named_unit.dimensions.electric_current_exponent = 0.00000)) AND
    (SELF\named_unit.dimensions.thermodynamic_temperature_exponent = 1.00000))
```

```
    AND (SELF\named_unit.dimensions.amount_of_substance_exponent = 0.00000))
    AND (SELF\named_unit.dimensions.luminous_intensity_exponent = 0.00000);
END_ENTITY;

ENTITY time_interval;
   id : identifier;
   name : label;
   description : OPTIONAL text;
END_ENTITY;

ENTITY time_interval_assignment
ABSTRACT SUPERTYPE;
   assigned_time_interval : time_interval;
   role : time_interval_role;
END_ENTITY;

ENTITY time_interval_based_effectivity
SUBTYPE OF (effectivity);
   effectivity_period : time_interval;
END_ENTITY;

ENTITY time_interval_relationship;
   name : label;
   description : OPTIONAL text;
   relating_time_interval : time_interval;
   related_time_interval : time_interval;
END_ENTITY;

ENTITY time_interval_role;
   name : label;
   description : OPTIONAL text;
END_ENTITY;

ENTITY time_interval_with_bounds
SUBTYPE OF (time_interval);
   primary_bound : OPTIONAL date_time_or_event_occurrence;
   secondary_bound : OPTIONAL date_time_or_event_occurrence;
   duration : OPTIONAL time_measure_with_unit;
WHERE
   WR1:
      NOT (EXISTS(secondary_bound) AND EXISTS(duration));
   WR2:
      EXISTS(primary_bound) OR EXISTS(secondary_bound);
END_ENTITY;

ENTITY time_measure_with_unit
SUBTYPE OF (measure_with_unit);
WHERE
   WR1:
       'ENGINEERING_PROPERTIES_SCHEMA.TIME_UNIT' IN
    TYPEOF(SELF\measure_with_unit.unit_component);
END_ENTITY;

ENTITY time_role;
   name : label;
DERIVE
   description : text := get_description_value(SELF);
WHERE
   WR1:
```

```
      SIZEOF(USEDIN(SELF, 'ENGINEERING_PROPERTIES_SCHEMA.' +
    'DESCRIPTION_ATTRIBUTE.DESCRIBED_ITEM')) <= 1;
END_ENTITY;

ENTITY time_unit
SUBTYPE OF (named_unit);
WHERE
   WR1:
      ((((((SELF\named_unit.dimensions.length_exponent = 0.00000) AND
    (SELF\named_unit.dimensions.mass_exponent = 0.00000)) AND
    (SELF\named_unit.dimensions.time_exponent = 1.00000)) AND
    (SELF\named_unit.dimensions.electric_current_exponent = 0.00000)) AND
    (SELF\named_unit.dimensions.thermodynamic_temperature_exponent = 0.00000))
    AND (SELF\named_unit.dimensions.amount_of_substance_exponent = 0.00000))
    AND (SELF\named_unit.dimensions.luminous_intensity_exponent = 0.00000);
END_ENTITY;

ENTITY tolerance_value;
   lower_bound : measure_with_unit;
   upper_bound : measure_with_unit;
WHERE
   WR1:
      upper_bound\measure_with_unit.value_component >
    lower_bound\measure_with_unit.value_component;
   WR2:
      upper_bound\measure_with_unit.unit_component =
    lower_bound\measure_with_unit.unit_component;
END_ENTITY;

ENTITY tolerance_with_statistical_distribution;
   associated_tolerance : shape_tolerance_select;
   tolerance_allocation : statistical_distribution_for_tolerance;
END_ENTITY;

ENTITY tolerance_zone
SUBTYPE OF (shape_aspect);
   defining_tolerance : SET [1:?] OF geometric_tolerance;
   form : tolerance_zone_form;
END_ENTITY;

ENTITY tolerance_zone_definition
SUPERTYPE OF (ONEOF(projected_zone_definition, runout_zone_definition));
   zone : tolerance_zone;
   boundaries : SET [1:?] OF shape_aspect;
END_ENTITY;

ENTITY tolerance_zone_form;
   name : label;
END_ENTITY;

ENTITY total_runout_tolerance
SUBTYPE OF (geometric_tolerance_with_datum_reference);
WHERE
   WR1:
      SIZEOF(SELF\geometric_tolerance_with_datum_reference.datum_system) <= 2;
END_ENTITY;

ENTITY triangular_matrix
SUBTYPE OF (linearized_table_function);
```

```
   default_entry : maths_value;
   lo_up : lower_upper;
   order : ordering_type;
WHERE
   WR1:
      SIZEOF(SELF\explicit_table_function.shape) = 2;
   WR2:
      member_of(default_entry, SELF\maths_function.range);
END_ENTITY;

ENTITY type_qualifier;
   name : label;
END_ENTITY;

ENTITY unary_boolean_expression
ABSTRACT SUPERTYPE OF (ONEOF(not_expression, odd_function))
SUBTYPE OF (boolean_expression, unary_generic_expression);
END_ENTITY;

ENTITY unary_function_call
    ABSTRACT SUPERTYPE OF (ONEOF(abs_function, minus_function, sin_function,
    cos_function, tan_function, asin_function, acos_function, exp_function,
    log_function, log2_function, log10_function, square_root_function))
SUBTYPE OF (unary_numeric_expression);
END_ENTITY;

ENTITY unary_generic_expression
ABSTRACT SUPERTYPE
SUBTYPE OF (generic_expression);
   operand : generic_expression;
END_ENTITY;

ENTITY unary_numeric_expression
ABSTRACT SUPERTYPE OF (unary_function_call)
SUBTYPE OF (numeric_expression, unary_generic_expression);
   SELF\unary_generic_expression.operand : numeric_expression;
END_ENTITY;

ENTITY uncertainty_assigned_representation
SUBTYPE OF (representation);
   uncertainty : SET [1:?] OF uncertainty_measure_with_unit;
END_ENTITY;

ENTITY uncertainty_measure_with_unit
SUBTYPE OF (measure_with_unit);
   name : label;
   description : OPTIONAL text;
WHERE
   WR1:
      valid_measure_value(SELF\measure_with_unit.value_component);
END_ENTITY;

ENTITY uncertainty_qualifier
SUPERTYPE OF (ONEOF(standard_uncertainty, qualitative_uncertainty));
   measure_name : label;
   description : text;
END_ENTITY;

ENTITY uniform_product_space
```

```
SUBTYPE OF (maths_space, generic_literal);
   base : maths_space;
   exponent : positive_integer;
WHERE
   WR1:
      expression_is_constant(base);
   WR2:
      no_cyclic_space_reference(SELF, []);
   WR3:
      base <> the_empty_space;
END_ENTITY;

ENTITY value_function
SUPERTYPE OF (int_value_function)
SUBTYPE OF (numeric_expression, unary_generic_expression);
   SELF\unary_generic_expression.operand : string_expression;
END_ENTITY;

ENTITY value_representation_item
SUBTYPE OF (representation_item);
   value_component : measure_value;
WHERE
   WR1:
       SIZEOF(QUERY (rep <* using_representations(SELF)| NOT
    ('ENGINEERING_PROPERTIES_SCHEMA.GLOBAL_UNIT_ASSIGNED_CONTEXT' IN
    TYPEOF(rep.context_of_items)))) = 0;
END_ENTITY;

ENTITY variable
    ABSTRACT SUPERTYPE OF (ONEOF(numeric_variable, boolean_variable,
    string_variable))
SUBTYPE OF (generic_variable);
END_ENTITY;

ENTITY variable_semantics
ABSTRACT SUPERTYPE;
END_ENTITY;

ENTITY vector
SUBTYPE OF (geometric_representation_item);
   orientation : direction;
   magnitude : length_measure;
WHERE
   WR1:
      magnitude >= 0.00000;
END_ENTITY;

ENTITY velocity_measure_with_unit
SUBTYPE OF (measure_with_unit);
WHERE
   WR1:
       'ENGINEERING_PROPERTIES_SCHEMA.VELOCITY_UNIT' IN
    TYPEOF(SELF\measure_with_unit.unit_component);
END_ENTITY;

ENTITY versioned_action_request;
   id : identifier;
   version : label;
   purpose : text;
```

```
      description : OPTIONAL text;
   END_ENTITY;

   ENTITY volume
   SUPERTYPE OF (ONEOF(block_volume, spherical_volume, cylindrical_volume))
   SUBTYPE OF (geometric_representation_item);
   WHERE
      WR1:
         SELF\geometric_representation_item.dim = 3;
   END_ENTITY;

   ENTITY volume_measure_with_unit
   SUBTYPE OF (measure_with_unit);
   WHERE
      WR1:
          'ENGINEERING_PROPERTIES_SCHEMA.VOLUME_UNIT' IN
       TYPEOF(SELF\measure_with_unit.unit_component);
   END_ENTITY;

   ENTITY volume_unit
   SUBTYPE OF (derived_unit);
   WHERE
      WR1:
          derive_dimensional_exponents(SELF) = dimensional_exponents(3.00000,
       0.00000, 0.00000, 0.00000, 0.00000, 0.00000, 0.00000);
   END_ENTITY;

   ENTITY week_of_year_and_day_date
   SUBTYPE OF (date);
      week_component : week_in_year_number;
      day_component : OPTIONAL day_in_week_number;
   END_ENTITY;

   ENTITY xor_expression
   SUBTYPE OF (binary_boolean_expression);
      SELF\binary_generic_expression.operands : LIST [2:2] OF boolean_expression;
   END_ENTITY;

   ENTITY year_month
   SUBTYPE OF (date);
      month_component : month_in_year_number;
   END_ENTITY;

(* *********************************
Types in the schema engineering_properties_schema
********************************* *)

   TYPE absorbed_dose_measure = REAL;
   END_TYPE;

   TYPE acceleration_measure = REAL;
   END_TYPE;

   TYPE action_item = SELECT
      (approval,
       certification,
       document,
       material_property,
       material_property_representation,
```

```
    product_definition,
    property_definition,
    property_definition_representation);
END_TYPE;

TYPE action_request_item = SELECT
   (action,
    approval,
    certification,
    document,
    executed_action,
    material_property,
    product,
    product_definition,
    product_definition_formation,
    property_definition,
    organizational_project,
    security_classification,
    security_classification_level);
END_TYPE;

TYPE ahead_or_behind = ENUMERATION OF
   (ahead,
    exact,
    behind);
END_TYPE;

TYPE amount_of_substance_measure = REAL;
END_TYPE;

TYPE angle_relator = ENUMERATION OF
   (equal,
    large,
    small);
END_TYPE;

TYPE approval_item = SELECT
   (material_property,
    product,
    product_definition,
    product_definition_formation,
    property_definition,
    representation,
    versioned_action_request);
END_TYPE;

TYPE area_measure = REAL;
END_TYPE;

TYPE atom_based_tuple = LIST OF atom_based_value;
END_TYPE;

TYPE atom_based_value = SELECT
   (maths_atom,
    atom_based_tuple);
END_TYPE;

TYPE attribute_language_item = SELECT
   (action,
```

```
    action_method,
    action_property,
    application_context,
    certification,
    document,
    descriptive_representation_item,
    material_designation,
    material_property,
    material_property_representation,
    product,
    product_definition,
    product_definition_formation,
    property_definition,
    qualification_type,
    representation);
END_TYPE;

TYPE attribute_type = SELECT
   (label,
    text);
END_TYPE;

TYPE axis2_placement = SELECT
   (axis2_placement_2d,
    axis2_placement_3d);
END_TYPE;

TYPE capacitance_measure = REAL;
END_TYPE;

TYPE celsius_temperature_measure = REAL;
END_TYPE;

TYPE certification_item = SELECT
   (action,
    action_method,
    material_property,
    organization,
    product,
    product_definition,
    product_definition_formation,
    person_and_organization,
    property_definition);
END_TYPE;

TYPE characterized_action_definition = SELECT
   (action,
    action_method,
    action_method_relationship,
    action_relationship);
END_TYPE;

TYPE characterized_definition = SELECT
   (characterized_object,
    characterized_product_definition,
    shape_definition);
END_TYPE;

TYPE characterized_material_property = SELECT
```

```
   (material_property_representation);
END_TYPE;

TYPE characterized_product_composition_value = SELECT
   (maths_value_with_unit,
    measure_with_unit);
END_TYPE;

TYPE characterized_product_definition = SELECT
   (product_definition,
    product_definition_relationship);
END_TYPE;

TYPE characterized_resource_definition = SELECT
   (action_resource,
    action_resource_relationship,
    action_resource_requirement,
    action_resource_requirement_relationship);
END_TYPE;

TYPE compound_item_definition = SELECT
   (list_representation_item,
    set_representation_item);
END_TYPE;

TYPE conductance_measure = REAL;
END_TYPE;

TYPE configuration_design_item = SELECT
   (product_definition,
    product_definition_formation);
END_TYPE;

TYPE context_dependent_measure = REAL;
END_TYPE;

TYPE contract_item = SELECT
   (action,
    material_property,
    organizational_project,
    person_organization_select,
    product,
    property_definition);
END_TYPE;

TYPE count_measure = NUMBER;
END_TYPE;

TYPE date_and_time_item = SELECT
   (action,
    event_occurrence,
    representation,
    versioned_action_request);
END_TYPE;

TYPE date_item = SELECT
   (action,
    approval,
    certification,
```

```
    contract,
    event_occurrence,
    product_definition_formation,
    representation,
    versioned_action_request);
END_TYPE;

TYPE date_time_or_event_occurrence = SELECT
   (date_time_select,
    event_occurrence);
END_TYPE;

TYPE date_time_select = SELECT
   (date,
    date_and_time,
    local_time);
END_TYPE;

TYPE day_in_month_number = INTEGER;
WHERE
   WR1:
      (1 <= SELF) AND (SELF <= 31);
END_TYPE;

TYPE day_in_week_number = INTEGER;
WHERE
   WR1:
      (1 <= SELF) AND (SELF <= 7);
END_TYPE;

TYPE day_in_year_number = INTEGER;
WHERE
   WR1:
      (1 <= SELF) AND (SELF <= 366);
END_TYPE;

TYPE derived_property_select = SELECT
   (property_definition,
    action_property,
    resource_property);
END_TYPE;

TYPE description_attribute_select = SELECT
   (action_request_solution,
    application_context,
    approval_role,
    configuration_design,
    date_role,
    date_time_role,
    context_dependent_shape_representation,
    effectivity,
    external_source,
    organization_role,
    person_and_organization_role,
    person_and_organization,
    person_role,
    property_definition_representation,
    representation,
    time_role);
```

```
END_TYPE;

TYPE descriptive_measure = STRING;
END_TYPE;

TYPE dimension_count = positive_integer;
END_TYPE;

TYPE dimensional_characteristic = SELECT
   (dimensional_location,
    dimensional_size);
END_TYPE;

TYPE document_item = SELECT
   (action,
    action_method,
    action_resource,
    action_resource_requirement,
    contract,
    geometric_tolerance,
    material_designation,
    material_property,
    product_definition,
    product_definition_formation,
    product_definition_process,
    property_definition,
    representation);
END_TYPE;

TYPE dose_equivalent_measure = REAL;
END_TYPE;

TYPE dotted_express_identifier = STRING;
WHERE
   syntax:
      dotted_identifiers_syntax(SELF);
END_TYPE;

TYPE effectivity_item = SELECT
   (action,
    document,
    product_definition_formation);
END_TYPE;

TYPE electric_charge_measure = REAL;
END_TYPE;

TYPE electric_current_measure = REAL;
END_TYPE;

TYPE electric_potential_measure = REAL;
END_TYPE;

TYPE elementary_function_enumerators = ENUMERATION OF
   (ef_and,
    ef_or,
    ef_not,
    ef_xor,
    ef_negate_i,
```

```
ef_add_i,
ef_subtract_i,
ef_multiply_i,
ef_divide_i,
ef_mod_i,
ef_exponentiate_i,
ef_eq_i,
ef_ne_i,
ef_gt_i,
ef_lt_i,
ef_ge_i,
ef_le_i,
ef_abs_i,
ef_max_i,
ef_min_i,
ef_if_i,
ef_negate_r,
ef_reciprocal_r,
ef_add_r,
ef_subtract_r,
ef_multiply_r,
ef_divide_r,
ef_mod_r,
ef_exponentiate_r,
ef_exponentiate_ri,
ef_eq_r,
ef_ne_r,
ef_gt_r,
ef_lt_r,
ef_ge_r,
ef_le_r,
ef_abs_r,
ef_max_r,
ef_min_r,
ef_acos_r,
ef_asin_r,
ef_atan2_r,
ef_cos_r,
ef_exp_r,
ef_ln_r,
ef_log2_r,
ef_log10_r,
ef_sin_r,
ef_sqrt_r,
ef_tan_r,
ef_if_r,
ef_form_c,
ef_rpart_c,
ef_ipart_c,
ef_negate_c,
ef_reciprocal_c,
ef_add_c,
ef_subtract_c,
ef_multiply_c,
ef_divide_c,
ef_exponentiate_c,
ef_exponentiate_ci,
ef_eq_c,
ef_ne_c,
```

```
ef_conjugate_c,
ef_abs_c,
ef_arg_c,
ef_cos_c,
ef_exp_c,
ef_ln_c,
ef_sin_c,
ef_sqrt_c,
ef_tan_c,
ef_if_c,
ef_subscript_s,
ef_eq_s,
ef_ne_s,
ef_gt_s,
ef_lt_s,
ef_ge_s,
ef_le_s,
ef_subsequence_s,
ef_concat_s,
ef_size_s,
ef_format,
ef_value,
ef_like,
ef_if_s,
ef_subscript_b,
ef_eq_b,
ef_ne_b,
ef_gt_b,
ef_lt_b,
ef_ge_b,
ef_le_b,
ef_subsequence_b,
ef_concat_b,
ef_size_b,
ef_if_b,
ef_subscript_t,
ef_eq_t,
ef_ne_t,
ef_concat_t,
ef_size_t,
ef_entuple,
ef_detuple,
ef_insert,
ef_remove,
ef_if_t,
ef_sum_it,
ef_product_it,
ef_add_it,
ef_subtract_it,
ef_scalar_mult_it,
ef_dot_prod_it,
ef_sum_rt,
ef_product_rt,
ef_add_rt,
ef_subtract_rt,
ef_scalar_mult_rt,
ef_dot_prod_rt,
ef_norm_rt,
ef_sum_ct,
```

```
    ef_product_ct,
    ef_add_ct,
    ef_subtract_ct,
    ef_scalar_mult_ct,
    ef_dot_prod_ct,
    ef_norm_ct,
    ef_if,
    ef_ensemble,
    ef_member_of);
END_TYPE;

TYPE elementary_space_enumerators = ENUMERATION OF
   (es_numbers,
    es_complex_numbers,
    es_reals,
    es_integers,
    es_logicals,
    es_booleans,
    es_strings,
    es_binarys,
    es_maths_spaces,
    es_maths_functions,
    es_generics);
END_TYPE;

TYPE energy_measure = REAL;
END_TYPE;

TYPE event_occurred_item = SELECT
   (action);
END_TYPE;

TYPE express_identifier = dotted_express_identifier;
WHERE
   syntax:
      dot_count(SELF) = 0;
END_TYPE;

TYPE extension_options = ENUMERATION OF
   (eo_none,
    eo_cont,
    eo_cont_right,
    eo_cont_left);
END_TYPE;

TYPE external_identification_item = SELECT
   (document,
    product,
    product_definition,
    externally_defined_class,

    externally_defined_engineering_property);
END_TYPE;

TYPE force_measure = REAL;
END_TYPE;

TYPE founded_item_select = SELECT
   (founded_item,
```

```
    representation_item);
END_TYPE;

TYPE frequency_measure = REAL;
END_TYPE;

TYPE geometric_set_select = SELECT
   (point,
    curve,
    surface);
END_TYPE;

TYPE groupable_item = SELECT
   (action,
    action_method,
    material_property,
    property_definition,
    product,
    product_definition);
END_TYPE;

TYPE hour_in_day = INTEGER;
WHERE
   WR1:
      (0 <= SELF) AND (SELF < 24);
END_TYPE;

TYPE id_attribute_select = SELECT
   (action,
    address,
    product_category,
    property_definition,
    shape_aspect,
    shape_aspect_relationship,
    application_context,
    group,
    organizational_project,
    representation);
END_TYPE;

TYPE identification_item = SELECT
   (certification,
    document,
    product,
    product_definition,
    organization,
    person_and_organization);
END_TYPE;

TYPE identifier = STRING;
END_TYPE;

TYPE illuminance_measure = REAL;
END_TYPE;

TYPE inductance_measure = REAL;
END_TYPE;

TYPE input_selector = positive_integer;
```

```
END_TYPE;

TYPE label = STRING;
END_TYPE;

TYPE language_item = SELECT
   (action,
    action_method,
    action_property,
    application_context,
    certification,
    document,
    descriptive_representation_item,
    material_designation,
    material_property,
    material_property_representation,
    product,
    product_definition,
    product_definition_formation,
    property_definition,
    qualification_type,
    representation);
END_TYPE;

TYPE length_measure = REAL;
END_TYPE;

TYPE limit_condition = ENUMERATION OF
   (maximum_material_condition,
    least_material_condition,
    regardless_of_feature_size);
END_TYPE;

TYPE list_representation_item = LIST [1:?] OF representation_item;
END_TYPE;

TYPE location_item = SELECT
   (action,
    event_occurrence,
    product,
    product_definition,
    product_definition_formation);
END_TYPE;

TYPE location_representation_item = SELECT
   (representation);
END_TYPE;

TYPE lower_upper = ENUMERATION OF
   (lower,
    upper);
END_TYPE;

TYPE luminous_flux_measure = REAL;
END_TYPE;

TYPE luminous_intensity_measure = REAL;
END_TYPE;
```

```
TYPE magnetic_flux_density_measure = REAL;
END_TYPE;

TYPE magnetic_flux_measure = REAL;
END_TYPE;

TYPE mass_measure = REAL;
END_TYPE;

TYPE maths_atom = SELECT
   (maths_simple_atom,
    maths_enum_atom);
END_TYPE;

TYPE maths_binary = BINARY;
END_TYPE;

TYPE maths_boolean = BOOLEAN;
END_TYPE;

TYPE maths_enum_atom = SELECT
   (elementary_space_enumerators,
    ordering_type,
    lower_upper,
    symmetry_type,
    elementary_function_enumerators,
    open_closed,
    space_constraint_type,
    repackage_options,
    extension_options);
END_TYPE;

TYPE maths_expression = SELECT
   (atom_based_value,
    maths_tuple,
    generic_expression);
END_TYPE;

TYPE maths_function_select = SELECT
   (maths_function,
    elementary_function_enumerators);
END_TYPE;

TYPE maths_integer = INTEGER;
END_TYPE;

TYPE maths_logical = LOGICAL;
END_TYPE;

TYPE maths_number = NUMBER;
END_TYPE;

TYPE maths_real = REAL;
END_TYPE;

TYPE maths_simple_atom = SELECT
   (maths_number,
    maths_real,
    maths_logical,
```

```
    maths_boolean,
    maths_string,
    maths_binary);
END_TYPE;

TYPE maths_space_or_function = SELECT
   (maths_space,
    maths_function);
END_TYPE;

TYPE maths_string = STRING;
END_TYPE;

TYPE maths_tuple = LIST [0:?] OF maths_value;
END_TYPE;

TYPE maths_value = SELECT
   (atom_based_value,
    maths_tuple,
    generic_expression);
WHERE
   constancy:
             NOT ('GENERIC_EXPRESSION' IN stripped_typeof(SELF)) OR
    expression_is_constant(SELF);
END_TYPE;

TYPE measure_value = SELECT
   (absorbed_dose_measure,
    dose_equivalent_measure,
    radioactivity_measure,
    acceleration_measure,
    amount_of_substance_measure,
    area_measure,
    celsius_temperature_measure,
    context_dependent_measure,
    count_measure,
    descriptive_measure,
    capacitance_measure,
    electric_charge_measure,
    conductance_measure,
    electric_current_measure,
    electric_potential_measure,
    energy_measure,
    magnetic_flux_density_measure,
    force_measure,
    frequency_measure,
    illuminance_measure,
    inductance_measure,
    length_measure,
    luminous_flux_measure,
    luminous_intensity_measure,
    magnetic_flux_measure,
    mass_measure,
    numeric_measure,
    non_negative_length_measure,
    parameter_value,
    plane_angle_measure,
    positive_length_measure,
    positive_plane_angle_measure,
```

```
    positive_ratio_measure,
    power_measure,
    pressure_measure,
    ratio_measure,
    resistance_measure,
    solid_angle_measure,
    thermodynamic_temperature_measure,
    time_measure,
    velocity_measure,
    volume_measure);
END_TYPE;

TYPE message = STRING;
END_TYPE;

TYPE minute_in_hour = INTEGER;
WHERE
   WR1:
      (0 <= SELF) AND (SELF <= 59);
END_TYPE;

TYPE month_in_year_number = INTEGER;
WHERE
   WR1:
      (1 <= SELF) AND (SELF <= 12);
END_TYPE;

TYPE multi_language_attribute_item = SELECT
   (action,
    action_method,
    action_property,
    application_context,
    certification,
    document,
    descriptive_representation_item,
    material_designation,
    material_property,
    material_property_representation,
    product,
    product_definition,
    product_definition_formation,
    property_definition,
    qualification_type,
    representation);
END_TYPE;

TYPE name_attribute_select = SELECT
   (action_request_solution,
    address,
    configuration_design,
    context_dependent_shape_representation,
    derived_unit,
    effectivity,
    person_and_organization,
    product_definition,
    product_definition_substitute,
    property_definition_representation);
END_TYPE;
```

```
TYPE non_negative_length_measure = length_measure;
WHERE
   WR1:
      SELF >= 0.00000;
END_TYPE;

TYPE nonnegative_integer = INTEGER;
WHERE
   nonnegativity:
      SELF >= 0;
END_TYPE;

TYPE numeric_measure = NUMBER;
END_TYPE;

TYPE one_or_two = positive_integer;
WHERE
   in_range:
      (SELF = 1) OR (SELF = 2);
END_TYPE;

TYPE open_closed = ENUMERATION OF
   (open,
    closed);
END_TYPE;

TYPE ordering_type = ENUMERATION OF
   (by_rows,
    by_columns);
END_TYPE;

TYPE organization_item = SELECT
   (action,
    approval,
    certification,
    document,
    material_designation,
    versioned_action_request);
END_TYPE;

TYPE organizational_project_item = SELECT
   (action,
    action_method,
    document,
    product,
    material_property);
END_TYPE;

TYPE parameter_value = REAL;
END_TYPE;

TYPE pcurve_or_surface = SELECT
   (surface);
END_TYPE;

TYPE person_and_organization_item = SELECT
   (action,
    certification,
    product_definition_formation,
```

```
    versioned_action_request);
END_TYPE;

TYPE person_item = SELECT
   (action,
    document,
    versioned_action_request);
END_TYPE;

TYPE person_organization_select = SELECT
   (person,
    organization,
    person_and_organization);
END_TYPE;

TYPE plane_angle_measure = REAL;
END_TYPE;

TYPE positive_integer = nonnegative_integer;
WHERE
   positivity:
      SELF > 0;
END_TYPE;

TYPE positive_length_measure = non_negative_length_measure;
WHERE
   WR1:
      SELF > 0.00000;
END_TYPE;

TYPE positive_plane_angle_measure = plane_angle_measure;
WHERE
   WR1:
      SELF > 0.00000;
END_TYPE;

TYPE positive_ratio_measure = ratio_measure;
WHERE
   WR1:
      SELF > 0.00000;
END_TYPE;

TYPE power_measure = REAL;
END_TYPE;

TYPE pressure_measure = REAL;
END_TYPE;

TYPE process_or_process_relationship = SELECT
   (product_definition_process,
    property_process,
    relationship_with_condition);
END_TYPE;

TYPE product_or_formation_or_definition = SELECT
   (product,
    product_definition_formation,
    product_definition);
END_TYPE;
```

```
TYPE product_space = SELECT
   (uniform_product_space,
    listed_product_space);
END_TYPE;

TYPE property_or_shape_select = SELECT
   (property_definition,
    shape_definition);
END_TYPE;

TYPE qualification_item = SELECT
   (person,
    person_and_organization,
    organization);
END_TYPE;

TYPE radioactivity_measure = REAL;
END_TYPE;

TYPE ratio_measure = REAL;
END_TYPE;

TYPE real_interval = SELECT
   (real_interval_from_min,
    real_interval_to_max,
    finite_real_interval,
    elementary_space);
WHERE
   WR1:
        NOT ('ELEMENTARY_SPACE' IN stripped_typeof(SELF)) OR
    (SELF\elementary_space.space_id = es_reals);
END_TYPE;

TYPE relationship_with_condition = SELECT
   (action_method_relationship,
    action_relationship,
    context_dependent_action_method_relationship,
    context_dependent_action_relationship);
END_TYPE;

TYPE repackage_options = ENUMERATION OF
   (ro_nochange,
    ro_wrap_as_tuple,
    ro_unwrap_tuple);
END_TYPE;

TYPE represented_definition = SELECT
   (general_property,
    property_definition,
    property_definition_relationship,
    shape_aspect,
    shape_aspect_relationship);
END_TYPE;

TYPE resistance_measure = REAL;
END_TYPE;

TYPE role_select = SELECT
```

```
   (action_assignment,
    action_request_assignment,
    approval_assignment,
    approval_date_time,
    certification_assignment,
    contract_assignment,
    document_reference,
    effectivity_assignment,
    external_referent_assignment,
    group_assignment,
    name_assignment,
    security_classification_assignment);
END_TYPE;

TYPE second_in_minute = REAL;
WHERE
   WR1:
      (0 <= SELF) AND (SELF <= 60.0000);
END_TYPE;

TYPE security_classified_item = SELECT
   (action,
    action_method,
    document,
    material_property,
    representation,
    representation_item);
END_TYPE;

TYPE set_representation_item = SET [1:?] OF representation_item;
END_TYPE;

TYPE shape_definition = SELECT
   (product_definition_shape,
    shape_aspect,
    shape_aspect_relationship);
END_TYPE;

TYPE shape_tolerance_select = SELECT
   (geometric_tolerance,
    plus_minus_tolerance);
END_TYPE;

TYPE si_prefix = ENUMERATION OF
   (exa,
    peta,
    tera,
    giga,
    mega,
    kilo,
    hecto,
    deca,
    deci,
    centi,
    milli,
    micro,
    nano,
    pico,
    femto,
```

```
    atto);
END_TYPE;

TYPE si_unit_name = ENUMERATION OF
   (metre,
    gram,
    second,
    ampere,
    kelvin,
    mole,
    candela,
    radian,
    steradian,
    hertz,
    newton,
    pascal,
    joule,
    watt,
    coulomb,
    volt,
    farad,
    ohm,
    siemens,
    weber,
    tesla,
    henry,
    degree_Celsius,
    lumen,
    lux,
    becquerel,
    gray,
    sievert);
END_TYPE;

TYPE solid_angle_measure = REAL;
END_TYPE;

TYPE source_item = SELECT
   (identifier,
    message);
END_TYPE;

TYPE space_constraint_type = ENUMERATION OF
   (sc_equal,
    sc_subspace,
    sc_member);
END_TYPE;

TYPE state_item = SELECT
   (action,
    action_method,
    product,
    product_definition,
    product_definition_formation,
    property_definition,
    material_property,
    material_property_representation);
END_TYPE;
```

```
TYPE state_observed_item = SELECT
   (action,
    action_method,
    product,
    product_definition,
    product_definition_formation,
    property_definition,
    material_property,
    material_property_representation);
END_TYPE;

TYPE supported_item = SELECT
   (action_directive,
    action,
    action_method);
END_TYPE;

TYPE symmetry_type = ENUMERATION OF
   (identity,
    skew,
    hermitian,
    skew_hermitian);
END_TYPE;

TYPE text = STRING;
END_TYPE;

TYPE thermodynamic_temperature_measure = REAL;
END_TYPE;

TYPE time_interval_item = SELECT
   (action,
    approval,
    effectivity,
    document,
    qualification);
END_TYPE;

TYPE time_measure = REAL;
END_TYPE;

TYPE tolerance_method_definition = SELECT
   (tolerance_value,
    limits_and_fits);
END_TYPE;

TYPE transformation = SELECT
   (item_defined_transformation,
    functionally_defined_transformation);
END_TYPE;

TYPE trimming_select = SELECT
   (cartesian_point,
    parameter_value);
END_TYPE;

TYPE tuple_space = SELECT
   (product_space,
    extended_tuple_space);
```

```
   END_TYPE;

   TYPE unit = SELECT
      (derived_unit,
       named_unit);
   END_TYPE;

   TYPE value_qualifier = SELECT
      (precision_qualifier,
       type_qualifier,
       uncertainty_qualifier);
   END_TYPE;

   TYPE vector_or_direction = SELECT
      (vector,
       direction);
   END_TYPE;

   TYPE velocity_measure = REAL;
   END_TYPE;


   TYPE volume_measure = REAL;
   END_TYPE;

   TYPE week_in_year_number = INTEGER;
   WHERE
      WR1:
         (1 <= SELF) AND (SELF <= 53);
   END_TYPE;

   TYPE year_number = INTEGER;
   END_TYPE;

   TYPE zero_or_one = nonnegative_integer;
   WHERE
      in_range:
         (SELF = 0) OR (SELF = 1);
   END_TYPE;

(* *********************************
Functions in the schema engineering_properties_schema
********************************* *)

   FUNCTION acyclic
      (arg1 : generic_expression;
       arg2 : SET OF generic_expression ) : BOOLEAN;
   LOCAL
      result : BOOLEAN;
   END_LOCAL;
            IF 'ENGINEERING_PROPERTIES_SCHEMA.SIMPLE_GENERIC_EXPRESSION' IN
       TYPEOF(arg1) THEN
         RETURN (TRUE);
      END_IF;
      IF arg1 IN arg2 THEN
         RETURN (FALSE);
      END_IF;
            IF 'ENGINEERING_PROPERTIES_SCHEMA.UNARY_GENERIC_EXPRESSION' IN
       TYPEOF(arg1) THEN
```

```
      RETURN (acyclic(arg1\unary_generic_expression.operand, arg2 +
    [ arg1 ]));
   END_IF;
         IF 'ENGINEERING_PROPERTIES_SCHEMA.BINARY_GENERIC_EXPRESSION' IN
    TYPEOF(arg1) THEN
      RETURN (acyclic(arg1\binary_generic_expression.operands[1], (arg2 +
    [ arg1 ])) AND acyclic(arg1\binary_generic_expression.operands[2], (arg2 +
    [ arg1 ])));
   END_IF;
         IF 'ENGINEERING_PROPERTIES_SCHEMA.MULTIPLE_ARITY_GENERIC_EXPRESSION'
    IN TYPEOF(arg1) THEN
      result := TRUE;
            REPEAT i := 1 TO
    SIZEOF(arg1\multiple_arity_generic_expression.operands);
         result := result AND
    acyclic(arg1\multiple_arity_generic_expression.operands[i], (arg2 +
    [ arg1 ]));
      END_REPEAT;
      RETURN (result);
   END_IF;
END_FUNCTION;

FUNCTION acyclic_mapped_representation
   (parent_set : SET OF representation;
    children_set : SET OF representation_item ) : BOOLEAN;
LOCAL
   x : SET OF representation_item;
   y : SET OF representation_item;
END_LOCAL;
          x := QUERY (z <* children_set|
    'ENGINEERING_PROPERTIES_SCHEMA.MAPPED_ITEM' IN TYPEOF(z));
   IF SIZEOF(x) > 0 THEN
      REPEAT i := 1 TO HIINDEX(x);
               IF x[i]\mapped_item.mapping_source.mapped_representation IN
    parent_set THEN
            RETURN (FALSE);
         END_IF;
               IF NOT acyclic_mapped_representation((parent_set +
    x[i]\mapped_item.mapping_source.mapped_representation),
    x[i]\mapped_item.mapping_source.mapped_representation.items) THEN
            RETURN (FALSE);
         END_IF;
      END_REPEAT;
   END_IF;
   x := children_set - x;
   IF SIZEOF(x) > 0 THEN
      REPEAT i := 1 TO HIINDEX(x);
             y := QUERY (z <* bag_to_set(USEDIN(x[i], ''))|
    'ENGINEERING_PROPERTIES_SCHEMA.REPRESENTATION_ITEM' IN TYPEOF(z));
         IF NOT acyclic_mapped_representation(parent_set, y) THEN
            RETURN (FALSE);
         END_IF;
      END_REPEAT;
   END_IF;
   RETURN (TRUE);
END_FUNCTION;

FUNCTION acyclic_product_category_relationship
   (relation : product_category_relationship;
```

```
    children : SET OF product_category ) : BOOLEAN;
LOCAL
   x : SET OF product_category_relationship;
   local_children : SET OF product_category;
END_LOCAL;
   REPEAT i := 1 TO HIINDEX(children);
      IF relation.category :=: children[i] THEN
         RETURN (FALSE);
      END_IF;
   END_REPEAT;
         x := bag_to_set(USEDIN(relation.category,
    'ENGINEERING_PROPERTIES_SCHEMA.' +
    'PRODUCT_CATEGORY_RELATIONSHIP.SUB_CATEGORY'));
   local_children := children + relation.category;
   IF SIZEOF(x) > 0 THEN
      REPEAT i := 1 TO HIINDEX(x);
                IF NOT acyclic_product_category_relationship(x[i],
    local_children) THEN
            RETURN (FALSE);
         END_IF;
      END_REPEAT;
   END_IF;
   RETURN (TRUE);
END_FUNCTION;

FUNCTION all_members_of_es
   (sv : SET OF maths_value;
    es : elementary_space_enumerators ) : LOGICAL;
CONSTANT
      base_types : SET OF STRING := [ 'NUMBER', 'COMPLEX_NUMBER_LITERAL',
'REAL', 'INTEGER', 'LOGICAL', 'BOOLEAN', 'STRING', 'BINARY', 'MATHS_SPACE',
'MATHS_FUNCTION', 'LIST', 'ELEMENTARY_SPACE_ENUMERATORS', 'ORDERING_TYPE',
'LOWER_UPPER', 'SYMMETRY_TYPE', 'ELEMENTARY_FUNCTION_ENUMERATORS',
'OPEN_CLOSED', 'SPACE_CONSTRAINT_TYPE', 'REPACKAGE_OPTIONS',
'EXTENSION_OPTIONS' ];
   END_CONSTANT;
LOCAL
   v : maths_value;
   key_type : STRING := '';
   types : SET OF STRING;
   ge : generic_expression;
   cum : LOGICAL := TRUE;
   vspc : maths_space;
END_LOCAL;
   IF NOT EXISTS(sv) OR NOT EXISTS(es) THEN
      RETURN (FALSE);
   END_IF;
   CASE es OF
      es_numbers :
            key_type := 'NUMBER';
      es_complex_numbers :
            key_type := 'COMPLEX_NUMBER_LITERAL';
      es_reals :
            key_type := 'REAL';
      es_integers :
            key_type := 'INTEGER';
      es_logicals :
            key_type := 'LOGICAL';
      es_booleans :
```

```
            key_type := 'BOOLEAN';
         es_strings :
            key_type := 'STRING';
         es_binarys :
            key_type := 'BINARY';
         es_maths_spaces :
            key_type := 'MATHS_SPACE';
         es_maths_functions :
            key_type := 'MATHS_FUNCTION';
         es_generics :
            RETURN (TRUE);
      END_CASE;
      REPEAT i := 1 TO SIZEOF(sv);
         IF NOT EXISTS(sv[i]) THEN
            RETURN (FALSE);
         END_IF;
         v := simplify_maths_value(sv[i]);
         types := stripped_typeof(v);
         IF key_type IN types THEN
            SKIP;
         END_IF;
         IF (es = es_numbers) AND ('COMPLEX_NUMBER_LITERAL' IN types) THEN
            SKIP;
         END_IF;
         IF SIZEOF(base_types * types) > 0 THEN
            RETURN (FALSE);
         END_IF;
         ge := v;
         IF has_values_space(ge) THEN
            vspc := values_space_of(ge);
            IF NOT subspace_of_es(vspc, es) THEN
               IF NOT compatible_spaces(vspc, make_elementary_space(es)) THEN
                  RETURN (FALSE);
               END_IF;
               cum := UNKNOWN;
            END_IF;
         ELSE
            cum := UNKNOWN;
         END_IF;
         IF cum = FALSE THEN
            RETURN (FALSE);
         END_IF;
      END_REPEAT;
      RETURN (cum);
   END_FUNCTION;

   FUNCTION any_space_satisfies
      (sc : space_constraint_type;
       spc : maths_space ) : BOOLEAN;
   LOCAL
      spc_id : elementary_space_enumerators;
   END_LOCAL;
      IF (sc = sc_equal) OR NOT ('ELEMENTARY_SPACE' IN stripped_typeof(spc)) THEN
         RETURN (FALSE);
      END_IF;
      spc_id := spc\elementary_space.space_id;
      IF sc = sc_subspace THEN
         RETURN (bool(spc_id = es_generics));
      END_IF;
```

```
      IF sc = sc_member THEN
         RETURN (bool((spc_id = es_generics) OR (spc_id = es_maths_spaces)));
      END_IF;
      RETURN (?);
   END_FUNCTION;

   FUNCTION assoc_product_space
      (ts1 : tuple_space;
       ts2 : tuple_space ) : tuple_space;
   LOCAL
      types1 : SET OF STRING := stripped_typeof(ts1);
      types2 : SET OF STRING := stripped_typeof(ts2);
      up1 : uniform_product_space := make_uniform_product_space(the_reals, 1);
      up2 : uniform_product_space := make_uniform_product_space(the_reals, 1);
      lp1 : listed_product_space := the_zero_tuple_space;
      lp2 : listed_product_space := the_zero_tuple_space;
      lps : listed_product_space := the_zero_tuple_space;
      et1 : extended_tuple_space := the_tuples;
      et2 : extended_tuple_space := the_tuples;
      ets : extended_tuple_space := the_tuples;
      use_up1 : BOOLEAN;
      use_up2 : BOOLEAN;
      use_lp1 : BOOLEAN;
      use_lp2 : BOOLEAN;
      factors : LIST OF maths_space := [];
      tspace : tuple_space;
   END_LOCAL;
      IF 'UNIFORM_PRODUCT_SPACE' IN types1 THEN
         up1 := ts1;
         use_up1 := FALSE;
         use_lp1 := FALSE;
      ELSE
         IF 'LISTED_PRODUCT_SPACE' IN types1 THEN
            lp1 := ts1;
            use_up1 := FALSE;
            use_lp1 := FALSE;
         ELSE
            IF NOT ('EXTENDED_TUPLE_SPACE' IN types1) THEN
               RETURN (?);
            END_IF;
            et1 := ts1;
            use_up1 := FALSE;
            use_lp1 := FALSE;
         END_IF;
      END_IF;
      IF 'UNIFORM_PRODUCT_SPACE' IN types2 THEN
         up2 := ts2;
         use_up2 := FALSE;
         use_lp2 := FALSE;
      ELSE
         IF 'LISTED_PRODUCT_SPACE' IN types2 THEN
            lp2 := ts2;
            use_up2 := FALSE;
            use_lp2 := FALSE;
         ELSE
            IF NOT ('EXTENDED_TUPLE_SPACE' IN types2) THEN
               RETURN (?);
            END_IF;
            et2 := ts2;
```

```
      use_up2 := FALSE;
      use_lp2 := FALSE;
   END_IF;
END_IF;
IF use_up1 THEN
   IF use_up2 THEN
      IF up1.base = up2.base THEN
       tspace := make_uniform_product_space(up1.base, up1.exponent +
       up2.exponent);
      ELSE
         factors := [ up1.base, up2.base ];
         tspace := make_listed_product_space(factors);
      END_IF;
   ELSE
      IF use_lp2 THEN
         factors := [ up1.base ];
         factors := factors + lp2.factors;
         tspace := make_listed_product_space(factors);
      ELSE
         tspace := assoc_product_space(up1, et2.base);
         tspace := make_extended_tuple_space(tspace, et2.extender);
      END_IF;
   END_IF;
ELSE
   IF use_lp1 THEN
      IF use_up2 THEN
         factors := [ up2.base ];
         factors := lp1.factors + factors;
         tspace := make_listed_product_space(factors);
      ELSE
         IF use_lp2 THEN
            tspace := make_listed_product_space(lp1.factors + lp2.factors);
         ELSE
            tspace := assoc_product_space(lp1, et2.base);
            tspace := make_extended_tuple_space(tspace, et2.extender);
         END_IF;
      END_IF;
   ELSE
      IF use_up2 THEN
         IF et1.extender = up2.base THEN
            tspace := assoc_product_space(et1.base, up2);
            tspace := make_extended_tuple_space(tspace, et1.extender);
         ELSE
            RETURN (?);
         END_IF;
      ELSE
         IF use_lp2 THEN
            factors := lp2.factors;
            REPEAT i := 1 TO SIZEOF(factors);
               IF et1.extender <> factors[i] THEN
                  RETURN (?);
               END_IF;
            END_REPEAT;
            tspace := assoc_product_space(et1.base, lp2);
            tspace := make_extended_tuple_space(tspace, et1.extender);
         ELSE
            IF et1.extender = et2.extender THEN
               tspace := assoc_product_space(et1, et2.base);
            ELSE
```

```
                  RETURN (?);
               END_IF;
            END_IF;
         END_IF;
      END_IF;
   END_IF;
   RETURN (tspace);
END_FUNCTION;

FUNCTION atan2
   (y : REAL;
    x : REAL ) : REAL;
LOCAL
   r : REAL;
END_LOCAL;
   IF (y = 0.00000) AND (x = 0.00000) THEN
      RETURN (?);
   END_IF;
   r := ATAN(y, x);
   IF x < 0.00000 THEN
      IF y < 0.00000 THEN
         r := r - 3.14159;
      ELSE
         r := r + 3.14159;
      END_IF;
   END_IF;
   RETURN (r);
END_FUNCTION;

FUNCTION bag_to_set
   (the_bag : BAG OF GENERIC : intype ) : SET OF GENERIC : intype;
LOCAL
   the_set : SET OF GENERIC : intype := [];
END_LOCAL;
   IF SIZEOF(the_bag) > 0 THEN
      REPEAT i := 1 TO HIINDEX(the_bag);
         the_set := the_set + the_bag[i];
      END_REPEAT;
   END_IF;
   RETURN (the_set);
END_FUNCTION;

FUNCTION base_axis
   (dim : INTEGER;
    axis1 : direction;
    axis2 : direction;
    axis3 : direction ) : LIST [2:3] OF direction;
LOCAL
   u : LIST [2:3] OF direction;
   factor : REAL;
   d1 : direction;
   d2 : direction;
END_LOCAL;
   IF dim = 3 THEN
      d1 := NVL(normalise(axis3), dummy_gri || direction([ 0.00000, 0.00000,
    1.00000 ]));
      d2 := first_proj_axis(d1, axis1);
      u := [ d2, second_proj_axis(d1, d2, axis2), d1 ];
   ELSE
```

```
      IF EXISTS(axis1) THEN
         d1 := normalise(axis1);
         u := [ d1, orthogonal_complement(d1) ];
         IF EXISTS(axis2) THEN
            factor := dot_product(axis2, u[2]);
            IF factor < 0.00000 THEN
               u[2].direction_ratios[1] := -u[2].direction_ratios[1];
               u[2].direction_ratios[2] := -u[2].direction_ratios[2];
            END_IF;
         END_IF;
      ELSE
         IF EXISTS(axis2) THEN
            d1 := normalise(axis2);
            u := [ orthogonal_complement(d1), d1 ];
            u[1].direction_ratios[1] := -u[1].direction_ratios[1];
            u[1].direction_ratios[2] := -u[1].direction_ratios[2];
         ELSE
              u := [ dummy_gri || direction([ 1.00000, 0.00000 ]), dummy_gri
          || direction([ 0.00000, 1.00000 ]) ];
         END_IF;
      END_IF;
   END_IF;
   RETURN (u);
END_FUNCTION;

FUNCTION bool
   (lgcl : LOGICAL ) : BOOLEAN;
   IF NOT EXISTS(lgcl) THEN
      RETURN (FALSE);
   END_IF;
   IF lgcl <> TRUE THEN
      RETURN (FALSE);
   END_IF;
   RETURN (TRUE);
END_FUNCTION;

FUNCTION build_2axes
   (ref_direction : direction ) : LIST [2:2] OF direction;
LOCAL
      d : direction := NVL(normalise(ref_direction), dummy_gri ||
    direction([ 1.00000, 0.00000 ]));
END_LOCAL;
   RETURN ([ d, orthogonal_complement(d) ]);
END_FUNCTION;

FUNCTION build_axes
   (axis : direction;
    ref_direction : direction ) : LIST [3:3] OF direction;
LOCAL
   d1 : direction;
   d2 : direction;
END_LOCAL;
     d1 := NVL(normalise(axis), dummy_gri || direction([ 0.00000, 0.00000,
    1.00000 ]));
   d2 := first_proj_axis(d1, ref_direction);
   RETURN ([ d2, normalise(cross_product(d1, d2)).orientation, d1 ]);
END_FUNCTION;

FUNCTION check_sparse_index_domain
```

```
   (idxdom : tuple_space;
    base : zero_or_one;
    shape : LIST [1:?] OF positive_integer;
    order : ordering_type ) : BOOLEAN;
LOCAL
   mthspc : maths_space;
   interval : finite_integer_interval;
   i : INTEGER;
END_LOCAL;
   mthspc := factor1(idxdom);
   interval := mthspc;
   IF order = by_rows THEN
      i := 1;
   ELSE
      i := 2;
   END_IF;
    RETURN (bool((interval.min <= base) AND (interval.max >= base +
shape[i])));
END_FUNCTION;

FUNCTION check_sparse_index_to_loc
   (index_range : tuple_space;
    loc_domain : tuple_space ) : BOOLEAN;
LOCAL
   temp : maths_space;
   idx_rng_itvl : finite_integer_interval;
   loc_dmn_itvl : finite_integer_interval;
END_LOCAL;
   temp := factor1(index_range);
   IF schema_prefix + 'TUPLE_SPACE' IN TYPEOF(temp) THEN
      temp := factor1(temp);
   END_IF;
   IF NOT (schema_prefix + 'FINITE_INTEGER_INTERVAL' IN TYPEOF(temp)) THEN
      RETURN (FALSE);
   END_IF;
   idx_rng_itvl := temp;
   temp := factor1(loc_domain);
   IF schema_prefix + 'TUPLE_SPACE' IN TYPEOF(temp) THEN
      temp := factor1(temp);
   END_IF;
   IF NOT (schema_prefix + 'FINITE_INTEGER_INTERVAL' IN TYPEOF(temp)) THEN
      RETURN (FALSE);
   END_IF;
   loc_dmn_itvl := temp;
    RETURN (bool((loc_dmn_itvl.min <= idx_rng_itvl.min) AND (idx_rng_itvl.max
   <= loc_dmn_itvl.max + 1)));
END_FUNCTION;

FUNCTION check_sparse_loc_range
   (locrng : tuple_space;
    base : zero_or_one;
    shape : LIST [1:?] OF positive_integer;
    order : ordering_type ) : BOOLEAN;
LOCAL
   mthspc : maths_space;
   interval : finite_integer_interval;
   i : INTEGER;
END_LOCAL;
   IF space_dimension(locrng) <> 1 THEN
```

```
      RETURN (FALSE);
   END_IF;
   mthspc := factor1(locrng);
   IF NOT (schema_prefix + 'FINITE_INTEGER_INTERVAL' IN TYPEOF(mthspc)) THEN
      RETURN (FALSE);
   END_IF;
   interval := mthspc;
   IF order = by_rows THEN
      i := 2;
   ELSE
      i := 1;
   END_IF;
      RETURN (bool((interval.min >= base) AND (interval.max <= base + shape[i]
    - 1)));
END_FUNCTION;

FUNCTION compare_basis_and_coef
   (basis : LIST [1:?] OF b_spline_basis;
    coef : maths_function ) : BOOLEAN;
LOCAL
   shape : LIST OF positive_integer;
END_LOCAL;
   IF NOT EXISTS(basis) OR NOT EXISTS(coef) THEN
      RETURN (FALSE);
   END_IF;
   shape := shape_of_array(coef);
   IF NOT EXISTS(shape) THEN
      RETURN (FALSE);
   END_IF;

   IF SIZEOF(shape) < SIZEOF(basis) THEN
      RETURN (FALSE);
   END_IF;
   REPEAT i := 1 TO SIZEOF(basis);
      IF (basis[i].num_basis = shape[i]) <> TRUE THEN
         RETURN (FALSE);
      END_IF;
   END_REPEAT;
   RETURN (TRUE);
END_FUNCTION;

FUNCTION compatible_complex_number_regions
   (sp1 : maths_space;
    sp2 : maths_space ) : BOOLEAN;
LOCAL
   typenames : SET OF STRING := stripped_typeof(sp1);
   crgn1 : cartesian_complex_number_region;
   crgn2 : cartesian_complex_number_region;
   prgn1 : polar_complex_number_region;
   prgn2 : polar_complex_number_region;
   prgn1c2 : polar_complex_number_region;
   prgn2c1 : polar_complex_number_region;
   sp1_is_crgn : BOOLEAN;
   sp2_is_crgn : BOOLEAN;
END_LOCAL;
   IF 'CARTESIAN_COMPLEX_NUMBER_REGION' IN typenames THEN
      sp1_is_crgn := TRUE;
      crgn1 := sp1;
   ELSE
```

```
      IF 'POLAR_COMPLEX_NUMBER_REGION' IN typenames THEN
         sp1_is_crgn := FALSE;
         prgn1 := sp1;
      ELSE
         RETURN (TRUE);
      END_IF;
   END_IF;
   typenames := stripped_typeof(sp2);
   IF 'CARTESIAN_COMPLEX_NUMBER_REGION' IN typenames THEN
      sp2_is_crgn := TRUE;
      crgn2 := sp2;
   ELSE
      IF 'POLAR_COMPLEX_NUMBER_REGION' IN typenames THEN
         sp2_is_crgn := FALSE;
         prgn2 := sp2;
      ELSE
         RETURN (TRUE);
      END_IF;
   END_IF;
   IF sp1_is_crgn AND sp2_is_crgn THEN
      RETURN (compatible_intervals(crgn1.real_constraint,
    crgn2.real_constraint) AND compatible_intervals(crgn1.imag_constraint,
    crgn2.imag_constraint));
   END_IF;
     IF ((NOT sp1_is_crgn AND NOT sp2_is_crgn) AND (prgn1.centre.real_part =
    prgn2.centre.real_part)) AND (prgn1.centre.imag_part =
    prgn2.centre.imag_part) THEN
    IF NOT compatible_intervals(prgn1.distance_constraint,
    prgn2.distance_constraint) THEN
      RETURN (FALSE);
      END_IF;
         IF compatible_intervals(prgn1.direction_constraint,
       prgn2.direction_constraint) THEN
      RETURN (TRUE);
      END_IF;
        IF (prgn1.direction_constraint.max > 3.14159) AND
       (prgn2.direction_constraint.max < 3.14159) THEN
       RETURN (compatible_intervals(prgn2.direction_constraint,
       make_finite_real_interval(-3.14159, open,
       prgn1.direction_constraint.max - 2.00000 * 3.14159,
       prgn1.direction_constraint.max_closure)));
      END_IF;
       IF (prgn2.direction_constraint.max > 3.14159) AND
       (prgn1.direction_constraint.max < 3.14159) THEN
       RETURN (compatible_intervals(prgn1.direction_constraint,
       make_finite_real_interval(-3.14159, open,
       prgn2.direction_constraint.max - 2.00000 * 3.14159,
       prgn2.direction_constraint.max_closure)));
      END_IF;
      RETURN (FALSE);
   END_IF;
   IF sp1_is_crgn AND NOT sp2_is_crgn THEN
      crgn2 := enclose_pregion_in_cregion(prgn2);
      prgn1 := enclose_cregion_in_pregion(crgn1, prgn2.centre);
    RETURN (compatible_complex_number_regions(crgn1, crgn2) AND
    compatible_complex_number_regions(prgn1, prgn2));
   END_IF;
   IF NOT sp1_is_crgn AND sp2_is_crgn THEN
      crgn1 := enclose_pregion_in_cregion(prgn1);
```

```
      prgn2 := enclose_cregion_in_pregion(crgn2, prgn1.centre);
    RETURN (compatible_complex_number_regions(crgn1, crgn2) AND
    compatible_complex_number_regions(prgn1, prgn2));
   END_IF;
   prgn1c2 := enclose_pregion_in_pregion(prgn1, prgn2.centre);
   prgn2c1 := enclose_pregion_in_pregion(prgn2, prgn1.centre);
    RETURN (compatible_complex_number_regions(prgn1, prgn2c1) AND
    compatible_complex_number_regions(prgn1c2, prgn2));
END_FUNCTION;

FUNCTION compatible_es_values
   (esval1 : elementary_space_enumerators;
    esval2 : elementary_space_enumerators ) : BOOLEAN;
LOCAL
   esval1_is_numeric : LOGICAL;
   esval2_is_numeric : LOGICAL;
END_LOCAL;
    IF ((esval1 = esval2) OR (esval1 = es_generics)) OR (esval2 = es_generics)
    THEN
    RETURN (TRUE);
   END_IF;
   esval1_is_numeric := (esval1 >= es_numbers) AND (esval1 <= es_integers);
   esval2_is_numeric := (esval2 >= es_numbers) AND (esval2 <= es_integers);
    IF esval1_is_numeric AND (esval2 = es_numbers) OR esval2_is_numeric AND
    (esval1 = es_numbers) THEN
      RETURN (TRUE);
   END_IF;
   IF esval1_is_numeric XOR esval2_is_numeric THEN
      RETURN (FALSE);
   END_IF;
    IF (esval1 = es_logicals) AND (esval2 = es_booleans) OR (esval1 =
    es_booleans) AND (esval2 = es_logicals) THEN
      RETURN (TRUE);
   END_IF;
   RETURN (FALSE);
END_FUNCTION;

FUNCTION compatible_intervals
   (sp1 : maths_space;
    sp2 : maths_space ) : BOOLEAN;
LOCAL
   amin : REAL;
   amax : REAL;
END_LOCAL;
   IF min_exists(sp1) AND max_exists(sp2) THEN
      amin := real_min(sp1);
      amax := real_max(sp2);
      IF amin > amax THEN
         RETURN (FALSE);
      END_IF;
      IF amin = amax THEN
         RETURN (min_included(sp1) AND max_included(sp2));
      END_IF;
   END_IF;
   IF min_exists(sp2) AND max_exists(sp1) THEN
      amin := real_min(sp2);
      amax := real_max(sp1);
      IF amin > amax THEN
         RETURN (FALSE);
```

```
      END_IF;
      IF amin = amax THEN
         RETURN (min_included(sp2) AND max_included(sp1));
      END_IF;
   END_IF;
   RETURN (TRUE);
END_FUNCTION;

FUNCTION compatible_spaces
   (sp1 : maths_space;
    sp2 : maths_space ) : BOOLEAN;
LOCAL
   types1 : SET OF STRING := stripped_typeof(sp1);
   types2 : SET OF STRING := stripped_typeof(sp2);
   lgcl : LOGICAL := UNKNOWN;
   m : INTEGER;
   n : INTEGER;
   s1 : maths_space;
   s2 : maths_space;
END_LOCAL;
   IF 'FINITE_SPACE' IN types1 THEN
      REPEAT i := 1 TO SIZEOF(sp1\finite_space.members);
         lgcl := member_of(sp1\finite_space.members[i], sp2);
         IF lgcl <> FALSE THEN
            RETURN (TRUE);
         END_IF;
      END_REPEAT;
      RETURN (FALSE);
   END_IF;
   IF 'FINITE_SPACE' IN types2 THEN
      REPEAT i := 1 TO SIZEOF(sp2\finite_space.members);
         lgcl := member_of(sp2\finite_space.members[i], sp1);
         IF lgcl <> FALSE THEN
            RETURN (TRUE);
         END_IF;
      END_REPEAT;
      RETURN (FALSE);
   END_IF;
   IF 'ELEMENTARY_SPACE' IN types1 THEN
      IF sp1\elementary_space.space_id = es_generics THEN
         RETURN (TRUE);
      END_IF;
      IF 'ELEMENTARY_SPACE' IN types2 THEN
       RETURN (compatible_es_values(sp1\elementary_space.space_id,
       sp2\elementary_space.space_id));
      END_IF;
       IF (('FINITE_INTEGER_INTERVAL' IN types2) OR
       ('INTEGER_INTERVAL_FROM_MIN' IN types2)) OR ('INTEGER_INTERVAL_TO_MAX'
       IN types2) THEN
         RETURN (compatible_es_values(sp1\elementary_space.space_id,
       es_integers));
      END_IF;
    IF (('FINITE_REAL_INTERVAL' IN types2) OR ('REAL_INTERVAL_FROM_MIN' IN
    types2)) OR ('REAL_INTERVAL_TO_MAX' IN types2) THEN
     RETURN (compatible_es_values(sp1\elementary_space.space_id, es_reals));
     END_IF;
    IF ('CARTESIAN_COMPLEX_NUMBER_REGION' IN types2) OR
    ('POLAR_COMPLEX_NUMBER_REGION' IN types2) THEN
```

```
 RETURN (compatible_es_values(sp1\elementary_space.space_id,
es_complex_numbers));
 END_IF;
 IF 'TUPLE_SPACE' IN types2 THEN
 RETURN (FALSE);
 END_IF;
 IF 'FUNCTION_SPACE' IN types2 THEN
 RETURN (bool(sp1\elementary_space.space_id = es_maths_functions));
   END_IF;
   RETURN (TRUE);
END_IF;
IF 'ELEMENTARY_SPACE' IN types2 THEN
   IF sp2\elementary_space.space_id = es_generics THEN
      RETURN (TRUE);
   END_IF;
 IF (('FINITE_INTEGER_INTERVAL' IN types1) OR ('INTEGER_INTERVAL_FROM_MIN'
 IN types1)) OR ('INTEGER_INTERVAL_TO_MAX' IN types1) THEN
      RETURN (compatible_es_values(sp2\elementary_space.space_id,
      es_integers));
   END_IF;
 IF (('FINITE_REAL_INTERVAL' IN types1) OR ('REAL_INTERVAL_FROM_MIN' IN
 types1)) OR ('REAL_INTERVAL_TO_MAX' IN types1) THEN
      RETURN (compatible_es_values(sp2\elementary_space.space_id,
      es_reals));
   END_IF;
 IF ('CARTESIAN_COMPLEX_NUMBER_REGION' IN types1) OR
 ('POLAR_COMPLEX_NUMBER_REGION' IN types1) THEN
      RETURN (compatible_es_values(sp2\elementary_space.space_id,
      es_complex_numbers));
   END_IF;
   IF 'TUPLE_SPACE' IN types1 THEN
      RETURN (FALSE);
   END_IF;
   IF 'FUNCTION_SPACE' IN types1 THEN
      RETURN (bool(sp2\elementary_space.space_id = es_maths_functions));
   END_IF;
   RETURN (TRUE);
END_IF;
IF subspace_of_es(sp1, es_integers) THEN
   IF subspace_of_es(sp2, es_integers) THEN
      RETURN (compatible_intervals(sp1, sp2));
   END_IF;
   RETURN (FALSE);
END_IF;
IF subspace_of_es(sp2, es_integers) THEN
   RETURN (FALSE);
END_IF;
IF subspace_of_es(sp1, es_reals) THEN
   IF subspace_of_es(sp2, es_reals) THEN
      RETURN (compatible_intervals(sp1, sp2));
   END_IF;
   RETURN (FALSE);
END_IF;
IF subspace_of_es(sp2, es_reals) THEN
   RETURN (FALSE);
END_IF;
IF subspace_of_es(sp1, es_complex_numbers) THEN
   IF subspace_of_es(sp2, es_complex_numbers) THEN
      RETURN (compatible_complex_number_regions(sp1, sp2));
```

```
      END_IF;
      RETURN (FALSE);
   END_IF;
   IF subspace_of_es(sp2, es_complex_numbers) THEN
      RETURN (FALSE);
   END_IF;
   IF 'UNIFORM_PRODUCT_SPACE' IN types1 THEN
      IF 'UNIFORM_PRODUCT_SPACE' IN types2 THEN
          IF sp1\uniform_product_space.exponent <>
       sp2\uniform_product_space.exponent THEN
            RETURN (FALSE);
         END_IF;
            RETURN (compatible_spaces(sp1\uniform_product_space.base,
       sp2\uniform_product_space.base));
      END_IF;
      IF 'LISTED_PRODUCT_SPACE' IN types2 THEN
         n := SIZEOF(sp2\listed_product_space.factors);
         IF sp1\uniform_product_space.exponent <> n THEN
            RETURN (FALSE);
         END_IF;
         REPEAT i := 1 TO n;
            IF NOT compatible_spaces(sp1\uniform_product_space.base,
           sp2\listed_product_space.factors[i]) THEN
               RETURN (FALSE);
            END_IF;
         END_REPEAT;
         RETURN (TRUE);
      END_IF;
      IF 'EXTENDED_TUPLE_SPACE' IN types2 THEN
         m := sp1\uniform_product_space.exponent;
         n := space_dimension(sp2\extended_tuple_space.base);
         IF m < n THEN
            RETURN (FALSE);
         END_IF;
         IF m = n THEN
            RETURN (compatible_spaces(sp1, sp2\extended_tuple_space.base));
         END_IF;
            RETURN (compatible_spaces(sp1,
           assoc_product_space(sp2\extended_tuple_space.base,
           make_uniform_product_space(sp2\extended_tuple_space.extender, m -
           n))));
      END_IF;
      IF 'FUNCTION_SPACE' IN types2 THEN
         RETURN (FALSE);
      END_IF;
      RETURN (TRUE);
   END_IF;
   IF 'LISTED_PRODUCT_SPACE' IN types1 THEN
      n := SIZEOF(sp1\listed_product_space.factors);
      IF 'UNIFORM_PRODUCT_SPACE' IN types2 THEN
         IF n <> sp2\uniform_product_space.exponent THEN
            RETURN (FALSE);
         END_IF;
         REPEAT i := 1 TO n;
            IF NOT compatible_spaces(sp2\uniform_product_space.base,
           sp1\listed_product_space.factors[i]) THEN
               RETURN (FALSE);
            END_IF;
         END_REPEAT;
```

```
      RETURN (TRUE);
    END_IF;
    IF 'LISTED_PRODUCT_SPACE' IN types2 THEN
      IF n <> SIZEOF(sp2\listed_product_space.factors) THEN
        RETURN (FALSE);
      END_IF;
      REPEAT i := 1 TO n;
       IF NOT compatible_spaces(sp1\listed_product_space.factors[i],
       sp2\listed_product_space.factors[i]) THEN
           RETURN (FALSE);
       END_IF;
      END_REPEAT;
      RETURN (TRUE);
    END_IF;
    IF 'EXTENDED_TUPLE_SPACE' IN types2 THEN
      m := space_dimension(sp2\extended_tuple_space.base);
      IF n < m THEN
        RETURN (FALSE);
      END_IF;
      IF n = m THEN
        RETURN (compatible_spaces(sp1, sp2\extended_tuple_space.base));
      END_IF;
      RETURN (compatible_spaces(sp1,
     assoc_product_space(sp2\extended_tuple_space.base,
     make_uniform_product_space(sp2\extended_tuple_space.extender, n -
     m))));
    END_IF;
    IF schema_prefix + 'FUNCTION_SPACE' IN types2 THEN
      RETURN (FALSE);
    END_IF;
    RETURN (TRUE);
  END_IF;
  IF 'EXTENDED_TUPLE_SPACE' IN types1 THEN
     IF ('UNIFORM_PRODUCT_SPACE' IN types2) OR ('LISTED_PRODUCT_SPACE' IN
     types2) THEN
       RETURN (compatible_spaces(sp2, sp1));
    END_IF;
    IF 'EXTENDED_TUPLE_SPACE' IN types2 THEN
      IF NOT compatible_spaces(sp1\extended_tuple_space.extender,
     sp2\extended_tuple_space.extender) THEN
          RETURN (FALSE);
       END_IF;
       n := space_dimension(sp1\extended_tuple_space.base);
       m := space_dimension(sp2\extended_tuple_space.base);
       IF n < m THEN
         RETURN
        (compatible_spaces(assoc_product_space(sp1\extended_tuple_space.bas
        e, make_uniform_product_space(sp1\extended_tuple_space.extender, m
        - n)), sp2\extended_tuple_space.base));
       END_IF;
       IF n = m THEN
        RETURN (compatible_spaces(sp1\extended_tuple_space.base,
        sp2\extended_tuple_space.base));
       END_IF;
       IF n > m THEN
        RETURN (compatible_spaces(sp1\extended_tuple_space.base,
        assoc_product_space(sp2\extended_tuple_space.base,
        make_uniform_product_space(sp2\extended_tuple_space.extender, n -
        m))));
```

```
      END_IF;
   END_IF;
   IF 'FUNCTION_SPACE' IN types2 THEN
      RETURN (FALSE);
   END_IF;
   RETURN (TRUE);
END_IF;
IF 'FUNCTION_SPACE' IN types1 THEN
   IF 'FUNCTION_SPACE' IN types2 THEN
      s1 := sp1\function_space.domain_argument;
      s2 := sp2\function_space.domain_argument;
      CASE sp1\function_space.domain_constraint OF
         sc_equal :
               BEGIN
                  CASE sp2\function_space.domain_constraint OF
                     sc_equal :
                     lgcl := subspace_of(s1, s2) AND subspace_of(s2, s1);
                     sc_subspace :
                     lgcl := subspace_of(s1, s2);
                     sc_member :
                     lgcl := member_of(s1, s2);
                  END_CASE;
               END;
         sc_subspace :
               BEGIN
                  CASE sp2\function_space.domain_constraint OF
                     sc_equal :
                           lgcl := subspace_of(s2, s1);
                     sc_subspace :
                           lgcl := compatible_spaces(s1, s2);
                     sc_member :
                           lgcl := UNKNOWN;
                  END_CASE;
               END;
         sc_member :
               BEGIN
                  CASE sp2\function_space.domain_constraint OF
                     sc_equal :
                           lgcl := member_of(s2, s1);
                     sc_subspace :
                           lgcl := UNKNOWN;
                     sc_member :
                           lgcl := compatible_spaces(s1, s2);
                  END_CASE;
               END;
      END_CASE;
      IF lgcl = FALSE THEN
         RETURN (FALSE);
      END_IF;
      s1 := sp1\function_space.range_argument;
      s2 := sp2\function_space.range_argument;
      CASE sp1\function_space.range_constraint OF
         sc_equal :
               BEGIN
                  CASE sp2\function_space.range_constraint OF
                     sc_equal :
                           lgcl := subspace_of(s1, s2) AND subspace_of(s2,
                        s1);
                     sc_subspace :
```

```
                              lgcl := subspace_of(s1, s2);
                        sc_member :
                              lgcl := member_of(s1, s2);
                     END_CASE;
                  END;
            sc_subspace :
                  BEGIN
                     CASE sp2\function_space.range_constraint OF
                        sc_equal :
                              lgcl := subspace_of(s2, s1);
                        sc_subspace :
                              lgcl := compatible_spaces(s1, s2);
                        sc_member :
                              lgcl := UNKNOWN;
                     END_CASE;
                  END;
            sc_member :
                  BEGIN
                     CASE sp2\function_space.range_constraint OF
                        sc_equal :
                              lgcl := member_of(s2, s1);
                        sc_subspace :
                              lgcl := UNKNOWN;
                        sc_member :
                              lgcl := compatible_spaces(s1, s2);
                     END_CASE;
                  END;
         END_CASE;
         IF lgcl = FALSE THEN
            RETURN (FALSE);
         END_IF;
         RETURN (TRUE);
      END_IF;
      RETURN (TRUE);
   END_IF;
   RETURN (TRUE);
END_FUNCTION;

FUNCTION composable_sequence
   (operands : LIST [2:?] OF maths_function ) : BOOLEAN;
   REPEAT i := 1 TO SIZEOF(operands) - 1;
            IF NOT compatible_spaces(operands[i].range, operands[(i +
    1)].domain) THEN
         RETURN (FALSE);
      END_IF;
   END_REPEAT;
   RETURN (TRUE);
END_FUNCTION;

FUNCTION convert_to_literal
   (val : maths_atom ) : generic_literal;
LOCAL
   types : SET OF STRING := TYPEOF(val);
END_LOCAL;
   IF 'INTEGER' IN types THEN
      RETURN (make_int_literal(val));
   END_IF;
   IF 'REAL' IN types THEN
      RETURN (make_real_literal(val));
```

```
      END_IF;
      IF 'BOOLEAN' IN types THEN
         RETURN (make_boolean_literal(val));
      END_IF;
      IF 'STRING' IN types THEN
         RETURN (make_string_literal(val));
      END_IF;
      IF 'LOGICAL' IN types THEN
         RETURN (make_logical_literal(val));
      END_IF;
      IF 'BINARY' IN types THEN
         RETURN (make_binary_literal(val));
      END_IF;
      IF schema_prefix + 'MATHS_ENUM_ATOM' IN types THEN
         RETURN (make_maths_enum_literal(val));
      END_IF;
      RETURN (?);
   END_FUNCTION;

   FUNCTION convert_to_maths_function
      (func : maths_function_select ) : maths_function;
   LOCAL
      efenum : elementary_function_enumerators;
      mthfun : maths_function;
   END_LOCAL;
      IF schema_prefix + 'MATHS_FUNCTION' IN TYPEOF(func) THEN
         mthfun := func;
      ELSE
         efenum := func;
         mthfun := make_elementary_function(efenum);
      END_IF;
      RETURN (mthfun);
   END_FUNCTION;

   FUNCTION convert_to_maths_value
      (val : GENERIC : G ) : maths_value;
   LOCAL
      types : SET OF STRING := TYPEOF(val);
      ival : maths_integer;
      rval : maths_real;
      nval : maths_number;
      tfval : maths_boolean;
      lval : maths_logical;
      sval : maths_string;
      bval : maths_binary;
      tval : maths_tuple := the_empty_maths_tuple;
      mval : maths_value;
   END_LOCAL;
      IF schema_prefix + 'MATHS_VALUE' IN types THEN
         RETURN (val);
      END_IF;
      IF 'INTEGER' IN types THEN
         ival := val;
         RETURN (ival);
      END_IF;
      IF 'REAL' IN types THEN
         rval := val;
         RETURN (rval);
      END_IF;
```

```
   IF 'NUMBER' IN types THEN
      nval := val;
      RETURN (nval);
   END_IF;
   IF 'BOOLEAN' IN types THEN
      tfval := val;
      RETURN (tfval);
   END_IF;
   IF 'LOGICAL' IN types THEN
      lval := val;
      RETURN (lval);
   END_IF;
   IF 'STRING' IN types THEN
      sval := val;
      RETURN (sval);
   END_IF;
   IF 'BINARY' IN types THEN
      bval := val;
      RETURN (bval);
   END_IF;
   IF 'LIST' IN types THEN
      REPEAT i := 1 TO SIZEOF(val);
         mval := convert_to_maths_value(val[i]);
         IF NOT EXISTS(mval) THEN
            RETURN (?);
         END_IF;
         INSERT( tval, mval, i - 1 );
      END_REPEAT;
      RETURN (tval);
   END_IF;
   RETURN (?);
END_FUNCTION;

FUNCTION convert_to_operand
   (val : maths_value ) : generic_expression;
LOCAL
   types : SET OF STRING := stripped_typeof(val);
END_LOCAL;
   IF 'GENERIC_EXPRESSION' IN types THEN
      RETURN (val);
   END_IF;
   IF 'MATHS_ATOM' IN types THEN
      RETURN (convert_to_literal(val));
   END_IF;
   IF 'ATOM_BASED_VALUE' IN types THEN
      RETURN (make_atom_based_literal(val));
   END_IF;
   IF 'MATHS_TUPLE' IN types THEN
      RETURN (make_maths_tuple_literal(val));
   END_IF;
   RETURN (?);
END_FUNCTION;

FUNCTION convert_to_operands
   (values : AGGREGATE OF maths_value ) : LIST OF generic_expression;
LOCAL
   operands : LIST OF generic_expression := [];
   loc : INTEGER := 0;
END_LOCAL;
```

```
      IF NOT EXISTS(values) THEN
         RETURN (?);
      END_IF;
      REPEAT i := LOINDEX(values) TO HIINDEX(values);
         INSERT( operands, convert_to_operand(values[i]), loc );
         loc := loc + 1;
      END_REPEAT;
      RETURN (operands);
   END_FUNCTION;

   FUNCTION convert_to_operands_prcmfn
      (srcdom : maths_space_or_function;
       prepfun : LIST OF maths_function;
       finfun : maths_function_select ) : LIST [2:?] OF generic_expression;
   LOCAL
      operands : LIST OF generic_expression := [];
   END_LOCAL;
      INSERT( operands, srcdom, 0 );
      REPEAT i := 1 TO SIZEOF(prepfun);
         INSERT( operands, prepfun[i], i );
      END_REPEAT;
      INSERT( operands, convert_to_maths_function(finfun), SIZEOF(prepfun) + 1 );
      RETURN (operands);
   END_FUNCTION;

   FUNCTION cross_product
      (arg1 : direction;
       arg2 : direction ) : vector;
   LOCAL
      mag : REAL;
      res : direction;
      v1 : LIST [3:3] OF REAL;
      v2 : LIST [3:3] OF REAL;
      result : vector;
   END_LOCAL;
            IF (NOT EXISTS(arg1) OR (arg1.dim = 2)) OR (NOT EXISTS(arg2) OR
       (arg2.dim = 2)) THEN
         RETURN (?);
      ELSE
         BEGIN
            v1 := normalise(arg1).direction_ratios;
            v2 := normalise(arg2).direction_ratios;
             res := dummy_gri || direction([ (v1[2] * v2[3] - v1[3] * v2[2]),
             (v1[3] * v2[1] - v1[1] * v2[3]), (v1[1] * v2[2] - v1[2] *
             v2[1]) ]);
            mag := 0.00000;
            REPEAT i := 1 TO 3;
               mag := mag + res.direction_ratios[i] * res.direction_ratios[i];
            END_REPEAT;
            IF mag > 0.00000 THEN
               result := dummy_gri || vector(res, SQRT(mag));
            ELSE
               result := dummy_gri || vector(arg1, 0.00000);
            END_IF;
            RETURN (result);
         END;
      END_IF;
   END_FUNCTION;
```

```
FUNCTION definite_integral_check
   (domain : tuple_space;
    vrblint : input_selector;
    lowerinf : BOOLEAN;
    upperinf : BOOLEAN ) : BOOLEAN;
LOCAL
   domn : tuple_space := domain;
   fspc : maths_space;
   dim : nonnegative_integer;
   k : positive_integer;
END_LOCAL;
    IF (space_dimension(domain) = 1) AND (schema_prefix + 'TUPLE_SPACE' IN
    TYPEOF(factor1(domain))) THEN
      domn := factor1(domain);
   END_IF;
   dim := space_dimension(domn);
   k := vrblint;
   IF k > dim THEN
      RETURN (FALSE);
   END_IF;
   fspc := factor_space(domn, k);
   IF NOT (schema_prefix + 'REAL_INTERVAL' IN TYPEOF(fspc)) THEN
      RETURN (FALSE);
   END_IF;
   IF lowerinf AND min_exists(fspc) THEN
      RETURN (FALSE);
   END_IF;
   IF upperinf AND max_exists(fspc) THEN
      RETURN (FALSE);
   END_IF;
   RETURN (TRUE);
END_FUNCTION;

FUNCTION definite_integral_expr_check
   (operands : LIST [2:?] OF generic_expression;
    lowerinf : BOOLEAN;
    upperinf : BOOLEAN ) : BOOLEAN;
LOCAL
   nops : INTEGER := 2;
   vspc : maths_space;
   dim : nonnegative_integer;
   k : positive_integer;
   bspc : maths_space;
END_LOCAL;
   IF NOT lowerinf THEN
      nops := nops + 1;
   END_IF;
   IF NOT upperinf THEN
      nops := nops + 1;
   END_IF;
   IF SIZEOF(operands) <> nops THEN
      RETURN (FALSE);
   END_IF;
   IF NOT ('GENERIC_VARIABLE' IN stripped_typeof(operands[2])) THEN
      RETURN (FALSE);
   END_IF;
   IF NOT has_values_space(operands[2]) THEN
      RETURN (FALSE);
   END_IF;
```

```
      vspc := values_space_of(operands[2]);
      IF NOT ('REAL_INTERVAL' IN stripped_typeof(vspc)) THEN
         RETURN (FALSE);
      END_IF;
      IF lowerinf THEN
         IF min_exists(vspc) THEN
            RETURN (FALSE);
         END_IF;
         k := 3;
      ELSE
         IF NOT has_values_space(operands[3]) THEN
            RETURN (FALSE);
         END_IF;
         bspc := values_space_of(operands[3]);
         IF NOT compatible_spaces(bspc, vspc) THEN
            RETURN (FALSE);
         END_IF;
         k := 4;
      END_IF;
      IF upperinf THEN
         IF max_exists(vspc) THEN
            RETURN (FALSE);
         END_IF;
      ELSE
         IF NOT has_values_space(operands[k]) THEN
            RETURN (FALSE);
         END_IF;
         bspc := values_space_of(operands[k]);
         IF NOT compatible_spaces(bspc, vspc) THEN
            RETURN (FALSE);
         END_IF;
      END_IF;
      RETURN (TRUE);
   END_FUNCTION;

   FUNCTION derive_definite_integral_domain
      (igrl : definite_integral_function ) : tuple_space;
      FUNCTION process_product_space
         (spc : product_space;
          idx : INTEGER;
          prefix : INTEGER;
          vdomn : maths_space ) : product_space;
      LOCAL
         uspc : uniform_product_space;
         expnt : INTEGER;
         factors : LIST OF maths_space;
      END_LOCAL;
         IF schema_prefix + 'UNIFORM_PRODUCT_SPACE' IN TYPEOF(spc) THEN
            uspc := spc;
            expnt := uspc.exponent + prefix;
            IF idx <= uspc.exponent THEN
               expnt := expnt - 1;
            END_IF;
            IF expnt = 0 THEN
               RETURN (make_listed_product_space([]));
            ELSE
               RETURN (make_uniform_product_space(uspc.base, expnt));
            END_IF;
         ELSE
```

```
         factors := spc\listed_product_space.factors;
         IF idx <= SIZEOF(factors) THEN
            REMOVE( factors, idx );
         END_IF;
         IF prefix > 0 THEN
            INSERT( factors, vdomn, 0 );
            IF prefix > 1 THEN
               INSERT( factors, vdomn, 0 );
            END_IF;
         END_IF;
         RETURN (make_listed_product_space(factors));
      END_IF;
   END_FUNCTION;
LOCAL
   idomn : tuple_space := igrl.integrand.domain;
   types : SET OF STRING := TYPEOF(idomn);
   idx : INTEGER := igrl.variable_of_integration;
    tupled : BOOLEAN := bool((space_dimension(idomn) = 1) AND (schema_prefix +
    'TUPLE_SPACE' IN types));
   prefix : INTEGER := 0;
   espc : extended_tuple_space;
   vdomn : maths_space;
END_LOCAL;
   IF tupled THEN
      idomn := factor1(idomn);
      types := TYPEOF(idomn);
   END_IF;
   IF igrl.lower_limit_neg_infinity THEN
      prefix := prefix + 1;
   END_IF;
   IF igrl.upper_limit_pos_infinity THEN
      prefix := prefix + 1;
   END_IF;
   vdomn := factor_space(idomn, idx);
   IF schema_prefix + 'EXTENDED_TUPLE_SPACE' IN types THEN
      espc := idomn;
      idomn := make_extended_tuple_space(process_product_space(espc.base, idx,
    prefix, vdomn), espc.extender);
   ELSE
      idomn := process_product_space(idomn, idx, prefix, vdomn);
   END_IF;
   IF tupled THEN
      RETURN (one_tuples_of(idomn));
   ELSE
      RETURN (idomn);
   END_IF;
END_FUNCTION;

FUNCTION derive_dimensional_exponents
   (x : unit ) : dimensional_exponents;
LOCAL
      result : dimensional_exponents := dimensional_exponents(0.00000,
0.00000, 0.00000, 0.00000, 0.00000, 0.00000, 0.00000);
END_LOCAL;
   IF 'ENGINEERING_PROPERTIES_SCHEMA.DERIVED_UNIT' IN TYPEOF(x) THEN
            REPEAT i := LOINDEX(x\derived_unit.elements) TO
    HIINDEX(x\derived_unit.elements);
               result.length_exponent := result.length_exponent +
    x\derived_unit.elements[i]\derived_unit_element.exponent *
```

```
      x\derived_unit.elements[i]\derived_unit_element.unit\named_unit.dimensions
      .length_exponent;
                  result.mass_exponent := result.mass_exponent +
      x\derived_unit.elements[i]\derived_unit_element.exponent *
      x\derived_unit.elements[i]\derived_unit_element.unit\named_unit.dimensions
      .mass_exponent;
                  result.time_exponent := result.time_exponent +
      x\derived_unit.elements[i]\derived_unit_element.exponent *
      x\derived_unit.elements[i]\derived_unit_element.unit\named_unit.dimensions
      .time_exponent;
                  result.electric_current_exponent :=
      result.electric_current_exponent +
      x\derived_unit.elements[i]\derived_unit_element.exponent *
      x\derived_unit.elements[i]\derived_unit_element.unit\named_unit.dimensions
      .electric_current_exponent;
                  result.thermodynamic_temperature_exponent :=
      result.thermodynamic_temperature_exponent +
      x\derived_unit.elements[i]\derived_unit_element.exponent *
      x\derived_unit.elements[i]\derived_unit_element.unit\named_unit.dimensions
      .thermodynamic_temperature_exponent;
                  result.amount_of_substance_exponent :=
      result.amount_of_substance_exponent +
      x\derived_unit.elements[i]\derived_unit_element.exponent *
      x\derived_unit.elements[i]\derived_unit_element.unit\named_unit.dimensions
      .amount_of_substance_exponent;
                  result.luminous_intensity_exponent :=
      result.luminous_intensity_exponent +
      x\derived_unit.elements[i]\derived_unit_element.exponent *
      x\derived_unit.elements[i]\derived_unit_element.unit\named_unit.dimensions
      .luminous_intensity_exponent;
         END_REPEAT;
      ELSE
         result := x\named_unit.dimensions;
      END_IF;
      RETURN (result);
   END_FUNCTION;

   FUNCTION derive_elementary_function_domain
      (ef_val : elementary_function_enumerators ) : tuple_space;
      IF NOT EXISTS(ef_val) THEN
         RETURN (?);
      END_IF;
      CASE ef_val OF
         ef_and :
               RETURN (make_extended_tuple_space(the_zero_tuple_space,
          the_logicals));
         ef_or :
               RETURN (make_extended_tuple_space(the_zero_tuple_space,
          the_logicals));
         ef_not :
               RETURN (make_uniform_product_space(the_logicals, 1));
         ef_xor :
               RETURN (make_uniform_product_space(the_logicals, 2));
         ef_negate_i :
               RETURN (make_uniform_product_space(the_integers, 1));
         ef_add_i :
               RETURN (the_integer_tuples);
         ef_subtract_i :
               RETURN (make_uniform_product_space(the_integers, 2));
```

```
ef_multiply_i :
      RETURN (the_integer_tuples);
ef_divide_i :
      RETURN (make_uniform_product_space(the_integers, 2));
ef_mod_i :
      RETURN (make_uniform_product_space(the_integers, 2));
ef_exponentiate_i :
      RETURN (make_uniform_product_space(the_integers, 2));
ef_eq_i :
      RETURN (make_uniform_product_space(the_integers, 2));
ef_ne_i :
      RETURN (make_uniform_product_space(the_integers, 2));
ef_gt_i :
      RETURN (make_uniform_product_space(the_integers, 2));
ef_lt_i :
      RETURN (make_uniform_product_space(the_integers, 2));
ef_ge_i :
      RETURN (make_uniform_product_space(the_integers, 2));
ef_le_i :
      RETURN (make_uniform_product_space(the_integers, 2));
ef_abs_i :
      RETURN (make_uniform_product_space(the_integers, 1));
ef_if_i :
      RETURN (make_listed_product_space([ the_logicals, the_integers,
 the_integers ]));
ef_negate_r :
      RETURN (make_uniform_product_space(the_reals, 1));
ef_reciprocal_r :
      RETURN (make_uniform_product_space(the_reals, 1));
ef_add_r :
      RETURN (the_real_tuples);
ef_subtract_r :
      RETURN (make_uniform_product_space(the_reals, 2));
ef_multiply_r :
      RETURN (the_real_tuples);
ef_divide_r :
      RETURN (make_uniform_product_space(the_reals, 2));
ef_mod_r :
      RETURN (make_uniform_product_space(the_reals, 2));
ef_exponentiate_r :
      RETURN (make_listed_product_space([ the_nonnegative_reals,
 the_reals ]));
ef_exponentiate_ri :
      RETURN (make_listed_product_space([ the_reals, the_integers ]));
ef_eq_r :
      RETURN (make_uniform_product_space(the_reals, 2));
ef_ne_r :
      RETURN (make_uniform_product_space(the_reals, 2));
ef_gt_r :
      RETURN (make_uniform_product_space(the_reals, 2));
ef_lt_r :
      RETURN (make_uniform_product_space(the_reals, 2));
ef_ge_r :
      RETURN (make_uniform_product_space(the_reals, 2));
ef_le_r :
      RETURN (make_uniform_product_space(the_reals, 2));
ef_abs_r :
      RETURN (make_uniform_product_space(the_reals, 1));
ef_acos_r :
```

```
      RETURN (make_uniform_product_space(the_neg1_one_interval, 1));
ef_asin_r :
      RETURN (make_uniform_product_space(the_neg1_one_interval, 1));
ef_atan2_r :
      RETURN (make_uniform_product_space(the_reals, 2));
ef_cos_r :
      RETURN (make_uniform_product_space(the_reals, 1));
ef_exp_r :
      RETURN (make_uniform_product_space(the_reals, 1));
ef_ln_r :
      RETURN (make_uniform_product_space(the_nonnegative_reals, 1));
ef_log2_r :
      RETURN (make_uniform_product_space(the_nonnegative_reals, 1));
ef_log10_r :
      RETURN (make_uniform_product_space(the_nonnegative_reals, 1));
ef_sin_r :
      RETURN (make_uniform_product_space(the_reals, 1));
ef_sqrt_r :
      RETURN (make_uniform_product_space(the_nonnegative_reals, 1));
ef_tan_r :
      RETURN (make_uniform_product_space(the_reals, 1));
ef_if_r :
      RETURN (make_listed_product_space([ the_logicals, the_reals,
 the_reals ]));
ef_negate_c :
      RETURN (make_uniform_product_space(the_complex_numbers, 1));
ef_reciprocal_c :
      RETURN (make_uniform_product_space(the_complex_numbers, 1));
ef_add_c :
      RETURN (the_complex_tuples);
ef_subtract_c :
      RETURN (make_uniform_product_space(the_complex_numbers, 2));
ef_multiply_c :
      RETURN (the_complex_tuples);
ef_divide_c :
      RETURN (make_uniform_product_space(the_complex_numbers, 2));
ef_exponentiate_c :
      RETURN (make_uniform_product_space(the_complex_numbers, 2));
ef_exponentiate_ci :
     RETURN (make_listed_product_space([ the_complex_numbers,
 the_integers ]));
ef_eq_c :
      RETURN (make_uniform_product_space(the_complex_numbers, 2));
ef_ne_c :
      RETURN (make_uniform_product_space(the_complex_numbers, 2));
ef_conjugate_c :
      RETURN (make_uniform_product_space(the_complex_numbers, 1));
ef_abs_c :
      RETURN (make_uniform_product_space(the_complex_numbers, 1));
ef_arg_c :
      RETURN (make_uniform_product_space(the_complex_numbers, 1));
ef_cos_c :
      RETURN (make_uniform_product_space(the_complex_numbers, 1));
ef_exp_c :
      RETURN (make_uniform_product_space(the_complex_numbers, 1));
ef_ln_c :
      RETURN (make_uniform_product_space(the_complex_numbers, 1));
ef_sin_c :
      RETURN (make_uniform_product_space(the_complex_numbers, 1));
```

```
ef_sqrt_c :
      RETURN (make_uniform_product_space(the_complex_numbers, 1));
ef_tan_c :
      RETURN (make_uniform_product_space(the_complex_numbers, 1));
ef_if_c :
      RETURN (make_listed_product_space([ the_logicals,
 the_complex_numbers, the_complex_numbers ]));
ef_subscript_s :
      RETURN (make_listed_product_space([ the_strings, the_integers ]));
ef_eq_s :
      RETURN (make_uniform_product_space(the_strings, 2));
ef_ne_s :
      RETURN (make_uniform_product_space(the_strings, 2));
ef_gt_s :
      RETURN (make_uniform_product_space(the_strings, 2));
ef_lt_s :
      RETURN (make_uniform_product_space(the_strings, 2));
ef_ge_s :
      RETURN (make_uniform_product_space(the_strings, 2));
ef_le_s :
      RETURN (make_uniform_product_space(the_strings, 2));
ef_subsequence_s :
      RETURN (make_listed_product_space([ the_strings, the_integers,
 the_integers ]));
ef_concat_s :
      RETURN (make_extended_tuple_space(the_zero_tuple_space,
 the_strings));
ef_size_s :
      RETURN (make_uniform_product_space(the_strings, 1));
ef_format :
      RETURN (make_listed_product_space([ the_numbers, the_strings ]));
ef_value :
      RETURN (make_uniform_product_space(the_strings, 1));
ef_like :
      RETURN (make_uniform_product_space(the_strings, 2));
ef_if_s :
      RETURN (make_listed_product_space([ the_logicals, the_strings,
 the_strings ]));
ef_subscript_b :
      RETURN (make_listed_product_space([ the_binarys, the_integers ]));
ef_eq_b :
      RETURN (make_uniform_product_space(the_binarys, 2));
ef_ne_b :
      RETURN (make_uniform_product_space(the_binarys, 2));
ef_gt_b :
      RETURN (make_uniform_product_space(the_binarys, 2));
ef_lt_b :
      RETURN (make_uniform_product_space(the_binarys, 2));
ef_ge_b :
      RETURN (make_uniform_product_space(the_binarys, 2));
ef_le_b :
      RETURN (make_uniform_product_space(the_binarys, 2));
ef_subsequence_b :
      RETURN (make_listed_product_space([ the_binarys, the_integers,
 the_integers ]));
ef_concat_b :
      RETURN (make_extended_tuple_space(the_zero_tuple_space,
 the_binarys));
ef_size_b :
```

```
      RETURN (make_uniform_product_space(the_binarys, 1));
ef_if_b :
      RETURN (make_listed_product_space([ the_logicals, the_binarys,
 the_binarys ]));
ef_subscript_t :
      RETURN (make_listed_product_space([ the_tuples, the_integers ]));
ef_eq_t :
      RETURN (make_uniform_product_space(the_tuples, 2));
ef_ne_t :
      RETURN (make_uniform_product_space(the_tuples, 2));
ef_concat_t :
      RETURN (make_extended_tuple_space(the_zero_tuple_space,
 the_tuples));
ef_size_t :
      RETURN (make_uniform_product_space(the_tuples, 1));
ef_entuple :
      RETURN (the_tuples);
ef_detuple :
      RETURN (make_uniform_product_space(the_generics, 1));
ef_insert :
      RETURN (make_listed_product_space([ the_tuples, the_generics,
 the_integers ]));
ef_remove :
      RETURN (make_listed_product_space([ the_tuples, the_integers ]));
ef_if_t :
      RETURN (make_listed_product_space([ the_logicals, the_tuples,
 the_tuples ]));
ef_sum_it :
      RETURN (make_uniform_product_space(the_integer_tuples, 1));
ef_product_it :
      RETURN (make_uniform_product_space(the_integer_tuples, 1));
ef_add_it :
      RETURN (make_extended_tuple_space(the_integer_tuples,
 the_integer_tuples));
ef_subtract_it :
      RETURN (make_uniform_product_space(the_integer_tuples, 2));
ef_scalar_mult_it :
      RETURN (make_listed_product_space([ the_integers,
 the_integer_tuples ]));
ef_dot_prod_it :
      RETURN (make_uniform_product_space(the_integer_tuples, 2));
ef_sum_rt :
      RETURN (make_uniform_product_space(the_real_tuples, 1));
ef_product_rt :
      RETURN (make_uniform_product_space(the_real_tuples, 1));
ef_add_rt :
      RETURN (make_extended_tuple_space(the_real_tuples,
 the_real_tuples));
ef_subtract_rt :
      RETURN (make_uniform_product_space(the_real_tuples, 2));
ef_scalar_mult_rt :
      RETURN (make_listed_product_space([ the_reals,
 the_real_tuples ]));
ef_dot_prod_rt :
      RETURN (make_uniform_product_space(the_real_tuples, 2));
ef_norm_rt :
      RETURN (make_uniform_product_space(the_real_tuples, 1));
ef_sum_ct :
      RETURN (make_uniform_product_space(the_complex_tuples, 1));
```

```
      ef_product_ct :
            RETURN (make_uniform_product_space(the_complex_tuples, 1));
      ef_add_ct :
            RETURN (make_extended_tuple_space(the_complex_tuples,
       the_complex_tuples));
      ef_subtract_ct :
            RETURN (make_uniform_product_space(the_complex_tuples, 2));
      ef_scalar_mult_ct :
            RETURN (make_listed_product_space([ the_complex_numbers,
       the_complex_tuples ]));
      ef_dot_prod_ct :
            RETURN (make_uniform_product_space(the_complex_tuples, 2));
      ef_norm_ct :
            RETURN (make_uniform_product_space(the_complex_tuples, 1));
      ef_if :
            RETURN (make_listed_product_space([ the_logicals, the_generics,
       the_generics ]));
      ef_ensemble :
            RETURN (the_tuples);
      ef_member_of :
            RETURN (make_listed_product_space([ the_generics,
       the_maths_spaces ]));
   OTHERWISE :
         RETURN (?);
   END_CASE;
END_FUNCTION;

FUNCTION derive_elementary_function_range
   (ef_val : elementary_function_enumerators ) : tuple_space;
   IF NOT EXISTS(ef_val) THEN
      RETURN (?);
   END_IF;

   CASE ef_val OF
      ef_and :
            RETURN (make_uniform_product_space(the_logicals, 1));
      ef_or :
            RETURN (make_uniform_product_space(the_logicals, 1));
      ef_not :
            RETURN (make_uniform_product_space(the_logicals, 1));
      ef_xor :
            RETURN (make_uniform_product_space(the_logicals, 2));
      ef_negate_i :
            RETURN (make_uniform_product_space(the_integers, 1));
      ef_add_i :
            RETURN (make_uniform_product_space(the_integers, 1));
      ef_subtract_i :
            RETURN (make_uniform_product_space(the_integers, 1));
      ef_multiply_i :
            RETURN (make_uniform_product_space(the_integers, 1));
      ef_divide_i :
            RETURN (make_uniform_product_space(the_integers, 1));
      ef_mod_i :
            RETURN (make_uniform_product_space(the_integers, 1));
      ef_exponentiate_i :
            RETURN (make_uniform_product_space(the_integers, 1));
      ef_eq_i :
            RETURN (make_uniform_product_space(the_logicals, 1));
      ef_ne_i :
```

```
      RETURN (make_uniform_product_space(the_logicals, 1));
ef_gt_i :
      RETURN (make_uniform_product_space(the_logicals, 1));
ef_lt_i :
      RETURN (make_uniform_product_space(the_logicals, 1));
ef_ge_i :
      RETURN (make_uniform_product_space(the_logicals, 1));
ef_le_i :
      RETURN (make_uniform_product_space(the_logicals, 1));
ef_abs_i :
      RETURN (make_uniform_product_space(the_integers, 1));
ef_if_i :
      RETURN (make_uniform_product_space(the_integers, 1));
ef_negate_r :
      RETURN (make_uniform_product_space(the_reals, 1));
ef_reciprocal_r :
      RETURN (make_uniform_product_space(the_reals, 1));
ef_add_r :
      RETURN (make_uniform_product_space(the_reals, 1));
ef_subtract_r :
      RETURN (make_uniform_product_space(the_reals, 1));
ef_multiply_r :
      RETURN (make_uniform_product_space(the_reals, 1));
ef_divide_r :
      RETURN (make_uniform_product_space(the_reals, 1));
ef_mod_r :
      RETURN (make_uniform_product_space(the_reals, 1));
ef_exponentiate_r :
      RETURN (make_uniform_product_space(the_reals, 1));
ef_exponentiate_ri :
      RETURN (make_uniform_product_space(the_reals, 1));
ef_eq_r :
      RETURN (make_uniform_product_space(the_logicals, 1));
ef_ne_r :
      RETURN (make_uniform_product_space(the_logicals, 1));
ef_gt_r :
      RETURN (make_uniform_product_space(the_logicals, 1));
ef_lt_r :
      RETURN (make_uniform_product_space(the_logicals, 1));
ef_ge_r :
      RETURN (make_uniform_product_space(the_logicals, 1));
ef_le_r :
      RETURN (make_uniform_product_space(the_logicals, 1));
ef_abs_r :
      RETURN (make_uniform_product_space(the_nonnegative_reals, 1));
ef_acos_r :
      RETURN (make_uniform_product_space(the_zero_pi_interval, 1));
ef_asin_r :
       RETURN (make_uniform_product_space(the_neghalfpi_halfpi_interval,
    1));
ef_atan2_r :
      RETURN (make_uniform_product_space(the_negpi_pi_interval, 1));
ef_cos_r :
      RETURN (make_uniform_product_space(the_neg1_one_interval, 1));
ef_exp_r :
      RETURN (make_uniform_product_space(the_nonnegative_reals, 1));
ef_ln_r :
      RETURN (make_uniform_product_space(the_reals, 1));
ef_log2_r :
```

```
      RETURN (make_uniform_product_space(the_reals, 1));
ef_log10_r :
      RETURN (make_uniform_product_space(the_reals, 1));
ef_sin_r :
      RETURN (make_uniform_product_space(the_neg1_one_interval, 1));
ef_sqrt_r :
      RETURN (make_uniform_product_space(the_nonnegative_reals, 1));
ef_tan_r :
      RETURN (make_uniform_product_space(the_reals, 1));
ef_if_r :
      RETURN (make_uniform_product_space(the_reals, 1));
ef_negate_c :
      RETURN (make_uniform_product_space(the_complex_numbers, 1));
ef_reciprocal_c :
      RETURN (make_uniform_product_space(the_complex_numbers, 1));
ef_add_c :
      RETURN (make_uniform_product_space(the_complex_numbers, 1));
ef_subtract_c :
      RETURN (make_uniform_product_space(the_complex_numbers, 1));
ef_multiply_c :
      RETURN (make_uniform_product_space(the_complex_numbers, 1));
ef_divide_c :
      RETURN (make_uniform_product_space(the_complex_numbers, 1));
ef_exponentiate_c :
      RETURN (make_uniform_product_space(the_complex_numbers, 1));
ef_exponentiate_ci :
      RETURN (make_uniform_product_space(the_complex_numbers, 1));
ef_eq_c :
      RETURN (make_uniform_product_space(the_logicals, 1));
ef_ne_c :
      RETURN (make_uniform_product_space(the_logicals, 1));
ef_conjugate_c :
      RETURN (make_uniform_product_space(the_complex_numbers, 1));
ef_abs_c :
      RETURN (make_uniform_product_space(the_nonnegative_reals, 1));
ef_arg_c :
      RETURN (make_uniform_product_space(the_negpi_pi_interval, 1));
ef_cos_c :
      RETURN (make_uniform_product_space(the_complex_numbers, 1));
ef_exp_c :
      RETURN (make_uniform_product_space(the_complex_numbers, 1));
ef_ln_c :
      RETURN (make_uniform_product_space(the_complex_numbers, 1));
ef_sin_c :
      RETURN (make_uniform_product_space(the_complex_numbers, 1));
ef_sqrt_c :
      RETURN (make_uniform_product_space(the_complex_numbers, 1));
ef_tan_c :
      RETURN (make_uniform_product_space(the_complex_numbers, 1));
ef_if_c :
      RETURN (make_uniform_product_space(the_complex_numbers, 1));
ef_subscript_s :
      RETURN (make_uniform_product_space(the_strings, 1));
ef_eq_s :
      RETURN (make_uniform_product_space(the_logicals, 1));
ef_ne_s :
      RETURN (make_uniform_product_space(the_logicals, 1));
ef_gt_s :
      RETURN (make_uniform_product_space(the_logicals, 1));
```

```
ef_lt_s :
      RETURN (make_uniform_product_space(the_logicals, 1));
ef_ge_s :
      RETURN (make_uniform_product_space(the_logicals, 1));
ef_le_s :
      RETURN (make_uniform_product_space(the_logicals, 1));
ef_subsequence_s :
      RETURN (make_uniform_product_space(the_strings, 1));
ef_concat_s :
      RETURN (make_uniform_product_space(the_strings, 1));
ef_size_s :
      RETURN (make_uniform_product_space(the_integers, 1));
ef_format :
      RETURN (make_uniform_product_space(the_strings, 1));
ef_value :
      RETURN (make_uniform_product_space(the_reals, 1));
ef_like :
      RETURN (make_uniform_product_space(the_booleans, 1));
ef_if_s :
      RETURN (make_uniform_product_space(the_strings, 1));
ef_subscript_b :
      RETURN (make_uniform_product_space(the_binarys, 1));
ef_eq_b :
      RETURN (make_uniform_product_space(the_logicals, 1));
ef_ne_b :
      RETURN (make_uniform_product_space(the_logicals, 1));
ef_gt_b :
      RETURN (make_uniform_product_space(the_logicals, 1));
ef_lt_b :
      RETURN (make_uniform_product_space(the_logicals, 1));
ef_ge_b :
      RETURN (make_uniform_product_space(the_logicals, 1));
ef_le_b :
      RETURN (make_uniform_product_space(the_logicals, 1));
ef_subsequence_b :
      RETURN (make_uniform_product_space(the_binarys, 1));
ef_concat_b :
      RETURN (make_uniform_product_space(the_binarys, 1));
ef_size_b :
      RETURN (make_uniform_product_space(the_integers, 1));
ef_if_b :
      RETURN (make_uniform_product_space(the_binarys, 1));
ef_subscript_t :
      RETURN (make_uniform_product_space(the_generics, 1));
ef_eq_t :
      RETURN (make_uniform_product_space(the_logicals, 1));
ef_ne_t :
      RETURN (make_uniform_product_space(the_logicals, 1));
ef_concat_t :
      RETURN (make_uniform_product_space(the_tuples, 1));
ef_size_t :
      RETURN (make_uniform_product_space(the_integers, 1));
ef_entuple :
      RETURN (make_uniform_product_space(the_tuples, 1));
ef_detuple :
      RETURN (the_tuples);
ef_insert :
      RETURN (make_uniform_product_space(the_tuples, 1));
ef_remove :
```

```
            RETURN (make_uniform_product_space(the_tuples, 1));
         ef_if_t :
            RETURN (make_uniform_product_space(the_tuples, 1));
         ef_sum_it :
            RETURN (make_uniform_product_space(the_integers, 1));
         ef_product_it :
            RETURN (make_uniform_product_space(the_integers, 1));
         ef_add_it :
            RETURN (make_uniform_product_space(the_integer_tuples, 1));
         ef_subtract_it :
            RETURN (make_uniform_product_space(the_integer_tuples, 1));
         ef_scalar_mult_it :
            RETURN (make_uniform_product_space(the_integer_tuples, 1));
         ef_dot_prod_it :
            RETURN (make_uniform_product_space(the_integers, 1));
         ef_sum_rt :
            RETURN (make_uniform_product_space(the_reals, 1));
         ef_product_rt :
            RETURN (make_uniform_product_space(the_reals, 1));
         ef_add_rt :
            RETURN (make_uniform_product_space(the_real_tuples, 1));
         ef_subtract_rt :
            RETURN (make_uniform_product_space(the_real_tuples, 1));
         ef_scalar_mult_rt :
            RETURN (make_uniform_product_space(the_real_tuples, 1));
         ef_dot_prod_rt :
            RETURN (make_uniform_product_space(the_reals, 1));
         ef_norm_rt :
            RETURN (make_uniform_product_space(the_reals, 1));
         ef_sum_ct :
            RETURN (make_uniform_product_space(the_complex_numbers, 1));
         ef_product_ct :
            RETURN (make_uniform_product_space(the_complex_numbers, 1));
         ef_add_ct :
            RETURN (make_uniform_product_space(the_complex_tuples, 1));
         ef_subtract_ct :
            RETURN (make_uniform_product_space(the_complex_tuples, 1));
         ef_scalar_mult_ct :
            RETURN (make_uniform_product_space(the_complex_tuples, 1));
         ef_dot_prod_ct :
            RETURN (make_uniform_product_space(the_complex_numbers, 1));
         ef_norm_ct :
            RETURN (make_uniform_product_space(the_nonnegative_reals, 1));
         ef_if :
            RETURN (make_uniform_product_space(the_generics, 1));
         ef_ensemble :
            RETURN (make_uniform_product_space(the_maths_spaces, 1));
         ef_member_of :
            RETURN (make_uniform_product_space(the_logicals, 1));
      OTHERWISE :
            RETURN (?);
      END_CASE;
   END_FUNCTION;

   FUNCTION derive_finite_function_domain
      (pairs : SET [1:?] OF LIST [2:2] OF maths_value ) : tuple_space;
   LOCAL
      result : SET OF maths_value := [];
   END_LOCAL;
```

```
      result := result + list_selected_components(pairs, 1);
      RETURN (one_tuples_of(make_finite_space(result)));
   END_FUNCTION;

   FUNCTION derive_finite_function_range
      (pairs : SET [1:?] OF LIST [2:2] OF maths_value ) : tuple_space;
   LOCAL
      result : SET OF maths_value := [];
   END_LOCAL;
      result := result + list_selected_components(pairs, 2);
      RETURN (one_tuples_of(make_finite_space(result)));
   END_FUNCTION;

   FUNCTION derive_function_domain
      (func : maths_function ) : tuple_space;
   LOCAL
      typenames : SET OF STRING := stripped_typeof(func);
      tspace : tuple_space := make_listed_product_space([]);
      shape : LIST OF positive_integer;
      sidxs : LIST OF INTEGER := [ 0 ];
      itvl : finite_integer_interval;
      factors : LIST OF finite_integer_interval := [];
      is_uniform : BOOLEAN := TRUE;
   END_LOCAL;
      IF 'FINITE_FUNCTION' IN typenames THEN
         RETURN (derive_finite_function_domain(func\finite_function.pairs));
      END_IF;
      IF 'CONSTANT_FUNCTION' IN typenames THEN
         RETURN (domain_from(func\constant_function.source_of_domain));
      END_IF;
      IF 'SELECTOR_FUNCTION' IN typenames THEN
         RETURN (domain_from(func\selector_function.source_of_domain));
      END_IF;
      IF 'ELEMENTARY_FUNCTION' IN typenames THEN
          RETURN
       (derive_elementary_function_domain(func\elementary_function.func_id));
      END_IF;
      IF 'RESTRICTION_FUNCTION' IN typenames THEN
         RETURN (one_tuples_of(func\restriction_function.operand));
      END_IF;
      IF 'REPACKAGING_FUNCTION' IN typenames THEN
         IF func\repackaging_function.input_repack = ro_nochange THEN
            RETURN (func\repackaging_function.operand.domain);
         END_IF;
         IF func\repackaging_function.input_repack = ro_wrap_as_tuple THEN
            RETURN (factor1(func\repackaging_function.operand.domain));
         END_IF;
         IF func\repackaging_function.input_repack = ro_unwrap_tuple THEN
            RETURN (one_tuples_of(func\repackaging_function.operand.domain));
         END_IF;
         RETURN (?);
      END_IF;
      IF 'REINDEXED_ARRAY_FUNCTION' IN typenames THEN
         shape := shape_of_array(func\unary_generic_expression.operand);
         sidxs := func\reindexed_array_function.starting_indices;
         REPEAT i := 1 TO SIZEOF(shape);
                itvl := make_finite_integer_interval(sidxs[i], sidxs[i] +
          shape[i] - 1);
            INSERT( factors, itvl, i - 1 );
```

```
      IF shape[i] <> shape[1] THEN
         is_uniform := FALSE;
      END_IF;
   END_REPEAT;
   IF is_uniform THEN
      RETURN (make_uniform_product_space(factors[1], SIZEOF(shape)));
   END_IF;
   RETURN (make_listed_product_space(factors));
END_IF;
IF 'SERIES_COMPOSED_FUNCTION' IN typenames THEN
   RETURN (func\series_composed_function.operands[1].domain);
END_IF;
IF 'PARALLEL_COMPOSED_FUNCTION' IN typenames THEN
   RETURN (domain_from(func\parallel_composed_function.source_of_domain));
END_IF;
IF 'EXPLICIT_TABLE_FUNCTION' IN typenames THEN
   shape := func\explicit_table_function.shape;
   sidxs[1] := func\explicit_table_function.index_base;
   REPEAT i := 1 TO SIZEOF(shape);
          itvl := make_finite_integer_interval(sidxs[1], sidxs[1] +
    shape[i] - 1);
      INSERT( factors, itvl, i - 1 );
      IF shape[i] <> shape[1] THEN
         is_uniform := FALSE;
      END_IF;
   END_REPEAT;
   IF is_uniform THEN
      RETURN (make_uniform_product_space(factors[1], SIZEOF(shape)));
   END_IF;
   RETURN (make_listed_product_space(factors));
END_IF;
IF 'HOMOGENEOUS_LINEAR_FUNCTION' IN typenames THEN
         RETURN
(one_tuples_of(make_uniform_product_space(factor1(func\homogeneous_linear_
function.mat.range),
func\homogeneous_linear_function.mat\explicit_table_function.shape[func\ho
mogeneous_linear_function.sum_index])));
END_IF;
IF 'GENERAL_LINEAR_FUNCTION' IN typenames THEN
         RETURN
(one_tuples_of(make_uniform_product_space(factor1(func\general_linear_func
tion.mat.range),
func\general_linear_function.mat\explicit_table_function.shape[func\genera
l_linear_function.sum_index] - 1)));
END_IF;
IF 'B_SPLINE_BASIS' IN typenames THEN
         RETURN
(one_tuples_of(make_finite_real_interval(func\b_spline_basis.repeated_knot
s[func\b_spline_basis.order], closed,
func\b_spline_basis.repeated_knots[(func\b_spline_basis.num_basis + 1)],
closed)));
END_IF;
IF 'B_SPLINE_FUNCTION' IN typenames THEN
   REPEAT i := 1 TO SIZEOF(func\b_spline_function.basis);
          tspace := assoc_product_space(tspace,
    func\b_spline_function.basis[i].domain);
   END_REPEAT;
   RETURN (one_tuples_of(tspace));
END_IF;
```

```
    IF 'RATIONALIZE_FUNCTION' IN typenames THEN
       RETURN (func\rationalize_function.fun.domain);
    END_IF;
    IF 'PARTIAL_DERIVATIVE_FUNCTION' IN typenames THEN
       RETURN (func\partial_derivative_function.derivand.domain);
    END_IF;
    IF 'DEFINITE_INTEGRAL_FUNCTION' IN typenames THEN
       RETURN (derive_definite_integral_domain(func));
    END_IF;
    IF 'ABSTRACTED_EXPRESSION_FUNCTION' IN typenames THEN
       REPEAT i := 1 TO SIZEOF(func\abstracted_expression_function.variables);
              tspace := assoc_product_space(tspace,
        one_tuples_of(values_space_of(func\abstracted_expression_function.varia
        bles[i])));
       END_REPEAT;
       RETURN (tspace);
    END_IF;
    IF 'EXPRESSION_DENOTED_FUNCTION' IN typenames THEN
             RETURN
     (values_space_of(func\expression_denoted_function.expr)\function_space.dom
     ain_argument);
    END_IF;
    IF 'IMPORTED_POINT_FUNCTION' IN typenames THEN
       RETURN (one_tuples_of(make_listed_product_space([])));
    END_IF;
    IF 'IMPORTED_CURVE_FUNCTION' IN typenames THEN
       RETURN (func\imported_curve_function.parametric_domain);
    END_IF;
    IF 'IMPORTED_SURFACE_FUNCTION' IN typenames THEN
       RETURN (func\imported_surface_function.parametric_domain);
    END_IF;
    IF 'IMPORTED_VOLUME_FUNCTION' IN typenames THEN
       RETURN (func\imported_volume_function.parametric_domain);
    END_IF;
    IF 'APPLICATION_DEFINED_FUNCTION' IN typenames THEN
       RETURN (func\application_defined_function.explicit_domain);
    END_IF;
    RETURN (?);
  END_FUNCTION;

  FUNCTION derive_function_range
     (func : maths_function ) : tuple_space;
  LOCAL
     typenames : SET OF STRING := stripped_typeof(func);
     tspace : tuple_space := make_listed_product_space([]);
     m : nonnegative_integer := 0;
     n : nonnegative_integer := 0;
  END_LOCAL;
     IF 'FINITE_FUNCTION' IN typenames THEN
        RETURN (derive_finite_function_range(func\finite_function.pairs));
     END_IF;
     IF 'CONSTANT_FUNCTION' IN typenames THEN
        RETURN
      (one_tuples_of(make_finite_space([ func\constant_function.sole_output ])))
      ;
     END_IF;
     IF 'SELECTOR_FUNCTION' IN typenames THEN
        tspace := func.domain;
```

```
    IF (space_dimension(tspace) = 1) AND (schema_prefix + 'TUPLE_SPACE' IN
 TYPEOF(tspace)) THEN
      tspace := factor1(tspace);
   END_IF;
     RETURN (one_tuples_of(factor_space(tspace,
    func\selector_function.selector)));
END_IF;
IF 'ELEMENTARY_FUNCTION' IN typenames THEN
     RETURN
 (derive_elementary_function_range(func\elementary_function.func_id));
END_IF;
IF 'RESTRICTION_FUNCTION' IN typenames THEN
   RETURN (one_tuples_of(func\restriction_function.operand));
END_IF;
IF 'REPACKAGING_FUNCTION' IN typenames THEN
   tspace := func\repackaging_function.operand.range;
   IF func\repackaging_function.output_repack = ro_wrap_as_tuple THEN
      tspace := one_tuples_of(tspace);
   END_IF;
   IF func\repackaging_function.output_repack = ro_unwrap_tuple THEN
      tspace := factor1(tspace);
   END_IF;
   IF func\repackaging_function.selected_output > 0 THEN
        tspace := one_tuples_of(factor_space(tspace,
    func\repackaging_function.selected_output));
   END_IF;
   RETURN (tspace);
END_IF;
IF 'REINDEXED_ARRAY_FUNCTION' IN typenames THEN
   RETURN (func\unary_generic_expression.operand\maths_function.range);
END_IF;
IF 'SERIES_COMPOSED_FUNCTION' IN typenames THEN
    RETURN
 (func\series_composed_function.operands[SIZEOF(func\series_composed_functi
 on.operands)].range);
END_IF;
IF 'PARALLEL_COMPOSED_FUNCTION' IN typenames THEN
   RETURN (func\parallel_composed_function.final_function.range);
END_IF;
IF 'EXPLICIT_TABLE_FUNCTION' IN typenames THEN
   IF 'LISTED_REAL_DATA' IN typenames THEN
      RETURN (one_tuples_of(the_reals));
   END_IF;
   IF 'LISTED_INTEGER_DATA' IN typenames THEN
      RETURN (one_tuples_of(the_integers));
   END_IF;
   IF 'LISTED_LOGICAL_DATA' IN typenames THEN
      RETURN (one_tuples_of(the_logicals));
   END_IF;
   IF 'LISTED_STRING_DATA' IN typenames THEN
      RETURN (one_tuples_of(the_strings));
   END_IF;
   IF 'LISTED_COMPLEX_NUMBER_DATA' IN typenames THEN
      RETURN (one_tuples_of(the_complex_numbers));
   END_IF;
   IF 'LISTED_DATA' IN typenames THEN
      RETURN (one_tuples_of(func\listed_data.value_range));
   END_IF;
   IF 'EXTERNALLY_LISTED_DATA' IN typenames THEN
```

```
      RETURN (one_tuples_of(func\externally_listed_data.value_range));
   END_IF;
   IF 'LINEARIZED_TABLE_FUNCTION' IN typenames THEN
      RETURN (func\linearized_table_function.source.range);
   END_IF;
   IF 'BASIC_SPARSE_MATRIX' IN typenames THEN
      RETURN (func\basic_sparse_matrix.val.range);
   END_IF;
   RETURN (?);
END_IF;

IF 'HOMOGENEOUS_LINEAR_FUNCTION' IN typenames THEN
         RETURN
 (one_tuples_of(make_uniform_product_space(factor1(func\homogeneous_linear_
 function.mat.range),
 func\homogeneous_linear_function.mat\explicit_table_function.shape[(3 -
 func\homogeneous_linear_function.sum_index)])));
END_IF;
IF 'GENERAL_LINEAR_FUNCTION' IN typenames THEN
         RETURN
 (one_tuples_of(make_uniform_product_space(factor1(func\general_linear_func
 tion.mat.range),
 func\general_linear_function.mat\explicit_table_function.shape[(3 -
 func\general_linear_function.sum_index)])));
END_IF;
IF 'B_SPLINE_BASIS' IN typenames THEN
         RETURN (one_tuples_of(make_uniform_product_space(the_reals,
 func\b_spline_basis.num_basis)));
END_IF;
IF 'B_SPLINE_FUNCTION' IN typenames THEN
   tspace := factor1(func\b_spline_function.coef.domain);
   m := SIZEOF(func\b_spline_function.basis);
   n := space_dimension(tspace);
   IF m = n THEN
      RETURN (one_tuples_of(the_reals));
   END_IF;
   IF m = n - 1 THEN
       RETURN (one_tuples_of(make_uniform_product_space(the_reals,
    factor_space(tspace, n)\finite_integer_interval.size)));
   END_IF;
   tspace := extract_factors(tspace, m + 1, n);
       RETURN (one_tuples_of(make_function_space(sc_equal, tspace,
 sc_subspace, number_superspace_of(func\b_spline_function.coef.range))));
END_IF;
IF 'RATIONALIZE_FUNCTION' IN typenames THEN
   tspace := factor1(func\rationalize_function.fun.range);
   n := space_dimension(tspace);
      RETURN
 (one_tuples_of(make_uniform_product_space(number_superspace_of(factor1(tsp
 ace)), n - 1)));
END_IF;
IF 'PARTIAL_DERIVATIVE_FUNCTION' IN typenames THEN
     RETURN
 (drop_numeric_constraints(func\partial_derivative_function.derivand.range)
 );
END_IF;
IF 'DEFINITE_INTEGRAL_FUNCTION' IN typenames THEN
```

```
        RETURN
     (drop_numeric_constraints(func\definite_integral_function.integrand.range)
     );
    END_IF;
    IF 'ABSTRACTED_EXPRESSION_FUNCTION' IN typenames THEN
        RETURN
     (one_tuples_of(values_space_of(func\abstracted_expression_function.expr)))
     ;
    END_IF;
    IF 'EXPRESSION_DENOTED_FUNCTION' IN typenames THEN
         RETURN
     (values_space_of(func\expression_denoted_function.expr)\function_space.ran
     ge_argument);
    END_IF;
    IF 'IMPORTED_POINT_FUNCTION' IN typenames THEN
        RETURN (one_tuples_of(make_uniform_product_space(the_reals,
     dimension_of(func\imported_point_function.geometry))));
    END_IF;
    IF 'IMPORTED_CURVE_FUNCTION' IN typenames THEN
        RETURN (one_tuples_of(make_uniform_product_space(the_reals,
     dimension_of(func\imported_curve_function.geometry))));
    END_IF;
    IF 'IMPORTED_SURFACE_FUNCTION' IN typenames THEN
        RETURN (one_tuples_of(make_uniform_product_space(the_reals,
     dimension_of(func\imported_surface_function.geometry))));
    END_IF;
    IF 'IMPORTED_VOLUME_FUNCTION' IN typenames THEN
        RETURN (one_tuples_of(make_uniform_product_space(the_reals,
     dimension_of(func\imported_volume_function.geometry))));
    END_IF;
    IF 'APPLICATION_DEFINED_FUNCTION' IN typenames THEN
       RETURN (func\application_defined_function.explicit_range);
    END_IF;
    RETURN (?);
END_FUNCTION;

FUNCTION dimension_of
   (item : geometric_representation_item ) : dimension_count;
LOCAL
   x : SET OF representation;
   y : representation_context;
   dim : dimension_count;
END_LOCAL;
   IF 'ENGINEERING_PROPERTIES_SCHEMA.CARTESIAN_POINT' IN TYPEOF(item) THEN
      dim := SIZEOF(item\cartesian_point.coordinates);
      RETURN (dim);
   END_IF;
   IF 'ENGINEERING_PROPERTIES_SCHEMA.DIRECTION' IN TYPEOF(item) THEN
      dim := SIZEOF(item\direction.direction_ratios);
      RETURN (dim);
   END_IF;
   IF 'ENGINEERING_PROPERTIES_SCHEMA.VECTOR' IN TYPEOF(item) THEN
      dim := SIZEOF(item\vector.orientation\direction.direction_ratios);
      RETURN (dim);
   END_IF;
   x := using_representations(item);
   y := x[1].context_of_items;
   dim := y\geometric_representation_context.coordinate_space_dimension;
   RETURN (dim);
```

```
END_FUNCTION;

FUNCTION dimensions_for_si_unit
   (n : si_unit_name ) : dimensional_exponents;
   CASE n OF
      metre :
             RETURN (dimensional_exponents(1.00000, 0.00000, 0.00000, 0.00000,
       0.00000, 0.00000, 0.00000));
      gram :
            RETURN (dimensional_exponents(0.00000, 1.00000, 0.00000, 0.00000,
       0.00000, 0.00000, 0.00000));
      second :
             RETURN (dimensional_exponents(0.00000, 0.00000, 1.00000, 0.00000,
       0.00000, 0.00000, 0.00000));
      ampere :
              RETURN (dimensional_exponents(0.00000, 0.00000, 0.00000, 1.00000,
       0.00000, 0.00000, 0.00000));
      kelvin :
              RETURN (dimensional_exponents(0.00000, 0.00000, 0.00000, 0.00000,
       1.00000, 0.00000, 0.00000));
      mole :
            RETURN (dimensional_exponents(0.00000, 0.00000, 0.00000, 0.00000,
       0.00000, 1.00000, 0.00000));
      candela :
               RETURN (dimensional_exponents(0.00000, 0.00000, 0.00000,
       0.00000, 0.00000, 0.00000, 1.00000));
      radian :
              RETURN (dimensional_exponents(0.00000, 0.00000, 0.00000, 0.00000,
       0.00000, 0.00000, 0.00000));
      steradian :
                 RETURN (dimensional_exponents(0.00000, 0.00000, 0.00000,
       0.00000, 0.00000, 0.00000, 0.00000));
      hertz :
             RETURN (dimensional_exponents(0.00000, 0.00000, -1.00000, 0.00000,
       0.00000, 0.00000, 0.00000));
      newton :
              RETURN (dimensional_exponents(1.00000, 1.00000, -2.00000,
       0.00000, 0.00000, 0.00000, 0.00000));
      pascal :
              RETURN (dimensional_exponents(-1.00000, 1.00000, -2.00000,
       0.00000, 0.00000, 0.00000, 0.00000));
      joule :
             RETURN (dimensional_exponents(2.00000, 1.00000, -2.00000, 0.00000,
       0.00000, 0.00000, 0.00000));
      watt :
            RETURN (dimensional_exponents(2.00000, 1.00000, -3.00000, 0.00000,
       0.00000, 0.00000, 0.00000));
      coulomb :
               RETURN (dimensional_exponents(0.00000, 0.00000, 1.00000,
       1.00000, 0.00000, 0.00000, 0.00000));
      volt :
            RETURN (dimensional_exponents(2.00000, 1.00000, -3.00000, -1.00000,
       0.00000, 0.00000, 0.00000));
      farad :
             RETURN (dimensional_exponents(-2.00000, -1.00000, 4.00000,
       1.00000, 0.00000, 0.00000, 0.00000));
      ohm :
           RETURN (dimensional_exponents(2.00000, 1.00000, -3.00000, -2.00000,
       0.00000, 0.00000, 0.00000));
```

```
      siemens :
               RETURN (dimensional_exponents(-2.00000, -1.00000, 3.00000,
       2.00000, 0.00000, 0.00000, 0.00000));
      weber :
             RETURN (dimensional_exponents(2.00000, 1.00000, -2.00000, -
       1.00000, 0.00000, 0.00000, 0.00000));
      tesla :
             RETURN (dimensional_exponents(0.00000, 1.00000, -2.00000, -
       1.00000, 0.00000, 0.00000, 0.00000));
      henry :
             RETURN (dimensional_exponents(2.00000, 1.00000, -2.00000, -
       2.00000, 0.00000, 0.00000, 0.00000));
      degree_Celsius :
             RETURN (dimensional_exponents(0.00000, 0.00000, 0.00000, 0.00000,
       1.00000, 0.00000, 0.00000));
      lumen :
             RETURN (dimensional_exponents(0.00000, 0.00000, 0.00000, 0.00000,
       0.00000, 0.00000, 1.00000));
      lux :
           RETURN (dimensional_exponents(-2.00000, 0.00000, 0.00000, 0.00000,
       0.00000, 0.00000, 1.00000));
      becquerel :
           RETURN (dimensional_exponents(0.00000, 0.00000, -1.00000, 0.00000,
       0.00000, 0.00000, 0.00000));
      gray :
            RETURN (dimensional_exponents(2.00000, 0.00000, -2.00000, 0.00000,
       0.00000, 0.00000, 0.00000));
      sievert :
             RETURN (dimensional_exponents(2.00000, 0.00000, -2.00000, 0.00000,
       0.00000, 0.00000, 0.00000));
   OTHERWISE :
         RETURN (?);
   END_CASE;
END_FUNCTION;

FUNCTION domain_from
   (ref : maths_space_or_function ) : tuple_space;
LOCAL
   typenames : SET OF STRING := stripped_typeof(ref);
   func : maths_function;
END_LOCAL;
   IF NOT EXISTS(ref) THEN
      RETURN (?);
   END_IF;
   IF 'TUPLE_SPACE' IN typenames THEN
      RETURN (ref);
   END_IF;
   IF 'MATHS_SPACE' IN typenames THEN
      RETURN (one_tuples_of(ref));
   END_IF;
   func := ref;
   IF 'CONSTANT_FUNCTION' IN typenames THEN
      RETURN (domain_from(func\constant_function.source_of_domain));
   END_IF;
   IF 'SELECTOR_FUNCTION' IN typenames THEN
      RETURN (domain_from(func\selector_function.source_of_domain));
   END_IF;
   IF 'PARALLEL_COMPOSED_FUNCTION' IN typenames THEN
      RETURN (domain_from(func\parallel_composed_function.source_of_domain));
```

```
      END_IF;
      RETURN (func.domain);
   END_FUNCTION;

   FUNCTION dot_count
      (str : STRING ) : INTEGER;
   LOCAL
      n : INTEGER := 0;
   END_LOCAL;

      REPEAT i := 1 TO LENGTH(str);
         IF str[i] = '.' THEN
            n := n + 1;
         END_IF;
      END_REPEAT;
      RETURN (n);
   END_FUNCTION;

   FUNCTION dot_product
      (arg1 : direction;
       arg2 : direction ) : REAL;
   LOCAL
      scalar : REAL;
      vec1 : direction;
      vec2 : direction;
      ndim : INTEGER;
   END_LOCAL;
      IF NOT EXISTS(arg1) OR NOT EXISTS(arg2) THEN
         scalar := ?;
      ELSE
         IF arg1.dim <> arg2.dim THEN
            scalar := ?;
         ELSE
            BEGIN
               vec1 := normalise(arg1);
               vec2 := normalise(arg2);
               ndim := arg1.dim;
               scalar := 0.00000;
               REPEAT i := 1 TO ndim;
                 scalar := scalar + vec1.direction_ratios[i] *
                 vec2.direction_ratios[i];
               END_REPEAT;
            END;
         END_IF;
      END_IF;
      RETURN (scalar);
   END_FUNCTION;

   FUNCTION dotted_identifiers_syntax
      (str : STRING ) : BOOLEAN;
   LOCAL
      k : positive_integer;
      m : positive_integer;
   END_LOCAL;
      IF NOT EXISTS(str) THEN
         RETURN (FALSE);
      END_IF;
      k := parse_express_identifier(str, 1);
      IF k = 1 THEN
```

```
      RETURN (FALSE);
   END_IF;
   REPEAT WHILE k <= LENGTH(str);
      IF (str[k] <> '.') OR (k = LENGTH(str)) THEN
         RETURN (FALSE);
      END_IF;
      m := parse_express_identifier(str, k + 1);
      IF m = k + 1 THEN
         RETURN (FALSE);
      END_IF;
      k := m;
   END_REPEAT;
   RETURN (TRUE);
END_FUNCTION;

FUNCTION drop_numeric_constraints
   (spc : maths_space ) : maths_space;
LOCAL
   typenames : SET OF STRING := stripped_typeof(spc);
   tspc : listed_product_space;
   factors : LIST OF maths_space := [];
   xspc : extended_tuple_space;
END_LOCAL;
   IF 'UNIFORM_PRODUCT_SPACE' IN typenames THEN
            RETURN
    (make_uniform_product_space(drop_numeric_constraints(spc\uniform_product_s
    pace.base), spc\uniform_product_space.exponent));
   END_IF;
   IF 'LISTED_PRODUCT_SPACE' IN typenames THEN
      tspc := spc;
      REPEAT i := 1 TO SIZEOF(tspc.factors);
         INSERT( factors, drop_numeric_constraints(tspc.factors[i]), i - 1 );
      END_REPEAT;
      RETURN (make_listed_product_space(factors));
   END_IF;
   IF 'EXTENDED_TUPLE_SPACE' IN typenames THEN
      xspc := spc;
            RETURN
    (make_extended_tuple_space(drop_numeric_constraints(xspc.base),
    drop_numeric_constraints(xspc.extender)));
   END_IF;
   IF subspace_of_es(spc, es_numbers) THEN
      RETURN (number_superspace_of(spc));
   END_IF;
   RETURN (spc);
END_FUNCTION;

FUNCTION enclose_cregion_in_pregion
   (crgn : cartesian_complex_number_region;
    centre : complex_number_literal ) : polar_complex_number_region;
   FUNCTION angle
      (a : REAL ) : REAL;
      REPEAT WHILE a > 3.14159;
         a := a - 2.00000 * 3.14159;
      END_REPEAT;
      REPEAT WHILE a <= -3.14159;
         a := a + 2.00000 * 3.14159;
      END_REPEAT;
      RETURN (a);
```

```
    END_FUNCTION;
    FUNCTION strictly_in
       (z : REAL;
        zitv : real_interval ) : LOGICAL;
             RETURN ((NOT min_exists(zitv) OR (z > real_min(zitv))) AND (NOT
     max_exists(zitv) OR (z < real_max(zitv))));
    END_FUNCTION;
    PROCEDURE angle_minmax
       (ab : REAL;
        a : REAL;
        a_in : BOOLEAN;
        VAR amin : REAL;
        VAR amax : REAL;
        VAR amin_in : BOOLEAN;
        VAR amax_in : BOOLEAN );
       a := angle(a - ab);
       IF amin = a THEN
          amin_in := amin_in OR a_in;
       END_IF;
       IF amin > a THEN
          amin := a;
          amin_in := a_in;
       END_IF;
       IF amax = a THEN
          amax_in := amax_in OR a_in;
       END_IF;
       IF amax < a THEN
          amax := a;
          amax_in := a_in;
       END_IF;
    END_PROCEDURE;
    PROCEDURE range_max
       (r : REAL;
        incl : BOOLEAN;
        VAR rmax : REAL;
        VAR rmax_in : BOOLEAN );
       IF rmax = r THEN
          rmax_in := rmax_in OR incl;
       END_IF;
       IF rmax < r THEN
          rmax := r;
          rmax_in := incl;
       END_IF;
    END_PROCEDURE;
    PROCEDURE range_min
       (r : REAL;
        incl : BOOLEAN;
        VAR rmin : REAL;
        VAR rmin_in : BOOLEAN );
       IF rmin = r THEN
          rmin_in := rmin_in OR incl;
       END_IF;
       IF (rmin < 0.00000) OR (rmin > r) THEN
          rmin := r;
          rmin_in := incl;
       END_IF;
    END_PROCEDURE;
 LOCAL
    xitv : real_interval;
```

```
      yitv : real_interval;
      is_xmin : BOOLEAN;
      is_xmax : BOOLEAN;
      is_ymin : BOOLEAN;
      is_ymax : BOOLEAN;
      xmin : REAL := 0.00000;
      xmax : REAL := 0.00000;
      ymin : REAL := 0.00000;
      ymax : REAL := 0.00000;
      xc : REAL := 0.00000;
      yc : REAL := 0.00000;
      xmin_in : BOOLEAN := FALSE;
      xmax_in : BOOLEAN := FALSE;
      ymin_in : BOOLEAN := FALSE;
      ymax_in : BOOLEAN := FALSE;
      rmin : REAL := -1.00000;
      rmax : REAL := -1.00000;
      amin : REAL := 4.00000;
      amax : REAL := -4.00000;
      rmax_exists : BOOLEAN := TRUE;
      outside : BOOLEAN := TRUE;
      rmin_in : BOOLEAN := FALSE;
      rmax_in : BOOLEAN := FALSE;
      amin_in : BOOLEAN := FALSE;
      amax_in : BOOLEAN := FALSE;
      ab : REAL := 0.00000;
      a : REAL := 0.00000;
      r : REAL := 0.00000;
      incl : BOOLEAN;
      ritv : real_interval;
      aitv : finite_real_interval;
      minclo : open_closed := open;
      maxclo : open_closed := open;
   END_LOCAL;
      IF NOT EXISTS(crgn) OR NOT EXISTS(centre) THEN
         RETURN (?);
      END_IF;
      xitv := crgn.real_constraint;
      yitv := crgn.imag_constraint;
      xc := centre.real_part;
      yc := centre.imag_part;
      is_xmin := min_exists(xitv);
      is_xmax := max_exists(xitv);
      is_ymin := min_exists(yitv);
      is_ymax := max_exists(yitv);
      IF is_xmin THEN
         xmin := real_min(xitv);
         xmin_in := min_included(xitv);
      END_IF;
      IF is_xmax THEN
         xmax := real_max(xitv);
         xmax_in := max_included(xitv);
      END_IF;
      IF is_ymin THEN
         ymin := real_min(yitv);
         ymin_in := min_included(yitv);
      END_IF;
      IF is_ymax THEN
         ymax := real_max(yitv);
```

```
      ymax_in := max_included(yitv);
   END_IF;
   rmax_exists := ((is_xmin AND is_xmax) AND is_ymin) AND is_ymax;
   IF is_xmin AND (xc <= xmin) THEN
      ab := 0.00000;
   ELSE
      IF is_ymin AND (yc <= ymin) THEN
         ab := 0.500000 * 3.14159;
      ELSE
         IF is_ymax AND (yc >= ymax) THEN
            ab := -0.500000 * 3.14159;
         ELSE
            IF is_xmax AND (xc >= xmax) THEN
               ab := 3.14159;
            ELSE
               outside := FALSE;
            END_IF;
         END_IF;
      END_IF;
   END_IF;
   IF NOT outside AND NOT rmax_exists THEN
      RETURN (?);
   END_IF;
   IF (is_xmin AND (xc <= xmin)) AND strictly_in(yc, yitv) THEN
      rmin := xmin - xc;
      rmin_in := xmin_in;
   ELSE
      IF (is_ymin AND (yc <= ymin)) AND strictly_in(xc, xitv) THEN
         rmin := ymin - yc;
         rmin_in := ymin_in;
      ELSE
         IF (is_ymax AND (yc >= ymax)) AND strictly_in(xc, xitv) THEN
            rmin := yc - ymax;
            rmin_in := ymax_in;
         ELSE
            IF (is_xmax AND (xc >= xmax)) AND strictly_in(yc, yitv) THEN
               rmin := xc - xmax;
               rmin_in := xmax_in;
            END_IF;
         END_IF;
      END_IF;
   END_IF;
   IF is_xmin THEN
      IF is_ymin THEN
         r := SQRT((xmin - xc) ** 2 + (ymin - yc) ** 2);
         incl := xmin_in AND ymin_in;
         IF rmax_exists THEN
            range_max( r, incl, rmax, rmax_in );
         END_IF;
         IF outside THEN
            IF r > 0.00000 THEN
               range_min( r, incl, rmin, rmin_in );
               a := angle(atan2(ymin - yc, xmin - xc) - ab);
               IF xc = xmin THEN
                  incl := xmin_in;
               END_IF;
               IF yc = ymin THEN
                  incl := ymin_in;
               END_IF;
```

```
            angle_minmax( ab, a, incl, amin, amax, amin_in, amax_in );
         ELSE
            rmin := 0.00000;
            rmin_in := xmin_in AND ymin_in;
            amin := angle(0.00000 - ab);
            amin_in := ymin_in;
            amax := angle(0.500000 * 3.14159 - ab);
            amax_in := xmin_in;
         END_IF;
      END_IF;
   ELSE
      IF xc <= xmin THEN
                     angle_minmax( ab, -0.500000 * 3.14159, (xc = xmin)
       AND xmin_in, amin, amax, amin_in, amax_in );
      END_IF;
   END_IF;
   IF NOT is_ymax AND (xc <= xmin) THEN
                angle_minmax( ab, 0.500000 * 3.14159, (xc = xmin) AND
    xmin_in, amin, amax, amin_in, amax_in );
   END_IF;
END_IF;
IF is_ymin THEN
   IF is_xmax THEN
      r := SQRT((xmax - xc) ** 2 + (ymin - yc) ** 2);
      incl := xmax_in AND ymin_in;
      IF rmax_exists THEN
         range_max( r, incl, rmax, rmax_in );
      END_IF;
      IF outside THEN
         IF r > 0.00000 THEN
            range_min( r, incl, rmin, rmin_in );
            a := angle(atan2(ymin - yc, xmax - xc) - ab);
            IF xc = xmax THEN
               incl := xmax_in;
            END_IF;
            IF yc = ymin THEN
               incl := ymin_in;
            END_IF;
            angle_minmax( ab, a, incl, amin, amax, amin_in, amax_in );
         ELSE
            rmin := 0.00000;
            rmin_in := xmax_in AND ymin_in;
            amin := angle(0.500000 * 3.14159 - ab);
            amin_in := ymin_in;
            amax := angle(3.14159 - ab);
            amax_in := xmax_in;
         END_IF;
      END_IF;
   ELSE
      IF yc <= ymin THEN
                     angle_minmax( ab, 0.00000, (yc = ymin) AND ymin_in,
       amin, amax, amin_in, amax_in );
      END_IF;
   END_IF;
   IF NOT is_xmin AND (yc <= ymin) THEN
                angle_minmax( ab, 3.14159, (yc = ymin) AND ymin_in, amin,
    amax, amin_in, amax_in );
   END_IF;
END_IF;
```

```
IF is_xmax THEN
   IF is_ymax THEN
      r := SQRT((xmax - xc) ** 2 + (ymax - yc) ** 2);
      incl := xmax_in AND ymax_in;
      IF rmax_exists THEN
         range_max( r, incl, rmax, rmax_in );
      END_IF;
      IF outside THEN
         IF r > 0.00000 THEN
            range_min( r, incl, rmin, rmin_in );
            a := angle(atan2(ymax - yc, xmax - xc) - ab);
            IF xc = xmax THEN
               incl := xmax_in;
            END_IF;
            IF yc = ymax THEN
               incl := ymax_in;
            END_IF;
            angle_minmax( ab, a, incl, amin, amax, amin_in, amax_in );
         ELSE
            rmin := 0.00000;
            rmin_in := xmax_in AND ymax_in;
            amin := angle(-3.14159 - ab);
            amin_in := ymax_in;
            amax := angle(-0.500000 * 3.14159 - ab);
            amax_in := xmax_in;
         END_IF;
      END_IF;
   ELSE
      IF xc >= xmax THEN
                     angle_minmax( ab, 0.500000 * 3.14159, (xc = xmax)
       AND xmax_in, amin, amax, amin_in, amax_in );
      END_IF;
   END_IF;
   IF NOT is_ymin AND (xc >= xmax) THEN
                angle_minmax( ab, -0.500000 * 3.14159, (xc = xmax) AND
    xmax_in, amin, amax, amin_in, amax_in );
   END_IF;
END_IF;
IF is_ymax THEN
   IF is_xmin THEN
      r := SQRT((xmin - xc) ** 2 + (ymax - yc) ** 2);
      incl := xmin_in AND ymax_in;
      IF rmax_exists THEN
         range_max( r, incl, rmax, rmax_in );
      END_IF;
      IF outside THEN
         IF r > 0.00000 THEN
            range_min( r, incl, rmin, rmin_in );
            a := angle(atan2(ymax - yc, xmin - xc) - ab);
            IF xc = xmin THEN
               incl := xmin_in;
            END_IF;
            IF yc = ymax THEN
               incl := ymax_in;
            END_IF;
            angle_minmax( ab, a, incl, amin, amax, amin_in, amax_in );
         ELSE
            rmin := 0.00000;
            rmin_in := xmin_in AND ymax_in;
```

```
                amin := angle(0.500000 * 3.14159 - ab);
                amin_in := ymax_in;
                amax := angle(3.14159 - ab);
                amax_in := xmin_in;
             END_IF;
          END_IF;
       ELSE
          IF yc >= ymax THEN
                          angle_minmax( ab, 3.14159, (yc = ymax) AND ymax_in,
           amin, amax, amin_in, amax_in );
          END_IF;
       END_IF;
       IF NOT is_xmax AND (yc >= ymax) THEN
                   angle_minmax( ab, 0.00000, (yc = ymax) AND ymax_in, amin,
        amax, amin_in, amax_in );
       END_IF;
    END_IF;
    IF outside THEN
       amin := angle(amin + ab);
       IF amin = 3.14159 THEN
          amin := -3.14159;
       END_IF;
       amax := angle(amax + ab);
       IF amax <= amin THEN
          amax := amax + 2.00000 * 3.14159;
       END_IF;
    ELSE
       amin := -3.14159;
       amin_in := FALSE;
       amax := 3.14159;
       amax_in := FALSE;
    END_IF;
    IF amin_in THEN
       minclo := closed;
    END_IF;
    IF amax_in THEN
       maxclo := closed;
    END_IF;
    aitv := make_finite_real_interval(amin, minclo, amax, maxclo);
    minclo := open;
    IF rmin_in THEN
       minclo := closed;
    END_IF;
    IF rmax_exists THEN
       maxclo := open;
       IF rmax_in THEN
          maxclo := closed;
       END_IF;
       ritv := make_finite_real_interval(rmin, minclo, rmax, maxclo);
    ELSE
       ritv := make_real_interval_from_min(rmin, minclo);
    END_IF;
    RETURN (make_polar_complex_number_region(centre, ritv, aitv));
END_FUNCTION;

FUNCTION enclose_pregion_in_cregion
   (prgn : polar_complex_number_region ) : cartesian_complex_number_region;
   PROCEDURE nearest_good_direction
      (acart : REAL;
```

```
        aitv : finite_real_interval;
        VAR a : REAL;
        VAR a_in : BOOLEAN );
      a := acart;
      a_in := TRUE;
      IF a < aitv.min THEN
         IF a + 2.00000 * 3.14159 < aitv.max THEN
            RETURN;
         END_IF;
         IF a + 2.00000 * 3.14159 = aitv.max THEN
            a_in := max_included(aitv);
            RETURN;
         END_IF;
      ELSE
         IF a = aitv.min THEN
            a_in := min_included(aitv);
            RETURN;
         ELSE
            IF a < aitv.max THEN
               RETURN;
            ELSE
               IF a = aitv.max THEN
                  a_in := max_included(aitv);
                  RETURN;
               END_IF;
            END_IF;
         END_IF;
      END_IF;
      IF COS(acart - aitv.max) >= COS(acart - aitv.min) THEN
         a := aitv.max;
         a_in := max_included(aitv);
      ELSE
         a := aitv.min;
         a_in := min_included(aitv);
      END_IF;
   END_PROCEDURE;
LOCAL
   xc : REAL := 0.00000;
   yc : REAL := 0.00000;
   xmin : REAL := 0.00000;
   xmax : REAL := 0.00000;
   ymin : REAL := 0.00000;
   ymax : REAL := 0.00000;
   ritv : real_interval;
   xitv : real_interval;
   yitv : real_interval;
   aitv : finite_real_interval;
   xmin_exists : BOOLEAN;
   xmax_exists : BOOLEAN;
   ymin_exists : BOOLEAN;
   ymax_exists : BOOLEAN;
   xmin_in : BOOLEAN := FALSE;
   xmax_in : BOOLEAN := FALSE;
   ymin_in : BOOLEAN := FALSE;
   ymax_in : BOOLEAN := FALSE;
   a : REAL := 0.00000;
   r : REAL := 0.00000;
   a_in : BOOLEAN := FALSE;
   min_clo : open_closed := open;
```

```
   max_clo : open_closed := open;
END_LOCAL;
   IF NOT EXISTS(prgn) THEN
      RETURN (?);
   END_IF;
   xc := prgn.centre.real_part;
   yc := prgn.centre.imag_part;
   ritv := prgn.distance_constraint;
   aitv := prgn.direction_constraint;
   nearest_good_direction( 3.14159, aitv, a, a_in );
   IF COS(a) >= 0.00000 THEN
      xmin_exists := TRUE;
      xmin := xc + real_min(ritv) * COS(a);
      xmin_in := a_in AND (min_included(ritv) OR (COS(a) = 0.00000));
   ELSE
      IF max_exists(ritv) THEN
         xmin_exists := TRUE;
         xmin := xc + real_max(ritv) * COS(a);
         xmin_in := a_in AND max_included(ritv);
      ELSE
         xmin_exists := FALSE;
      END_IF;
   END_IF;
   nearest_good_direction( 0.00000, aitv, a, a_in );
   IF COS(a) <= 0.00000 THEN
      xmax_exists := TRUE;
      xmax := xc + real_min(ritv) * COS(a);
      xmax_in := a_in AND (min_included(ritv) OR (COS(a) = 0.00000));
   ELSE
      IF max_exists(ritv) THEN
         xmax_exists := TRUE;
         xmax := xc + real_max(ritv) * COS(a);
         xmax_in := a_in AND max_included(ritv);
      ELSE
         xmax_exists := FALSE;
      END_IF;
   END_IF;
   nearest_good_direction( -0.500000 * 3.14159, aitv, a, a_in );
   IF SIN(a) >= 0.00000 THEN
      ymin_exists := TRUE;
      ymin := yc + real_min(ritv) * SIN(a);
      ymin_in := a_in AND (min_included(ritv) OR (SIN(a) = 0.00000));
   ELSE
      IF max_exists(ritv) THEN
         ymin_exists := TRUE;
         ymin := yc + real_max(ritv) * SIN(a);
         ymin_in := a_in AND max_included(ritv);
      ELSE
         ymin_exists := FALSE;
      END_IF;
   END_IF;
   nearest_good_direction( 0.500000 * 3.14159, aitv, a, a_in );
   IF SIN(a) <= 0.00000 THEN
      ymax_exists := TRUE;
      ymax := yc + real_min(ritv) * SIN(a);
      ymax_in := a_in AND (min_included(ritv) OR (SIN(a) = 0.00000));
   ELSE
      IF max_exists(ritv) THEN
         ymax_exists := TRUE;
```

```
      ymax := yc + real_max(ritv) * SIN(a);
      ymax_in := a_in AND max_included(ritv);
   ELSE
      ymax_exists := FALSE;
   END_IF;
END_IF;
IF NOT (((xmin_exists OR xmax_exists) OR ymin_exists) OR ymax_exists) THEN
   RETURN (?);
END_IF;
IF xmin_exists THEN
   IF xmin_in THEN
      min_clo := closed;
   ELSE
      min_clo := open;
   END_IF;
   IF xmax_exists THEN
      IF xmax_in THEN
         max_clo := closed;
      ELSE
         max_clo := open;
      END_IF;
      xitv := make_finite_real_interval(xmin, min_clo, xmax, max_clo);
   ELSE
      xitv := make_real_interval_from_min(xmin, min_clo);
   END_IF;
ELSE
   IF xmax_exists THEN
      IF xmax_in THEN
         max_clo := closed;
      ELSE
         max_clo := open;
      END_IF;
      xitv := make_real_interval_to_max(xmax, max_clo);
   ELSE
      xitv := the_reals;
   END_IF;
END_IF;
IF ymin_exists THEN
   IF ymin_in THEN
      min_clo := closed;
   ELSE
      min_clo := open;
   END_IF;
   IF ymax_exists THEN
      IF ymax_in THEN
         max_clo := closed;

      ELSE
         max_clo := open;
      END_IF;
      yitv := make_finite_real_interval(ymin, min_clo, ymax, max_clo);
   ELSE
      yitv := make_real_interval_from_min(ymin, min_clo);
   END_IF;
ELSE
   IF ymax_exists THEN
      IF ymax_in THEN
         max_clo := closed;
      ELSE
```

```
            max_clo := open;
         END_IF;
         yitv := make_real_interval_to_max(ymax, max_clo);
      ELSE
         yitv := the_reals;
      END_IF;
   END_IF;
   RETURN (make_cartesian_complex_number_region(xitv, yitv));
END_FUNCTION;

FUNCTION enclose_pregion_in_pregion
   (prgn : polar_complex_number_region;
    centre : complex_number_literal ) : polar_complex_number_region;
   FUNCTION angle
      (a : REAL ) : REAL;
      REPEAT WHILE a > 3.14159;
         a := a - 2.00000 * 3.14159;
      END_REPEAT;
      REPEAT WHILE a <= -3.14159;
         a := a + 2.00000 * 3.14159;
      END_REPEAT;
      RETURN (a);
   END_FUNCTION;
   PROCEDURE angle_range
      (VAR amin : REAL;
       VAR amax : REAL );
      amin := angle(amin);
      IF amin = 3.14159 THEN
         amin := -3.14159;
      END_IF;
      amax := angle(amax);
      IF amax <= amin THEN
         amax := amax + 2.00000 * 3.14159;
      END_IF;
   END_PROCEDURE;
   FUNCTION strictly_in
      (a : REAL;
       aitv : finite_real_interval ) : LOGICAL;
      a := angle(a);
            RETURN ((aitv.min < a) AND (a < aitv.max) OR (aitv.min < a +
    2.00000 * 3.14159) AND (a + 2.00000 * 3.14159 < aitv.max));
   END_FUNCTION;
   PROCEDURE find_aminmax
      (ab : REAL;
       a0 : REAL;
       a1 : REAL;
       a2 : REAL;
       a3 : REAL;
       in0 : BOOLEAN;
       in1 : BOOLEAN;
       in2 : BOOLEAN;
       in3 : BOOLEAN;
       VAR amin : REAL;
       VAR amax : REAL;
       VAR amin_in : BOOLEAN;
       VAR amax_in : BOOLEAN );
   LOCAL
      a : REAL;
   END_LOCAL;
```

```
      amin := angle(a0 - ab);
      amin_in := in0;
      amax := amin;
      amax_in := in0;
      a := angle(a1 - ab);
      IF a = amin THEN
         amin_in := amin_in OR in1;
      END_IF;
      IF a < amin THEN
         amin := a;
         amin_in := in1;
      END_IF;
      IF a = amax THEN
         amax_in := amax_in OR in1;
      END_IF;
      IF a > amax THEN
         amax := a;
         amax_in := in1;
      END_IF;
      a := angle(a2 - ab);
      IF a = amin THEN
         amin_in := amin_in OR in2;
      END_IF;
      IF a < amin THEN
         amin := a;
         amin_in := in2;
      END_IF;
      IF a = amax THEN
         amax_in := amax_in OR in2;
      END_IF;
      IF a > amax THEN
         amax := a;
         amax_in := in2;
      END_IF;
      a := angle(a3 - ab);
      IF a = amin THEN
         amin_in := amin_in OR in3;
      END_IF;
      IF a < amin THEN
         amin := a;
         amin_in := in3;
      END_IF;
      IF a = amax THEN
         amax_in := amax_in OR in3;
      END_IF;
      IF a > amax THEN
         amax := a;
         amax_in := in3;
      END_IF;
      amin := amin + ab;
      amax := amax + ab;
      angle_range( amin, amax );
   END_PROCEDURE;
LOCAL
   ritp : real_interval;
   ritv : real_interval;
   aitp : finite_real_interval;
   aitv : finite_real_interval;
   xp : REAL := 0.00000;
```

```
    yp : REAL := 0.00000;
    xc : REAL := 0.00000;
    yc : REAL := 0.00000;
    rmax : REAL := 0.00000;
    rmin : REAL := 0.00000;
    amin : REAL := 0.00000;
    amax : REAL := 0.00000;
    rc : REAL := 0.00000;
    acp : REAL := 0.00000;
    apc : REAL := 0.00000;
    rmax_in : BOOLEAN := FALSE;
    rmin_in : BOOLEAN := FALSE;
    amin_in : BOOLEAN := FALSE;
    amax_in : BOOLEAN := FALSE;
    rmxp : REAL := 0.00000;
    rmnp : REAL := 0.00000;
    x : REAL := 0.00000;
    y : REAL := 0.00000;
    r : REAL := 0.00000;
    a : REAL := 0.00000;
    ab : REAL := 0.00000;
    r0 : REAL := 0.00000;
    a0 : REAL := 0.00000;
    r1 : REAL := 0.00000;
    a1 : REAL := 0.00000;
    r2 : REAL := 0.00000;
    a2 : REAL := 0.00000;
    r3 : REAL := 0.00000;
    a3 : REAL := 0.00000;
    in0 : BOOLEAN := FALSE;
    in1 : BOOLEAN := FALSE;
    in2 : BOOLEAN := FALSE;
    in3 : BOOLEAN := FALSE;
    inn : BOOLEAN := FALSE;
    minclo : open_closed := open;
    maxclo : open_closed := open;
  END_LOCAL;
    IF NOT EXISTS(prgn) OR NOT EXISTS(centre) THEN
       RETURN (?);
    END_IF;
    xp := prgn.centre.real_part;
    yp := prgn.centre.imag_part;
    ritp := prgn.distance_constraint;
    aitp := prgn.direction_constraint;
    xc := centre.real_part;
    yc := centre.imag_part;
    IF (xc = xp) AND (yc = yp) THEN
       RETURN (prgn);
    END_IF;
    rc := SQRT((xp - xc) ** 2 + (yp - yc) ** 2);
    acp := atan2(yp - yc, xp - xc);
    apc := atan2(yc - yp, xc - xp);
    rmnp := real_min(ritp);
    IF max_exists(ritp) THEN
       rmxp := real_max(ritp);
       IF aitp.max - aitp.min = 2.00000 * 3.14159 THEN
          inn := NOT max_included(aitp);
          a := angle(aitp.min);
          rmax := rc + rmxp;
```


439

```
rmax_in := max_included(ritp);
IF inn AND (acp = a) THEN
   rmax_in := FALSE;
END_IF;
IF rc > rmxp THEN
   a0 := ASIN(rmxp / rc);
   amin := angle(acp - a0);
   amin_in := max_included(ritp);
   IF amin = 3.14159 THEN
      amin := -3.14159;
   END_IF;
   amax := angle(acp + a0);
   amax_in := amin_in;
   IF amax < amin THEN
      amax := amax + 2.00000 * 3.14159;
   END_IF;
   rmin := rc - rmxp;
   rmin_in := amin_in;
   IF inn THEN
      IF apc = a THEN
         rmin_in := FALSE;
      END_IF;
      IF angle(amin + 0.500000 * 3.14159) = a THEN
         amin_in := FALSE;
      END_IF;
      IF angle(amax - 0.500000 * 3.14159) = a THEN
         amax_in := FALSE;
      END_IF;
   END_IF;
ELSE
   IF rc = rmxp THEN
      amin := angle(acp - 0.500000 * 3.14159);
      amin_in := FALSE;
      IF amin = 3.14159 THEN

         amin := -3.14159;
      END_IF;
      amax := angle(acp + 0.500000 * 3.14159);
      amax_in := FALSE;
      IF amax < amin THEN
         amax := amax + 2.00000 * 3.14159;
      END_IF;
      rmin := 0.00000;
      rmin_in := max_included(ritp);
      IF inn AND (apc = a) THEN
         rmin_in := FALSE;
      END_IF;
   ELSE
      IF rc > rmnp THEN
         IF inn AND (apc = a) THEN
            rmin := 0.00000;
            rmin_in := FALSE;
            amin := aitp.min;
            amin_in := FALSE;
            amax := aitp.max;
            amax_in := FALSE;
         ELSE
            rmin := 0.00000;
            rmin_in := TRUE;
```

```
                  amin := -3.14159;
                  amin_in := FALSE;
                  amax := 3.14159;
                  amax_in := TRUE;
               END_IF;
            ELSE
               rmin := rmnp - rc;
               rmin_in := min_included(ritp);
               amin := -3.14159;
               amin_in := FALSE;
               amax := 3.14159;
               amax_in := TRUE;
               IF inn THEN
                  IF apc = a THEN
                     rmin_in := FALSE;
                     amin := aitp.min;
                     amin_in := FALSE;
                     amax := aitp.max;
                     amax_in := FALSE;
                  ELSE
                     IF acp = a THEN
                        amin := aitp.min;
                        amin_in := FALSE;
                        amax := aitp.max;
                        amax_in := FALSE;
                     END_IF;
                  END_IF;
               END_IF;
            END_IF;
         END_IF;
      END_IF;
   ELSE
      x := xp + rmxp * COS(aitp.min) - xc;
      y := yp + rmxp * SIN(aitp.min) - yc;
      r0 := SQRT(x ** 2 + y ** 2);
      in0 := max_included(ritp) AND min_included(aitp);
      IF r0 <> 0.00000 THEN
         a0 := atan2(y, x);
      END_IF;
      x := xp + rmxp * COS(aitp.max) - xc;
      y := yp + rmxp * SIN(aitp.max) - yc;
      r1 := SQRT(x ** 2 + y ** 2);
      in1 := max_included(ritp) AND max_included(aitp);
      IF r1 <> 0.00000 THEN
         a1 := atan2(y, x);
      END_IF;
      x := xp + rmnp * COS(aitp.max) - xc;
      y := yp + rmnp * SIN(aitp.max) - yc;
      r2 := SQRT(x ** 2 + y ** 2);
      in2 := min_included(ritp) AND max_included(aitp);
      IF r2 <> 0.00000 THEN
         a2 := atan2(y, x);
      ELSE
         a2 := a1;
         in2 := in1;
      END_IF;
      IF r1 = 0.00000 THEN
         a1 := a2;
         in1 := in2;
```

```
END_IF;
x := xp + rmnp * COS(aitp.min) - xc;
y := yp + rmnp * SIN(aitp.min) - yc;
r3 := SQRT(x ** 2 + y ** 2);
in3 := min_included(ritp) AND min_included(aitp);
IF r3 <> 0.00000 THEN
   a3 := atan2(y, x);
ELSE
   a3 := a0;
   in3 := in0;
END_IF;
IF r0 = 0.00000 THEN
   a0 := a3;
   in0 := in3;
END_IF;
IF rmnp = 0.00000 THEN
   in2 := min_included(ritp);
   in3 := in2;
END_IF;
IF (apc = angle(aitp.min)) OR (acp = angle(aitp.min)) THEN
   in0 := min_included(aitp);
   in3 := in0;
ELSE
   IF (apc = angle(aitp.max)) OR (acp = angle(aitp.max)) THEN
      in1 := max_included(aitp);
      in2 := in1;
   END_IF;
END_IF;
IF strictly_in(acp, aitp) THEN
   rmax := rc + rmxp;
   rmax_in := max_included(ritp);
ELSE
   rmax := r0;
   rmax_in := in0;
   IF rmax = r1 THEN
      rmax_in := rmax_in OR in1;
   END_IF;
   IF rmax < r1 THEN
      rmax := r1;
      rmax_in := in1;
   END_IF;
   IF rmax = r2 THEN
      rmax_in := rmax_in OR in2;
   END_IF;
   IF rmax < r2 THEN
      rmax := r2;
      rmax_in := in2;
   END_IF;
   IF rmax = r3 THEN
      rmax_in := rmax_in OR in3;
   END_IF;
   IF rmax < r3 THEN
      rmax := r3;
      rmax_in := in3;
   END_IF;
END_IF;
IF strictly_in(apc, aitp) THEN
   IF rc >= rmxp THEN
      rmin := rc - rmxp;
```

```
      rmin_in := max_included(ritp);
   ELSE
      IF rc <= rmnp THEN
         rmin := rmnp - rc;
         rmin_in := min_included(ritp);
      ELSE
         rmin := 0.00000;
         rmin_in := TRUE;
      END_IF;
   END_IF;
ELSE
   rmin := r0;
   rmin_in := in0;
   a := apc - aitp.min;
   r := rc * COS(a);
   IF (rmnp < r) AND (r < rmxp) THEN
      rmin := rc * SIN(ABS(a));
      rmin_in := min_included(aitp);
   END_IF;
   a := apc - aitp.max;
   r := rc * COS(a);
   IF (rmnp < r) AND (r < rmxp) THEN
      r := rc * SIN(ABS(a));
      inn := max_included(aitp);
      IF r = rmin THEN
         rmin_in := rmin_in OR inn;
      END_IF;
      IF r < rmin THEN
         rmin := r;
         rmin_in := inn;
      END_IF;
   END_IF;
   IF r1 = rmin THEN
      rmin_in := rmin_in OR in1;
   END_IF;
   IF r1 < rmin THEN
      rmin := r1;
      rmin_in := in1;
   END_IF;
   IF r2 = rmin THEN
      rmin_in := rmin_in OR in2;
   END_IF;
   IF r2 < rmin THEN
      rmin := r2;
      rmin_in := in2;
   END_IF;
   IF r3 = rmin THEN
      rmin_in := rmin_in OR in3;
   END_IF;
   IF r3 < rmin THEN
      rmin := r3;
      rmin_in := in3;
   END_IF;
END_IF;
IF rc >= rmxp THEN
   ab := acp;
  find_aminmax( ab, a0, a1, a2, a3, in0, in1, in2, in3, amin, amax,
 amin_in, amax_in );
   a := ACOS(rmxp / rc);
```

```
            IF strictly_in(apc - a, aitp) THEN
               amin := ab - ASIN(rmxp / rc);
               amin_in := max_included(ritp);
            END_IF;
            IF strictly_in(apc + a, aitp) THEN
               amax := ab + ASIN(rmxp / rc);
               amax_in := max_included(ritp);
            END_IF;
            angle_range( amin, amax );
         ELSE
            IF rc > rmnp THEN
               ab := angle(0.500000 * (aitp.min + aitp.max));
               find_aminmax( ab, a0, a1, a2, a3, in0, in1, in2, in3, amin,
              amax, amin_in, amax_in );
            ELSE
               ab := angle(0.500000 * (aitp.min + aitp.max));
               a0 := angle(a0 - ab);
               a1 := angle(a1 - ab);
               a2 := angle(a2 - ab);
               a3 := angle(a3 - ab);
               IF a3 > a2 THEN
                  a2 := a2 + 2.00000 * 3.14159;
               END_IF;
               IF a0 > a1 THEN
                  a0 := a0 + 2.00000 * 3.14159;
               END_IF;
               IF a3 < a0 THEN
                  amin := a3;
                  amin_in := in3;
               ELSE
                  amin := a0;
                  amin_in := in0;
               END_IF;
               IF a2 > a1 THEN
                  amax := a2;
                  amax_in := in2;
               ELSE
                  amax := a1;
                  amax_in := in1;
               END_IF;
                 IF (amax - amin > 2.00000 * 3.14159) OR (amax - amin =
                 2.00000 * 3.14159) AND (amin_in OR amax_in) THEN
                  amin := -3.14159;
                  amin_in := FALSE;
                  amax := 3.14159;
                  amax_in := TRUE;
               ELSE
                  amin := amin + ab;
                  amax := amax + ab;
                  angle_range( amin, amax );
               END_IF;
            END_IF;
         END_IF;
      END_IF;
      IF rmin_in THEN
         minclo := closed;
      END_IF;
      IF rmax_in THEN
         maxclo := closed;
```

```
      END_IF;
      ritv := make_finite_real_interval(rmin, minclo, rmax, maxclo);
   ELSE
      IF (rc > rmnp) AND strictly_in(apc, aitp) THEN
         RETURN (?);
      END_IF;
      IF aitp.max - aitp.min = 2.00000 * 3.14159 THEN
         a := angle(aitp.min);
         IF rc > rmnp THEN
            IF max_included(aitp) THEN
               RETURN (?);
            END_IF;
            rmin := 0.00000;
            rmin_in := FALSE;
            amin := aitp.min;
            amin_in := FALSE;
            amax := aitp.max;
            amax_in := FALSE;
         ELSE
            rmin := rmnp - rc;
            rmin_in := min_included(ritp);
            amin := -3.14159;
            amin_in := FALSE;
            amax := 3.14159;
            amax_in := TRUE;
            IF NOT max_included(aitp) THEN
               IF apc = a THEN
                  rmin_in := FALSE;
                  amin := aitp.min;
                  amin_in := FALSE;
                  amax := aitp.max;
                  amax_in := FALSE;
               ELSE
                  IF acp = a THEN
                     amin := aitp.min;
                     amin_in := FALSE;
                     amax := aitp.max;
                     amax_in := FALSE;
                  END_IF;
               END_IF;
            END_IF;
         END_IF;
      ELSE
         a0 := angle(aitp.min);
         in0 := FALSE;
         a1 := angle(aitp.max);
         in1 := FALSE;
         x := xp + rmnp * COS(aitp.max) - xc;
         y := yp + rmnp * SIN(aitp.max) - yc;
         r2 := SQRT(x ** 2 + y ** 2);
         in2 := min_included(ritp) AND max_included(aitp);
         IF r2 <> 0.00000 THEN
            a2 := atan2(y, x);
         ELSE
            a2 := a1;
            in2 := in1;
         END_IF;
         x := xp + rmnp * COS(aitp.min) - xc;
         y := yp + rmnp * SIN(aitp.min) - yc;
```

```
r3 := SQRT(x ** 2 + y ** 2);
in3 := min_included(ritp) AND min_included(aitp);
IF r3 <> 0.00000 THEN
   a3 := atan2(y, x);
ELSE
   a3 := a0;
   in3 := in0;
END_IF;
IF rmnp = 0.00000 THEN
   in2 := min_included(ritp);
   in3 := in2;
END_IF;
IF (apc = angle(aitp.min)) OR (acp = angle(aitp.min)) THEN
   in0 := min_included(aitp);
   in3 := in0;
ELSE
   IF (apc = angle(aitp.max)) OR (acp = angle(aitp.max)) THEN
      in1 := max_included(aitp);

      in2 := in1;
   END_IF;
END_IF;
IF strictly_in(apc, aitp) THEN
   rmin := rmnp - rc;
   rmin_in := min_included(ritp);
ELSE
   rmin := r2;
   rmin_in := in2;
   a := apc - aitp.min;
   r := rc * COS(a);
   IF rmnp < r THEN
      rmin := rc * SIN(ABS(a));
      rmin_in := min_included(aitp);
   END_IF;
   a := apc - aitp.max;
   r := rc * COS(a);
   IF rmnp < r THEN
      r := rc * SIN(ABS(a));
      inn := max_included(aitp);
      IF r = rmin THEN
         rmin_in := rmin_in OR inn;
      END_IF;
      IF r < rmin THEN
         rmin := r;
         rmin_in := inn;
      END_IF;
   END_IF;
   IF r3 = rmin THEN
      rmin_in := rmin_in OR in3;
   END_IF;
   IF r3 < rmin THEN
      rmin := r3;
      rmin_in := in3;
   END_IF;
END_IF;
ab := angle(0.500000 * (aitp.min + aitp.max));
IF rc > rmnp THEN
 find_aminmax( ab, a0, a1, a2, a3, in0, in1, in2, in3, amin, amax,
 amin_in, amax_in );
```

```
            ELSE
               a0 := angle(a0 - ab);
               a1 := angle(a1 - ab);
               a2 := angle(a2 - ab);
               a3 := angle(a3 - ab);
               IF a3 > a2 THEN
                  a2 := a2 + 2.00000 * 3.14159;
               END_IF;
               IF a0 > a1 THEN
                  a0 := a0 + 2.00000 * 3.14159;
               END_IF;
               IF a3 < a0 THEN
                  amin := a3;
                  amin_in := in3;
               ELSE
                  amin := a0;
                  amin_in := in0;
               END_IF;
               IF a2 > a1 THEN
                  amax := a2;
                  amax_in := in2;
               ELSE
                  amax := a1;
                  amax_in := in1;
               END_IF;
                 IF (amax - amin > 2.00000 * 3.14159) OR (amax - amin = 2.00000 *
                 3.14159) AND (amin_in OR amax_in) THEN
                   amin := -3.14159;
                   amin_in := FALSE;
                   amax := 3.14159;
                   amax_in := TRUE;
                   IF (rmin = 0.00000) AND rmin_in THEN
                      RETURN (?);
                   END_IF;
               ELSE
                  amin := amin + ab;
                  amax := amax + ab;
                  angle_range( amin, amax );
               END_IF;
            END_IF;
         END_IF;
         IF rmin_in THEN
            minclo := closed;
         END_IF;
         ritv := make_real_interval_from_min(rmin, minclo);
      END_IF;
      minclo := open;
      maxclo := open;
      IF amin_in THEN
         minclo := closed;
      END_IF;
      IF amax_in THEN
         maxclo := closed;
      END_IF;
      aitv := make_finite_real_interval(amin, minclo, amax, maxclo);
      RETURN (make_polar_complex_number_region(centre, ritv, aitv));
   END_FUNCTION;

   FUNCTION equal_cregion_pregion
```

```
   (crgn : cartesian_complex_number_region;
    prgn : polar_complex_number_region ) : LOGICAL;
LOCAL
   arng : REAL;
   amin : REAL;
   xc : REAL;
   yc : REAL;
   aitv : real_interval;
   xitv : real_interval;
   yitv : real_interval;
   c_in : BOOLEAN;
END_LOCAL;
   IF NOT EXISTS(crgn) OR NOT EXISTS(prgn) THEN
      RETURN (FALSE);
   END_IF;
   IF max_exists(prgn.distance_constraint) THEN
      RETURN (FALSE);
   END_IF;
   IF real_min(prgn.distance_constraint) <> 0.00000 THEN
      RETURN (FALSE);
   END_IF;
   c_in := min_included(prgn.distance_constraint);
   aitv := prgn.direction_constraint;
   amin := aitv.min;
   arng := aitv.max - amin;
   xc := prgn.centre.real_part;
   yc := prgn.centre.imag_part;
   xitv := crgn.real_constraint;
   yitv := crgn.imag_constraint;
   IF arng = 0.500000 * 3.14159 THEN
      IF amin = 0.00000 THEN
                RETURN ((((((NOT max_exists(xitv) AND NOT max_exists(yitv))
    AND min_exists(xitv)) AND min_exists(yitv)) AND (real_min(xitv) = xc)) AND
    (real_min(yitv) = yc)) AND ((((((c_in AND min_included(aitv)) AND
    max_included(aitv)) AND min_included(xitv)) AND min_included(yitv) OR
    (((NOT c_in AND NOT min_included(aitv)) AND max_included(aitv)) AND
    min_included(xitv)) AND NOT min_included(yitv)) OR (((NOT c_in AND
    min_included(aitv)) AND NOT max_included(aitv)) AND NOT
    min_included(xitv)) AND min_included(yitv)) OR (((NOT c_in AND NOT
    min_included(aitv)) AND NOT max_included(aitv)) AND NOT
    min_included(xitv)) AND NOT min_included(yitv)));
      END_IF;
      IF amin = 0.500000 * 3.14159 THEN
                RETURN ((((((max_exists(xitv) AND NOT max_exists(yitv)) AND
    NOT min_exists(xitv)) AND min_exists(yitv)) AND (real_max(xitv) = xc)) AND
    (real_min(yitv) = yc)) AND ((((((c_in AND min_included(aitv)) AND
    max_included(aitv)) AND max_included(xitv)) AND min_included(yitv) OR
    (((NOT c_in AND NOT min_included(aitv)) AND max_included(aitv)) AND
    max_included(xitv)) AND NOT min_included(yitv)) OR (((NOT c_in AND
    min_included(aitv)) AND NOT max_included(aitv)) AND NOT
    max_included(xitv)) AND min_included(yitv)) OR (((NOT c_in AND NOT
    min_included(aitv)) AND NOT max_included(aitv)) AND NOT
    max_included(xitv)) AND NOT min_included(yitv)));
      END_IF;
      IF amin = -3.14159 THEN
                RETURN ((((((max_exists(xitv) AND max_exists(yitv)) AND NOT
    min_exists(xitv)) AND NOT min_exists(yitv)) AND (real_max(xitv) = xc)) AND
    (real_max(yitv) = yc)) AND ((((((c_in AND min_included(aitv)) AND
    max_included(aitv)) AND max_included(xitv)) AND max_included(yitv) OR
```

```
    (((NOT c_in AND NOT min_included(aitv)) AND max_included(aitv)) AND
    max_included(xitv)) AND NOT max_included(yitv)) OR (((NOT c_in AND
    min_included(aitv)) AND NOT max_included(aitv)) AND NOT
    max_included(xitv)) AND max_included(yitv)) OR (((NOT c_in AND NOT
    min_included(aitv)) AND NOT max_included(aitv)) AND NOT
    max_included(xitv)) AND NOT max_included(yitv)));
      END_IF;
      IF amin = -0.500000 * 3.14159 THEN
               RETURN ((((((NOT max_exists(xitv) AND max_exists(yitv)) AND
    min_exists(xitv)) AND NOT min_exists(yitv)) AND (real_min(xitv) = xc)) AND
    (real_max(yitv) = yc)) AND ((((((c_in AND min_included(aitv)) AND
    max_included(aitv)) AND min_included(xitv)) AND max_included(yitv) OR
    (((NOT c_in AND NOT min_included(aitv)) AND max_included(aitv)) AND
    min_included(xitv)) AND NOT max_included(yitv)) OR (((NOT c_in AND
    min_included(aitv)) AND NOT max_included(aitv)) AND NOT
    min_included(xitv)) AND max_included(yitv)) OR (((NOT c_in AND NOT
    min_included(aitv)) AND NOT max_included(aitv)) AND NOT
    min_included(xitv)) AND NOT max_included(yitv)));
      END_IF;
   END_IF;
   IF arng = 3.14159 THEN
      IF amin = 0.00000 THEN
               RETURN (((((NOT max_exists(xitv) AND NOT max_exists(yitv)) AND
    NOT min_exists(xitv)) AND min_exists(yitv)) AND (real_min(yitv) = yc)) AND
    (((c_in AND min_included(aitv)) AND max_included(aitv)) AND
    min_included(yitv) OR ((NOT c_in AND NOT min_included(aitv)) AND NOT
    max_included(aitv)) AND NOT min_included(yitv)));
      END_IF;
      IF amin = 0.500000 * 3.14159 THEN
               RETURN (((((max_exists(xitv) AND NOT max_exists(yitv)) AND NOT
    min_exists(xitv)) AND NOT min_exists(yitv)) AND (real_max(xitv) = xc)) AND
    (((c_in AND min_included(aitv)) AND max_included(aitv)) AND
    max_included(xitv) OR ((NOT c_in AND NOT min_included(aitv)) AND NOT
    max_included(aitv)) AND NOT max_included(xitv)));
      END_IF;
      IF amin = -3.14159 THEN
               RETURN (((((NOT max_exists(xitv) AND max_exists(yitv)) AND NOT
    min_exists(xitv)) AND NOT min_exists(yitv)) AND (real_max(yitv) = yc)) AND
    (((c_in AND min_included(aitv)) AND max_included(aitv)) AND
    max_included(yitv) OR ((NOT c_in AND NOT min_included(aitv)) AND NOT
    max_included(aitv)) AND NOT max_included(yitv)));
      END_IF;
      IF amin = -0.500000 * 3.14159 THEN
               RETURN (((((NOT max_exists(xitv) AND NOT max_exists(yitv)) AND
    min_exists(xitv)) AND NOT min_exists(yitv)) AND (real_min(xitv) = xc)) AND
    (((c_in AND min_included(aitv)) AND max_included(aitv)) AND
    min_included(xitv) OR ((NOT c_in AND NOT min_included(aitv)) AND NOT
    max_included(aitv)) AND NOT min_included(xitv)));
      END_IF;
   END_IF;
   RETURN (FALSE);
END_FUNCTION;

FUNCTION equal_maths_functions
   (fun1 : maths_function;
    fun2 : maths_function ) : LOGICAL;
LOCAL
   cum : LOGICAL;
END_LOCAL;
```

```
      IF fun1 = fun2 THEN
         RETURN (TRUE);
      END_IF;
      cum := equal_maths_spaces(fun1.domain, fun2.domain);
      IF cum = FALSE THEN
         RETURN (FALSE);
      END_IF;
      cum := cum AND equal_maths_spaces(fun1.range, fun2.range);
      IF cum = FALSE THEN
         RETURN (FALSE);
      END_IF;
      RETURN (UNKNOWN);
   END_FUNCTION;

   FUNCTION equal_maths_spaces
      (spc1 : maths_space;
       spc2 : maths_space ) : LOGICAL;
   LOCAL
      spc1types : SET OF STRING := stripped_typeof(spc1);
      spc2types : SET OF STRING := stripped_typeof(spc2);
      set1 : SET OF maths_value;
      set2 : SET OF maths_value;
      cum : LOGICAL := TRUE;
      base : maths_space;
      expnt : INTEGER;
      factors : LIST OF maths_space;
      factors2 : LIST OF maths_space;
      fs1 : function_space;
      fs2 : function_space;
      cum2 : LOGICAL;
   END_LOCAL;
      IF spc1 = spc2 THEN
         RETURN (TRUE);
      END_IF;
      IF 'FINITE_SPACE' IN spc1types THEN
         set1 := spc1\finite_space.members;
         IF 'FINITE_SPACE' IN spc2types THEN
            set2 := spc2\finite_space.members;
            REPEAT i := 1 TO SIZEOF(set1);
               cum := cum AND member_of(set1[i], spc2);
               IF cum = FALSE THEN
                  RETURN (FALSE);
               END_IF;
            END_REPEAT;
            IF cum = TRUE THEN
               REPEAT i := 1 TO SIZEOF(set2);
                  cum := cum AND member_of(set2[i], spc1);
                  IF cum = FALSE THEN
                     RETURN (FALSE);
                  END_IF;
               END_REPEAT;
            END_IF;
            RETURN (cum);
         END_IF;
         IF 'FINITE_INTEGER_INTERVAL' IN spc2types THEN
            set2 := [];
            REPEAT i := spc2\finite_integer_interval.min TO
spc2\finite_integer_interval.max;
               set2 := set2 + [ i ];
```

```
      END_REPEAT;
      RETURN (equal_maths_spaces(spc1, make_finite_space(set2)));
   END_IF;
END_IF;
       IF ('FINITE_INTEGER_INTERVAL' IN spc1types) AND ('FINITE_SPACE' IN
 spc2types) THEN
   set1 := [];
          REPEAT i := spc1\finite_integer_interval.min TO
 spc1\finite_integer_interval.max;
      set1 := set1 + [ i ];
   END_REPEAT;
   RETURN (equal_maths_spaces(make_finite_space(set1), spc2));
END_IF;
       IF ('CARTESIAN_COMPLEX_NUMBER_REGION' IN spc1types) AND
 ('POLAR_COMPLEX_NUMBER_REGION' IN spc2types) THEN
   RETURN (equal_cregion_pregion(spc1, spc2));
END_IF;
       IF ('POLAR_COMPLEX_NUMBER_REGION' IN spc1types) AND
 ('CARTESIAN_COMPLEX_NUMBER_REGION' IN spc2types) THEN
   RETURN (equal_cregion_pregion(spc2, spc1));
END_IF;
IF 'UNIFORM_PRODUCT_SPACE' IN spc1types THEN
   base := spc1\uniform_product_space.base;
   expnt := spc1\uniform_product_space.exponent;
   IF 'UNIFORM_PRODUCT_SPACE' IN spc2types THEN
      IF expnt <> spc2\uniform_product_space.exponent THEN
         RETURN (FALSE);
      END_IF;
      RETURN (equal_maths_spaces(base, spc2\uniform_product_space.base));
   END_IF;
   IF 'LISTED_PRODUCT_SPACE' IN spc2types THEN
      factors := spc2\listed_product_space.factors;
      IF expnt <> SIZEOF(factors) THEN
         RETURN (FALSE);
      END_IF;
      REPEAT i := 1 TO SIZEOF(factors);
         cum := cum AND equal_maths_spaces(base, factors[i]);
         IF cum = FALSE THEN
            RETURN (FALSE);
         END_IF;
      END_REPEAT;
      RETURN (cum);
   END_IF;
END_IF;
IF 'LISTED_PRODUCT_SPACE' IN spc1types THEN
   factors := spc1\listed_product_space.factors;
   IF 'UNIFORM_PRODUCT_SPACE' IN spc2types THEN
      IF spc2\uniform_product_space.exponent <> SIZEOF(factors) THEN
         RETURN (FALSE);
      END_IF;
      base := spc2\uniform_product_space.base;
      REPEAT i := 1 TO SIZEOF(factors);
         cum := cum AND equal_maths_spaces(base, factors[i]);
         IF cum = FALSE THEN
            RETURN (FALSE);
         END_IF;
      END_REPEAT;
      RETURN (cum);
   END_IF;
```

```
      IF 'LISTED_PRODUCT_SPACE' IN spc2types THEN
         factors2 := spc2\listed_product_space.factors;
         IF SIZEOF(factors) <> SIZEOF(factors2) THEN
            RETURN (FALSE);
         END_IF;
         REPEAT i := 1 TO SIZEOF(factors);
            cum := cum AND equal_maths_spaces(factors[i], factors2[i]);
            IF cum = FALSE THEN
               RETURN (FALSE);
            END_IF;
         END_REPEAT;
         RETURN (cum);
      END_IF;
   END_IF;
         IF ('EXTENDED_TUPLE_SPACE' IN spc1types) AND ('EXTENDED_TUPLE_SPACE'
    IN spc2types) THEN
            RETURN (equal_maths_spaces(spc1\extended_tuple_space.extender,
    spc2\extended_tuple_space.extender) AND
    equal_maths_spaces(spc1\extended_tuple_space.base,
    spc2\extended_tuple_space.base));
         END_IF;
   IF ('FUNCTION_SPACE' IN spc1types) AND ('FUNCTION_SPACE' IN spc2types) THEN
      fs1 := spc1;
      fs2 := spc2;
      IF fs1.domain_constraint <> fs2.domain_constraint THEN
                  IF (fs1.domain_constraint = sc_equal) OR
       (fs2.domain_constraint = sc_equal) THEN
            RETURN (FALSE);
         END_IF;
         IF fs1.domain_constraint <> sc_subspace THEN
            fs1 := spc2;
            fs2 := spc1;
         END_IF;
                  IF (fs1.domain_constraint <> sc_subspace) OR
       (fs2.domain_constraint <> sc_member) THEN
            RETURN (UNKNOWN);
         END_IF;
                  IF any_space_satisfies(fs1.domain_constraint,
       fs1.domain_argument) <> any_space_satisfies(fs2.domain_constraint,
       fs2.domain_argument) THEN
            RETURN (FALSE);
         END_IF;
         IF NOT ('FINITE_SPACE' IN stripped_typeof(fs2.domain_argument)) THEN
            RETURN (FALSE);
         END_IF;
                  IF SIZEOF([ 'FINITE_SPACE', 'FINITE_INTEGER_INTERVAL' ] *
       stripped_typeof(fs1.domain_argument)) = 0 THEN
            RETURN (FALSE);
         END_IF;
         RETURN (UNKNOWN);
      END_IF;
      cum := equal_maths_spaces(fs1.domain_argument, fs2.domain_argument);
      IF cum = FALSE THEN
         RETURN (FALSE);
      END_IF;
      IF fs1.range_constraint <> fs2.range_constraint THEN
                  IF (fs1.range_constraint = sc_equal) OR
       (fs2.range_constraint = sc_equal) THEN
            RETURN (FALSE);
```

```
            END_IF;
            IF fs1.range_constraint <> sc_subspace THEN
               fs1 := spc2;
               fs2 := spc1;
            END_IF;
                      IF (fs1.range_constraint <> sc_subspace) OR
          (fs2.range_constraint <> sc_member) THEN
               RETURN (UNKNOWN);
            END_IF;
                      IF any_space_satisfies(fs1.range_constraint,
          fs1.range_argument) <> any_space_satisfies(fs2.range_constraint,
          fs2.range_argument) THEN
               RETURN (FALSE);
            END_IF;
            IF NOT ('FINITE_SPACE' IN stripped_typeof(fs2.range_argument)) THEN
               RETURN (FALSE);
            END_IF;
                      IF SIZEOF([ 'FINITE_SPACE', 'FINITE_INTEGER_INTERVAL' ] *
          stripped_typeof(fs1.range_argument)) = 0 THEN
               RETURN (FALSE);
            END_IF;
            RETURN (UNKNOWN);
         END_IF;
          cum := cum AND equal_maths_spaces(fs1.range_argument,
          fs2.range_argument);
         RETURN (cum);
      END_IF;
      RETURN (FALSE);
   END_FUNCTION;

   FUNCTION equal_maths_values
      (val1 : maths_value;
       val2 : maths_value ) : LOGICAL;
      FUNCTION mem_of_vs
         (val1 : maths_value;
          val2 : maths_value ) : LOGICAL;
         IF NOT has_values_space(val2) THEN
            RETURN (UNKNOWN);
         END_IF;
         IF NOT member_of(val1, values_space_of(val2)) THEN
            RETURN (FALSE);
         END_IF;
         RETURN (UNKNOWN);
      END_FUNCTION;
   LOCAL
      types1 : SET OF STRING;

      types2 : SET OF STRING;
      list1 : LIST OF maths_value;
      list2 : LIST OF maths_value;
      cum : LOGICAL := TRUE;
   END_LOCAL;
      IF NOT EXISTS(val1) OR NOT EXISTS(val2) THEN
         RETURN (FALSE);
      END_IF;
      IF val1 = val2 THEN
         RETURN (TRUE);
      END_IF;
      types1 := stripped_typeof(val1);
```

```
types2 := stripped_typeof(val2);
IF ('MATHS_ATOM' IN types1) OR ('COMPLEX_NUMBER_LITERAL' IN types1) THEN
   IF 'MATHS_ATOM' IN types2 THEN
      RETURN (FALSE);
   END_IF;
   IF 'COMPLEX_NUMBER_LITERAL' IN types2 THEN
      RETURN (FALSE);
   END_IF;
   IF 'LIST' IN types2 THEN
      RETURN (FALSE);
   END_IF;
   IF 'MATHS_SPACE' IN types2 THEN
      RETURN (FALSE);
   END_IF;
   IF 'MATHS_FUNCTION' IN types2 THEN
      RETURN (FALSE);
   END_IF;
   IF 'GENERIC_EXPRESSION' IN types2 THEN
      RETURN (mem_of_vs(val1, val2));
   END_IF;
   RETURN (UNKNOWN);
END_IF;
IF ('MATHS_ATOM' IN types2) OR ('COMPLEX_NUMBER_LITERAL' IN types2) THEN
   RETURN (equal_maths_values(val2, val1));
END_IF;
IF 'LIST' IN types1 THEN
   IF 'LIST' IN types2 THEN
      list1 := val1;
      list2 := val2;
      IF SIZEOF(list1) <> SIZEOF(list2) THEN
         RETURN (FALSE);
      END_IF;
      REPEAT i := 1 TO SIZEOF(list1);
         cum := cum AND equal_maths_values(list1[i], list2[i]);
         IF cum = FALSE THEN
            RETURN (FALSE);
         END_IF;
      END_REPEAT;
      RETURN (cum);
   END_IF;
   IF 'MATHS_SPACE' IN types2 THEN
      RETURN (FALSE);
   END_IF;
   IF 'MATHS_FUNCTION' IN types2 THEN
      RETURN (FALSE);
   END_IF;
   IF 'GENERIC_EXPRESSION' IN types2 THEN
      RETURN (mem_of_vs(val1, val2));
   END_IF;
   RETURN (UNKNOWN);
END_IF;
IF 'LIST' IN types2 THEN
   RETURN (equal_maths_values(val2, val1));
END_IF;
IF 'MATHS_SPACE' IN types1 THEN
   IF 'MATHS_SPACE' IN types2 THEN
      RETURN (equal_maths_spaces(val1, val2));
   END_IF;
   IF 'MATHS_FUNCTION' IN types2 THEN
```

```
         RETURN (FALSE);
      END_IF;
      IF 'GENERIC_EXPRESSION' IN types2 THEN
         RETURN (mem_of_vs(val1, val2));
      END_IF;
      RETURN (UNKNOWN);
   END_IF;
   IF 'MATHS_SPACE' IN types2 THEN
      RETURN (equal_maths_values(val2, val1));
   END_IF;
   IF 'MATHS_FUNCTION' IN types1 THEN
      IF 'MATHS_FUNCTION' IN types2 THEN
         RETURN (equal_maths_functions(val1, val2));
      END_IF;
      IF 'GENERIC_EXPRESSION' IN types2 THEN
         RETURN (mem_of_vs(val1, val2));
      END_IF;
      RETURN (UNKNOWN);
   END_IF;
   IF 'MATHS_FUNCTION' IN types2 THEN
      RETURN (equal_maths_values(val2, val1));
   END_IF;
         IF ('GENERIC_EXPRESSION' IN types1) AND ('GENERIC_EXPRESSION' IN
    types2) THEN
      IF NOT has_values_space(val1) OR NOT has_values_space(val2) THEN
         RETURN (UNKNOWN);
      END_IF;
            IF NOT compatible_spaces(values_space_of(val1),
    values_space_of(val2)) THEN
         RETURN (FALSE);
      END_IF;
   END_IF;
   RETURN (UNKNOWN);
END_FUNCTION;

FUNCTION es_subspace_of_es
   (es1 : elementary_space_enumerators;
    es2 : elementary_space_enumerators ) : BOOLEAN;
   IF NOT EXISTS(es1) OR NOT EXISTS(es2) THEN
      RETURN (FALSE);
   END_IF;
   IF es1 = es2 THEN
      RETURN (TRUE);
   END_IF;
   IF es2 = es_generics THEN
      RETURN (TRUE);
   END_IF;
   IF (es1 = es_booleans) AND (es2 = es_logicals) THEN
      RETURN (TRUE);
   END_IF;
         IF (es2 = es_numbers) AND (((es1 = es_complex_numbers) OR (es1 =
    es_reals)) OR (es1 = es_integers)) THEN
      RETURN (TRUE);
   END_IF;
   RETURN (FALSE);
END_FUNCTION;

FUNCTION expression_is_constant
   (expr : generic_expression ) : BOOLEAN;
```

```
      RETURN (bool(SIZEOF(free_variables_of(expr)) = 0));
   END_FUNCTION;

   FUNCTION extract_factors
      (tspace : tuple_space;
       m : INTEGER;
       n : INTEGER ) : tuple_space;
   LOCAL
      tsp : tuple_space := the_zero_tuple_space;
   END_LOCAL;
      REPEAT i := m TO n;
         tsp := assoc_product_space(tsp, factor_space(tspace, i));
      END_REPEAT;
      RETURN (tsp);
   END_FUNCTION;

   FUNCTION extremal_position_check
      (fun : linearized_table_function ) : BOOLEAN;
   LOCAL
      source_domain : maths_space;
      source_interval : finite_integer_interval;
      index : INTEGER := 1;
      base : INTEGER;
      shape : LIST OF positive_integer;
      ndim : positive_integer;
      slo : INTEGER;
      shi : INTEGER;
      sublo : LIST OF INTEGER := [];
      subhi : LIST OF INTEGER := [];
   END_LOCAL;

      IF NOT EXISTS(fun) THEN
         RETURN (FALSE);
      END_IF;
      source_domain := factor1(fun.source.domain);
      IF schema_prefix + 'TUPLE_SPACE' IN TYPEOF(source_domain) THEN
         source_domain := factor1(source_domain);
      END_IF;
            IF NOT (schema_prefix + 'FINITE_INTEGER_INTERVAL' IN
       TYPEOF(source_domain)) THEN
         RETURN (FALSE);
      END_IF;
      source_interval := source_domain;
      base := fun\explicit_table_function.index_base;
      shape := fun\explicit_table_function.shape;
      IF schema_prefix + 'STANDARD_TABLE_FUNCTION' IN TYPEOF(fun) THEN
         REPEAT j := 1 TO SIZEOF(shape);
            index := index * shape[j];
         END_REPEAT;
         index := fun.first + index - 1;
                RETURN (bool((source_interval.min <= index) AND (index <=
       source_interval.max)));
      END_IF;
      IF schema_prefix + 'REGULAR_TABLE_FUNCTION' IN TYPEOF(fun) THEN
         ndim := SIZEOF(fun\explicit_table_function.shape);
         REPEAT j := 1 TO ndim;
            slo := base;
            shi := base + shape[j] - 1;
            IF fun\regular_table_function.increments[j] >= 0 THEN
```

```
            INSERT( sublo, slo, j - 1 );
            INSERT( subhi, shi, j - 1 );
         ELSE
            INSERT( sublo, shi, j - 1 );
            INSERT( subhi, slo, j - 1 );
         END_IF;
      END_REPEAT;
               index := regular_indexing(sublo, base, shape,
       fun\regular_table_function.increments, fun.first);
      IF NOT ((source_interval.min <= index) AND (index <=
      source_interval.max)) THEN
         RETURN (FALSE);
      END_IF;
               index := regular_indexing(subhi, base, shape,
       fun\regular_table_function.increments, fun.first);
       IF NOT ((source_interval.min <= index) AND (index <=
       source_interval.max)) THEN
         RETURN (FALSE);
      END_IF;
      RETURN (TRUE);
   END_IF;
   RETURN (FALSE);
END_FUNCTION;

FUNCTION factor1
   (tspace : tuple_space ) : maths_space;
LOCAL
   typenames : SET OF STRING := TYPEOF(tspace);
END_LOCAL;
   IF schema_prefix + 'UNIFORM_PRODUCT_SPACE' IN typenames THEN
      RETURN (tspace\uniform_product_space.base);
   END_IF;
   IF schema_prefix + 'LISTED_PRODUCT_SPACE' IN typenames THEN
      RETURN (tspace\listed_product_space.factors[1]);
   END_IF;
   IF schema_prefix + 'EXTENDED_TUPLE_SPACE' IN typenames THEN
      RETURN (factor1(tspace\extended_tuple_space.base));
   END_IF;
   RETURN (?);
END_FUNCTION;

FUNCTION factor_space
   (tspace : tuple_space;
    idx : positive_integer ) : maths_space;
LOCAL
   typenames : SET OF STRING := TYPEOF(tspace);
END_LOCAL;
   IF schema_prefix + 'UNIFORM_PRODUCT_SPACE' IN typenames THEN
      IF idx <= tspace\uniform_product_space.exponent THEN
         RETURN (tspace\uniform_product_space.base);
      END_IF;
      RETURN (?);
   END_IF;
   IF schema_prefix + 'LISTED_PRODUCT_SPACE' IN typenames THEN
      IF idx <= SIZEOF(tspace\listed_product_space.factors) THEN
         RETURN (tspace\listed_product_space.factors[idx]);
      END_IF;
      RETURN (?);
   END_IF;
```

```
      IF schema_prefix + 'EXTENDED_TUPLE_SPACE' IN typenames THEN
         IF idx <= space_dimension(tspace\extended_tuple_space.base) THEN
            RETURN (factor_space(tspace\extended_tuple_space.base, idx));
         END_IF;
         RETURN (tspace\extended_tuple_space.extender);
      END_IF;
      RETURN (?);
   END_FUNCTION;

   FUNCTION first_proj_axis
      (z_axis : direction;
       arg : direction ) : direction;
   LOCAL
      x_axis : direction;
      v : direction;
      z : direction;
      x_vec : vector;
   END_LOCAL;
      IF NOT EXISTS(z_axis) THEN
         RETURN (?);
      ELSE
         z := normalise(z_axis);
         IF NOT EXISTS(arg) THEN
             IF (z.direction_ratios <> [ 1.00000, 0.00000, 0.00000 ]) AND
          (z.direction_ratios <> [ -1.00000, 0.00000, 0.00000 ]) THEN
               v := dummy_gri || direction([ 1.00000, 0.00000, 0.00000 ]);
           ELSE
               v := dummy_gri || direction([ 0.00000, 1.00000, 0.00000 ]);
           END_IF;
         ELSE
            IF arg.dim <> 3 THEN
               RETURN (?);
            END_IF;
            IF cross_product(arg, z).magnitude = 0.00000 THEN
               RETURN (?);
            ELSE
               v := normalise(arg);
            END_IF;
         END_IF;
         x_vec := scalar_times_vector(dot_product(v, z), z);
         x_axis := vector_difference(v, x_vec).orientation;
         x_axis := normalise(x_axis);
      END_IF;
      RETURN (x_axis);
   END_FUNCTION;

   FUNCTION free_variables_of
      (expr : generic_expression ) : SET OF generic_variable;
   LOCAL
      typenames : SET OF STRING := stripped_typeof(expr);
      result : SET OF generic_variable := [];
      exprs : LIST OF generic_expression := [];
   END_LOCAL;
      IF 'GENERIC_LITERAL' IN typenames THEN
         RETURN (result);
      END_IF;
      IF 'GENERIC_VARIABLE' IN typenames THEN
         result := result + expr;
         RETURN (result);
```

```
   END_IF;
   IF 'QUANTIFIER_EXPRESSION' IN typenames THEN
               exprs := QUERY (ge <*
       expr\multiple_arity_generic_expression.operands| NOT (ge IN
       expr\quantifier_expression.variables));
      REPEAT i := 1 TO SIZEOF(exprs);
         result := result + free_variables_of(exprs[i]);
      END_REPEAT;
      REPEAT i := 1 TO SIZEOF(expr\quantifier_expression.variables);
         result := result - expr\quantifier_expression.variables[i];
      END_REPEAT;
      RETURN (result);
   END_IF;
   IF 'UNARY_GENERIC_EXPRESSION' IN typenames THEN
      RETURN (free_variables_of(expr\unary_generic_expression.operand));
   END_IF;
   IF 'BINARY_GENERIC_EXPRESSION' IN typenames THEN
      result := free_variables_of(expr\binary_generic_expression.operands[1]);
               RETURN (result +
       free_variables_of(expr\binary_generic_expression.operands[2]));
   END_IF;
   IF 'MULTIPLE_ARITY_GENERIC_EXPRESSION' IN typenames THEN
               REPEAT i := 1 TO
       SIZEOF(expr\multiple_arity_generic_expression.operands);
       result := result +
       free_variables_of(expr\multiple_arity_generic_expression.operands[i]);
      END_REPEAT;
      RETURN (result);
   END_IF;
   RETURN (result);
END_FUNCTION;

FUNCTION function_applicability
   (func : maths_function_select;
    arguments : LIST [1:?] OF maths_value ) : BOOLEAN;
LOCAL
   domain : tuple_space := convert_to_maths_function(func).domain;
   domain_types : SET OF STRING := TYPEOF(domain);
   narg : positive_integer := SIZEOF(arguments);
   arg : generic_expression;
END_LOCAL;
   IF schema_prefix + 'PRODUCT_SPACE' IN domain_types THEN
      IF space_dimension(domain) <> narg THEN
         RETURN (FALSE);
      END_IF;
   ELSE
      IF schema_prefix + 'EXTENDED_TUPLE_SPACE' IN domain_types THEN
         IF space_dimension(domain) > narg THEN
            RETURN (FALSE);
         END_IF;
      ELSE
         RETURN (FALSE);
      END_IF;
   END_IF;
   REPEAT i := 1 TO narg;
      arg := convert_to_operand(arguments[i]);
      IF NOT has_values_space(arg) THEN
         RETURN (FALSE);
      END_IF;
```

```
        IF NOT compatible_spaces(factor_space(domain, i), values_space_of(arg))
        THEN
          RETURN (FALSE);
       END_IF;
   END_REPEAT;
   RETURN (TRUE);
END_FUNCTION;

FUNCTION function_is_1d_array
   (func : maths_function ) : BOOLEAN;
LOCAL
   temp : maths_space;
END_LOCAL;
   IF NOT EXISTS(func) THEN
      RETURN (FALSE);
   END_IF;
   IF space_dimension(func.domain) <> 1 THEN
      RETURN (FALSE);
   END_IF;
   temp := factor1(func.domain);
   IF schema_prefix + 'PRODUCT_SPACE' IN TYPEOF(temp) THEN
      IF space_dimension(temp) <> 1 THEN
         RETURN (FALSE);
      END_IF;
      temp := factor1(temp);
   END_IF;
   IF schema_prefix + 'FINITE_INTEGER_INTERVAL' IN TYPEOF(temp) THEN
      RETURN (TRUE);
   END_IF;
   RETURN (FALSE);
END_FUNCTION;

FUNCTION function_is_1d_table
   (func : maths_function ) : BOOLEAN;
LOCAL
   temp : maths_space;
   itvl : finite_integer_interval;
END_LOCAL;
   IF NOT EXISTS(func) THEN
      RETURN (FALSE);
   END_IF;
   IF space_dimension(func.domain) <> 1 THEN
      RETURN (FALSE);
   END_IF;
   temp := factor1(func.domain);
   IF schema_prefix + 'PRODUCT_SPACE' IN TYPEOF(temp) THEN
      IF space_dimension(temp) <> 1 THEN
         RETURN (FALSE);
      END_IF;
      temp := factor1(temp);
   END_IF;
   IF schema_prefix + 'FINITE_INTEGER_INTERVAL' IN TYPEOF(temp) THEN
      itvl := temp;
      RETURN (bool((itvl.min = 0) OR (itvl.min = 1)));
   END_IF;
   RETURN (FALSE);
END_FUNCTION;

FUNCTION function_is_2d_table
```

```
   (func : maths_function ) : BOOLEAN;
LOCAL
   temp : maths_space;
   pspace : product_space;
   itvl1 : finite_integer_interval;
   itvl2 : finite_integer_interval;
END_LOCAL;
   IF NOT EXISTS(func) THEN
      RETURN (FALSE);
   END_IF;
   IF space_dimension(func.domain) <> 1 THEN
      RETURN (FALSE);
   END_IF;
   temp := factor1(func.domain);
   IF NOT ('PRODUCT_SPACE' IN stripped_typeof(temp)) THEN
      RETURN (FALSE);
   END_IF;
   pspace := temp;
   IF space_dimension(pspace) <> 2 THEN
      RETURN (FALSE);
   END_IF;
   temp := factor1(pspace);
   IF NOT ('FINITE_INTEGER_INTERVAL' IN stripped_typeof(temp)) THEN
      RETURN (FALSE);
   END_IF;
   itvl1 := temp;
   temp := factor_space(pspace, 2);
   IF NOT ('FINITE_INTEGER_INTERVAL' IN stripped_typeof(temp)) THEN
      RETURN (FALSE);
   END_IF;
   itvl2 := temp;
          RETURN (bool((itvl1.min = itvl2.min) AND ((itvl1.min = 0) OR
    (itvl1.min = 1))));
END_FUNCTION;

FUNCTION function_is_array
   (func : maths_function ) : BOOLEAN;
LOCAL
   tspace : tuple_space;
   temp : maths_space;
END_LOCAL;
   IF NOT EXISTS(func) THEN
      RETURN (FALSE);
   END_IF;
   tspace := func.domain;
          IF (space_dimension(tspace) = 1) AND (schema_prefix + 'TUPLE_SPACE'
    IN TYPEOF(factor1(tspace))) THEN
      tspace := factor1(tspace);
   END_IF;
   IF NOT (schema_prefix + 'PRODUCT_SPACE' IN TYPEOF(tspace)) THEN
      RETURN (FALSE);
   END_IF;
   REPEAT i := 1 TO space_dimension(tspace);
      temp := factor_space(tspace, i);
      IF NOT (schema_prefix + 'FINITE_INTEGER_INTERVAL' IN TYPEOF(temp)) THEN
         RETURN (FALSE);
      END_IF;
   END_REPEAT;
   RETURN (TRUE);
```

```
END_FUNCTION;

FUNCTION function_is_table
   (func : maths_function ) : BOOLEAN;
LOCAL
   tspace : tuple_space;
   temp : maths_space;
   base : INTEGER;
END_LOCAL;
   IF NOT EXISTS(func) THEN
      RETURN (FALSE);
   END_IF;
   tspace := func.domain;
          IF (space_dimension(tspace) = 1) AND (schema_prefix + 'TUPLE_SPACE'
    IN TYPEOF(factor1(tspace))) THEN
      tspace := factor1(tspace);
   END_IF;
   IF NOT (schema_prefix + 'PRODUCT_SPACE' IN TYPEOF(tspace)) THEN
      RETURN (FALSE);
   END_IF;
   temp := factor1(tspace);
   IF NOT (schema_prefix + 'FINITE_INTEGER_INTERVAL' IN TYPEOF(temp)) THEN
      RETURN (FALSE);
   END_IF;
   base := temp\finite_integer_interval.min;
   IF (base <> 0) AND (base <> 1) THEN
      RETURN (FALSE);
   END_IF;
   REPEAT i := 2 TO space_dimension(tspace);
      temp := factor_space(tspace, i);
      IF NOT (schema_prefix + 'FINITE_INTEGER_INTERVAL' IN TYPEOF(temp)) THEN
         RETURN (FALSE);
      END_IF;
      IF temp\finite_integer_interval.min <> base THEN
         RETURN (FALSE);
      END_IF;
   END_REPEAT;
   RETURN (TRUE);
END_FUNCTION;

FUNCTION get_description_value
   (obj : description_attribute_select ) : text;
LOCAL
          description_bag : BAG OF description_attribute := USEDIN(obj,
    'ENGINEERING_PROPERTIES_SCHEMA.' + 'DESCRIPTION_ATTRIBUTE.' +
    'DESCRIBED_ITEM');
END_LOCAL;
   IF SIZEOF(description_bag) = 1 THEN
      RETURN (description_bag[1].attribute_value);
   ELSE
      RETURN (?);
   END_IF;
END_FUNCTION;

FUNCTION get_id_value
   (obj : id_attribute_select ) : identifier;
LOCAL
          id_bag : BAG OF id_attribute := USEDIN(obj,
    'ENGINEERING_PROPERTIES_SCHEMA.' + 'ID_ATTRIBUTE.' + 'IDENTIFIED_ITEM');
```

```
   END_LOCAL;
      IF SIZEOF(id_bag) = 1 THEN
         RETURN (id_bag[1].attribute_value);
      ELSE
         RETURN (?);
      END_IF;
   END_FUNCTION;

   FUNCTION get_multi_language
      (x : attribute_value_assignment ) : label;
         LOCAL
         alas : BAG OF attribute_language_assignment := USEDIN(x.items[1],
'ENGINEERING_PROPERTIES_SCHEMA.' + 'ATTRIBUTE_LANGUAGE_ASSIGNMENT.ITEMS');
       END_LOCAL;
       IF SIZEOF(alas) > 0 THEN
          RETURN (alas[1].language);
       END_IF;
       RETURN (?);
   END_FUNCTION;

   FUNCTION get_name_value
      (obj : name_attribute_select ) : label;
   LOCAL
         name_bag : BAG OF name_attribute := USEDIN(obj,
    'ENGINEERING_PROPERTIES_SCHEMA.' + 'NAME_ATTRIBUTE.' + 'NAMED_ITEM');
   END_LOCAL;
      IF SIZEOF(name_bag) = 1 THEN
         RETURN (name_bag[1].attribute_value);
      ELSE
         RETURN (?);
      END_IF;
   END_FUNCTION;

   FUNCTION get_role
      (obj : role_select ) : object_role;
   LOCAL
         role_bag : BAG OF role_association := USEDIN(obj,
    'ENGINEERING_PROPERTIES_SCHEMA.' + 'ROLE_ASSOCIATION.' + 'ITEM_WITH_ROLE');
   END_LOCAL;
      IF SIZEOF(role_bag) = 1 THEN
         RETURN (role_bag[1].role);
      ELSE
         RETURN (?);
      END_IF;
   END_FUNCTION;

   FUNCTION has_values_space
      (expr : generic_expression ) : BOOLEAN;
   LOCAL
      typenames : SET OF STRING := stripped_typeof(expr);
   END_LOCAL;
      IF 'EXPRESSION' IN typenames THEN
               RETURN (bool((('NUMERIC_EXPRESSION' IN typenames) OR
       ('STRING_EXPRESSION' IN typenames)) OR ('BOOLEAN_EXPRESSION' IN
       typenames)));
      END_IF;
      IF 'MATHS_FUNCTION' IN typenames THEN
         RETURN (TRUE);
      END_IF;
```

```
      IF 'FUNCTION_APPLICATION' IN typenames THEN
         RETURN (TRUE);
      END_IF;
      IF 'MATHS_SPACE' IN typenames THEN
         RETURN (TRUE);
      END_IF;
      IF 'MATHS_VARIABLE' IN typenames THEN
         RETURN (TRUE);
      END_IF;
      IF 'DEPENDENT_VARIABLE_DEFINITION' IN typenames THEN
         RETURN (has_values_space(expr\unary_generic_expression.operand));
      END_IF;
      IF 'COMPLEX_NUMBER_LITERAL' IN typenames THEN
         RETURN (TRUE);
      END_IF;
      IF 'LOGICAL_LITERAL' IN typenames THEN
         RETURN (TRUE);
      END_IF;
      IF 'BINARY_LITERAL' IN typenames THEN
         RETURN (TRUE);
      END_IF;
      IF 'MATHS_ENUM_LITERAL' IN typenames THEN
         RETURN (TRUE);
      END_IF;
      IF 'REAL_TUPLE_LITERAL' IN typenames THEN
         RETURN (TRUE);
      END_IF;
      IF 'INTEGER_TUPLE_LITERAL' IN typenames THEN
         RETURN (TRUE);
      END_IF;
      IF 'ATOM_BASED_LITERAL' IN typenames THEN
         RETURN (TRUE);
      END_IF;
      IF 'MATHS_TUPLE_LITERAL' IN typenames THEN
         RETURN (TRUE);
      END_IF;
      IF 'PARTIAL_DERIVATIVE_EXPRESSION' IN typenames THEN
         RETURN (TRUE);
      END_IF;
      IF 'DEFINITE_INTEGRAL_EXPRESSION' IN typenames THEN
         RETURN (TRUE);
      END_IF;
      RETURN (FALSE);
   END_FUNCTION;

   FUNCTION is_SQL_mappable
      (arg : expression ) : BOOLEAN;
   LOCAL
      i : INTEGER;
   END_LOCAL;
            IF 'ENGINEERING_PROPERTIES_SCHEMA.SIMPLE_NUMERIC_EXPRESSION' IN
       TYPEOF(arg) THEN
         RETURN (TRUE);
      END_IF;
            IF 'ENGINEERING_PROPERTIES_SCHEMA.SQL_MAPPABLE_DEFINED_FUNCTION' IN
       TYPEOF(arg) THEN
         RETURN (TRUE);
      END_IF;
      IF 'ENGINEERING_PROPERTIES_SCHEMA.MINUS_FUNCTION' IN TYPEOF(arg) THEN
```

```
   RETURN (is_SQL_mappable(arg\unary_numeric_expression.operand));
END_IF;
       IF ((((((((((('ENGINEERING_PROPERTIES_SCHEMA.ABS_FUNCTION' IN
 TYPEOF(arg)) OR ('ENGINEERING_PROPERTIES_SCHEMA.SIN_FUNCTION' IN
 TYPEOF(arg))) OR ('ENGINEERING_PROPERTIES_SCHEMA.COS_FUNCTION' IN
 TYPEOF(arg))) OR ('ENGINEERING_PROPERTIES_SCHEMA.TAN_FUNCTION' IN
 TYPEOF(arg))) OR ('ENGINEERING_PROPERTIES_SCHEMA.ASIN_FUNCTION' IN
 TYPEOF(arg))) OR ('ENGINEERING_PROPERTIES_SCHEMA.ACOS_FUNCTION' IN
 TYPEOF(arg))) OR ('ENGINEERING_PROPERTIES_SCHEMA.ATAN_FUNCTION' IN
 TYPEOF(arg))) OR ('ENGINEERING_PROPERTIES_SCHEMA.EXP_FUNCTION' IN
 TYPEOF(arg))) OR ('ENGINEERING_PROPERTIES_SCHEMA.LOG_FUNCTION' IN
 TYPEOF(arg))) OR ('ENGINEERING_PROPERTIES_SCHEMA.LOG2_FUNCTION' IN
 TYPEOF(arg))) OR ('ENGINEERING_PROPERTIES_SCHEMA.LOG10_FUNCTION' IN
 TYPEOF(arg))) OR ('ENGINEERING_PROPERTIES_SCHEMA.SQUARE_ROOT_FUNCTION' IN
 TYPEOF(arg))) OR ('ENGINEERING_PROPERTIES_SCHEMA.VALUE_FUNCTION' IN
 TYPEOF(arg))) OR ('ENGINEERING_PROPERTIES_SCHEMA.LENGTH_FUNCTION' IN
 TYPEOF(arg)) THEN
         RETURN (FALSE);
END_IF;
       IF ((('ENGINEERING_PROPERTIES_SCHEMA.PLUS_EXPRESSION' IN
 TYPEOF(arg)) OR ('ENGINEERING_PROPERTIES_SCHEMA.MULT_EXPRESSION' IN
 TYPEOF(arg))) OR ('ENGINEERING_PROPERTIES_SCHEMA.MAXIMUM_FUNCTION' IN
 TYPEOF(arg))) OR ('ENGINEERING_PROPERTIES_SCHEMA.MINIMUM_FUNCTION' IN
 TYPEOF(arg)) THEN
   REPEAT i := 1 TO SIZEOF(arg\multiple_arity_numeric_expression.operands);
              IF NOT
    is_SQL_mappable(arg\multiple_arity_numeric_expression.operands[i]) THEN
         RETURN (FALSE);
      END_IF;
   END_REPEAT;
   RETURN (TRUE);
END_IF;
       IF ('ENGINEERING_PROPERTIES_SCHEMA.MINUS_EXPRESSION' IN TYPEOF(arg))
 OR ('ENGINEERING_PROPERTIES_SCHEMA.SLASH_EXPRESSION' IN TYPEOF(arg)) THEN
   RETURN (is_SQL_mappable(arg\binary_numeric_expression.operands[1]) AND
   is_SQL_mappable(arg\binary_numeric_expression.operands[2]));
END_IF;
       IF (('ENGINEERING_PROPERTIES_SCHEMA.DIV_EXPRESSION' IN TYPEOF(arg))
 OR ('ENGINEERING_PROPERTIES_SCHEMA.MOD_EXPRESSION' IN TYPEOF(arg))) OR
 ('ENGINEERING_PROPERTIES_SCHEMA.POWER_EXPRESSION' IN TYPEOF(arg)) THEN
   RETURN (FALSE);
END_IF;
       IF 'ENGINEERING_PROPERTIES_SCHEMA.SIMPLE_BOOLEAN_EXPRESSION' IN
 TYPEOF(arg) THEN
   RETURN (TRUE);
END_IF;
IF 'ENGINEERING_PROPERTIES_SCHEMA.NOT_EXPRESSION' IN TYPEOF(arg) THEN
   RETURN (is_SQL_mappable(arg\unary_generic_expression.operand));
END_IF;
       IF ('ENGINEERING_PROPERTIES_SCHEMA.ODD_FUNCTION' IN TYPEOF(arg)) OR
 ('ENGINEERING_PROPERTIES_SCHEMA.XOR_EXPRESSION' IN TYPEOF(arg)) THEN
   RETURN (FALSE);
END_IF;
       IF ('ENGINEERING_PROPERTIES_SCHEMA.AND_EXPRESSION' IN TYPEOF(arg))
 OR ('ENGINEERING_PROPERTIES_SCHEMA.OR_EXPRESSION' IN TYPEOF(arg)) THEN
   REPEAT i := 1 TO SIZEOF(arg\multiple_arity_boolean_expression.operands);
             IF NOT
 is_SQL_mappable(arg\multiple_arity_boolean_expression.operands[i]) THEN
         RETURN (FALSE);
```

```
         END_IF;
      END_REPEAT;
      RETURN (TRUE);
   END_IF;
   IF 'ENGINEERING_PROPERTIES_SCHEMA.EQUALS_EXPRESSION' IN TYPEOF(arg) THEN
            RETURN
    (is_SQL_mappable(arg\binary_generic_expression.operands[1]) AND
    is_SQL_mappable(arg\binary_generic_expression.operands[2]));
   END_IF;
          IF ((((((('ENGINEERING_PROPERTIES_SCHEMA.COMPARISON_EQUAL' IN
    TYPEOF(arg)) OR ('ENGINEERING_PROPERTIES_SCHEMA.COMPARISON_GREATER' IN
    TYPEOF(arg))) OR ('ENGINEERING_PROPERTIES_SCHEMA.COMPARISON_GREATER_EQUAL'
    IN TYPEOF(arg))) OR ('ENGINEERING_PROPERTIES_SCHEMA.COMPARISON_LESS' IN
    TYPEOF(arg))) OR ('ENGINEERING_PROPERTIES_SCHEMA.COMPARISON_LESS_EQUAL' IN
    TYPEOF(arg))) OR ('ENGINEERING_PROPERTIES_SCHEMA.COMPARISON_NOT_EQUAL' IN
    TYPEOF(arg))) OR ('ENGINEERING_PROPERTIES_SCHEMA.LIKE_EXPRESSION' IN
    TYPEOF(arg)) THEN
      RETURN (is_SQL_mappable(arg\comparison_expression.operands[1]) AND
      is_SQL_mappable(arg\comparison_expression.operands[2]));
   END_IF;
   IF 'ENGINEERING_PROPERTIES_SCHEMA.INTERVAL_EXPRESSION' IN TYPEOF(arg) THEN
            RETURN ((is_SQL_mappable(arg\interval_expression.interval_low)
    AND is_SQL_mappable(arg\interval_expression.interval_high)) AND
    is_SQL_mappable(arg\interval_expression.interval_item));
   END_IF;
          IF (('ENGINEERING_PROPERTIES_SCHEMA.NUMERIC_DEFINED_FUNCTION' IN
    TYPEOF(arg)) OR ('ENGINEERING_PROPERTIES_SCHEMA.BOOLEAN_DEFINED_FUNCTION'
    IN TYPEOF(arg))) OR
    ('ENGINEERING_PROPERTIES_SCHEMA.STRING_DEFINED_FUNCTION' IN TYPEOF(arg))
    THEN
      RETURN (FALSE);
   END_IF;
          IF 'ENGINEERING_PROPERTIES_SCHEMA.SIMPLE_STRING_EXPRESSION' IN
    TYPEOF(arg) THEN
      RETURN (TRUE);
   END_IF;
          IF ((('ENGINEERING_PROPERTIES_SCHEMA.INDEX_EXPRESSION' IN
    TYPEOF(arg)) OR ('ENGINEERING_PROPERTIES_SCHEMA.SUBSTRING_EXPRESSION' IN
    TYPEOF(arg))) OR ('ENGINEERING_PROPERTIES_SCHEMA.CONCAT_EXPRESSION' IN
    TYPEOF(arg))) OR ('ENGINEERING_PROPERTIES_SCHEMA.FORMAT_FUNCTION' IN
    TYPEOF(arg)) THEN
      RETURN (FALSE);
   END_IF;
   RETURN (FALSE);
END_FUNCTION;

FUNCTION is_acyclic
   (arg : generic_expression ) : BOOLEAN;
   RETURN (acyclic(arg, []));
END_FUNCTION;

FUNCTION is_int_expr
   (arg : numeric_expression ) : BOOLEAN;
LOCAL
   i : INTEGER;
END_LOCAL;
   IF 'ENGINEERING_PROPERTIES_SCHEMA.INT_LITERAL' IN TYPEOF(arg) THEN
      RETURN (TRUE);
   END_IF;
```

```
IF 'ENGINEERING_PROPERTIES_SCHEMA.REAL_LITERAL' IN TYPEOF(arg) THEN
   RETURN (FALSE);
END_IF;
IF 'ENGINEERING_PROPERTIES_SCHEMA.INT_NUMERIC_VARIABLE' IN TYPEOF(arg) THEN
   RETURN (TRUE);
END_IF;
       IF 'ENGINEERING_PROPERTIES_SCHEMA.REAL_NUMERIC_VARIABLE' IN
 TYPEOF(arg) THEN
   RETURN (FALSE);
END_IF;
IF 'ENGINEERING_PROPERTIES_SCHEMA.ABS_FUNCTION' IN TYPEOF(arg) THEN
   RETURN (is_int_expr(arg\unary_numeric_expression.operand));
END_IF;
IF 'ENGINEERING_PROPERTIES_SCHEMA.MINUS_FUNCTION' IN TYPEOF(arg) THEN
   RETURN (is_int_expr(arg\unary_numeric_expression.operand));
END_IF;
       IF ((((((((('ENGINEERING_PROPERTIES_SCHEMA.SIN_FUNCTION' IN
 TYPEOF(arg)) OR ('ENGINEERING_PROPERTIES_SCHEMA.COS_FUNCTION' IN
 TYPEOF(arg))) OR ('ENGINEERING_PROPERTIES_SCHEMA.TAN_FUNCTION' IN
 TYPEOF(arg))) OR ('ENGINEERING_PROPERTIES_SCHEMA.ASIN_FUNCTION' IN
 TYPEOF(arg))) OR ('ENGINEERING_PROPERTIES_SCHEMA.ACOS_FUNCTION' IN
 TYPEOF(arg))) OR ('ENGINEERING_PROPERTIES_SCHEMA.ATAN_FUNCTION' IN
 TYPEOF(arg))) OR ('ENGINEERING_PROPERTIES_SCHEMA.EXP_FUNCTION' IN
 TYPEOF(arg))) OR ('ENGINEERING_PROPERTIES_SCHEMA.LOG_FUNCTION' IN
 TYPEOF(arg))) OR ('ENGINEERING_PROPERTIES_SCHEMA.LOG2_FUNCTION' IN
 TYPEOF(arg))) OR ('ENGINEERING_PROPERTIES_SCHEMA.LOG10_FUNCTION' IN
 TYPEOF(arg))) OR ('ENGINEERING_PROPERTIES_SCHEMA.SQUARE_ROOT_FUNCTION' IN
 TYPEOF(arg)) THEN
   RETURN (FALSE);
END_IF;
       IF ((('ENGINEERING_PROPERTIES_SCHEMA.PLUS_EXPRESSION' IN
 TYPEOF(arg)) OR ('ENGINEERING_PROPERTIES_SCHEMA.MULT_EXPRESSION' IN
 TYPEOF(arg))) OR ('ENGINEERING_PROPERTIES_SCHEMA.MAXIMUM_FUNCTION' IN
 TYPEOF(arg))) OR ('ENGINEERING_PROPERTIES_SCHEMA.MINIMUM_FUNCTION' IN
 TYPEOF(arg)) THEN
   REPEAT i := 1 TO SIZEOF(arg\multiple_arity_numeric_expression.operands);
             IF NOT
 is_int_expr(arg\multiple_arity_numeric_expression.operands[i]) THEN
         RETURN (FALSE);
      END_IF;
   END_REPEAT;
   RETURN (TRUE);
END_IF;
       IF ('ENGINEERING_PROPERTIES_SCHEMA.MINUS_EXPRESSION' IN TYPEOF(arg))
 OR ('ENGINEERING_PROPERTIES_SCHEMA.POWER_EXPRESSION' IN TYPEOF(arg)) THEN
   RETURN (is_int_expr(arg\binary_numeric_expression.operands[1]) AND
 is_int_expr(arg\binary_numeric_expression.operands[2]));
END_IF;
       IF ('ENGINEERING_PROPERTIES_SCHEMA.DIV_EXPRESSION' IN TYPEOF(arg))
 OR ('ENGINEERING_PROPERTIES_SCHEMA.MOD_EXPRESSION' IN TYPEOF(arg)) THEN
   RETURN (TRUE);
END_IF;
IF 'ENGINEERING_PROPERTIES_SCHEMA.SLASH_EXPRESSION' IN TYPEOF(arg) THEN
   RETURN (FALSE);
END_IF;
IF 'ENGINEERING_PROPERTIES_SCHEMA.LENGTH_FUNCTION' IN TYPEOF(arg) THEN
   RETURN (TRUE);
END_IF;
IF 'ENGINEERING_PROPERTIES_SCHEMA.VALUE_FUNCTION' IN TYPEOF(arg) THEN
```

```
            IF 'ENGINEERING_PROPERTIES_SCHEMA.INT_VALUE_FUNCTION' IN
    TYPEOF(arg) THEN
         RETURN (TRUE);
      ELSE
         RETURN (FALSE);
      END_IF;
   END_IF;
         IF 'ENGINEERING_PROPERTIES_SCHEMA.INTEGER_DEFINED_FUNCTION' IN
    TYPEOF(arg) THEN
      RETURN (TRUE);
   END_IF;
         IF 'ENGINEERING_PROPERTIES_SCHEMA.REAL_DEFINED_FUNCTION' IN
    TYPEOF(arg) THEN
      RETURN (FALSE);
   END_IF;
         IF 'ENGINEERING_PROPERTIES_SCHEMA.BOOLEAN_DEFINED_FUNCTION' IN
    TYPEOF(arg) THEN
      RETURN (FALSE);
   END_IF;
         IF 'ENGINEERING_PROPERTIES_SCHEMA.STRING_DEFINED_FUNCTION' IN
    TYPEOF(arg) THEN
      RETURN (FALSE);
   END_IF;
   RETURN (FALSE);
END_FUNCTION;

FUNCTION item_correlation
   (items : SET OF GENERIC;
     c_items : SET OF STRING ) : LOGICAL;
      LOCAL
         c_types : SET OF STRING := [];
         c_hit : INTEGER := 0;
       END_LOCAL;
     REPEAT i := 1 TO HIINDEX(c_items);
           c_types := c_types + [ 'ENGINEERING_PROPERTIES_SCHEMA.' +
      c_items[i] ];
     END_REPEAT;
     REPEAT i := 1 TO HIINDEX(items);
        IF SIZEOF(c_types * TYPEOF(items[i])) = 1 THEN
           c_hit := c_hit + 1;
        END_IF;
     END_REPEAT;
     IF SIZEOF(items) = c_hit THEN
        RETURN (TRUE);
     ELSE
        RETURN (FALSE);
     END_IF;
END_FUNCTION;

FUNCTION item_in_context
   (item : representation_item;
    cntxt : representation_context ) : BOOLEAN;
LOCAL
   y : BAG OF representation_item;
END_LOCAL;
         IF SIZEOF(USEDIN(item,
    'ENGINEERING_PROPERTIES_SCHEMA.REPRESENTATION.ITEMS') *
    cntxt.representations_in_context) > 0 THEN
      RETURN (TRUE);
```

```
   ELSE
            y := QUERY (z <* USEDIN(item, '')|
    'ENGINEERING_PROPERTIES_SCHEMA.REPRESENTATION_ITEM' IN TYPEOF(z));
      IF SIZEOF(y) > 0 THEN
         REPEAT i := 1 TO HIINDEX(y);
            IF item_in_context(y[i], cntxt) THEN
               RETURN (TRUE);
            END_IF;
         END_REPEAT;
      END_IF;
   END_IF;
   RETURN (FALSE);
END_FUNCTION;

FUNCTION leap_year
   (year : year_number ) : BOOLEAN;
   IF (year MOD 4 = 0) AND (year MOD 100 <> 0) OR (year MOD 400 = 0) THEN
      RETURN (TRUE);
   ELSE
      RETURN (FALSE);
   END_IF;
END_FUNCTION;

FUNCTION list_selected_components
   (aggr : AGGREGATE OF LIST OF maths_value;
    k : positive_integer ) : LIST OF maths_value;
LOCAL
   result : LIST OF maths_value := [];
   j : INTEGER := 0;
END_LOCAL;
   REPEAT i := LOINDEX(aggr) TO HIINDEX(aggr);
      IF k <= SIZEOF(aggr[i]) THEN
         INSERT( result, aggr[i][k], j );
         j := j + 1;
      END_IF;
   END_REPEAT;
   RETURN (result);
END_FUNCTION;

FUNCTION make_atom_based_literal
   (lit_value : atom_based_value ) : atom_based_literal;
          RETURN (atom_based_literal(lit_value) || generic_literal() ||
    simple_generic_expression() || generic_expression());
END_FUNCTION;

FUNCTION make_binary_literal
   (lit_value : BINARY ) : binary_literal;
          RETURN (binary_literal(lit_value) || generic_literal() ||
    simple_generic_expression() || generic_expression());
END_FUNCTION;

FUNCTION make_boolean_literal
   (lit_value : BOOLEAN ) : boolean_literal;
          RETURN (boolean_literal(lit_value) || simple_boolean_expression() ||
    boolean_expression() || expression() || generic_expression() ||
    simple_generic_expression() || generic_literal());
END_FUNCTION;

FUNCTION make_cartesian_complex_number_region
```

```
   (real_constraint : real_interval;
    imag_constraint : real_interval ) : cartesian_complex_number_region;
          RETURN (cartesian_complex_number_region(real_constraint,
    imag_constraint) || maths_space() || generic_expression() ||
    generic_literal() || simple_generic_expression());
END_FUNCTION;

FUNCTION make_complex_number_literal(rpart, ipart : REAL) :
complex_number_literal;
  RETURN (complex_number_literal (rpart, ipart)
 || generic_literal()
   || simple_generic_expression()
     || generic_expression() );
END_FUNCTION;  -- make_complex_number_literal

FUNCTION make_elementary_function
   (func_id : elementary_function_enumerators ) : elementary_function;
          RETURN (elementary_function(func_id) || maths_function() ||
    generic_expression() || generic_literal() || simple_generic_expression());
END_FUNCTION;

FUNCTION make_elementary_space
   (space_id : elementary_space_enumerators ) : elementary_space;
          RETURN (elementary_space(space_id) || maths_space() ||
    generic_expression() || generic_literal() || simple_generic_expression());
END_FUNCTION;

FUNCTION make_extended_tuple_space
   (base : product_space;
    extender : maths_space ) : extended_tuple_space;
          RETURN (extended_tuple_space(base, extender) || maths_space() ||
    generic_expression() || generic_literal() || simple_generic_expression());
END_FUNCTION;

FUNCTION make_finite_integer_interval
   (min : INTEGER;
    max : INTEGER ) : finite_integer_interval;
          RETURN (finite_integer_interval(min, max) || maths_space() ||
    generic_expression() || generic_literal() || simple_generic_expression());
END_FUNCTION;

FUNCTION make_finite_real_interval
   (min : REAL;
    minclo : open_closed;
    max : REAL;
    maxclo : open_closed ) : finite_real_interval;
          RETURN (finite_real_interval(min, minclo, max, maxclo) ||
    maths_space() || generic_expression() || generic_literal() ||
    simple_generic_expression());
END_FUNCTION;

FUNCTION make_finite_space
   (members : SET OF maths_value ) : finite_space;
      RETURN (finite_space(members) || maths_space() || generic_expression()
|| generic_literal() || simple_generic_expression());
END_FUNCTION;

FUNCTION make_function_application
   (afunction : maths_function_select;
```

```
    arguments : LIST [1:?] OF maths_value ) : function_application;
          RETURN (function_application(afunction, arguments) ||
    multiple_arity_generic_expression((convert_to_maths_function(afunction) +
    convert_to_operands(arguments))) || generic_expression());
END_FUNCTION;

FUNCTION make_function_space
   (domain_constraint : space_constraint_type;
    domain_argument : maths_space;
    range_constraint : space_constraint_type;
    range_argument : maths_space ) : function_space;
          RETURN (function_space(domain_constraint, domain_argument,
    range_constraint, range_argument) || maths_space() || generic_expression()
    || generic_literal() || simple_generic_expression());
END_FUNCTION;

FUNCTION make_int_literal
   (lit_value : INTEGER ) : int_literal;
          RETURN (int_literal() || literal_number(lit_value) ||
    simple_numeric_expression() || numeric_expression() || expression() ||
    generic_expression() || simple_generic_expression() || generic_literal());
END_FUNCTION;

FUNCTION make_listed_product_space
   (factors : LIST OF maths_space ) : listed_product_space;
          RETURN (listed_product_space(factors) || maths_space() ||
    generic_expression() || generic_literal() || simple_generic_expression());
END_FUNCTION;

FUNCTION make_logical_literal
   (lit_value : LOGICAL ) : logical_literal;
          RETURN (logical_literal(lit_value) || generic_literal() ||
    simple_generic_expression() || generic_expression());
END_FUNCTION;

FUNCTION make_maths_enum_literal
   (lit_value : maths_enum_atom ) : maths_enum_literal;
          RETURN (maths_enum_literal(lit_value) || generic_literal() ||
    simple_generic_expression() || generic_expression());
END_FUNCTION;

FUNCTION make_maths_tuple_literal
   (lit_value : LIST OF maths_value ) : maths_tuple_literal;
          RETURN (maths_tuple_literal(lit_value) || generic_literal() ||
    simple_generic_expression() || generic_expression());
END_FUNCTION;

FUNCTION make_parallel_composed_function
   (srcdom : maths_space_or_function;
    prepfuncs : LIST [2:?] OF maths_function;
    finfunc : maths_function_select ) : parallel_composed_function;
          RETURN (parallel_composed_function(srcdom, prepfuncs, finfunc) ||
    maths_function() || generic_expression() ||
    multiple_arity_generic_expression(convert_to_operands_prcmfn(srcdom,
    prepfuncs, finfunc)));
END_FUNCTION;

FUNCTION make_polar_complex_number_region
   (centre : complex_number_literal;
```

```
    dis_constraint : real_interval;
    dir_constraint : finite_real_interval ) : polar_complex_number_region;
          RETURN (polar_complex_number_region(centre, dis_constraint,
    dir_constraint) || maths_space() || generic_expression() ||
    generic_literal() || simple_generic_expression());
END_FUNCTION;

FUNCTION make_real_interval_from_min
   (min : REAL;
    minclo : open_closed ) : real_interval_from_min;
          RETURN (real_interval_from_min(min, minclo) || maths_space() ||
    generic_expression() || generic_literal() || simple_generic_expression());
END_FUNCTION;

FUNCTION make_real_interval_to_max
   (max : REAL;
    maxclo : open_closed ) : real_interval_to_max;
          RETURN (real_interval_to_max(max, maxclo) || maths_space() ||
    generic_expression() || generic_literal() || simple_generic_expression());
END_FUNCTION;

FUNCTION make_real_literal
   (lit_value : REAL ) : real_literal;
          RETURN (real_literal() || literal_number(lit_value) ||
    simple_numeric_expression() || numeric_expression() || expression() ||
    generic_expression() || simple_generic_expression() || generic_literal());
END_FUNCTION;

FUNCTION make_string_literal
   (lit_value : STRING ) : string_literal;
          RETURN (string_literal(lit_value) || simple_string_expression() ||
    string_expression() || expression() || generic_expression() ||
    simple_generic_expression() || generic_literal());
END_FUNCTION;

FUNCTION make_uniform_product_space
   (base : maths_space;
    exponent : positive_integer ) : uniform_product_space;
          RETURN (uniform_product_space(base, exponent) || maths_space() ||
    generic_expression() || generic_literal() || simple_generic_expression());
END_FUNCTION;

FUNCTION max_exists
   (spc : maths_space ) : BOOLEAN;
LOCAL
   types : SET OF STRING := TYPEOF(spc);
END_LOCAL;
      RETURN (bool((((schema_prefix + 'FINITE_INTEGER_INTERVAL' IN types) OR
(schema_prefix + 'INTEGER_INTERVAL_TO_MAX' IN types)) OR (schema_prefix +
'FINITE_REAL_INTERVAL' IN types)) OR (schema_prefix + 'REAL_INTERVAL_TO_MAX'
IN types)));
END_FUNCTION;

FUNCTION max_included
   (spc : maths_space ) : BOOLEAN;
LOCAL
   types : SET OF STRING := TYPEOF(spc);
END_LOCAL;
```

```
          IF (schema_prefix + 'FINITE_INTEGER_INTERVAL' IN types) OR
     (schema_prefix + 'INTEGER_INTERVAL_TO_MAX' IN types) THEN
        RETURN (TRUE);
     END_IF;
     IF schema_prefix + 'FINITE_REAL_INTERVAL' IN types THEN
        RETURN (bool(spc\finite_real_interval.max_closure = closed));
     END_IF;
     IF schema_prefix + 'REAL_INTERVAL_TO_MAX' IN types THEN
        RETURN (bool(spc\real_interval_to_max.max_closure = closed));
     END_IF;
     RETURN (FALSE);
END_FUNCTION;

FUNCTION member_of
   (val : GENERIC : G;
    spc : maths_space ) : LOGICAL;
   FUNCTION fedex
      (val : AGGREGATE OF GENERIC : x;

       i : INTEGER ) : GENERIC : x;
      RETURN (val[i]);
   END_FUNCTION;
LOCAL
   v : maths_value := simplify_maths_value(convert_to_maths_value(val));
   vtypes : SET OF STRING := stripped_typeof(v);
   s : maths_space := simplify_maths_space(spc);
   stypes : SET OF STRING := stripped_typeof(s);
   tmp_int : INTEGER;
   tmp_real : REAL;
   tmp_cmplx : complex_number_literal;
   lgcl : LOGICAL;
   cum : LOGICAL;
   vspc : maths_space;
   sspc : maths_space;
   smem : SET OF maths_value;
   factors : LIST OF maths_space;
END_LOCAL;
   IF NOT EXISTS(s) THEN
      RETURN (FALSE);
   END_IF;
   IF NOT EXISTS(v) THEN
      RETURN (s = the_generics);
   END_IF;
       IF ((('GENERIC_EXPRESSION' IN vtypes) AND NOT ('MATHS_SPACE' IN
       vtypes)) AND NOT ('MATHS_FUNCTION' IN vtypes)) AND NOT
       ('COMPLEX_NUMBER_LITERAL' IN vtypes) THEN
      IF has_values_space(v) THEN
         vspc := values_space_of(v);
         IF subspace_of(vspc, s) THEN
            RETURN (TRUE);
         END_IF;
         IF NOT compatible_spaces(vspc, s) THEN
            RETURN (FALSE);
         END_IF;
         RETURN (UNKNOWN);
      END_IF;
      RETURN (UNKNOWN);
   END_IF;
   IF 'ELEMENTARY_SPACE' IN stypes THEN
```

```
      CASE s\elementary_space.space_id OF
         es_numbers :
               RETURN (('NUMBER' IN vtypes) OR ('COMPLEX_NUMBER_LITERAL' IN
          vtypes));
         es_complex_numbers :
               RETURN ('COMPLEX_NUMBER_LITERAL' IN vtypes);
         es_reals :
               RETURN (('REAL' IN vtypes) AND NOT ('INTEGER' IN vtypes));
         es_integers :
               RETURN ('INTEGER' IN vtypes);
         es_logicals :
               RETURN ('LOGICAL' IN vtypes);
         es_booleans :
               RETURN ('BOOLEAN' IN vtypes);
         es_strings :
               RETURN ('STRING' IN vtypes);
         es_binarys :
               RETURN ('BINARY' IN vtypes);
         es_maths_spaces :
               RETURN ('MATHS_SPACE' IN vtypes);
         es_maths_functions :
               RETURN ('MATHS_FUNCTION' IN vtypes);
         es_generics :
               RETURN (TRUE);
      END_CASE;
   END_IF;
   IF 'FINITE_INTEGER_INTERVAL' IN stypes THEN
      IF 'INTEGER' IN vtypes THEN
         tmp_int := v;
                  RETURN ((s\finite_integer_interval.min <= tmp_int) AND
       (tmp_int <= s\finite_integer_interval.max));
      END_IF;
      RETURN (FALSE);
   END_IF;
   IF 'INTEGER_INTERVAL_FROM_MIN' IN stypes THEN
      IF 'INTEGER' IN vtypes THEN
         tmp_int := v;
         RETURN (s\integer_interval_from_min.min <= tmp_int);
      END_IF;
      RETURN (FALSE);
   END_IF;
   IF 'INTEGER_INTERVAL_TO_MAX' IN stypes THEN
      IF 'INTEGER' IN vtypes THEN
         tmp_int := v;
         RETURN (tmp_int <= s\integer_interval_to_max.max);
      END_IF;
      RETURN (FALSE);
   END_IF;
   IF 'FINITE_REAL_INTERVAL' IN stypes THEN
      IF ('REAL' IN vtypes) AND NOT ('INTEGER' IN vtypes) THEN
         tmp_real := v;
         IF s\finite_real_interval.min_closure = closed THEN
            IF s\finite_real_interval.max_closure = closed THEN
            RETURN ((s\finite_real_interval.min <= tmp_real) AND (tmp_real <=
          s\finite_real_interval.max));
            ELSE
            RETURN ((s\finite_real_interval.min <= tmp_real) AND (tmp_real <
          s\finite_real_interval.max));
            END_IF;
```

```
      ELSE
         IF s\finite_real_interval.max_closure = closed THEN
         RETURN ((s\finite_real_interval.min < tmp_real) AND (tmp_real <=
       s\finite_real_interval.max));
         ELSE
           RETURN ((s\finite_real_interval.min < tmp_real) AND (tmp_real <
           s\finite_real_interval.max));
         END_IF;
      END_IF;
   END_IF;
   RETURN (FALSE);
END_IF;
IF 'REAL_INTERVAL_FROM_MIN' IN stypes THEN
   IF ('REAL' IN vtypes) AND NOT ('INTEGER' IN vtypes) THEN
      tmp_real := v;
      IF s\real_interval_from_min.min_closure = closed THEN
         RETURN (s\real_interval_from_min.min <= tmp_real);
      ELSE
         RETURN (s\real_interval_from_min.min < tmp_real);
      END_IF;
   END_IF;
   RETURN (FALSE);
END_IF;
IF 'REAL_INTERVAL_TO_MAX' IN stypes THEN
   IF ('REAL' IN vtypes) AND NOT ('INTEGER' IN vtypes) THEN
      tmp_real := v;
      IF s\real_interval_to_max.max_closure = closed THEN
         RETURN (tmp_real <= s\real_interval_to_max.max);
      ELSE
         RETURN (tmp_real < s\real_interval_to_max.max);
      END_IF;
   END_IF;
   RETURN (FALSE);
END_IF;
IF 'CARTESIAN_COMPLEX_NUMBER_REGION' IN stypes THEN
   IF 'COMPLEX_NUMBER_LITERAL' IN vtypes THEN
    RETURN (member_of(v\complex_number_literal.real_part,
    s\cartesian_complex_number_region.real_constraint) AND
    member_of(v\complex_number_literal.imag_part,
    s\cartesian_complex_number_region.imag_constraint));
   END_IF;
   RETURN (FALSE);
END_IF;
IF 'POLAR_COMPLEX_NUMBER_REGION' IN stypes THEN
   IF 'COMPLEX_NUMBER_LITERAL' IN vtypes THEN
      tmp_cmplx := v;
       tmp_cmplx.real_part := tmp_cmplx.real_part -
       s\polar_complex_number_region.centre.real_part;
       tmp_cmplx.imag_part := tmp_cmplx.imag_part -
       s\polar_complex_number_region.centre.imag_part;
       tmp_real := SQRT(tmp_cmplx.real_part ** 2 + tmp_cmplx.imag_part **
       2);
      IF NOT member_of(tmp_real,
      s\polar_complex_number_region.distance_constraint) THEN
         RETURN (FALSE);
      END_IF;
      IF tmp_real = 0.00000 THEN
         RETURN (TRUE);
      END_IF;
```

```
      tmp_real := atan2(tmp_cmplx.imag_part, tmp_cmplx.real_part);
         RETURN (member_of(tmp_real,
     s\polar_complex_number_region.direction_constraint) OR
     member_of(tmp_real + 2.00000 * 3.14159,
     s\polar_complex_number_region.direction_constraint));
   END_IF;
   RETURN (FALSE);
END_IF;
IF 'FINITE_SPACE' IN stypes THEN
   smem := s\finite_space.members;
   cum := FALSE;
   REPEAT i := 1 TO SIZEOF(smem);
      cum := cum OR equal_maths_values(v, smem[i]);
      IF cum = TRUE THEN
         RETURN (TRUE);
      END_IF;
   END_REPEAT;
   RETURN (cum);
END_IF;
IF 'UNIFORM_PRODUCT_SPACE' IN stypes THEN
   IF 'LIST' IN vtypes THEN
      IF SIZEOF(v) = s\uniform_product_space.exponent THEN
         sspc := s\uniform_product_space.base;
         cum := TRUE;
         REPEAT i := 1 TO SIZEOF(v);
            cum := cum AND member_of(v[i], sspc);
            IF cum = FALSE THEN
               RETURN (FALSE);
            END_IF;
         END_REPEAT;
         RETURN (cum);
      END_IF;
   END_IF;
   RETURN (FALSE);
END_IF;
IF 'LISTED_PRODUCT_SPACE' IN stypes THEN
   IF 'LIST' IN vtypes THEN
      factors := s\listed_product_space.factors;
      IF SIZEOF(v) = SIZEOF(factors) THEN
         cum := TRUE;
         REPEAT i := 1 TO SIZEOF(v);
            cum := cum AND member_of(v[i], factors[i]);
            IF cum = FALSE THEN
               RETURN (FALSE);
            END_IF;
         END_REPEAT;
         RETURN (cum);
      END_IF;
   END_IF;
   RETURN (FALSE);
END_IF;
IF 'EXTENDED_TUPLE_SPACE' IN stypes THEN
   IF 'LIST' IN vtypes THEN
      sspc := s\extended_tuple_space.base;
      tmp_int := space_dimension(sspc);
      IF SIZEOF(v) >= tmp_int THEN
         cum := TRUE;
         REPEAT i := 1 TO tmp_int;
            cum := cum AND member_of(v[i], factor_space(sspc, i));
```

```
                IF cum = FALSE THEN
                   RETURN (FALSE);
                END_IF;
             END_REPEAT;
             sspc := s\extended_tuple_space.extender;
             REPEAT i := tmp_int + 1 TO SIZEOF(v);
                cum := cum AND member_of(v[i], sspc);
                IF cum = FALSE THEN
                   RETURN (FALSE);
                END_IF;
             END_REPEAT;
             RETURN (cum);
          END_IF;
       END_IF;
       RETURN (FALSE);
    END_IF;
    IF 'FUNCTION_SPACE' IN stypes THEN
       IF 'MATHS_FUNCTION' IN vtypes THEN
          vspc := v\maths_function.domain;
          sspc := s\function_space.domain_argument;
          CASE s\function_space.domain_constraint OF
             sc_equal :
                   cum := equal_maths_spaces(vspc, sspc);
             sc_subspace :
                   cum := subspace_of(vspc, sspc);
             sc_member :
                   cum := member_of(vspc, sspc);
          END_CASE;
          IF cum = FALSE THEN
             RETURN (FALSE);
          END_IF;
          vspc := v\maths_function.range;
          sspc := s\function_space.range_argument;
          CASE s\function_space.range_constraint OF
             sc_equal :
                   cum := cum AND equal_maths_spaces(vspc, sspc);
             sc_subspace :
                   cum := cum AND subspace_of(vspc, sspc);

             sc_member :
                   cum := cum AND member_of(vspc, sspc);
          END_CASE;
          RETURN (cum);
       END_IF;
       RETURN (FALSE);
    END_IF;
    RETURN (UNKNOWN);
END_FUNCTION;

FUNCTION min_exists
   (spc : maths_space ) : BOOLEAN;
LOCAL
   types : SET OF STRING := TYPEOF(spc);
END_LOCAL;
          RETURN (bool((((schema_prefix + 'FINITE_INTEGER_INTERVAL' IN types)
    OR (schema_prefix + 'INTEGER_INTERVAL_FROM_MIN' IN types)) OR
    (schema_prefix + 'FINITE_REAL_INTERVAL' IN types)) OR (schema_prefix +
    'REAL_INTERVAL_FROM_MIN' IN types)));
END_FUNCTION;
```

```
FUNCTION min_included
   (spc : maths_space ) : BOOLEAN;
LOCAL
   types : SET OF STRING := TYPEOF(spc);
END_LOCAL;
         IF (schema_prefix + 'FINITE_INTEGER_INTERVAL' IN types) OR
    (schema_prefix + 'INTEGER_INTERVAL_FROM_MIN' IN types) THEN
      RETURN (TRUE);
   END_IF;
   IF schema_prefix + 'FINITE_REAL_INTERVAL' IN types THEN
      RETURN (bool(spc\finite_real_interval.min_closure = closed));
   END_IF;
   IF schema_prefix + 'REAL_INTERVAL_FROM_MIN' IN types THEN
      RETURN (bool(spc\real_interval_from_min.min_closure = closed));
   END_IF;
   RETURN (FALSE);
END_FUNCTION;

FUNCTION no_cyclic_domain_reference
   (ref : maths_space_or_function;
    used : SET OF maths_function ) : BOOLEAN;
LOCAL
   typenames : SET OF STRING := TYPEOF(ref);
   func : maths_function;
END_LOCAL;
   IF NOT EXISTS(ref) OR NOT EXISTS(used) THEN
      RETURN (FALSE);
   END_IF;
   IF schema_prefix + 'MATHS_SPACE' IN typenames THEN
      RETURN (TRUE);
   END_IF;
   func := ref;
   IF func IN used THEN
      RETURN (FALSE);
   END_IF;
   IF schema_prefix + 'CONSTANT_FUNCTION' IN typenames THEN
            RETURN
    (no_cyclic_domain_reference(func\constant_function.source_of_domain, used
    + [ func ]));
   END_IF;
   IF schema_prefix + 'SELECTOR_FUNCTION' IN typenames THEN
            RETURN
    (no_cyclic_domain_reference(func\selector_function.source_of_domain, used
    + [ func ]));
   END_IF;
   IF schema_prefix + 'PARALLEL_COMPOSED_FUNCTION' IN typenames THEN
            RETURN
    (no_cyclic_domain_reference(func\parallel_composed_function.source_of_doma
    in, used + [ func ]));
   END_IF;
   RETURN (TRUE);
END_FUNCTION;

FUNCTION no_cyclic_space_reference
   (spc : maths_space;
    refs : SET OF maths_space ) : BOOLEAN;
LOCAL
   types : SET OF STRING;
```

```
   refs_plus : SET OF maths_space;
END_LOCAL;
   IF spc IN refs THEN
      RETURN (FALSE);
   END_IF;
   types := TYPEOF(spc);
   refs_plus := refs + spc;
   IF schema_prefix + 'FINITE_SPACE' IN types THEN
            RETURN (bool(SIZEOF(QUERY (sp <* QUERY (mem <*
    spc\finite_space.members| (schema_prefix + 'MATHS_SPACE' IN TYPEOF(mem)))|
    NOT no_cyclic_space_reference(sp, refs_plus))) = 0));
   END_IF;
   IF schema_prefix + 'UNIFORM_PRODUCT_SPACE' IN types THEN
            RETURN (no_cyclic_space_reference(spc\uniform_product_space.base,
    refs_plus));
   END_IF;
   IF schema_prefix + 'LISTED_PRODUCT_SPACE' IN types THEN
            RETURN (bool(SIZEOF(QUERY (fac <*
    spc\listed_product_space.factors| NOT no_cyclic_space_reference(fac,
    refs_plus))) = 0));
   END_IF;
   IF schema_prefix + 'EXTENDED_TUPLE_SPACE' IN types THEN
            RETURN (no_cyclic_space_reference(spc\extended_tuple_space.base,
    refs_plus) AND no_cyclic_space_reference(spc\extended_tuple_space.extender,
    refs_plus));
   END_IF;
   RETURN (TRUE);
END_FUNCTION;

FUNCTION nondecreasing
   (lr : LIST OF REAL ) : BOOLEAN;
   IF NOT EXISTS(lr) THEN
      RETURN (FALSE);
   END_IF;
   REPEAT j := 2 TO SIZEOF(lr);
      IF lr[j] < lr[(j - 1)] THEN
         RETURN (FALSE);
      END_IF;
   END_REPEAT;
   RETURN (TRUE);
END_FUNCTION;

FUNCTION normalise
   (arg : vector_or_direction ) : vector_or_direction;
LOCAL
   ndim : INTEGER;
   v : direction;
   result : vector_or_direction;
   vec : vector;
   mag : REAL;
END_LOCAL;
   IF NOT EXISTS(arg) THEN
      result := ?;
   ELSE
      ndim := arg.dim;
      IF 'ENGINEERING_PROPERTIES_SCHEMA.VECTOR' IN TYPEOF(arg) THEN
         BEGIN
            v := dummy_gri || direction(arg.orientation.direction_ratios);
            IF arg.magnitude = 0.00000 THEN
```

```
               RETURN (?);
            ELSE
               vec := dummy_gri || vector(v, 1.00000);
            END_IF;
         END;
      ELSE
         v := dummy_gri || direction(arg.direction_ratios);
      END_IF;
      mag := 0.00000;
      REPEAT i := 1 TO ndim;
         mag := mag + v.direction_ratios[i] * v.direction_ratios[i];
      END_REPEAT;
      IF mag > 0.00000 THEN
         mag := SQRT(mag);
         REPEAT i := 1 TO ndim;
            v.direction_ratios[i] := v.direction_ratios[i] / mag;
         END_REPEAT;
         IF 'ENGINEERING_PROPERTIES_SCHEMA.VECTOR' IN TYPEOF(arg) THEN
            vec.orientation := v;
            result := vec;
         ELSE
            result := v;
         END_IF;
      ELSE
         RETURN (?);
      END_IF;
   END_IF;
   RETURN (result);
END_FUNCTION;

FUNCTION number_superspace_of
   (spc : maths_space ) : elementary_space;
   IF subspace_of_es(spc, es_integers) THEN
      RETURN (the_integers);
   END_IF;
   IF subspace_of_es(spc, es_reals) THEN
      RETURN (the_reals);
   END_IF;
   IF subspace_of_es(spc, es_complex_numbers) THEN
      RETURN (the_complex_numbers);
   END_IF;
   IF subspace_of_es(spc, es_numbers) THEN
      RETURN (the_numbers);
   END_IF;
   RETURN (?);
END_FUNCTION;

FUNCTION number_tuple_subspace_check
   (spc : maths_space ) : LOGICAL;
LOCAL
   types : SET OF STRING := stripped_typeof(spc);
   factors : LIST OF maths_space;
   cum : LOGICAL := TRUE;
END_LOCAL;
   IF 'UNIFORM_PRODUCT_SPACE' IN types THEN
      RETURN (subspace_of_es(spc\uniform_product_space.base, es_numbers));
   END_IF;
   IF 'LISTED_PRODUCT_SPACE' IN types THEN
      factors := spc\listed_product_space.factors;
```

```
      REPEAT i := 1 TO SIZEOF(factors);
         cum := cum AND subspace_of_es(factors[i], es_numbers);
      END_REPEAT;
      RETURN (cum);
   END_IF;
   IF 'EXTENDED_TUPLE_SPACE' IN types THEN
      cum := subspace_of_es(spc\extended_tuple_space.extender, es_numbers);
       cum := cum AND
       number_tuple_subspace_check(spc\extended_tuple_space.base);
      RETURN (cum);
   END_IF;
   RETURN (FALSE);
END_FUNCTION;

FUNCTION one_tuples_of
   (spc : maths_space ) : tuple_space;
   RETURN (make_uniform_product_space(spc, 1));
END_FUNCTION;

FUNCTION orthogonal_complement
   (vec : direction ) : direction;
LOCAL
   result : direction;
END_LOCAL;
   IF (vec.dim <> 2) OR NOT EXISTS(vec) THEN
      RETURN (?);
   ELSE
        result := dummy_gri || direction([ -vec.direction_ratios[2],
    vec.direction_ratios[1] ]);
      RETURN (result);
   END_IF;
END_FUNCTION;

FUNCTION parallel_composed_function_composability_check
   (funcs : LIST OF maths_function;
    final : maths_function_select ) : BOOLEAN;
LOCAL
   tplsp : tuple_space := the_zero_tuple_space;
   finfun : maths_function := convert_to_maths_function(final);
END_LOCAL;
   REPEAT i := 1 TO SIZEOF(funcs);
      tplsp := assoc_product_space(tplsp, funcs[i].range);
   END_REPEAT;
   RETURN (compatible_spaces(tplsp, finfun.domain));
END_FUNCTION;

FUNCTION parallel_composed_function_domain_check
   (comdom : tuple_space;
    funcs : LIST OF maths_function ) : BOOLEAN;
   REPEAT i := 1 TO SIZEOF(funcs);
      IF NOT compatible_spaces(comdom, funcs[i].domain) THEN
         RETURN (FALSE);
      END_IF;
   END_REPEAT;
   RETURN (TRUE);
END_FUNCTION;

FUNCTION parse_express_identifier
   (s : STRING;
```

```
    i : positive_integer ) : positive_integer;
LOCAL
   k : positive_integer;
END_LOCAL;
   k := i;
   IF i <= LENGTH(s) THEN
      IF s[i] LIKE '@' THEN
         REPEAT UNTIL (k > LENGTH(s)) OR ((s[k] <> '_') AND NOT (s[k] LIKE
       '@')) AND NOT (s[k] LIKE '#');
            k := k + 1;
         END_REPEAT;
      END_IF;
   END_IF;
   RETURN (k);
END_FUNCTION;

FUNCTION partial_derivative_check
   (domain : tuple_space;
    d_vars : LIST [1:?] OF input_selector ) : BOOLEAN;
LOCAL
   domn : tuple_space := domain;
   fspc : maths_space;
   dim : INTEGER;
   k : INTEGER;
END_LOCAL;
         IF (space_dimension(domain) = 1) AND (schema_prefix + 'TUPLE_SPACE'
    IN TYPEOF(factor1(domain))) THEN
      domn := factor1(domain);
   END_IF;
   dim := space_dimension(domn);
   REPEAT i := 1 TO SIZEOF(d_vars);
      k := d_vars[i];
      IF k > dim THEN
         RETURN (FALSE);
      END_IF;
      fspc := factor_space(domn, k);
                IF NOT subspace_of_es(fspc, es_reals) AND NOT
       subspace_of_es(fspc, es_complex_numbers) THEN
         RETURN (FALSE);
      END_IF;
   END_REPEAT;
   RETURN (TRUE);
END_FUNCTION;

FUNCTION real_max
   (spc : maths_space ) : REAL;
LOCAL
   types : SET OF STRING := TYPEOF(spc);
END_LOCAL;
   IF schema_prefix + 'FINITE_INTEGER_INTERVAL' IN types THEN
      RETURN (spc\finite_integer_interval.max);
   END_IF;
   IF schema_prefix + 'INTEGER_INTERVAL_TO_MAX' IN types THEN
      RETURN (spc\integer_interval_to_max.max);
   END_IF;
   IF schema_prefix + 'FINITE_REAL_INTERVAL' IN types THEN
      RETURN (spc\finite_real_interval.max);
   END_IF;
   IF schema_prefix + 'REAL_INTERVAL_TO_MAX' IN types THEN
```

```
      RETURN (spc\real_interval_to_max.max);
   END_IF;
   RETURN (?);
END_FUNCTION;

FUNCTION real_min
   (spc : maths_space ) : REAL;
LOCAL
   types : SET OF STRING := TYPEOF(spc);
END_LOCAL;
   IF schema_prefix + 'FINITE_INTEGER_INTERVAL' IN types THEN
      RETURN (spc\finite_integer_interval.min);
   END_IF;
   IF schema_prefix + 'INTEGER_INTERVAL_FROM_MIN' IN types THEN
      RETURN (spc\integer_interval_from_min.min);
   END_IF;
   IF schema_prefix + 'FINITE_REAL_INTERVAL' IN types THEN
      RETURN (spc\finite_real_interval.min);
   END_IF;
   IF schema_prefix + 'REAL_INTERVAL_FROM_MIN' IN types THEN
      RETURN (spc\real_interval_from_min.min);
   END_IF;
   RETURN (?);
END_FUNCTION;

FUNCTION regular_indexing
   (sub : LIST OF INTEGER;
    base : zero_or_one;
    shape : LIST [1:?] OF positive_integer;
    inc : LIST [1:?] OF INTEGER;
    first : INTEGER ) : INTEGER;
LOCAL
   k : INTEGER;
   index : INTEGER;
END_LOCAL;
         IF (((NOT EXISTS(sub) OR NOT EXISTS(base)) OR NOT EXISTS(shape)) OR
    NOT EXISTS(inc)) OR NOT EXISTS(first) THEN
      RETURN (?);
   END_IF;
   IF (SIZEOF(sub) <> SIZEOF(inc)) OR (SIZEOF(sub) <> SIZEOF(shape)) THEN
      RETURN (?);
   END_IF;
   index := first;
   REPEAT j := 1 TO SIZEOF(sub);
      IF NOT EXISTS(sub[j]) OR NOT EXISTS(inc[j]) THEN
         RETURN (?);
      END_IF;
      k := sub[j] - base;
      IF NOT ((0 <= k) AND (k < shape[j])) THEN
         RETURN (?);
      END_IF;
      index := index + k * inc[j];
   END_REPEAT;
   RETURN (index);
END_FUNCTION;

FUNCTION remove_first
   (alist : LIST OF GENERIC : GEN ) : LIST OF GENERIC : GEN;
LOCAL
```

```
      blist : LIST OF GENERIC : GEN := alist;
   END_LOCAL;
      IF SIZEOF(blist) > 0 THEN
         REMOVE( blist, 1 );
      END_IF;
      RETURN (blist);
   END_FUNCTION;

   FUNCTION repackage
      (tspace : tuple_space;
       repckg : repackage_options ) : tuple_space;
      CASE repckg OF
         ro_nochange :
               RETURN (tspace);
         ro_wrap_as_tuple :
               RETURN (one_tuples_of(tspace));
         ro_unwrap_tuple :
               RETURN (factor1(tspace));
      OTHERWISE :
            RETURN (?);
      END_CASE;
   END_FUNCTION;

   FUNCTION scalar_times_vector
      (scalar : REAL;
       vec : vector_or_direction ) : vector;
   LOCAL
      v : direction;
      mag : REAL;
      result : vector;
   END_LOCAL;
      IF NOT EXISTS(scalar) OR NOT EXISTS(vec) THEN
         RETURN (?);
      ELSE
         IF 'ENGINEERING_PROPERTIES_SCHEMA.VECTOR' IN TYPEOF(vec) THEN
            v := dummy_gri || direction(vec.orientation.direction_ratios);
            mag := scalar * vec.magnitude;
         ELSE
            v := dummy_gri || direction(vec.direction_ratios);
            mag := scalar;
         END_IF;
         IF mag < 0.00000 THEN
            REPEAT i := 1 TO SIZEOF(v.direction_ratios);
               v.direction_ratios[i] := -v.direction_ratios[i];
            END_REPEAT;
            mag := -mag;
         END_IF;
         result := dummy_gri || vector(normalise(v), mag);
      END_IF;
      RETURN (result);
   END_FUNCTION;

   FUNCTION second_proj_axis
      (z_axis : direction;
       x_axis : direction;
       arg : direction ) : direction;
   LOCAL
      y_axis : vector;
      v : direction;
```

```
   temp : vector;
END_LOCAL;
   IF NOT EXISTS(arg) THEN
      v := dummy_gri || direction([ 0.00000, 1.00000, 0.00000 ]);
   ELSE
      v := arg;
   END_IF;
   temp := scalar_times_vector(dot_product(v, z_axis), z_axis);
   y_axis := vector_difference(v, temp);
   temp := scalar_times_vector(dot_product(v, x_axis), x_axis);
   y_axis := vector_difference(y_axis, temp);
   y_axis := normalise(y_axis);
   RETURN (y_axis.orientation);
END_FUNCTION;

FUNCTION shape_of_array
   (func : maths_function ) : LIST OF positive_integer;
LOCAL
   tspace : tuple_space;
   temp : maths_space;
   result : LIST OF positive_integer := [];
END_LOCAL;
   IF schema_prefix + 'EXPLICIT_TABLE_FUNCTION' IN TYPEOF(func) THEN
      RETURN (func\explicit_table_function.shape);
   END_IF;
   tspace := func.domain;
         IF (space_dimension(tspace) = 1) AND (schema_prefix + 'TUPLE_SPACE'
    IN TYPEOF(factor1(tspace))) THEN
      tspace := factor1(tspace);
   END_IF;
   REPEAT i := 1 TO space_dimension(tspace);
      temp := factor_space(tspace, i);
      IF NOT (schema_prefix + 'FINITE_INTEGER_INTERVAL' IN TYPEOF(temp)) THEN
         RETURN (?);
      END_IF;
      INSERT( result, temp\finite_integer_interval.size, i - 1 );
   END_REPEAT;
   RETURN (result);
END_FUNCTION;

FUNCTION simplify_function_application(expr : function_application) :
maths_value;
FUNCTION ctmv(x : GENERIC:G) : maths_value;
  RETURN (convert_to_maths_value(x));
END_FUNCTION;  -- local abbreviation for convert_to_maths_value function
PROCEDURE parts(       c : complex_number_literal;
                VAR x, y : REAL);
  x := c.real_part;  y := c.imag_part;
END_PROCEDURE;  -- parts
FUNCTION makec(x, y : REAL) : complex_number_literal;
  RETURN (make_complex_number_literal(x,y));
END_FUNCTION;  -- local abbreviation for make_complex_number_literal function
FUNCTION good_t(v  : maths_value;
                tn : STRING) : BOOLEAN;
  LOCAL
    tpl : LIST OF maths_value;
  END_LOCAL;
  IF 'LIST' IN TYPEOF (v) THEN
    tpl := v;
```

```
    REPEAT i := 1 TO SIZEOF (tpl);
      IF NOT (tn IN TYPEOF (tpl[i])) THEN  RETURN (FALSE);  END_IF;
    END_REPEAT;
    RETURN (TRUE);
  END_IF;
  RETURN (FALSE);
END_FUNCTION;  -- good_t
CONSTANT
  cnlit : STRING := schema_prefix + 'COMPLEX_NUMBER_LITERAL';
END_CONSTANT;
LOCAL
  types : SET OF STRING := stripped_typeof(expr.func);
  ef_val : elementary_function_enumerators;
  is_elementary : BOOLEAN := FALSE;
  v, v1, v2, v3 : maths_value;
  vlist : LIST OF maths_value := [];
  gexpr : generic_expression;
  pairs : SET [1:?] OF LIST [2:2] OF maths_value;
  boo : BOOLEAN;
  lgc, cum : LOGICAL;
  j, k, n : INTEGER;
  p, q, r, s, t, u : REAL;
  str, st2 : STRING;
  bin, bi2 : BINARY;
  tpl, tp2 : LIST OF maths_value;
  mem :SET OF maths_value := [];
END_LOCAL;
REPEAT i := 1 TO SIZEOF (expr.arguments);
  v := simplify_maths_value(expr.arguments[i]);
  INSERT (vlist, v, i-1);
END_REPEAT;
IF SIZEOF (vlist) >= 1 THEN  v1 := vlist[1];  END_IF;
IF SIZEOF (vlist) >= 2 THEN  v2 := vlist[2];  END_IF;
IF SIZEOF (vlist) >= 3 THEN  v3 := vlist[3];  END_IF;
IF 'ELEMENTARY_FUNCTION_ENUMERATORS' IN types THEN
  ef_val := expr.func;
  is_elementary := TRUE;
END_IF;
IF 'ELEMENTARY_FUNCTION' IN types THEN
  ef_val := expr.func\elementary_function.func_id;
  is_elementary := TRUE;
END_IF;
IF is_elementary THEN
  CASE ef_val OF
  ef_and : BEGIN
    cum := TRUE;
    REPEAT i := SIZEOF (vlist) TO 1 BY -1;
      IF 'LOGICAL' IN TYPEOF (vlist[i]) THEN
        lgc := vlist[i];  cum := cum AND lgc;
        IF lgc = FALSE THEN  RETURN (ctmv(FALSE));  END_IF;
        REMOVE (vlist, i);
      END_IF;
    END_REPEAT;
    IF SIZEOF (vlist) = 0 THEN  RETURN (ctmv(cum));  END_IF;
    IF cum <> TRUE THEN  INSERT (vlist, ctmv(cum), 0);  END_IF;
    IF SIZEOF (vlist) = 1 THEN  RETURN (vlist[1]);  END_IF;
    END;
  ef_or : BEGIN
    cum := FALSE;
```

```
  REPEAT i := SIZEOF (vlist) TO 1 BY -1;
    IF 'LOGICAL' IN TYPEOF (vlist[i]) THEN
      lgc := vlist[i];  cum := cum OR lgc;
      IF lgc = TRUE THEN  RETURN (ctmv(TRUE));  END_IF;
      REMOVE (vlist, i);
    END_IF;
  END_REPEAT;
  IF SIZEOF (vlist) = 0 THEN  RETURN (ctmv(cum));  END_IF;
  IF cum <> FALSE THEN  INSERT (vlist, ctmv(cum), 0);  END_IF;
  IF SIZEOF (vlist) = 1 THEN  RETURN (vlist[1]);  END_IF;
  END;
ef_not :
   IF 'LOGICAL' IN TYPEOF (v1) THEN  lgc := v1;  RETURN (ctmv(NOT lgc));
   END_IF;
ef_xor : BEGIN
  IF 'LOGICAL' IN TYPEOF (v1) THEN
    lgc := v1;
    IF 'LOGICAL' IN TYPEOF (v2) THEN  cum := v2;  RETURN (ctmv(lgc XOR cum));
    ELSE IF lgc = FALSE THEN  RETURN (ctmv(v2));
    ELSE IF lgc = UNKNOWN THEN  RETURN (ctmv(UNKNOWN));
    ELSE  RETURN (make_function_application(ef_not,[v2]));
    END_IF;  END_IF;  END_IF;
  ELSE IF 'LOGICAL' IN TYPEOF (v2) THEN
    lgc := v2;
    IF lgc = FALSE THEN  RETURN (ctmv(v1));
    ELSE IF lgc = UNKNOWN THEN  RETURN (ctmv(UNKNOWN));
    ELSE  RETURN (make_function_application(ef_not,[v1]));
    END_IF;  END_IF;
  END_IF;  END_IF;
  END;
ef_negate_i :
  IF 'INTEGER' IN TYPEOF (v1) THEN  j := v1;  RETURN (ctmv(-j));  END_IF;
ef_add_i : BEGIN
  j := 0;
  REPEAT i := SIZEOF (vlist) TO 1 BY -1;
    IF 'INTEGER' IN TYPEOF (vlist[i]) THEN
      k := vlist[i];  j := j + k;
      REMOVE (vlist, i);
    END_IF;
  END_REPEAT;
  IF SIZEOF (vlist) = 0 THEN  RETURN (ctmv(j));  END_IF;
  IF j <> 0 THEN  INSERT (vlist, ctmv(j), 0);  END_IF;
  IF SIZEOF (vlist) = 1 THEN  RETURN (vlist[1]);  END_IF;
  END;
ef_subtract_i :
  IF ('INTEGER' IN TYPEOF (v1)) AND ('INTEGER' IN TYPEOF (v2)) THEN
    j := v1;  k := v2;  RETURN (ctmv(j - k));
  END_IF;
ef_multiply_i : BEGIN
  j := 1;
  REPEAT i := SIZEOF (vlist) TO 1 BY -1;
    IF 'INTEGER' IN TYPEOF (vlist[i]) THEN
      k := vlist[i];  j := j * k;
      REMOVE (vlist, i);
    END_IF;
  END_REPEAT;
  IF SIZEOF (vlist) = 0 THEN  RETURN (ctmv(j));  END_IF;
  IF j <> 1 THEN  INSERT (vlist, ctmv(j), 0);  END_IF;
  IF SIZEOF (vlist) = 1 THEN  RETURN (vlist[1]);  END_IF;
```

```
  END;
ef_divide_i :
  IF ('INTEGER' IN TYPEOF (v1)) AND ('INTEGER' IN TYPEOF (v2)) THEN
    j := v1;  k := v2;  RETURN (ctmv(j DIV k));
  END_IF;
ef_mod_i :
  IF ('INTEGER' IN TYPEOF (v1)) AND ('INTEGER' IN TYPEOF (v2)) THEN
    j := v1;  k := v2;  RETURN (ctmv(j MOD k));
  END_IF;
ef_exponentiate_i :
  IF ('INTEGER' IN TYPEOF (v1)) AND ('INTEGER' IN TYPEOF (v2)) THEN
    j := v1;  k := v2;  n := 1;
    REPEAT i := 1 TO ABS(k);  n := n * j;  END_REPEAT;
    IF k < 0 THEN  n := 1 DIV n;  END_IF;
    RETURN (ctmv(n));
  END_IF;
ef_eq_i :
  IF ('INTEGER' IN TYPEOF (v1)) AND ('INTEGER' IN TYPEOF (v2)) THEN
    j := v1;  k := v2;  RETURN (ctmv(j = k));
  END_IF;
ef_ne_i :
  IF ('INTEGER' IN TYPEOF (v1)) AND ('INTEGER' IN TYPEOF (v2)) THEN
    j := v1;  k := v2;  RETURN (ctmv(j <> k));
  END_IF;
ef_gt_i :
  IF ('INTEGER' IN TYPEOF (v1)) AND ('INTEGER' IN TYPEOF (v2)) THEN
    j := v1;  k := v2;  RETURN (ctmv(j > k));
  END_IF;
ef_lt_i :
  IF ('INTEGER' IN TYPEOF (v1)) AND ('INTEGER' IN TYPEOF (v2)) THEN
    j := v1;  k := v2;  RETURN (ctmv(j < k));
  END_IF;
ef_ge_i :
  IF ('INTEGER' IN TYPEOF (v1)) AND ('INTEGER' IN TYPEOF (v2)) THEN
    j := v1;  k := v2;  RETURN (ctmv(j >= k));
  END_IF;
ef_le_i :
  IF ('INTEGER' IN TYPEOF (v1)) AND ('INTEGER' IN TYPEOF (v2)) THEN
    j := v1;  k := v2;  RETURN (ctmv(j <= k));
  END_IF;
ef_abs_i :
  IF 'INTEGER' IN TYPEOF (v1) THEN  j := v1;  RETURN (ctmv(ABS(j)));  END_IF;
ef_max_i : BEGIN
  boo := FALSE;
  REPEAT i := SIZEOF (vlist) TO 1 BY -1;
    IF 'INTEGER' IN TYPEOF (vlist[i]) THEN
      IF boo THEN  k := vlist[i];  IF k > j THEN  j := k;  END_IF;
      ELSE  j := vlist[i];  boo := TRUE;  END_IF;
      REMOVE (vlist, i);
    END_IF;
  END_REPEAT;
  IF SIZEOF (vlist) = 0 THEN  RETURN (ctmv(j));  END_IF;
  IF boo THEN  INSERT (vlist, ctmv(j), 0);  END_IF;
  IF SIZEOF (vlist) = 1 THEN  RETURN (vlist[1]);  END_IF;
  END;
ef_min_i : BEGIN
  boo := FALSE;
  REPEAT i := SIZEOF (vlist) TO 1 BY -1;
    IF 'INTEGER' IN TYPEOF (vlist[i]) THEN
```

```
      IF boo THEN  k := vlist[i];  IF k < j THEN  j := k;  END_IF;
      ELSE  j := vlist[i];  boo := TRUE;  END_IF;
      REMOVE (vlist, i);
    END_IF;
  END_REPEAT;
  IF SIZEOF (vlist) = 0 THEN  RETURN (ctmv(j));  END_IF;
  IF boo THEN  INSERT (vlist, ctmv(j), 0);  END_IF;
  IF SIZEOF (vlist) = 1 THEN  RETURN (vlist[1]);  END_IF;
  END;
-- ef_if_i : combined with ef_if
ef_negate_r :
  IF 'REAL' IN TYPEOF (v1) THEN  r := v1;  RETURN (ctmv(-r));  END_IF;
ef_reciprocal_r :
  IF 'REAL' IN TYPEOF (v1) THEN  r := v1;  RETURN (ctmv(1.0/r));  END_IF;
ef_add_r : BEGIN
  r := 0.0;
  REPEAT i := SIZEOF (vlist) TO 1 BY -1;
    IF 'REAL' IN TYPEOF (vlist[i]) THEN
      s := vlist[i];  r := r + s;
      REMOVE (vlist, i);
    END_IF;
  END_REPEAT;
  IF SIZEOF (vlist) = 0 THEN  RETURN (ctmv(r));  END_IF;
  IF r <> 0.0 THEN  INSERT (vlist, ctmv(r), 0);  END_IF;
  IF SIZEOF (vlist) = 1 THEN  RETURN (vlist[1]);  END_IF;
  END;
ef_subtract_r :
  IF ('REAL' IN TYPEOF (v1)) AND ('REAL' IN TYPEOF (v2)) THEN
    r := v1;  s := v2;  RETURN (ctmv(r - s));
  END_IF;
ef_multiply_r : BEGIN
  r := 1.0;
  REPEAT i := SIZEOF (vlist) TO 1 BY -1;
    IF 'REAL' IN TYPEOF (vlist[i]) THEN
      s := vlist[i];  r := r * s;
      REMOVE (vlist, i);
    END_IF;
  END_REPEAT;
  IF SIZEOF (vlist) = 0 THEN  RETURN (ctmv(r));  END_IF;
  IF r <> 1.0 THEN  INSERT (vlist, ctmv(r), 0);  END_IF;
  IF SIZEOF (vlist) = 1 THEN  RETURN (vlist[1]);  END_IF;
  END;
ef_divide_r :
  IF ('REAL' IN TYPEOF (v1)) AND ('REAL' IN TYPEOF (v2)) THEN
    r := v1;  s := v2;  RETURN (ctmv(r / s));
  END_IF;
ef_mod_r :
  IF ('REAL' IN TYPEOF (v1)) AND ('REAL' IN TYPEOF (v2)) THEN

    r := v1;  s := v2;  t := r/s;  j := t DIV 1;
    IF (t < 0.0) AND (j <> t) THEN  j := j - 1;  END_IF;
    RETURN (ctmv(r - j * s));
  END_IF;
ef_exponentiate_r :
  IF ('REAL' IN TYPEOF (v1)) AND ('REAL' IN TYPEOF (v2)) THEN
    r := v1;  s := v2;  RETURN (ctmv(r ** s));
  END_IF;
ef_exponentiate_ri :
  IF ('REAL' IN TYPEOF (v1)) AND ('INTEGER' IN TYPEOF (v2)) THEN
```

```
    r := v1;  k := v2;  t := 1.0;
    REPEAT i := 1 TO ABS(k);  t := t * r;  END_REPEAT;
    IF k < 0 THEN  t := 1.0/t;  END_IF;
    RETURN (ctmv(t));
  END_IF;
ef_eq_r :
  IF ('REAL' IN TYPEOF (v1)) AND ('REAL' IN TYPEOF (v2)) THEN
    r := v1;  s := v2;  RETURN (ctmv(r = s));
  END_IF;
ef_ne_r :
  IF ('REAL' IN TYPEOF (v1)) AND ('REAL' IN TYPEOF (v2)) THEN
    r := v1;  s := v2;  RETURN (ctmv(r <> s));
  END_IF;
ef_gt_r :
  IF ('REAL' IN TYPEOF (v1)) AND ('REAL' IN TYPEOF (v2)) THEN
    r := v1;  s := v2;  RETURN (ctmv(r > s));
  END_IF;
ef_lt_r :
  IF ('REAL' IN TYPEOF (v1)) AND ('REAL' IN TYPEOF (v2)) THEN
    r := v1;  s := v2;  RETURN (ctmv(r < s));
  END_IF;
ef_ge_r :
  IF ('REAL' IN TYPEOF (v1)) AND ('REAL' IN TYPEOF (v2)) THEN
    r := v1;  s := v2;  RETURN (ctmv(r >= s));
  END_IF;
ef_le_r :
  IF ('REAL' IN TYPEOF (v1)) AND ('REAL' IN TYPEOF (v2)) THEN
    r := v1;  s := v2;  RETURN (ctmv(r <= s));
  END_IF;
ef_abs_r :
  IF 'REAL' IN TYPEOF (v1) THEN  r := v1;  RETURN (ctmv(ABS(r)));  END_IF;
ef_max_r : BEGIN
  boo := FALSE;
  REPEAT i := SIZEOF (vlist) TO 1 BY -1;
    IF 'REAL' IN TYPEOF (vlist[i]) THEN
      IF boo THEN  s := vlist[i];  IF s > r THEN  r := s;  END_IF;
      ELSE  r := vlist[i];  boo := TRUE;  END_IF;
      REMOVE (vlist, i);
    END_IF;
  END_REPEAT;
  IF SIZEOF (vlist) = 0 THEN  RETURN (ctmv(r));  END_IF;
  IF boo THEN  INSERT (vlist, ctmv(r), 0);  END_IF;
  IF SIZEOF (vlist) = 1 THEN  RETURN (vlist[1]);  END_IF;
  END;
ef_min_r : BEGIN
  boo := FALSE;
  REPEAT i := SIZEOF (vlist) TO 1 BY -1;
    IF 'REAL' IN TYPEOF (vlist[i]) THEN
      IF boo THEN  s := vlist[i];  IF s < r THEN  r := s;  END_IF;
      ELSE  r := vlist[i];  boo := TRUE;  END_IF;
      REMOVE (vlist, i);
    END_IF;
  END_REPEAT;
  IF SIZEOF (vlist) = 0 THEN  RETURN (ctmv(r));  END_IF;
  IF boo THEN  INSERT (vlist, ctmv(r), 0);  END_IF;
  IF SIZEOF (vlist) = 1 THEN  RETURN (vlist[1]);  END_IF;
  END;
ef_acos_r :
  IF 'REAL' IN TYPEOF (v1) THEN  r := v1;  RETURN (ctmv(ACOS(r)));  END_IF;
```

```
ef_asin_r :
  IF 'REAL' IN TYPEOF (v1) THEN  r := v1;  RETURN (ctmv(ASIN(r)));  END_IF;
ef_atan2_r :
  IF ('REAL' IN TYPEOF (v1)) AND ('REAL' IN TYPEOF (v2)) THEN
    r := v1;  s := v2;  RETURN (ctmv(atan2(r,s)));
  END_IF;
ef_cos_r :
  IF 'REAL' IN TYPEOF (v1) THEN  r := v1;  RETURN (ctmv(COS(r)));  END_IF;
ef_exp_r :
  IF 'REAL' IN TYPEOF (v1) THEN  r := v1;  RETURN (ctmv(EXP(r)));  END_IF;
ef_ln_r :
  IF 'REAL' IN TYPEOF (v1) THEN  r := v1;  RETURN (ctmv(LOG(r)));  END_IF;
ef_log2_r :
  IF 'REAL' IN TYPEOF (v1) THEN  r := v1;  RETURN (ctmv(LOG2(r)));  END_IF;
ef_log10_r :
  IF 'REAL' IN TYPEOF (v1) THEN  r := v1;  RETURN (ctmv(LOG10(r)));  END_IF;
ef_sin_r :
  IF 'REAL' IN TYPEOF (v1) THEN  r := v1;  RETURN (ctmv(SIN(r)));  END_IF;
ef_sqrt_r :
  IF 'REAL' IN TYPEOF (v1) THEN  r := v1;  RETURN (ctmv(SQRT(r)));  END_IF;
ef_tan_r :
  IF 'REAL' IN TYPEOF (v1) THEN  r := v1;  RETURN (ctmv(TAN(r)));  END_IF;
-- ef_if_r : combined with ef_if
ef_form_c :
  IF ('REAL' IN TYPEOF (v1)) AND ('REAL' IN TYPEOF (v2)) THEN
    r := v1;  s := v2;  RETURN (makec(r,s));
  END_IF;
ef_rpart_c :
  IF cnlit IN TYPEOF (v1) THEN
    RETURN (ctmv(v1\complex_number_literal.real_part));
  END_IF;
ef_ipart_c :
  IF cnlit IN TYPEOF (v1) THEN
    RETURN (ctmv(v1\complex_number_literal.imag_part));
  END_IF;
ef_negate_c :
   IF cnlit IN TYPEOF (v1) THEN  parts(v1,p,q);  RETURN (makec(-p,-q));
   END_IF;
ef_reciprocal_c :
  IF cnlit IN TYPEOF (v1) THEN
    parts(v1,p,q);  t := p*p + q*q;  RETURN (makec(p/t,-q/t));
  END_IF;
ef_add_c : BEGIN
  p := 0.0;  q := 0.0;
  REPEAT i := SIZEOF (vlist) TO 1 BY -1;
    IF cnlit IN TYPEOF (vlist[i]) THEN
      parts(vlist[i],r,s);  p := p + r;  q := q + s;
      REMOVE (vlist, i);
    END_IF;
  END_REPEAT;
  IF SIZEOF (vlist) = 0 THEN  RETURN (makec(p,q));  END_IF;
  IF p*p+q*q <> 0.0 THEN  INSERT (vlist, makec(p,q), 0);  END_IF;
  IF SIZEOF (vlist) = 1 THEN  RETURN (vlist[1]);  END_IF;
  END;
ef_subtract_c :
  IF (cnlit IN TYPEOF (v1)) AND (cnlit IN TYPEOF (v2)) THEN
    parts(v1,p,q);  parts(v2,r,s);  RETURN (makec(p-r,q-s));
  END_IF;
ef_multiply_c : BEGIN
```

```
    p := 1.0;  q := 0.0;
    REPEAT i := SIZEOF (vlist) TO 1 BY -1;
      IF cnlit IN TYPEOF (vlist[i]) THEN
        parts(vlist[i],r,s);  p := p*r-q*s;  q := p*s+q*r;
        REMOVE (vlist, i);
      END_IF;
    END_REPEAT;
    IF SIZEOF (vlist) = 0 THEN  RETURN (makec(p,q));  END_IF;
    IF (p <> 1.0) OR (q <> 0.0) THEN  INSERT (vlist, makec(p,q), 0);  END_IF;
    IF SIZEOF (vlist) = 1 THEN  RETURN (vlist[1]);  END_IF;
    END;
  ef_divide_c :
    IF (cnlit IN TYPEOF (v1)) AND (cnlit IN TYPEOF (v2)) THEN
      parts(v1,p,q);  parts(v2,r,s);  t := r*r+s*s;
      RETURN (makec((p*r+q*s)/t,(q*r-p*s)/t));
    END_IF;
  ef_exponentiate_c :
    IF (cnlit IN TYPEOF (v1)) AND (cnlit IN TYPEOF (v2)) THEN
      parts(v1,p,q);  parts(v2,r,s);  t := 0.5*LOG(p*p+q*q);  u := atan2(q,p);
      p := r*t-s*u;  q := r*u+s*t;  r := EXP(p);
      RETURN (makec(r*COS(q),r*SIN(q)));
    END_IF;
  ef_exponentiate_ci :
    IF (cnlit IN TYPEOF (v1)) AND ('INTEGER' IN TYPEOF (v2)) THEN
      parts(v1,p,q);  k := v2;  r := 1.0;  s := 0.0;
      REPEAT i := 1 TO ABS(k);  r := p*r-q*s;  s := p*s+q*r;  END_REPEAT;
      IF k < 0 THEN  t := r*r+s*s;  r := r/t;  s := -s/t;  END_IF;
      RETURN (makec(r,s));
    END_IF;
  ef_eq_c :
    IF (cnlit IN TYPEOF (v1)) AND (cnlit IN TYPEOF (v2)) THEN
      parts(v1,p,q);  parts(v2,r,s);  RETURN (ctmv((p = r) AND (q = s)));
    END_IF;
  ef_ne_c :
    IF (cnlit IN TYPEOF (v1)) AND (cnlit IN TYPEOF (v2)) THEN
      parts(v1,p,q);  parts(v2,r,s);  RETURN (ctmv((p <> r) OR (q <> s)));
    END_IF;
  ef_conjugate_c :
     IF cnlit IN TYPEOF (v1) THEN  parts(v1,p,q);  RETURN (makec(p,-q));
     END_IF;
  ef_abs_c :
    IF cnlit IN TYPEOF (v1) THEN
      parts(v1,p,q);  RETURN (ctmv(SQRT(p*p+q*q)));
    END_IF;
  ef_arg_c :
    IF cnlit IN TYPEOF (v1) THEN
      parts(v1,p,q);  RETURN (ctmv(atan2(q,p)));
    END_IF;
  ef_cos_c :
    IF cnlit IN TYPEOF (v1) THEN
      parts(v1,p,q);  t := 0.5*EXP(-q);  u := 0.5*EXP(q);
      RETURN (makec((t+u)*COS(p),(t-u)*SIN(p)));
    END_IF;
  ef_exp_c :
    IF cnlit IN TYPEOF (v1) THEN
      parts(v1,p,q);  RETURN (makec(EXP(p)*COS(q),EXP(p)*SIN(q)));
    END_IF;
  ef_ln_c :
    IF cnlit IN TYPEOF (v1) THEN
```

```
    parts(v1,p,q);  RETURN (makec(0.5*LOG(p*p+q*q),atan2(q,p)));
  END_IF;
ef_sin_c :
  IF cnlit IN TYPEOF (v1) THEN
    parts(v1,p,q);  t := 0.5*EXP(-q);  u := 0.5*EXP(q);
    RETURN (makec((t+u)*SIN(p),(u-t)*COS(p)));
  END_IF;
ef_sqrt_c :
  IF cnlit IN TYPEOF (v1) THEN
    parts(v1,p,q);  t := SQRT(SQRT(p*p+q*q));  u := 0.5*atan2(q,p);
    RETURN (makec(t*COS(u),t*SIN(u)));
  END_IF;
ef_tan_c :
  IF cnlit IN TYPEOF (v1) THEN
    parts(v1,p,q);  t := EXP(2.0*q) + EXP(-2.0*q) + 2.0*COS(2.0*p);
    RETURN (makec(2.0*SIN(2.0*p)/t,(EXP(-2.0*q)-EXP(2.0*q))/t));
  END_IF;
-- ef_if_c : combined with ef_if
ef_subscript_s :
  IF ('STRING' IN TYPEOF (v1)) AND ('INTEGER' IN TYPEOF (v2)) THEN
    str := v1;  k := v2;  RETURN (ctmv(str[k]));
  END_IF;
ef_eq_s :
  IF ('STRING' IN TYPEOF (v1)) AND ('STRING' IN TYPEOF (v2)) THEN
    str := v1;  st2 := v2;  RETURN (ctmv(str = st2));
  END_IF;
ef_ne_s :
  IF ('STRING' IN TYPEOF (v1)) AND ('STRING' IN TYPEOF (v2)) THEN
    str := v1;  st2 := v2;  RETURN (ctmv(str <> st2));
  END_IF;
ef_gt_s :
  IF ('STRING' IN TYPEOF (v1)) AND ('STRING' IN TYPEOF (v2)) THEN
    str := v1;  st2 := v2;  RETURN (ctmv(str > st2));
  END_IF;
ef_lt_s :
  IF ('STRING' IN TYPEOF (v1)) AND ('STRING' IN TYPEOF (v2)) THEN
    str := v1;  st2 := v2;  RETURN (ctmv(str < st2));
  END_IF;
ef_ge_s :
  IF ('STRING' IN TYPEOF (v1)) AND ('STRING' IN TYPEOF (v2)) THEN
    str := v1;  st2 := v2;  RETURN (ctmv(str >= st2));
  END_IF;
ef_le_s :
  IF ('STRING' IN TYPEOF (v1)) AND ('STRING' IN TYPEOF (v2)) THEN
    str := v1;  st2 := v2;  RETURN (ctmv(str <= st2));
  END_IF;
ef_subsequence_s :
  IF ('STRING' IN TYPEOF (v1)) AND ('INTEGER' IN TYPEOF (v2)) AND
    ('INTEGER' IN TYPEOF (v3)) THEN
    str := v1;  j := v2;  k := v3;  RETURN (ctmv(str[j:k]));
  END_IF;
ef_concat_s : BEGIN
  str := '';
  REPEAT i := SIZEOF (vlist) TO 1 BY -1;
    IF 'STRING' IN TYPEOF (vlist[i]) THEN
      st2 := vlist[i];  str := str + st2;
      REMOVE (vlist, i);
    ELSE IF str <> '' THEN
      INSERT (vlist, ctmv(str), i);
```

```
      str := '';
    END_IF;  END_IF;
  END_REPEAT;
  IF SIZEOF (vlist) = 0 THEN  RETURN (ctmv(str));  END_IF;
  IF str <> '' THEN  INSERT (vlist, ctmv(str), 0);  END_IF;
  IF SIZEOF (vlist) = 1 THEN  RETURN (vlist[1]);  END_IF;
  END;
ef_size_s :
   IF 'STRING' IN TYPEOF (v1) THEN  str:=v1;  RETURN (ctmv(LENGTH(str)));
   END_IF;
ef_format :
  IF ('NUMBER' IN TYPEOF (v1)) AND ('STRING' IN TYPEOF (v2)) THEN
    RETURN (ctmv(FORMAT(v1,v2)));
  END_IF;
ef_value :
IF 'STRING' IN TYPEOF (v1) THEN  str:=v1;  RETURN (ctmv(VALUE(str)));
END_IF;
ef_like :
  IF ('STRING' IN TYPEOF (v1)) AND ('STRING' IN TYPEOF (v2)) THEN
    RETURN (ctmv(v1 LIKE v2));
  END_IF;
-- ef_if_s : combined with ef_if
ef_subscript_b :
  IF ('BINARY' IN TYPEOF (v1)) AND ('INTEGER' IN TYPEOF (v2)) THEN
    bin := v1;  k := v2;  RETURN (ctmv(bin[k]));
  END_IF;
ef_eq_b :
  IF ('BINARY' IN TYPEOF (v1)) AND ('BINARY' IN TYPEOF (v2)) THEN
    bin := v1;  bi2 := v2;  RETURN (ctmv(bin = bi2));
  END_IF;
ef_ne_b :
  IF ('BINARY' IN TYPEOF (v1)) AND ('BINARY' IN TYPEOF (v2)) THEN
    bin := v1;  bi2 := v2;  RETURN (ctmv(bin <> bi2));
  END_IF;
ef_gt_b :
  IF ('BINARY' IN TYPEOF (v1)) AND ('BINARY' IN TYPEOF (v2)) THEN
    bin := v1;  bi2 := v2;  RETURN (ctmv(bin > bi2));
  END_IF;
ef_lt_b :
  IF ('BINARY' IN TYPEOF (v1)) AND ('BINARY' IN TYPEOF (v2)) THEN
    bin := v1;  bi2 := v2;  RETURN (ctmv(bin < bi2));
  END_IF;
ef_ge_b :
  IF ('BINARY' IN TYPEOF (v1)) AND ('BINARY' IN TYPEOF (v2)) THEN
    bin := v1;  bi2 := v2;  RETURN (ctmv(bin >= bi2));
  END_IF;
ef_le_b :
  IF ('BINARY' IN TYPEOF (v1)) AND ('BINARY' IN TYPEOF (v2)) THEN
    bin := v1;  bi2 := v2;  RETURN (ctmv(bin <= bi2));
  END_IF;
ef_subsequence_b :
  IF ('BINARY' IN TYPEOF (v1)) AND ('INTEGER' IN TYPEOF (v2)) AND
    ('INTEGER' IN TYPEOF (v3)) THEN
    bin := v1;  j := v2;  k := v3;  RETURN (ctmv(bin[j:k]));
  END_IF;
ef_concat_b : BEGIN
  boo := FALSE;
  REPEAT i := SIZEOF (vlist) TO 1 BY -1;
    IF 'BINARY' IN TYPEOF (vlist[i]) THEN
```

```
      IF boo THEN  bi2 := vlist[i];  bin := bin + bi2;
      ELSE         bin := vlist[i];  boo := TRUE;  END_IF;
      REMOVE (vlist, i);
    ELSE IF boo THEN
      INSERT (vlist, ctmv(bin), i);
      boo := FALSE;
    END_IF;  END_IF;
  END_REPEAT;
  IF SIZEOF (vlist) = 0 THEN  RETURN (ctmv(bin));  END_IF;
  IF boo THEN  INSERT (vlist, ctmv(bin), 0);  END_IF;
  IF SIZEOF (vlist) = 1 THEN  RETURN (vlist[1]);  END_IF;
  END;
ef_size_b :
IF 'BINARY' IN TYPEOF (v1) THEN  bin:=v1;  RETURN (ctmv(BLENGTH(bin)));
END_IF;
 -- ef_if_b : combined with ef_if
ef_subscript_t :
  IF ('LIST' IN TYPEOF (v1)) AND ('INTEGER' IN TYPEOF (v2)) THEN
    tpl := v1;  k := v2;  RETURN (ctmv(tpl[k]));
  END_IF;
ef_eq_t :
  IF ('LIST' IN TYPEOF (v1)) AND ('LIST' IN TYPEOF (v2)) THEN
    lgc := equal_maths_values(v1,v2);
    IF lgc <> UNKNOWN THEN  RETURN (ctmv(lgc));  END_IF;
  END_IF;
ef_ne_t :
  IF ('LIST' IN TYPEOF (v1)) AND ('LIST' IN TYPEOF (v2)) THEN
    lgc := equal_maths_values(v1,v2);
    IF lgc <> UNKNOWN THEN  RETURN (ctmv(NOT lgc));  END_IF;
  END_IF;
ef_concat_t : BEGIN
  tpl := [];
  REPEAT i := SIZEOF (vlist) TO 1 BY -1;
    IF 'STRING' IN TYPEOF (vlist[i]) THEN
      tp2 := vlist[i];  tpl := tpl + tp2;
      REMOVE (vlist, i);
    ELSE IF SIZEOF (tpl) <> 0 THEN
      INSERT (vlist, ctmv(tpl), i);
      tpl := [];
    END_IF;  END_IF;
  END_REPEAT;
  IF SIZEOF (vlist) = 0 THEN  RETURN (ctmv(tpl));  END_IF;
  IF SIZEOF (tpl) <> 0 THEN  INSERT (vlist, ctmv(tpl), 0);  END_IF;
  IF SIZEOF (vlist) = 1 THEN  RETURN (vlist[1]);  END_IF;
  END;
ef_size_t :
   IF 'LIST' IN TYPEOF (v1) THEN  tpl:=v1;  RETURN (ctmv(SIZEOF(tpl)));
   END_IF;
ef_entuple :
  RETURN (ctmv(vlist));
ef_detuple :  -- This can have multiple outputs, but the expression only
              -- denotes the first.
  IF 'LIST' IN TYPEOF (v1) THEN  tpl:=v1;  RETURN (ctmv(tpl[1]));  END_IF;
ef_insert :
  IF ('LIST' IN TYPEOF (v1)) AND ('INTEGER' IN TYPEOF (v3)) THEN
    tpl := v1;  k := v3;  INSERT (tpl, v2, k);  RETURN (ctmv(tpl));
  END_IF;
ef_remove :
  IF ('LIST' IN TYPEOF (v1)) AND ('INTEGER' IN TYPEOF (v2)) THEN
```

```
    tpl := v1;  k := v2;  REMOVE (tpl, k);  RETURN (ctmv(tpl));
  END_IF;
-- ef_if_t : combined with ef_if
ef_sum_it :
  IF good_t(v1,'INTEGER') THEN
    tpl := v1;  j := 0;
    REPEAT i := 1 TO SIZEOF (tpl);  j := j + tpl[i];  END_REPEAT;
    RETURN (ctmv(j));
  END_IF;
ef_product_it :
  IF good_t(v1,'INTEGER') THEN
    tpl := v1;  j := 1;
    REPEAT i := 1 TO SIZEOF (tpl);  j := j * tpl[i];  END_REPEAT;
    RETURN (ctmv(j));
  END_IF;
ef_add_it : BEGIN
  boo := FALSE;
  REPEAT i := SIZEOF (vlist) TO 1 BY -1;
    IF good_t(vlist[i],'INTEGER') THEN
      IF NOT boo THEN  tpl := vlist[i];  boo := TRUE;
      ELSE
        tp2 := vlist[i];
        IF SIZEOF (tpl) <> SIZEOF (tp2) THEN  RETURN (?);  END_IF;
   REPEAT l := 1 TO SIZEOF (tpl);  tpl[j] := tpl[j] + tp2[j];  END_REPEAT;
      END_IF;
      REMOVE (vlist, i);
    END_IF;
  END_REPEAT;
  IF SIZEOF (vlist) = 0 THEN  RETURN (ctmv(tpl));  END_IF;
  IF boo THEN  INSERT (vlist, ctmv(tpl), 0);  END_IF;
  IF SIZEOF (vlist) = 1 THEN  RETURN (vlist[1]);  END_IF;
  END;
ef_subtract_it :
  IF good_t(v1,'INTEGER') AND good_t(v2,'INTEGER') THEN
    tpl := v1;  tp2 := v2;
    IF SIZEOF (tpl) <> SIZEOF (tp2) THEN  RETURN (?);  END_IF;
    REPEAT i := 1 TO SIZEOF (tpl);  tpl[i] := tpl[i] - tp2[i];  END_REPEAT;
    RETURN (ctmv(tpl));
  END_IF;
ef_scalar_mult_it :
  IF ('INTEGER' IN TYPEOF (v1)) AND good_t(v2,'INTEGER') THEN
    j := v1;  tpl := v2;
    REPEAT i := 1 TO SIZEOF (tpl);  tpl[i] := j * tpl[i];  END_REPEAT;
    RETURN (ctmv(tpl));
  END_IF;
ef_dot_prod_it :
  IF good_t(v1,'INTEGER') AND good_t(v2,'INTEGER') THEN
    tpl := v1;  tp2 := v2;  j := 0;
    IF SIZEOF (tpl) <> SIZEOF (tp2) THEN  RETURN (?);  END_IF;
    REPEAT i := 1 TO SIZEOF (tpl);  j := j + tpl[i] * tp2[i];  END_REPEAT;
    RETURN (ctmv(j));
  END_IF;
ef_sum_rt :
  IF good_t(v1,'REAL') THEN
    tpl := v1;  r := 0.0;
    REPEAT i := 1 TO SIZEOF (tpl);  r := r + tpl[i];  END_REPEAT;
    RETURN (ctmv(r));
  END_IF;
ef_product_rt :
```

```
  IF good_t(v1,'REAL') THEN
    tpl := v1;  r := 1.0;
    REPEAT i := 1 TO SIZEOF (tpl);  r := r * tpl[i];  END_REPEAT;
    RETURN (ctmv(r));
  END_IF;
ef_add_rt : BEGIN
  boo := FALSE;
  REPEAT i := SIZEOF (vlist) TO 1 BY -1;
    IF good_t(vlist[i],'REAL') THEN
      IF NOT boo THEN  tpl := vlist[i];  boo := TRUE;
      ELSE
        tp2 := vlist[i];
        IF SIZEOF (tpl) <> SIZEOF (tp2) THEN  RETURN (?);  END_IF;
        REPEAT l := 1 TO SIZEOF (tpl);  tpl[j] := tpl[j] + tp2[j];
      END_REPEAT;
      END_IF;
      REMOVE (vlist, i);
    END_IF;
  END_REPEAT;
  IF SIZEOF (vlist) = 0 THEN  RETURN (ctmv(tpl));  END_IF;
  IF boo THEN  INSERT (vlist, ctmv(tpl), 0);  END_IF;
  IF SIZEOF (vlist) = 1 THEN  RETURN (vlist[1]);  END_IF;
  END;
ef_subtract_rt :
  IF good_t(v1,'REAL') AND good_t(v2,'REAL') THEN
    tpl := v1;  tp2 := v2;
    IF SIZEOF (tpl) <> SIZEOF (tp2) THEN  RETURN (?);  END_IF;
    REPEAT i := 1 TO SIZEOF (tpl);  tpl[i] := tpl[i] - tp2[i];  END_REPEAT;
    RETURN (ctmv(tpl));
  END_IF;
ef_scalar_mult_rt :
  IF ('REAL' IN TYPEOF (v1)) AND good_t(v2,'REAL') THEN
    r := v1;  tpl := v2;
    REPEAT i := 1 TO SIZEOF (tpl);  tpl[i] := r * tpl[i];  END_REPEAT;
    RETURN (ctmv(tpl));
  END_IF;
ef_dot_prod_rt :
  IF good_t(v1,'REAL') AND good_t(v2,'REAL') THEN
    tpl := v1;  tp2 := v2;  r := 0;
    IF SIZEOF (tpl) <> SIZEOF (tp2) THEN  RETURN (?);  END_IF;
    REPEAT i := 1 TO SIZEOF (tpl);  r := r + tpl[i] * tp2[i];  END_REPEAT;
    RETURN (ctmv(r));
  END_IF;
ef_norm_rt :
  IF good_t(v1,'REAL') THEN
    tpl := v1;  r := 0.0;
    REPEAT i := 1 TO SIZEOF (tpl);  r := r + tpl[i]*tpl[i];  END_REPEAT;
    RETURN (ctmv(SQRT(r)));
  END_IF;
ef_sum_ct :
  IF good_t(v1,cnlit) THEN
    tpl := v1;  p := 0.0;  q := 0.0;
   REPEAT i:=1 TO SIZEOF (tpl);  parts(tpl[i],r,s);  p:=p+r;  q:=q+s;
  END_REPEAT;
    RETURN (makec(p,q));
  END_IF;
ef_product_ct :
  IF good_t(v1,cnlit) THEN
    tpl := v1;  p := 1.0;  q := 0.0;
```

```
    REPEAT i := 1 TO SIZEOF (tpl);
      parts(tpl[i],r,s);  p := p*r-q*s;  q := p*s+q*r;
    END_REPEAT;
    RETURN (makec(p,q));
  END_IF;
ef_add_ct : BEGIN
  boo := FALSE;
  REPEAT i := SIZEOF (vlist) TO 1 BY -1;
    IF good_t(vlist[i],cnlit) THEN
      IF NOT boo THEN  tpl := vlist[i];  boo := TRUE;
      ELSE
        tp2 := vlist[i];
        IF SIZEOF (tpl) <> SIZEOF (tp2) THEN  RETURN (?);  END_IF;
        REPEAT l := 1 TO SIZEOF (tpl);
          parts(tpl[j],p,q); parts(tp2[j],r,s);  tpl[j] := makec(p+r,q+s);
        END_REPEAT;
      END_IF;
      REMOVE (vlist, i);
    END_IF;
  END_REPEAT;
  IF SIZEOF (vlist) = 0 THEN  RETURN (ctmv(tpl));  END_IF;
  IF boo THEN  INSERT (vlist, ctmv(tpl), 0);  END_IF;
  IF SIZEOF (vlist) = 1 THEN  RETURN (vlist[1]);  END_IF;
  END;
ef_subtract_ct :
  IF good_t(v1,cnlit) AND good_t(v2,cnlit) THEN
    tpl := v1;  tp2 := v2;
    IF SIZEOF (tpl) <> SIZEOF (tp2) THEN  RETURN (?);  END_IF;
    REPEAT i := 1 TO SIZEOF (tpl);
      parts(tpl[i],p,q);  parts(tp2[i],r,s);  tpl[i] := makec(p-r,q-s);
    END_REPEAT;
    RETURN (ctmv(tpl));
  END_IF;
ef_scalar_mult_ct :
  IF (cnlit IN TYPEOF (v1)) AND good_t(v2,cnlit) THEN
    parts(v1,p,q);  tpl := v2;
    REPEAT i := 1 TO SIZEOF (tpl);
      parts(tpl[i],r,s);  tpl[i] := makec(p*r-q*s,p*s+q*r);
    END_REPEAT;
    RETURN (ctmv(tpl));
  END_IF;
ef_dot_prod_ct :
  IF good_t(v1,cnlit) AND good_t(v2,cnlit) THEN
    tpl := v1;  tp2 := v2;  t := 0.0;  u := 0.0;
    IF SIZEOF (tpl) <> SIZEOF (tp2) THEN  RETURN (?);  END_IF;
    REPEAT i := 1 TO SIZEOF (tpl);
     parts(tpl[i],p,q);  parts(tp2[i],r,s);  t := t + p*r+q*s;  u := u + q*r-
   p*s;
    END_REPEAT;
    RETURN (makec(t,u));
  END_IF;
ef_norm_ct :
  IF good_t(v1,cnlit) THEN
    tpl := v1;  r := 0.0;
  REPEAT i := 1 TO SIZEOF (tpl);  parts(tpl[i],p,q);  r:=r+p*p+q*q;
  END_REPEAT;
    RETURN (ctmv(SQRT(r)));
  END_IF;
ef_if, ef_if_i, ef_if_r, ef_if_c, ef_if_s, ef_if_b, ef_if_t :
```

```
      IF 'LOGICAL' IN TYPEOF (v1) THEN
        lgc := v1;  IF lgc THEN  RETURN (v2);  ELSE  RETURN (v3);  END_IF;
      END_IF;
    ef_ensemble :   -- (mem + vlist) effectively converts list to set
      RETURN (make_finite_space(mem + vlist));
    ef_member_of :
      IF (schema_prefix + 'MATHS_SPACE') IN TYPEOF (v2) THEN
        lgc := member_of(v1,v2);
        IF lgc <> UNKNOWN THEN  RETURN (ctmv(lgc));  END_IF;
      END_IF;
    END_CASE;
    RETURN (make_function_application(expr.func,vlist));
  END_IF;
  IF 'ABSTRACTED_EXPRESSION_FUNCTION' IN types THEN
    gexpr := substitute(expr.func\abstracted_expression_function.expr,
      expr.func\quantifier_expression.variables,vlist);
    RETURN (simplify_generic_expression(gexpr));
  END_IF;
  IF 'FINITE_FUNCTION' IN types THEN
    pairs := expr.func\finite_function.pairs;
    REPEAT i := 1 TO SIZEOF (pairs);
      IF equal_maths_values(vlist[1],pairs[i][1]) THEN
        RETURN (simplify_maths_value(pairs[i][2]));
      END_IF;
    END_REPEAT;
    RETURN (make_function_application(expr.func,vlist));
  END_IF;
  RETURN (expr);
END_FUNCTION;  -- simplify_function_application

 FUNCTION simplify_generic_expression(expr : generic_expression) : maths_value;
FUNCTION restore_unary(expr : unary_generic_expression;
                       opnd : generic_expression) : generic_expression;
  expr.operand := opnd;
  RETURN (expr);
END_FUNCTION;  -- restore_unary
FUNCTION restore_binary(expr       : binary_generic_expression;
                        opd1, opd2 : generic_expression) : generic_expression;
  expr.operands[1] := opd1;
  expr.operands[2] := opd2;
  RETURN (expr);
END_FUNCTION;  -- restore_binary
FUNCTION restore_mulary(expr : multiple_arity_generic_expression;
                            ops  : LIST OF generic_expression) :
    generic_expression;
  expr.operands := ops;
  RETURN (expr);
END_FUNCTION;  -- restore_mulary
FUNCTION make_number_literal(nmb : NUMBER) : generic_literal;
  IF 'INTEGER' IN TYPEOF (nmb) THEN  RETURN (make_int_literal(nmb));  END_IF;
  RETURN (make_real_literal(nmb));
END_FUNCTION;  -- make_number_literal;
LOCAL
  types : SET OF STRING := stripped_typeof (expr);
  v1, v2 : maths_value;
  vlist : LIST OF maths_value := [];
  op1, op2 : generic_expression;
  oplist : LIST OF generic_expression := [];
  opnds : LIST [2:?] OF generic_expression;
```

```
    n, m : INTEGER;
    finfun : maths_function_select;
    boo : BOOLEAN;
    str : STRING;
    nmb : NUMBER;
  END_LOCAL;
  -- Unwrap the elementary kinds of literals
  IF 'INT_LITERAL' IN types THEN
    RETURN (convert_to_maths_value (expr\int_literal.the_value));
  END_IF;
  IF 'REAL_LITERAL' IN types THEN
    RETURN (convert_to_maths_value (expr\real_literal.the_value));
  END_IF;
  IF 'BOOLEAN_LITERAL' IN types THEN
    RETURN (convert_to_maths_value (expr\boolean_literal.the_value));
  END_IF;
  IF 'STRING_LITERAL' IN types THEN
    RETURN (convert_to_maths_value (expr\string_literal.the_value));
  END_IF;
  IF 'COMPLEX_NUMBER_LITERAL' IN types THEN
    RETURN (expr);  -- No simpler expression available
  END_IF;
  IF 'LOGICAL_LITERAL' IN types THEN
    RETURN (convert_to_maths_value (expr\logical_literal.lit_value));
  END_IF;
  IF 'BINARY_LITERAL' IN types THEN
    RETURN (convert_to_maths_value (expr\binary_literal.lit_value));
  END_IF;
  IF 'MATHS_ENUM_LITERAL' IN types THEN
    RETURN (expr\maths_enum_literal.lit_value);
  END_IF;
  IF 'REAL_TUPLE_LITERAL' IN types THEN
    RETURN (convert_to_maths_value (expr\real_tuple_literal.lit_value));
  END_IF;
  IF 'INTEGER_TUPLE_LITERAL' IN types THEN
    RETURN (convert_to_maths_value (expr\integer_tuple_literal.lit_value));
  END_IF;
  IF 'ATOM_BASED_LITERAL' IN types THEN
    RETURN (expr\atom_based_literal.lit_value);
  END_IF;
  IF 'MATHS_TUPLE_LITERAL' IN types THEN
    RETURN (convert_to_maths_value (expr\maths_tuple_literal.lit_value));
  END_IF;
  -- Simplify one special class of literals
  IF 'MATHS_SPACE' IN types THEN
    RETURN (simplify_maths_space(expr));
  END_IF;
  -- Simplify one special kind of expression
  IF 'FUNCTION_APPLICATION' IN types THEN
    RETURN (simplify_function_application(expr));
  END_IF;
  -- Separate and simplify the operands
  IF 'UNARY_GENERIC_EXPRESSION' IN types THEN
    v1 := simplify_generic_expression(expr\unary_generic_expression.operand);
    op1 := convert_to_operand(v1);
  END_IF;
  IF 'BINARY_GENERIC_EXPRESSION' IN types THEN
     v1 :=
   simplify_generic_expression(expr\binary_generic_expression.operands[1]);
```

```
  op1 := convert_to_operand(v1);
   v2 :=
  simplify_generic_expression(expr\binary_generic_expression.operands[2]);
  op2 := convert_to_operand(v2);
END_IF;
IF 'MULTIPLE_ARITY_GENERIC_EXPRESSION' IN types THEN
  opnds := expr\multiple_arity_generic_expression.operands;
  REPEAT i := 1 TO SIZEOF (opnds);
    v1 := simplify_generic_expression(opnds[i]);
    INSERT (vlist, v1, i-1);
    INSERT (oplist, convert_to_operand(v1), i-1);
  END_REPEAT;
END_IF;
-- Simplify the one kind of maths_function which derives its operands.
IF 'PARALLEL_COMPOSED_FUNCTION' IN types THEN
  v1 := vlist[1];
  n := SIZEOF (vlist);
  finfun := vlist[n];
  REMOVE (vlist, n);
  REMOVE (vlist, 1);
  RETURN (make_parallel_composed_function(v1,vlist,finfun));
END_IF;
   -- Simplify individual kinds of expressions.  It is not necessary to cover
  all cases.
IF ('ABS_EXPRESSION' IN types) AND ('NUMBER' IN TYPEOF (v1)) THEN
  RETURN (convert_to_maths_value (ABS(v1)));
END_IF;
IF ('ACOS_EXPRESSION' IN types) AND ('NUMBER' IN TYPEOF (v1)) THEN
  RETURN (convert_to_maths_value (ACOS(v1)));
END_IF;
IF 'AND_EXPRESSION' IN types THEN
  REPEAT i := SIZEOF (vlist) TO 1 BY -1;
    IF 'BOOLEAN' IN TYPEOF (vlist[i]) THEN
      boo := vlist[i];
      IF NOT boo THEN  RETURN (convert_to_maths_value(FALSE));  END_IF;
      REMOVE (oplist, i);
    END_IF;
  END_REPEAT;
  IF SIZEOF (oplist) = 0 THEN  RETURN (convert_to_maths_value(TRUE));  END_IF;
  IF SIZEOF (oplist) = 1 THEN  RETURN (oplist[1]);  END_IF;
END_IF;
IF ('ASIN_EXPRESSION' IN types) AND ('NUMBER' IN TYPEOF (v1)) THEN
  RETURN (convert_to_maths_value (ASIN(v1)));
END_IF;
IF ('ATAN_EXPRESSION' IN types) AND
  ('NUMBER' IN TYPEOF (v1)) AND ('NUMBER' IN TYPEOF (v2)) THEN
  RETURN (convert_to_maths_value (ATAN(v1,v2)));
END_IF;
IF ('COMPARISON_EXPRESSION' IN types) AND (
  (('NUMBER' IN TYPEOF (v1)) AND ('NUMBER' IN TYPEOF (v2))) OR
  (('STRING' IN TYPEOF (v1)) AND ('STRING' IN TYPEOF (v2))) OR
  (('BOOLEAN' IN TYPEOF (v1)) AND ('BOOLEAN' IN TYPEOF (v2))) ) THEN
  IF      'COMPARISON_EQUAL'         IN types THEN  boo := bool(v1 = v2);
  ELSE IF 'COMPARISON_GREATER'       IN types THEN  boo := bool(v1 > v2);
  ELSE IF 'COMPARISON_GREATER_EQUAL' IN types THEN  boo := bool(v1 >= v2);
  ELSE IF 'COMPARISON_LESS'          IN types THEN  boo := bool(v1 < v2);
  ELSE IF 'COMPARISON_LESS_EQUAL'    IN types THEN  boo := bool(v1 <= v2);
  ELSE IF 'COMPARISON_NOT_EQUAL'     IN types THEN  boo := bool(v1 <> v2);
  ELSE IF 'LIKE_EXPRESSION'          IN types THEN  boo := bool(v1 LIKE v2);
```

```
  ELSE  RETURN (?);  -- Unreachable
  END_IF;  END_IF;  END_IF;  END_IF;  END_IF;  END_IF;  END_IF;
  RETURN (convert_to_maths_value (boo));
END_IF;
IF 'CONCAT_EXPRESSION' IN types THEN
  str := '';
  REPEAT i := SIZEOF (vlist) TO 1 BY -1;
    IF 'STRING' IN TYPEOF (vlist[i]) THEN
      str := vlist[i] + str;
      REMOVE (oplist, i);
    ELSE IF LENGTH(str) > 0 THEN
      INSERT (oplist, make_string_literal(str), i);
      str := '';
    END_IF;  END_IF;
  END_REPEAT;
  IF SIZEOF (oplist) = 0 THEN  RETURN (convert_to_maths_value(str));  END_IF;
     IF LENGTH(str) > 0 THEN  INSERT (oplist, make_string_literal(str), 0);
     END_IF;
  IF SIZEOF (oplist) = 1 THEN  RETURN (oplist[1]);  END_IF;
END_IF;
IF ('COS_EXPRESSION' IN types) AND ('NUMBER' IN TYPEOF (v1)) THEN
  RETURN (convert_to_maths_value (COS(v1)));
END_IF;
IF ('DIV_EXPRESSION' IN types) AND
  ('NUMBER' IN TYPEOF (v1)) AND ('NUMBER' IN TYPEOF (v2)) THEN
  RETURN (convert_to_maths_value (v1 DIV v2));
END_IF;
IF 'EQUALS_EXPRESSION' IN types THEN
  opnds := expr\binary_generic_expression.operands;
  RETURN (convert_to_maths_value (opnds[1] :=: opnds[2]));
END_IF;
IF ('EXP_EXPRESSION' IN types) AND ('NUMBER' IN TYPEOF (v1)) THEN
  RETURN (convert_to_maths_value (EXP(v1)));
END_IF;
IF ('FORMAT_EXPRESSION' IN types) AND
  ('NUMBER' IN TYPEOF (v1)) AND ('STRING' IN TYPEOF (v2)) THEN
  RETURN (convert_to_maths_value (FORMAT(v1,v2)));
END_IF;
IF ('INDEX_EXPRESSION' IN types) AND
  ('STRING' IN TYPEOF (v1)) AND ('NUMBER' IN TYPEOF (v2)) THEN
  str := v1;  n := v2;
  RETURN (convert_to_maths_value (str[n]));
END_IF;
IF ('INT_VALUE_EXPRESSION' IN types) AND ('STRING' IN TYPEOF (v1)) THEN
  RETURN (convert_to_maths_value (VALUE(v1)));
END_IF;
IF 'INTERVAL_EXPRESSION' IN types THEN
  str := '';
  IF 'NUMBER'  IN TYPEOF (vlist[1]) THEN str := 'NUMBER';   END_IF;
  IF 'STRING'  IN TYPEOF (vlist[1]) THEN str := 'STRING';   END_IF;
  IF 'BOOLEAN' IN TYPEOF (vlist[1]) THEN str := 'BOOLEAN';  END_IF;
  IF (LENGTH (str) > 0) AND (str IN TYPEOF (vlist[2])) AND
    (str IN TYPEOF (vlist[3])) THEN
    RETURN (convert_to_maths_value ({vlist[1] <= vlist[2] <= vlist[3]}));
  END_IF;
END_IF;
IF ('LENGTH_EXPRESSION' IN types) AND ('STRING' IN TYPEOF (v1)) THEN
  RETURN (convert_to_maths_value (LENGTH(v1)));
END_IF;
```

```
IF ('LOG_EXPRESSION' IN types) AND ('NUMBER' IN TYPEOF (v1)) THEN
  RETURN (convert_to_maths_value (LOG(v1)));
END_IF;
IF ('LOG10_EXPRESSION' IN types) AND ('NUMBER' IN TYPEOF (v1)) THEN
  RETURN (convert_to_maths_value (LOG10(v1)));
END_IF;
IF ('LOG2_EXPRESSION' IN types) AND ('NUMBER' IN TYPEOF (v1)) THEN
  RETURN (convert_to_maths_value (LOG2(v1)));
END_IF;
IF 'MAXIMUM_EXPRESSION' IN types THEN
  boo := FALSE;
  REPEAT i := SIZEOF (vlist) TO 1 BY -1;
    IF 'NUMBER' IN TYPEOF (vlist[i]) THEN
      IF boo THEN
        IF nmb < vlist[i] THEN  nmb := vlist[i];  END_IF;
      ELSE
        nmb := vlist[i];  boo := TRUE;
      END_IF;
      REMOVE (oplist, i);
    END_IF;
  END_REPEAT;
  IF SIZEOF (oplist) = 0 THEN  RETURN (convert_to_maths_value(nmb));  END_IF;
  IF boo THEN  INSERT (oplist, make_number_literal(nmb), 0);  END_IF;
END_IF;
IF 'MINIMUM_EXPRESSION' IN types THEN
  boo := FALSE;
  REPEAT i := SIZEOF (vlist) TO 1 BY -1;
    IF 'NUMBER' IN TYPEOF (vlist[i]) THEN
      IF boo THEN
        IF nmb > vlist[i] THEN  nmb := vlist[i];  END_IF;
      ELSE
        nmb := vlist[i];  boo := TRUE;
      END_IF;
      REMOVE (oplist, i);
    END_IF;
  END_REPEAT;
  IF SIZEOF (oplist) = 0 THEN  RETURN (convert_to_maths_value(nmb));  END_IF;
  IF boo THEN  INSERT (oplist, make_number_literal(nmb), 0);  END_IF;
END_IF;
IF ('MINUS_EXPRESSION' IN types) AND
  ('NUMBER' IN TYPEOF (v1)) AND ('NUMBER' IN TYPEOF (v2)) THEN
  RETURN (convert_to_maths_value (v1 - v2));
END_IF;
IF ('MOD_EXPRESSION' IN types) AND
  ('NUMBER' IN TYPEOF (v1)) AND ('NUMBER' IN TYPEOF (v2)) THEN
  RETURN (convert_to_maths_value (v1 MOD v2));
END_IF;
IF 'MULT_EXPRESSION' IN types THEN
  nmb := 1;
  REPEAT i := SIZEOF (vlist) TO 1 BY -1;
    IF 'NUMBER' IN TYPEOF (vlist[i]) THEN
      nmb := nmb * vlist[i];
      REMOVE (oplist, i);
    END_IF;
  END_REPEAT;
  IF SIZEOF (oplist) = 0 THEN  RETURN (convert_to_maths_value(nmb));  END_IF;
  IF nmb <> 1 THEN  INSERT (oplist, make_number_literal(nmb), 0);  END_IF;
  IF SIZEOF (oplist) = 1 THEN  RETURN (oplist[1]);  END_IF;
END_IF;
```

```
IF ('NOT_EXPRESSION' IN types) AND ('BOOLEAN' IN TYPEOF (v1)) THEN
  boo := v1;
  RETURN (convert_to_maths_value (NOT(boo)));
END_IF;
IF ('ODD_EXPRESSION' IN types) AND ('INTEGER' IN TYPEOF (v1)) THEN
  RETURN (convert_to_maths_value (ODD(v1)));
END_IF;
IF 'OR_EXPRESSION' IN types THEN
  REPEAT i := SIZEOF (vlist) TO 1 BY -1;
    IF 'BOOLEAN' IN TYPEOF (vlist[i]) THEN
      boo := vlist[i];
      IF boo THEN  RETURN (convert_to_maths_value(TRUE));  END_IF;
      REMOVE (oplist, i);
    END_IF;
  END_REPEAT;
  IF SIZEOF (oplist) = 0 THEN  RETURN (convert_to_maths_value(FALSE));  END_IF;
  IF SIZEOF (oplist) = 1 THEN  RETURN (oplist[1]);  END_IF;
END_IF;
IF 'PLUS_EXPRESSION' IN types THEN
  nmb := 0;
  REPEAT i := SIZEOF (vlist) TO 1 BY -1;
    IF 'NUMBER' IN TYPEOF (vlist[i]) THEN
      nmb := nmb + vlist[i];
      REMOVE (oplist, i);
    END_IF;
  END_REPEAT;
  IF SIZEOF (oplist) = 0 THEN  RETURN (convert_to_maths_value(nmb));  END_IF;
  IF nmb <> 0 THEN  INSERT (oplist, make_number_literal(nmb), 0);  END_IF;
  IF SIZEOF (oplist) = 1 THEN  RETURN (oplist[1]);  END_IF;
END_IF;
IF ('POWER_EXPRESSION' IN types) AND
  ('NUMBER' IN TYPEOF (v1)) AND ('NUMBER' IN TYPEOF (v2)) THEN
  RETURN (convert_to_maths_value (v1 ** v2));
END_IF;
IF ('SIN_EXPRESSION' IN types) AND ('NUMBER' IN TYPEOF (v1)) THEN
  RETURN (convert_to_maths_value (SIN(v1)));
END_IF;
IF ('SLASH_EXPRESSION' IN types) AND
  ('NUMBER' IN TYPEOF (v1)) AND ('NUMBER' IN TYPEOF (v2)) THEN
  RETURN (convert_to_maths_value (v1 / v2));
END_IF;
IF ('SQUARE_ROOT_EXPRESSION' IN types) AND ('NUMBER' IN TYPEOF (v1)) THEN
  RETURN (convert_to_maths_value (SQRT(v1)));
END_IF;
IF ('SUBSTRING_EXPRESSION' IN types) AND
  ('STRING' IN TYPEOF (vlist[1])) AND ('NUMBER' IN TYPEOF (vlist[2])) AND
  ('NUMBER' IN TYPEOF (vlist[3])) THEN
  str := vlist[1];  n := vlist[2];  m := vlist[3];
  RETURN (convert_to_maths_value (str[n:m]));
END_IF;
IF ('TAN_EXPRESSION' IN types) AND ('NUMBER' IN TYPEOF (v1)) THEN
  RETURN (convert_to_maths_value (TAN(v1)));
END_IF;
IF ('UNARY_MINUS_EXPRESSION' IN types) AND ('NUMBER' IN TYPEOF (v1)) THEN
  nmb := v1;
  RETURN (convert_to_maths_value (-nmb));
END_IF;
IF ('VALUE_EXPRESSION' IN types) AND ('STRING' IN TYPEOF (v1)) THEN
  RETURN (convert_to_maths_value (VALUE(v1)));
```

```
  END_IF;
  IF ('XOR_EXPRESSION' IN types) AND
    ('BOOLEAN' IN TYPEOF (v1)) AND ('BOOLEAN' IN TYPEOF (v2)) THEN
    RETURN (convert_to_maths_value (v1 XOR v2));
  END_IF;
  -- No special simplification defined, return same with simplified operands.
  IF 'UNARY_GENERIC_EXPRESSION' IN types THEN
    RETURN (restore_unary(expr,op1));
  END_IF;
  IF 'BINARY_GENERIC_EXPRESSION' IN types THEN
    RETURN (restore_binary(expr,op1,op2));
  END_IF;
  IF 'MULTIPLE_ARITY_GENERIC_EXPRESSION' IN types THEN
    RETURN (restore_mulary(expr,oplist));
  END_IF;
  -- Should be unreachable, but for safety, return unsimplified expression.
  RETURN (expr);
END_FUNCTION;  -- simplify_generic_expression


   FUNCTION simplify_maths_space
      (spc : maths_space ) : maths_space;
   LOCAL
      stypes : SET OF STRING := stripped_typeof(spc);
      sset : SET OF maths_value;
      zset : SET OF maths_value := [];
      zval : maths_value;
      zspc : maths_space;
      zallint : BOOLEAN := TRUE;
      zint : INTEGER;
      zmin : INTEGER;
      zmax : INTEGER;
      factors : LIST OF maths_space;
      zfactors : LIST OF maths_space := [];
      rspc : maths_space;
   END_LOCAL;
      IF 'FINITE_SPACE' IN stypes THEN
         sset := spc\finite_space.members;
         REPEAT i := 1 TO SIZEOF(sset);
            zval := simplify_maths_value(sset[i]);
            zset := zset + [ zval ];
            IF zallint AND ('INTEGER' IN TYPEOF(zval)) THEN
               zint := zval;
               IF i = 1 THEN
                  zmin := zint;
                  zmax := zint;
               ELSE
                  IF zint < zmin THEN
                     zmin := zint;
                  END_IF;
                  IF zint > zmax THEN
                     zmax := zint;
                  END_IF;
               END_IF;
            ELSE
               zallint := FALSE;
            END_IF;
         END_REPEAT;
         IF zallint AND (SIZEOF(zset) = zmax - zmin + 1) THEN
```

```
         RETURN (make_finite_integer_interval(zmin, zmax));
      END_IF;
      RETURN (make_finite_space(zset));
   END_IF;
   IF 'UNIFORM_PRODUCT_SPACE' IN stypes THEN
      zspc := simplify_maths_space(spc\uniform_product_space.base);
       RETURN (make_uniform_product_space(zspc,
    spc\uniform_product_space.exponent));
   END_IF;
   IF 'LISTED_PRODUCT_SPACE' IN stypes THEN
      factors := spc\listed_product_space.factors;
      REPEAT i := 1 TO SIZEOF(factors);
         INSERT( zfactors, simplify_maths_space(factors[i]), i - 1 );
      END_REPEAT;
      RETURN (make_listed_product_space(zfactors));
   END_IF;
   IF 'EXTENDED_TUPLE_SPACE' IN stypes THEN
      zspc := simplify_maths_space(spc\extended_tuple_space.base);
      rspc := simplify_maths_space(spc\extended_tuple_space.extender);
      RETURN (make_extended_tuple_space(zspc, rspc));
   END_IF;
   IF 'FUNCTION_SPACE' IN stypes THEN
      zspc := simplify_maths_space(spc\function_space.domain_argument);
      rspc := simplify_maths_space(spc\function_space.range_argument);
     RETURN (make_function_space(spc\function_space.domain_constraint, zspc,
    spc\function_space.range_constraint, rspc));
   END_IF;
   RETURN (spc);
END_FUNCTION;

FUNCTION simplify_maths_value
   (val : maths_value ) : maths_value;
LOCAL
   vtypes : SET OF STRING := stripped_typeof(val);
   vlist : LIST OF maths_value;
   nlist : LIST OF maths_value := [];
END_LOCAL;
   IF 'GENERIC_EXPRESSION' IN vtypes THEN
      RETURN (simplify_generic_expression(val));
   END_IF;
   IF 'LIST' IN vtypes THEN
      vlist := val;
      REPEAT i := 1 TO SIZEOF(vlist);
         INSERT( nlist, simplify_maths_value(vlist[i]), i - 1 );
      END_REPEAT;
      RETURN (convert_to_maths_value(nlist));
   END_IF;
   RETURN (val);
END_FUNCTION;

FUNCTION singleton_member_of
   (spc : maths_space ) : maths_value;
LOCAL
   types : SET OF STRING := stripped_typeof(spc);
END_LOCAL;
   IF 'FINITE_SPACE' IN types THEN
      IF SIZEOF(spc\finite_space.members) = 1 THEN
         RETURN (spc\finite_space.members[1]);
      END_IF;
```

```
      RETURN (?);
   END_IF;
   IF 'FINITE_INTEGER_INTERVAL' IN types THEN
      IF spc\finite_integer_interval.size = 1 THEN
         RETURN (spc\finite_integer_interval.min);
      END_IF;
      RETURN (?);
   END_IF;
   RETURN (?);
END_FUNCTION;

FUNCTION space_dimension
   (tspace : tuple_space ) : nonnegative_integer;
LOCAL
   types : SET OF STRING := TYPEOF(tspace);
END_LOCAL;
   IF schema_prefix + 'UNIFORM_PRODUCT_SPACE' IN types THEN
      RETURN (tspace\uniform_product_space.exponent);
   END_IF;
   IF schema_prefix + 'LISTED_PRODUCT_SPACE' IN types THEN
      RETURN (SIZEOF(tspace\listed_product_space.factors));
   END_IF;
   IF schema_prefix + 'EXTENDED_TUPLE_SPACE' IN types THEN
      RETURN (space_dimension(tspace\extended_tuple_space.base));
   END_IF;
   RETURN (?);
END_FUNCTION;

FUNCTION space_is_continuum
   (space : maths_space ) : BOOLEAN;
LOCAL
   typenames : SET OF STRING := TYPEOF(space);
   factors : LIST OF maths_space;
END_LOCAL;
   IF NOT EXISTS(space) THEN
      RETURN (FALSE);
   END_IF;
          IF subspace_of_es(space, es_reals) OR subspace_of_es(space,
    es_complex_numbers) THEN
      RETURN (TRUE);
   END_IF;
   IF schema_prefix + 'UNIFORM_PRODUCT_SPACE' IN typenames THEN
      RETURN (space_is_continuum(space\uniform_product_space.base));
   END_IF;
   IF schema_prefix + 'LISTED_PRODUCT_SPACE' IN typenames THEN
      factors := space\listed_product_space.factors;
      IF SIZEOF(factors) = 0 THEN
         RETURN (FALSE);
      END_IF;
      REPEAT i := 1 TO SIZEOF(factors);
         IF NOT space_is_continuum(factors[i]) THEN
            RETURN (FALSE);
         END_IF;
      END_REPEAT;
      RETURN (TRUE);
   END_IF;
   RETURN (FALSE);
END_FUNCTION;
```

```
   FUNCTION space_is_singleton
      (spc : maths_space ) : BOOLEAN;
   LOCAL
      types : SET OF STRING := stripped_typeof(spc);
   END_LOCAL;
      IF 'FINITE_SPACE' IN types THEN
         RETURN (bool(SIZEOF(spc\finite_space.members) = 1));
      END_IF;
      IF 'FINITE_INTEGER_INTERVAL' IN types THEN
         RETURN (bool(spc\finite_integer_interval.size = 1));
      END_IF;
      RETURN (FALSE);
   END_FUNCTION;

   FUNCTION stripped_typeof(arg : GENERIC:G) : SET OF STRING;
  LOCAL
    types : SET OF STRING := TYPEOF (arg);
    stypes : SET OF STRING := [];
    n : INTEGER := LENGTH (schema_prefix);
  END_LOCAL;
  REPEAT i := 1 TO SIZEOF (types);
    IF types[i][1:n] = schema_prefix THEN
       stypes := stypes + [types[i][n+1:LENGTH(types[i])]];
    ELSE
       stypes := stypes + [types[i]];
    END_IF;
  END_REPEAT;
  RETURN (stypes);
END_FUNCTION;  -- stripped_typeof

   FUNCTION subspace_of
      (space1 : maths_space;
       space2 : maths_space ) : LOGICAL;
   LOCAL
      spc1 : maths_space := simplify_maths_space(space1);
      spc2 : maths_space := simplify_maths_space(space2);
      types1 : SET OF STRING := stripped_typeof(spc1);
      types2 : SET OF STRING := stripped_typeof(spc2);
      lgcl : LOGICAL;
      cum : LOGICAL;
      es_val : elementary_space_enumerators;
      bnd1 : REAL;
      bnd2 : REAL;
      n : INTEGER;
      sp1 : maths_space;
      sp2 : maths_space;
      prgn1 : polar_complex_number_region;
      prgn2 : polar_complex_number_region;
      aitv : finite_real_interval;
   END_LOCAL;
      IF NOT EXISTS(spc1) OR NOT EXISTS(spc2) THEN
         RETURN (FALSE);
      END_IF;
      IF spc2 = the_generics THEN
         RETURN (TRUE);
      END_IF;
      IF 'ELEMENTARY_SPACE' IN types1 THEN
         IF NOT ('ELEMENTARY_SPACE' IN types2) THEN
            RETURN (FALSE);
```

```
      END_IF;
      es_val := spc2\elementary_space.space_id;
      IF spc1\elementary_space.space_id = es_val THEN
         RETURN (TRUE);
      END_IF;
      CASE spc1\elementary_space.space_id OF
         es_numbers :
               RETURN (FALSE);
         es_complex_numbers :
               RETURN (es_val = es_numbers);
         es_reals :
               RETURN (es_val = es_numbers);
         es_integers :
               RETURN (es_val = es_numbers);
         es_logicals :
               RETURN (FALSE);
         es_booleans :
               RETURN (es_val = es_logicals);
         es_strings :
               RETURN (FALSE);
         es_binarys :
               RETURN (FALSE);
         es_maths_spaces :
               RETURN (FALSE);
         es_maths_functions :
               RETURN (FALSE);
         es_generics :
               RETURN (FALSE);
      END_CASE;
      RETURN (UNKNOWN);
   END_IF;
   IF 'FINITE_INTEGER_INTERVAL' IN types1 THEN
      cum := TRUE;
        REPEAT i := spc1\finite_integer_interval.min TO
       spc1\finite_integer_interval.max;
         cum := cum AND member_of(i, spc2);
         IF cum = FALSE THEN
            RETURN (FALSE);
         END_IF;
      END_REPEAT;
      RETURN (cum);
   END_IF;
   IF 'INTEGER_INTERVAL_FROM_MIN' IN types1 THEN
      IF 'ELEMENTARY_SPACE' IN types2 THEN
         es_val := spc2\elementary_space.space_id;
         RETURN ((es_val = es_numbers) OR (es_val = es_integers));
      END_IF;
      IF 'INTEGER_INTERVAL_FROM_MIN' IN types2 THEN
        RETURN (spc1\integer_interval_from_min.min >=
       spc2\integer_interval_from_min.min);
      END_IF;
      RETURN (FALSE);
   END_IF;
   IF 'INTEGER_INTERVAL_TO_MAX' IN types1 THEN
      IF 'ELEMENTARY_SPACE' IN types2 THEN
         es_val := spc2\elementary_space.space_id;
         RETURN ((es_val = es_numbers) OR (es_val = es_integers));
      END_IF;
      IF 'INTEGER_INTERVAL_TO_MAX' IN types2 THEN
```

```
     RETURN (spc1\integer_interval_to_max.max <=
    spc2\integer_interval_to_max.max);
   END_IF;
   RETURN (FALSE);
END_IF;
IF 'FINITE_REAL_INTERVAL' IN types1 THEN
   IF 'ELEMENTARY_SPACE' IN types2 THEN
      es_val := spc2\elementary_space.space_id;
      RETURN ((es_val = es_numbers) OR (es_val = es_reals));
   END_IF;
     IF (('FINITE_REAL_INTERVAL' IN types2) OR ('REAL_INTERVAL_FROM_MIN' IN
    types2)) OR ('REAL_INTERVAL_TO_MAX' IN types2) THEN
      IF min_exists(spc2) THEN
         bnd1 := spc1\finite_real_interval.min;
         bnd2 := real_min(spc2);
          IF (bnd1 < bnd2) OR ((bnd1 = bnd2) AND min_included(spc1)) AND
       NOT min_included(spc2) THEN
            RETURN (FALSE);
         END_IF;
      END_IF;
      IF max_exists(spc2) THEN
         bnd1 := spc1\finite_real_interval.max;
         bnd2 := real_max(spc2);
          IF (bnd1 > bnd2) OR ((bnd1 = bnd2) AND max_included(spc1)) AND
       NOT max_included(spc2) THEN
            RETURN (FALSE);
         END_IF;
      END_IF;
      RETURN (TRUE);
   END_IF;
   RETURN (FALSE);
END_IF;
IF 'REAL_INTERVAL_FROM_MIN' IN types1 THEN
   IF 'ELEMENTARY_SPACE' IN types2 THEN
      es_val := spc2\elementary_space.space_id;
      RETURN ((es_val = es_numbers) OR (es_val = es_reals));
   END_IF;
   IF 'REAL_INTERVAL_FROM_MIN' IN types2 THEN
      bnd1 := spc1\real_interval_from_min.min;
      bnd2 := spc2\real_interval_from_min.min;
      RETURN ((bnd2 < bnd1) OR (bnd2 = bnd1) AND (min_included(spc2) OR NOT
    min_included(spc1)));
   END_IF;
   RETURN (FALSE);
END_IF;
IF 'REAL_INTERVAL_TO_MAX' IN types1 THEN
   IF 'ELEMENTARY_SPACE' IN types2 THEN
      es_val := spc2\elementary_space.space_id;
      RETURN ((es_val = es_numbers) OR (es_val = es_reals));
   END_IF;
   IF 'REAL_INTERVAL_TO_MAX' IN types2 THEN
      bnd1 := spc1\real_interval_to_max.max;
      bnd2 := spc2\real_interval_to_max.max;
      RETURN ((bnd2 > bnd1) OR (bnd2 = bnd1) AND (max_included(spc2) OR NOT
    max_included(spc1)));
   END_IF;
   RETURN (FALSE);
END_IF;
IF 'CARTESIAN_COMPLEX_NUMBER_REGION' IN types1 THEN
```

```
      IF 'ELEMENTARY_SPACE' IN types2 THEN
         es_val := spc2\elementary_space.space_id;
         RETURN ((es_val = es_numbers) OR (es_val = es_complex_numbers));
      END_IF;
      IF 'CARTESIAN_COMPLEX_NUMBER_REGION' IN types2 THEN
        RETURN
       (subspace_of(spc1\cartesian_complex_number_region.real_constraint,
       spc2\cartesian_complex_number_region.real_constraint) AND
       subspace_of(spc1\cartesian_complex_number_region.imag_constraint,
       spc2\cartesian_complex_number_region.imag_constraint));
      END_IF;
      IF 'POLAR_COMPLEX_NUMBER_REGION' IN types2 THEN
       RETURN (subspace_of(enclose_cregion_in_pregion(spc1,
       spc2\polar_complex_number_region.centre), spc2));
      END_IF;
      RETURN (FALSE);
   END_IF;
   IF 'POLAR_COMPLEX_NUMBER_REGION' IN types1 THEN
      IF 'ELEMENTARY_SPACE' IN types2 THEN
         es_val := spc2\elementary_space.space_id;
         RETURN ((es_val = es_numbers) OR (es_val = es_complex_numbers));
      END_IF;
      IF 'CARTESIAN_COMPLEX_NUMBER_REGION' IN types2 THEN
         RETURN (subspace_of(enclose_pregion_in_cregion(spc1), spc2));
      END_IF;
      IF 'POLAR_COMPLEX_NUMBER_REGION' IN types2 THEN
         prgn1 := spc1;
         prgn2 := spc2;
         IF prgn1.centre = prgn2.centre THEN
            IF prgn2.direction_constraint.max > 3.14159 THEN
              aitv := make_finite_real_interval(-3.14159, open,
              prgn2.direction_constraint.max - 2.00000 * 3.14159,
              prgn2.direction_constraint.max_closure);
              RETURN (subspace_of(prgn1.distance_constraint,
              prgn2.distance_constraint) AND
              (subspace_of(prgn1.direction_constraint,
              prgn2.direction_constraint) OR
              subspace_of(prgn1.direction_constraint, aitv)));
            ELSE
                RETURN (subspace_of(prgn1.distance_constraint,
              prgn2.distance_constraint) AND
              subspace_of(prgn1.direction_constraint,
              prgn2.direction_constraint));
            END_IF;
         END_IF;
              RETURN (subspace_of(enclose_pregion_in_pregion(prgn1,
              prgn2.centre), prgn2));
      END_IF;
      RETURN (FALSE);
   END_IF;
   IF 'FINITE_SPACE' IN types1 THEN
      cum := TRUE;
      REPEAT i := 1 TO SIZEOF(spc1\finite_space.members);
         cum := cum AND member_of(spc1\finite_space.members[i], spc2);
         IF cum = FALSE THEN
            RETURN (FALSE);
         END_IF;
      END_REPEAT;
      RETURN (cum);
```

```
END_IF;
IF 'PRODUCT_SPACE' IN types1 THEN
   IF 'PRODUCT_SPACE' IN types2 THEN
      IF space_dimension(spc1) = space_dimension(spc2) THEN
         cum := TRUE;
         REPEAT i := 1 TO space_dimension(spc1);
                cum := cum AND subspace_of(factor_space(spc1, i),
           factor_space(spc2, i));
            IF cum = FALSE THEN
               RETURN (FALSE);
            END_IF;
         END_REPEAT;
         RETURN (cum);
      END_IF;
   END_IF;
   IF 'EXTENDED_TUPLE_SPACE' IN types2 THEN
      IF space_dimension(spc1) >= space_dimension(spc2) THEN
         cum := TRUE;
         REPEAT i := 1 TO space_dimension(spc1);
                cum := cum AND subspace_of(factor_space(spc1, i),
        factor_space(spc2, i));
            IF cum = FALSE THEN
               RETURN (FALSE);
            END_IF;
         END_REPEAT;
         RETURN (cum);
      END_IF;
   END_IF;
   RETURN (FALSE);
END_IF;
IF 'EXTENDED_TUPLE_SPACE' IN types1 THEN
   IF 'EXTENDED_TUPLE_SPACE' IN types2 THEN
      n := space_dimension(spc1);
      IF n < space_dimension(spc2) THEN
         n := space_dimension(spc2);
      END_IF;
      cum := TRUE;
      REPEAT i := 1 TO n + 1;
             cum := cum AND subspace_of(factor_space(spc1, i),
       factor_space(spc2, i));
         IF cum = FALSE THEN
            RETURN (FALSE);
         END_IF;
      END_REPEAT;
      RETURN (cum);
   END_IF;
   RETURN (FALSE);
END_IF;
IF 'FUNCTION_SPACE' IN types1 THEN
   IF 'ELEMENTARY_SPACE' IN types2 THEN
      RETURN (spc2\elementary_space.space_id = es_maths_functions);
   END_IF;
   IF 'FUNCTION_SPACE' IN types2 THEN
      cum := TRUE;
      sp1 := spc1\function_space.domain_argument;
      sp2 := spc2\function_space.domain_argument;
      CASE spc1\function_space.domain_constraint OF
         sc_equal :
               BEGIN
```

```
                CASE spc2\function_space.domain_constraint OF
                   sc_equal :

                         cum := cum AND equal_maths_spaces(sp1, sp2);
                   sc_subspace :
                         cum := cum AND subspace_of(sp1, sp2);
                   sc_member :
                         cum := cum AND member_of(sp1, sp2);
                END_CASE;
             END;
          sc_subspace :
             BEGIN
                CASE spc2\function_space.domain_constraint OF
                   sc_equal :
                         RETURN (FALSE);
                   sc_subspace :
                         cum := cum AND subspace_of(sp1, sp2);
                   sc_member :
                         BEGIN
                            IF NOT member_of(sp1, sp2) THEN
                               RETURN (FALSE);
                            END_IF;
                            cum := UNKNOWN;
                         END;
                END_CASE;
             END;
          sc_member :
             BEGIN
                CASE spc2\function_space.domain_constraint OF
                   sc_equal :
                   cum := (cum AND space_is_singleton(sp1)) AND
                   equal_maths_spaces(singleton_member_of(sp1), sp2);
                   sc_subspace :
                         BEGIN
                            IF NOT member_of(sp2, sp1) THEN
                               RETURN (FALSE);
                            END_IF;
                            cum := UNKNOWN;
                         END;
                   sc_member :
                         cum := cum AND subspace_of(sp1, sp2);
                END_CASE;
             END;
       END_CASE;
       IF cum = FALSE THEN
          RETURN (FALSE);
       END_IF;
       sp1 := spc1\function_space.range_argument;
       sp2 := spc2\function_space.range_argument;
       CASE spc1\function_space.range_constraint OF
          sc_equal :
             BEGIN
                CASE spc2\function_space.range_constraint OF
                   sc_equal :
                         cum := cum AND equal_maths_spaces(sp1, sp2);
                   sc_subspace :
                         cum := cum AND subspace_of(sp1, sp2);
                   sc_member :
                         cum := cum AND member_of(sp1, sp2);
```

```
                  END_CASE;
               END;
         sc_subspace :
               BEGIN
                  CASE spc2\function_space.domain_constraint OF
                     sc_equal :
                           RETURN (FALSE);
                     sc_subspace :
                           cum := cum AND subspace_of(sp1, sp2);
                     sc_member :
                           BEGIN
                              IF NOT member_of(sp1, sp2) THEN
                                 RETURN (FALSE);
                              END_IF;
                              cum := UNKNOWN;
                           END;
                  END_CASE;
               END;
         sc_member :
               BEGIN
                  CASE spc2\function_space.domain_constraint OF
                     sc_equal :
                        cum := (cum AND space_is_singleton(sp1)) AND
                        equal_maths_spaces(singleton_member_of(sp1), sp2);
                     sc_subspace :
                           BEGIN
                              IF NOT member_of(sp2, sp1) THEN
                                 RETURN (FALSE);
                              END_IF;
                              cum := UNKNOWN;
                           END;
                     sc_member :
                           cum := cum AND subspace_of(sp1, sp2);
                  END_CASE;
               END;
         END_CASE;
         RETURN (cum);
      END_IF;
      RETURN (FALSE);
   END_IF;
   RETURN (UNKNOWN);
END_FUNCTION;

FUNCTION subspace_of_es
   (spc : maths_space;
    es : elementary_space_enumerators ) : LOGICAL;
LOCAL
   types : SET OF STRING := stripped_typeof(spc);
END_LOCAL;
   IF NOT EXISTS(spc) OR NOT EXISTS(es) THEN
      RETURN (FALSE);
   END_IF;
   IF 'ELEMENTARY_SPACE' IN types THEN
      RETURN (es_subspace_of_es(spc\elementary_space.space_id, es));
   END_IF;
   IF 'FINITE_SPACE' IN types THEN
      RETURN (all_members_of_es(spc\finite_space.members, es));
   END_IF;
   CASE es OF
```

```
        es_numbers :
          RETURN (((((((('FINITE_INTEGER_INTERVAL' IN types) OR
         ('INTEGER_INTERVAL_FROM_MIN' IN types)) OR ('INTEGER_INTERVAL_TO_MAX'
         IN types)) OR ('FINITE_REAL_INTERVAL' IN types)) OR
         ('REAL_INTERVAL_FROM_MIN' IN types)) OR ('REAL_INTERVAL_TO_MAX' IN
         types)) OR ('CARTESIAN_COMPLEX_NUMBER_REGION' IN types)) OR
         ('POLAR_COMPLEX_NUMBER_REGION' IN types));
        es_complex_numbers :
          RETURN (('CARTESIAN_COMPLEX_NUMBER_REGION' IN types) OR
         ('POLAR_COMPLEX_NUMBER_REGION' IN types));
        es_reals :
          RETURN ((('FINITE_REAL_INTERVAL' IN types) OR
         ('REAL_INTERVAL_FROM_MIN' IN types)) OR ('REAL_INTERVAL_TO_MAX' IN
         types));
        es_integers :
          RETURN ((('FINITE_INTEGER_INTERVAL' IN types) OR
         ('INTEGER_INTERVAL_FROM_MIN' IN types)) OR ('INTEGER_INTERVAL_TO_MAX'
         IN types));
        es_logicals :
              RETURN (FALSE);
        es_booleans :
              RETURN (FALSE);
        es_strings :
              RETURN (FALSE);
        es_binarys :
              RETURN (FALSE);
        es_maths_spaces :
              RETURN (FALSE);
        es_maths_functions :
              RETURN ('FUNCTION_SPACE' IN types);
        es_generics :
              RETURN (TRUE);
    END_CASE;
    RETURN (UNKNOWN);
 END_FUNCTION;

 FUNCTION substitute(expr : generic_expression;
                 vars : LIST [1:?] OF generic_variable;
                 vals : LIST [1:?] OF maths_value) : generic_expression;
LOCAL
  types : SET OF STRING := stripped_typeof(expr);
  opnds : LIST OF generic_expression;
  op1, op2 : generic_expression;
  qvars : LIST OF generic_variable;
  srcdom : maths_space_or_function;
  prpfun : LIST [1:?] OF maths_function;
  finfun : maths_function_select;
END_LOCAL;
IF SIZEOF (vars) <> SIZEOF (vals) THEN  RETURN (?);  END_IF;
IF 'GENERIC_LITERAL' IN types THEN  RETURN (expr);  END_IF;
IF 'GENERIC_VARIABLE' IN types THEN
  REPEAT i := 1 TO SIZEOF (vars);
    IF expr :=: vars[i] THEN  RETURN (vals[i]);  END_IF;
  END_REPEAT;
  RETURN (expr);
END_IF;
IF 'QUANTIFIER_EXPRESSION' IN types THEN
  qvars := expr\quantifier_expression.variables;
  -- Variables subject to a quantifier do not participate in this kind of
```

```
    -- substitution process.
    REPEAT i := SIZEOF (vars) TO 1 BY -1;
      IF vars[i] IN qvars THEN
        REMOVE (vars, i);
        REMOVE (vals, i);
      END_IF;
    END_REPEAT;
    opnds := expr\multiple_arity_generic_expression.operands;
    REPEAT i := 1 TO SIZEOF (opnds);
      IF NOT (opnds[i] IN qvars) THEN
        expr\multiple_arity_generic_expression.operands[i] :=
          substitute(opnds[i],vars,vals);
        -- This technique will not work on subtypes of quantifier_expression
        -- which derive their operands from other attributes!
      END_IF;
    END_REPEAT;
    RETURN (expr);  -- operands modified!
  END_IF;
  IF 'UNARY_GENERIC_EXPRESSION' IN types THEN
    op1 := expr\unary_generic_expression.operand;
    expr\unary_generic_expression.operand := substitute(op1, vars, vals);
    -- This technique will not work on subtypes of unary_generic_expression
    -- which derive their operands from other attributes!
  END_IF;
  IF 'BINARY_GENERIC_EXPRESSION' IN types THEN
    op1 := expr\binary_generic_expression.operands[1];
    expr\binary_generic_expression.operands[1] := substitute(op1, vars, vals);
    op2 := expr\binary_generic_expression.operands[2];
    expr\binary_generic_expression.operands[2] := substitute(op2, vars, vals);
    -- This technique will not work on subtypes of binary_generic_expression
    -- which derive their operands from other attributes!
  END_IF;
  IF 'PARALLEL_COMPOSED_FUNCTION' IN types THEN
    -- Subtype of multiple_arity_generic_expression which derives its operands.
    srcdom := expr\parallel_composed_function.source_of_domain;
    prpfun := expr\parallel_composed_function.prep_functions;
    finfun := expr\parallel_composed_function.final_function;
    srcdom := substitute(srcdom,vars,vals);
    REPEAT i := 1 TO SIZEOF (prpfun);
      prpfun[i] := substitute(prpfun[i],vars,vals);
    END_REPEAT;
    IF 'MATHS_FUNCTION' IN stripped_typeof(finfun) THEN
      finfun := substitute(finfun,vars,vals);
    END_IF;
    RETURN (make_parallel_composed_function(srcdom,prpfun,finfun));
  END_IF;
  IF 'MULTIPLE_ARITY_GENERIC_EXPRESSION' IN types THEN
    opnds := expr\multiple_arity_generic_expression.operands;
    REPEAT i := 1 TO SIZEOF (opnds);
      expr\multiple_arity_generic_expression.operands[i] :=
        substitute(opnds[i],vars,vals);
        -- This technique will not work on subtypes of multiple_arity_generic_
        -- expression which derive their operands from other attributes!
    END_REPEAT;
  END_IF;
  RETURN (expr);
END_FUNCTION;  -- substitute

   FUNCTION using_items
```

```
   (item : founded_item_select;
    checked_items : SET OF founded_item_select ) : SET OF founded_item_select;
LOCAL
   new_check_items : SET OF founded_item_select;
   result_items : SET OF founded_item_select;
   next_items : SET OF founded_item_select;
END_LOCAL;
   result_items := [];
   new_check_items := checked_items + item;
      next_items := QUERY (z <* bag_to_set(USEDIN(item, ''))|
('ENGINEERING_PROPERTIES_SCHEMA.REPRESENTATION_ITEM' IN TYPEOF(z)) OR
('ENGINEERING_PROPERTIES_SCHEMA.FOUNDED_ITEM' IN TYPEOF(z)));
   IF SIZEOF(next_items) > 0 THEN
      REPEAT i := 1 TO HIINDEX(next_items);
         IF NOT (next_items[i] IN new_check_items) THEN
               result_items := result_items + next_items[i] +
using_items(next_items[i], new_check_items);
         END_IF;
      END_REPEAT;
   END_IF;
   RETURN (result_items);
END_FUNCTION;

FUNCTION using_representations
   (item : founded_item_select ) : SET OF representation;
LOCAL
   results : SET OF representation;
   result_bag : BAG OF representation;
   intermediate_items : SET OF founded_item_select;
END_LOCAL;
   results := [];
    result_bag := USEDIN(item,
'ENGINEERING_PROPERTIES_SCHEMA.REPRESENTATION.ITEMS');
   IF SIZEOF(result_bag) > 0 THEN
      REPEAT i := 1 TO HIINDEX(result_bag);
         results := results + result_bag[i];
      END_REPEAT;
   END_IF;
   intermediate_items := using_items(item, []);
   IF SIZEOF(intermediate_items) > 0 THEN
      REPEAT i := 1 TO HIINDEX(intermediate_items);
       result_bag := USEDIN(intermediate_items[i],
    'ENGINEERING_PROPERTIES_SCHEMA.REPRESENTATION.ITEMS');
         IF SIZEOF(result_bag) > 0 THEN
            REPEAT j := 1 TO HIINDEX(result_bag);
               results := results + result_bag[j];
            END_REPEAT;
         END_IF;
      END_REPEAT;
   END_IF;
   RETURN (results);
END_FUNCTION;

FUNCTION valid_calendar_date
   (date : calendar_date ) : LOGICAL;
   CASE date.month_component OF
      1 :
            RETURN ((1 <= date.day_component) AND (date.day_component <= 31));
      2 :
```

```
               BEGIN
                  IF leap_year(date.year_component) THEN
                    RETURN ((1 <= date.day_component) AND (date.day_component <=
                    29));
                  ELSE
                  RETURN ((1 <= date.day_component) AND (date.day_component <=
                 28));
                  END_IF;
               END;
         3 :
               RETURN ((1 <= date.day_component) AND (date.day_component <= 31));
         4 :
               RETURN ((1 <= date.day_component) AND (date.day_component <= 30));
         5 :
               RETURN ((1 <= date.day_component) AND (date.day_component <= 31));
         6 :
               RETURN ((1 <= date.day_component) AND (date.day_component <= 30));
         7 :
               RETURN ((1 <= date.day_component) AND (date.day_component <= 31));
         8 :
               RETURN ((1 <= date.day_component) AND (date.day_component <= 31));
         9 :
               RETURN ((1 <= date.day_component) AND (date.day_component <= 30));
         10 :
               RETURN ((1 <= date.day_component) AND (date.day_component <= 31));
         11 :
               RETURN ((1 <= date.day_component) AND (date.day_component <= 30));
         12 :
               RETURN ((1 <= date.day_component) AND (date.day_component <= 31));
      END_CASE;
      RETURN (FALSE);
   END_FUNCTION;

   FUNCTION valid_measure_value
      (m : measure_value ) : BOOLEAN;
      IF 'REAL' IN TYPEOF(m) THEN
         RETURN (m > 0.00000);
      ELSE
         IF 'INTEGER' IN TYPEOF(m) THEN
            RETURN (m > 0);
         ELSE
            RETURN (TRUE);
         END_IF;
      END_IF;
   END_FUNCTION;

   FUNCTION valid_time
      (time : local_time ) : BOOLEAN;
      IF EXISTS(time.second_component) THEN
         RETURN (EXISTS(time.minute_component));
      ELSE
         RETURN (TRUE);
      END_IF;
   END_FUNCTION;

   FUNCTION valid_units
      (m : measure_with_unit ) : BOOLEAN;
      IF 'ENGINEERING_PROPERTIES_SCHEMA.LENGTH_MEASURE' IN
TYPEOF(m.value_component) THEN
```

```
   IF derive_dimensional_exponents(m.unit_component) <>
 dimensional_exponents(1.00000, 0.00000, 0.00000, 0.00000, 0.00000, 0.00000,
 0.00000) THEN
      RETURN (FALSE);
   END_IF;
END_IF;
       IF 'ENGINEERING_PROPERTIES_SCHEMA.MASS_MEASURE' IN
 TYPEOF(m.value_component) THEN
       IF derive_dimensional_exponents(m.unit_component) <>
       dimensional_exponents(0.00000, 1.00000, 0.00000, 0.00000, 0.00000,
       0.00000, 0.00000) THEN
      RETURN (FALSE);
   END_IF;
END_IF;
       IF 'ENGINEERING_PROPERTIES_SCHEMA.TIME_MEASURE' IN
 TYPEOF(m.value_component) THEN
   IF derive_dimensional_exponents(m.unit_component) <>
   dimensional_exponents(0.00000, 0.00000, 1.00000, 0.00000, 0.00000,
   0.00000, 0.00000) THEN
      RETURN (FALSE);
   END_IF;
END_IF;
       IF 'ENGINEERING_PROPERTIES_SCHEMA.ELECTRIC_CURRENT_MEASURE' IN
 TYPEOF(m.value_component) THEN
   IF derive_dimensional_exponents(m.unit_component) <>
   dimensional_exponents(0.00000, 0.00000, 0.00000, 1.00000, 0.00000,
   0.00000, 0.00000) THEN
      RETURN (FALSE);
   END_IF;
END_IF;
       IF 'ENGINEERING_PROPERTIES_SCHEMA.THERMODYNAMIC_TEMPERATURE_MEASURE'
 IN TYPEOF(m.value_component) THEN
   IF derive_dimensional_exponents(m.unit_component) <>
   dimensional_exponents(0.00000, 0.00000, 0.00000, 0.00000, 1.00000,
   0.00000, 0.00000) THEN
      RETURN (FALSE);
   END_IF;
END_IF;
       IF 'ENGINEERING_PROPERTIES_SCHEMA.CELSIUS_TEMPERATURE_MEASURE' IN
 TYPEOF(m.value_component) THEN
   IF derive_dimensional_exponents(m.unit_component) <>
   dimensional_exponents(0.00000, 0.00000, 0.00000, 0.00000, 1.00000,
   0.00000, 0.00000) THEN
      RETURN (FALSE);
   END_IF;
END_IF;
       IF 'ENGINEERING_PROPERTIES_SCHEMA.AMOUNT_OF_SUBSTANCE_MEASURE' IN
 TYPEOF(m.value_component) THEN
   IF derive_dimensional_exponents(m.unit_component) <>
   dimensional_exponents(0.00000, 0.00000, 0.00000, 0.00000, 0.00000,
   1.00000, 0.00000) THEN
      RETURN (FALSE);
   END_IF;
END_IF;
       IF 'ENGINEERING_PROPERTIES_SCHEMA.LUMINOUS_INTENSITY_MEASURE' IN
 TYPEOF(m.value_component) THEN
   IF derive_dimensional_exponents(m.unit_component) <>
   dimensional_exponents(0.00000, 0.00000, 0.00000, 0.00000, 0.00000,
   0.00000, 1.00000) THEN
```

```
      RETURN (FALSE);
   END_IF;
END_IF;
       IF 'ENGINEERING_PROPERTIES_SCHEMA.PLANE_ANGLE_MEASURE' IN
 TYPEOF(m.value_component) THEN
   IF derive_dimensional_exponents(m.unit_component) <>
   dimensional_exponents(0.00000, 0.00000, 0.00000, 0.00000, 0.00000,
   0.00000, 0.00000) THEN
      RETURN (FALSE);
   END_IF;
END_IF;
       IF 'ENGINEERING_PROPERTIES_SCHEMA.SOLID_ANGLE_MEASURE' IN
 TYPEOF(m.value_component) THEN
   IF derive_dimensional_exponents(m.unit_component) <>
   dimensional_exponents(0.00000, 0.00000, 0.00000, 0.00000, 0.00000,
   0.00000, 0.00000) THEN
      RETURN (FALSE);
   END_IF;
END_IF;
       IF 'ENGINEERING_PROPERTIES_SCHEMA.AREA_MEASURE' IN
 TYPEOF(m.value_component) THEN
   IF derive_dimensional_exponents(m.unit_component) <>
   dimensional_exponents(2.00000, 0.00000, 0.00000, 0.00000, 0.00000,
   0.00000, 0.00000) THEN
      RETURN (FALSE);
   END_IF;
END_IF;
       IF 'ENGINEERING_PROPERTIES_SCHEMA.VOLUME_MEASURE' IN
 TYPEOF(m.value_component) THEN
   IF derive_dimensional_exponents(m.unit_component) <>
   dimensional_exponents(3.00000, 0.00000, 0.00000, 0.00000, 0.00000,
   0.00000, 0.00000) THEN
      RETURN (FALSE);
   END_IF;
END_IF;
       IF 'ENGINEERING_PROPERTIES_SCHEMA.RATIO_MEASURE' IN
 TYPEOF(m.value_component) THEN
   IF derive_dimensional_exponents(m.unit_component) <>
   dimensional_exponents(0.00000, 0.00000, 0.00000, 0.00000, 0.00000,
   0.00000, 0.00000) THEN
      RETURN (FALSE);
   END_IF;
END_IF;
       IF 'ENGINEERING_PROPERTIES_SCHEMA.POSITIVE_LENGTH_MEASURE' IN
 TYPEOF(m.value_component) THEN
   IF derive_dimensional_exponents(m.unit_component) <>
   dimensional_exponents(1.00000, 0.00000, 0.00000, 0.00000, 0.00000,
   0.00000, 0.00000) THEN
      RETURN (FALSE);
   END_IF;
END_IF;
       IF 'ENGINEERING_PROPERTIES_SCHEMA.POSITIVE_PLANE_ANGLE_MEASURE' IN
 TYPEOF(m.value_component) THEN
   IF derive_dimensional_exponents(m.unit_component) <>
   dimensional_exponents(0.00000, 0.00000, 0.00000, 0.00000, 0.00000,
   0.00000, 0.00000) THEN
      RETURN (FALSE);
   END_IF;
END_IF;
```

```
      IF 'ENGINEERING_PROPERTIES_SCHEMA.ACCELERATION_MEASURE' IN
 TYPEOF(m.value_component) THEN
   IF derive_dimensional_exponents(m.unit_component) <>
   dimensional_exponents(1.00000, 0.00000, -2.00000, 0.00000, 0.00000,
   0.00000, 0.00000) THEN
      RETURN (FALSE);
   END_IF;
END_IF;
      IF 'ENGINEERING_PROPERTIES_SCHEMA.CAPACITANCE_MEASURE' IN
 TYPEOF(m.value_component) THEN
   IF derive_dimensional_exponents(m.unit_component) <>
   dimensional_exponents(-2.00000, -1.00000, 4.00000, 1.00000, 0.00000,
   0.00000, 0.00000) THEN
      RETURN (FALSE);
   END_IF;
END_IF;
      IF 'ENGINEERING_PROPERTIES_SCHEMA.ELECTRIC_CHARGE_MEASURE' IN
 TYPEOF(m.value_component) THEN
   IF derive_dimensional_exponents(m.unit_component) <>
   dimensional_exponents(0.00000, 0.00000, 1.00000, 1.00000, 0.00000,
   0.00000, 0.00000) THEN
      RETURN (FALSE);
   END_IF;
END_IF;
      IF 'ENGINEERING_PROPERTIES_SCHEMA.CONDUCTANCE_MEASURE' IN
 TYPEOF(m.value_component) THEN
   IF derive_dimensional_exponents(m.unit_component) <>
   dimensional_exponents(-2.00000, -1.00000, 3.00000, 2.00000, 0.00000,
   0.00000, 0.00000) THEN
      RETURN (FALSE);
   END_IF;
END_IF;
      IF 'ENGINEERING_PROPERTIES_SCHEMA.ELECTRIC_POTENTIAL_MEASURE' IN
 TYPEOF(m.value_component) THEN
   IF derive_dimensional_exponents(m.unit_component) <>
   dimensional_exponents(2.00000, 1.00000, -3.00000, -1.00000, 0.00000,
   0.00000, 0.00000) THEN
      RETURN (FALSE);
   END_IF;
END_IF;
      IF 'ENGINEERING_PROPERTIES_SCHEMA.ENERGY_MEASURE' IN
 TYPEOF(m.value_component) THEN
   IF derive_dimensional_exponents(m.unit_component) <>
   dimensional_exponents(2.00000, 1.00000, -2.00000, 0.00000, 0.00000,
   0.00000, 0.00000) THEN
      RETURN (FALSE);
   END_IF;
END_IF;
      IF 'ENGINEERING_PROPERTIES_SCHEMA.FORCE_MEASURE' IN
 TYPEOF(m.value_component) THEN
   IF derive_dimensional_exponents(m.unit_component) <>
   dimensional_exponents(1.00000, 1.00000, -2.00000, 0.00000, 0.00000,
   0.00000, 0.00000) THEN
      RETURN (FALSE);
   END_IF;
END_IF;
      IF 'ENGINEERING_PROPERTIES_SCHEMA.FREQUENCY_MEASURE' IN
 TYPEOF(m.value_component) THEN
```

```
      IF derive_dimensional_exponents(m.unit_component) <>
      dimensional_exponents(0.00000, 0.00000, -1.00000, 0.00000, 0.00000,
      0.00000, 0.00000) THEN
         RETURN (FALSE);
      END_IF;
   END_IF;
          IF 'ENGINEERING_PROPERTIES_SCHEMA.ILLUMINANCE_MEASURE' IN
    TYPEOF(m.value_component) THEN
      IF derive_dimensional_exponents(m.unit_component) <>
      dimensional_exponents(-2.00000, 0.00000, 0.00000, 0.00000, 0.00000,
      0.00000, 1.00000) THEN
         RETURN (FALSE);
      END_IF;
   END_IF;
          IF 'ENGINEERING_PROPERTIES_SCHEMA.INDUCTANCE_MEASURE' IN
    TYPEOF(m.value_component) THEN
      IF derive_dimensional_exponents(m.unit_component) <>
      dimensional_exponents(2.00000, 1.00000, -2.00000, -2.00000, 0.00000,
      0.00000, 0.00000) THEN
         RETURN (FALSE);
      END_IF;
   END_IF;
          IF 'ENGINEERING_PROPERTIES_SCHEMA.LUMINOUS_FLUX_MEASURE' IN
    TYPEOF(m.value_component) THEN
      IF derive_dimensional_exponents(m.unit_component) <>
      dimensional_exponents(0.00000, 0.00000, 0.00000, 0.00000, 0.00000,
      0.00000, 1.00000) THEN
         RETURN (FALSE);
      END_IF;
   END_IF;
          IF 'ENGINEERING_PROPERTIES_SCHEMA.MAGNETIC_FLUX_MEASURE' IN
    TYPEOF(m.value_component) THEN
      IF derive_dimensional_exponents(m.unit_component) <>
      dimensional_exponents(2.00000, 1.00000, -2.00000, -1.00000, 0.00000,
      0.00000, 0.00000) THEN
         RETURN (FALSE);
      END_IF;
   END_IF;
          IF 'ENGINEERING_PROPERTIES_SCHEMA.MAGNETIC_FLUX_DENSITY_MEASURE' IN
    TYPEOF(m.value_component) THEN
      IF derive_dimensional_exponents(m.unit_component) <>
      dimensional_exponents(0.00000, 1.00000, -2.00000, -1.00000, 0.00000,
      0.00000, 0.00000) THEN
         RETURN (FALSE);
      END_IF;
   END_IF;
          IF 'ENGINEERING_PROPERTIES_SCHEMA.POWER_MEASURE' IN
    TYPEOF(m.value_component) THEN
      IF derive_dimensional_exponents(m.unit_component) <>
      dimensional_exponents(2.00000, 1.00000, -3.00000, 0.00000, 0.00000,
      0.00000, 0.00000) THEN
         RETURN (FALSE);
      END_IF;
   END_IF;
          IF 'ENGINEERING_PROPERTIES_SCHEMA.PRESSURE_MEASURE' IN
    TYPEOF(m.value_component) THEN
      IF derive_dimensional_exponents(m.unit_component) <>
      dimensional_exponents(-1.00000, 1.00000, -2.00000, 0.00000, 0.00000,
      0.00000, 0.00000) THEN
```

```
         RETURN (FALSE);
      END_IF;
   END_IF;
          IF 'ENGINEERING_PROPERTIES_SCHEMA.RESISTANCE_MEASURE' IN
    TYPEOF(m.value_component) THEN
      IF derive_dimensional_exponents(m.unit_component) <>
      dimensional_exponents(2.00000, 1.00000, -3.00000, -2.00000, 0.00000,
      0.00000, 0.00000) THEN
         RETURN (FALSE);
      END_IF;
   END_IF;
          IF 'ENGINEERING_PROPERTIES_SCHEMA.VELOCITY_MEASURE' IN
    TYPEOF(m.value_component) THEN
      IF derive_dimensional_exponents(m.unit_component) <>
      dimensional_exponents(1.00000, 0.00000, -1.00000, 0.00000, 0.00000,
      0.00000, 0.00000) THEN
         RETURN (FALSE);
      END_IF;
   END_IF;
          IF 'ENGINEERING_PROPERTIES_SCHEMA.RADIOACTIVITY_MEASURE' IN
    TYPEOF(m.value_component) THEN
      IF derive_dimensional_exponents(m.unit_component) <>
      dimensional_exponents(0.00000, 0.00000, -1.00000, 0.00000, 0.00000,
      0.00000, 0.00000) THEN
         RETURN (FALSE);
      END_IF;
   END_IF;
          IF 'ENGINEERING_PROPERTIES_SCHEMA.ABSORBED_DOSE_MEASURE' IN
    TYPEOF(m.value_component) THEN
      IF derive_dimensional_exponents(m.unit_component) <>
      dimensional_exponents(2.00000, 0.00000, -2.00000, 0.00000, 0.00000,
      0.00000, 0.00000) THEN
         RETURN (FALSE);
      END_IF;
   END_IF;
          IF 'ENGINEERING_PROPERTIES_SCHEMA.DOSE_EQUIVALENT_MEASURE' IN
    TYPEOF(m.value_component) THEN
      IF derive_dimensional_exponents(m.unit_component) <>
      dimensional_exponents(2.00000, 0.00000, -2.00000, 0.00000, 0.00000,
      0.00000, 0.00000) THEN
         RETURN (FALSE);
      END_IF;
   END_IF;
   RETURN (TRUE);
END_FUNCTION;

FUNCTION values_space_of
   (expr : generic_expression ) : maths_space;
LOCAL
   e_prefix : STRING := 'ENGINEERING_PROPERTIES_SCHEMA.';
   typenames : SET OF STRING := TYPEOF(expr);
END_LOCAL;
   IF schema_prefix + 'MATHS_VARIABLE' IN typenames THEN
      RETURN (expr\maths_variable.values_space);
   END_IF;
   IF e_prefix + 'EXPRESSION' IN typenames THEN
      IF e_prefix + 'NUMERIC_EXPRESSION' IN typenames THEN
         IF expr\numeric_expression.is_int THEN
            IF e_prefix + 'INT_LITERAL' IN typenames THEN
```

```
            RETURN (make_finite_space([ expr\int_literal.the_value ]));
         ELSE
            RETURN (the_integers);
         END_IF;
      ELSE
         IF e_prefix + 'REAL_LITERAL' IN typenames THEN
            RETURN (make_finite_space([ expr\real_literal.the_value ]));
         ELSE
            RETURN (the_reals);
         END_IF;
      END_IF;
   END_IF;
   IF e_prefix + 'BOOLEAN_EXPRESSION' IN typenames THEN
      IF e_prefix + 'BOOLEAN_LITERAL' IN typenames THEN
         RETURN (make_finite_space([ expr\boolean_literal.the_value ]));
      ELSE
         RETURN (the_booleans);
      END_IF;
   END_IF;
   IF e_prefix + 'STRING_EXPRESSION' IN typenames THEN
      IF e_prefix + 'STRING_LITERAL' IN typenames THEN
         RETURN (make_finite_space([ expr\string_literal.the_value ]));
      ELSE
         RETURN (the_strings);
      END_IF;
   END_IF;
   RETURN (?);
END_IF;
IF schema_prefix + 'MATHS_FUNCTION' IN typenames THEN
   IF expression_is_constant(expr) THEN
      RETURN (make_finite_space([ expr ]));
   ELSE
    RETURN (make_function_space(sc_equal, expr\maths_function.domain,
    sc_equal, expr\maths_function.range));
   END_IF;
END_IF;
IF schema_prefix + 'FUNCTION_APPLICATION' IN typenames THEN
   RETURN (expr\function_application.func.range);
END_IF;
IF schema_prefix + 'MATHS_SPACE' IN typenames THEN
   IF expression_is_constant(expr) THEN
      RETURN (make_finite_space([ expr ]));
   ELSE
      RETURN (make_elementary_space(es_maths_spaces));
   END_IF;
END_IF;
IF schema_prefix + 'DEPENDENT_VARIABLE_DEFINITION' IN typenames THEN
   RETURN (values_space_of(expr\unary_generic_expression.operand));
END_IF;
IF schema_prefix + 'COMPLEX_NUMBER_LITERAL' IN typenames THEN
   RETURN (make_finite_space([ expr ]));
END_IF;
IF schema_prefix + 'LOGICAL_LITERAL' IN typenames THEN
   RETURN (make_finite_space([ expr\logical_literal.lit_value ]));
END_IF;
IF schema_prefix + 'BINARY_LITERAL' IN typenames THEN
   RETURN (make_finite_space([ expr\binary_literal.lit_value ]));
END_IF;
IF schema_prefix + 'MATHS_ENUM_LITERAL' IN typenames THEN
```

```
      RETURN (make_finite_space([ expr\maths_enum_literal.lit_value ]));
   END_IF;
   IF schema_prefix + 'REAL_TUPLE_LITERAL' IN typenames THEN
      RETURN (make_finite_space([ expr\real_tuple_literal.lit_value ]));
   END_IF;
   IF schema_prefix + 'INTEGER_TUPLE_LITERAL' IN typenames THEN
      RETURN (make_finite_space([ expr\integer_tuple_literal.lit_value ]));
   END_IF;
   IF schema_prefix + 'ATOM_BASED_LITERAL' IN typenames THEN
      RETURN (make_finite_space([ expr\atom_based_literal.lit_value ]));
   END_IF;
   IF schema_prefix + 'MATHS_TUPLE_LITERAL' IN typenames THEN
      RETURN (make_finite_space([ expr\maths_tuple_literal.lit_value ]));
   END_IF;
   IF schema_prefix + 'PARTIAL_DERIVATIVE_EXPRESSION' IN typenames THEN
       RETURN
    (drop_numeric_constraints(values_space_of(expr\partial_derivative_expressi
    on.derivand)));
   END_IF;
   IF schema_prefix + 'DEFINITE_INTEGRAL_EXPRESSION' IN typenames THEN
      RETURN
    (drop_numeric_constraints(values_space_of(expr\definite_integral_expressio
    n.integrand)));
   END_IF;
   RETURN (?);
END_FUNCTION;

FUNCTION vector_difference
   (arg1 : vector_or_direction;
    arg2 : vector_or_direction ) : vector;
LOCAL
   result : vector;
   res : direction;
   vec1 : direction;
   vec2 : direction;
   mag : REAL;
   mag1 : REAL;
   mag2 : REAL;
   ndim : INTEGER;
END_LOCAL;
   IF (NOT EXISTS(arg1) OR NOT EXISTS(arg2)) OR (arg1.dim <> arg2.dim) THEN
      RETURN (?);
   ELSE
      BEGIN
         IF 'ENGINEERING_PROPERTIES_SCHEMA.VECTOR' IN TYPEOF(arg1) THEN
            mag1 := arg1.magnitude;
            vec1 := arg1.orientation;
         ELSE
            mag1 := 1.00000;
            vec1 := arg1;
         END_IF;
         IF 'ENGINEERING_PROPERTIES_SCHEMA.VECTOR' IN TYPEOF(arg2) THEN
            mag2 := arg2.magnitude;
            vec2 := arg2.orientation;
         ELSE
            mag2 := 1.00000;
            vec2 := arg2;
         END_IF;
         vec1 := normalise(vec1);
```

```
            vec2 := normalise(vec2);
            ndim := SIZEOF(vec1.direction_ratios);
            mag := 0.00000;
            res := dummy_gri || direction(vec1.direction_ratios);
            REPEAT i := 1 TO ndim;
             res.direction_ratios[i] := mag1 * vec1.direction_ratios[i] + mag2 *
             vec2.direction_ratios[i];
               mag := mag + res.direction_ratios[i] * res.direction_ratios[i];
            END_REPEAT;
            IF mag > 0.00000 THEN
               result := dummy_gri || vector(res, SQRT(mag));
            ELSE
               result := dummy_gri || vector(vec1, 0.00000);
            END_IF;
         END;
      END_IF;
      RETURN (result);
   END_FUNCTION;

(* *********************************
Rules in the schema engineering_properties_schema
********************************* *)

   RULE compatible_dimension FOR (cartesian_point, direction,
   representation_context, geometric_representation_context );
   WHERE
      WR1:
           SIZEOF(QUERY (x <* cartesian_point| (SIZEOF(QUERY (y <*
       geometric_representation_context| item_in_context(x, y) AND
       (HIINDEX(x.coordinates) <> y.coordinate_space_dimension))) > 0))) = 0;
      WR2:
          SIZEOF(QUERY (x <* direction| (SIZEOF(QUERY (y <*
       geometric_representation_context| item_in_context(x, y) AND
       (HIINDEX(x.direction_ratios) <> y.coordinate_space_dimension))) > 0))) =
       0;
   END_RULE;

   RULE dependent_instantiable_attribute_value_role FOR (attribute_value_role );
   WHERE
      WR1:
          SIZEOF(QUERY (a <* attribute_value_role| NOT (SIZEOF(USEDIN(a, '')) >
       0))) = 0;
   END_RULE;

   RULE dependent_instantiable_classification_role FOR (classification_role );
   WHERE
      WR1:
          SIZEOF(QUERY (c <* classification_role| NOT (SIZEOF(USEDIN(c, '')) >
       0))) = 0;
   END_RULE;

   RULE dependent_instantiable_identification_role FOR (identification_role );
   WHERE
      WR1:
          SIZEOF(QUERY (i <* identification_role| NOT (SIZEOF(USEDIN(i, '')) >
       0))) = 0;
   END_RULE;

   RULE plib_class_reference_requires_version FOR (externally_defined_class );
```

```
   WHERE
      WR1:
          SIZEOF(QUERY (edc <* externally_defined_class|
       ('ENGINEERING_PROPERTIES_SCHEMA.' + 'EXTERNAL_SOURCE' IN
       TYPEOF(edc.source)) AND (SIZEOF(QUERY (aei <* USEDIN(edc,
       'ENGINEERING_PROPERTIES_SCHEMA.APPLIED_EXTERNAL_IDENTIFICATION_ASSIGNMENT.
       ITEMS')| (aei.role.name = 'version'))) <> 1))) = 0;
      WR2:
          SIZEOF(QUERY (edc <* externally_defined_class|
       ('ENGINEERING_PROPERTIES_SCHEMA.' + 'EXTERNAL_SOURCE' IN
       TYPEOF(edc.source)) AND (SIZEOF(QUERY (aei <* USEDIN(edc,
       'ENGINEERING_PROPERTIES_SCHEMA.APPLIED_EXTERNAL_IDENTIFICATION_ASSIGNMENT.
       ITEMS')| (aei.role.name = 'version'))) > 0))) = 0;
   END_RULE;

   RULE plib_property_reference_requires_name_scope FOR
   (externally_defined_engineering_property );
   WHERE
      WR1:
          SIZEOF(QUERY (edep <* externally_defined_engineering_property|
       ('ENGINEERING_PROPERTIES_SCHEMA.' + 'EXTERNAL_SOURCE' IN
       TYPEOF(edep.source)) AND (SIZEOF(QUERY (edir <* USEDIN(edep,
       'ENGINEERING_PROPERTIES_SCHEMA.' + 'EXTERNALLY_DEFINED_ITEM_RELATIONSHIP.'
       + 'RELATING_ITEM')| ((edir.name = 'name scope') AND
       ('ENGINEERING_PROPERTIES_SCHEMA.' + 'EXTERNALLY_DEFINED_CLASS' IN
       TYPEOF(edir.related_item))) AND ('ENGINEERING_PROPERTIES_SCHEMA.' +
       'EXTERNAL_SOURCE' IN TYPEOF(edir.related_item.source)))) <> 1))) = 0;
   END_RULE;

    RULE plib_property_reference_requires_version FOR
    (externally_defined_engineering_property );
   WHERE
      WR1:
           SIZEOF(QUERY (edep <* externally_defined_engineering_property|
       ('ENGINEERING_PROPERTIES_SCHEMA.' + 'EXTERNAL_SOURCE' IN
       TYPEOF(edep.source)) AND (SIZEOF(QUERY (edir <* USEDIN(edep,
       'ENGINEERING_PROPERTIES_SCHEMA.APPLIED_EXTERNAL_IDENTIFICATION_ASSIGNMENT.
       ITEMS')| (edir.role.name = 'version'))) <> 1))) = 0
 END_RULE;
(* ********************************** *)
END_SCHEMA;
(* ********************************************************************** *)
```

Annex B
(normative)

AIM short names

Table B.1 provides the short names of entities specified in the AIM of this part of ISO 10303. Requirements on the use of short names are found in the implementation methods included in ISO 10303.

Table B.1 — Short names of entities specified in the AIM of this part of ISO 10303

Entity name	Short name
SQL_mappable_defined_function	SMDF
absorbed_dose_measure_with_unit	ADMWU
absorbed_dose_unit	ABDSUN
si_absorbed_dose_unit	SADU
acceleration_measure_with_unit	AMW0
acceleration_unit	ACCUNT
abs_function	ABSFNC
abstracted_expression_function	ABEXFN
acos_function	ACSFNC
action	ACTION
action_assignment	ACTASS
action_directive	ACTDRC
action_method	ACTMTH
action_method_relationship	ACMTRL
action_method_role	ACM0
action_method_with_associated_documents	AMWAD
action_method_with_associated_documents_constrained	AMWADC
action_property	ACTPRP
action_property_relationship	ACPRRL
action_property_representation	ACPRRP
action_relationship	ACTRLT

Entity name	Short name
action_request_assignment	ACRQAS
action_request_solution	ACRQSL
action_resource	ACTRSR
action_resource_relationship	ACRSRL
action_resource_requirement	ACRSRQ
action_resource_requirement_relationship	ARRR
action_resource_type	ACRSTY
action_status	ACTSTT
address	ADDRSS
amount_of_substance_measure_with_unit	AOSMWU
amount_of_substance_unit	AOSU
and_expression	ANDEXP
angularity_tolerance	ANGTLR
application_context	APPCNT
application_context_element	APCNEL
application_context_relationship	APCNRL
application_defined_function	APDFFN
applied_action_assignment	APACAS
applied_action_request_assignment	AARA
applied_approval_assignment	APAPAS
applied_certification_assignment	APCRAS
applied_contract_assignment	APCNAS
applied_date_and_time_assignment	ADATA
applied_date_assignment	APDTAS
applied_document_reference	APDCRF
applied_document_usage_contraint_assignment	ADU0
applied_effectivity_assignment	APEFAS
applied_event_occurrence_assignment	AEOA
applied_external_identification_assignment	AEIA

Entity name	Short name
applied_group_assignment	APGRAS
applied_identification_assignment	APIDAS
applied_location_assignment	APLCAS
applied_location_representation_assignment	ALRA
applied_organization_assignment	APORAS
applied_organizational_project_assignment	AOPA
applied_person_and_organization_assignment	APAOA
applied_person_assignment	APPRAS
applied_qualification_assignment	APQLAS
applied_security_classification_assignment	ASCA
applied_state_observed_assignment	ASOA
applied_state_type_assignment	ASTA
applied_time_interval_assignment	ATIA
approval	APPRVL
approval_assignment	APPASS
approval_date_time	APDTTM
approval_person_organization	APPROR
approval_relationship	APPRLT
approval_role	APPRL
approval_status	APPSTT
area_measure_with_unit	AMWU
area_unit	ARUNT
asin_function	ASNFNC
atan_function	ATNFNC
atom_based_literal	ATBSLT
attribute_classification_assignment	ATCLAS
attribute_language_assignment	ATLNAS
attribute_value_assignment	ATVLAS
attribute_value_role	ATVLRL

Entity name	Short name
axis1_placement	AX1PLC
axis2_placement_2d	A2PL2D
axis2_placement_3d	A2PL3D
b_spline_basis	BSPBS
b_spline_function	BSPFN
banded_matrix	BNDMTR
basic_sparse_matrix	BSSPMT
binary_boolean_expression	BNBLEX
binary_function_call	BNFNCL
binary_generic_expression	BNGNEX
binary_literal	BNRLTR
binary_numeric_expression	BNNMEX
boolean_defined_function	BLDFFN
boolean_expression	BLNEXP
boolean_literal	BLNLTR
boolean_variable	BLNVRB
bound_variable_semantics	BNVRSM
calendar_date	CLNDT
capacitance_measure_with_unit	CMWU
capacitance_unit	CPCUNT
si_capacitance_unit	SCPUN
cartesian_complex_number_region	CCNR
cartesian_point	CRTPNT
cartesian_transformation_operator	CRTROP
cartesian_transformation_operator_3d	CTO3
celsius_temperature_measure_with_unit	CTMWU
certification	CRTFCT
certification_assignment	CRTASS
certification_type	CRTTYP

Entity name	Short name
characterized_object	CHROBJ
class	CLASS
classification_assignment	CLSASS
classification_role	CLSRL
coaxiality_tolerance	CXLTLR
comparison_equal	CMPEQL
comparison_expression	CMPEXP
comparison_greater	CMPGRT
comparison_greater_equal	CMGREQ
comparison_less	CMPLSS
comparison_less_equal	CMLSEQ
comparison_not_equal	CMNTEQ
complex_number_literal	CMNMLT
composite_shape_aspect	CMSHAS
compound_representation_item	CMRPIT
concat_expression	CNCEXP
concentricity_tolerance	CNCTLR
concurrent_action_method	CNACMT
conductance_measure_with_unit	CMW0
conductance_unit	CNDUNT
si_conductance_unit	SCNUN
configuration_design	CNFDSG
configuration_item	CNFITM
constant_function	CNSFNC
context_dependent_action_method_relationship	CDAMR
context_dependent_action_relationship	CDAR
context_dependent_shape_representation	CDSR
context_dependent_unit	CNDPUN
contract	CNTRCT

Entity name	Short name
contract_assignment	CNTASS
contract_relationship	CNTRLT
contract_type	CNTTYP
conversion_based_unit	CNBSUN
coordinated_universal_time_offset	CUTO
cos_function	CSFNC
curve	CURVE
cylindricity_tolerance	CYLTLR
data_environment	DTENV
data_environment_relationship	DTENRL
date	DATE
date_and_time	DTANTM
date_and_time_assignment	DATA
date_assignment	DTASS
date_role	DTRL
date_time_role	DTTMRL
dated_effectivity	DTDEFF
datum	DATUM
datum_feature	DTMFTR
datum_reference	DTMRFR
datum_target	DTMTRG
defined_function	DFNFNC
definite_integral_expression	DFINEX
definite_integral_function	DFINFN
dependent_variable_definition	DPVRDF
derived_unit	DRVUNT
derived_unit_element	DRUNEL
description_attribute	DSCATT
descriptive_representation_item	DSRPIT

Entity name	Short name
dimension_related_tolerance_zone_element	DRTZE
dimensional_exponents	DMNEXP
dimensional_location	DMNLCT
direction	DRCTN
div_expression	DVEXP
document	DCMNT
document_product_association	DCP1
document_reference	DCMRFR
document_relationship	DCMRLT
document_representation_type	DCRPTY
document_type	DCMTYP
document_usage_constraint	DCUSCN
document_usage_constraint_assignment	DUCA
document_usage_role	DCUSRL
dose_equivalent_measure_with_unit	DEMWU
dose_equivalent_unit	DSEQUN
si_dose_equivalent_unit	SDEU
effectivity	EFFCTV
effectivity_assignment	EFFASS
effectivity_relationship	EFFRLT
electric_charge_measure_with_unit	ECM0
electric_charge_unit	ELCHUN
si_electric_charge_unit	SECU
electric_current_measure_with_unit	ECMWU
electric_current_unit	ELCRUN
electric_potential_measure_with_unit	EPMWU
electric_potential_unit	ELPTUN
si_electric_potential_unit	SEPU
elementary_function	ELMFNC

Entity name	Short name
elementary_space	ELMSPC
energy_measure_with_unit	EMWU
energy_unit	ENRUNT
si_energy_unit	SENUN
environment	ENVRNM
equals_expression	EQLEXP
event_occurrence	EVNOCC
event_occurrence_assignment	EVOCAS
event_occurrence_relationship	EVO0
event_occurrence_role	EVOCRL
executed_action	EXCACT
exp_function	EXPFNC
expanded_uncertainty	EXPUNC
explicit_table_function	EXTBFN
expression	EXPRSS
expression_denoted_function	EXDNFN
extended_tuple_space	EXTPSP
external_identification_assignment	EXIDAS
external_source	EXTSRC
external_source_relationship	EXSRRL
externally_defined_action_property	EDAP
externally_defined_class	EXD0
externally_defined_engineering_property	EDEP
externally_defined_item	EXDFIT
externally_defined_item_relationship	EDIR
externally_listed_data	EXLSDT
finite_function	FNTFNC
finite_integer_interval	FNININ
finite_real_interval	FNRLIN

Entity name	Short name
finite_space	FNTSPC
flatness_tolerance	FLTTLR
force_measure_with_unit	FMWU
force_unit	FRCUNT
si_force_unit	SFRUN
format_function	FRMFNC
free_variable_semantics	FRVRSM
frequency_measure_with_unit	FMW0
frequency_unit	FRQUNT
si_frequency_unit	SFR0
function_application	FNCAPP
function_space	FNCSPC
functionally_defined_transformation	FNDFTR
general_linear_function	GNLNFN
general_property	GNRPRP
generic_expression	GNREXP
generic_literal	GNRLTR
generic_variable	GNRVRB
geometric_representation_context	GMRPCN
geometric_representation_item	GMRPIT
geometric_tolerance	GMTTLR
geometric_tolerance_relationship	GMTLRL
geometric_tolerance_with_datum_reference	GTWDR
geometric_tolerance_with_defined_unit	GTWDU
global_unit_assigned_context	GUAC
group	GROUP
group_assignment	GRPASS
group_relationship	GRPRLT
homogeneous_linear_function	HMLNFN

Entity name	Short name
id_attribute	IDATT
identification_assignment	IDNASS
identification_assignment_relationship	IDASRL
identification_role	IDNRL
illuminance_measure_with_unit	IMWU
illuminance_unit	ILLUNT
si_illuminance_unit	SILUN
imported_curve_function	IMCRFN
imported_point_function	IMPNFN
imported_surface_function	IMSRFN
imported_volume_function	IMVLFN
index_expression	INDEXP
int_literal	INTLTR
int_numeric_variable	INNMVR
int_value_function	INVLFN
integer_defined_function	INDFFN
integer_interval_from_min	IIFM
integer_interval_to_max	IITM
integer_tuple_literal	INTPLT
interval_expression	INTEXP
inductance_measure_with_unit	IMW0
inductance_unit	INDUNT
si_inductance_unit	SINUN
item_defined_transformation	ITDFTR
item_identified_representation_usage	IIRU
language	LNGG
language_assignment	LNGASS
length_function	LNGFNC
length_measure_with_unit	LMWU

Entity name	Short name
length_unit	LNGUNT
like_expression	LKEXP
linearized_table_function	LNTBFN
listed_complex_number_data	LCND
listed_data	LSTDT
listed_integer_data	LSINDT
listed_logical_data	LSLGDT
listed_product_space	LSPRSP
listed_real_data	LSRLDT
listed_string_data	LSSTDT
literal_number	LTRNMB
local_time	LCLTM
location	LCTN
location_assignment	LCTASS
location_relationship	LCTRLT
location_representation_assignment	LCRPAS
log10_function	LG1FNC
log2_function	LG2FNC
log_function	LGFNC
logical_literal	LGCLTR
luminous_flux_measure_with_unit	LFMWU
luminous_flux_unit	LMFLUN
luminous_intensity_measure_with_unit	LIMWU
luminous_intensity_measure_with_unit	LIMWU
luminous_intensity_unit	LMINUN
magnetic_flux_density_measure_with_unit	MFDMWU
magnetic_flux_density_unit	MFDU
si_magnetic_flux_density_unit	SMFDU
magnetic_flux_measure_with_unit	MFMWU

Entity name	Short name
magnetic_flux_unit	MGFLUN
si_magnetic_flux_unit	SMFU
mass_measure_with_unit	MMWU
mass_unit	MSSUNT
material_designation	MTRDSG
material_designation_characterization	MTDSCH
material_property	MTRPRP
material_property_representation	MTPRRP
mathematical_description	MTHDSC
maths_boolean_variable	MTBLVR
maths_enum_literal	MTENLT
maths_function	MTH0
maths_integer_variable	MTINVR
maths_real_variable	MTRLVR
maths_space	MTH1
maths_string_variable	MTSTVR
maths_tuple_literal	MTTPLT
maths_value_qualification	MTVLQL
maths_value_representation_item	MVRI
maths_value_with_unit	MVWU
maths_variable	MTHVRB
maximum_function	MXMFNC
measure_qualification	MSRQLF
measure_representation_item	MSRPIT
measure_with_unit	MSWTUN
minimum_function	MNMFNC
minus_expression	MNSEXP
minus_function	MNSFNC
mod_expression	MDEXP

Entity name	Short name
modified_geometric_tolerance	MDGMTL
mult_expression	MLTEXP
multi_language_attribute_assignment	MLAA
multiple_arity_boolean_expression	MABE
multiple_arity_function_call	MAFC
multiple_arity_generic_expression	MAGE
multiple_arity_numeric_expression	MANE
name_attribute	NMATT
named_unit	NMDUNT
not_expression	NTEXP
numeric_defined_function	NMDFFN
numeric_expression	NMREXP
numeric_variable	NMRVRB
object_role	OBJRL
odd_function	ODDFNC
or_expression	OREXP
ordinal_date	ORDDT
organization	ORGNZT
organization_assignment	ORGASS
organization_relationship	ORGRLT
organization_role	ORGRL
organizational_address	ORGADD
organizational_project	ORGPRJ
organizational_project_assignment	ORPRAS
organizational_project_relationship	ORP0
organizational_project_role	ORPRRL
parallel_composed_function	PRCMFN
parallelism_tolerance	PRLTLR
partial_derivative_expression	PRDREX

Entity name	Short name
partial_derivative_function	PRDRFN
perpendicularity_tolerance	PRPTLR
person	PERSON
person_and_organization	PRANOR
person_and_organization_assignment	PAOA
person_and_organization_role	PAOR
person_assignment	PRSASS
person_role	PRSRL
personal_address	PRSADD
placement	PLCMNT
plane_angle_measure_with_unit	PAMWU
plane_angle_unit	PLANUN
plus_expression	PLSEXP
point	POINT
polar_complex_number_region	PCNR
position_tolerance	PSTTLR
positive_length_measure	PSLNMS
power_expression	PWREXP
power_measure_with_unit	PMWU
power_unit	PWRUNT
si_power_unit	SPWUN
pre_defined_item	PRDFIT
precision_qualifier	PRCQLF
pressure_measure_with_unit	PMW0
pressure_unit	PRSUNT
si_pressure_unit	SPRUN
process_or_process_relationship_effectivity	POPRE
process_product_association	PRPRAS
process_property_association	PRPRS

Entity name	Short name
product	PRDCT
product_as_individual	PRASIN
product_as_planned	PRASPL
product_as_realised	PRA0
product_category	PRDCTG
product_category_relationship	PRCTRL
product_concept	PRDCNC
product_concept_context	PRCNCN
product_context	PRDCNT
product_definition	PRDDFN
product_definition_context	PRDFCN
product_definition_formation	PRDFFR
product_definition_formation_relationship	PDFR
product_definition_process	PRDFPR
product_definition_relationship	PRDFRL
product_definition_shape	PRDFSH
product_definition_substitute	PRDFSB
product_definition_with_associated_documents	PDWAD
product_material_composition_relationship	PMCR
product_related_product_category	PRPC
product_relationship	PRDRLT
projected_zone_definition	PRZNDF
property_definition	PRPDFN
property_definition_relationship	PRDFR
property_definition_representation	PRDFRP
property_process	PRPPRC
qualification_type	QLFTYP
qualification_type_relationship	QLT0
qualification_type_assignment	QLTYAS

Entity name	Short name
qualified_representation_item	QLRPIT
qualitative_uncertainty	QLTUNC
quantifier_expression	QNTEXP
radioactivity_measure_with_unit	RMW0
radioactivity_unit	RDCUNT
si_radioactivity_unit	SRDUN
ratio_measure_with_unit	RMWU
ratio_unit	RTUNT
rationalize_function	RTNFNC
real_defined_function	RLDFFN
real_interval_from_min	RIFM
real_interval_to_max	RITM
real_literal	RLLTR
real_numeric_variable	RLNMVR
real_tuple_literal	RLTPLT
referenced_modified_datum	RFMDDT
regular_table_function	RGTBFN
reindexed_array_function	RNARFN
repackaging_function	RPCFNC
representation	RPRSNT
representation_context	RPRCNT
representation_item	RPRITM
representation_item_relationship	RPITRL
representation_relationship	RPRRLT
representation_relationship_with_transformation	RRWT
requirement_for_action_resource	RFAR
resistance_measure_with_unit	RMW1
resistance_unit	RSSUNT
si_resistance_unit	SRSUN

Entity name	Short name
resource_property	RSRPRP
resource_property_relationship	RSPRRL
resource_property_representation	RSPRRP
resource_requirement_type	RSRQTY
resource_requirement_type_relationship	RRTR
restriction_function	RSTFNC
role_association	RLASS
roundness_tolerance	RNDTLR
runout_zone_definition	RNZNDF
runout_zone_orientation	RNZNOR
runout_zone_orientation_reference_direction	RZORD
security_classification	SCRCLS
security_classification_assignment	SCCLAS
security_classification_level	SCCLLV
selector_function	SLC0
sequential_method	SQNMTH
serial_action_method	SRACMT
serial_numbered_effectivity	SRNMEF
series_composed_function	SRCMFN
shape_aspect	SHPASP
shape_aspect_relationship	SHASRL
shape_definition_representation	SHDFRP
shape_representation	SHPRPR
shape_representation_relationship	SHRPRL
si_unit	SUNT
simple_boolean_expression	SMBLEX
simple_generic_expression	SMGNEX
simple_numeric_expression	SMNMEX
simple_string_expression	SMSTEX

Entity name	Short name
sin_function	SNFNC
solid_angle_measure_with_unit	SAMWU
solid_angle_unit	SLANUN
square_root_function	SQRTFN
standard_table_function	STTBFN
standard_uncertainty	STNUNC
state_type	STTTYP
state_type_assignment	STTYAS
state_type_relationship	STTYRL
state_type_role	STT0
ascribable_state	ASCSTT
ascribable_state_relationship	ASSTRL
state_observed	STTOBS
state_observed_assignment	STOBAS
state_observed_relationship	STOBRL
state_observed_role	STO0
straightness_tolerance	STRTLR
strict_triangular_matrix	STTRMT
string_defined_function	STDFFN
string_expression	STREXP
string_literal	STRLTR
string_variable	STRVRB
substring_expression	SBSEXP
surface	SRFC
symmetric_banded_matrix	SYBNMT
symmetric_matrix	SYMMTR
symmetric_shape_aspect	SYSHAS
symmetry_tolerance	SYMTLR
tan_function	TNFNC

Entity name	Short name
thermodynamic_temperature_measure_with_unit	TTMWU
thermodynamic_temperature_unit	THTMUN
time_interval	TMINT
time_interval_assignment	TMINAS
time_interval_based_effectivity	TIBE
time_interval_relationship	TMINRL
time_interval_role	TMI0
time_interval_with_bounds	TIWB
time_measure_with_unit	TMWU
time_unit	TMUNT
tolerance_zone	TLRZN
tolerance_zone_definition	TLZNDF
tolerance_zone_form	TLZNFR
total_runout_tolerance	TTRNTL
triangular_matrix	TRNMTR
type_qualifier	TYPQLF
unary_boolean_expression	UNBLEX
unary_function_call	UNFNCL
unary_generic_expression	UNGNEX
unary_numeric_expression	UNNMEX
uncertainty_measure_with_unit	UMWU
uncertainty_qualifier	UNCQLF
uniform_product_space	UNPRSP
value_function	VLFNC
value_representation_item	VLRPIT
variable	VRBL
variable_semantics	VRBSMN
vector	VECTOR
velocity_measure_with_unit	VMW0

Entity name	Short name
velocity_unit	VLCUNT
versioned_action_request	VRACRQ
volume	VOLUME
volume_measure_with_unit	VMWU
volume_unit	VLMUNT
week_of_year_and_day_date	WOYADD
xor_expression	XREXP
year_month	YRMNT

Annex C
(normative)

Implementation method specific requirements

The implementation method defines what types of exchange behaviour are required with respect to this part of ISO 10303. Conformances to this part of ISO 10303 shall be realized in an exchange structure. The file format shall be encoded according to the syntax and EXPRESS language mapping defined in ISO 10303-21 and in the AIM defined in annex A of this part of ISO 10303. The heading of the exchange structure shall identify use of this part of ISO 10303 by the schema name 'engineering_properties_schema'.

Annex D
(normative)

Protocol Implementation Conformance Statement (PICS) proforma

D.1 General

This annex lists the optional elements of this part of ISO 10303. An implementation may choose to support any combination of these optional elements. However, certain combinations of options are likely to be implemented together. These combinations are called conformance classes amd are described in the subclauses of this annex.

This annex is in the form of a questionnaire. This questionnaire is intended to be filled out by the implementor and may be used in preparation for conformance testing by a testing laboratory. The completed PICS proforma is referred to as a "PICS".

D.2 Protocol implementation identification

ISO implementation name	
Current version and release date	

D.3 Implementation method

Indicate the chosen implementation method and supported directions of translation. If more than one implementation method is supported, a separate PICS proforma is to be filled in more each of them.

Implementation method	EXPRESS mapping	Preprocessor	Postprocessor
ISO 10303-21	Not applicable		
ISO 10303-28			

D.4 Implementation conformance classes

Conformance class	Preprocessor	Postprocessor
Engineering properties (CC1)		

Annex E
(normative)

Information object registration

E.1 Document identification

In order to provide for unambiguous identification of this part of ISO 10303 as an information object in an open system, it is assigned the object identifier

{iso standard 10303 part(235) version(1) }

The meaning of this value is defined in ISO/IEC 8824-1 and is further described in ISO 10303-1.

E.2 Schema identification

ISO 10303-1 further describes how ISO/IEC 8824-1 can be used to identify individual schemas. This part of ISO 10303 contains two such schemas, and each is assigned a unique object identifier to provide for unabiguous identifcation of the schema in an open information system.

E.2.1 engineering_properties expanded schema

The engineering_properties expanded schema (see Annex A) is assigned the object identifier

{ iso standard 10303 (part 235) version(1) object(1) engineering-properties-schema(1) }

E.2.2 engineering_properties short form schema

The engineering_properties short form schema (see 5.2) is assigned the object identifier

{ iso standard 10303 (part 235) version(1) object(1) engineering-properties-schema(2) }

Annex F
(informative)

Application activity model

F.1 General

The application activity model (AAM) is provided as an aid to understanding the scope and information requirements defined in this part of ISO 10303. The model is presented as a set of figures that contain the activity diagrams and a set of definitions of the activities and their data. Activities and data flows that are out of scope are marked with an asterisk (*) Activities and data flows that are partially out of scope are marked with an asterisk and # (*#). The application activity model assumes the determination of properties to related to materials as examples of engineering properties.

F.2 Application activity model definitions

The following terms are used in the application activity model. Terms marked with an asterisk (*) are outside the scope of this part of ISO 10303. Terms marked with an asterisk and # (*#) are partially within the scope of this part of ISO 10303.

The definitions given in this annex do not supersede the definitions given in the main body of the text.

F.2.1 Abstract idealised geometry *:

use a CAD system to modify design geometry to make it suitable for input to an engineering analysis package.

F.2.2 Allowables :

material properties of a typical batch or part, with approval status, resulting from a data reduction process on multiple tests on multiple batches or parts intended for use in the design and analysis of a material object.

NOTE These include statistical variations, uncertainties in measurement etc.

F.2.3 Analysis and test plan *:

a description of the analyses and testingwhich will be performed on the product design. This will include the objectives of the analyses and selection of analysis types.

F.2.4 Analysis and test results *:

the reduced results from product analyses and tests performed as part of the design process, including a record of the decisions made during the analysis process.

F.2.5 Analysis controls *#:

the additional information attached to a discretised model and environment which is necessary to run an analysis.

NOTE This would normally include output requests and analysis procedure controls.

F.2.6 Analysis model *:

a description of the behaviour of a material object or assembly of material objects in a form suitable for analysis software.

NOTE The description may include descriptions of the shape of the material object or objects and descriptions of properties which determine their behaviour such as elasticity.

F.2.7 Analysis output data *:

the field variable values and various output matrices that result from an engineering analysis.

F.2.8 Analysis results *:

the analysis output data combined with information about the activity that is analysed, the purpose of the analysis and a record of the decisions made during the analysis.

NOTE The analysis results constitute the predicted behaviour of the material object.

F.2.9 Approvals *:

appropriate authorisation for release or status raising.

NOTE The initial approval to begin the project would be included as a type of approval.

F.2.10 Assess results *:

the process of assessing analysis and/or test results against validation requirements which either results in a design aproval or a request for change.

F.2.11 Assign factors of safety and design values:

assign acceptable factors of safety, durability and damage tolerance design values to a discretised model.

F.2.12 Assigned design values:

design values which have been chosen and assigned to the discretised model.

F.2.13 Batch of material object *:

a batch of material or a particular material object.

F.2.14 Batch or material object definition:

administrative and technical information about a stock material or material object from which a test specimen is manufactured.

F.2.15 Boundary constraints and releases *:

the constraints and releases applied to a discretised model to simulate the presence of connecting structure and/or mountings/attachments.

F.1.16 Build component part list *:

the process of building a list of the parts that make up the component.

F.2.17 Complete analysis input *:

all the information needed to perform an analysis.

NOTE This information could typically include discretised model, discretised environment, output selection, mathematical model, information to control the analysis steps and a record of the decisions made during an analysis process.

F.2.18 Component design change requests *:

modification requests relating to the design of a component.

F.2.19 Component design R and O *:

the requirements and objectives for the design of a component which have been passed down from the design requirements and constraints placed on the whole product.

F.2.20 Component models and drawings *:

the models and drawings which make up the design data relating to a component.

F.1.21 Component part list data *:

the list of parts which make up a component of the whole assembly.

F.2.22 Component test R and O *:

the descriptions of the objectives and requirements for testing the component which have been passed down from those related to the testing of the whole product.

F.2.23 Conduct component and system design and analysis *#:

the whole process of design and analysis of a component of the product.

F.2.24 Conduct detail analyses *#:

the process of performing detail engineering analyses on a design of a material object.

F.2.25 Conduct detail assembly design *:

the process of collecting together component design data and producing a detail design of the whole product.

F.2.26 Conduct initial component analysis *:

the simple analysis which is performed on an interim design of a component during the course of that design.

F.2.27 Conduct initial whole system analysis *:

the preliminary analysis which is performed on an interim design of a product during the course of that design.

F.2.28 Conduct preliminary whole system and process design and analysis*:

the initial design and analysis of the whole system and process including the definition of component specifications.

F.2.29 Conduct preliminary whole system design *:

the preliminary design of a whole system.

F.2.30 Conduct product design, analysis and assessment *:

the whole process of designing and analysing a product including the assessment of the design and analysis in order to produce an accepted product design.

F.2.31 Create and document results database *:

populate a database with the results of engineering analyses together with references back to the analysis and design data.

F.2.32 Create discretised model *:

the process of producing a discretised model which is suitable for input to an engineering analysis package from design geometry

NOTE The discretised model may be a direct definition or a mathematical model based on design values.

F.2.33 Data reduction methodology *:

the description of the methodology used to reduce the data.

NOTE This will include reference to standards used.

F.2.34 Define analysis controls *:

definition of the information which will control the analysis output and the analysis procedure itself.

F.2.35 Define component specifications *:

the process of defining component test and design objectives and requirements based on preliminary design and analysis of the whole product.

F.2.36 Define material allowables:

the process of defining allowables based on reduced materials test data, and intended for use in design and analysis. This will include attaching an approval status to the values which indicates the use for which the values have been approved.

F.2.37 Define material specifications:

the process of defining material specifications based on reduced materials test data.

F.2.38 Define mathematical model *:

the specification of the mathematical equations which are to be used as the basis for the analysis program.

F.2.39 Delivered product *#:

the manufactured material object which is delivered to the customer.

F.2.40 Design requirements and objectives *:

the performance attributes that the whole product design shall satisfy.

F.2.41 Design values:

Maximum or minimum values of a property, together with a quality value, which are designed to be used as failure criteria at a particular stage in a design and analysis cycle. These values are based on allowables adjusted to account for programme criteria and actual structural conditions.

NOTE A design value can depend upon the nature of a material object and its service environment. A design value cannot be determined solely by any measurement upon a material object. A design value can include partial safety factors to account for uncertainties in the prediction of the behaviour of a material object.

F.2.42 Design, analyse and test a product *#:

the whole cycle of product design, analysis, testing and assessment which results in a certified product design and specifications for the manufacture, use and disposal of a product.

NOTE This includes using and managing material property data.

F.2.43 Detail assembly design data *:

all the models, drawings and parts lists that describe a product or assembly of material objects.

F.2.44 Detail component design data *:

all the models, drawings and parts lists that describe a component of a product.

F.2.45 Develop and manage material property information:

the process of generating material property information and making it available in a form suitable for use in engineering design and analysis.

F.2.46 Develop test plan *:

the process of developing a detailed plan of testing based on relevant standards, for the generation of materials information.

F.2.47 Develop test plan *:

develop a detailed plan of testing based on relevant standards.

F.2.48 discretised environment *:

a discretised model of the loads and constraints that act on the intended product which is suitable to be applied to the discretised analysis model and used as input to the analysis.

F.2.49 Discretised model *:

a discretised model of the intended product, suitable for input to an engineering analysis package.

NOTE – Examples of a discretised model include finite element, finite difference or finite volume models.

F.2.50 Disposal records *:

information which records the disposal activity carried out upon a material object.

NOTE This information includes dis-assembly and disposal problems which can be input to a re-design of a material object.

F.2.51 Disposal specification *:

the specification of a disposal activity.

NOTE This information includes the specification of how an assembly can be reduced to component parts

F.2.52 Dispose of an actual product *:

the processing of a material object into stock material or parts that can be recycled as input into manufacturing activities or stored in safety.

F.2.53 Environment model *:

a discrete model of the loads and constraints which will apply to the analysis to be performed.

F.2.54 expended or obselete product *:

a product which has come to the end of its useful life.

F.2.55 Field and maint. changes and revision history *:

the reviewed field/maintenance changes that result from in field use of the product and the history of these changes in field.

F.2.56 Generate and assign discrete attributes *:

generate and assign discrete geometrical and material attributes.

F.2.57 Generate and assign load sets and combinations *:

generate and assign discrete loadings that approximate the forces, temperatures, displacements and other loads acting on the product and request the combination of load sets to approximate complicated loading conditions from simpler loading components.

F.2.58 Generate design values:

the generation of design values which will be used as failure criteria during the design and analysis. These values are based on allowables adjusted to account for programme criteria and actual structural conditions.

F.2.59 Generate environment model *#:

generate, set and assign analysis environment data such as boundary constraints, loads, factors of safety.

F.2.60 Generate response models *#:

generate a discrete geometric approximation of the material object.

F.2.61 Geometry suitable for discretising *:

geometry which may have been simplified, defeatured or in any other way processed to make it appropriate for input to a mesh generator or other analysis preprocessor.

F.2.62 Idealisation methodology *:

the methods and equations which will be used to idealise the problem for use in a particular analysis package.

F.2.63 Idealise and discretise environment *:

the process of simplifying and discretising the environment operating on the intended material object to make it suitable for applying to the discretised model of the object for the purposes of an analysis.

F.2.64 Input, fix and modify geometry from CAD system *:

the process of importing geometry from a CAD system and then repairing any problems with that geometry.

F.2.65 Interpret results *:

reduce data from multiple tests, as per standard requirements, compare against allowables and present in an acceptable format.

F.2.66 Lifecycle history *:

all the records generated during the manufacturing, operation and disposal parts of the product lifecycle.

F.2.67 Lifecycle specifications *:

all the specifications needed during the manufacturing, operation and disposal of the product lifecycle.

F.2.68 Load sets and combinations *:

these provide a complete set of loads data.

NOTE There may be one or more load sets of combination of load sets in a given engineering analysis.

F.2.69 Maintain and use an actual product *:

the process of using a manufactured product in an operational process.

F.2.70 Maintain material object records:

the process of storing and keeping records of material object data and test data in a manner which enables various materials and process data to be extracted as required.

F.2.71 Manage material property information:

the management of material property information by storing it in a form whereby searches can be made and information retrieved, as in a database.

F.2.72 Managed materials data:

all the materials information generated by testing and evaluation which is available to the engineering design and analysis process in a searchable form.

F.2.73 Manufacture an actual product for in-service use *:

the manufacture of a material object that is intended for use.

NOTE 1 This activity can manufacture many material objects for use from the same manufacturing specification.

NOTE 2 A manufactured material object can be a simple component, such as a rivet, or a complex assembly, such as a complete aircraft.

F.2.74 Manufacture and condition test specimen *:

the process of manufacturing a test specimen (also known as a test coupon) from stock material for the purpose of performing a test.

F.2.75 Manufacture product or prototype *:

the process of making an actual product or a prototype of a product which is to be used for performing a test.

F.2.76 Manufacture, use and dispose of an acutal product *:

the parts of a product lifecycle which include producing, using and disposing of a material object following a design cycle.

F.2.77 Manufacturing records *:

this information records the manufacturing activity carried out on a physical object, and includes:

– the identification of input stock material and component parts;

– dates and times of manufacturing processes, and records of the tools used and their operational parameters or settings;

– results of tests carried out on the manufacturing problems which can be input to a re-design of a material object.

This information includes manufacturing problems which can be input to a re-design of a material object.

F.2.78 Manufacturing specification *:

this information is the specification of a manufacturing activity, and includes:

– the specification of input stock materials and component parts;

– the specification of the processes carried out on the inputs, the tools used and their operational parameters or settings (including NC data);

– the specification of the testing procedures to be carried out to ensure quality of the manufactured material objects.

F.2.79 Material object for use during test :

material objects which will be used to create the appropriate environment for the execution of the test.

NOTE This would include bottled gases of a specified level of purity and instant seawater powder for producing standard seawater.

F.2.80 Material object specifications :

the methodology for producing and processing a material object.

F.2.81 Material object used in creation of test specimen :

Material objects which are used to create the appropriate environment for the creation of the test speciment.

NOTE For example, argon shielding gas used in preparing a test weld, or etching reagent used to condition a specimen.

F.2.82 Material specifications:

standard specifications of required material properties.

F.2.83 Material test plan :

the specification of a test, to be performed on a material object, based on or including the relevant standards for the purposes of generating materials information.

NOTE This includes the specification of the manufacture and conditioning of the test specimen.

F.2.84 Materials information:

all the information which is generated about a material product during its testing including audit trails, processing information and the original batch or material object definition.

F.2.85 Mathematical model *:

the mathematical equations which are to be used by the analysis package to model the behaviour of the product.

F.2.86 Mathematical model factors *:

the factors that are input to analysis software, in particular to Computational Fluid Dynamics packages, that modify the description of the behaviour modelled in the analysis. These factors are chosen by comparing test results with the output from the analysis package.

F.2.87 Measured data and test information:

all the test data and test condition information which is to be archived for future use.

F.2.88 Operating environment *#:

all the physical constraints which will impact on the design of the product including any boundary conditions and the space envelope available.

F.2.89 Operating records *:

this information records the service that is provided by a material object. It includes operating and maintenance problems which can be input to a re-design of a material object.

NOTE Information about radioactive, chemical or biological contamination will determine the nature of the activity 'dispose of material object'.

F.2.90 Operating specification *:

this information is the specification of a service for which a material object has been designed, or for which it has been approved.

F.2.91 Partially stressed model *:

processed results from a previous analysis which denote some of the stresses which will act on the analysis model and which are to be used as input for another analysis.

F.2.92 Perform analysis *:

perform a finite element, finite difference, finite volume, boundary element or other engineering analysis of the material object by submitting the analysis model, together with the environment and controls, for analysis by the appropriate application.

F.2.93 Perform test *:

the activity of performing a test on a specimen of a material object to determine its material properties.

F.2.94 Perform test run *:

the activity of performing a test on a prototype of a material object to determine its measured behaviour.

F.2.95 Plan analyses and testing *:

plan which analyses and testing will be performed upon the product design to determine its predicted and measured behaviours.

F.2.96 Predicted environment *:

information which is to be used as an environment for an analysis which has been generated as a result of a physical test.

F.2.97 Prepare component models and drawings *:

produce detailed drawings and CAD models for a component of the product.

F.2.98 Previous relevant design histories *:

design data from previous designs of similar products or products which have some relevance for the current product.

F.2.99 Procure and test material object:

the procurement and testing of stock material or standard parts to generate test results.

F.2.100 Procure material object for testing *:

the procurement and testing of either stock material or a standard part to generate test results.

F.2.101 Product design acceptance *:

the approval of a product design indicating that it is accepted for manufacture or for use in a particular usage scenario.

F.2.102 Product design and test data *:

all design and test data generated about a product including detail designs and conceptual models.

F.2.103 Product design data *:

add design data generated about a product including detail designs and conceptual models.

F.2.104 Product or prototype for testing *:

a manufactured product or prototype material object which has been produced for the purposes of testing to establish the measured behaviour of the intended material object.

F.2.105 Raw test run results *:

the raw test results produced by a test run together with a description of the environment for that run.

F.2.106 Recommended changes *:

design change orders resulting from assessment of analysis and/or testing of a product.

NOTE A product may be either intended or actual.

F.2.107 Recycled or waste product *:

the final products of a disposal activity resulting in material which may be re-used as stock material or waste material which may be safely stored.

F.2.108 Reduce and evaluate data:

the production of useful data from raw test data by removal of spurious data, smoothing, averaging and other techniques.

F.2.109 Reduce data *:

the process of reducing test data from one or more tests for a specific batch of material. This may be achieved by averaging or smoothing processes.

F.2.110 Reduced material data:

material properties of a specific batch or part, resulting from a data reduction process on multiple tests on a single batch or part including administrative data.

F.2.111 Reduced test data :

the results of observations of measurements of a material object, on its own or as part of an assembly including information about the conditions under which the test was performed.

NOTE This information constitutes the measured behaviour of the material object.

F.2.112 Select material:

the process of selection of materials to be used in the design of the component.

F.2.113 Selected material:

the materials selected to be used in the design of the component.

F.2.114 Set and assign boundary conditions and loads *:

set and assign boundary constraints and releases that approximate the support and/or symmetry boundary conditions for the analysis of the material object.

F.2.115 Simplified design geometry *:

design geometry which had been defeatured or otherwise modified to be suitable for input to an engineering analysis package.

F.2.116 Specification, design, analysis, test, manufacture, use and disposal of a product *#:

the whole lifecycle of a product.

F.2.117 Specify functional design and physical interfaces *:

the process of specifying the operating environment, the test objectives and requirements and the design objectives and requirements of a product concept.

F.2.118 Staff and tools *#:

the staff and tools, including testing machines and computer software, that are used by the customer, its contractors and materials suppliers.

F.2.119 Standards *#:

the international, national and company standards which apply to the design and analysis of a material object and to the generation and use of materials information.

F.2.120 Stock material and component parts *#:

crude unprocessed or partially processed material used as a feedstock for a processing operation and the as-built parts which are procured from an associate or subcontractor.

F.2.121 Supplier records *#:

the supplier provided information about procured as-built parts and crude unprocessed or partially processed material used as a feedstock for a processing operation. This includes the drawings and material information such as the source stock and source properties.

F.2.122 System and component test objectives and requirements *:

the objectives and requirements of the tests which will be applied to the whole system and its components.

F.2.123 System design *:

the preliminary design, including drawings and models, of the whole system or product

F.2.124 System design change requests *:

requests for modification to the design of the overall system.

F.2.125 System test objectives and requirements *:

the objectives and requirements of the tests which will be applied to the whole system.

F.2.126 Test condition records:

all the information about the conditions under which a test was performed.

F.2.127 Test plan:

the specification of a test, to be performed on a material object, based on or including the relevant standards.

NOTE This includes the specification of the manufacture and conditioning of the test specimen.

F.2.128 Test product *:

a test to determine the measured behaviour of the product design.

F.2.129 Test results:

the raw test results produced by a material test together with a description of the environment for that run.

F.2.130 Test specimen :

a material object that is subjected to test.

NOTE a small material object that has been manufactured specially for testing can be called a 'coupon'.

F.1.131 Test specimen definition:

administrative and technical information from the appropriate test standard, about a test specimen, including its identification, description and process.

F.2.132 Tested material object:

a test specimen or actual product which has undergone some testing and is to be retained either for the purposes or further testing or for record keeping.

F.2.133 Verify product design *#:

the process of analysing and testing an interim design of a material object including assessing the results against requirements and either issuing the design as a certified design or generating design change orders.

F.3 Application activity model diagrams

The application activity model diagrams are given in Figures F.1 to F.16. The graphical form of the application activity model is presented in the IDEF0 activity modelling format (see Reference [12]). Activities and data flows that are out of scope are marked with asterisks (*). Activities and data flows that are partially out of scope are marked with asterisk and # (*#). Some activites produce data while other activities produce physical objects. The activities and data flows that produce physical objects are indicated by a thickened line.

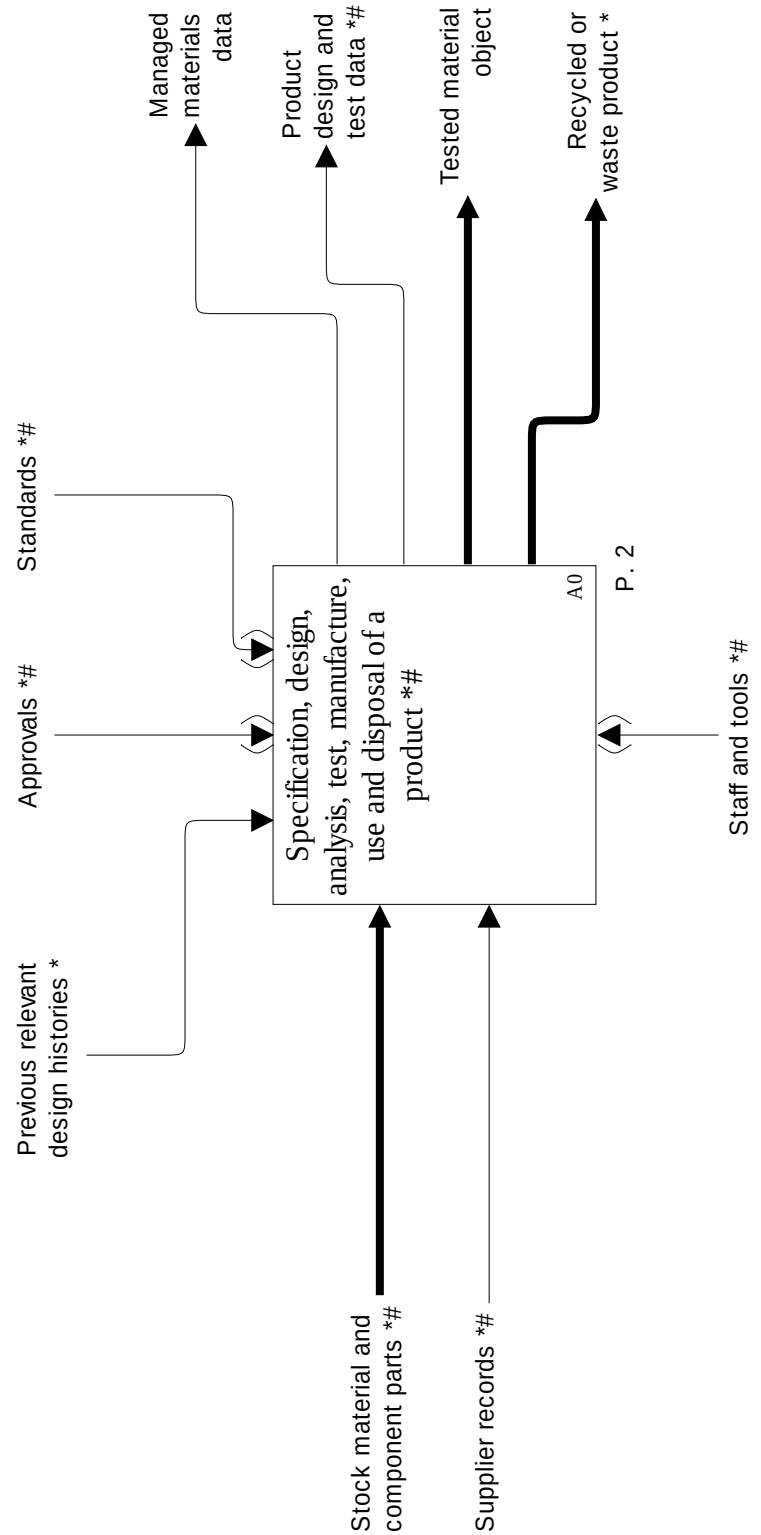


Figure F.1 — Application Activity Model top level

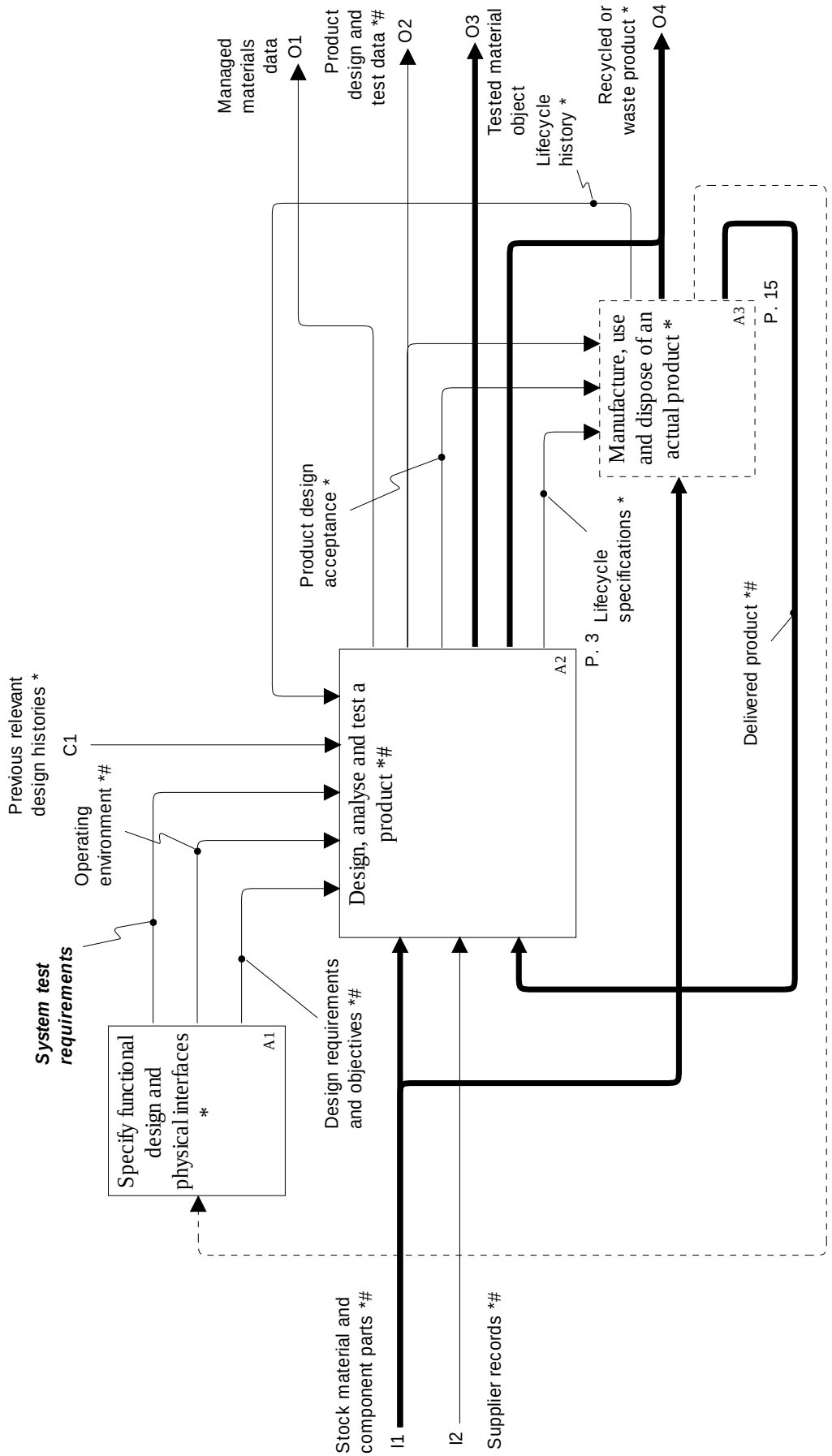


Figure F.2 — Specification, design, analysis, test, manufacture, use and dispose of an actual product

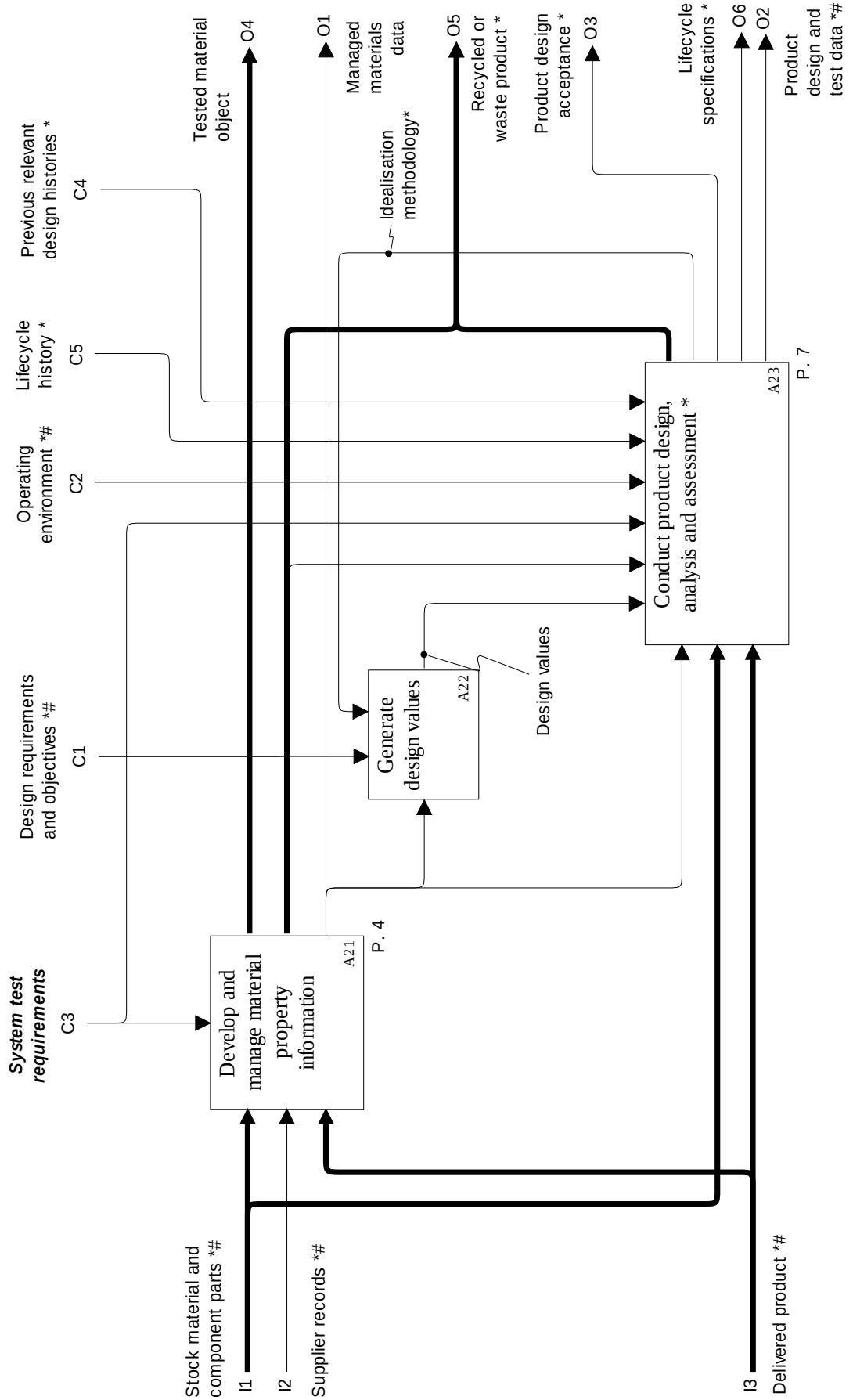


Figure F.3 — Design, analyse and test a product

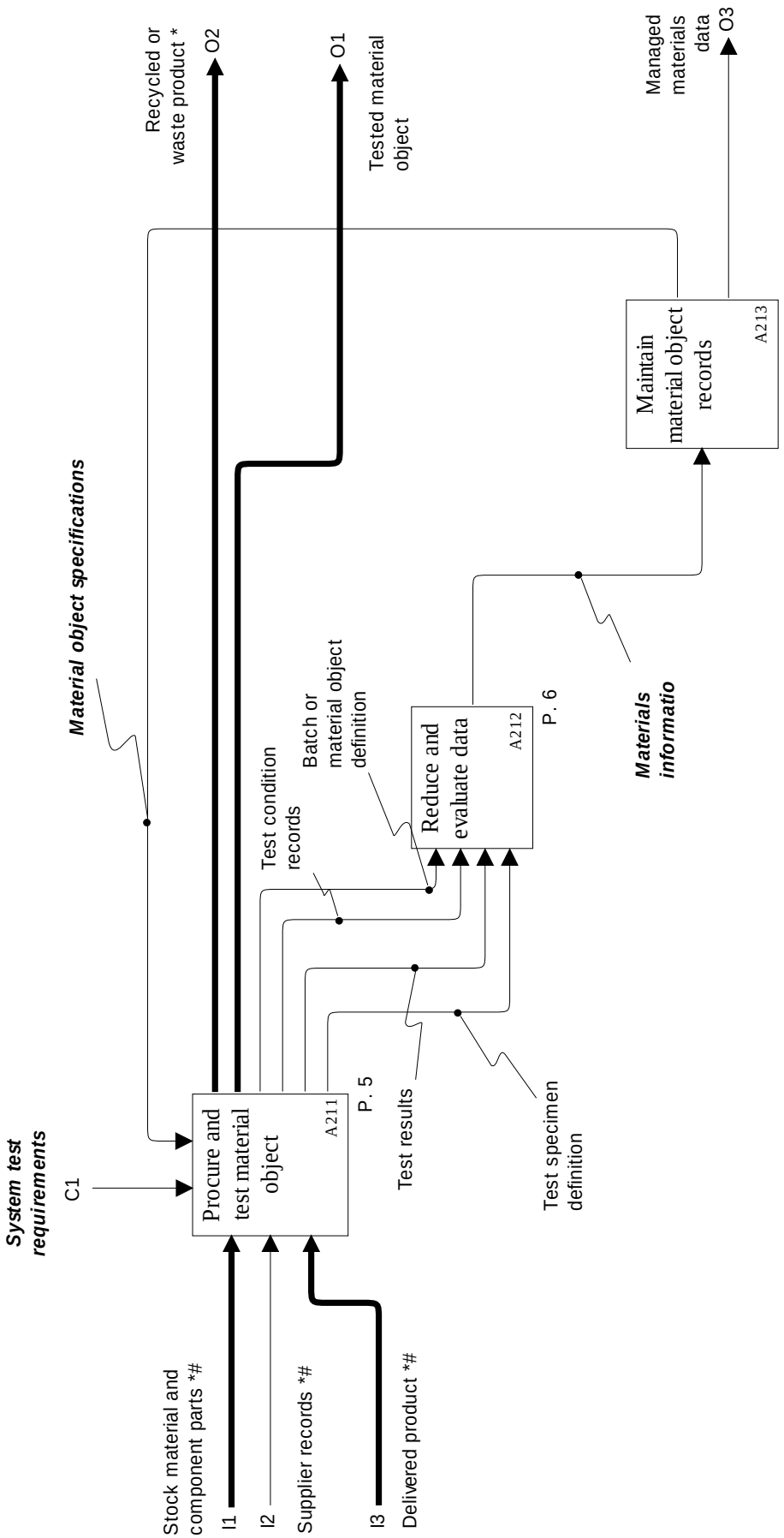


Figure F.4 — Develop and manage material property information

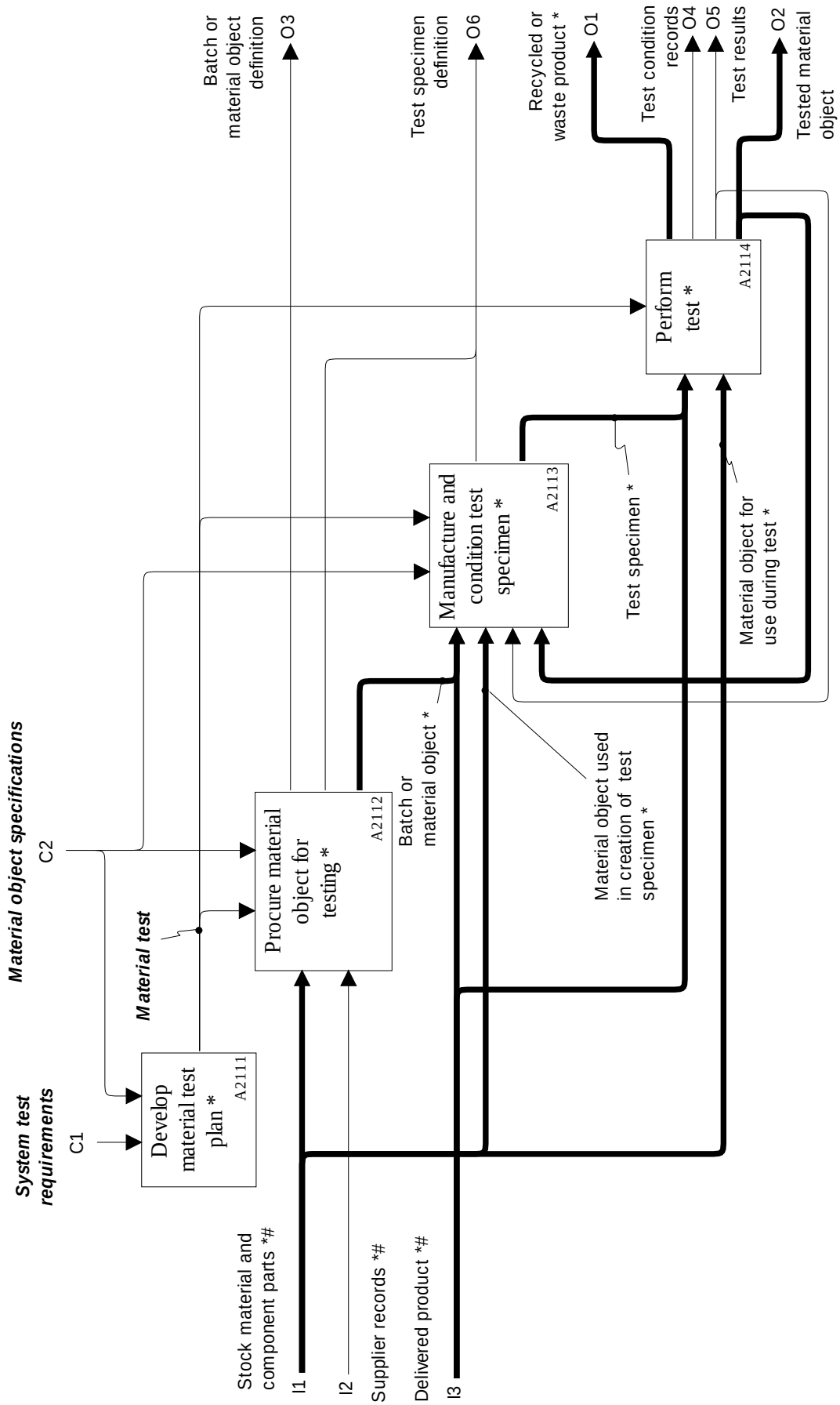


Figure F.5 — Procure and test material object

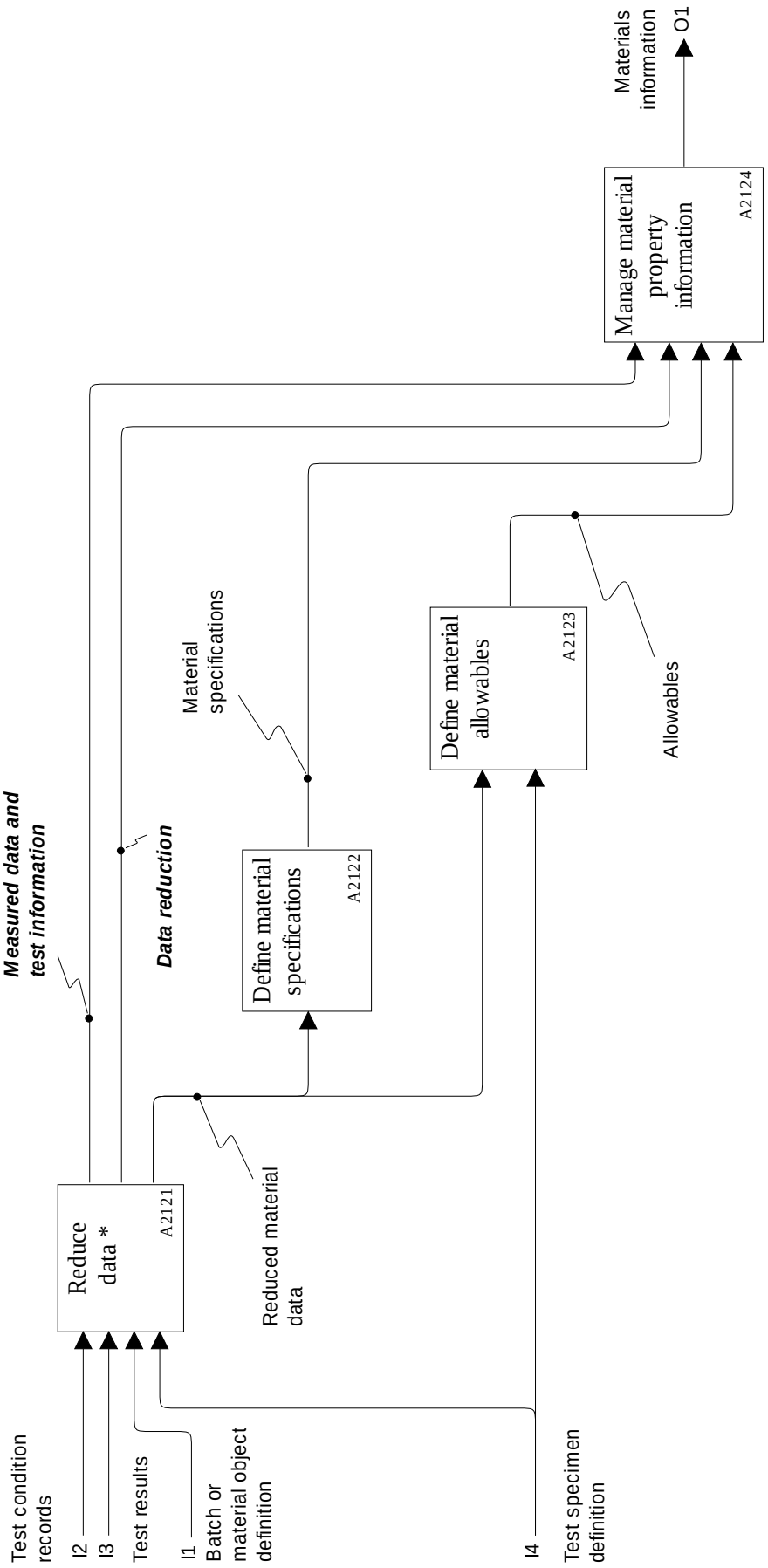


Figure F.6 — Reduce and evaluate data

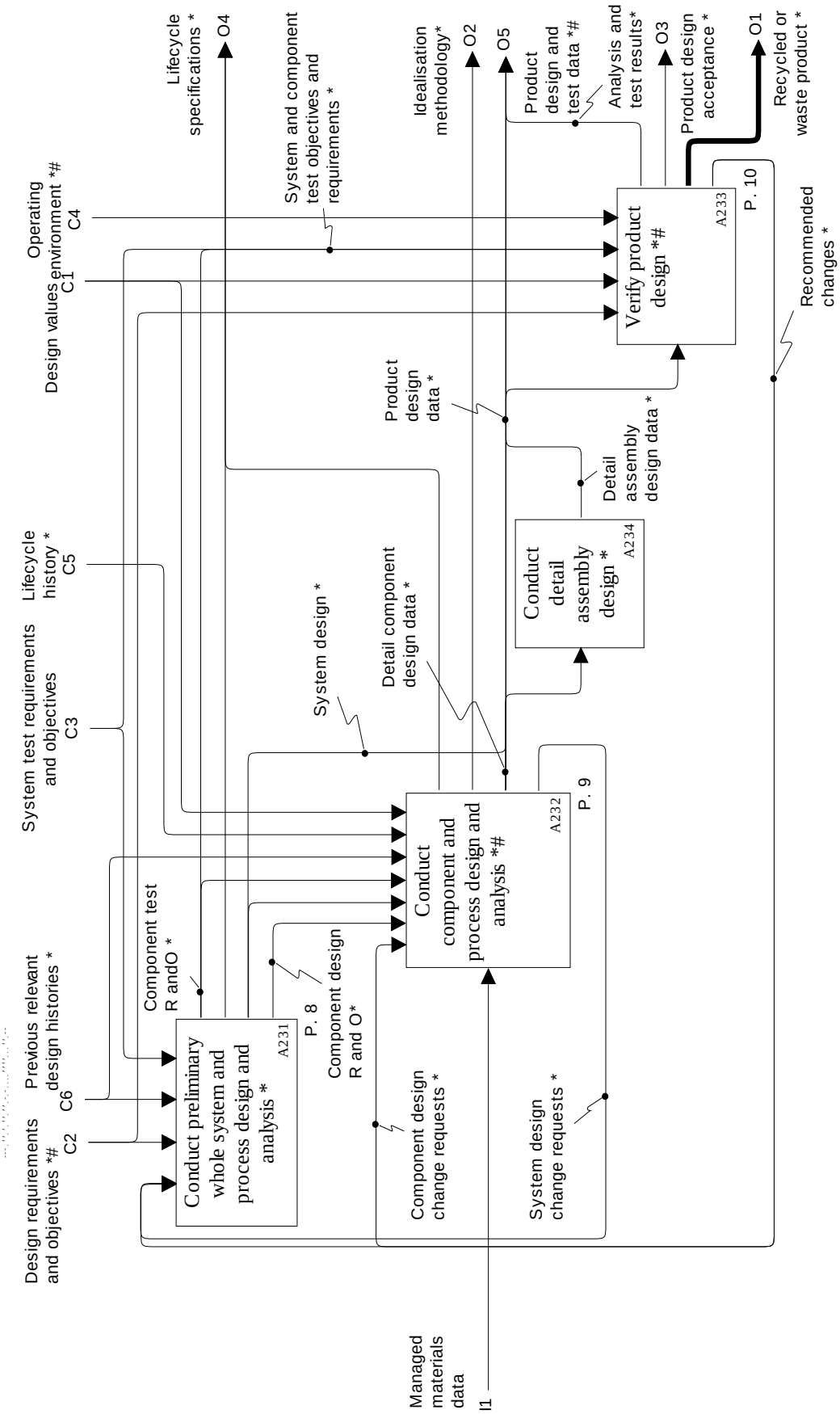


Figure F.7 — Conduct product design, analysis and assessment

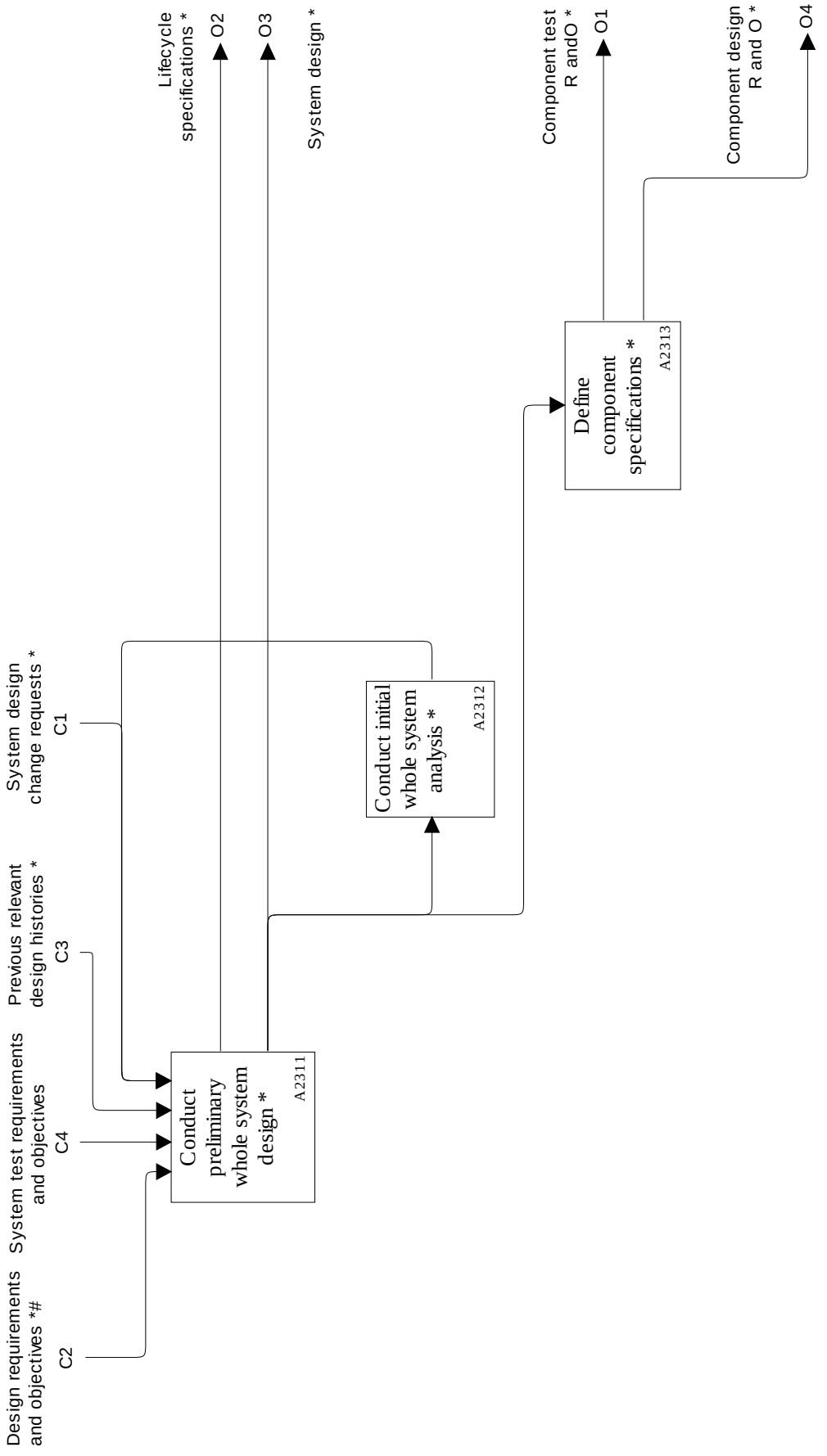


Figure F.8 — Conduct preliminary whole system and process design

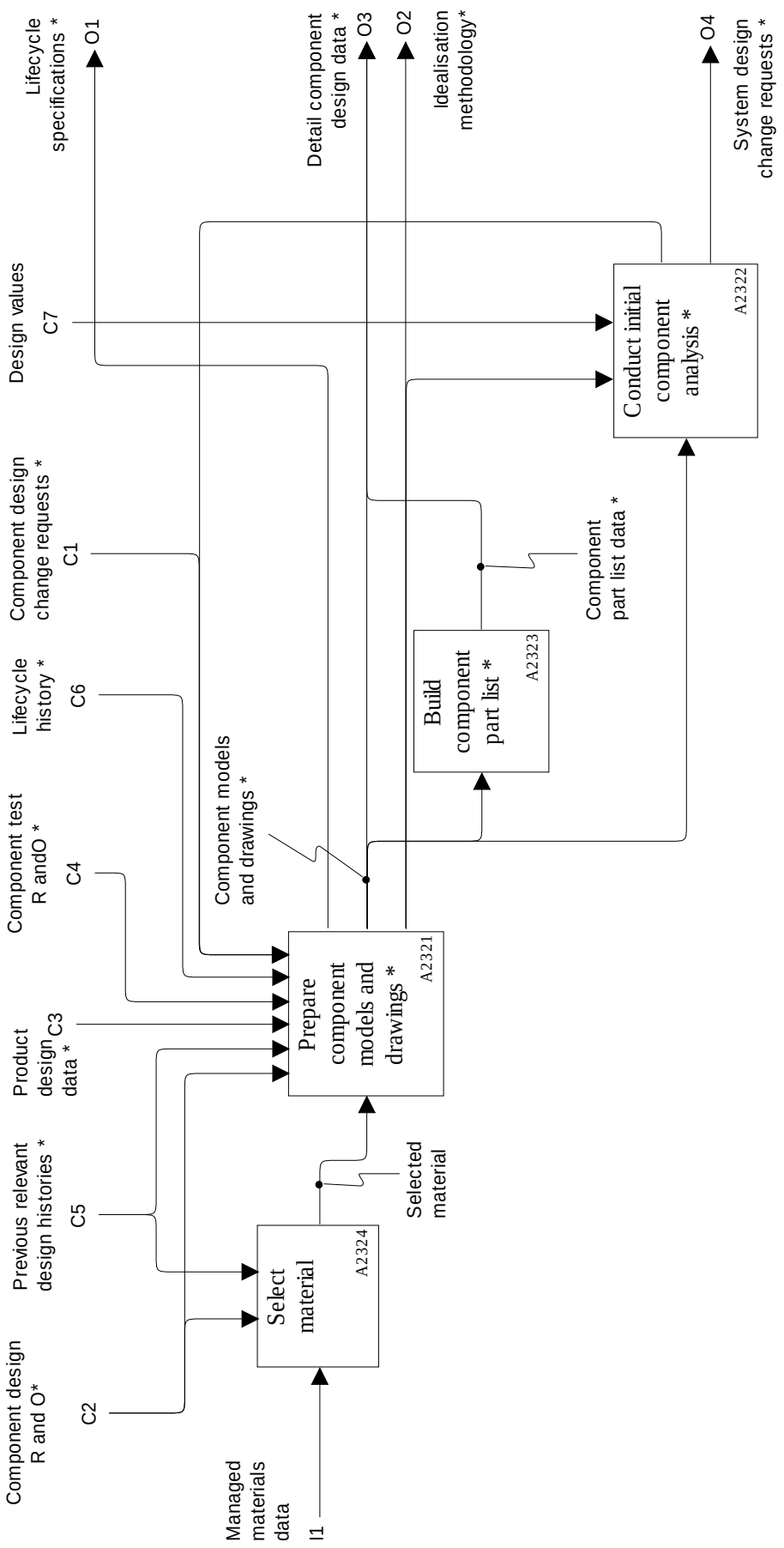


Figure F.9 — Conduct component and process design and analysis

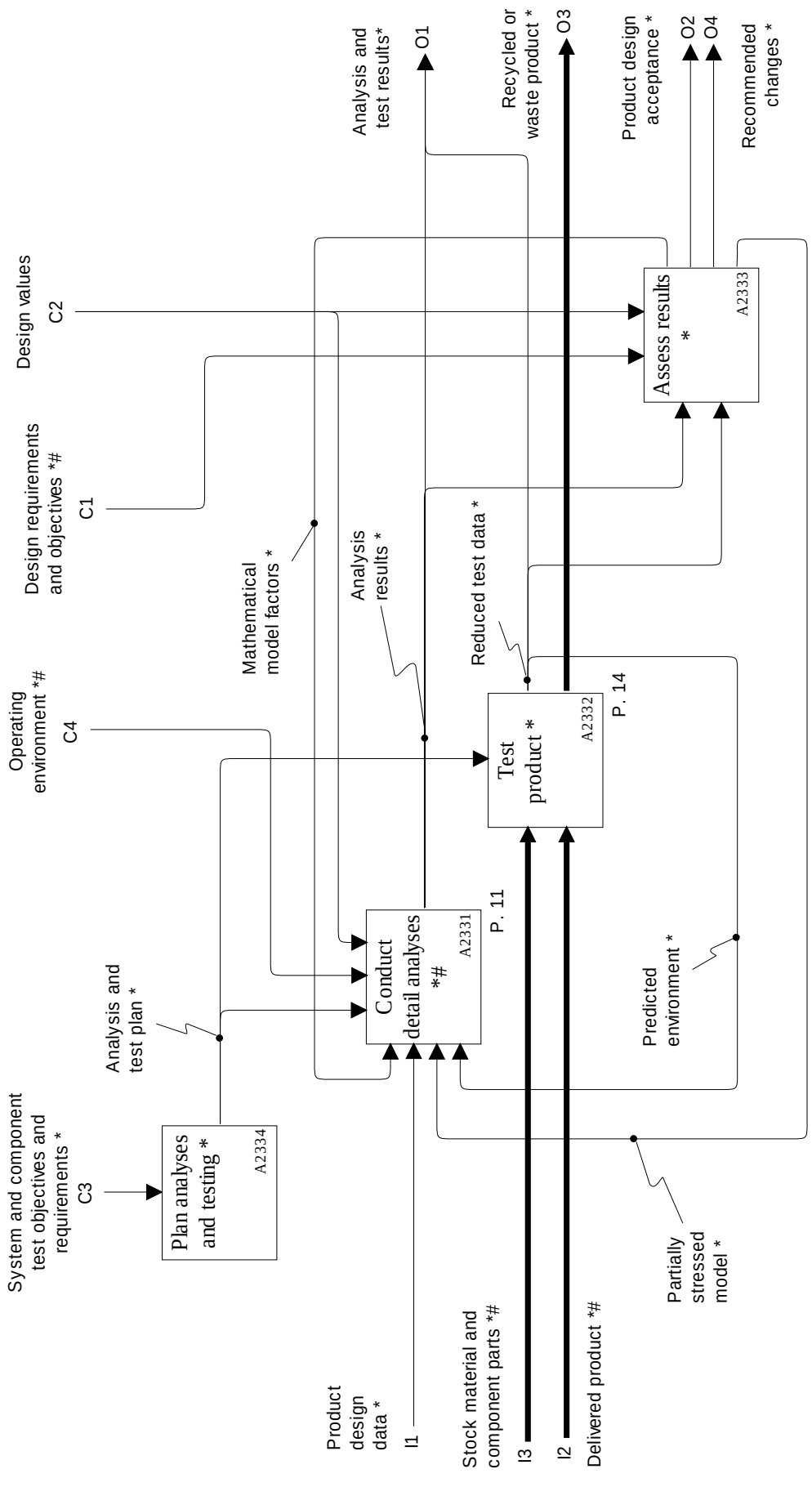


Figure F.10 — Verify product design

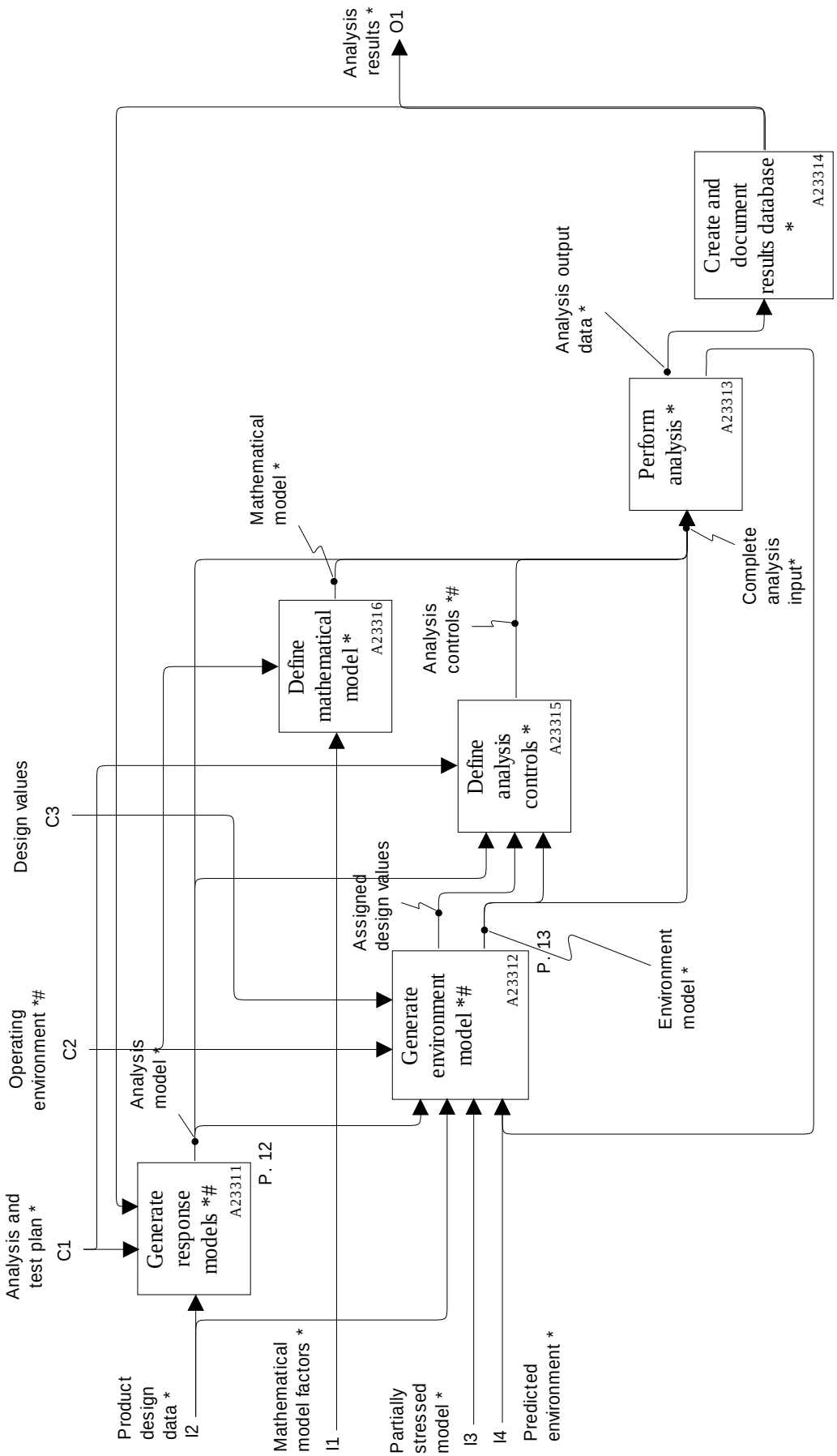


Figure F.11 — Conduct detail analysis

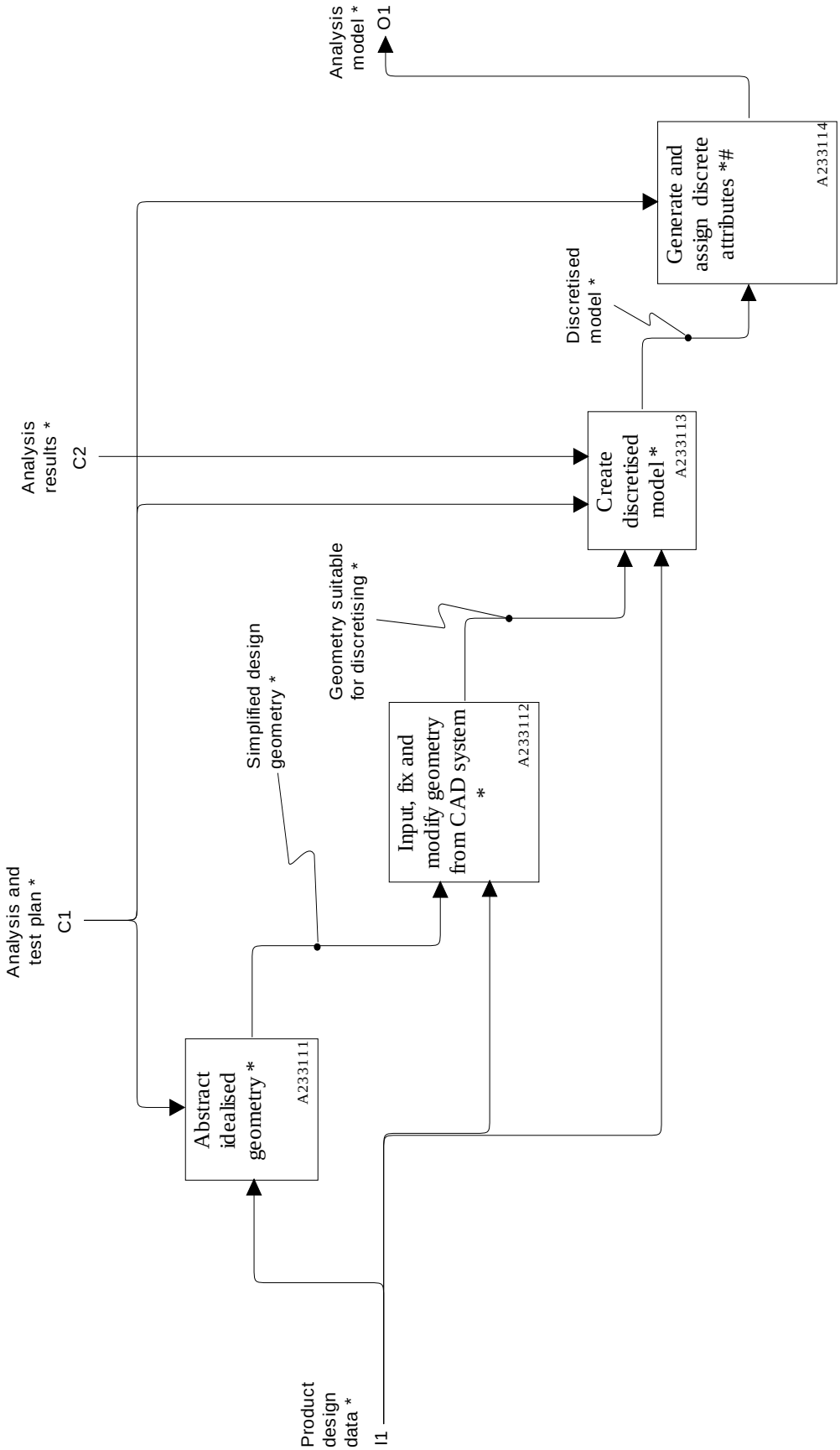


Figure F.12 — Generate response models

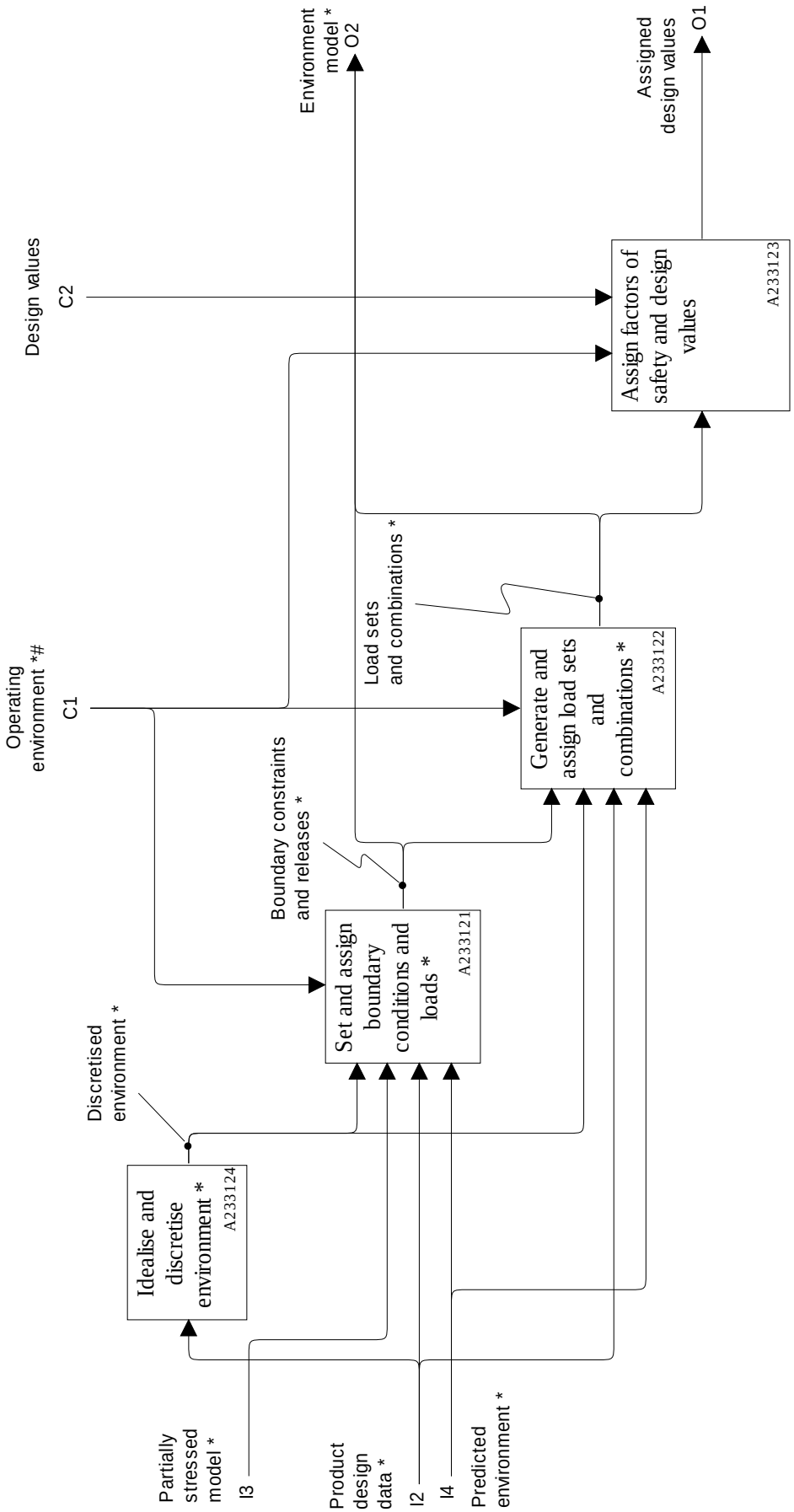


Figure F.13 — Generate environment model

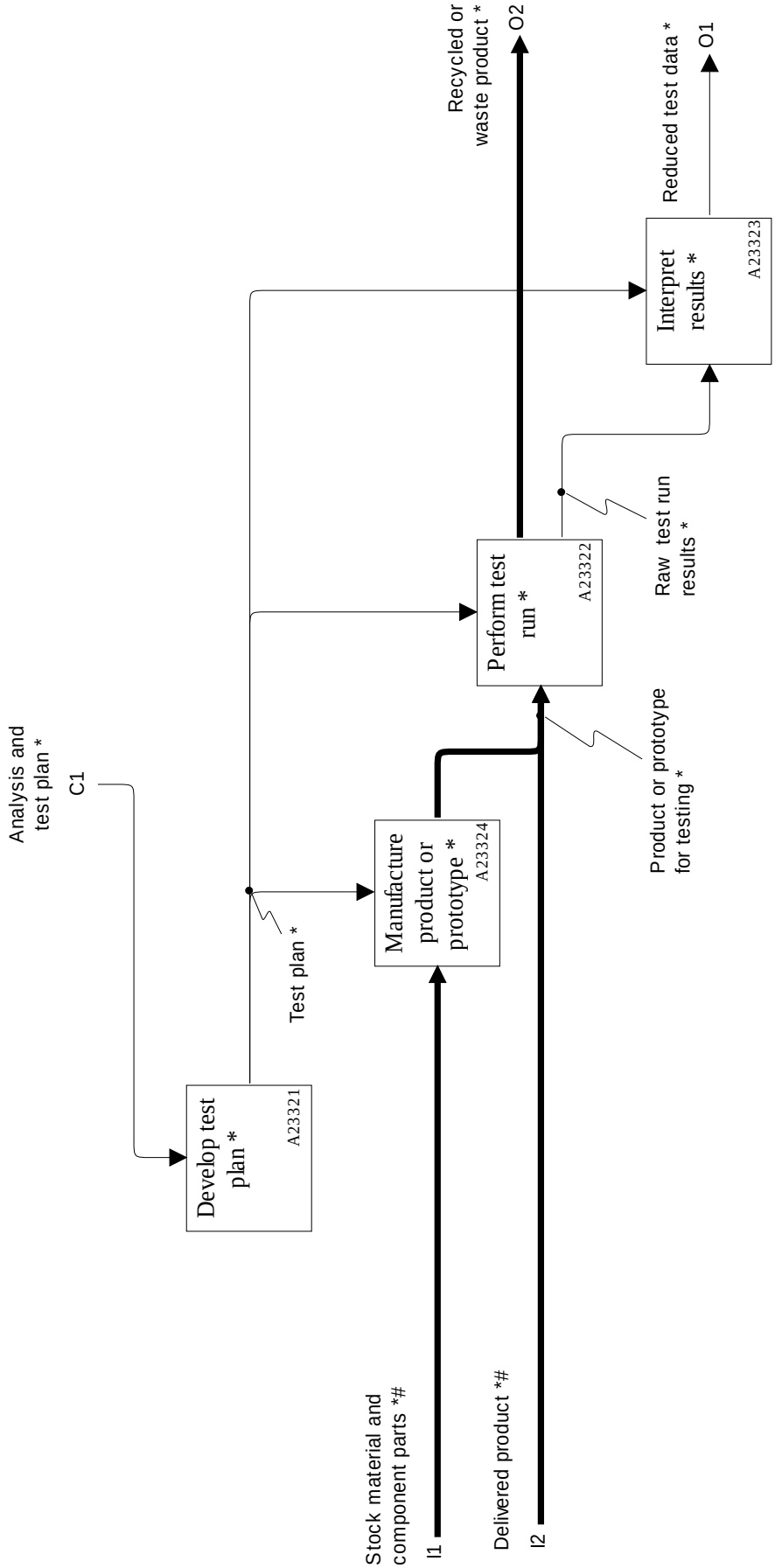


Figure F.14 — Test product

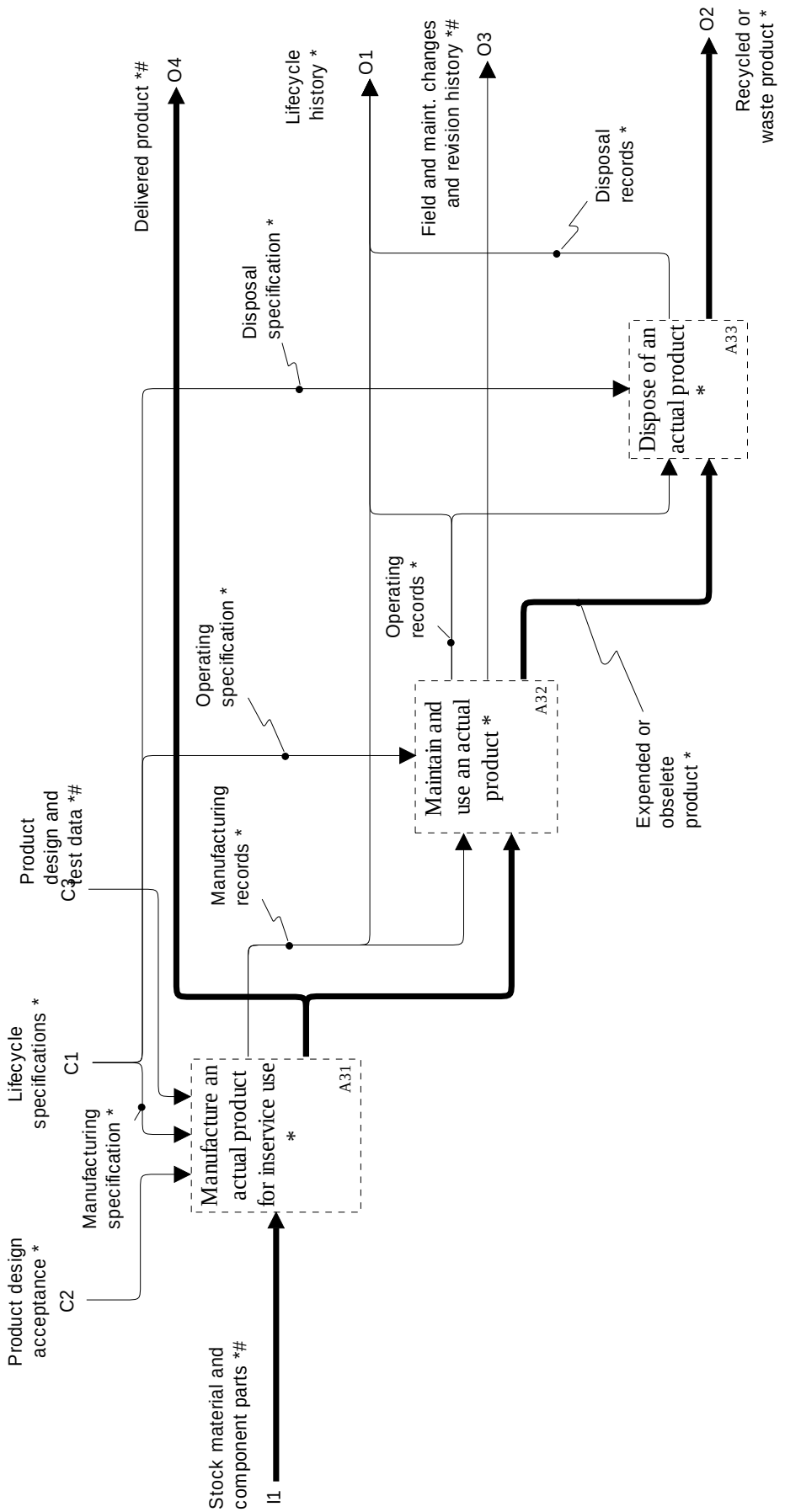


Figure F.15 — Manufacture, use and dispose of an actual product

Annex G
(informative)

Application reference model

This annex provides the digrams for the application reference model for this part of ISO 10303. The application reference model is a graphical representation of the structure and constraints of the application objects specified in Clause 4. The graphical form of the application reference model is presented in EXPRESS-G. The application reference model is independent from any implementation method. EXPRESS-G is defined in ISO 10303-11.

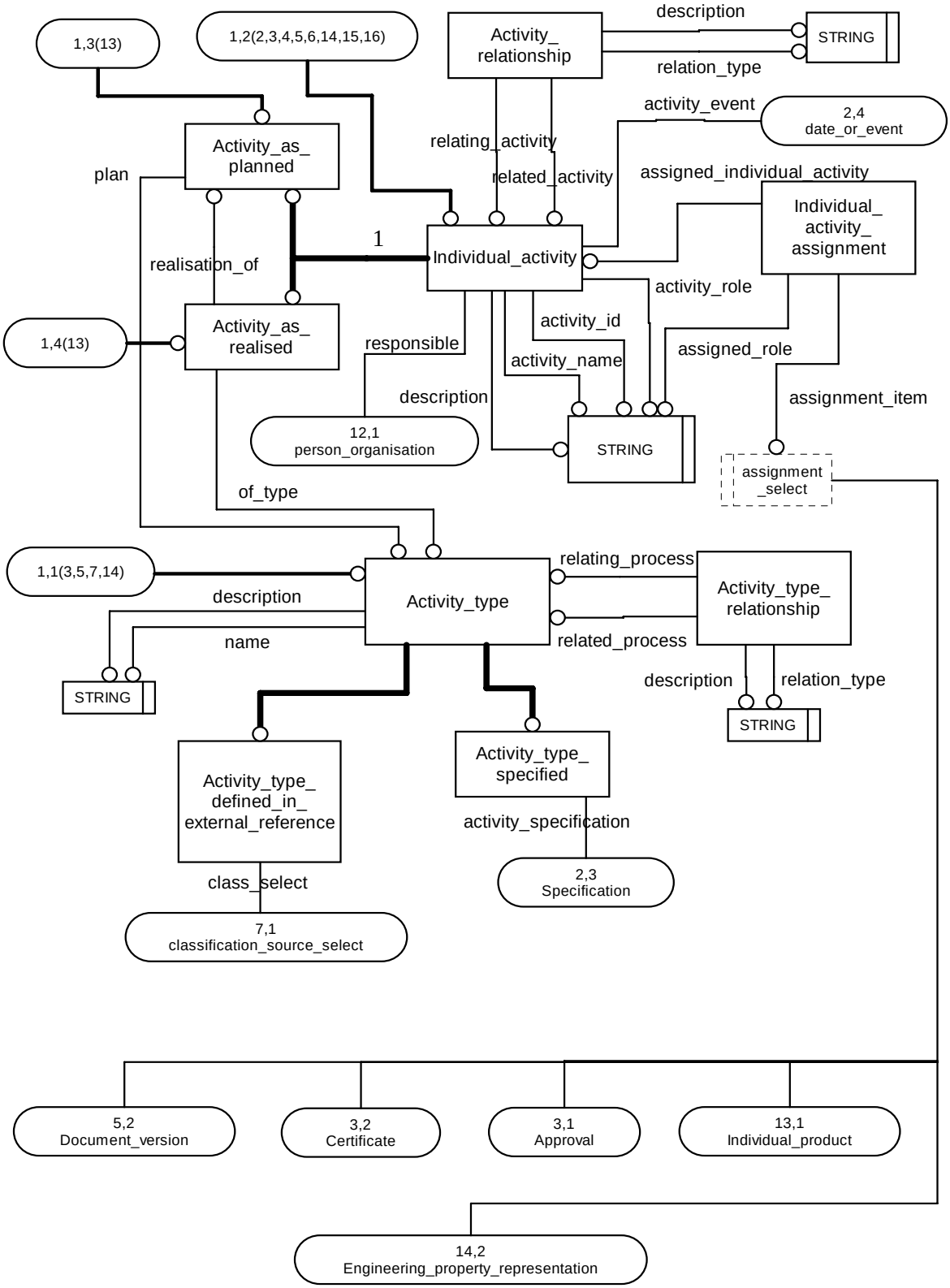


Figure G.1 — ARM EXPRESS-G diagram: activity UoF (1of 18)

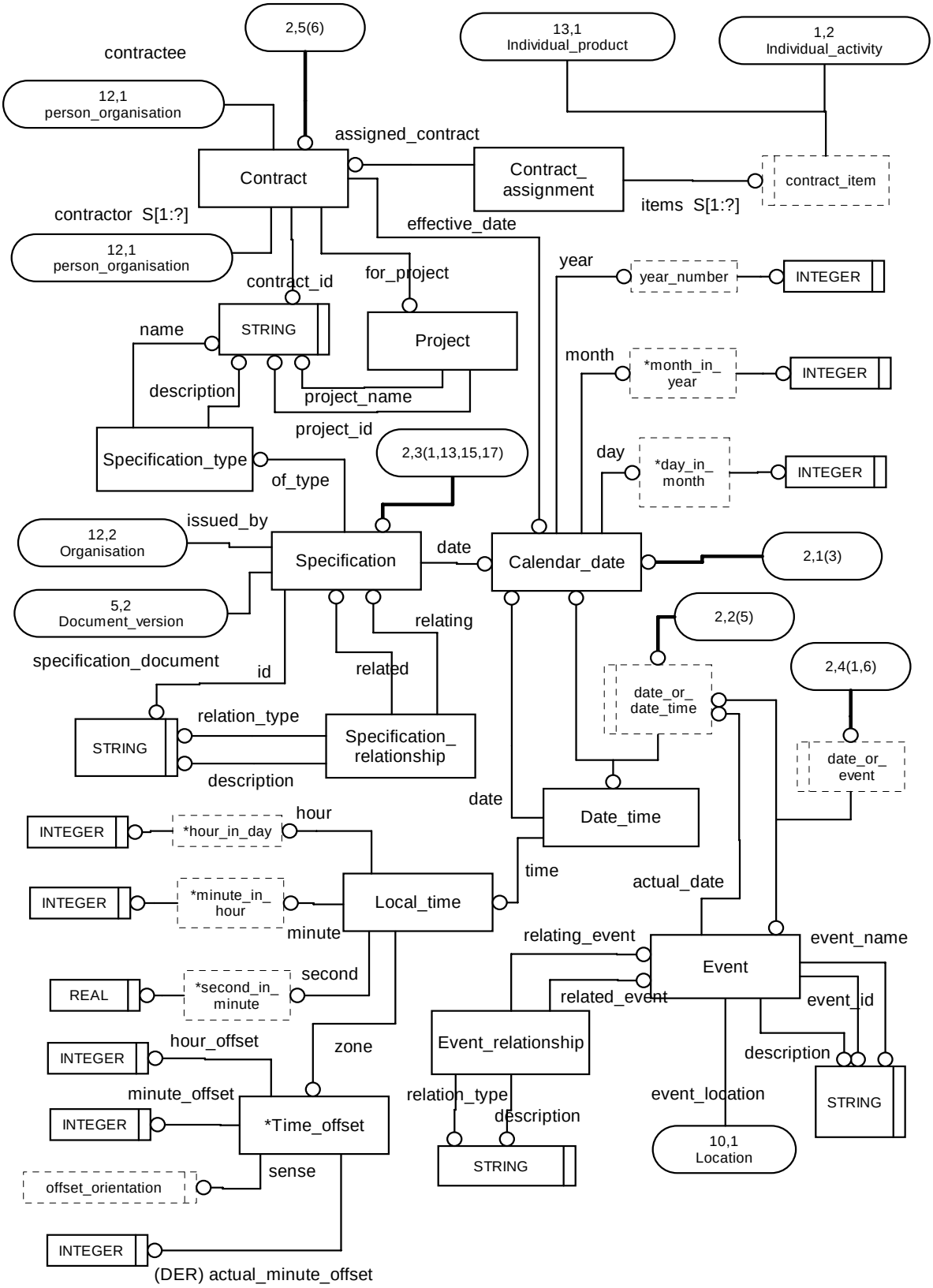


Figure G.2 — ARM EXPRESS-G diagram: administration UoF (2 of 18)

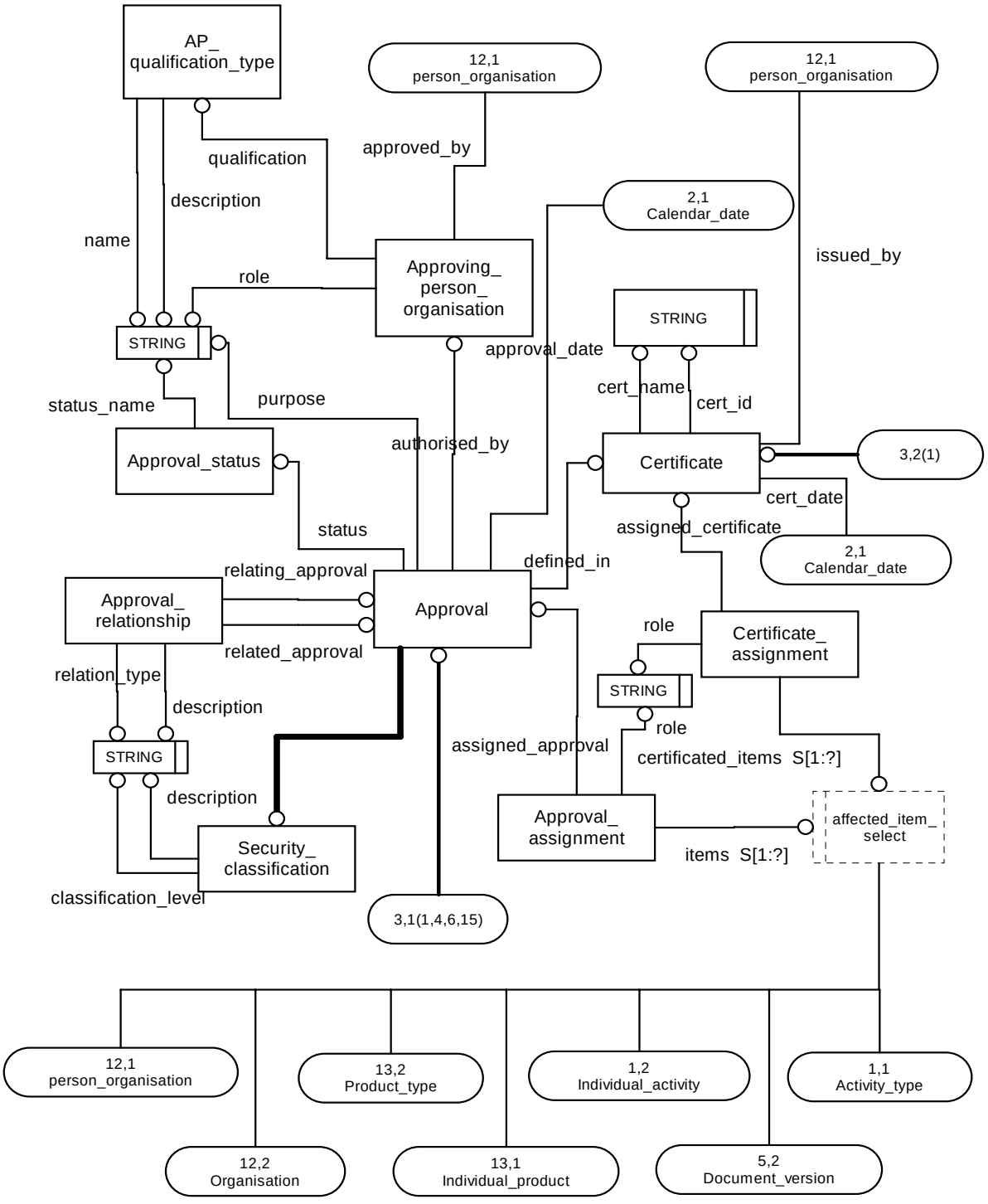


Figure G.3 — ARM EXPRESS-G diagram: approval UoF (3 of 18)

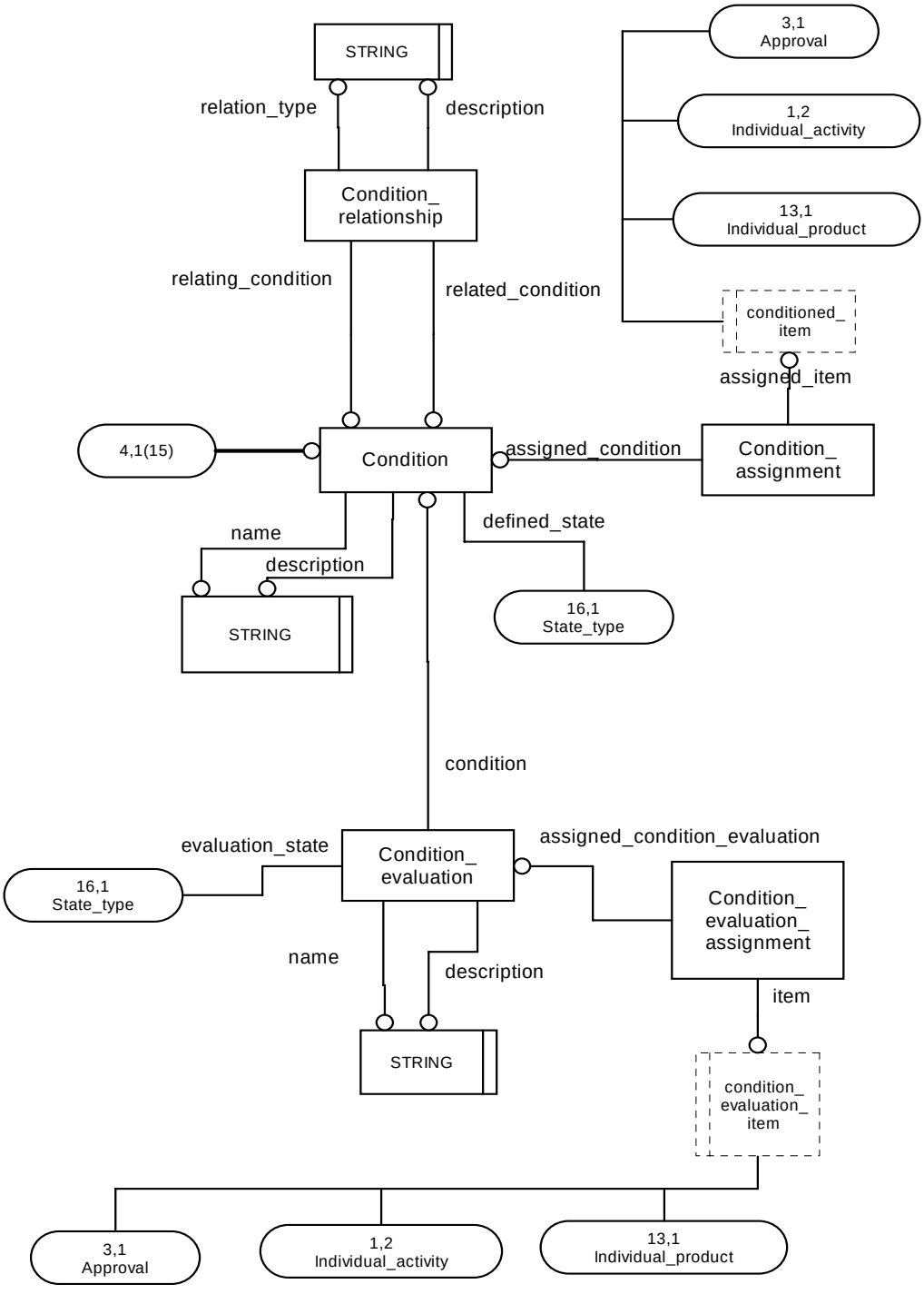


Figure G.4 — ARM EXPRESS-G diagram: condition UoF (4 of 18)

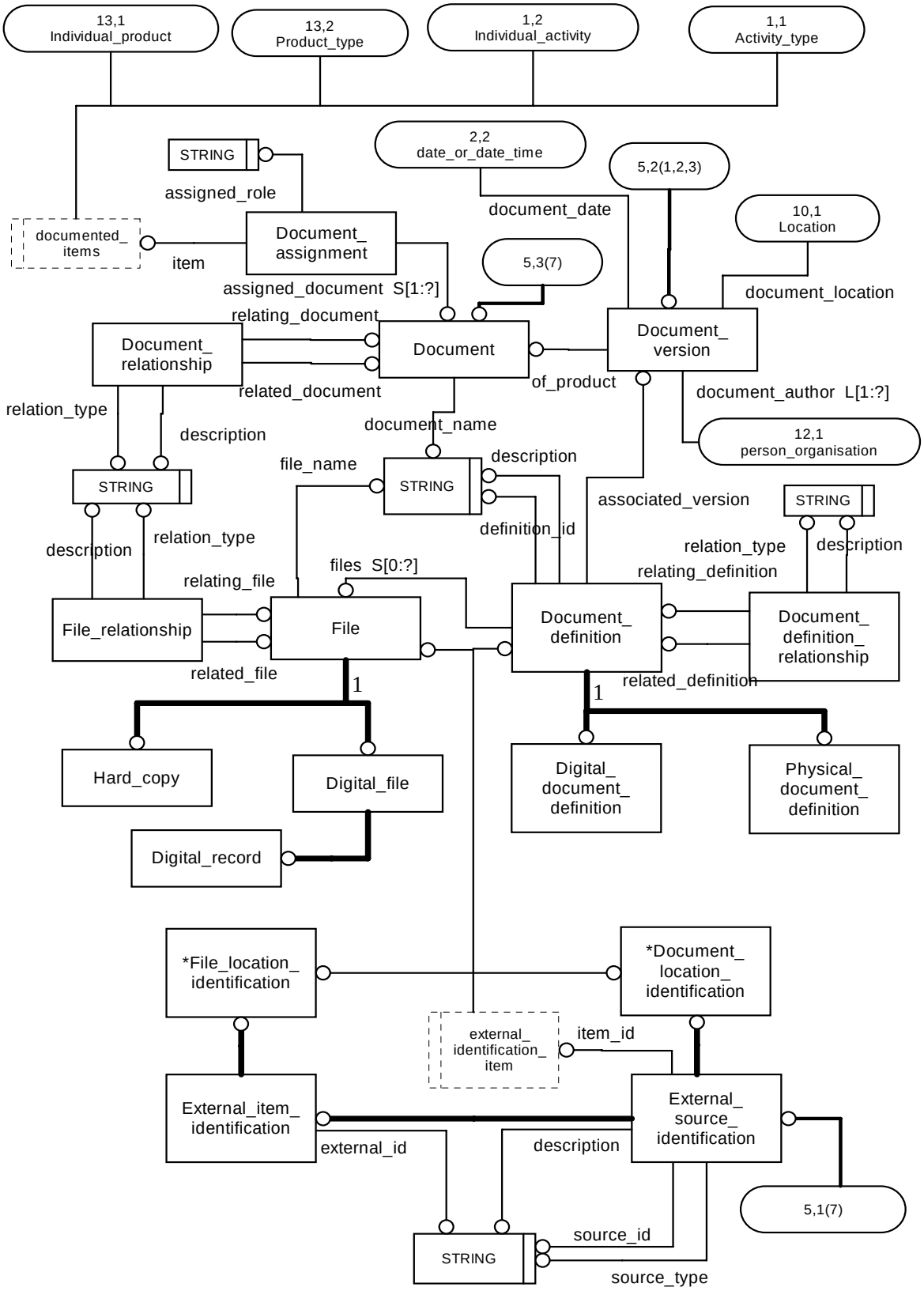


Figure G.5 — ARM EXPRESS-G diagram: document UoF (5 of 18)

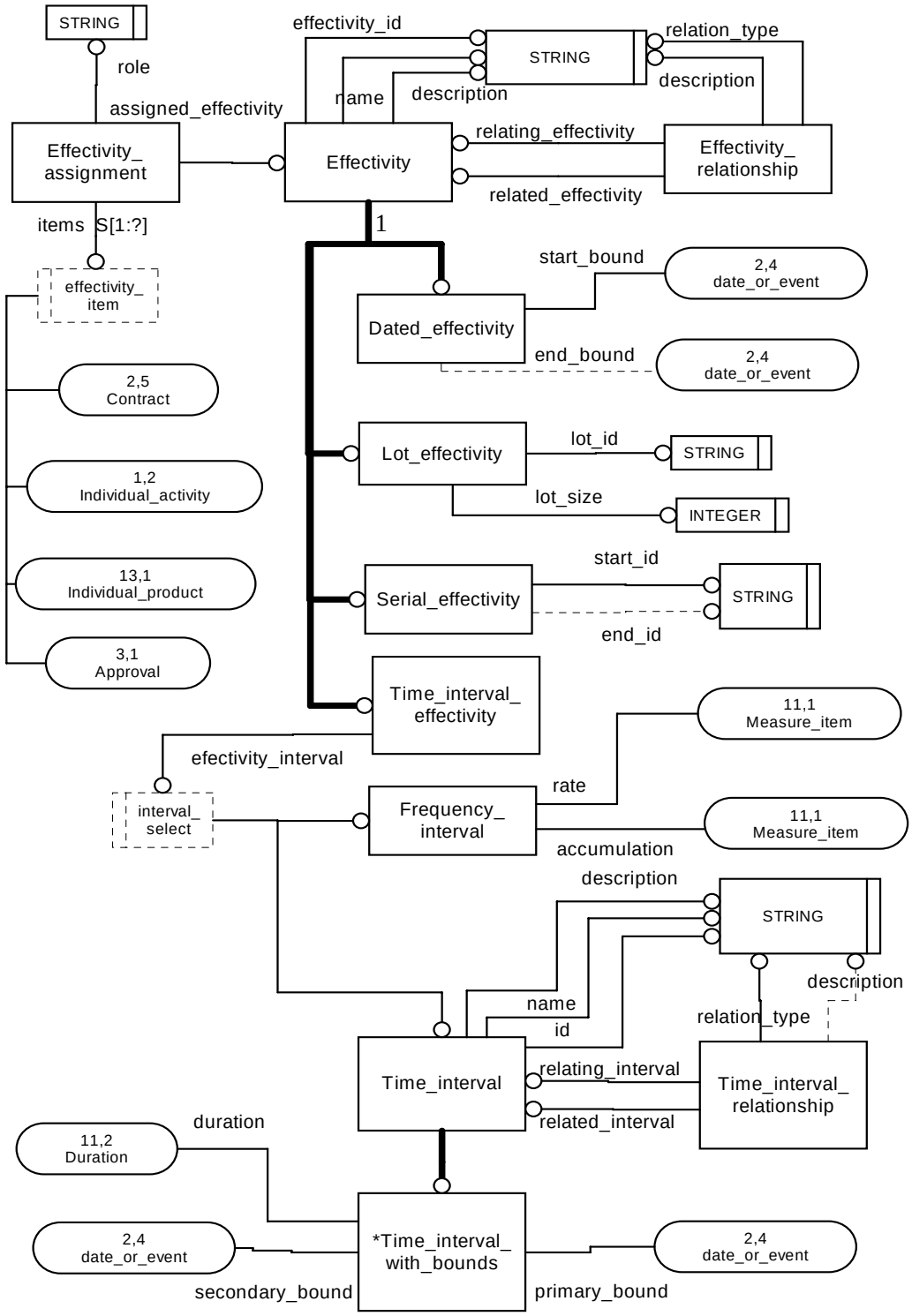


Figure G.6 — ARM EXPRESS-G diagram: effectivity UoF (6 of 18)

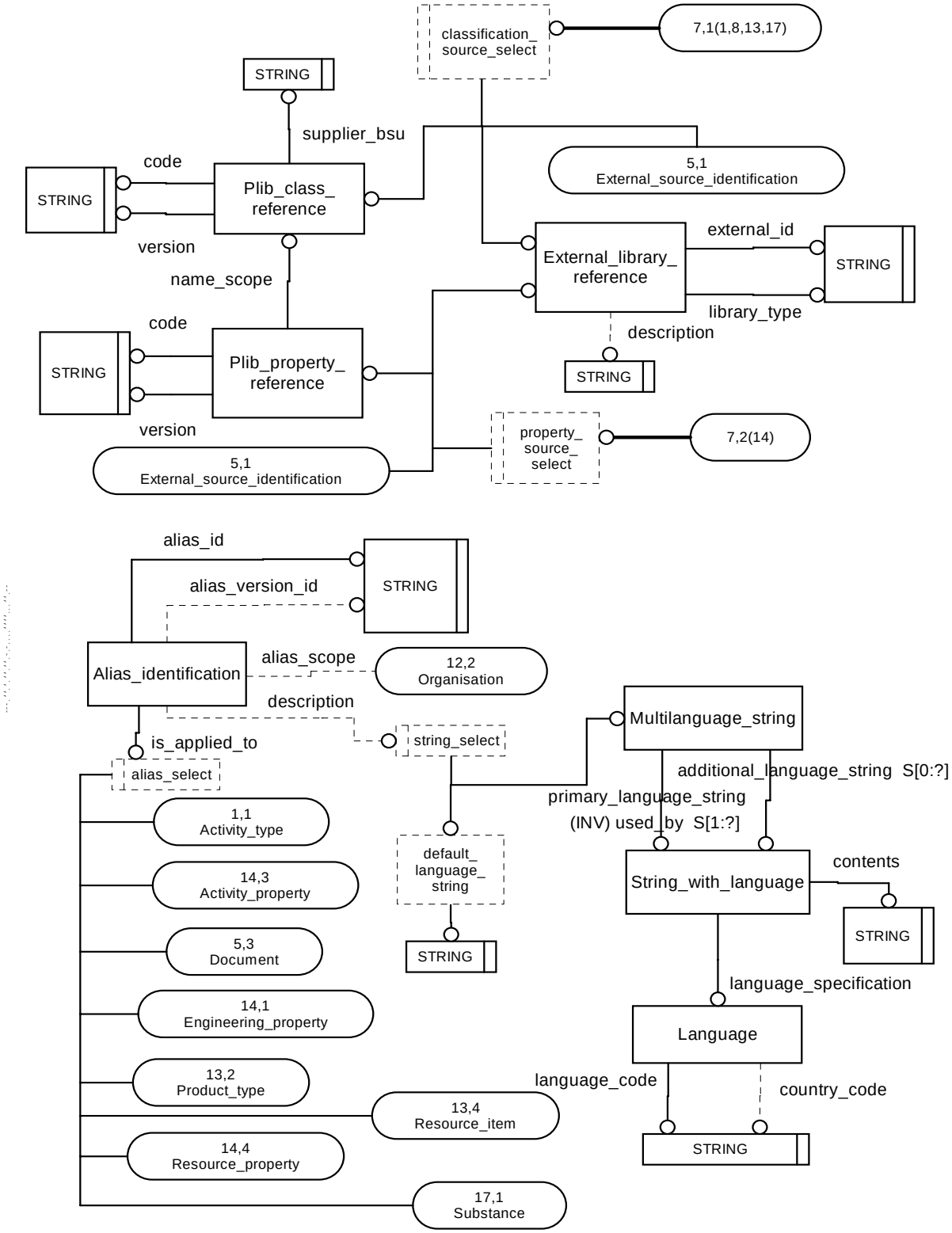


Figure G.7 — ARM EXPRESS-G diagram: external_reference UoF (7 of 18)

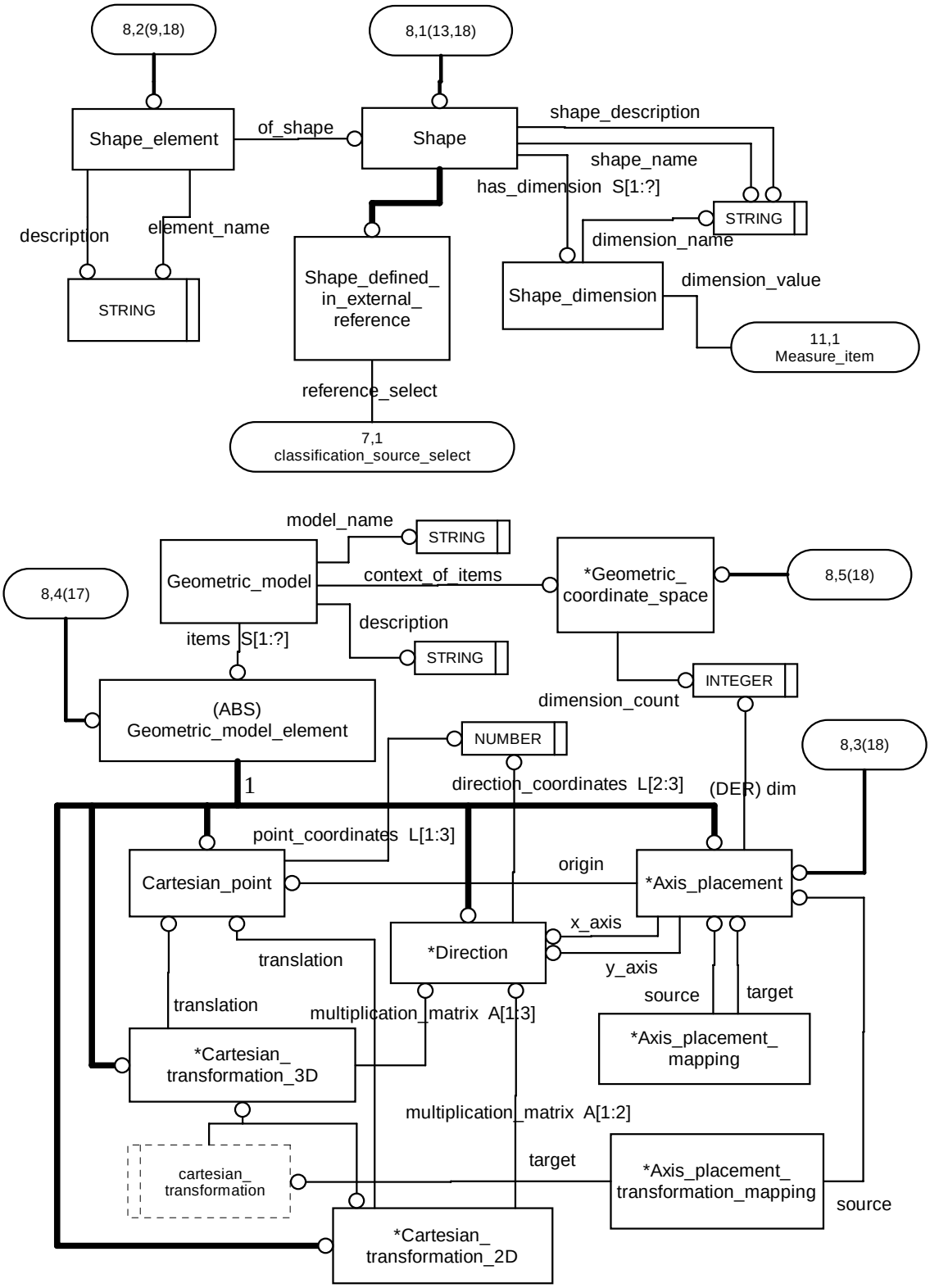


Figure G.8 — ARM EXPRESS-G diagram: geometry UoF (8 of 18)

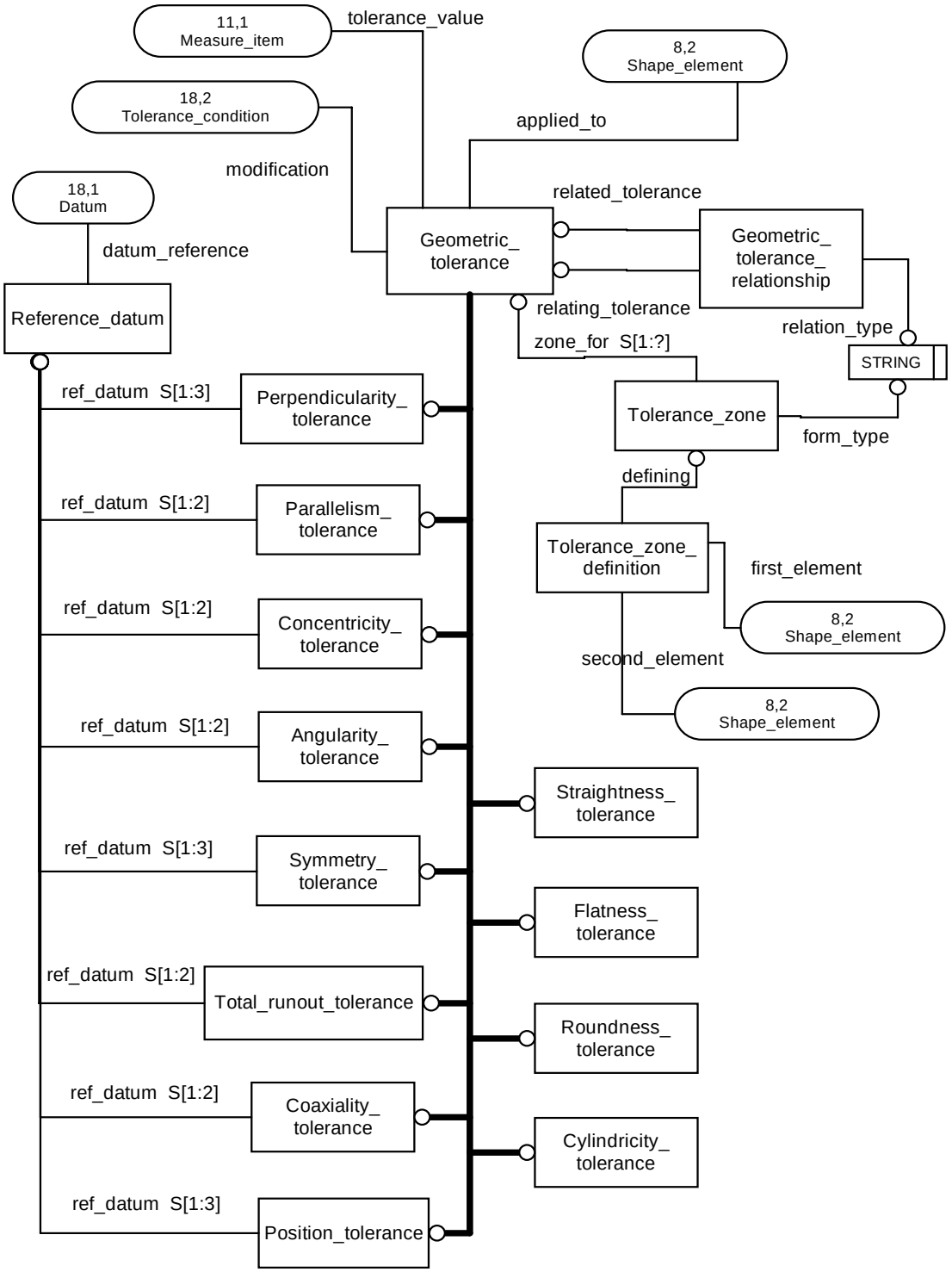


Figure G.9 — ARM EXPRESS-G diagram: geometric_tolerance UoF (9 of 18)

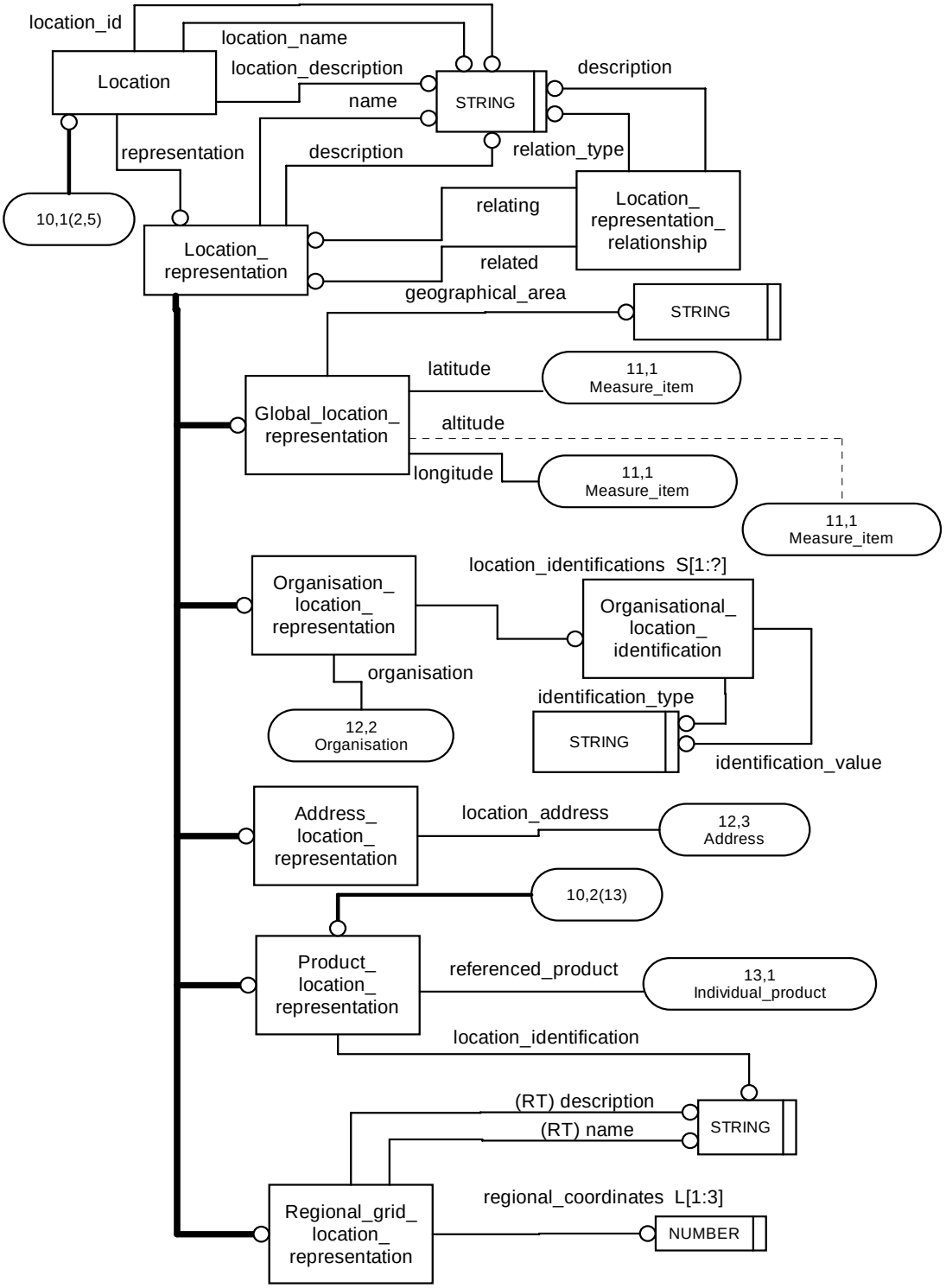


Figure G.10 — ARM EXPRESS-G diagram: location UoF (10 of 18)

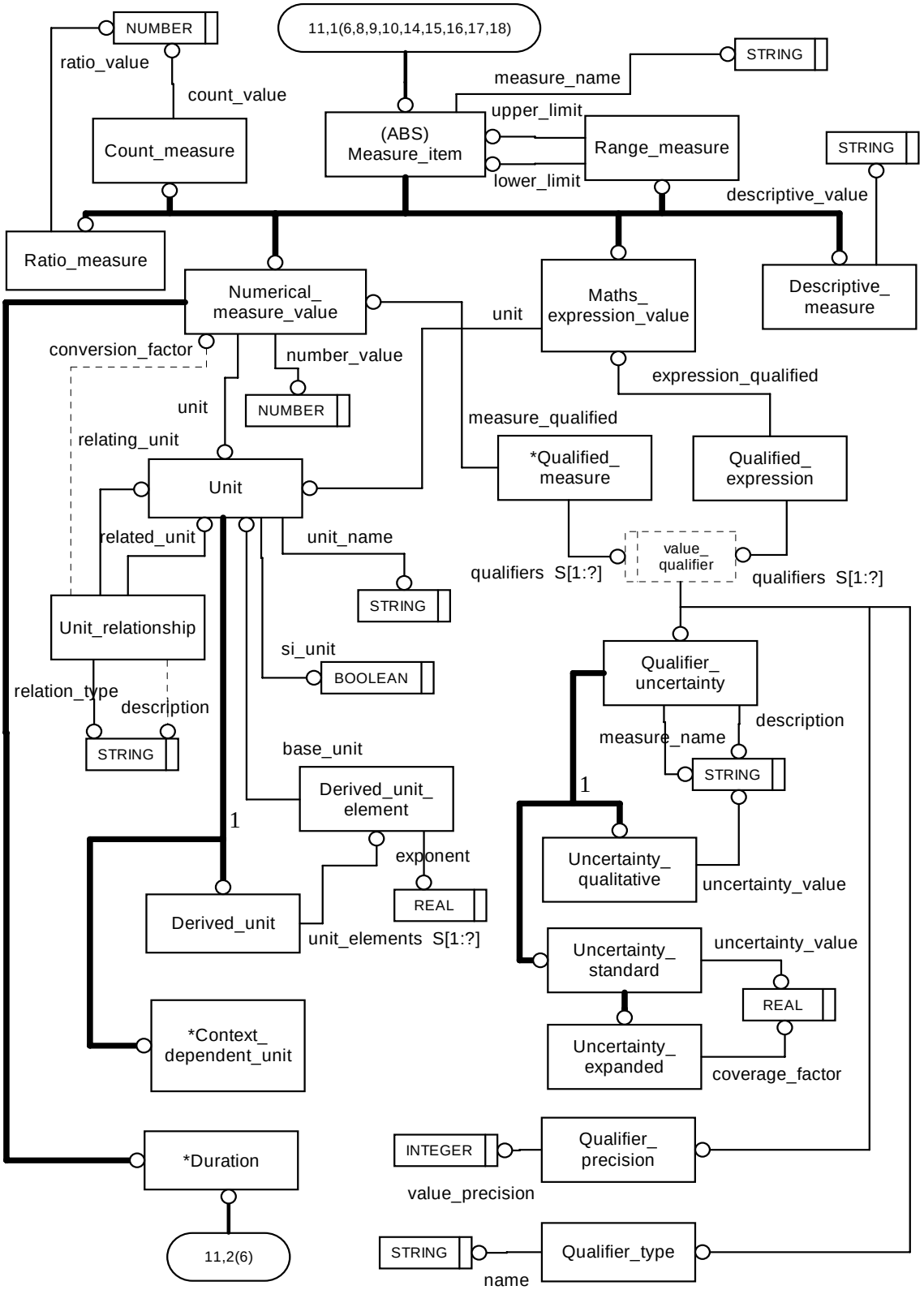


Figure G.11 — ARM EXPRESS-G diagram: measure UoF (11 of 18)

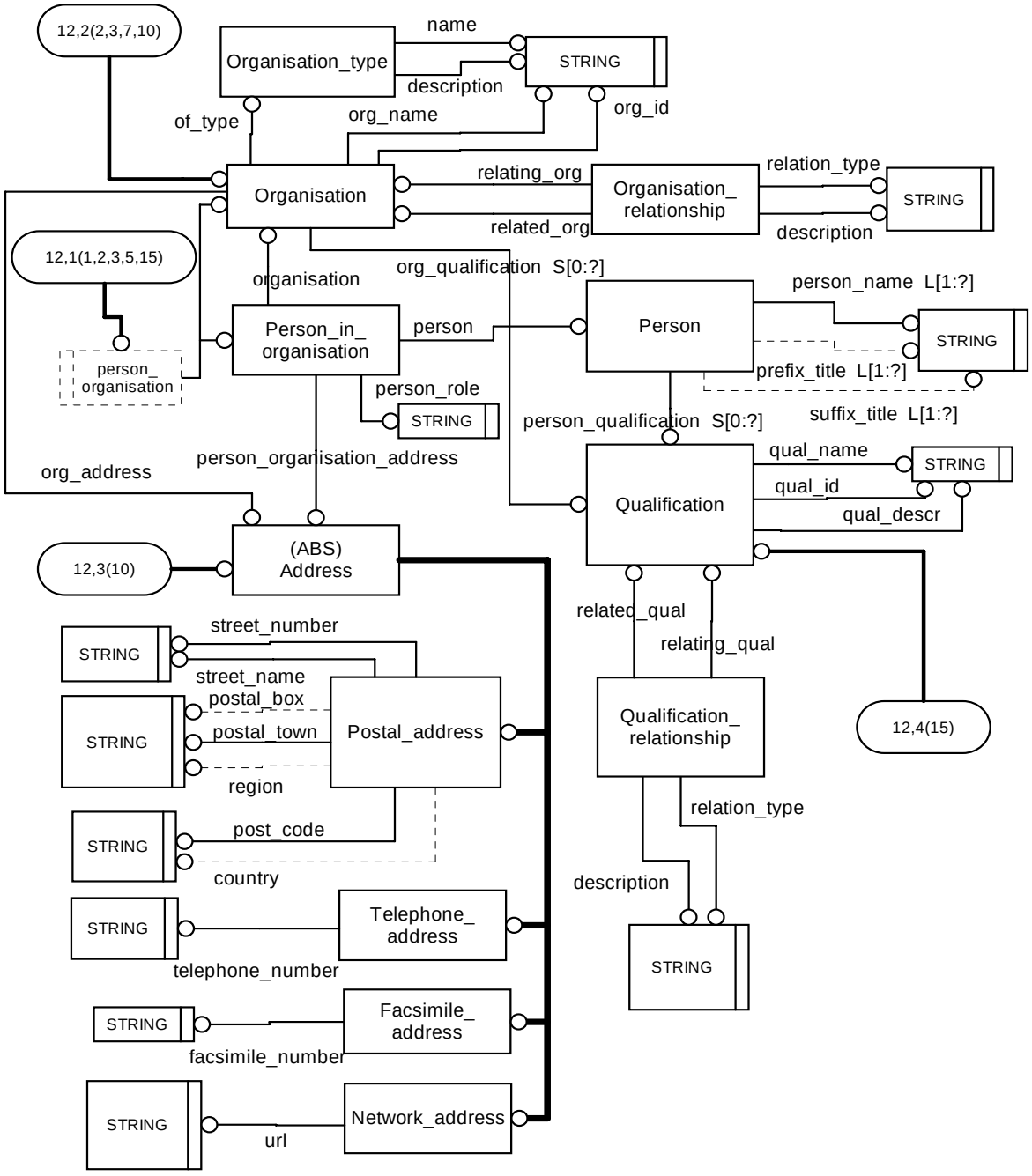


Figure G.12 — ARM EXPRESS-G diagram: person_organisation UoF (12 of 18)

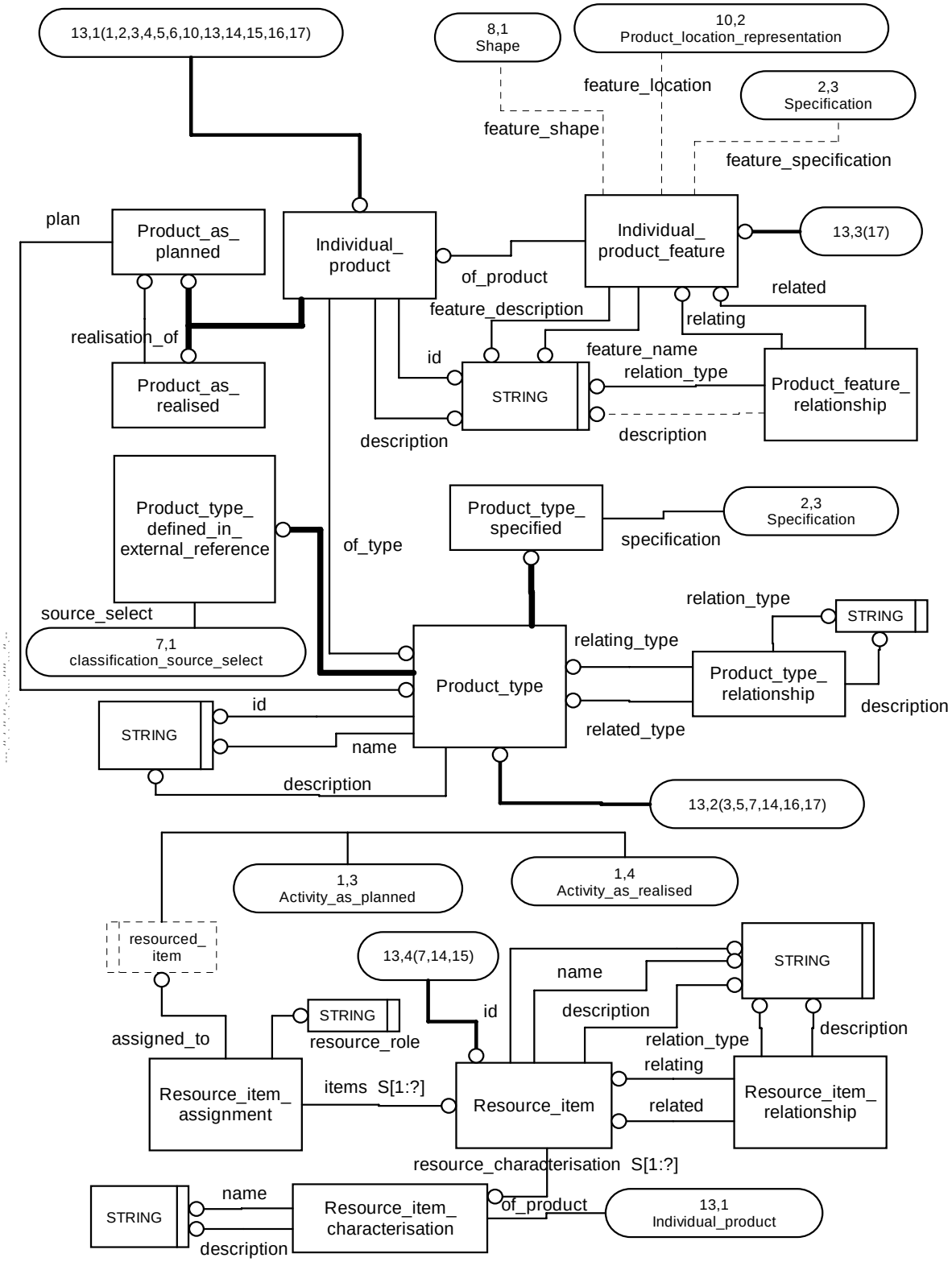


Figure G.13 — ARM EXPRESS-G diagram: product UoF (13 of 18)

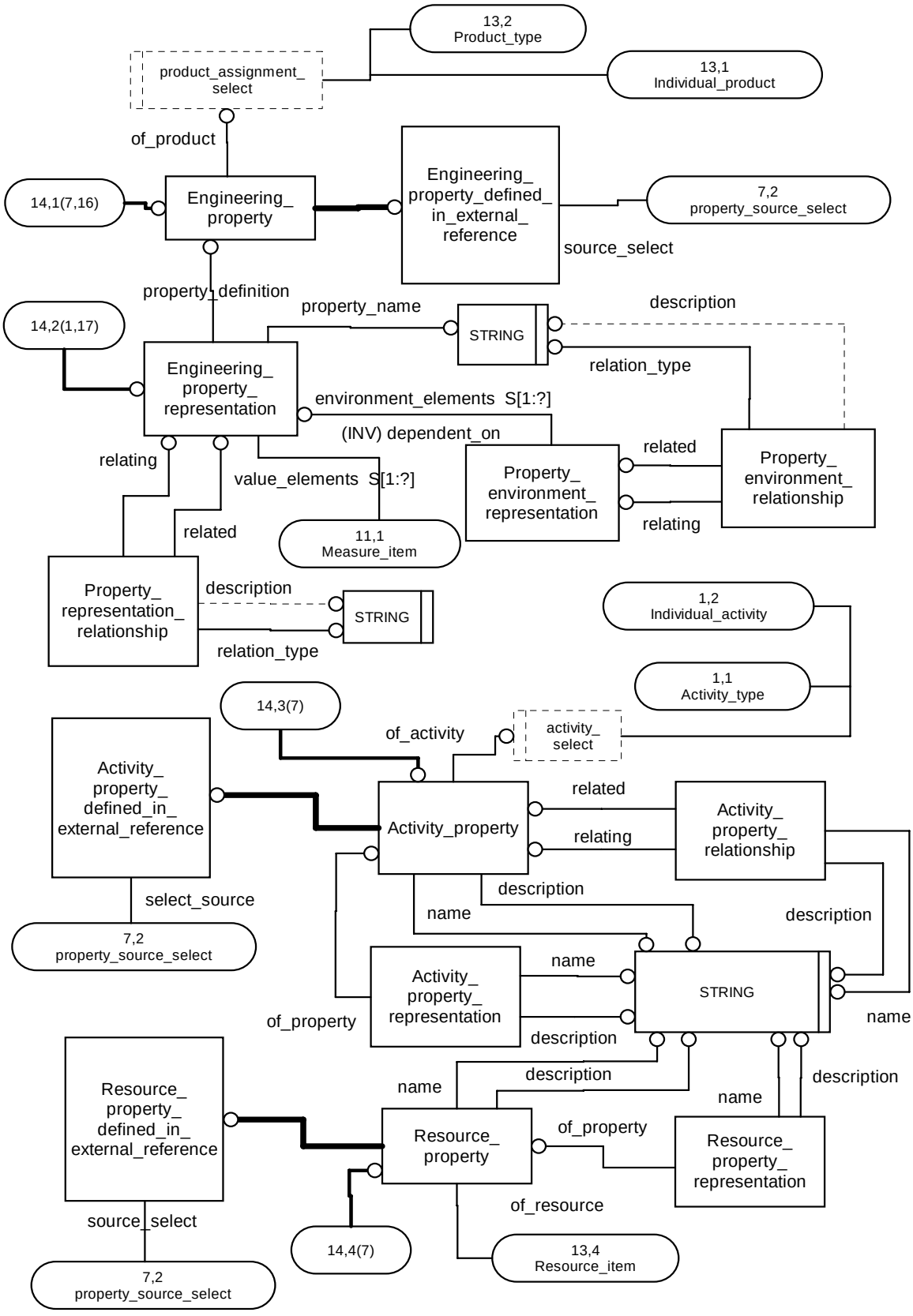


Figure G.14 — ARM EXPRESS-G diagram: properties UoF (14 of 18)

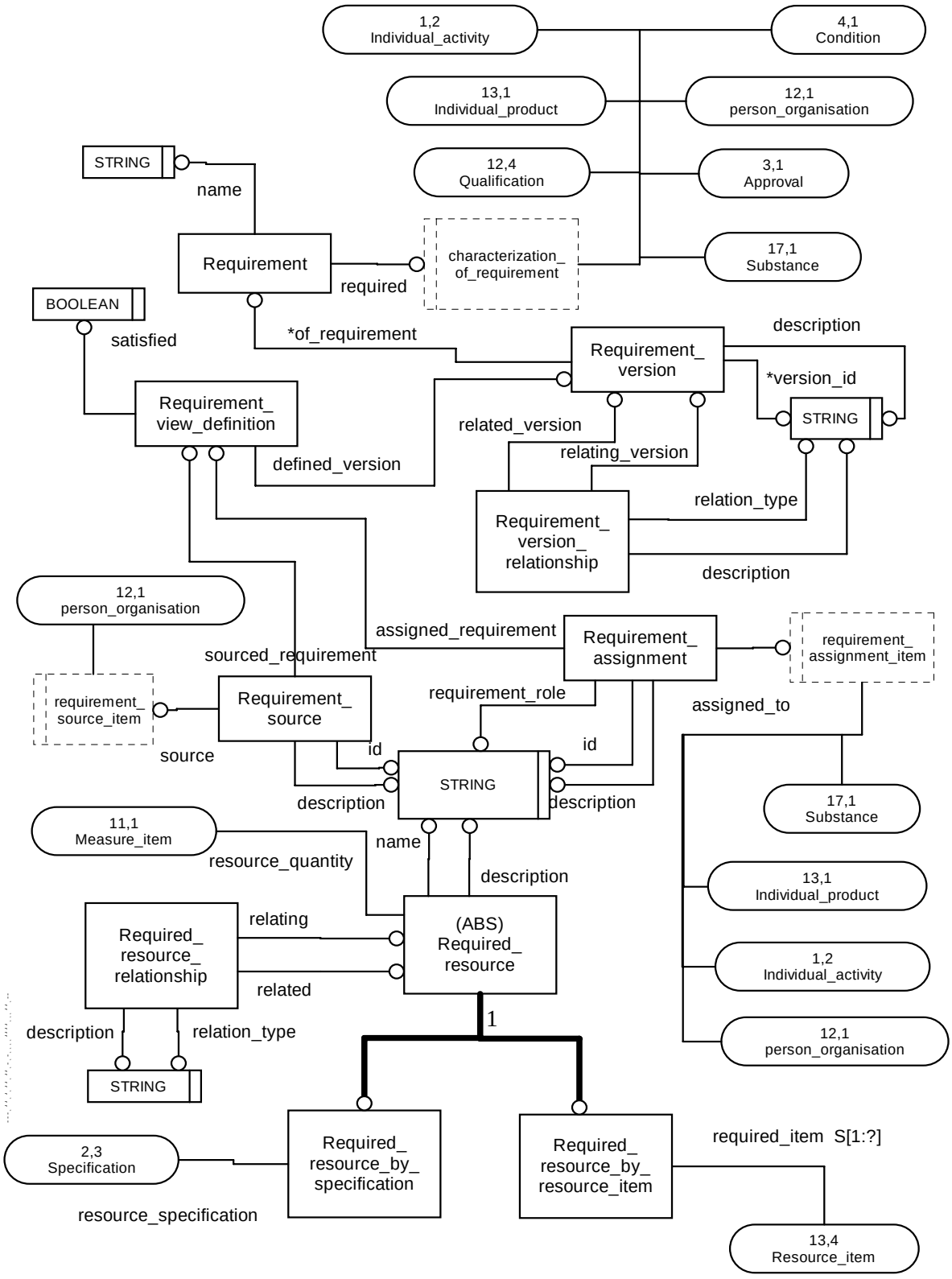


Figure G.15 — ARM EXPRESS-G diagram: requirements UoF (15 of 18)

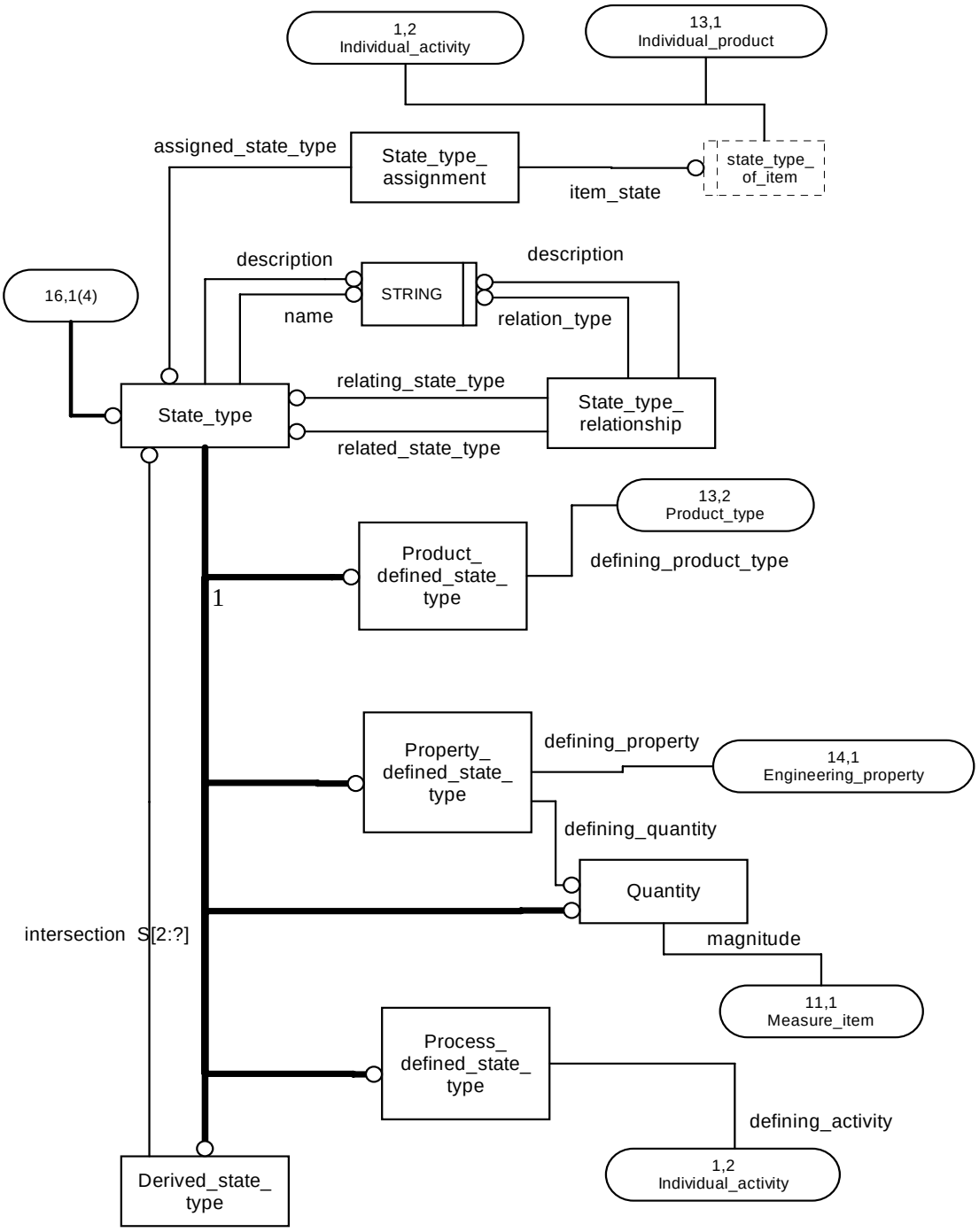


Figure G.16 — ARM EXPRESS-G diagram: state UoF (16 of 18)

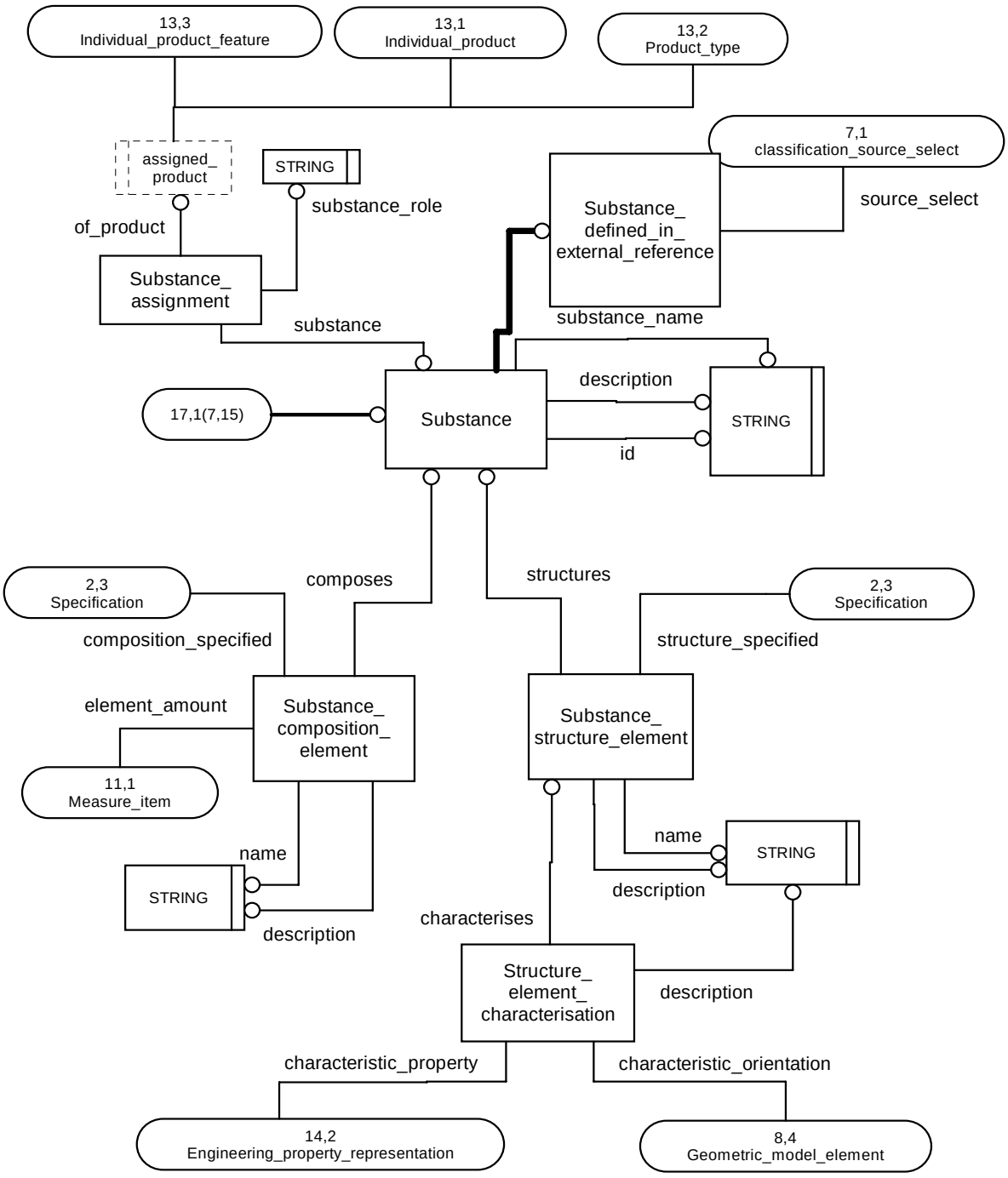


Figure G.17 — ARM EXPRESS-G diagram: substance UoF (17 of 18)

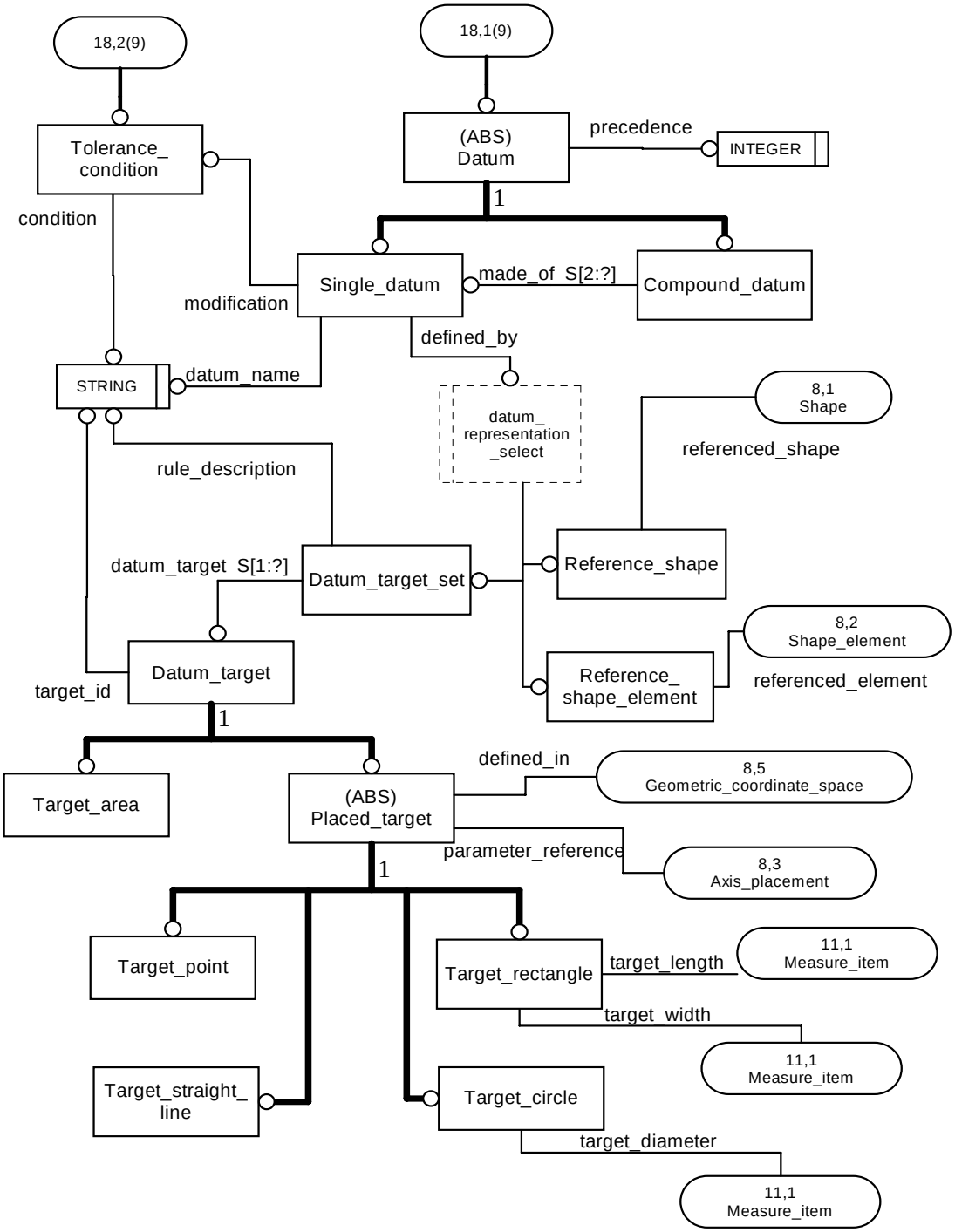


Figure G.18 — ARM EXPRESS-G diagram: tolerance_datum (18 of 18)

Annex H
(informative)

AIM EXPRESS-G

The diagrams in this annex correspond to the expanded listing of the Aplication Interpreted Model in Annex A. The diagrams use the EXPRESS-G graphical notation for the EXPRESS language. EXPRESS-G is defined in ISO 10303-11.

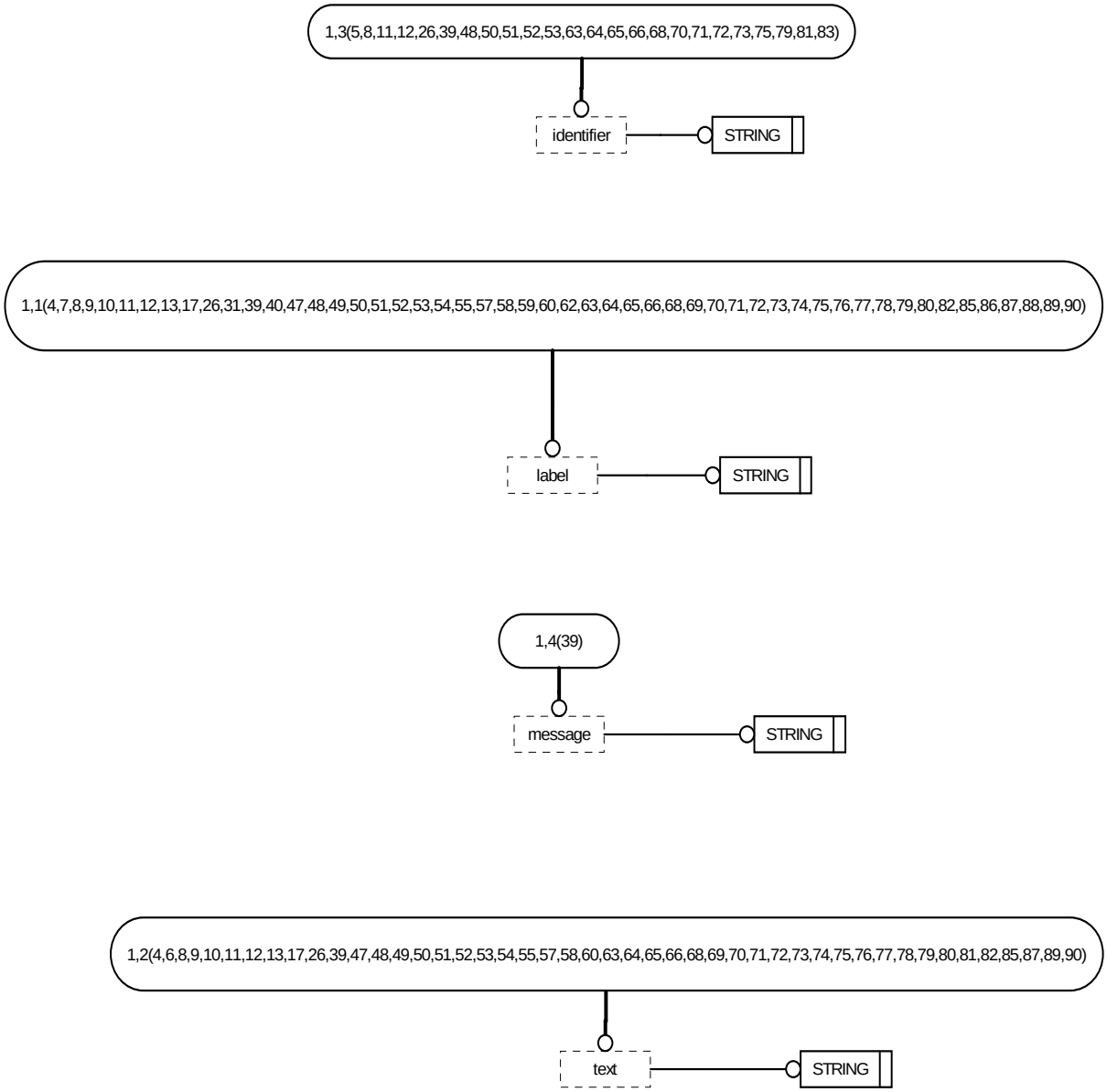


Figure H.1 — AIM EXPRESS-G diagram: common resources (1 of 91)

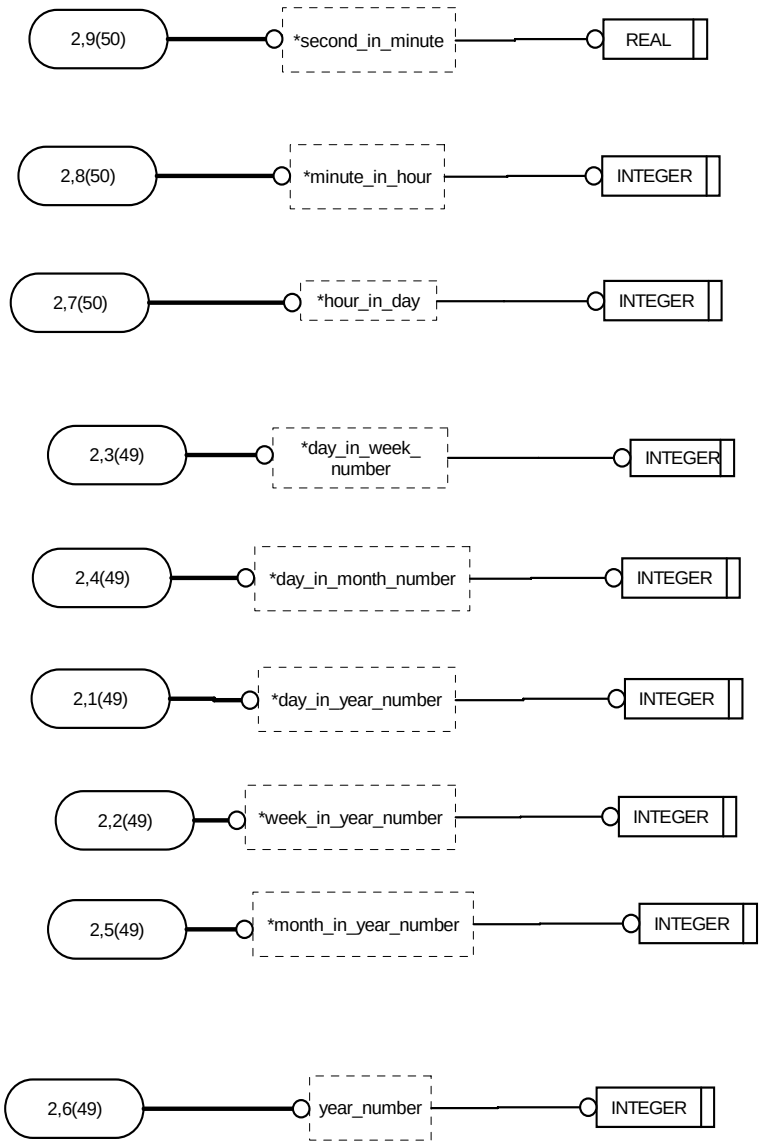


Figure H.2 — AIM EXPRESS-G diagram: date_time types (2 of 91)

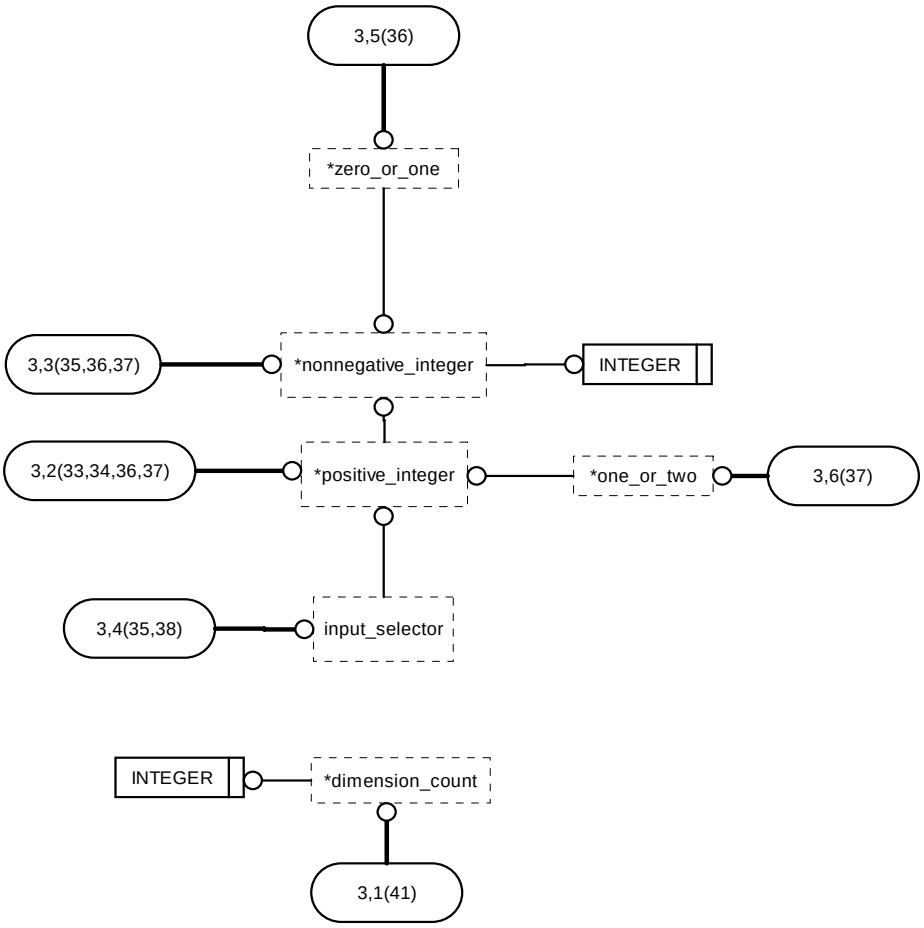


Figure H.3 — AIM EXPRESS-G diagram: integer numbers (3 of 91)

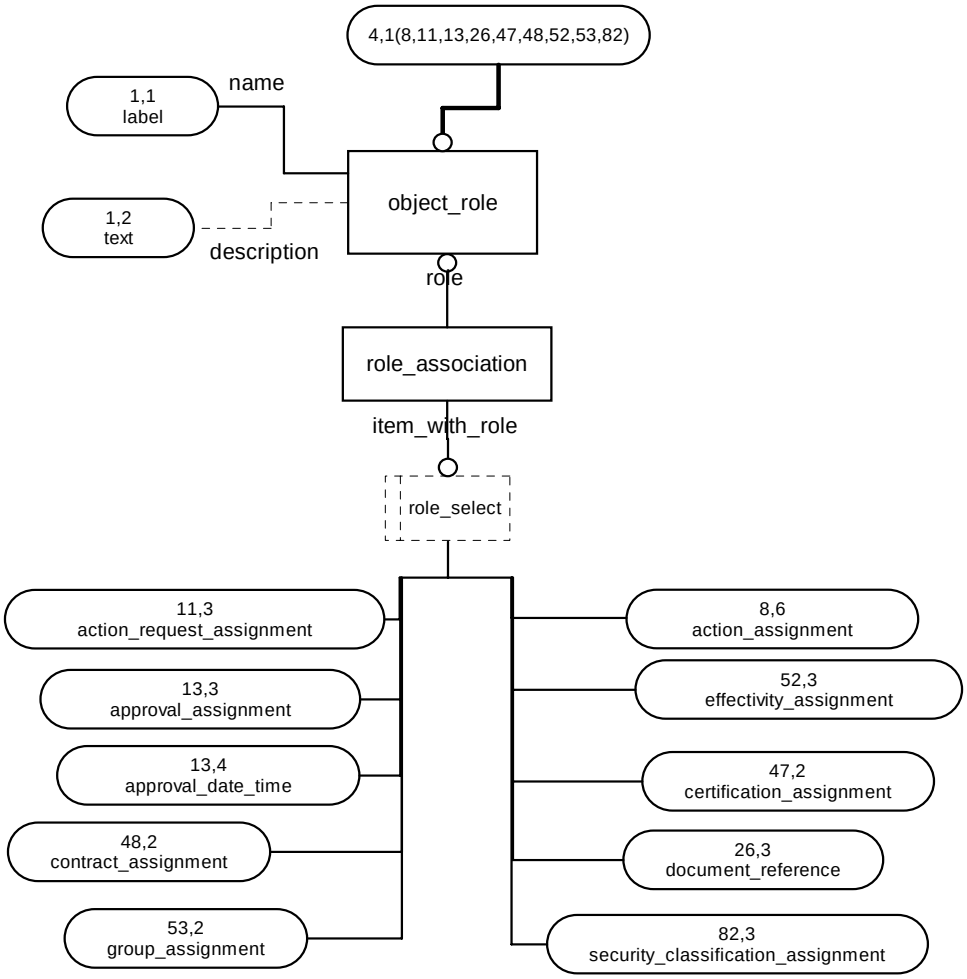


Figure H.4 — AIM EXPRESS-G diagram: object_role (4 of 91)

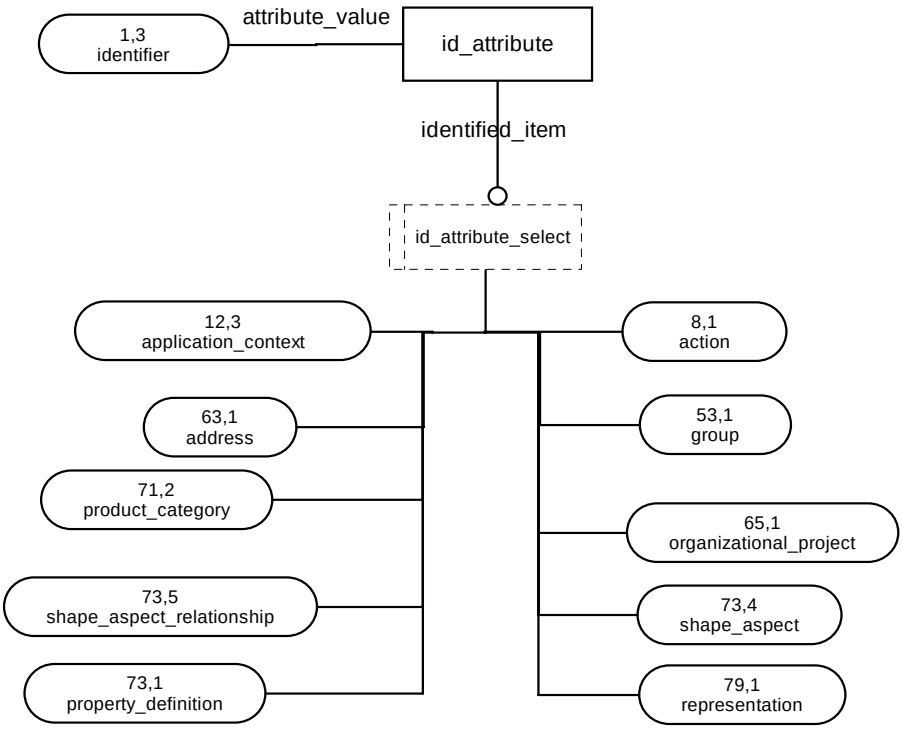


Figure H.5 — AIM EXPRESS-G diagram: id_attribute (5 of 91)

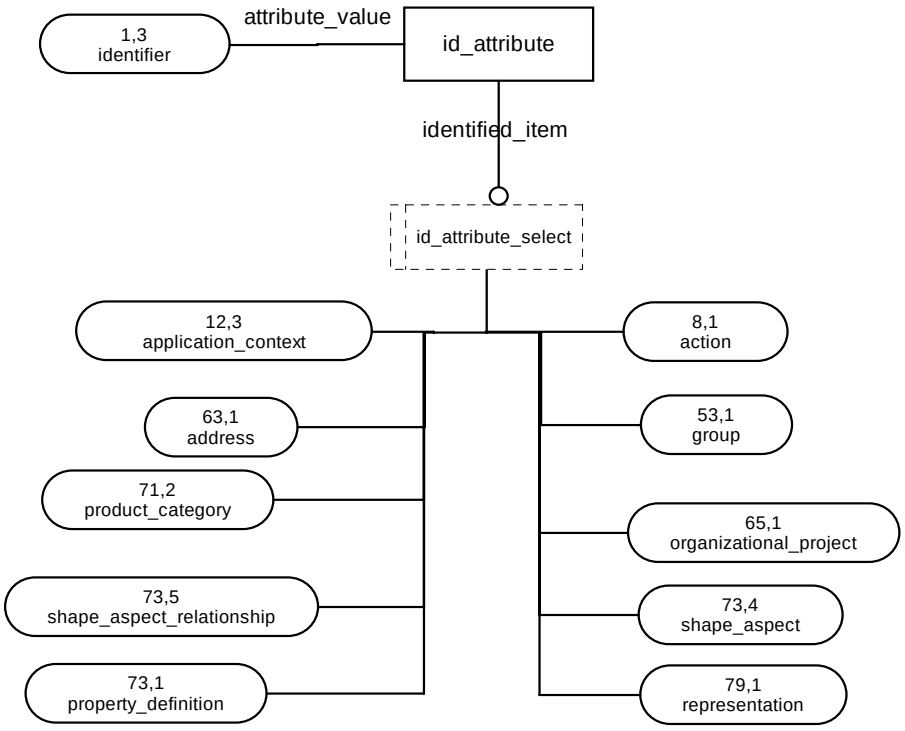


Figure H.6 — AIM EXPRESS-G diagram: description_attribute (6 of 91)

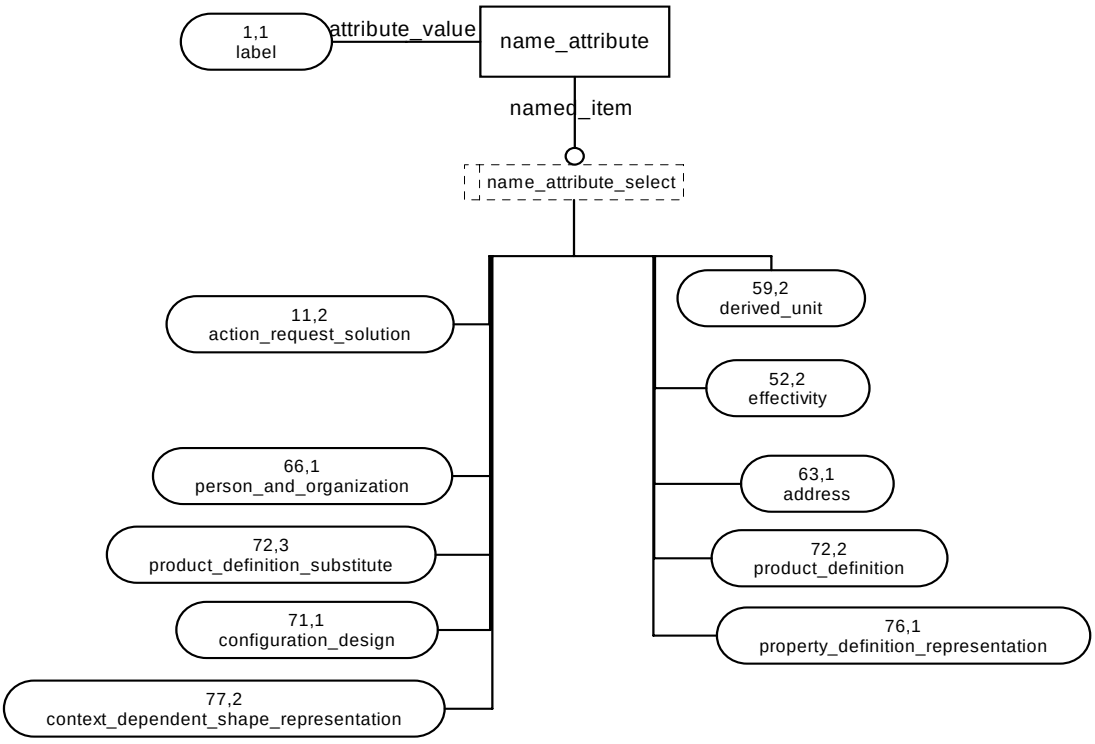


Figure H.7 — AIM EXPRESS-G diagram: name_attribute (7 of 91)

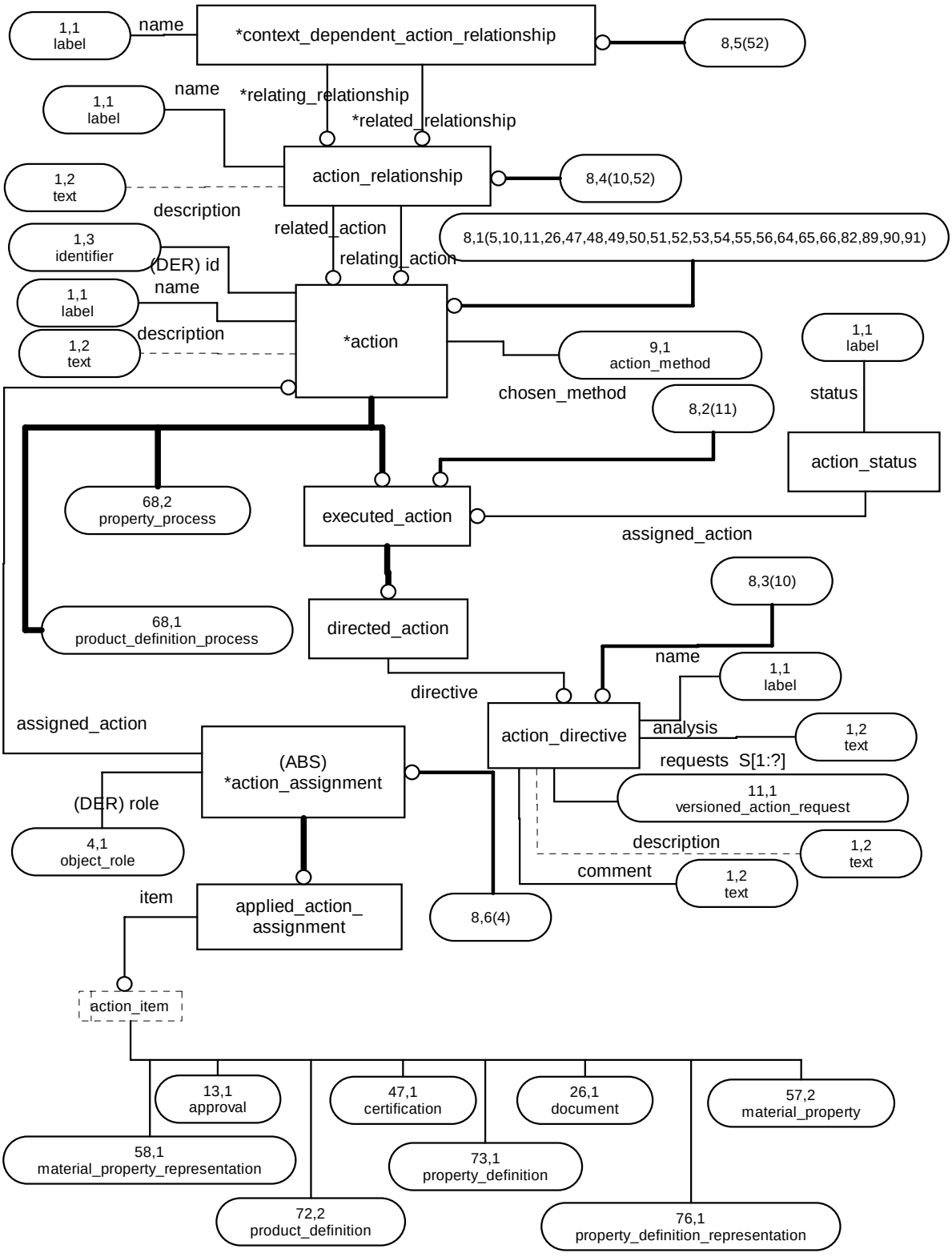


Figure H.8 — AIM EXPRESS-G diagram: action (8 of 91)

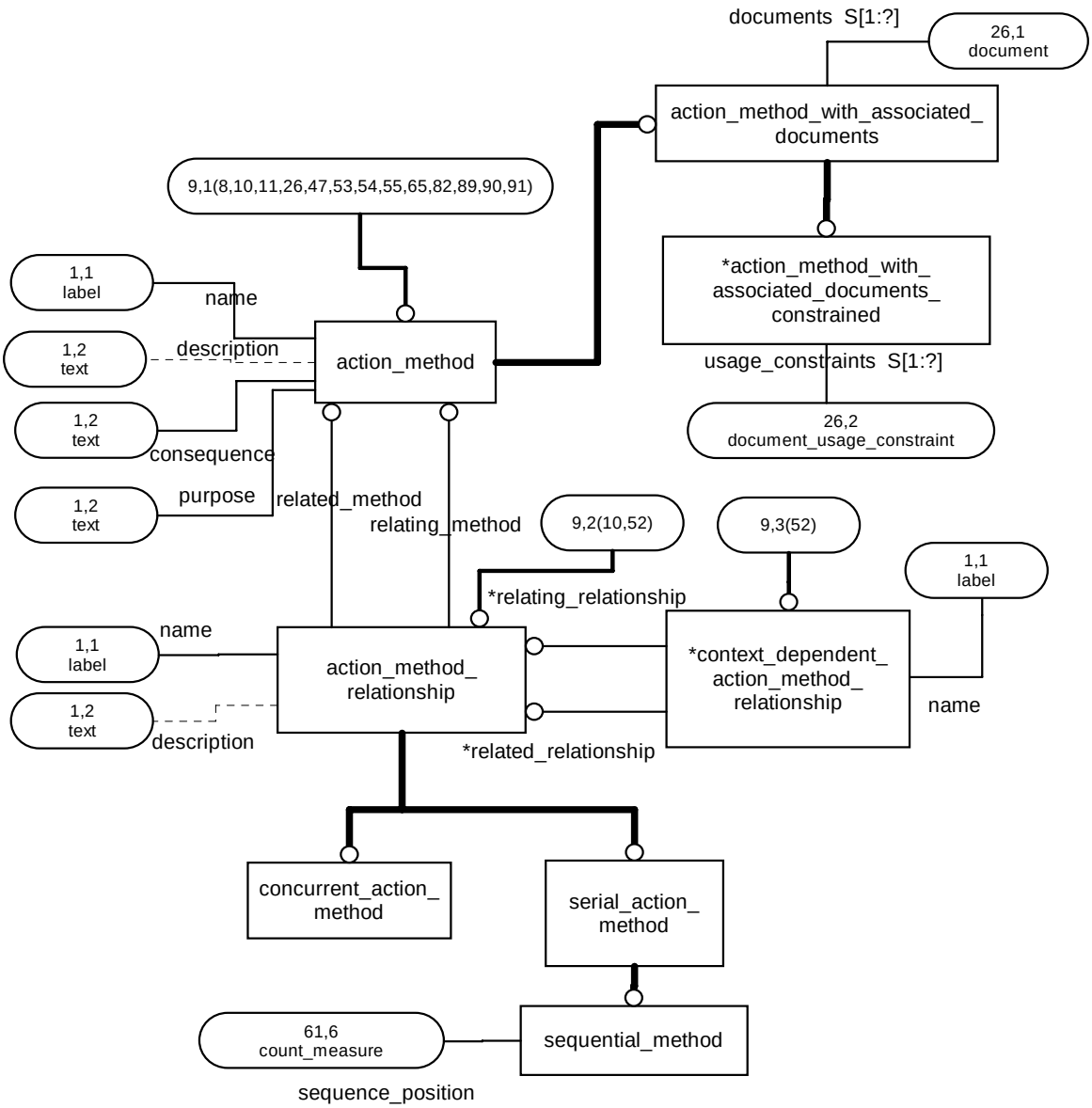


Figure H.9 — AIM EXPRESS-G diagram: action_method (9 of 91)

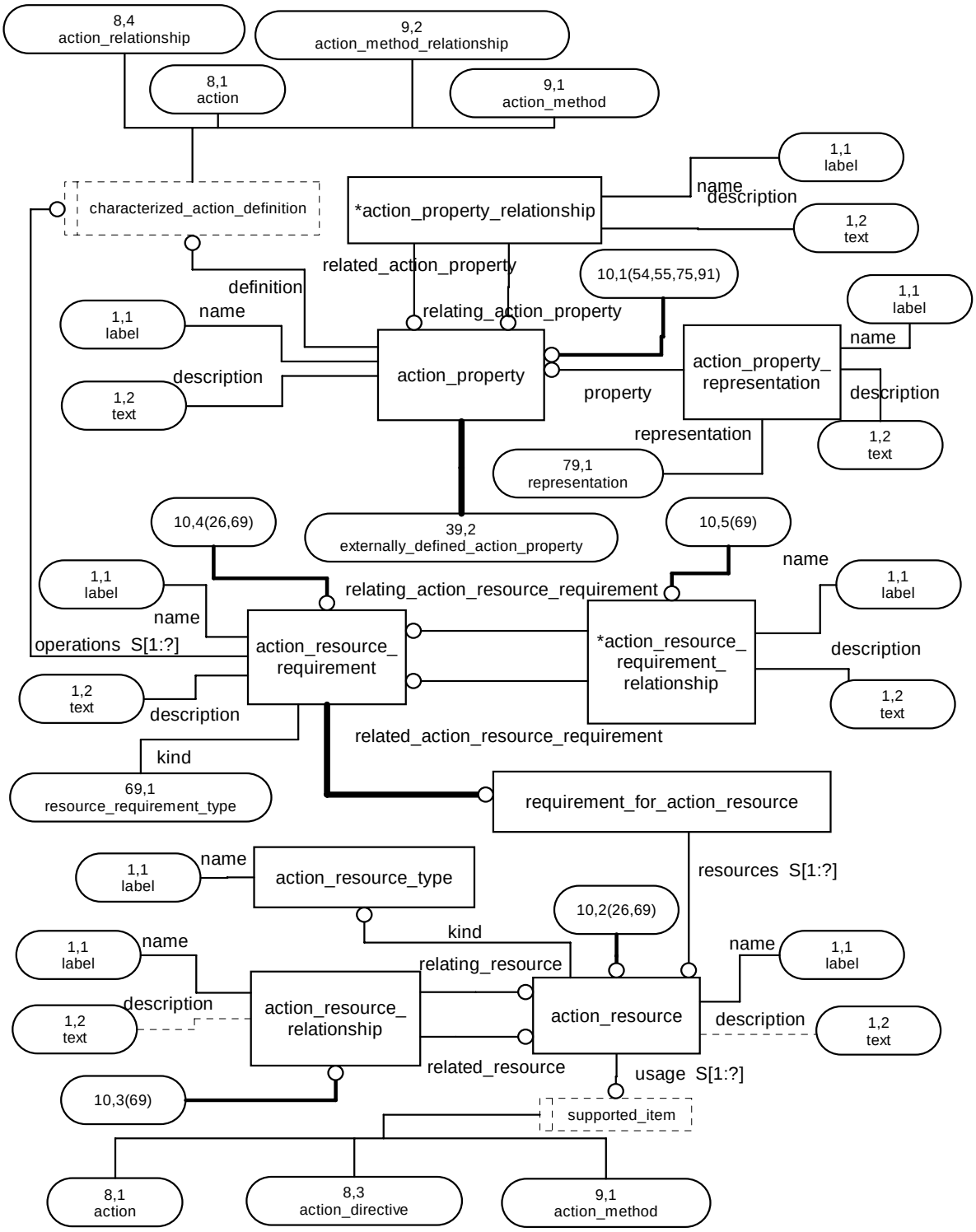


Figure H.10 — AIM EXPRESS-G diagram: action_property and action_resource (10 of 91)

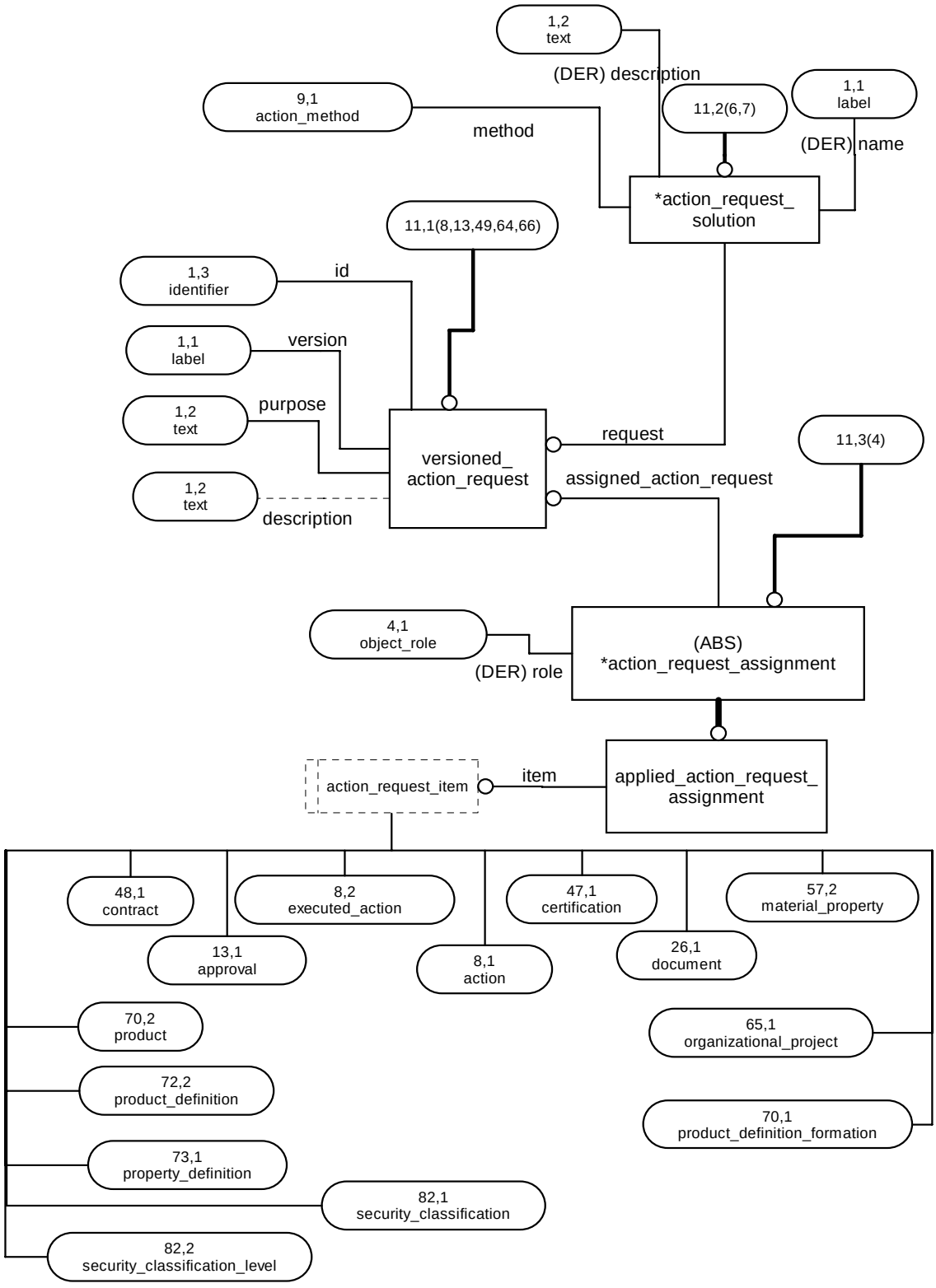


Figure H.11 — AIM EXPRESS-G diagram: action_request (11 of 91)

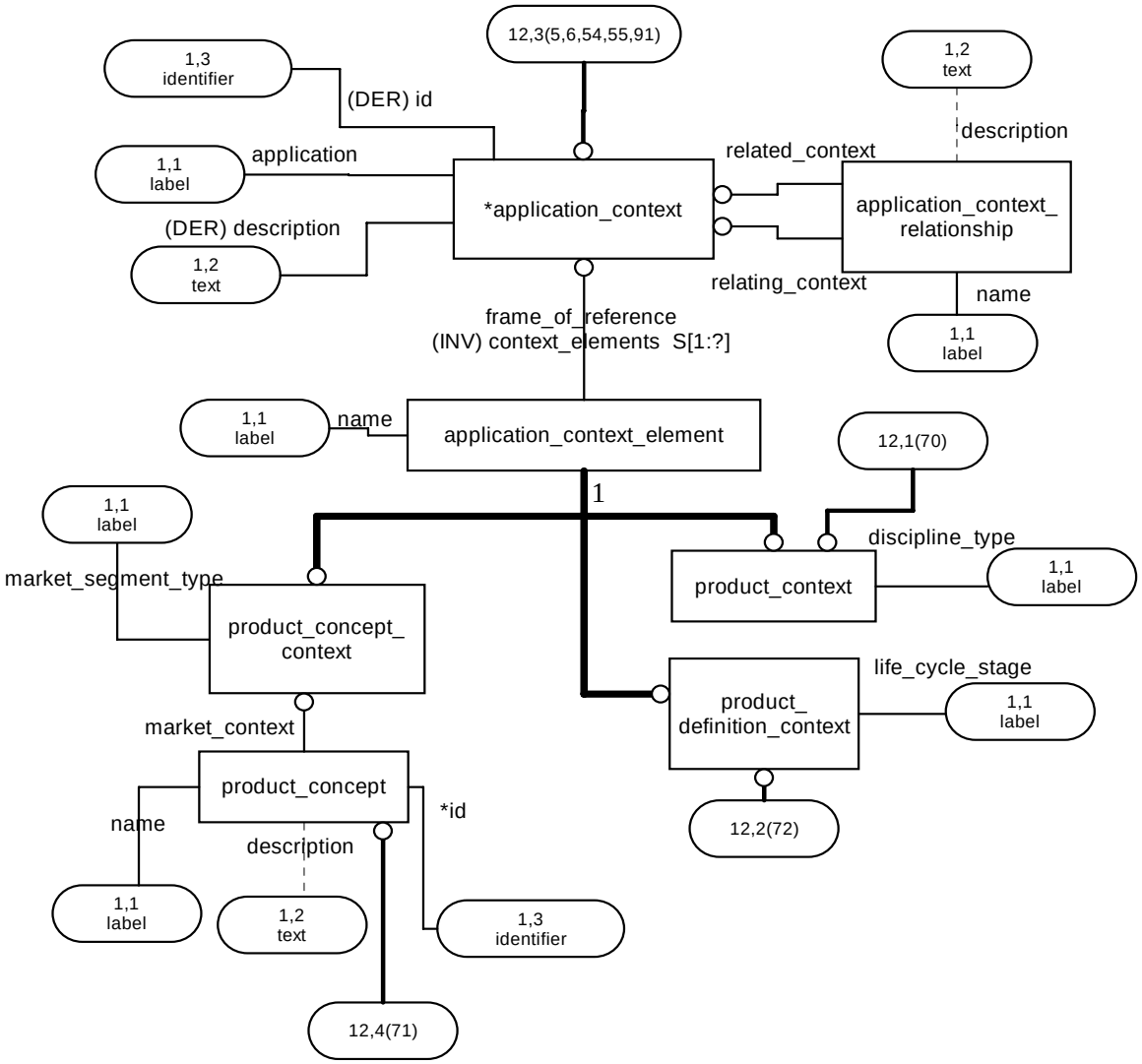


Figure H.12 — AIM EXPRESS-G diagram: application_context (12 of 91)

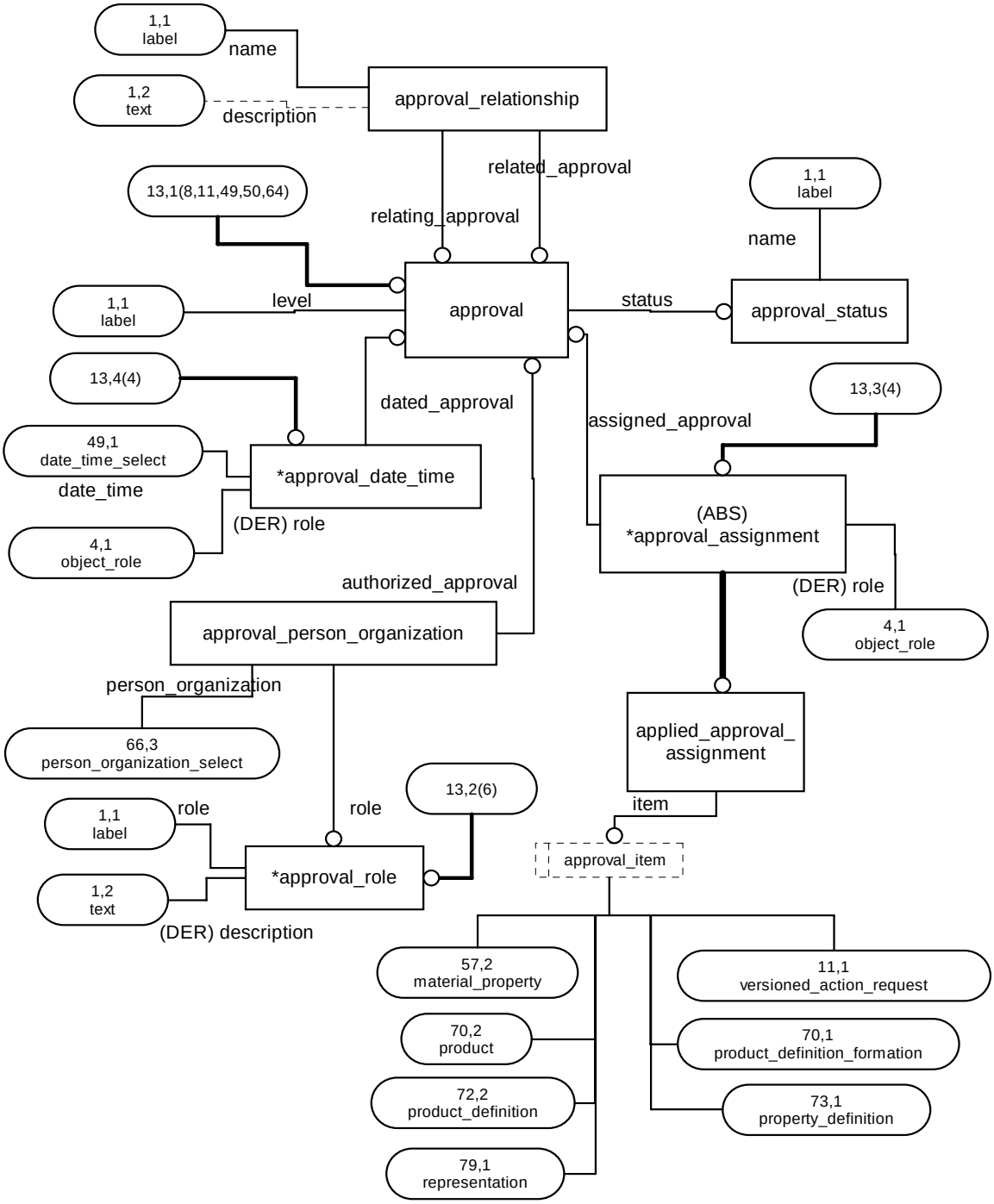


Figure H.13 — AIM EXPRESS-G diagram: approval (13 of 91)

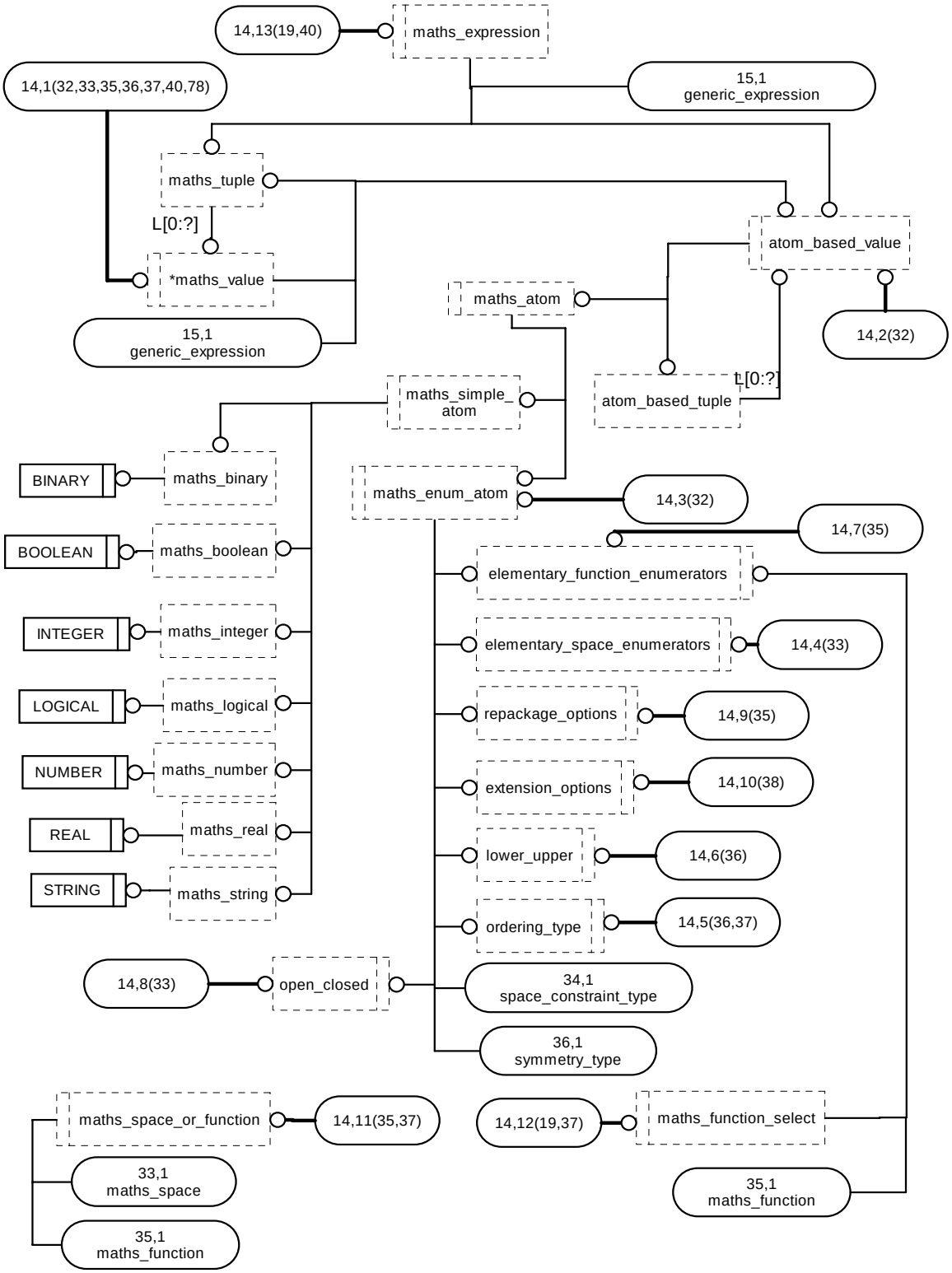


Figure H.14 — AIM EXPRESS-G diagram: math_types (14 of 91)

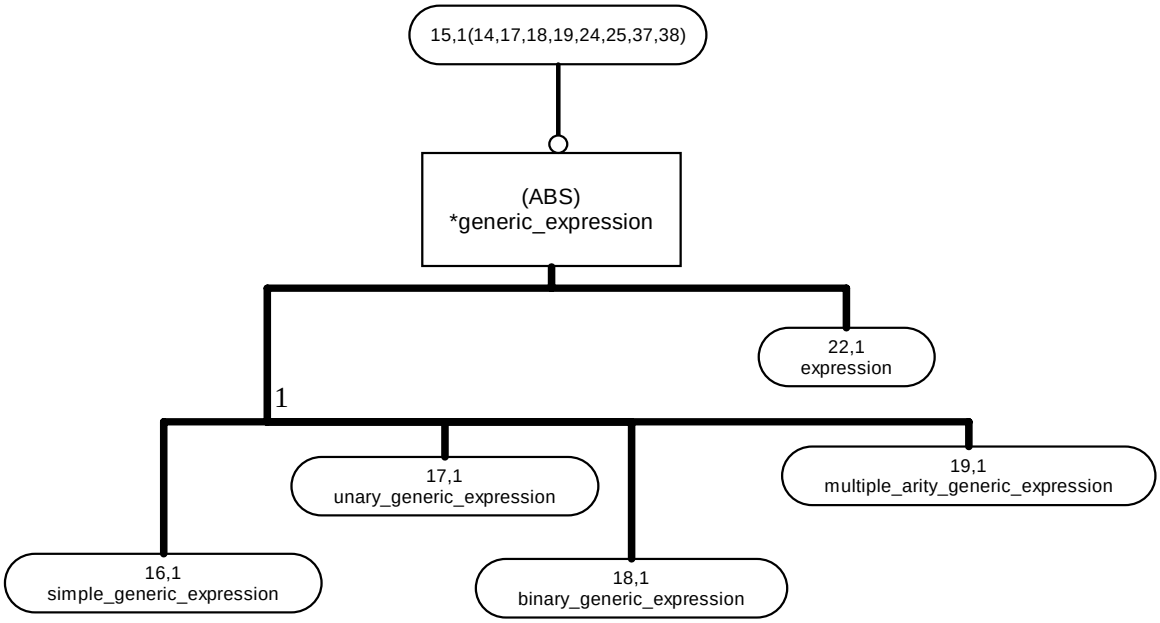


Figure H.15 — AIM EXPRESS-G diagram: generic_expression (15 of 91)

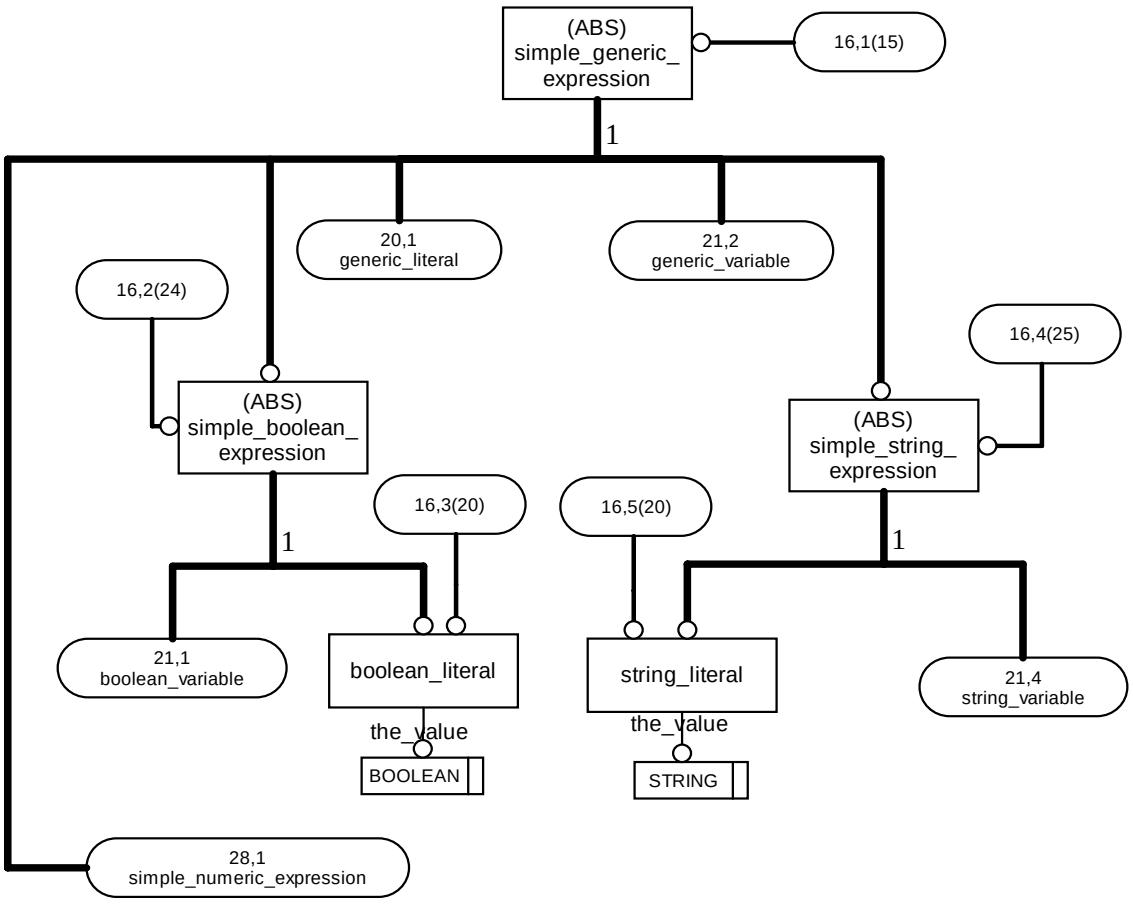


Figure H.16 — AIM EXPRESS-G diagram: simple_generic_expression (16 of 91)

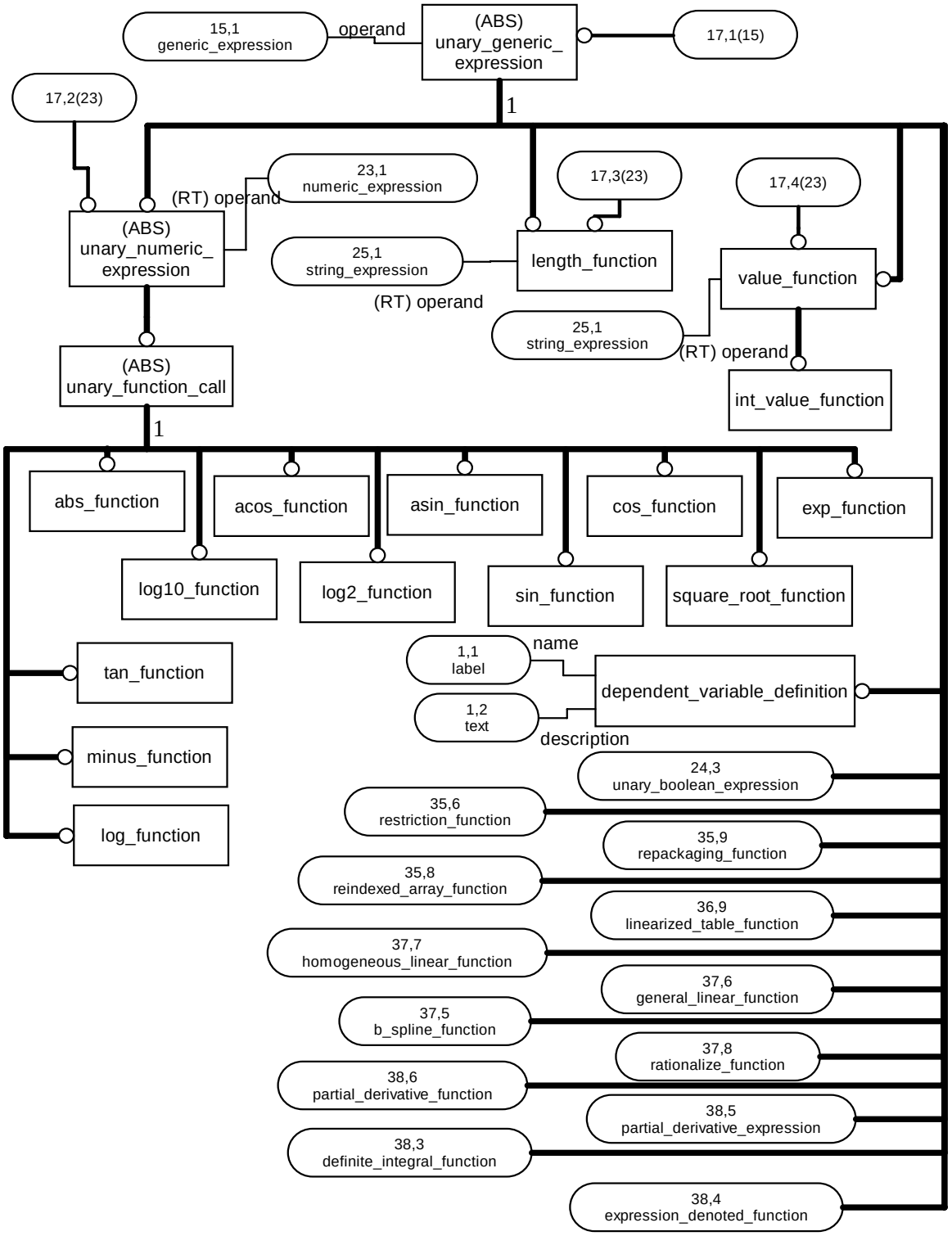


Figure H.17 — AIM EXPRESS-G diagram: unary_generic_expression (17 of 91)

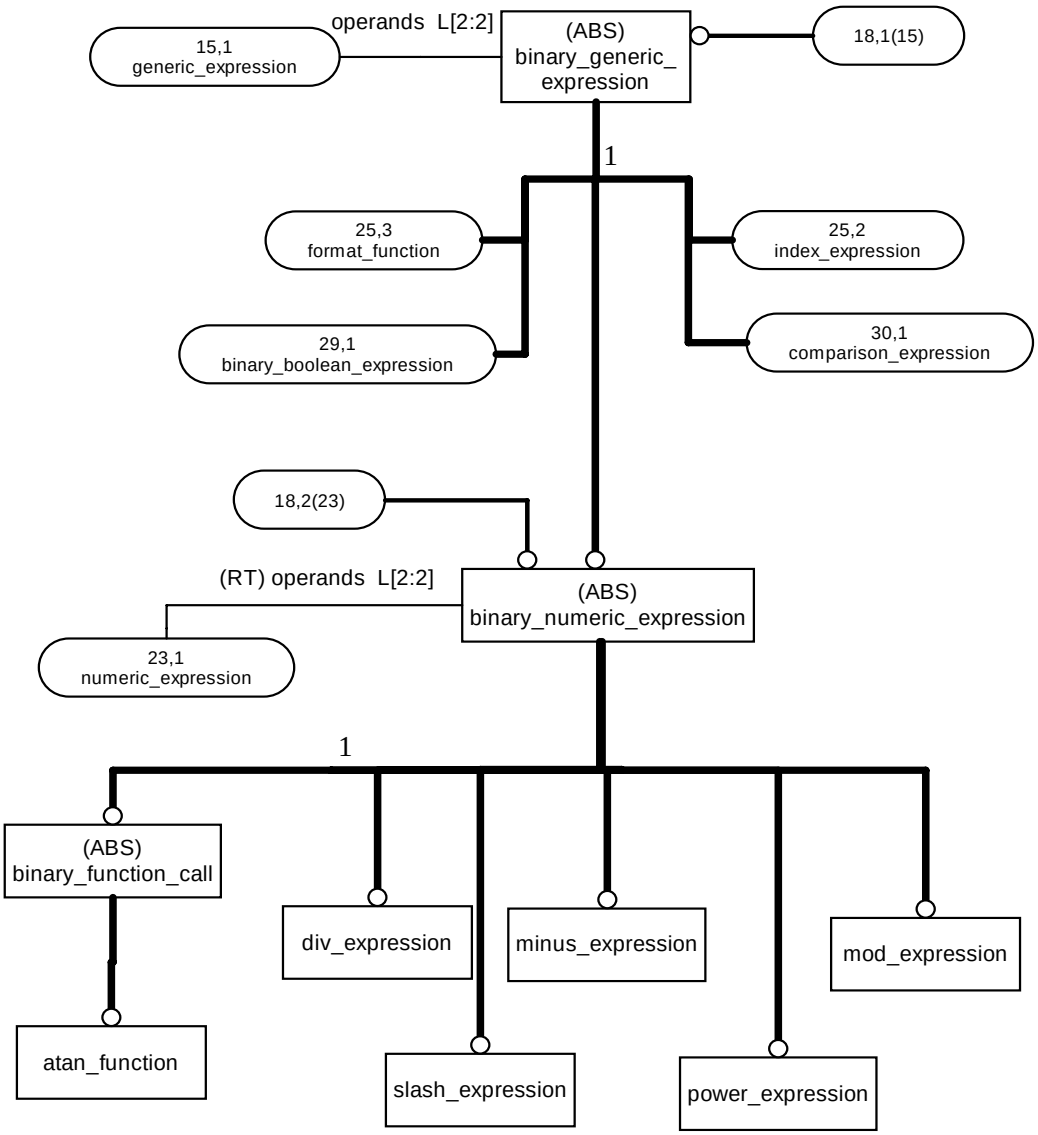


Figure H.18 — AIM EXPRESS-G diagram: binary_generic_expression (18 of 91)

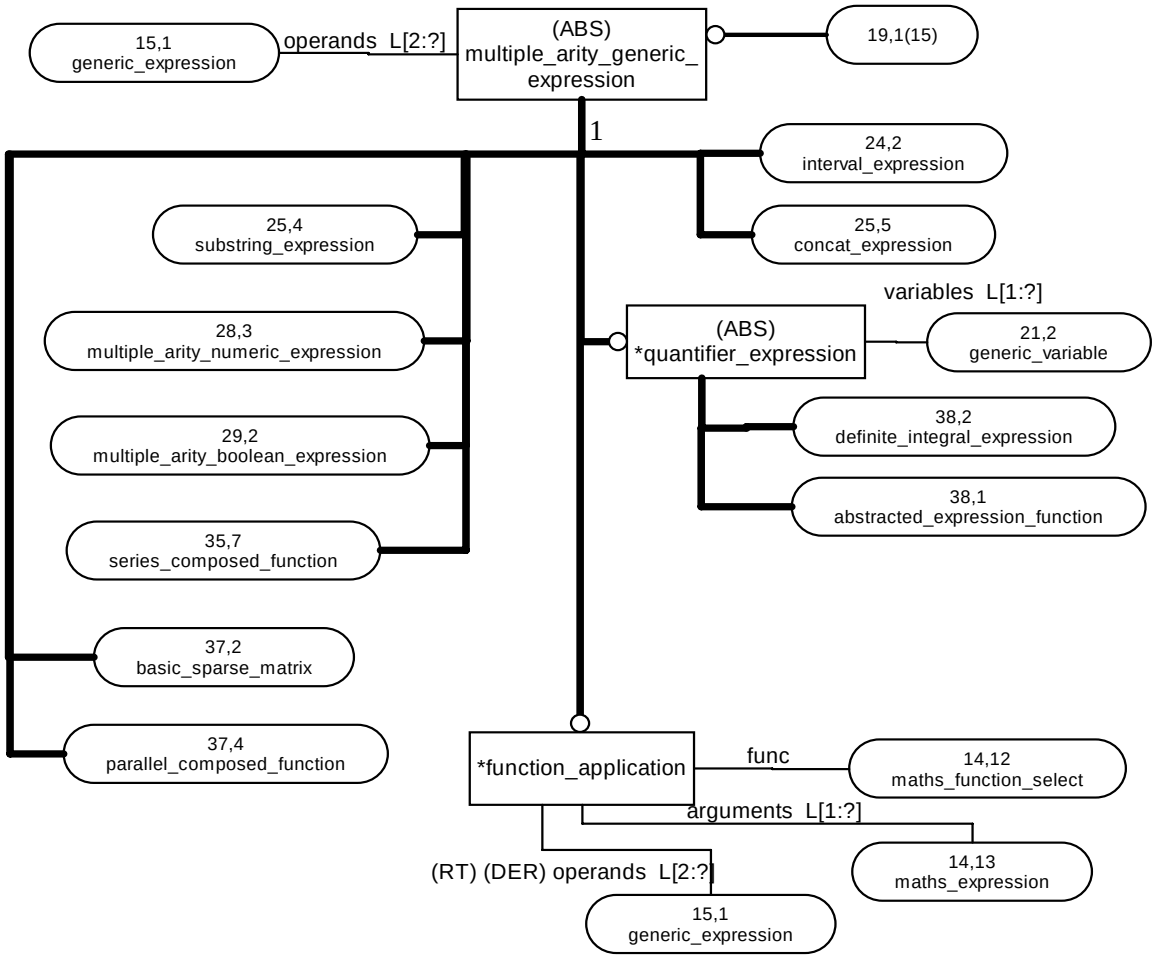


Figure H.19 — AIM EXPRESS-G diagram: multiple_arity_generic_expression (19 of 91)

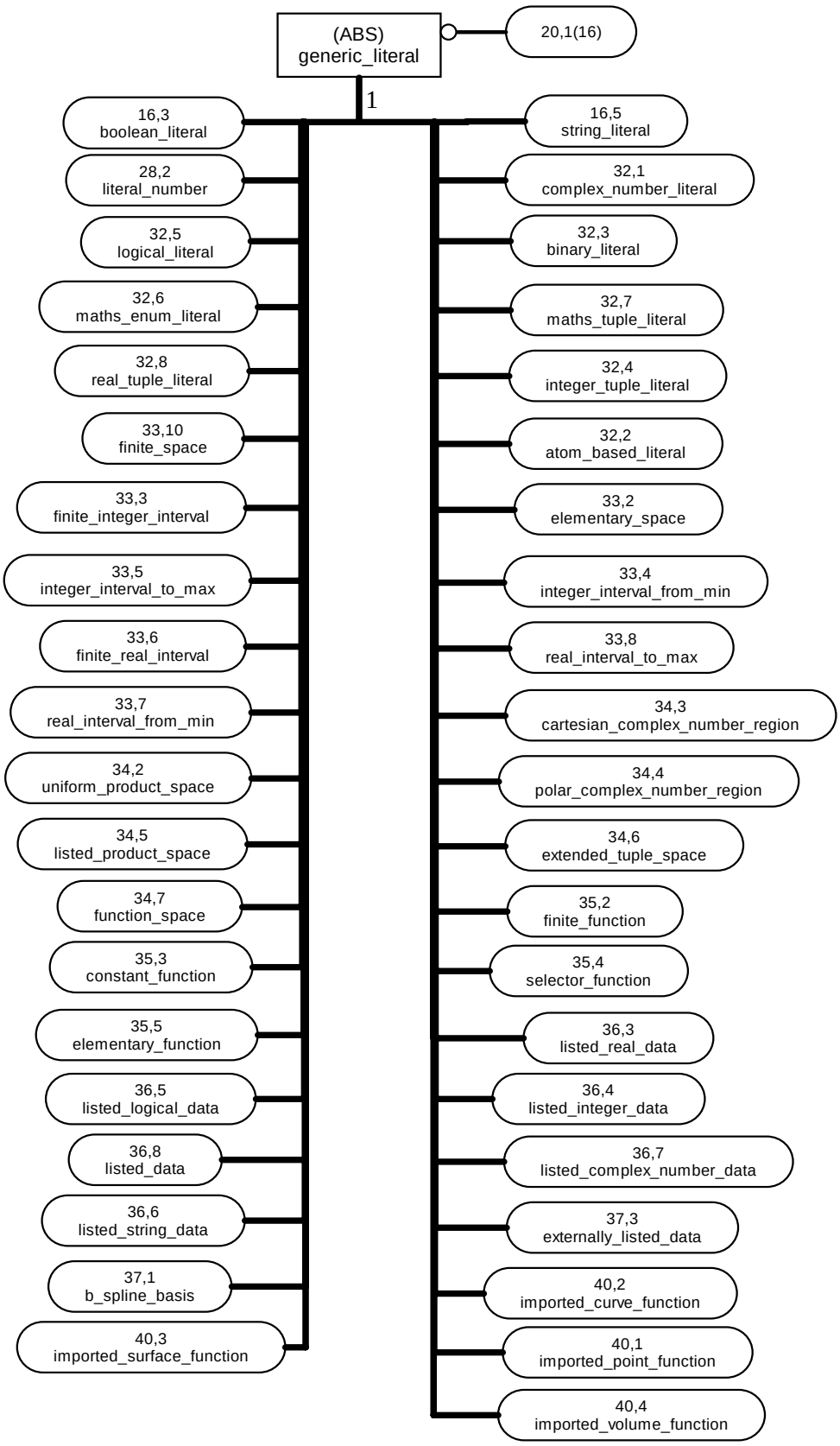


Figure H.20 — AIM EXPRESS-G diagram: generic_literal_expression (20 of 91)

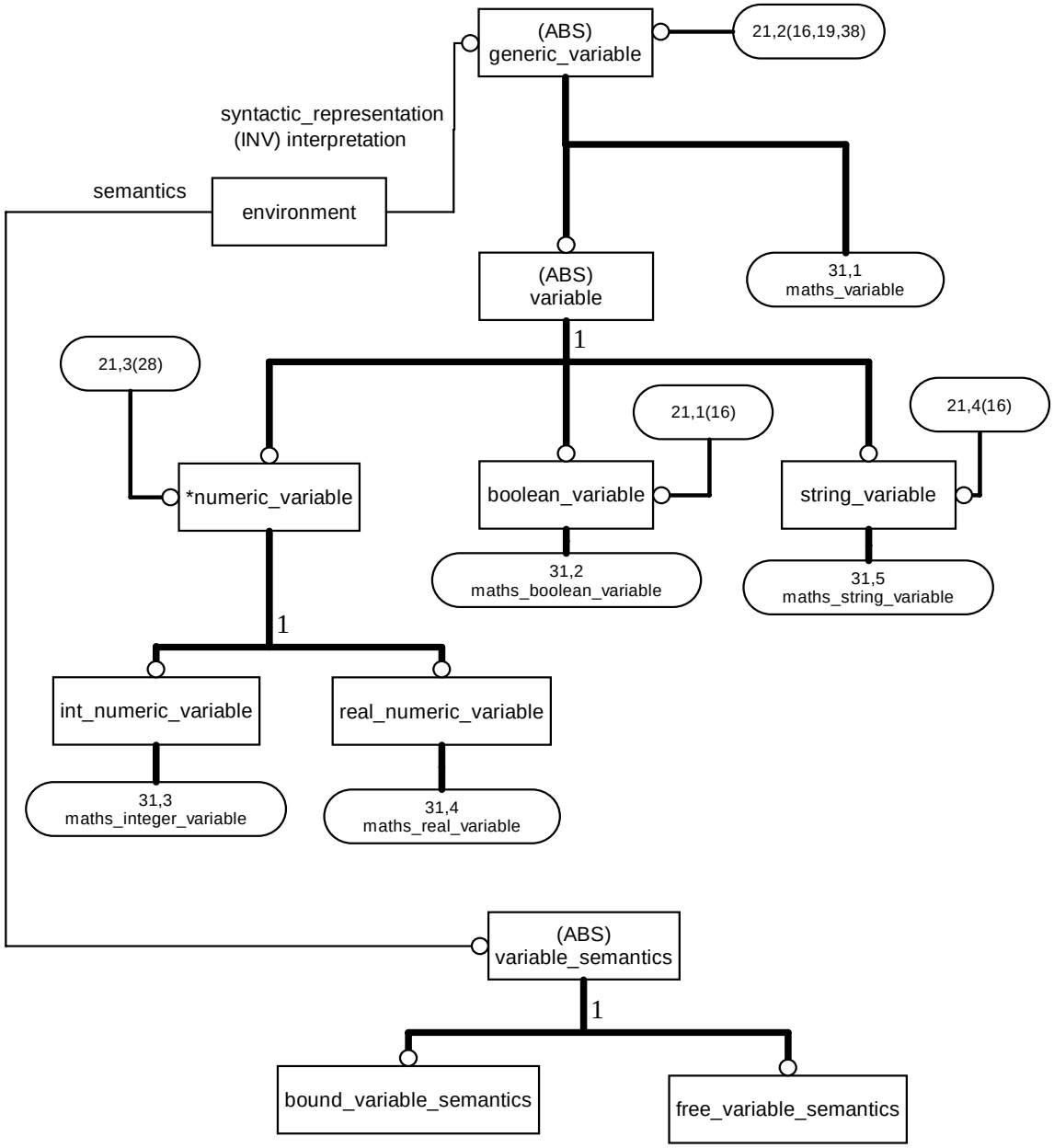


Figure H.21 — AIM EXPRESS-G diagram: generic_variable (21 of 91)

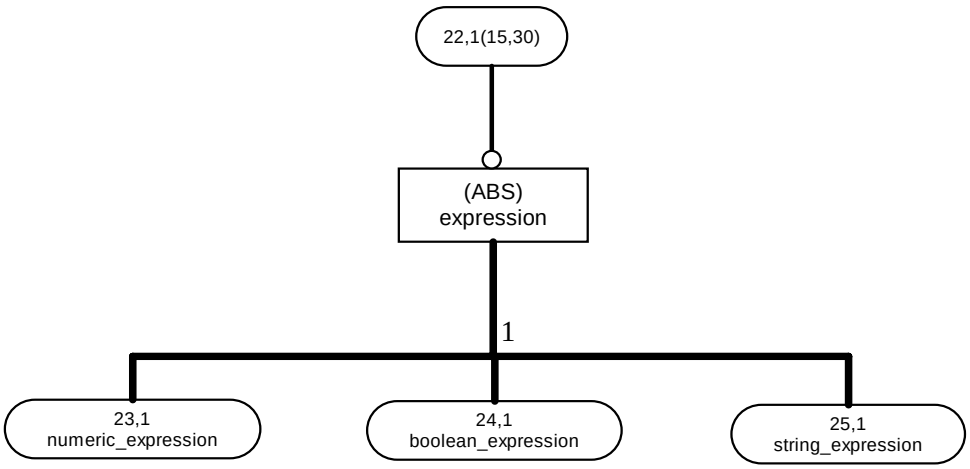


Figure H.22 — AIM EXPRESS-G diagram: expression (22 of 91)

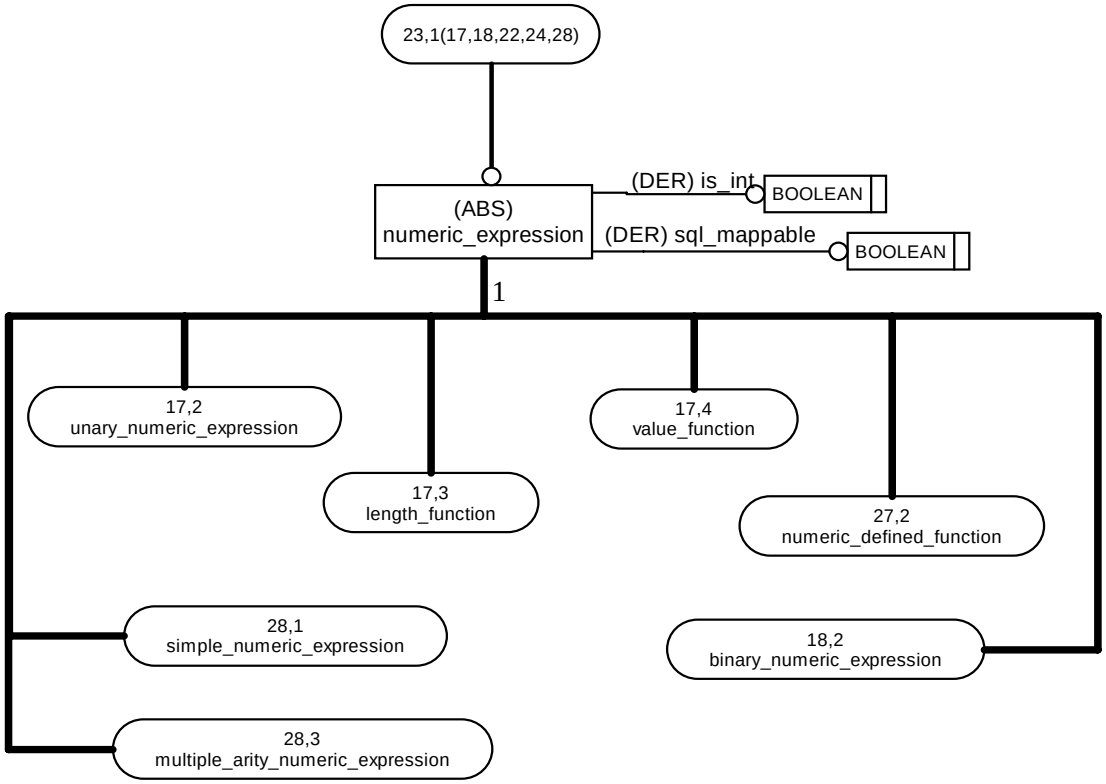


Figure H.23 — AIM EXPRESS-G diagram: numeric_expression (23 of 91)

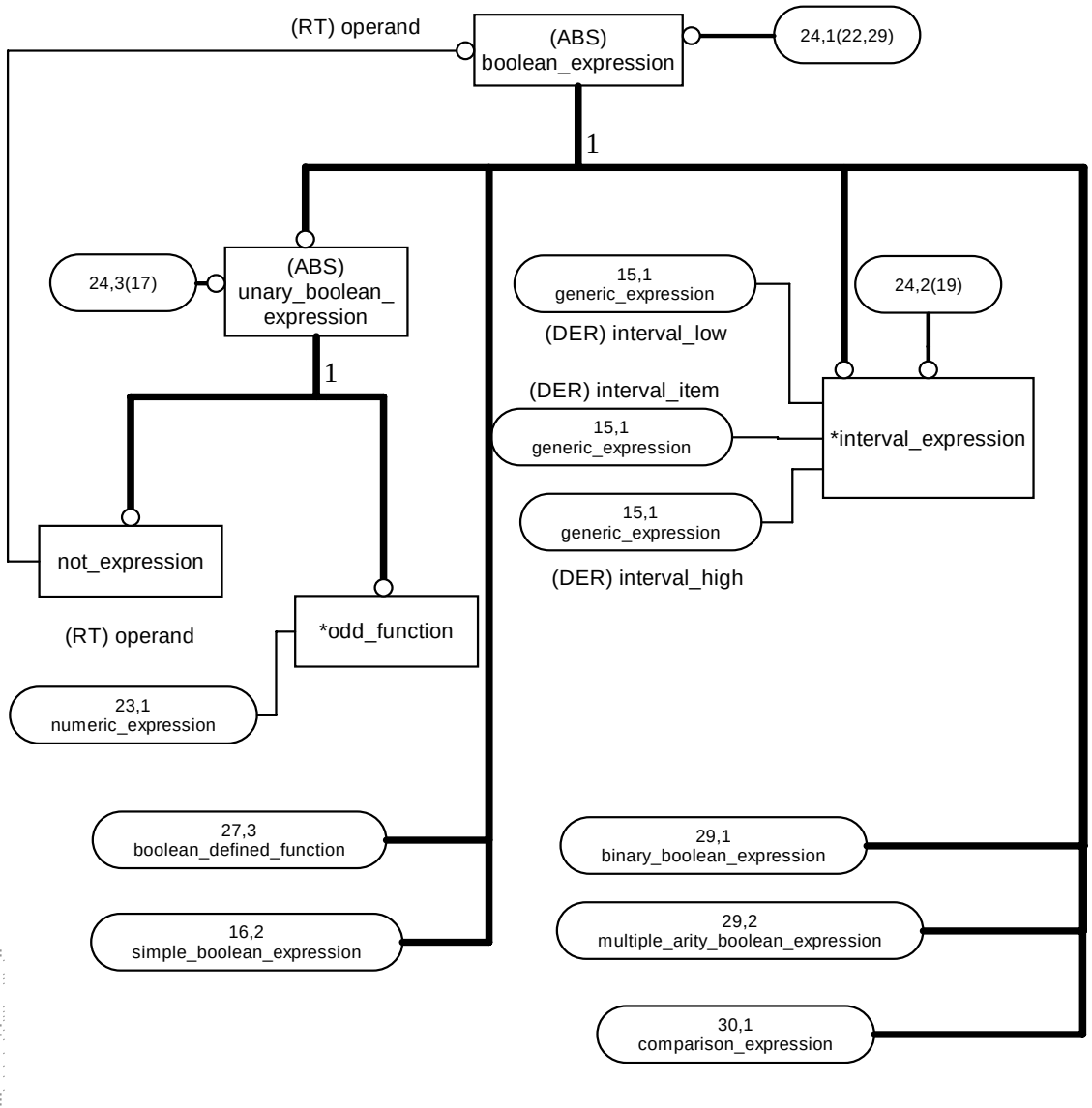


Figure H.24 — AIM EXPRESS-G diagram: boolean_expression (24 of 91)

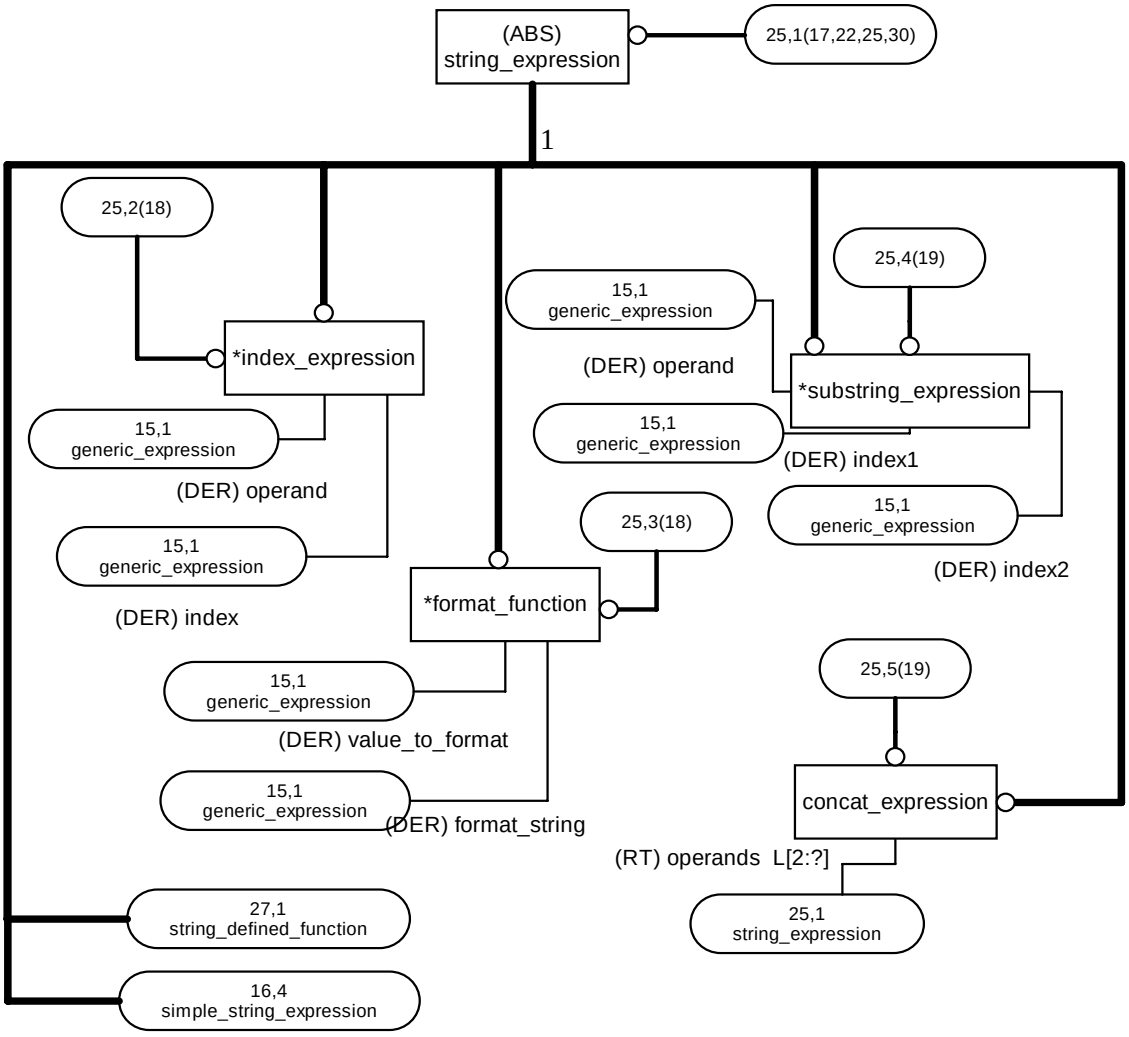


Figure H.25 — AIM EXPRESS-G diagram: string_expression (25 of 91)

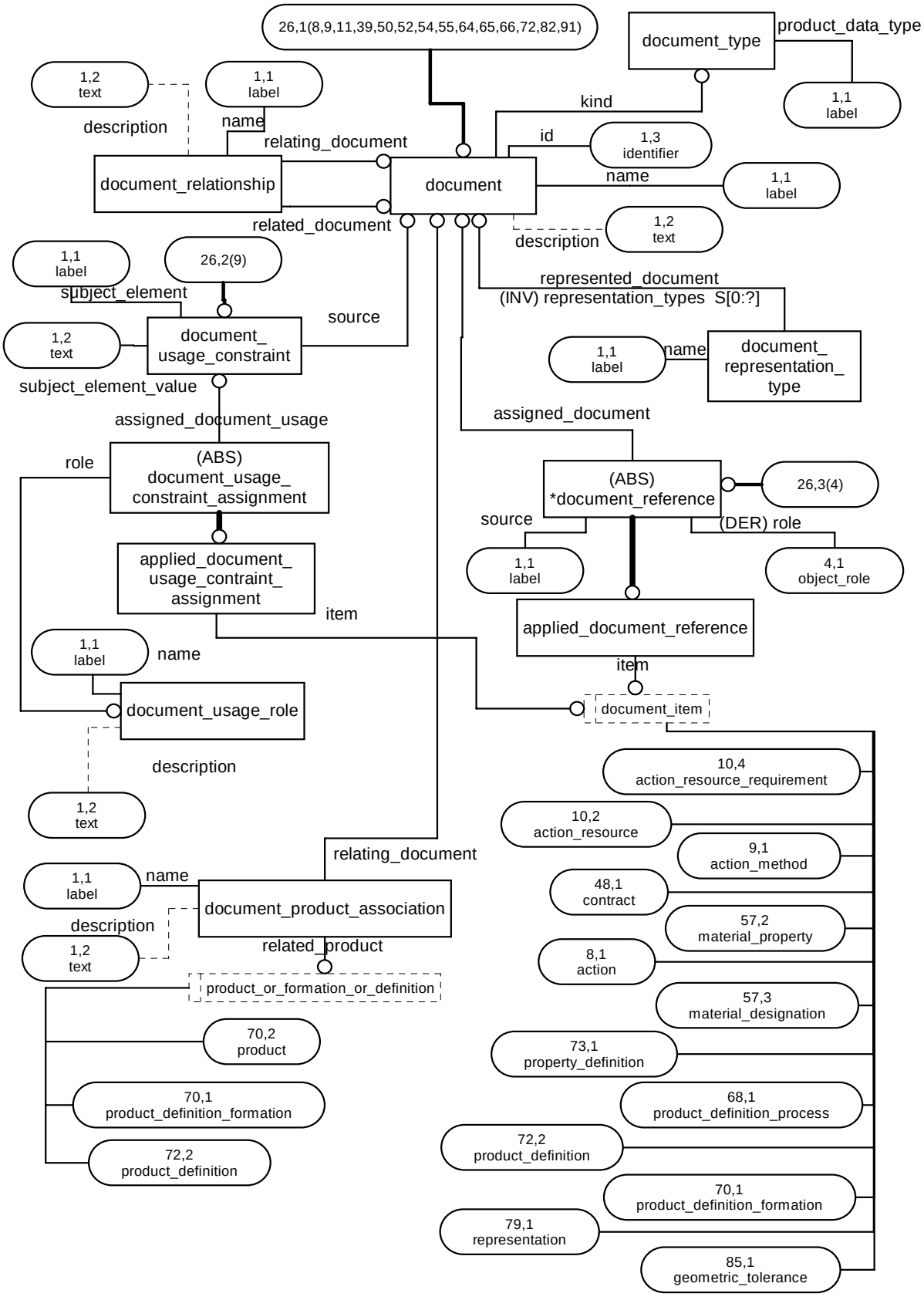


Figure H.26 — AIM EXPRESS-G diagram: document (26 of 91)

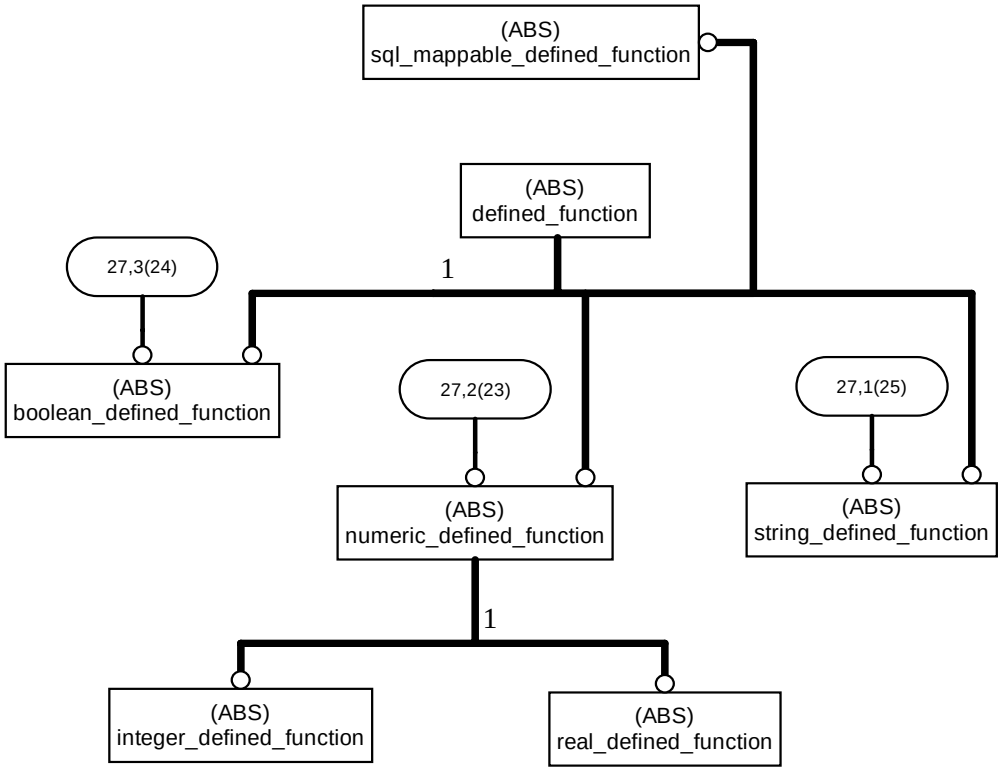


Figure H.27 — AIM EXPRESS-G diagram: defined_function (27 of 91)

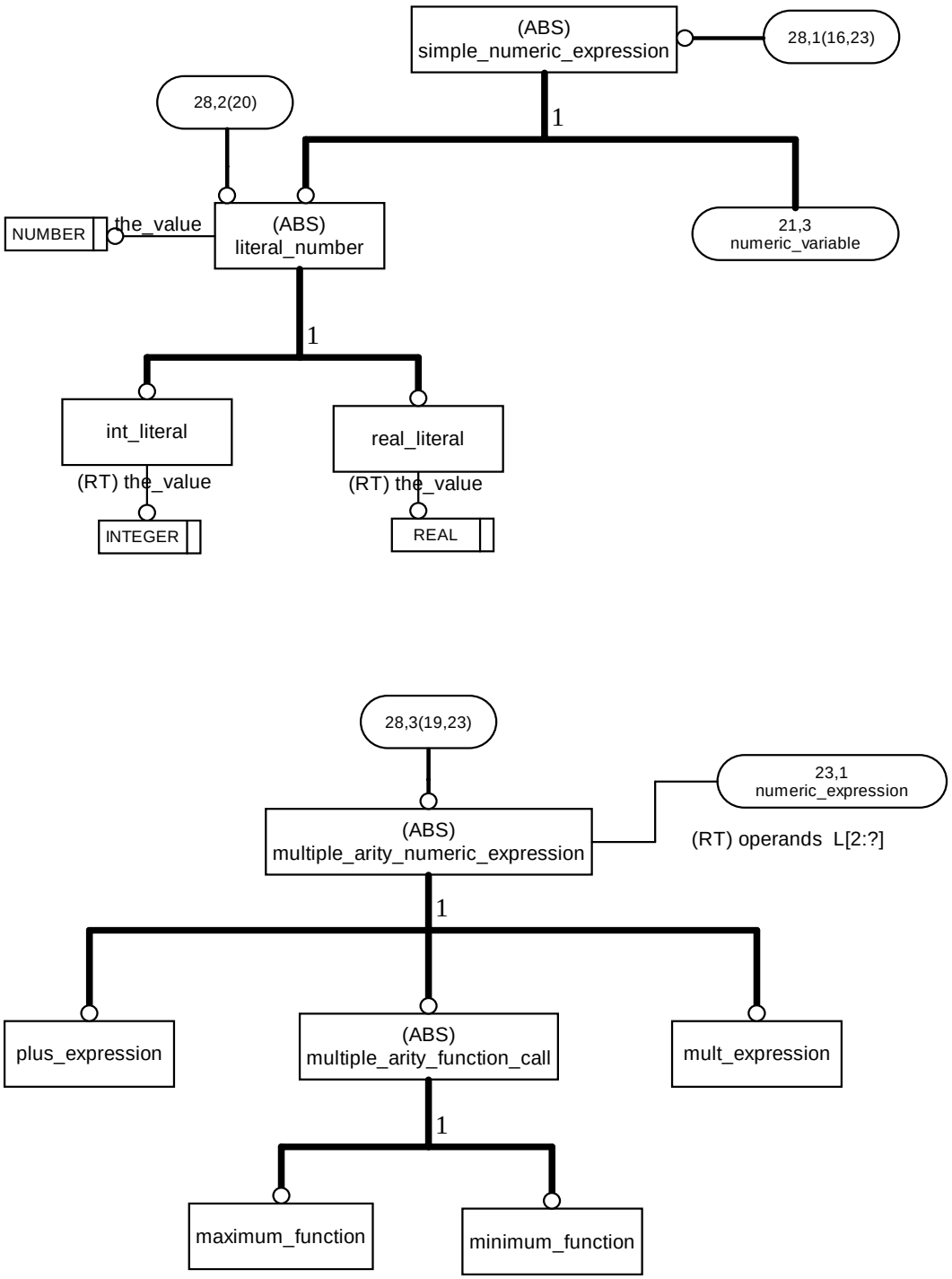


Figure H.28 — AIM EXPRESS-G diagram: numeric_expressions (28 of 91)

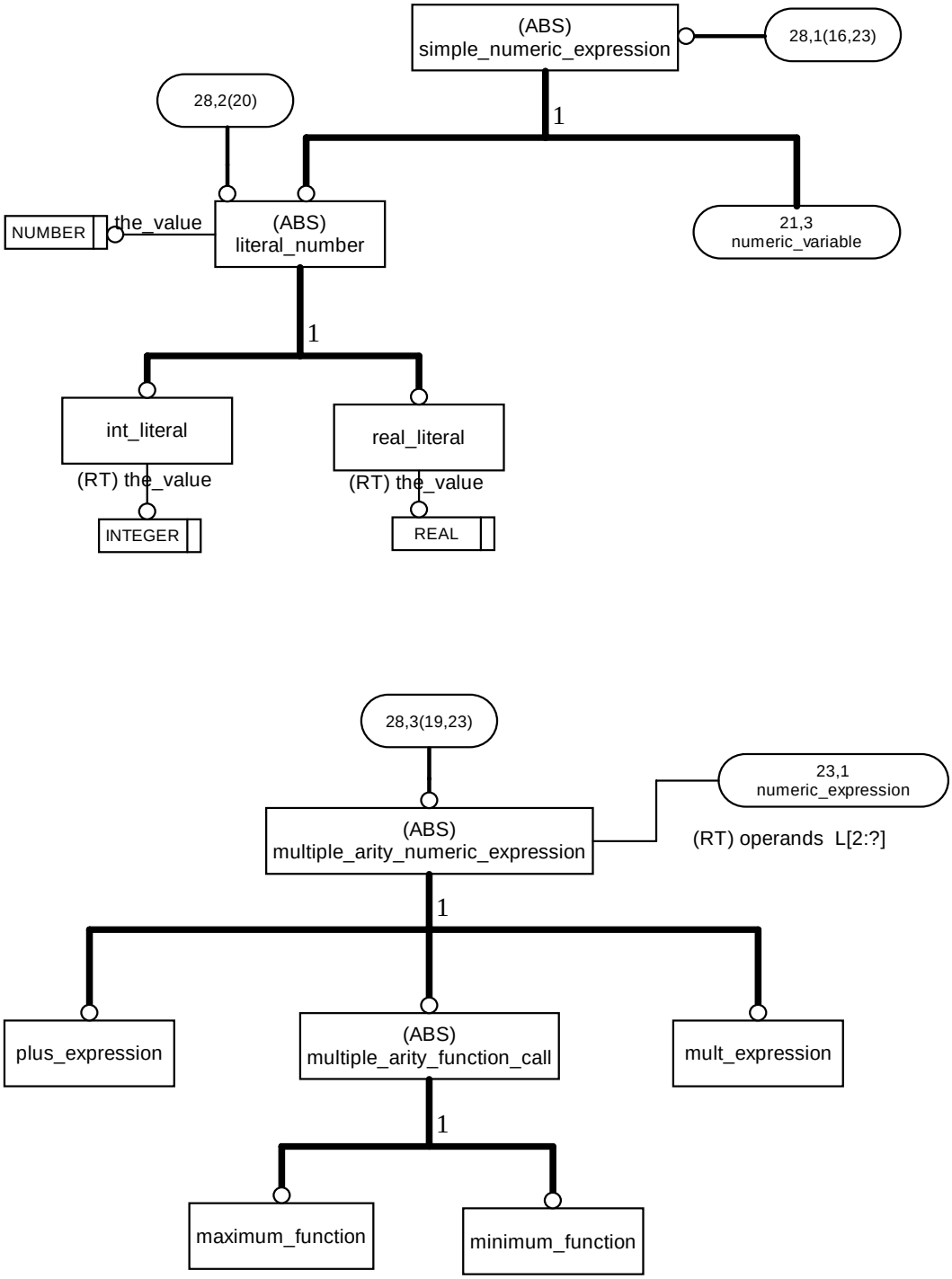


Figure H.29 — AIM EXPRESS-G diagram: boolean_expressions (29 of 91)

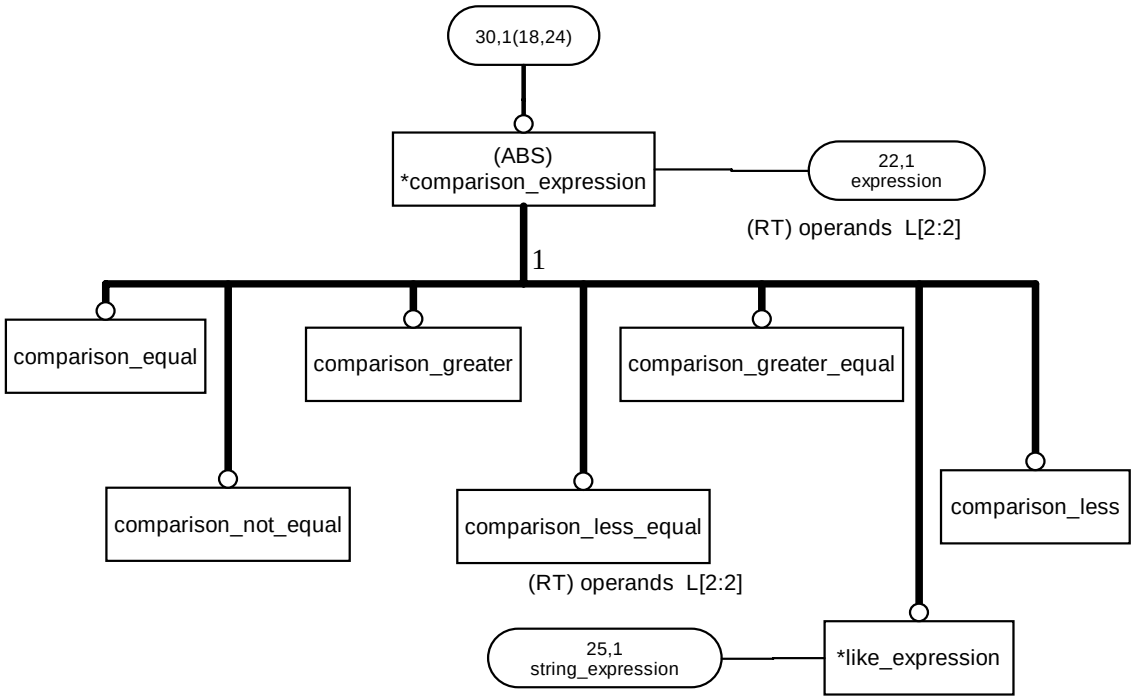


Figure H.30 — AIM EXPRESS-G diagram: comparison_expression

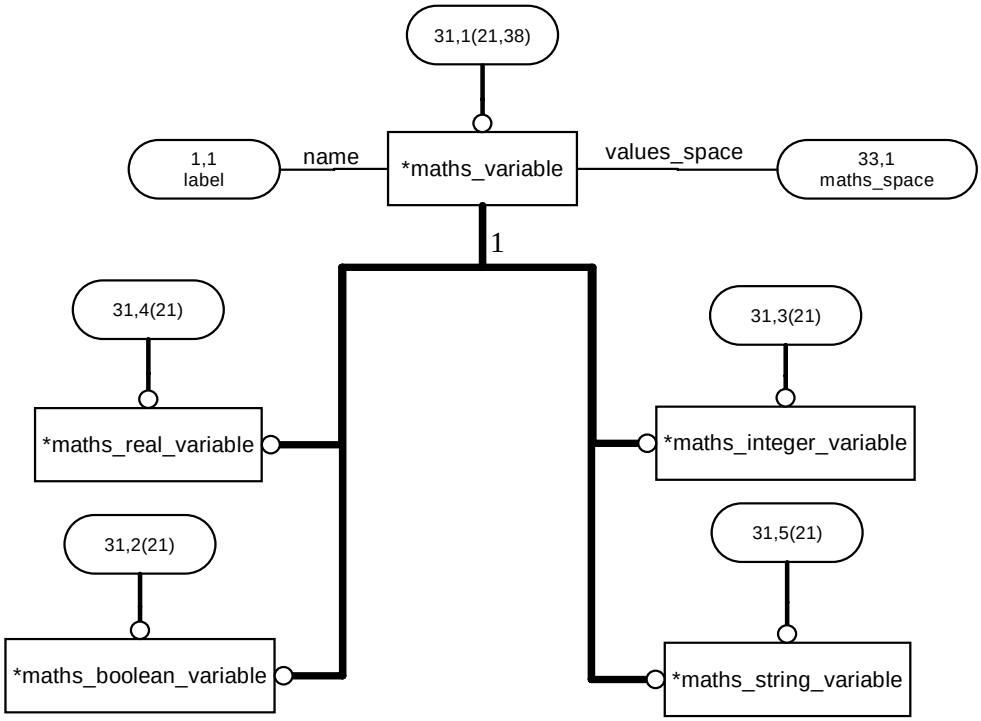


Figure H.31 — AIM EXPRESS-G diagram: maths_variable (31 of 91)

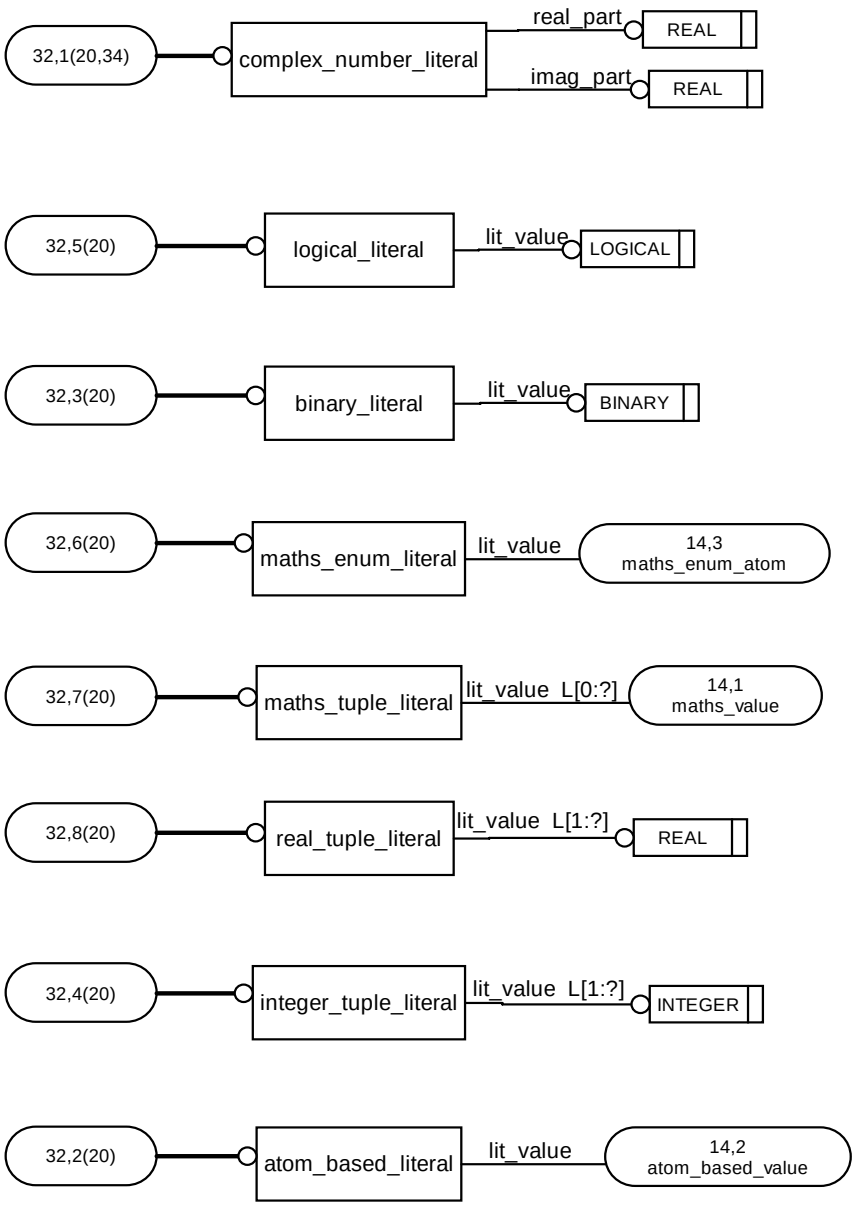


Figure H.32 — AIM EXPRESS-G diagram: maths_literal (32 of 91)

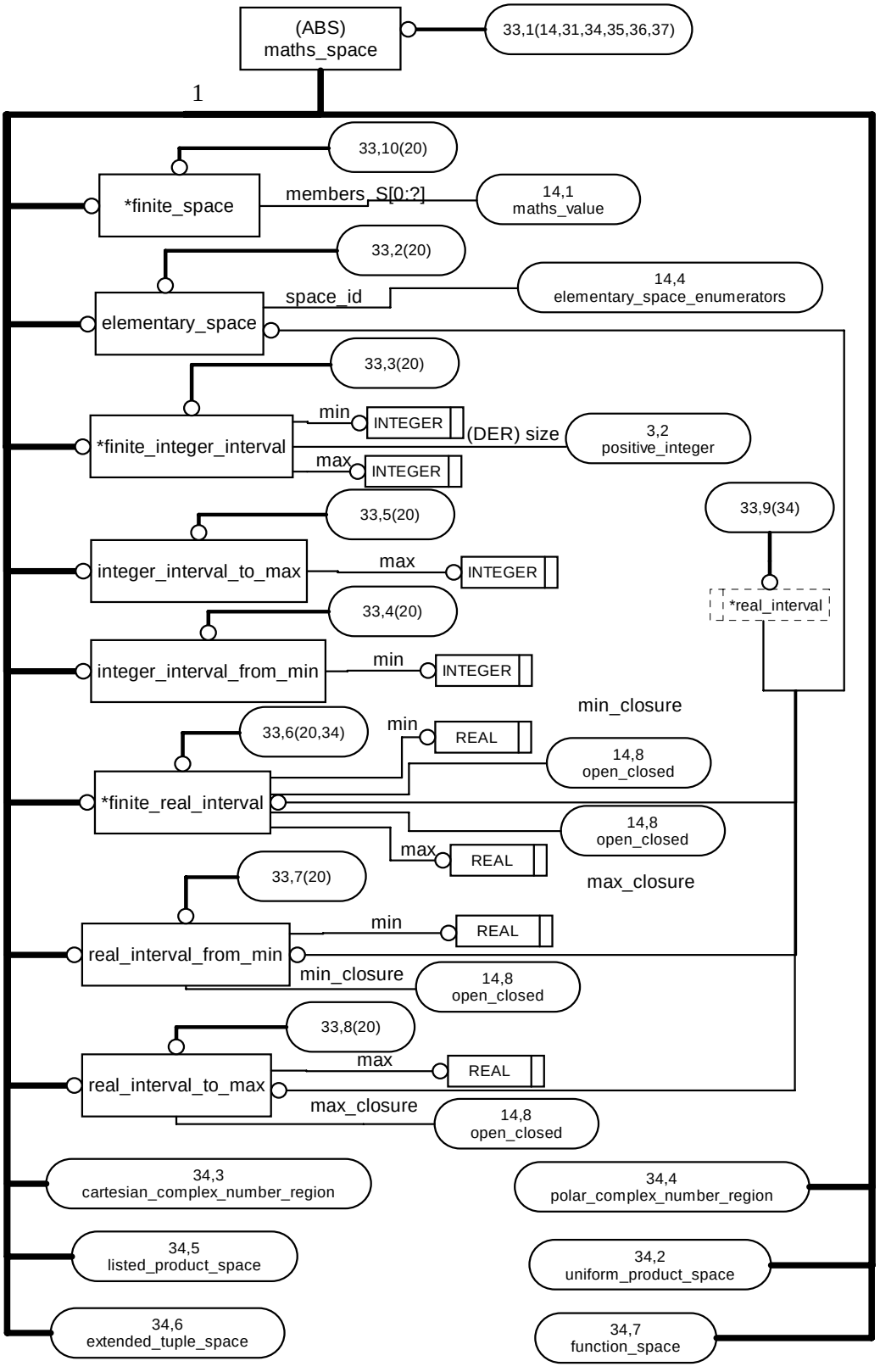


Figure H.33 — AIM EXPRESS-G diagram: maths_space (33 of 91)

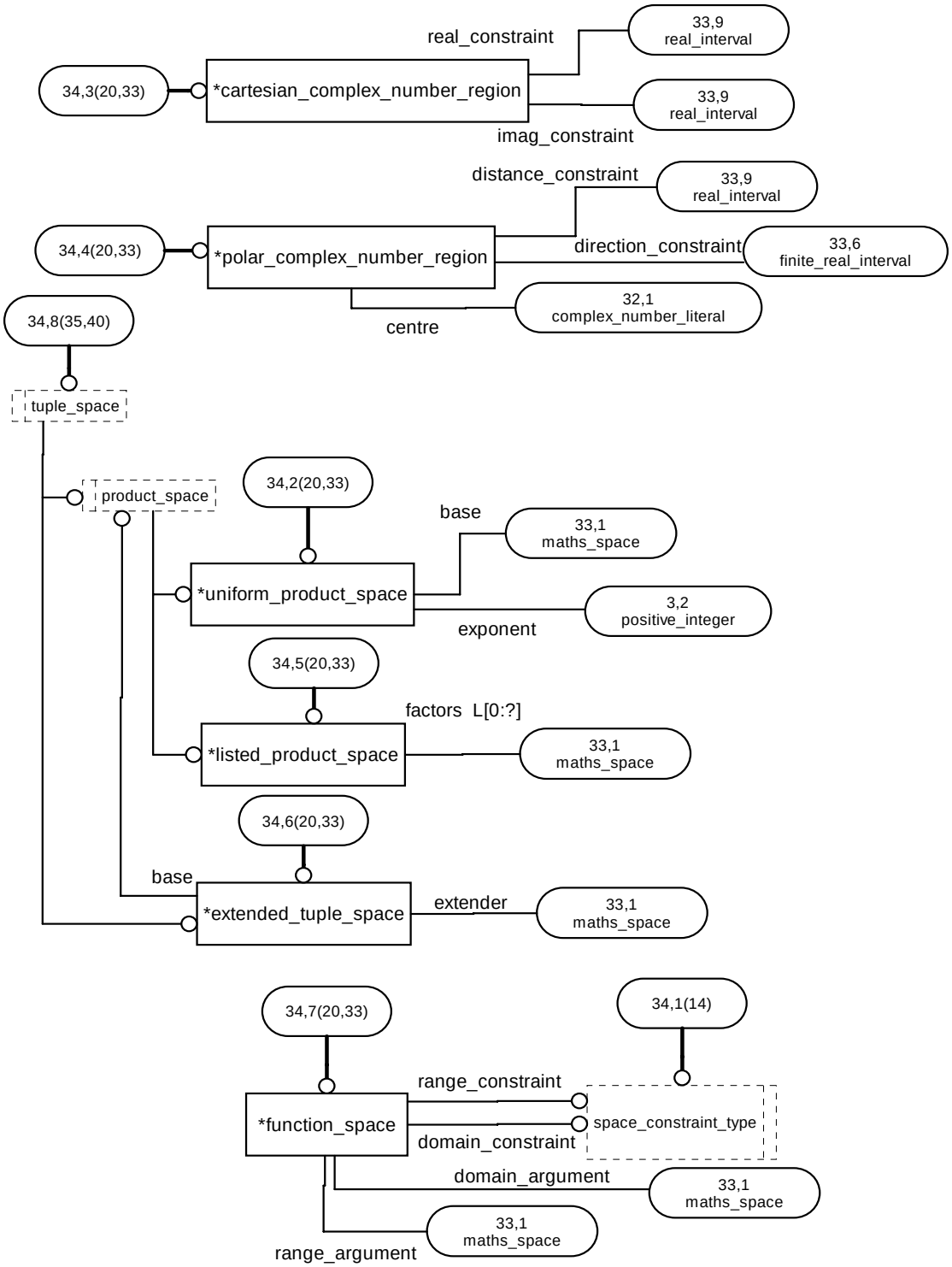


Figure H.34 — AIM EXPRESS-G diagram: maths_spaces (34 of 91)

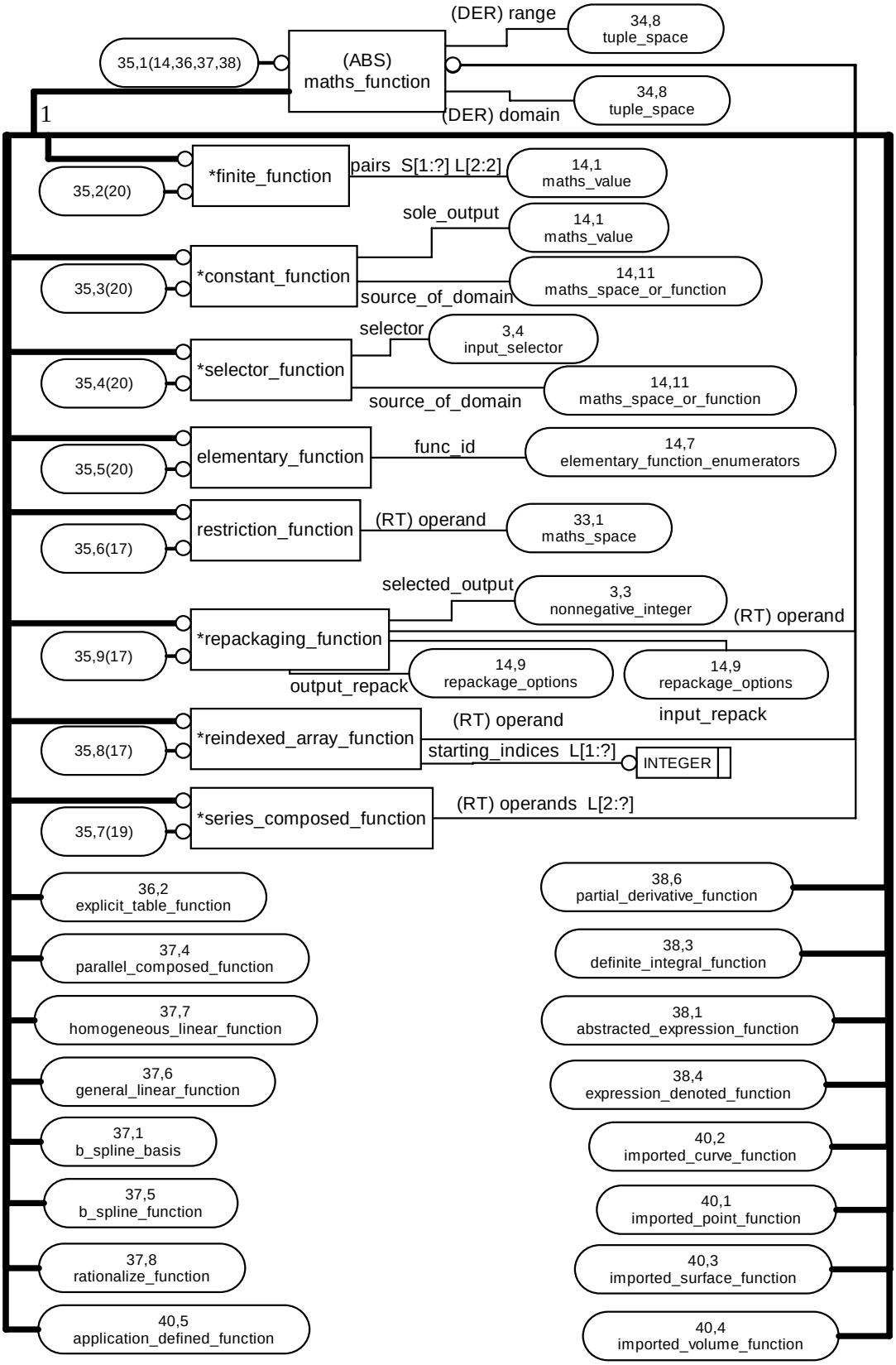


Figure H.35 — AIM EXPRESS-G diagram: maths_function (35 of 91)

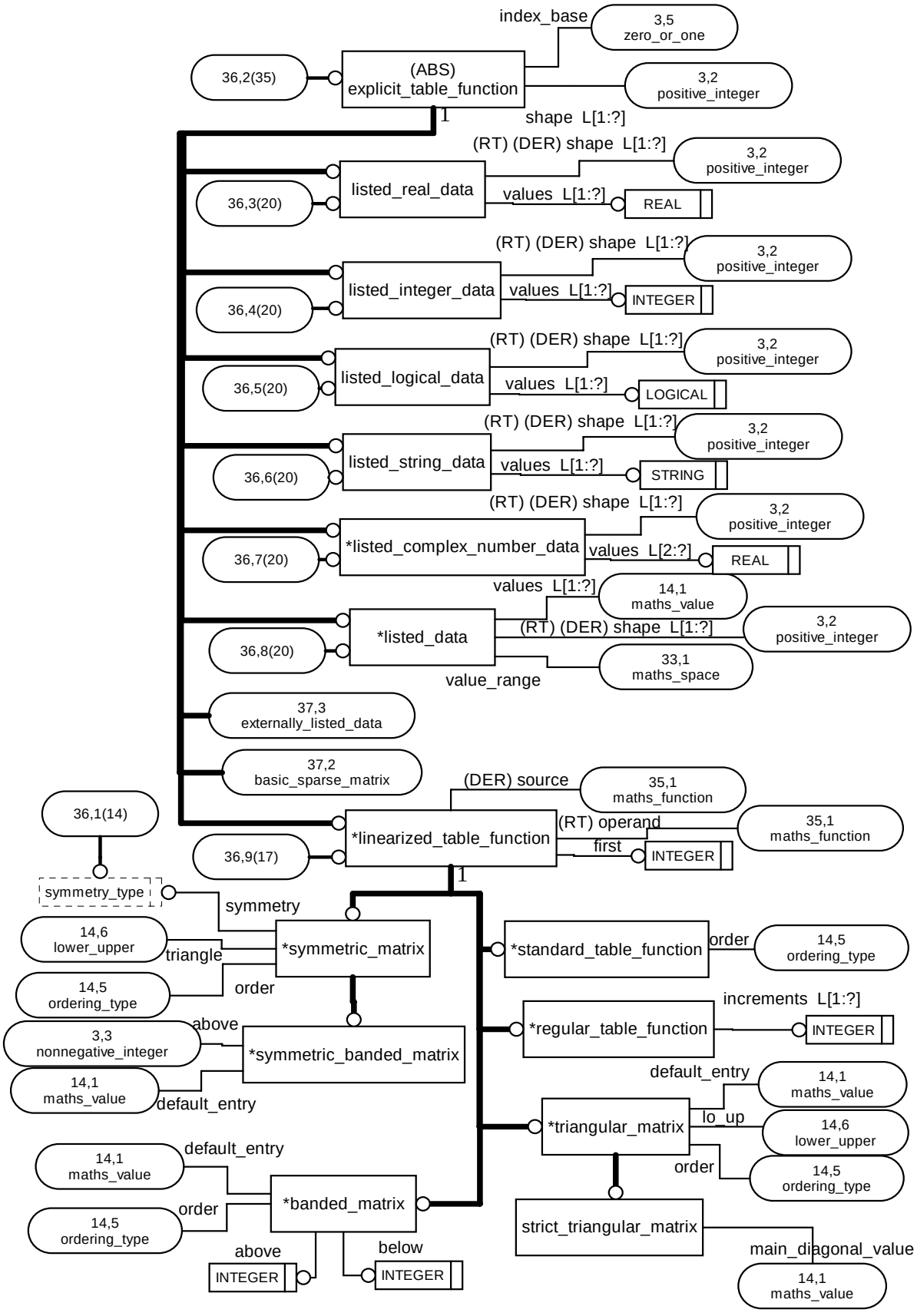


Figure H.36 — AIM EXPRESS-G diagram: explicit_table_function (36 of 91)

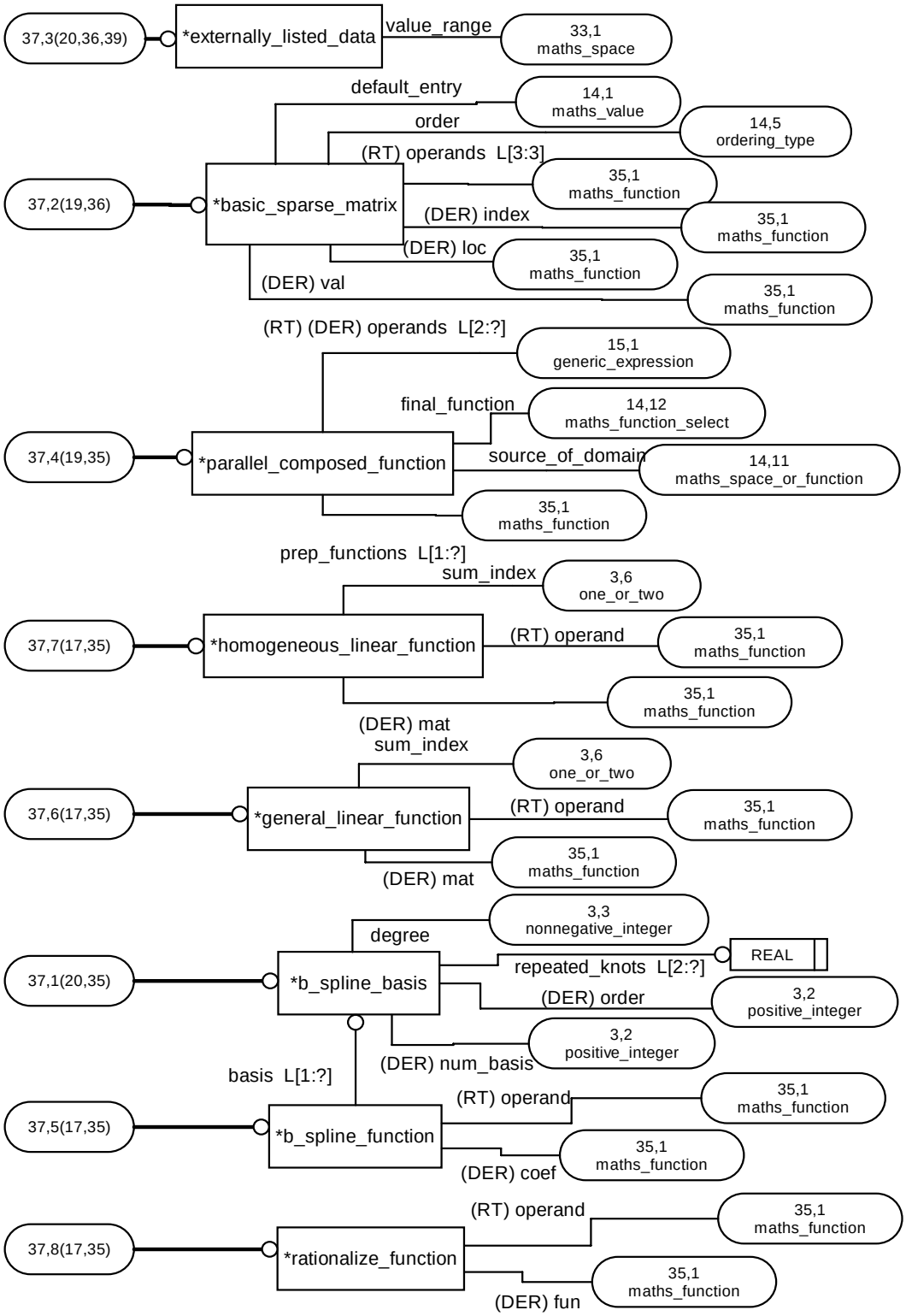


Figure H.37 — AIM EXPRESS-G diagram: maths_functions_1 (37 of 91)

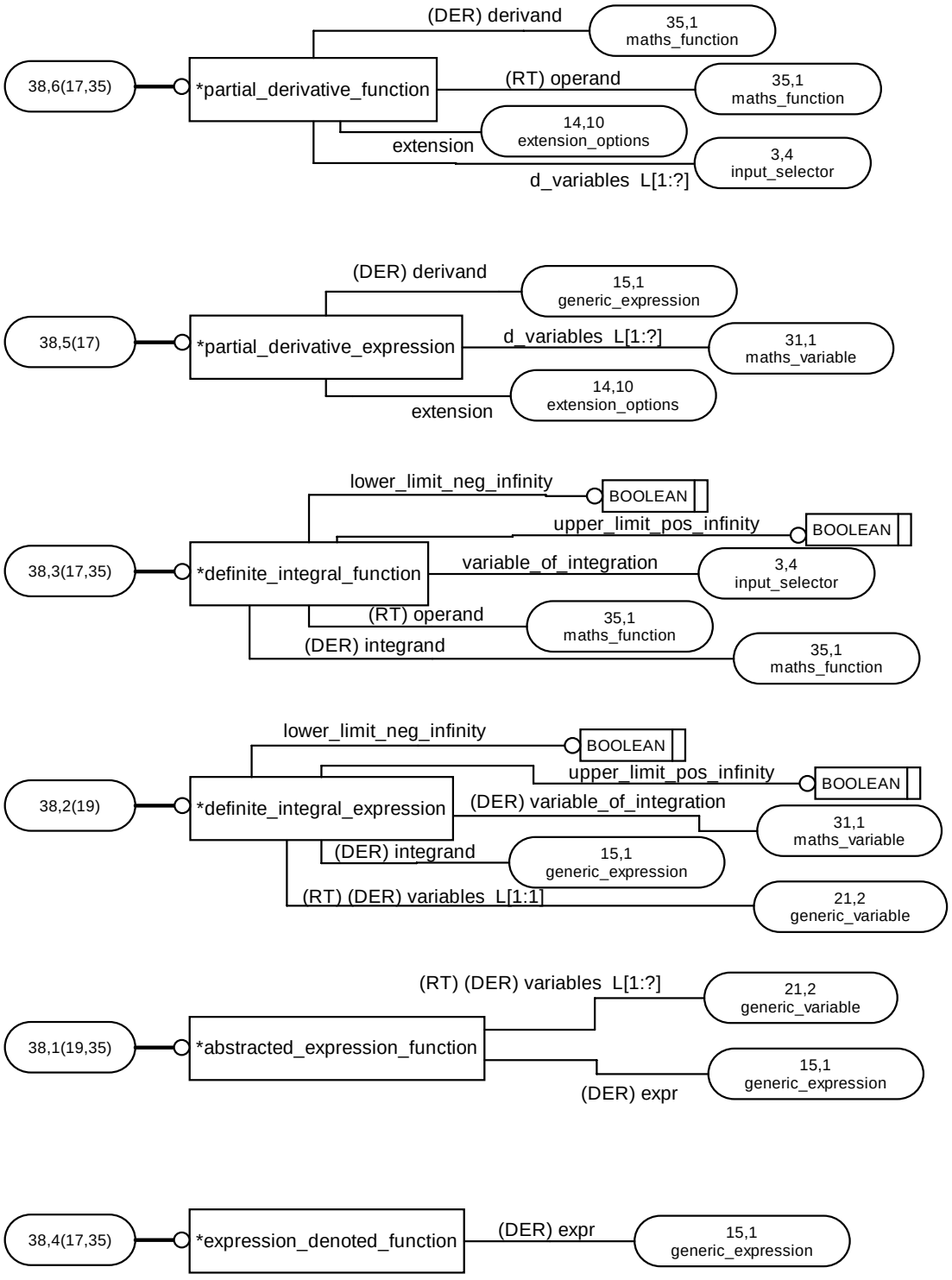


Figure H.38 — AIM EXPRESS-G diagram: maths_functions_2 (38 of 91)

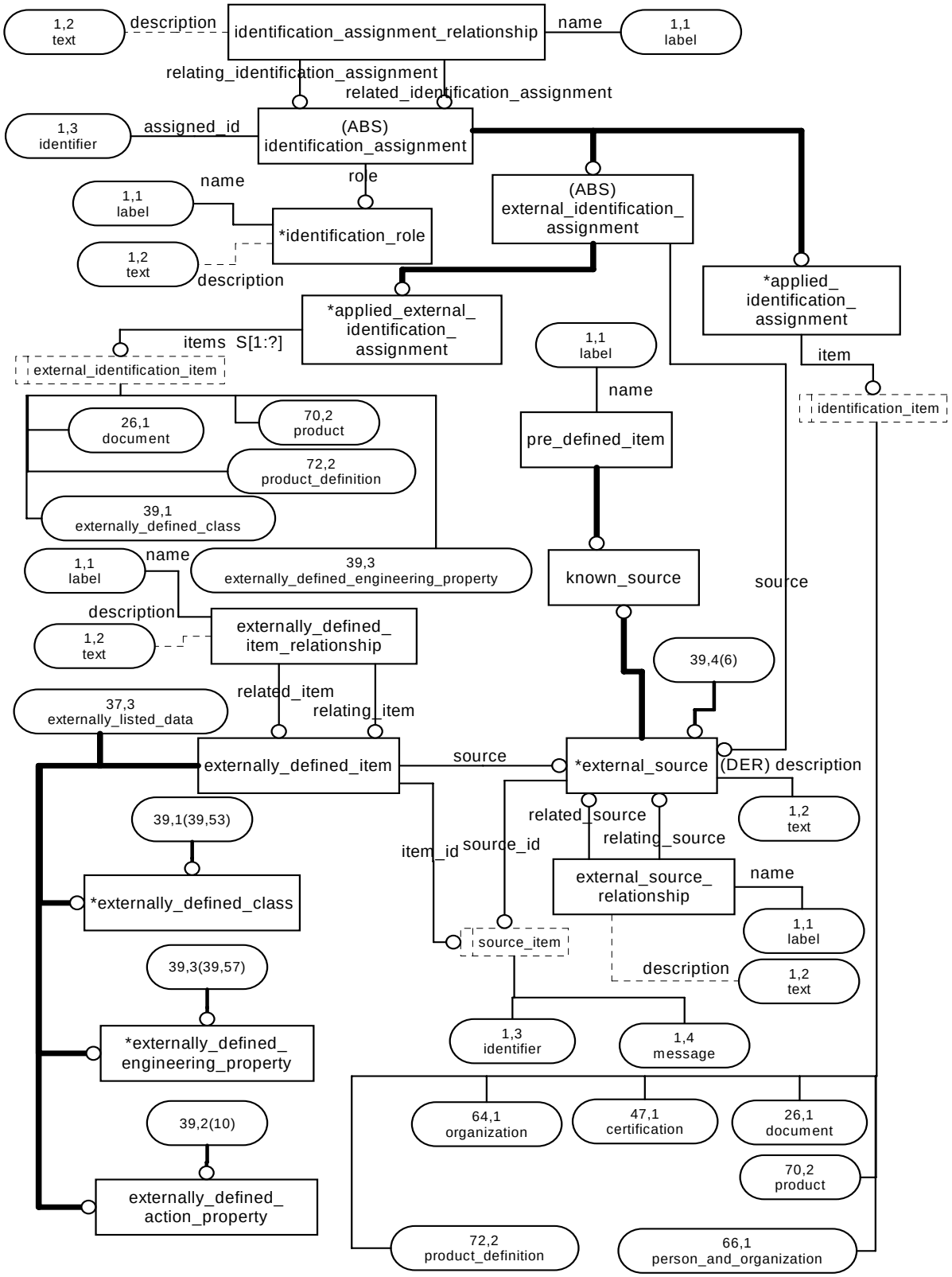


Figure H.39 — AIM EXPRESS-G diagram: external_item (39 of 91)

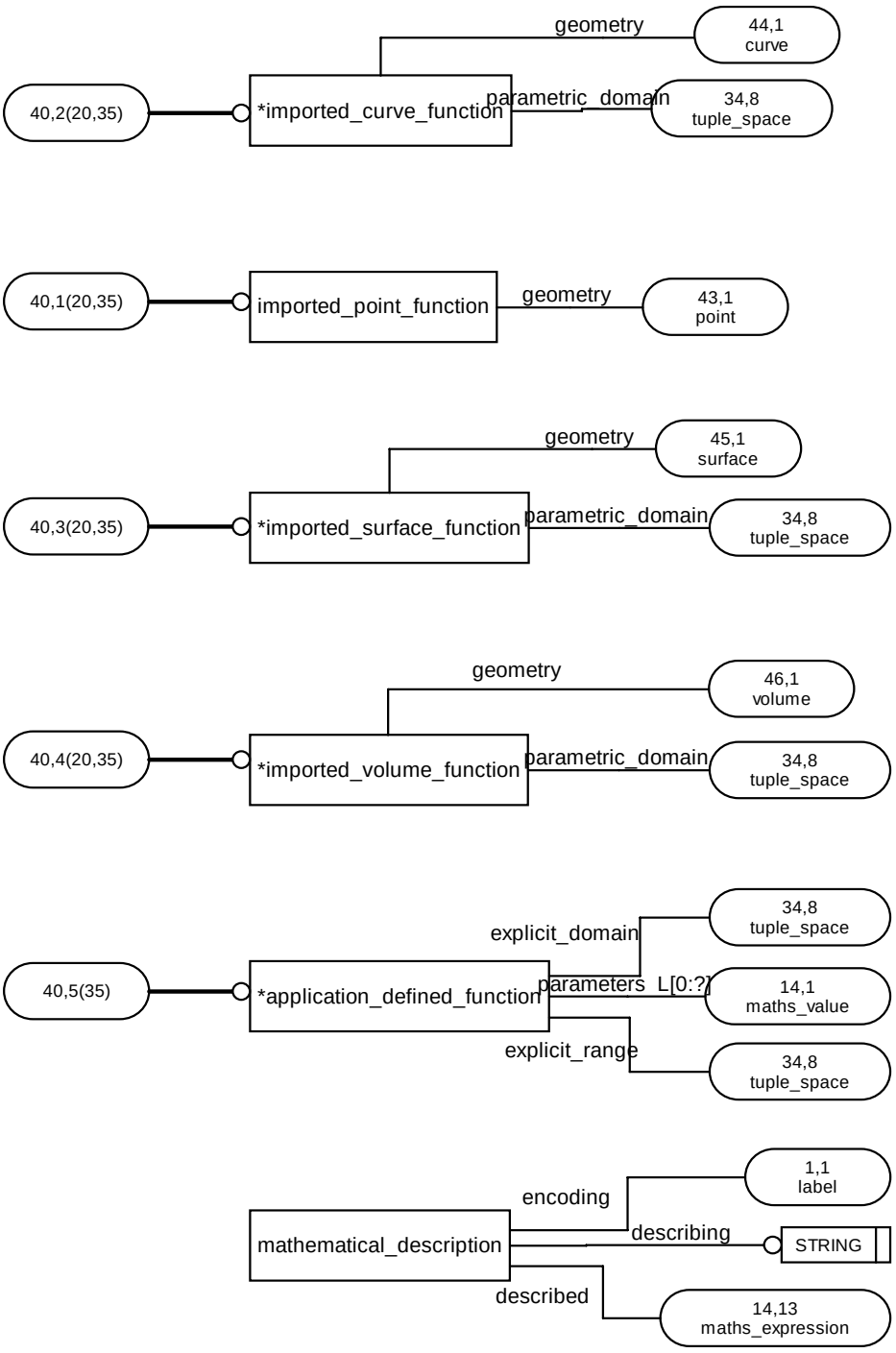


Figure H.40 — AIM EXPRESS-G diagram: imported_function (40 of 91)

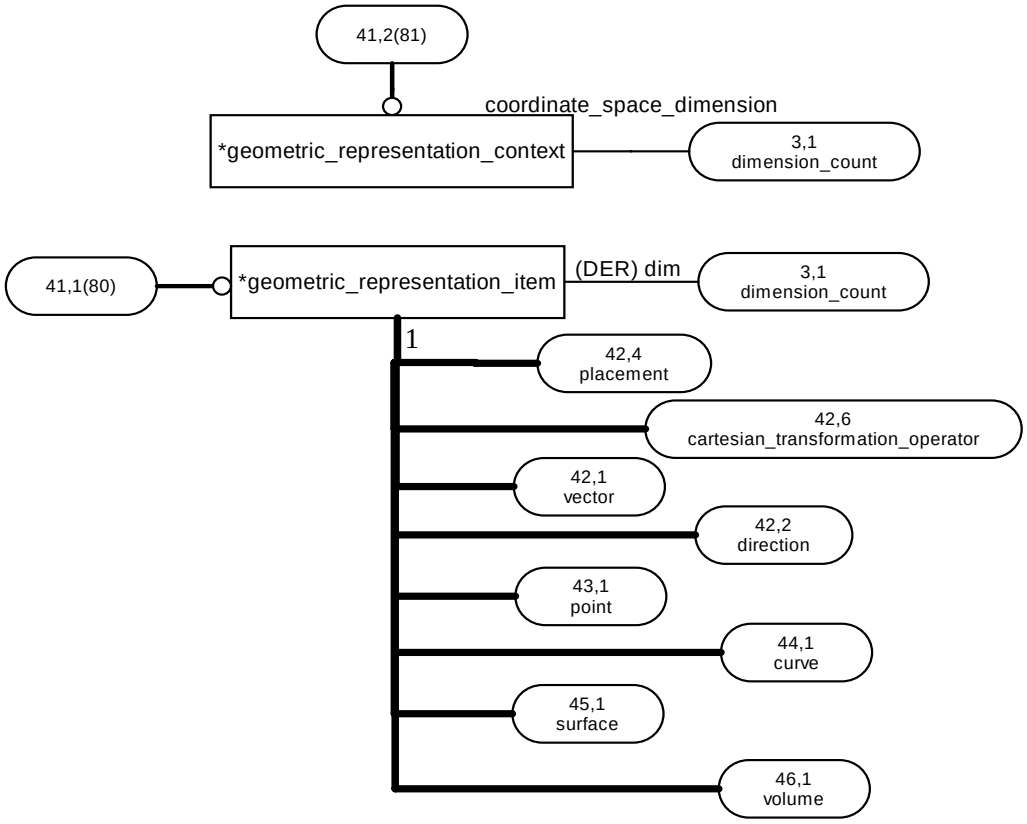


Figure H.41 — AIM EXPRESS-G diagram: geometric_representation_item (41 of 91)

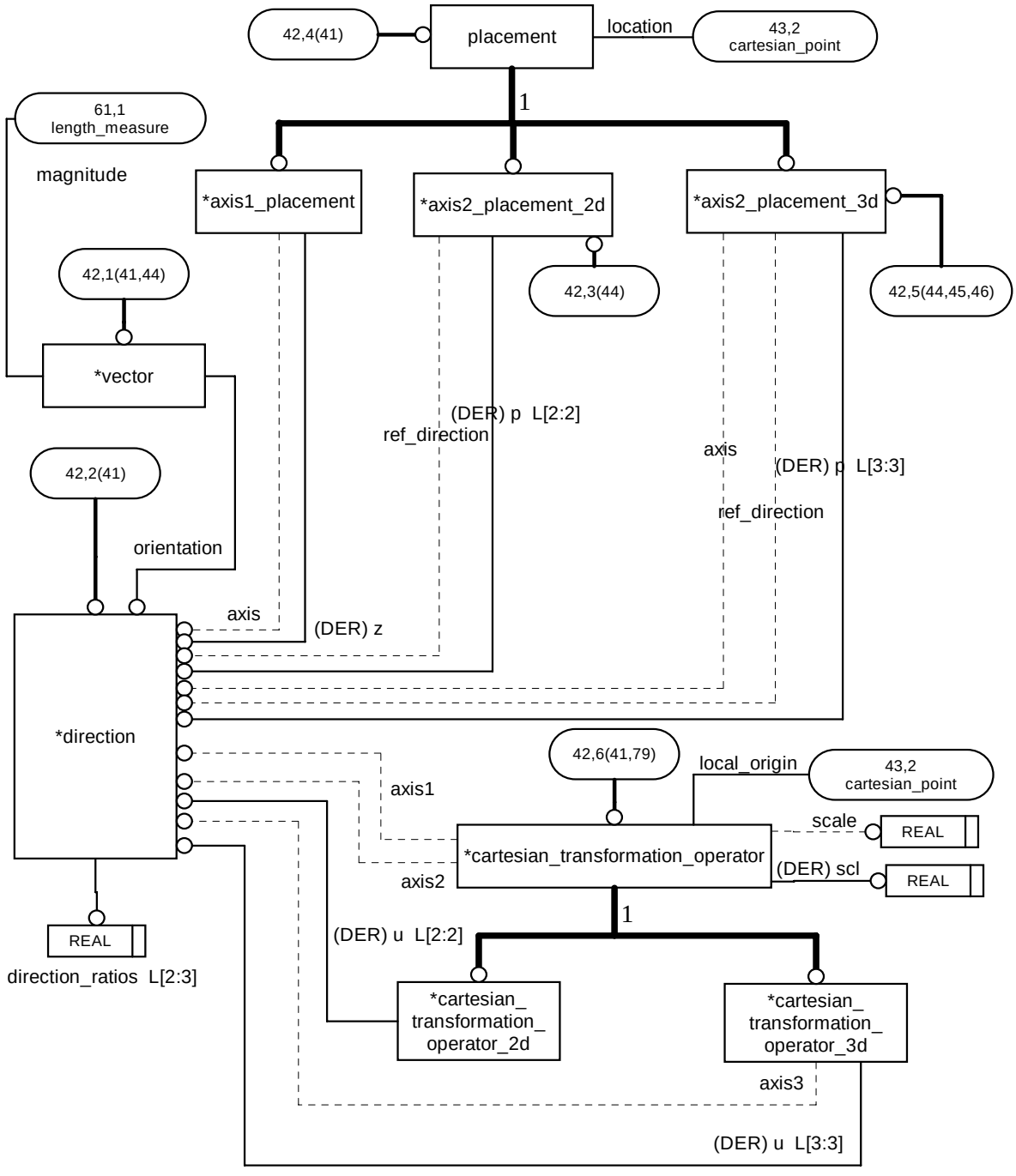


Figure H.42 — AIM EXPRESS-G diagram: placement (42 of 91)

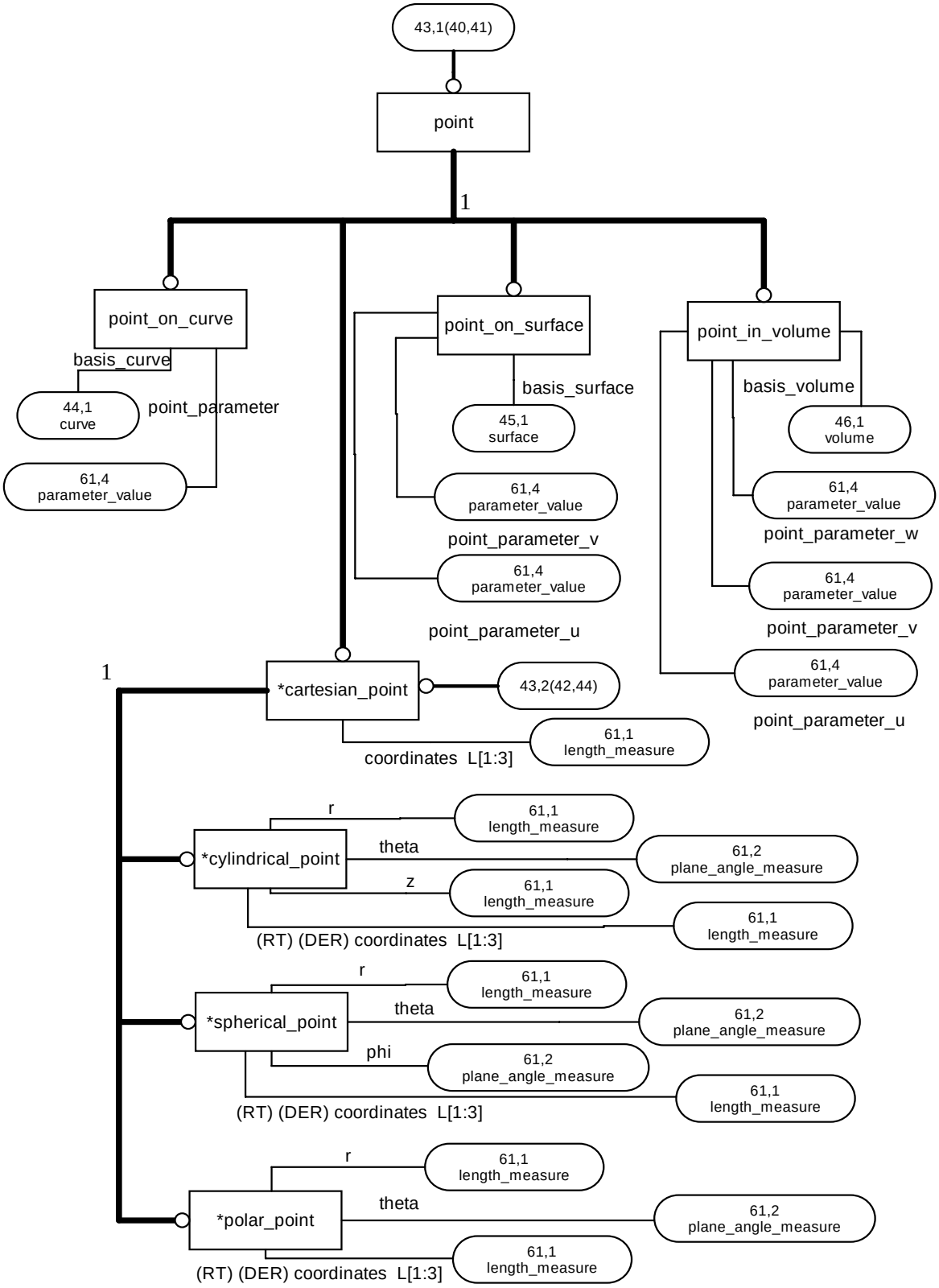


Figure H.43 — AIM EXPRESS-G diagram: point (43 of 91)

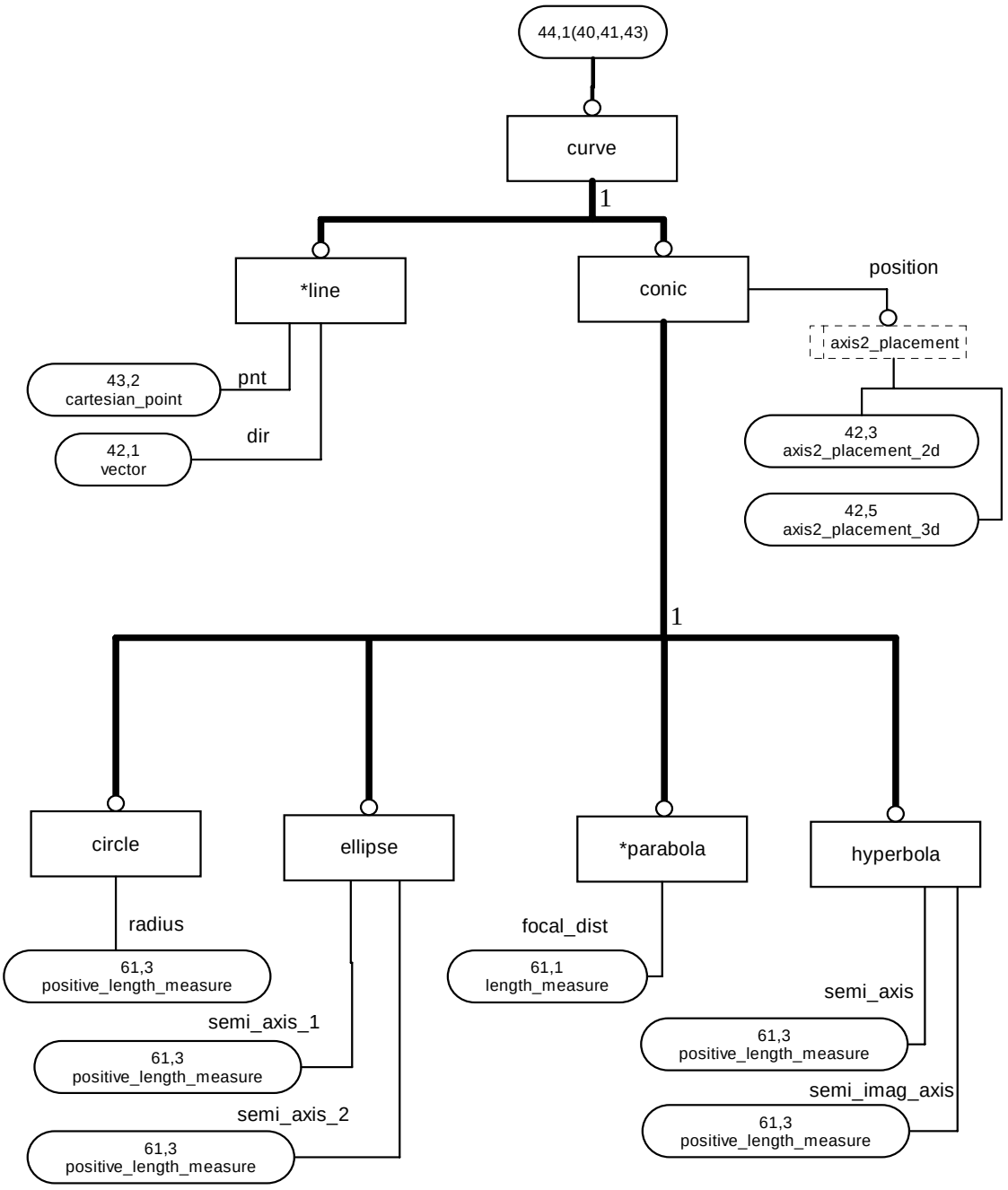


Figure H.44 — AIM EXPRESS-G diagram: curve (44 of 91)

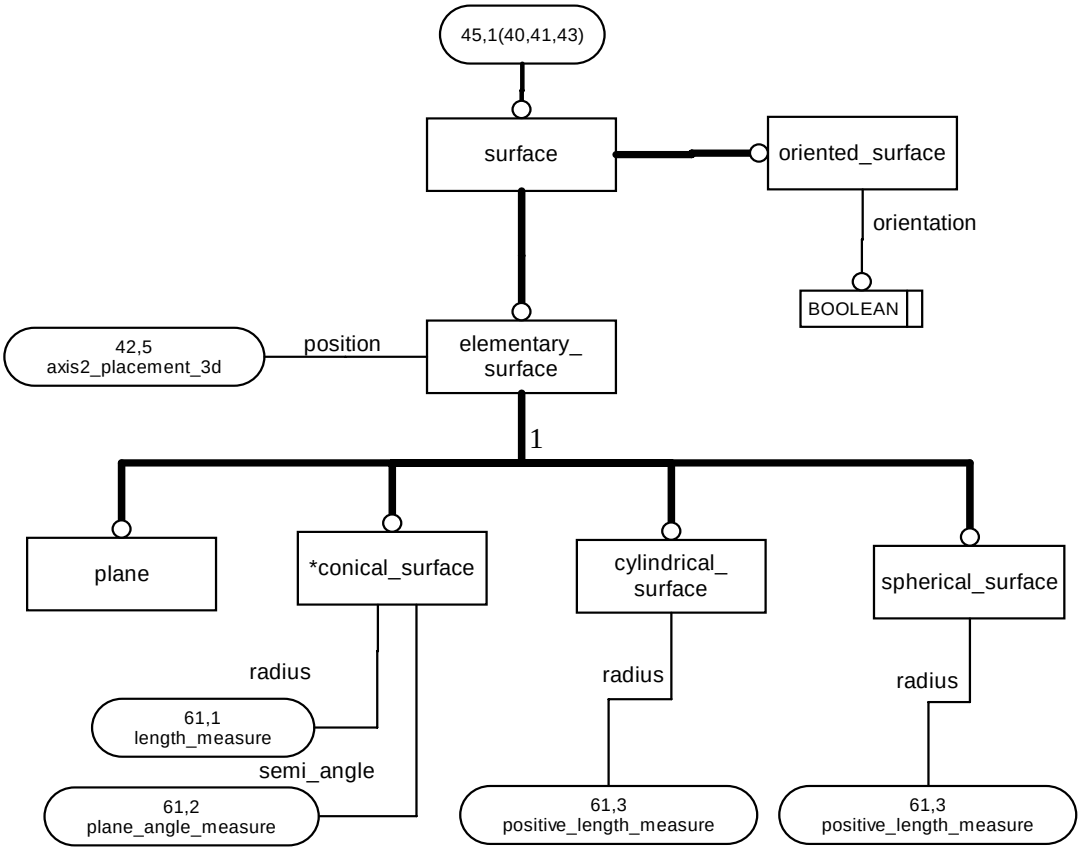


Figure H.45 — AIM EXPRESS-G diagram: surface (45 of 91)

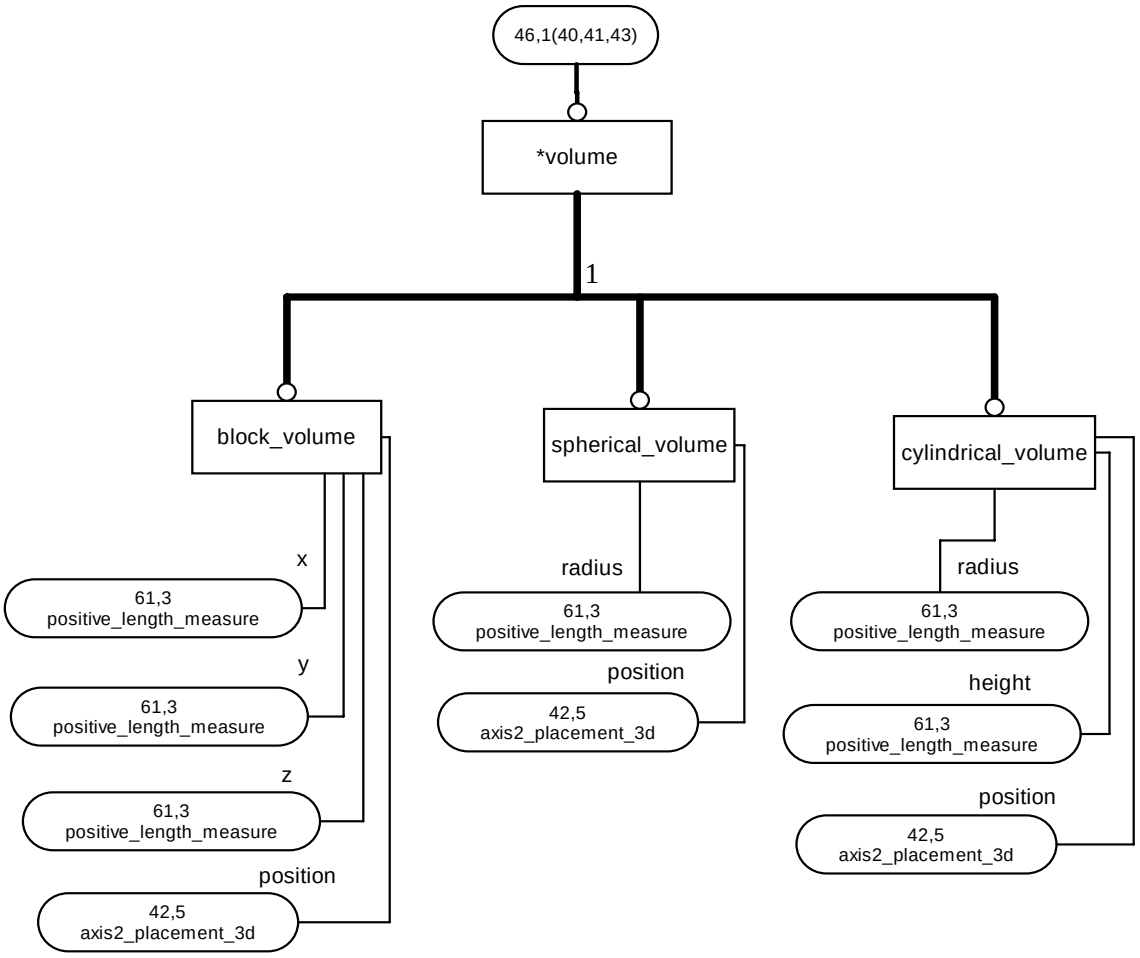


Figure H.46 — AIM EXPRESS-G diagram: volume (46 of 91)

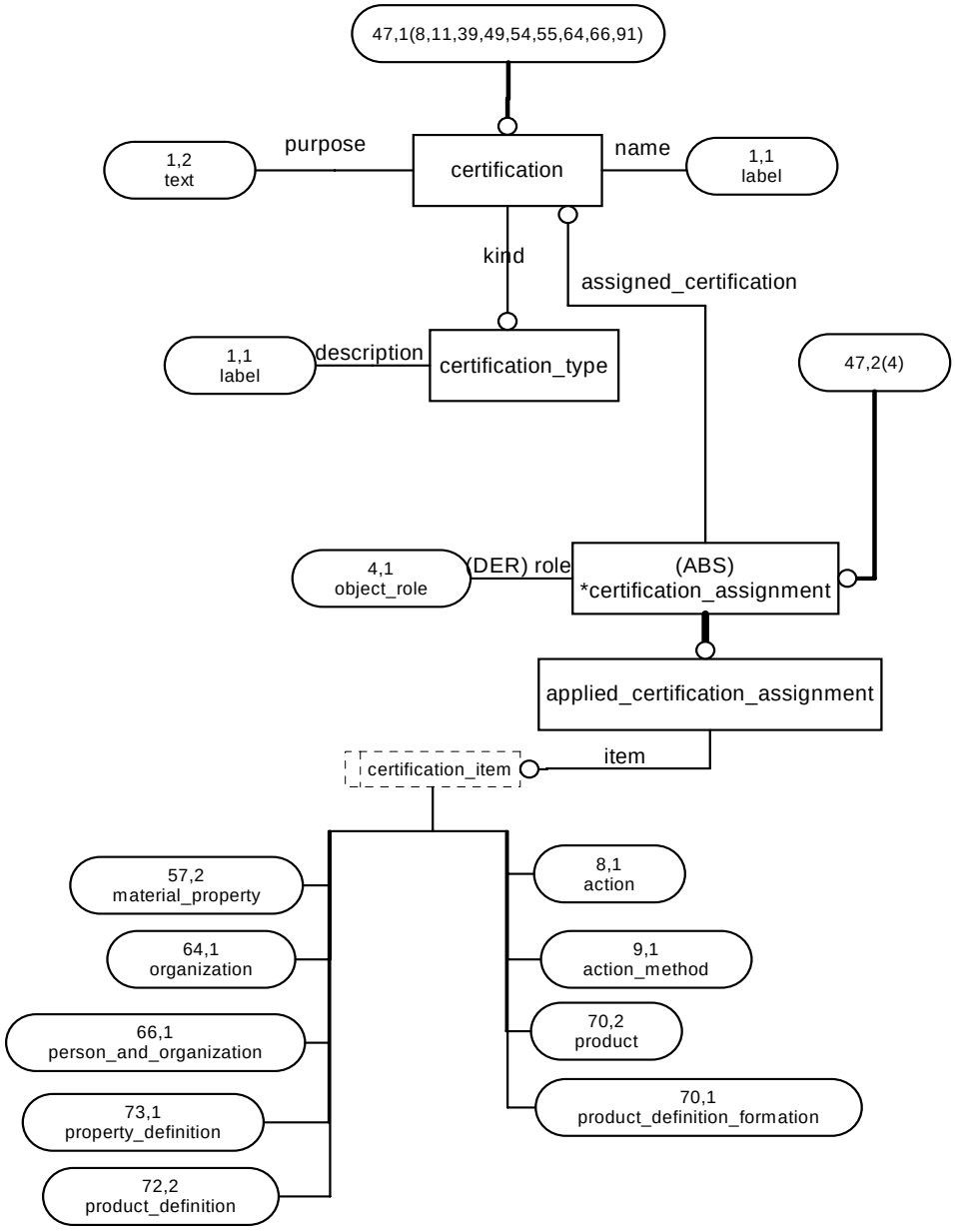


Figure H.47 — AIM EXPRESS-G diagram: certification (47 of 91)

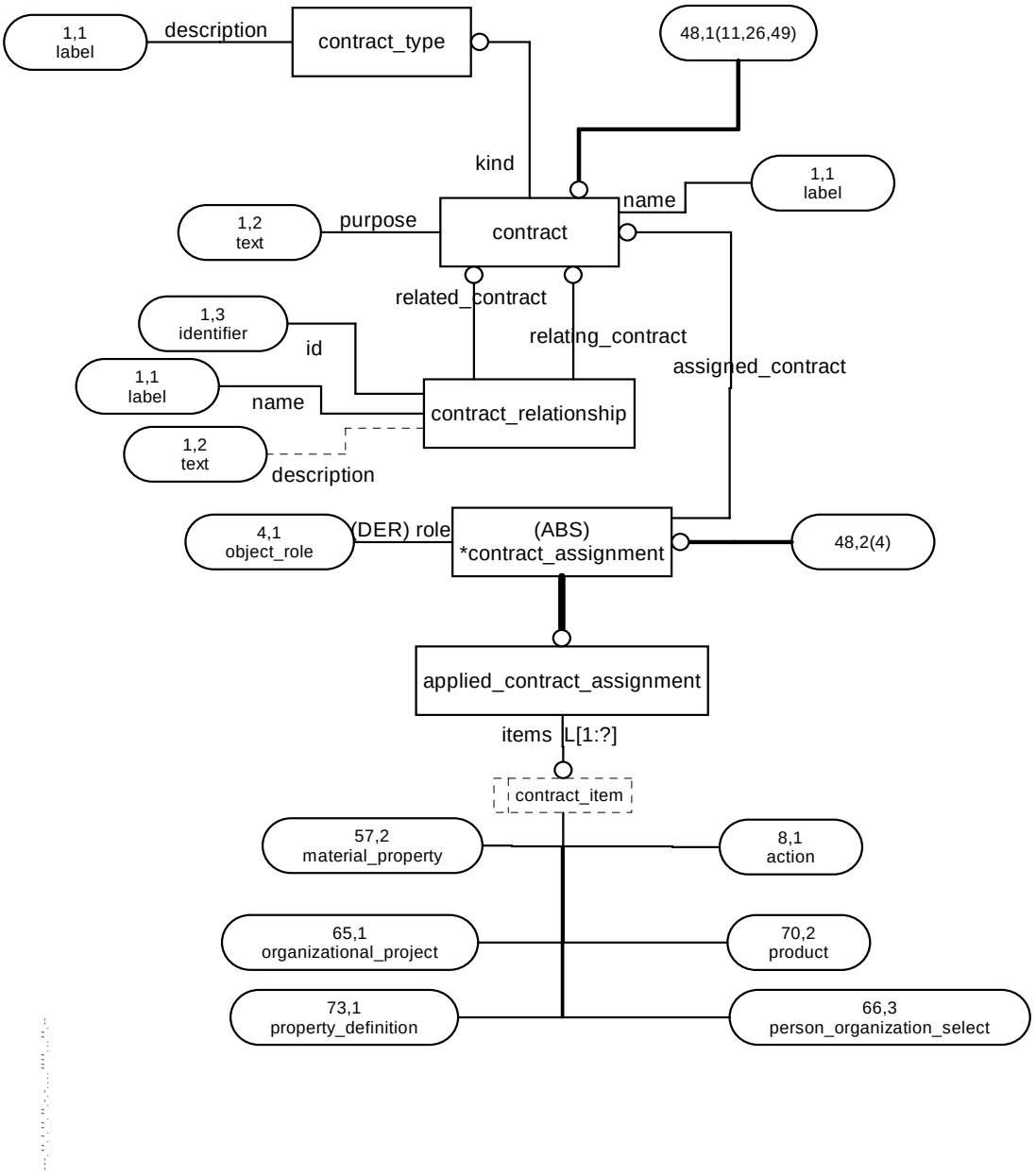


Figure H.48 — AIM EXPRESS-G diagram: contract (48 of 91)

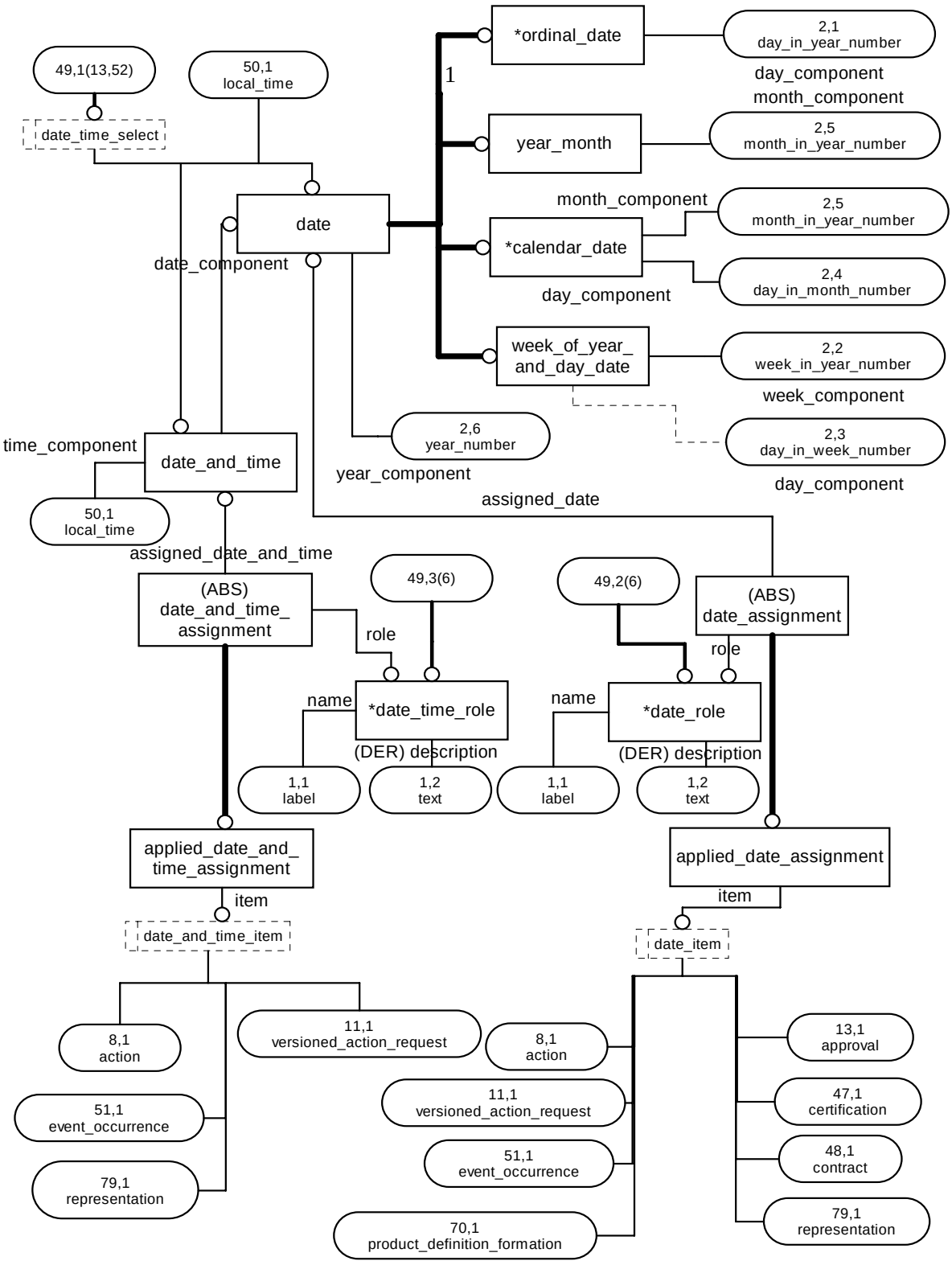


Figure H.49 — AIM EXPRESS-G diagram: date_and_time_entities (49 of 91)

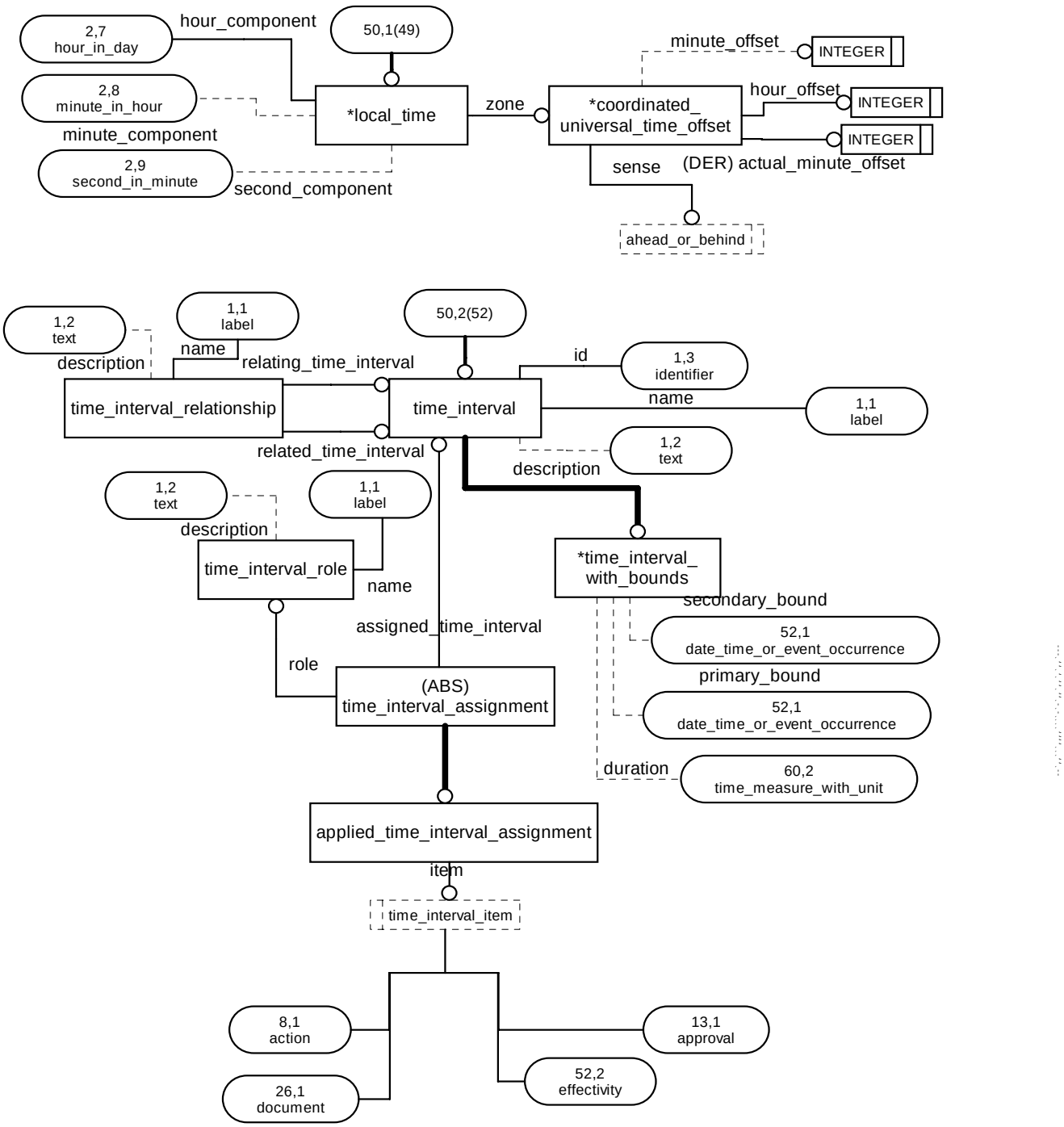


Figure H.50 — AIM EXPRESS-G diagram: time_interval (50 of 91)

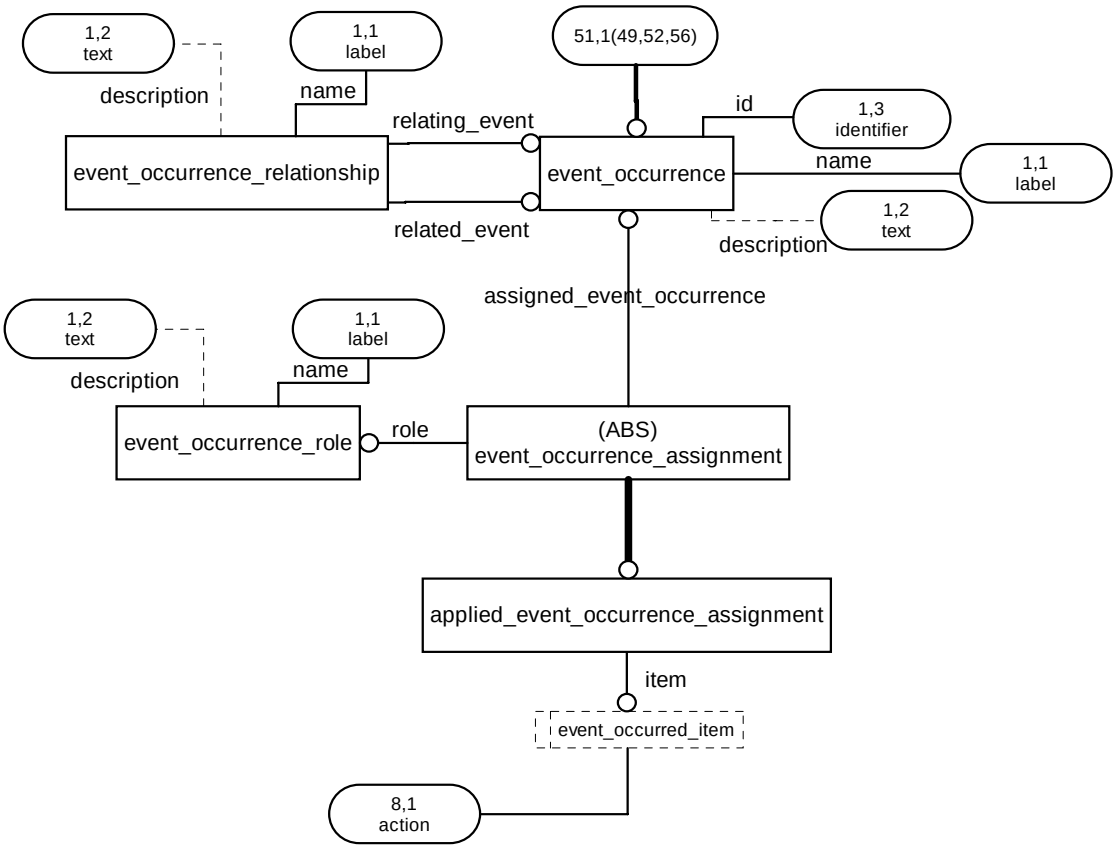


Figure H.51 — AIM EXPRESS-G diagram: event_occurrence (51 of 91)

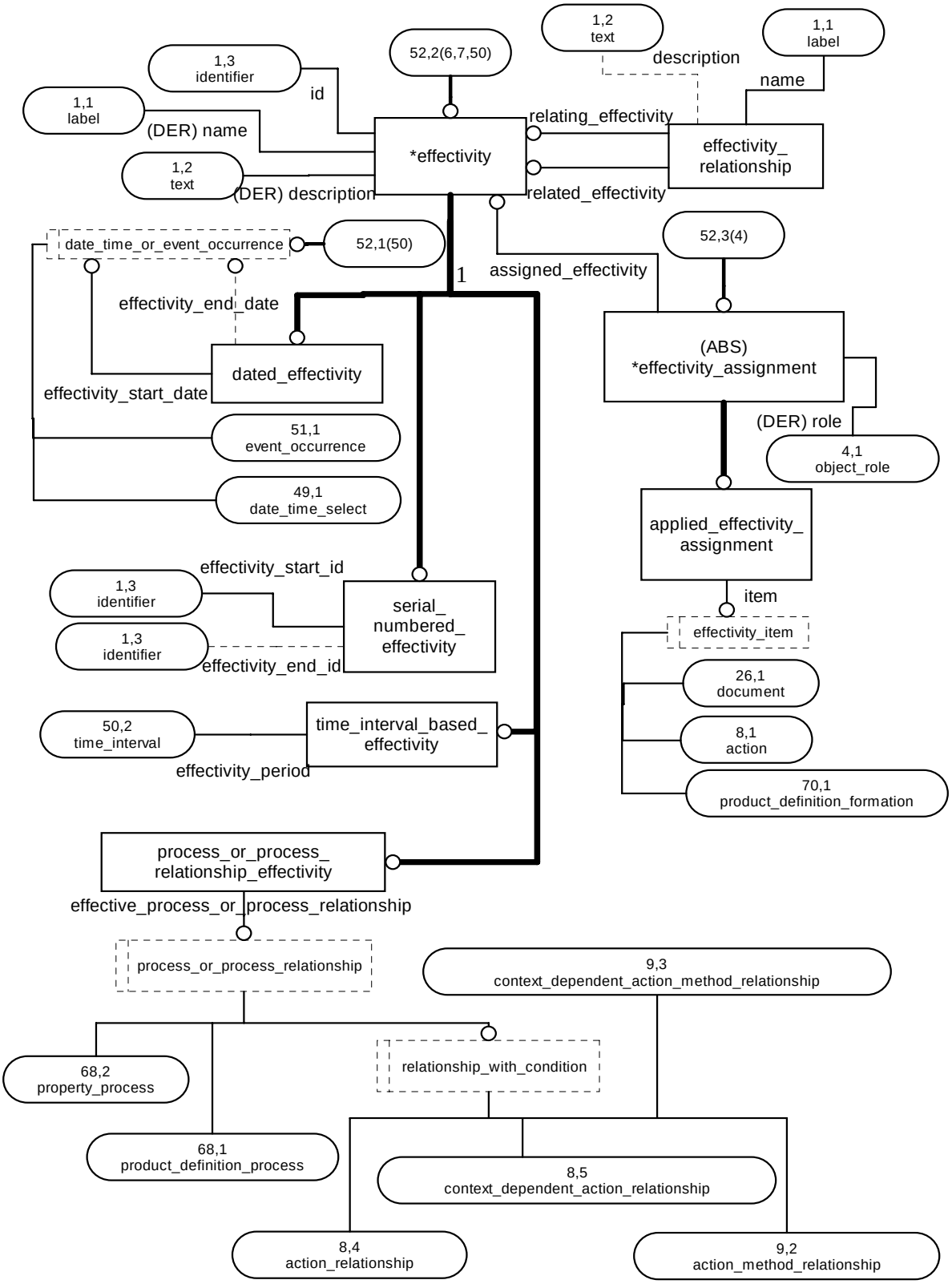


Figure H.52 — AIM EXPRESS-G diagram: event_occurrence (51 of 91)

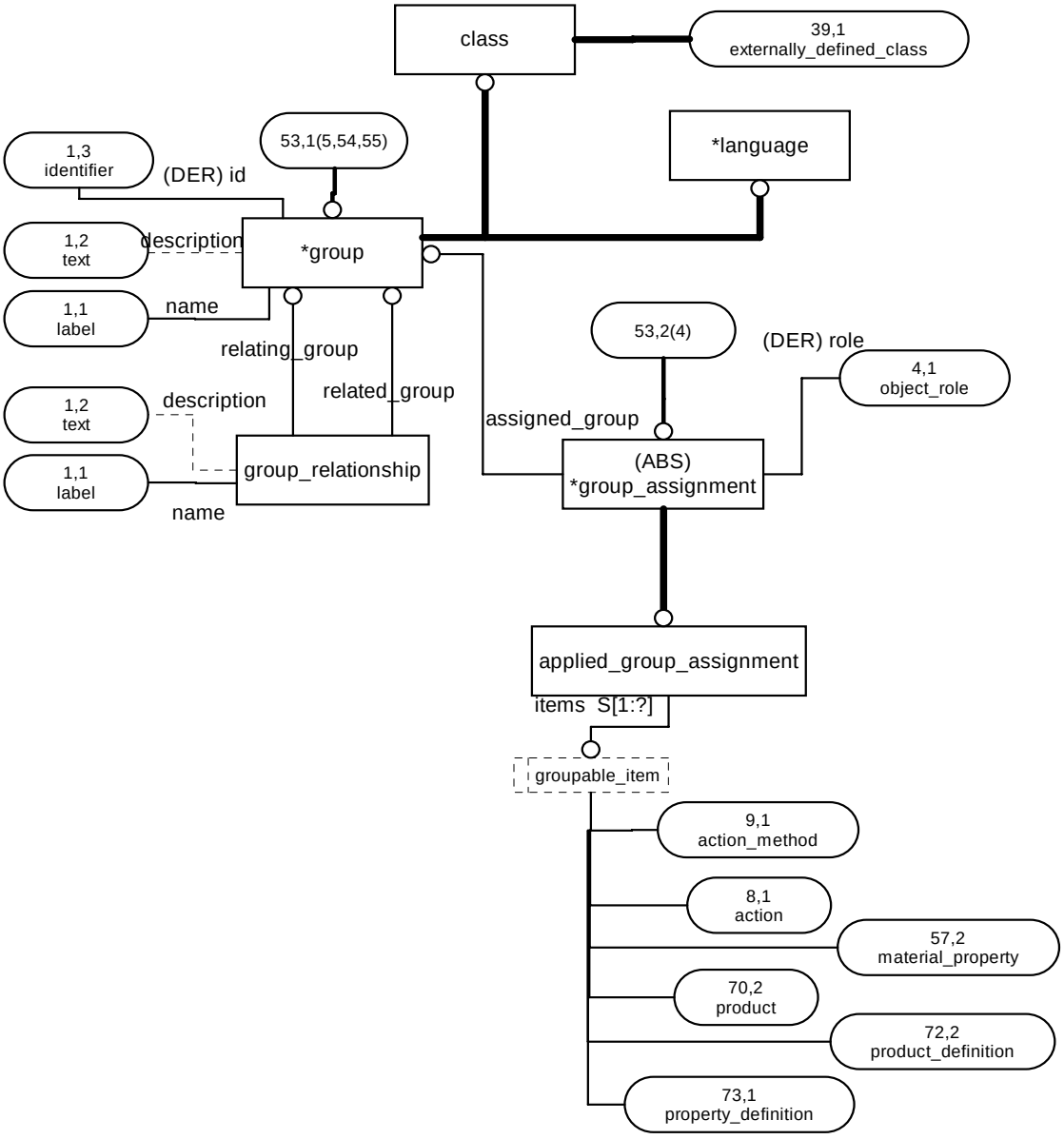


Figure H.53 — AIM EXPRESS-G diagram: group (53 of 91)

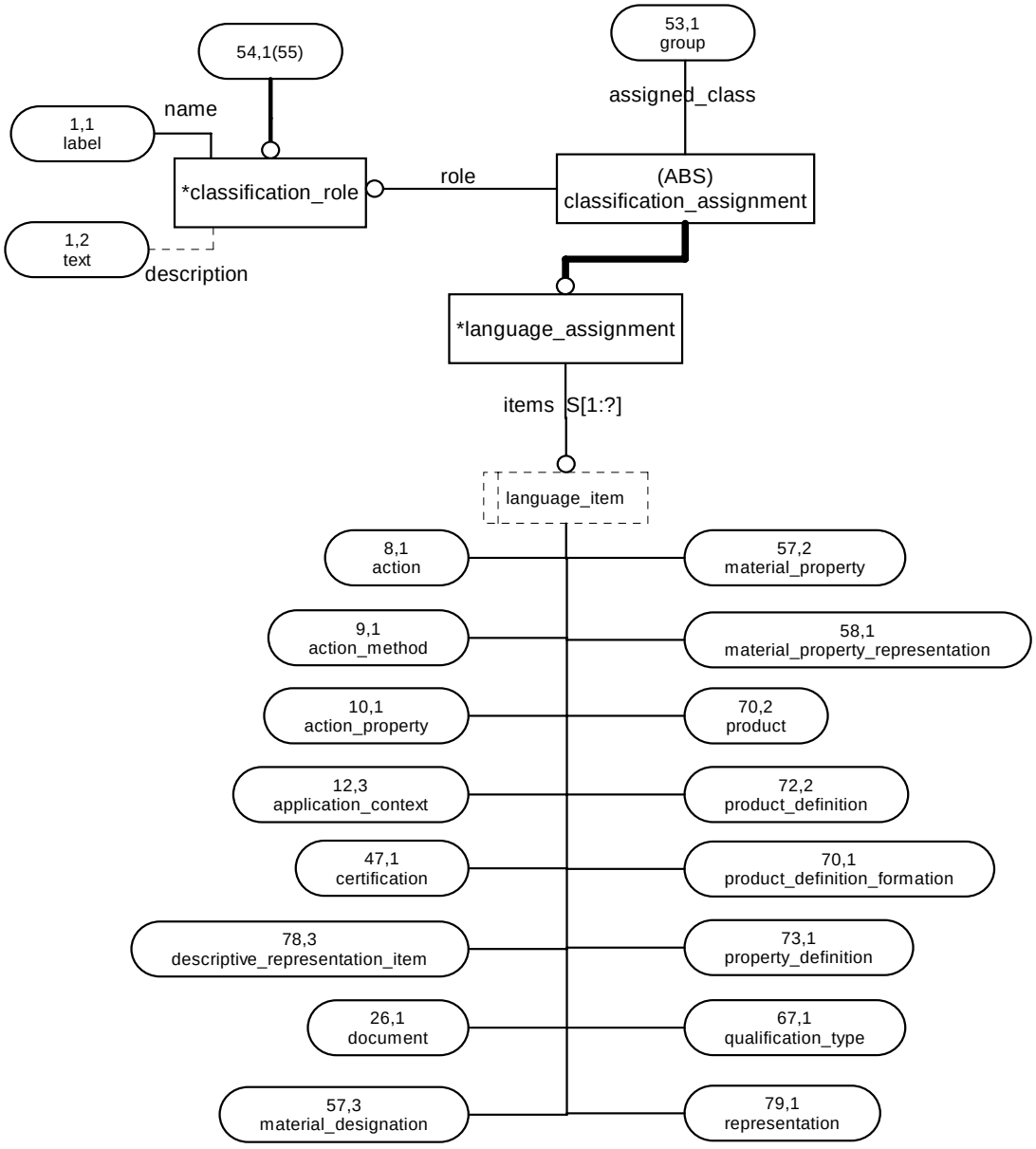


Figure H.54 — AIM EXPRESS-G diagram: language _assignment (54 of 91)

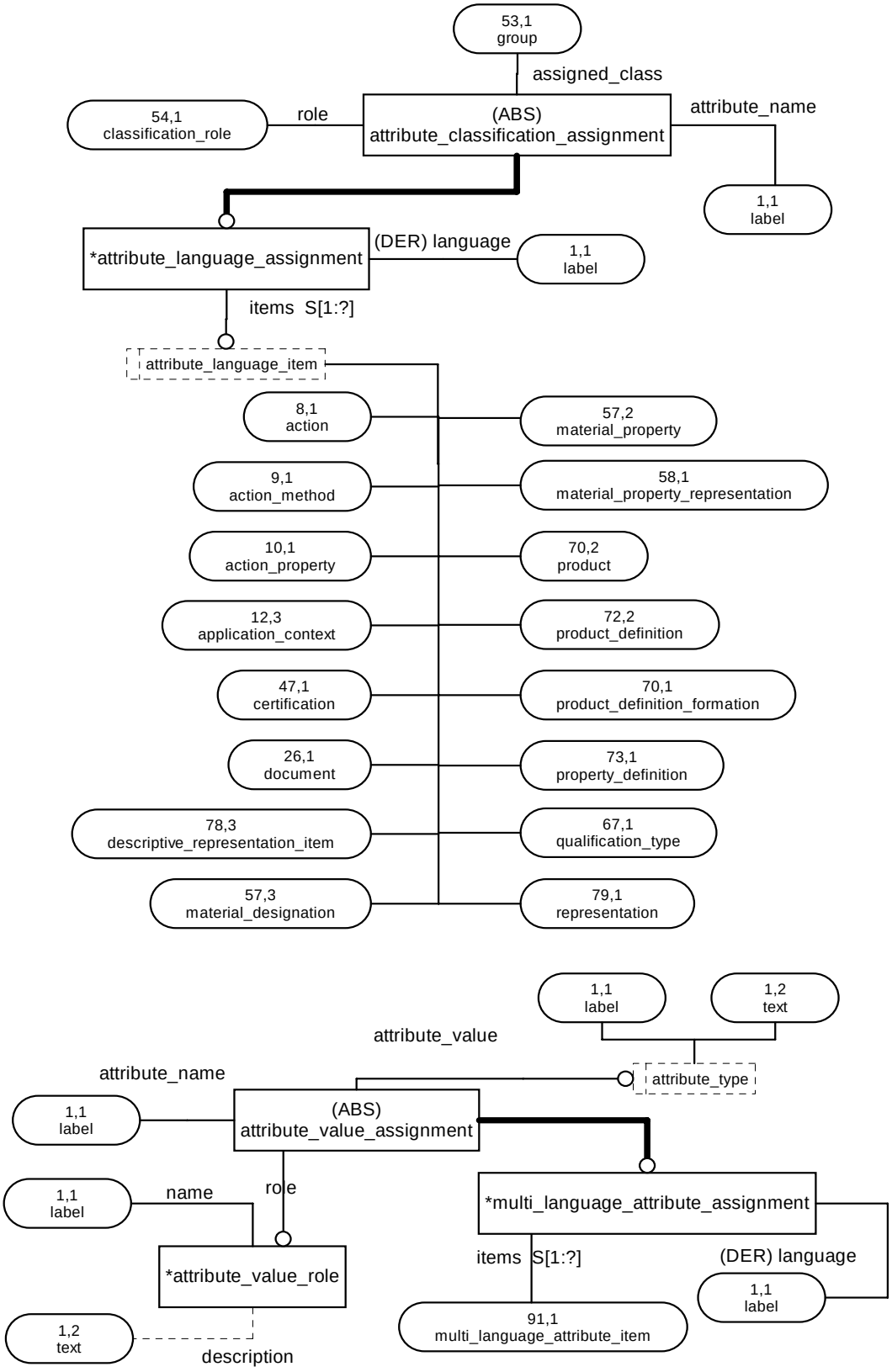


Figure H.55 — AIM EXPRESS-G diagram: attribute_language_assignment (55 of 91)

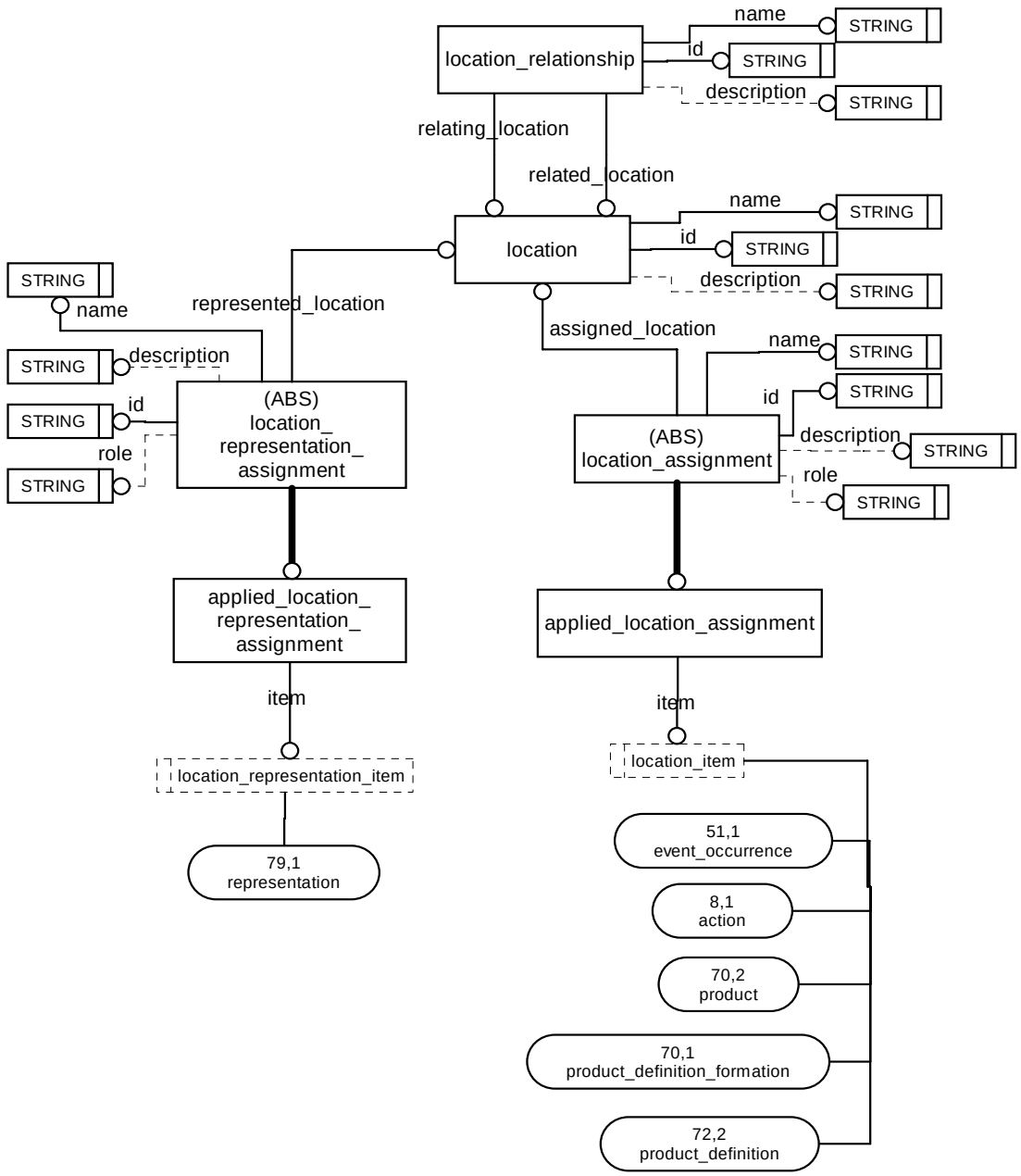


Figure H.56 — AIM EXPRESS-G diagram: location (56 of 91)

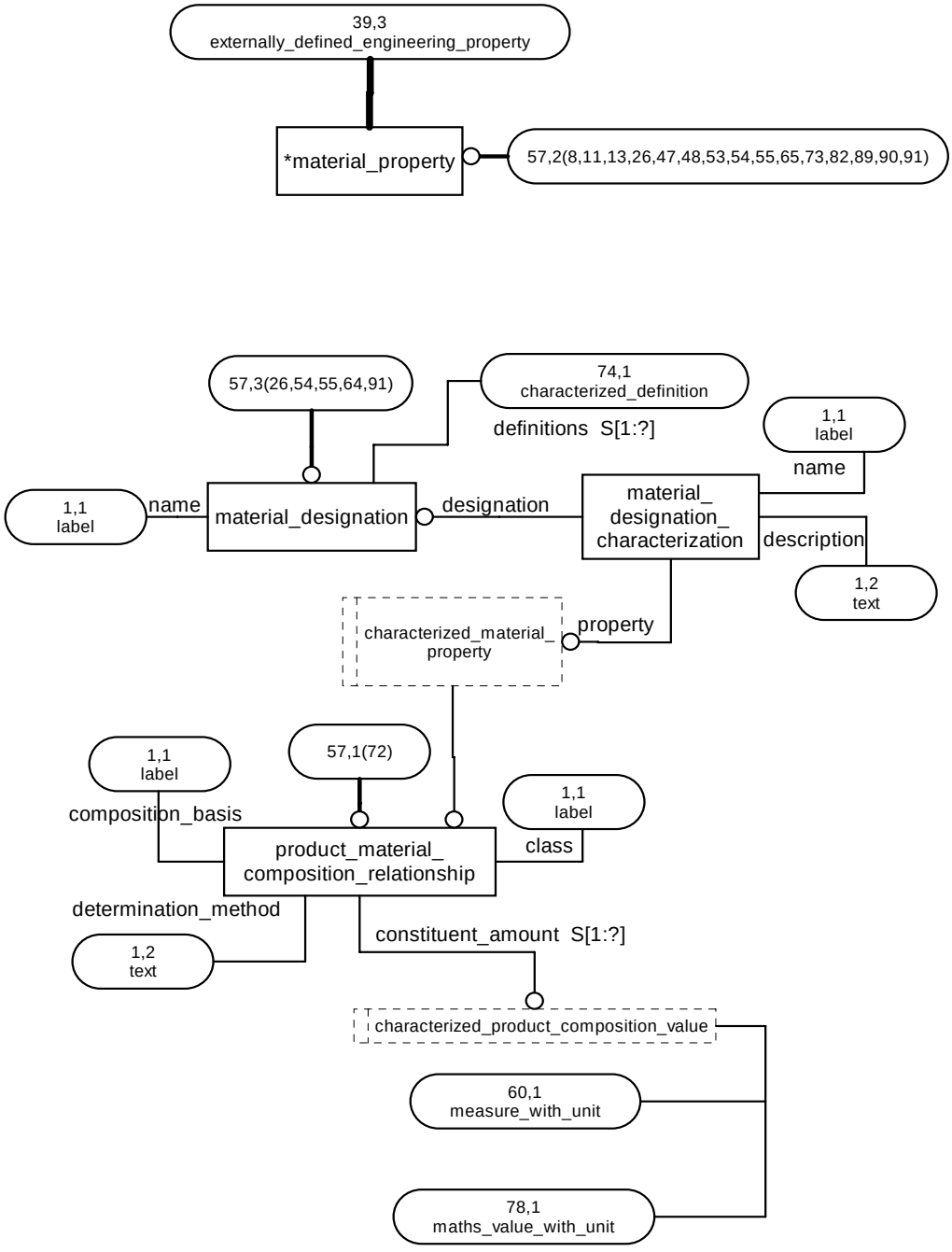


Figure H.57 — AIM EXPRESS-G diagram: material_property (57 of 91)

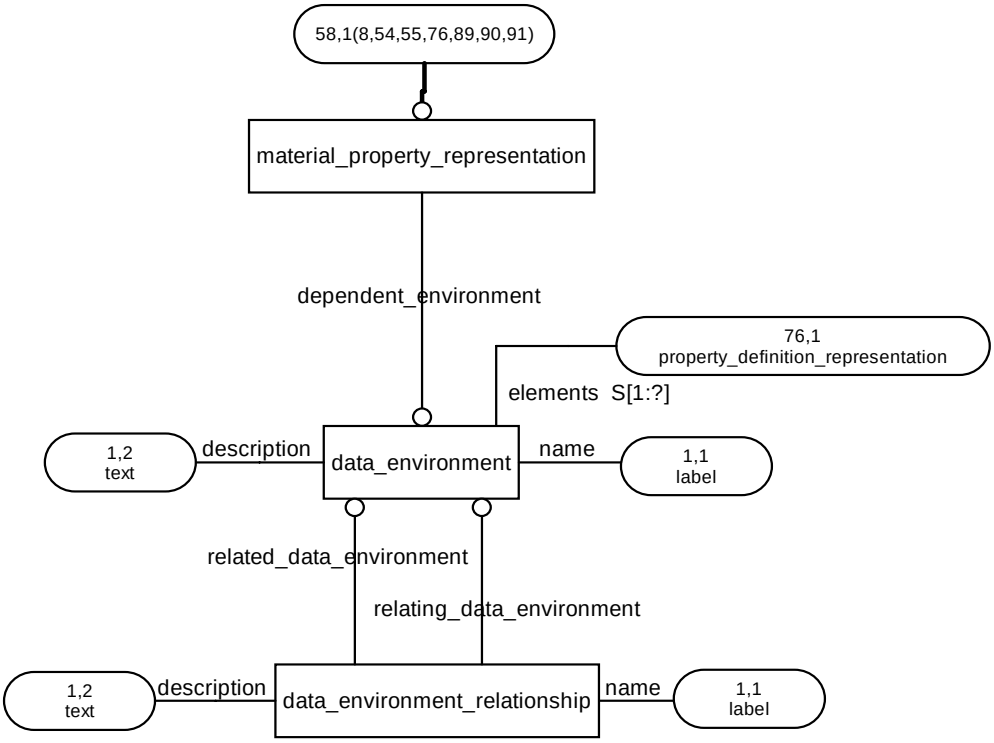


Figure H.58 — AIM EXPRESS-G diagram: material_property_representation (58 of 91)

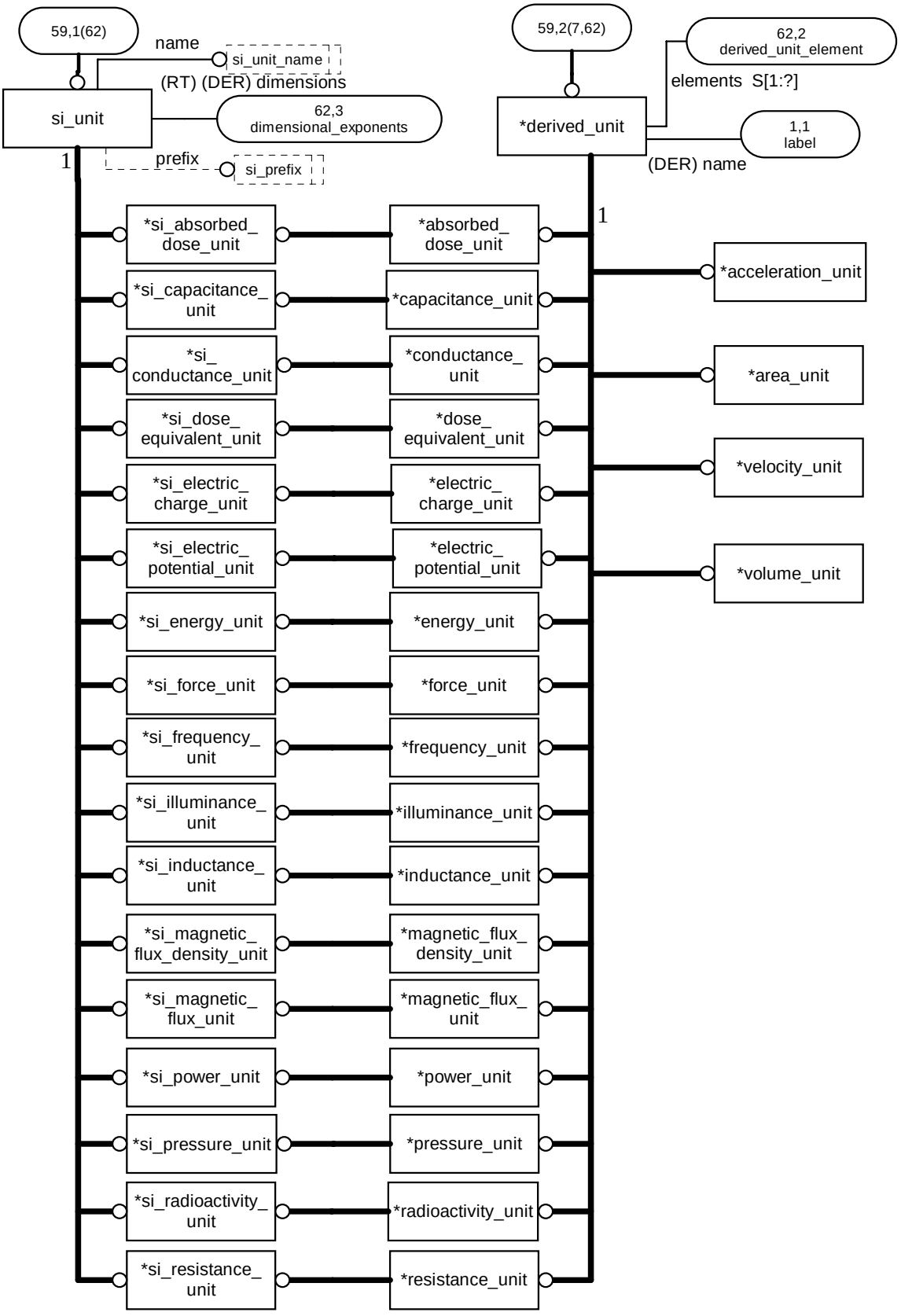


Figure H.59 — AIM EXPRESS-G diagram: SI_unit and derived _unit (59 of 91)

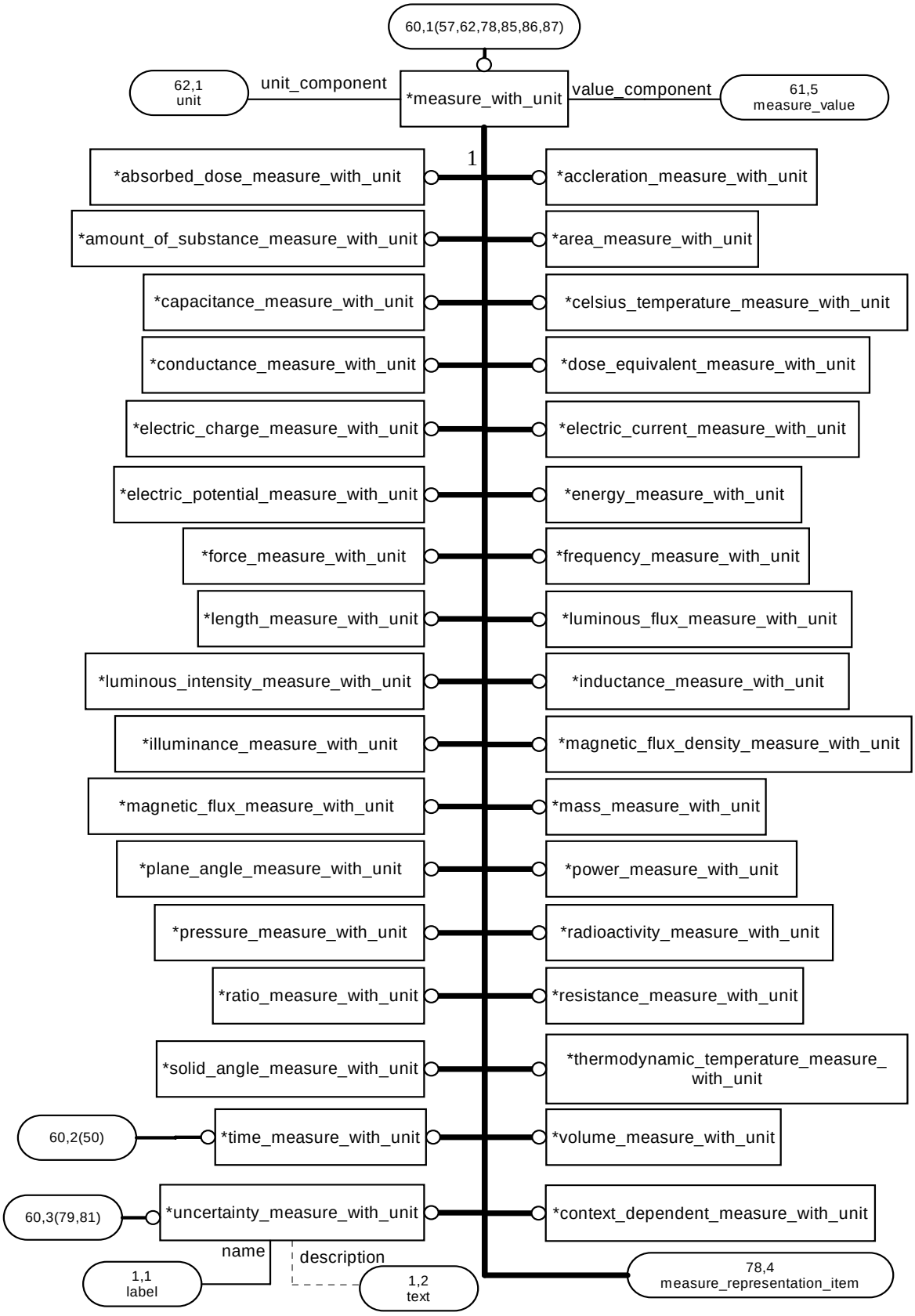


Figure H.60 — AIM EXPRESS-G diagram: measure_with_unit (60 of 91)

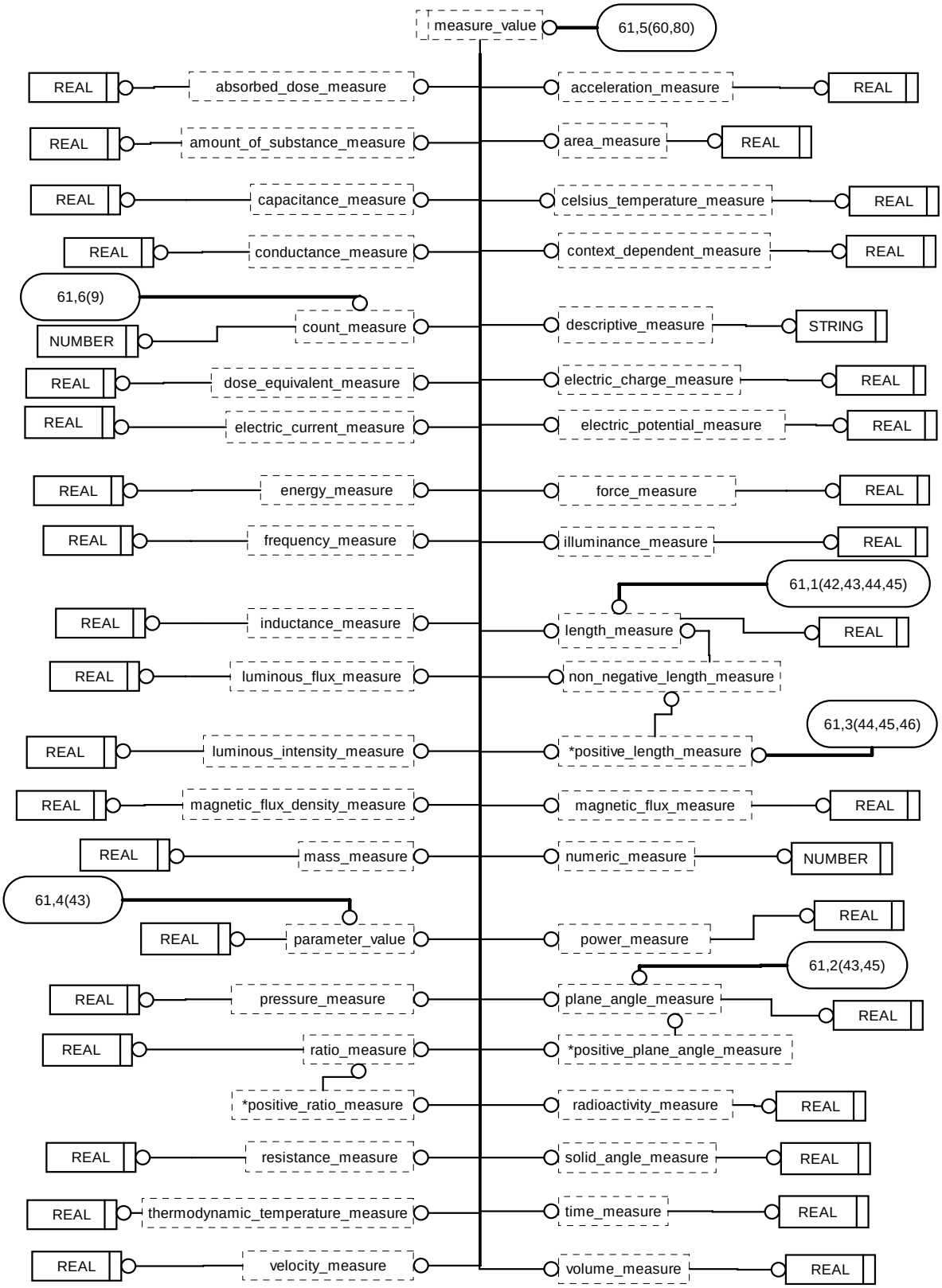


Figure H.61 — AIM EXPRESS-G diagram: measure_value (61 of 91)

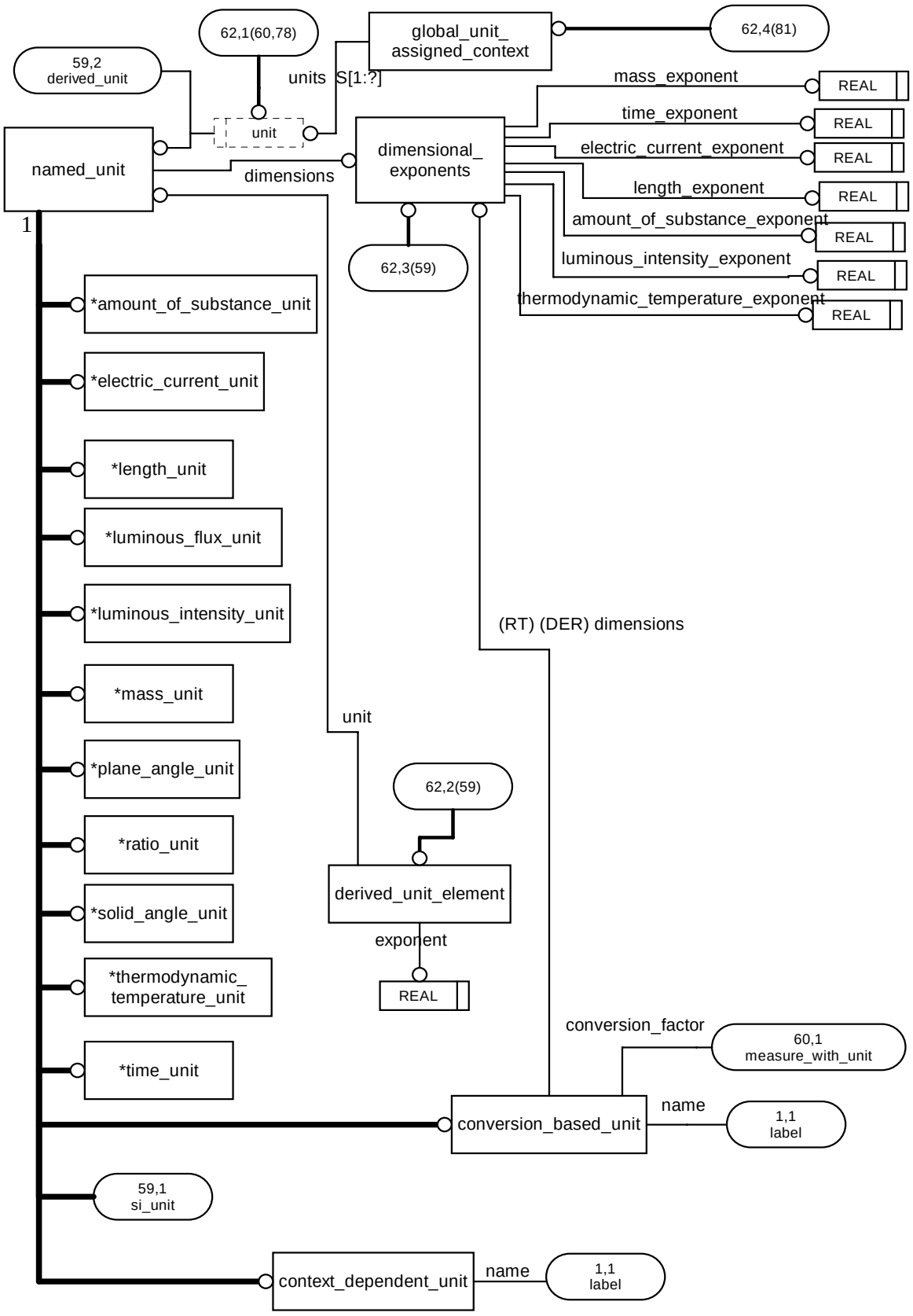


Figure H.62 — AIM EXPRESS-G diagram: named_unit

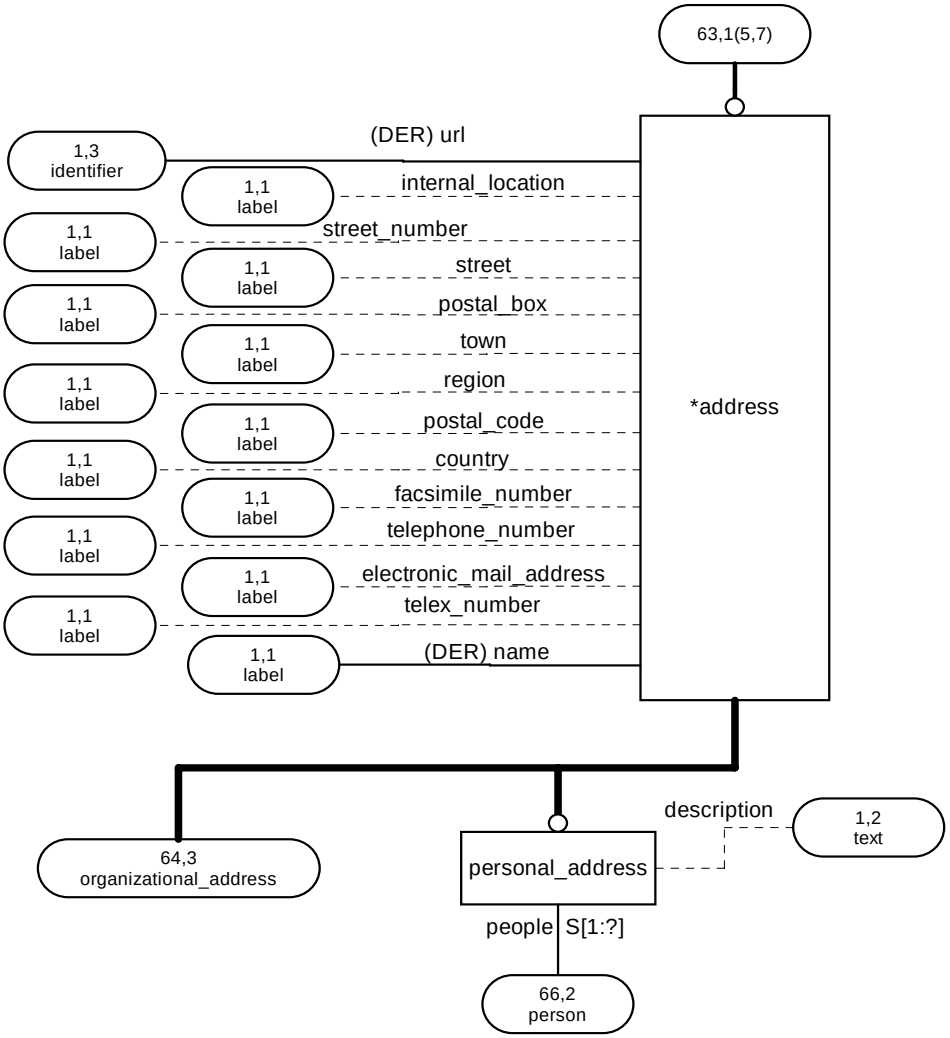


Figure H.63 — AIM EXPRESS-G diagram:address (63 of 91)

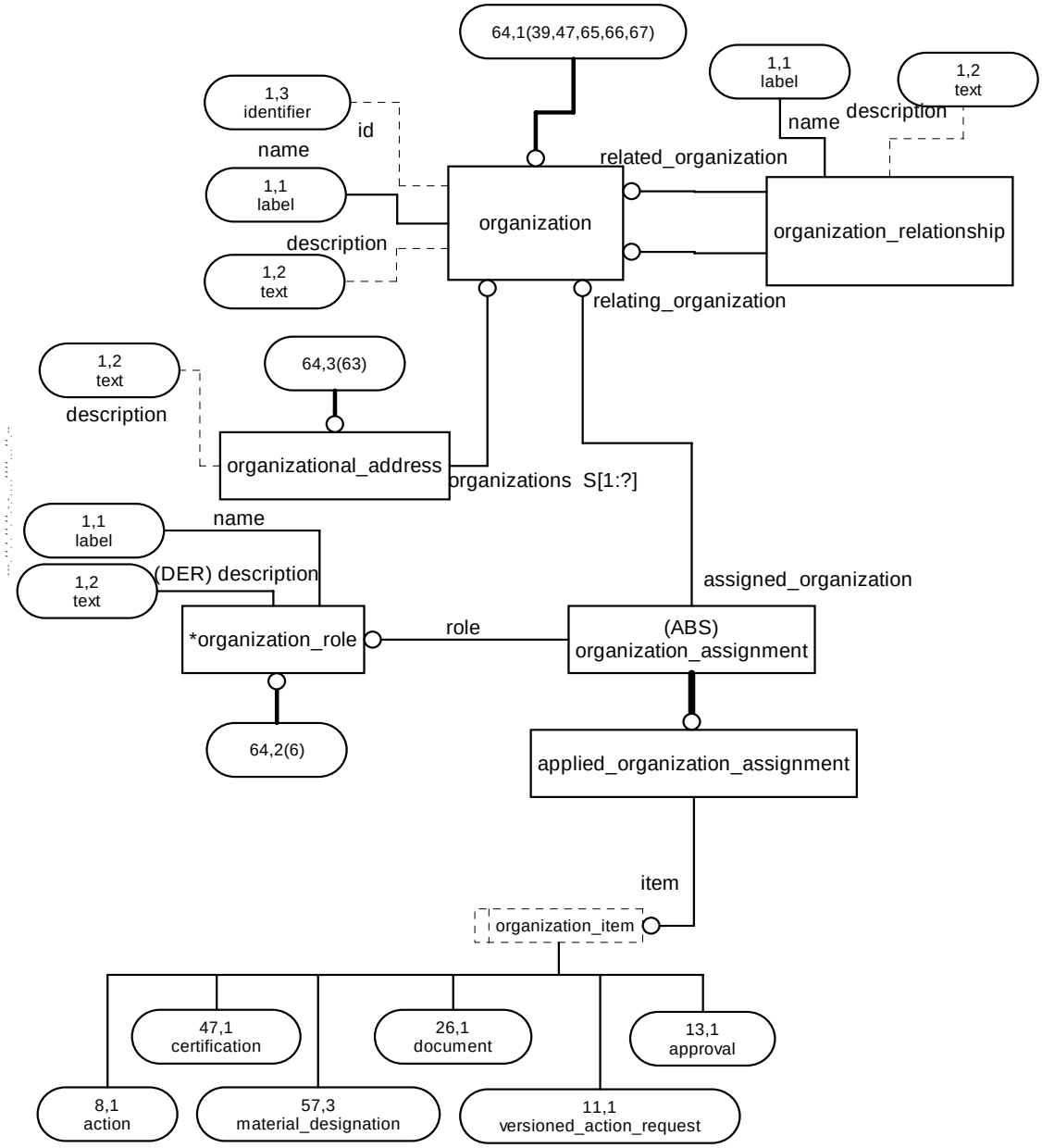


Figure H.64 — AIM EXPRESS-G diagram: organization (64 of 91)

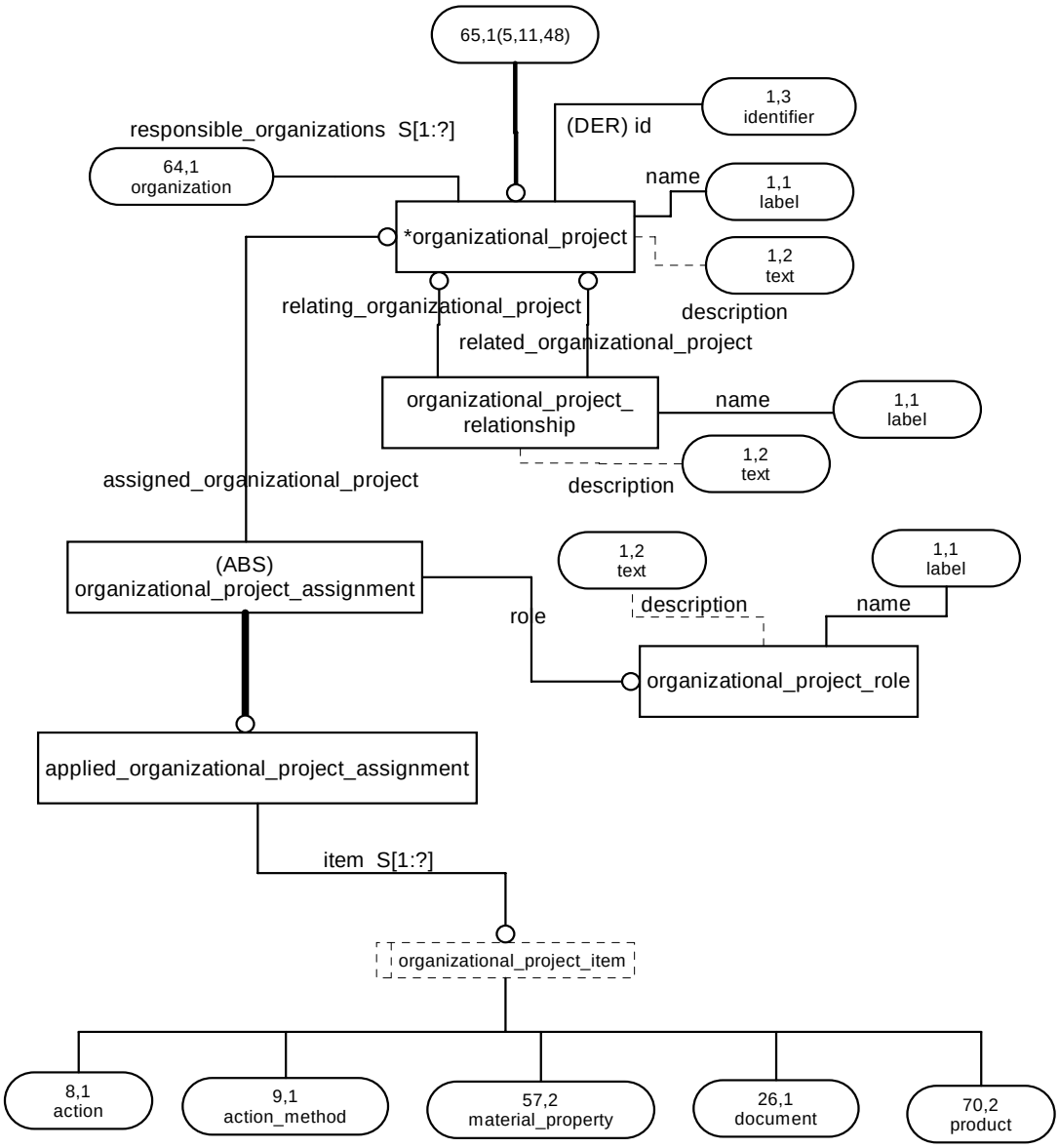


Figure H.65 — AIM EXPRESS-G diagram: organizational_project (65 of 91)

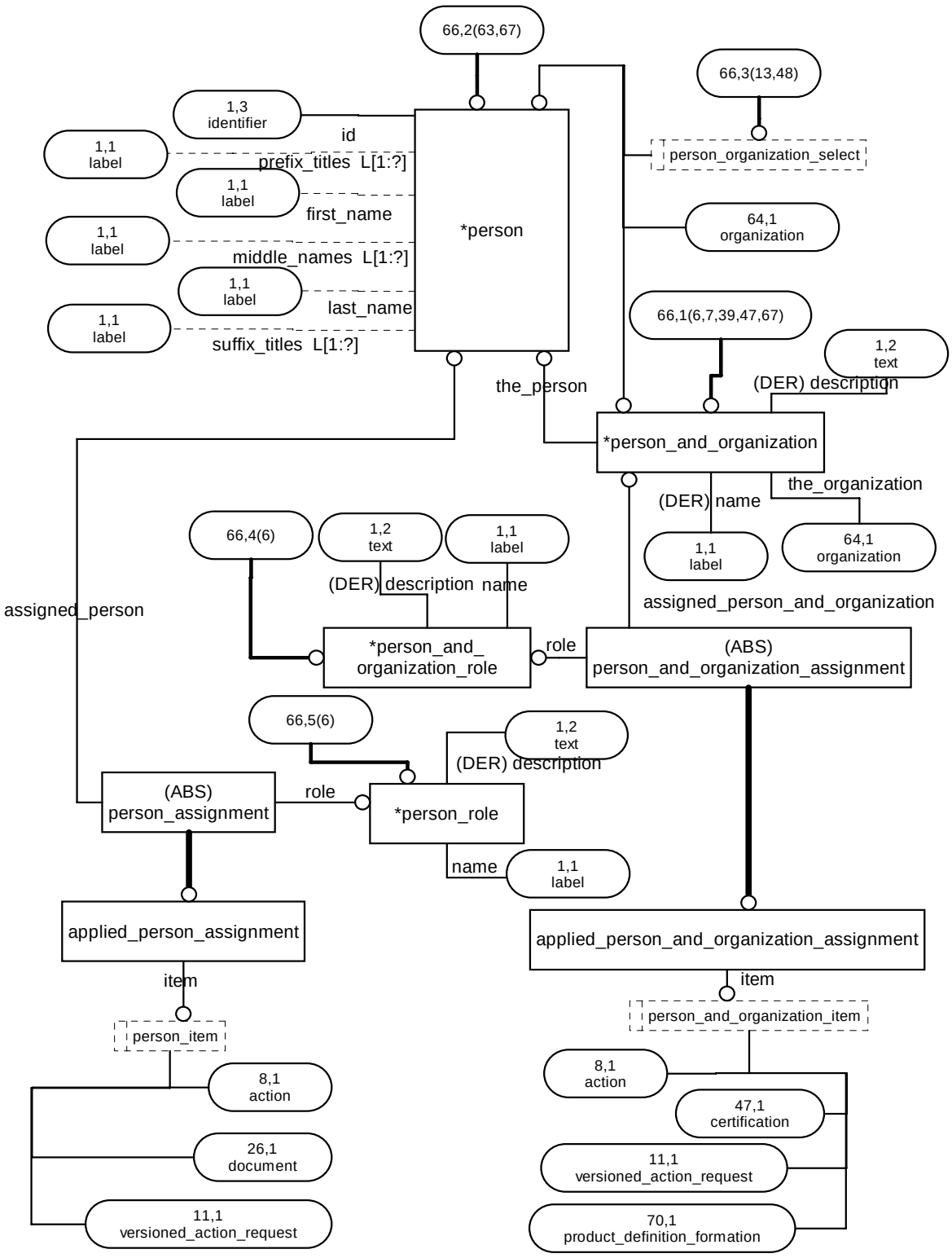


Figure H.66 — AIM EXPRESS-G diagram: person (66 of 91)

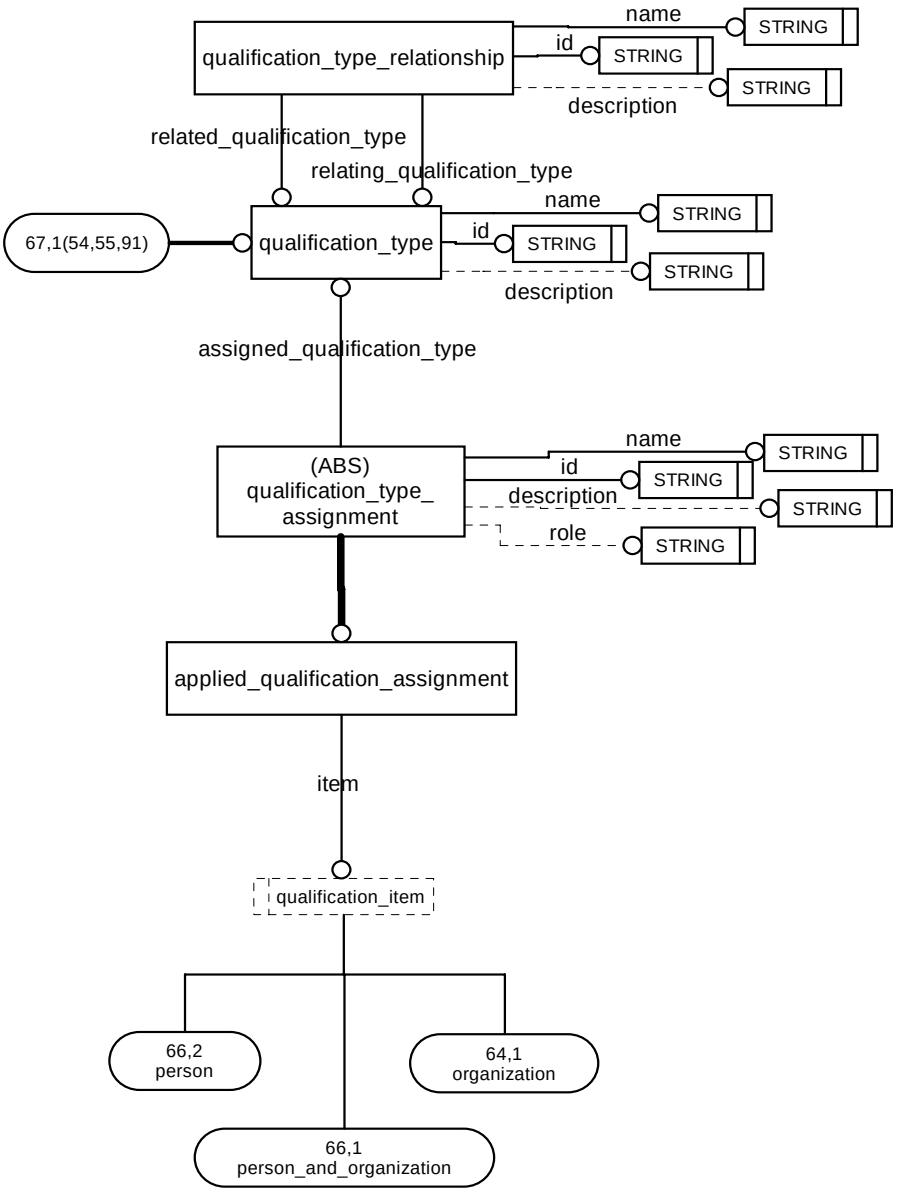


Figure H.67 — AIM EXPRESS-G diagram: qualification_type (67 of 91)

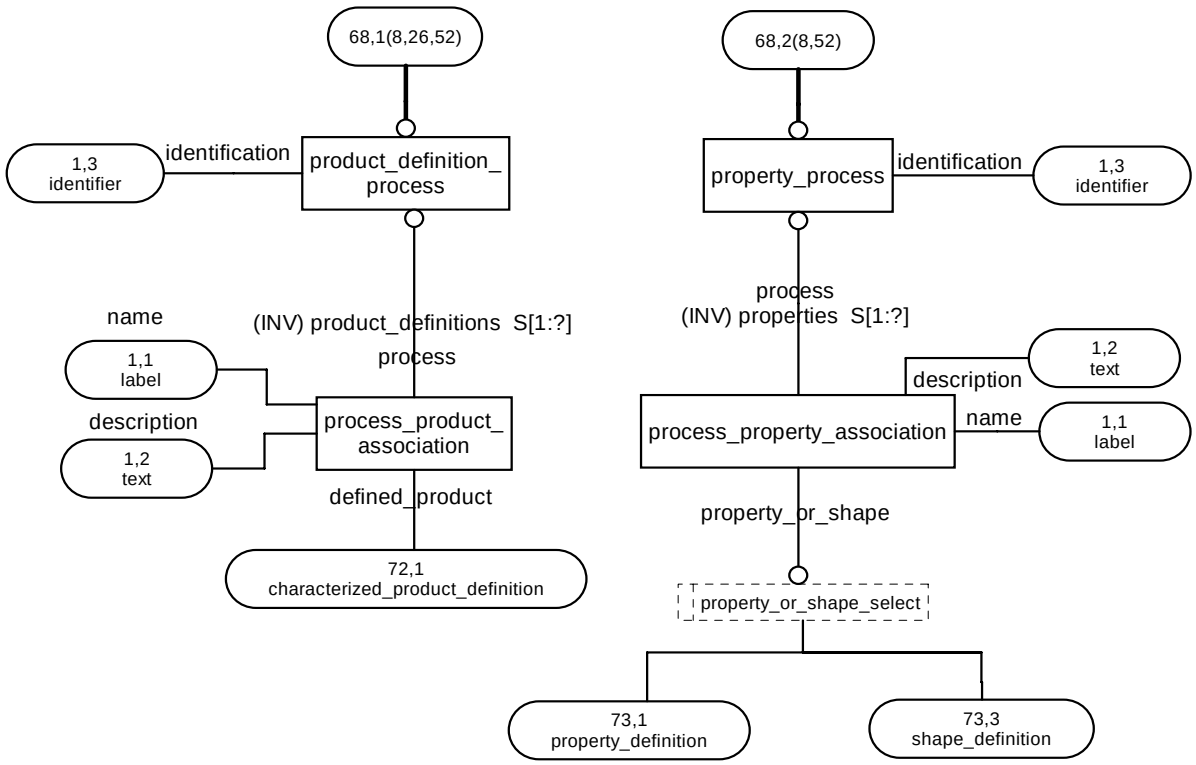


Figure H.68 — AIM EXPRESS-G diagram: property_process (68 of 91)

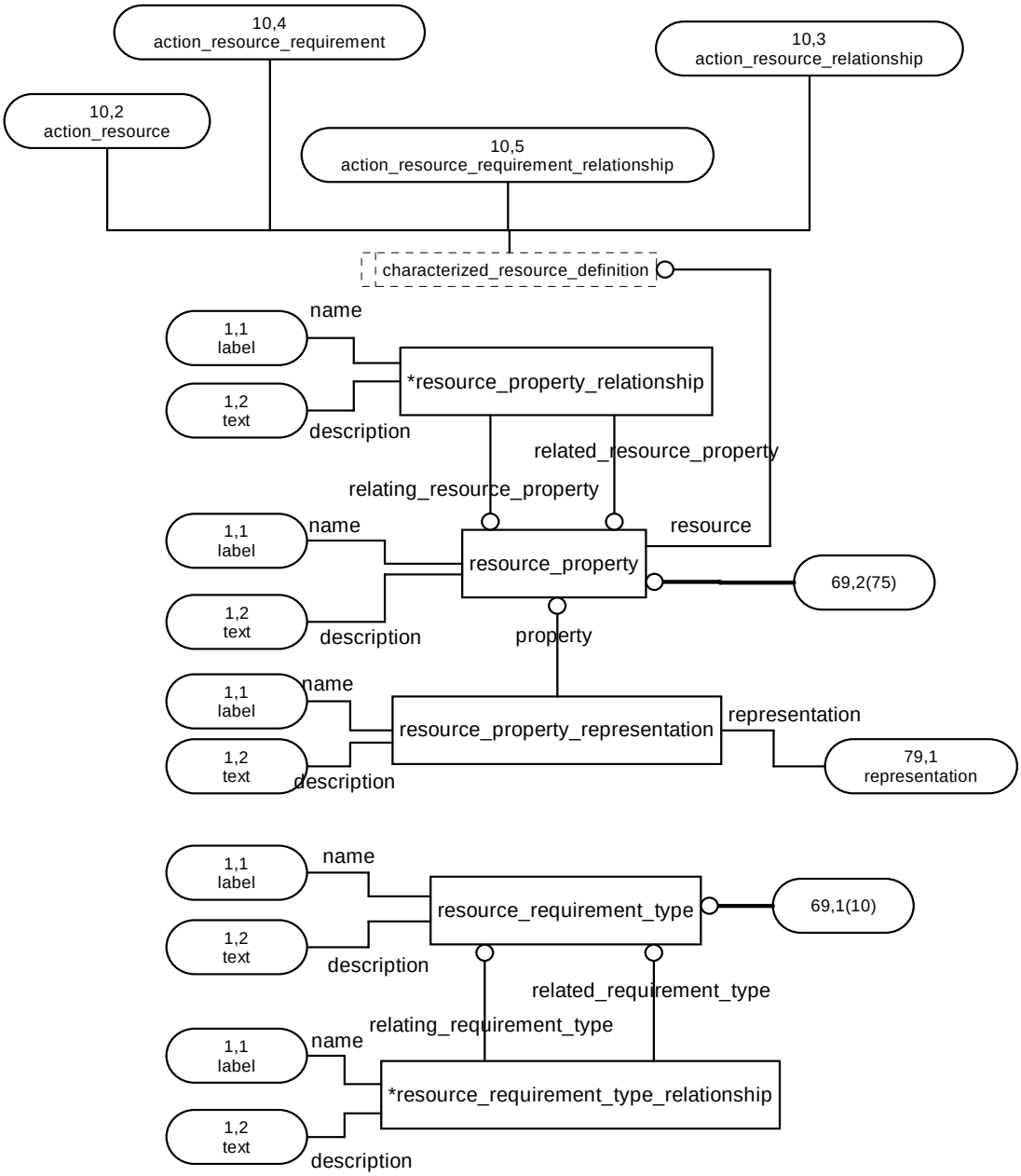


Figure H.69 — AIM EXPRESS-G diagram: resource_property (69 of 91)

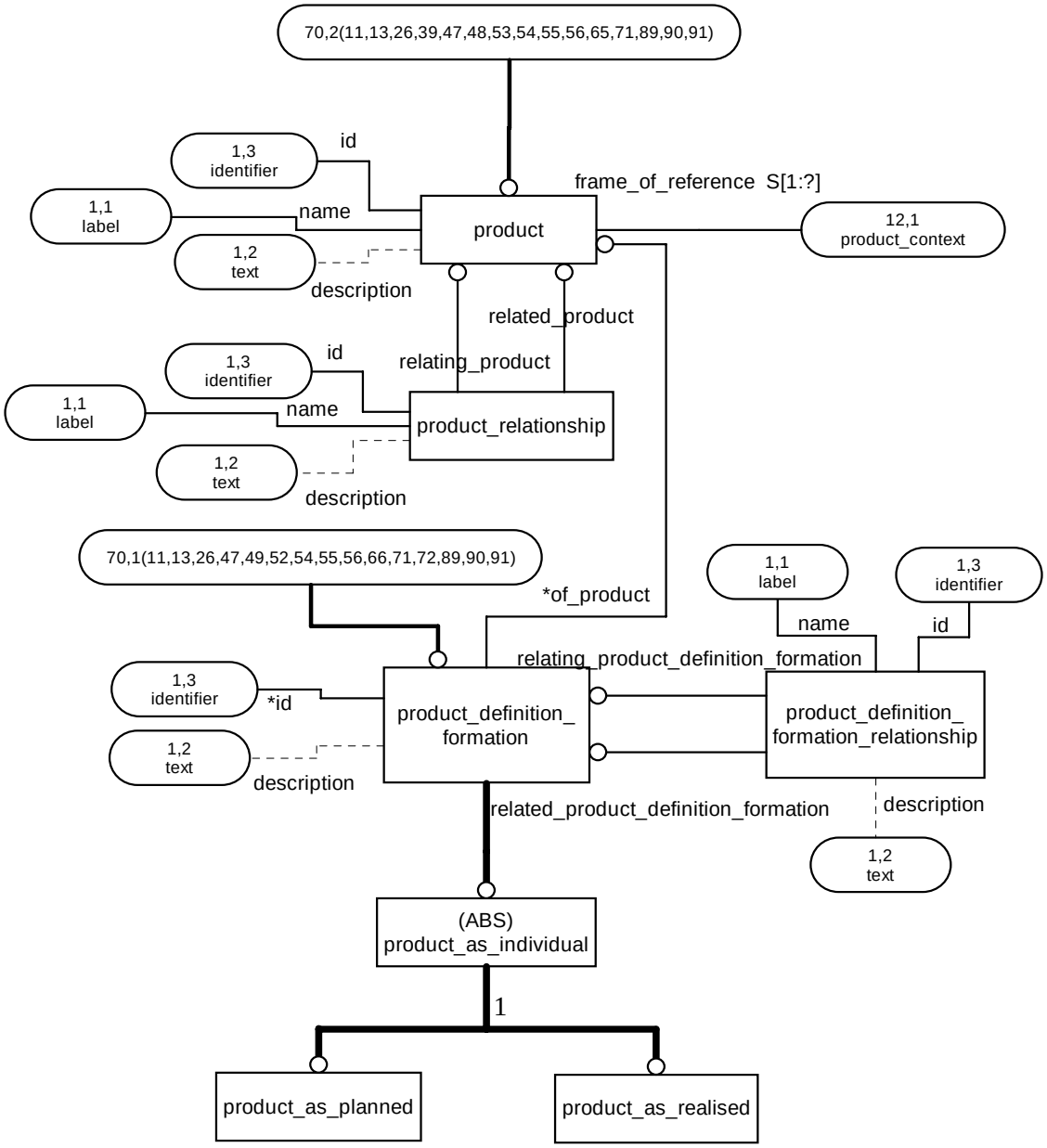
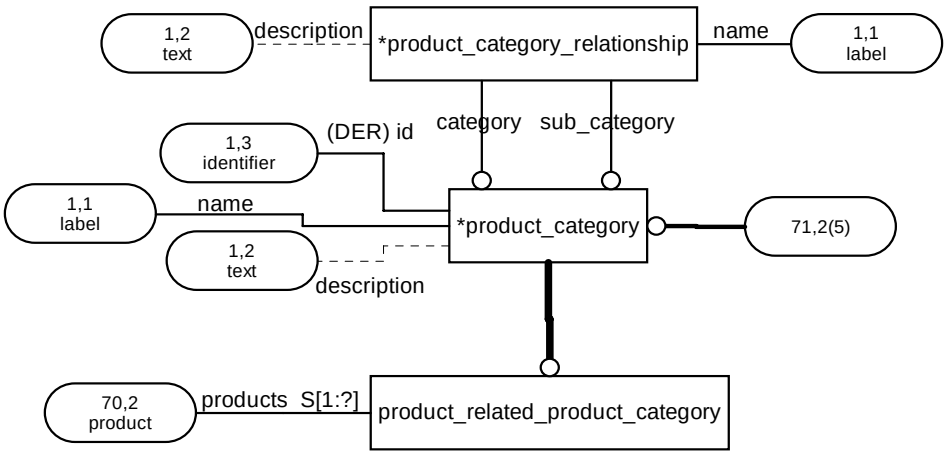


Figure H.70 — AIM EXPRESS-G diagram: product (70 of 91)



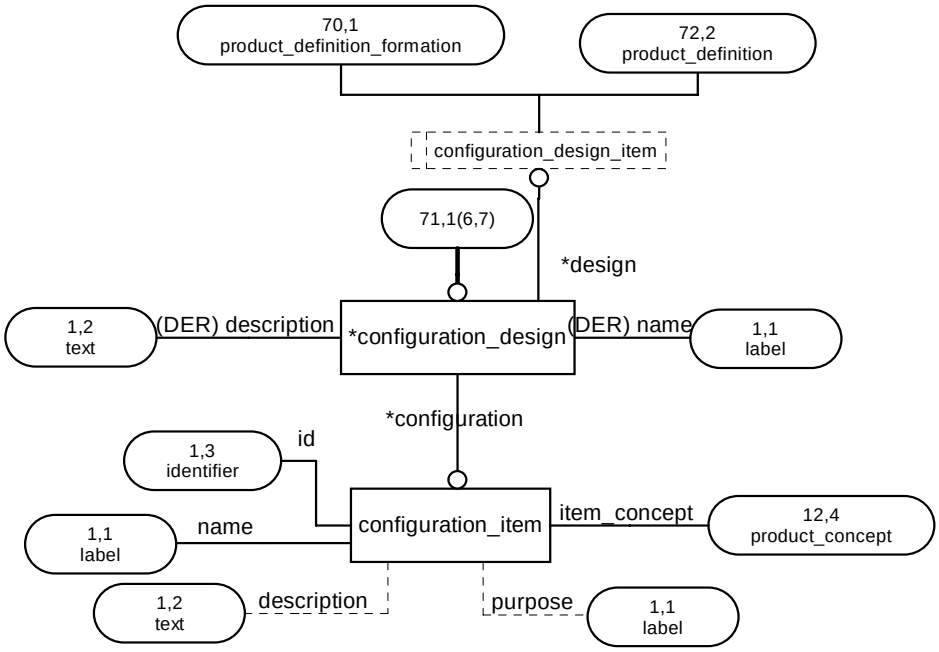


Figure H.71 — AIM EXPRESS-G diagram: product_category and configuration_design (71 of 91)

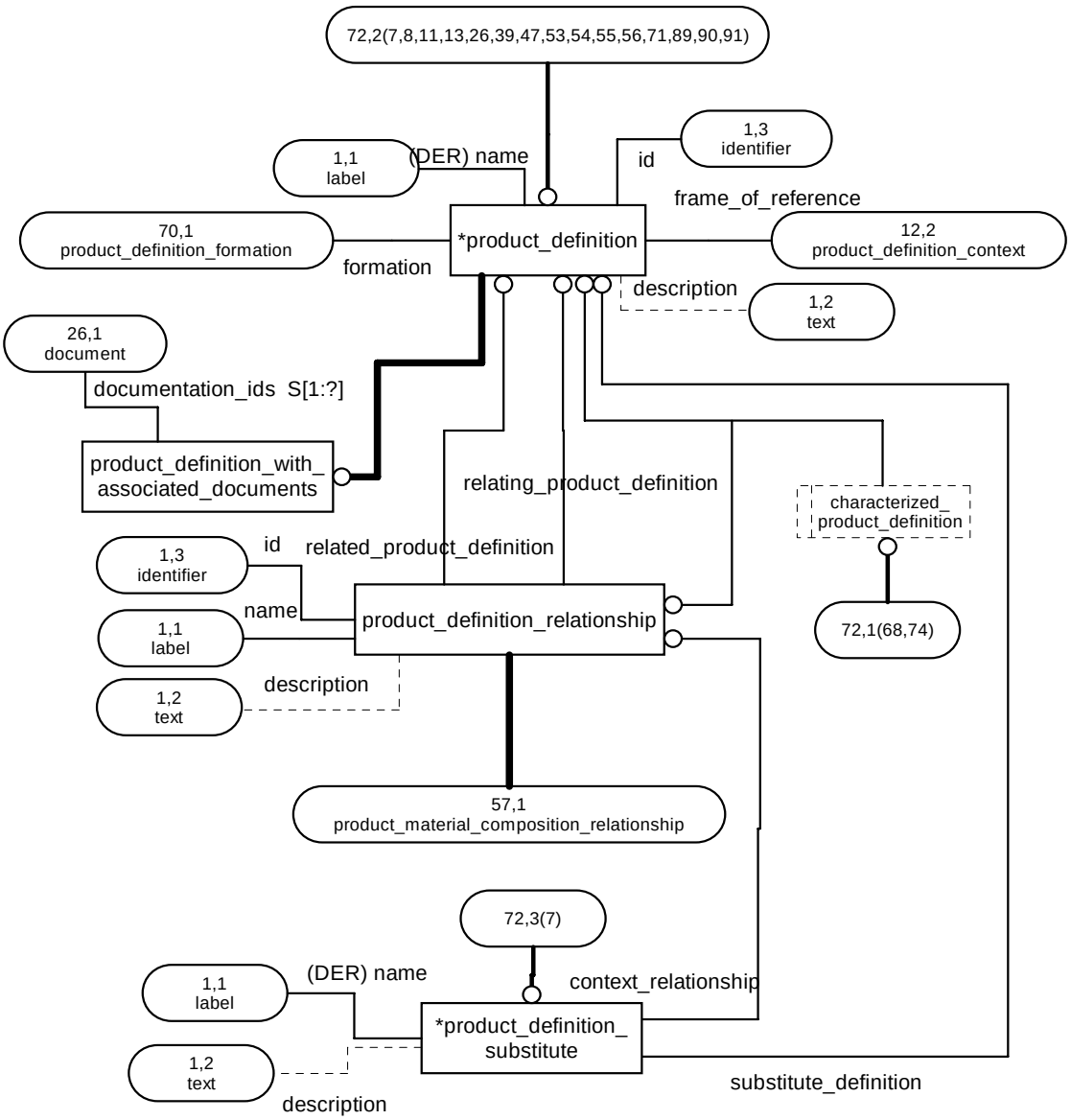


Figure H.72 — AIM EXPRESS-G diagram: product_definition (72 of 91)

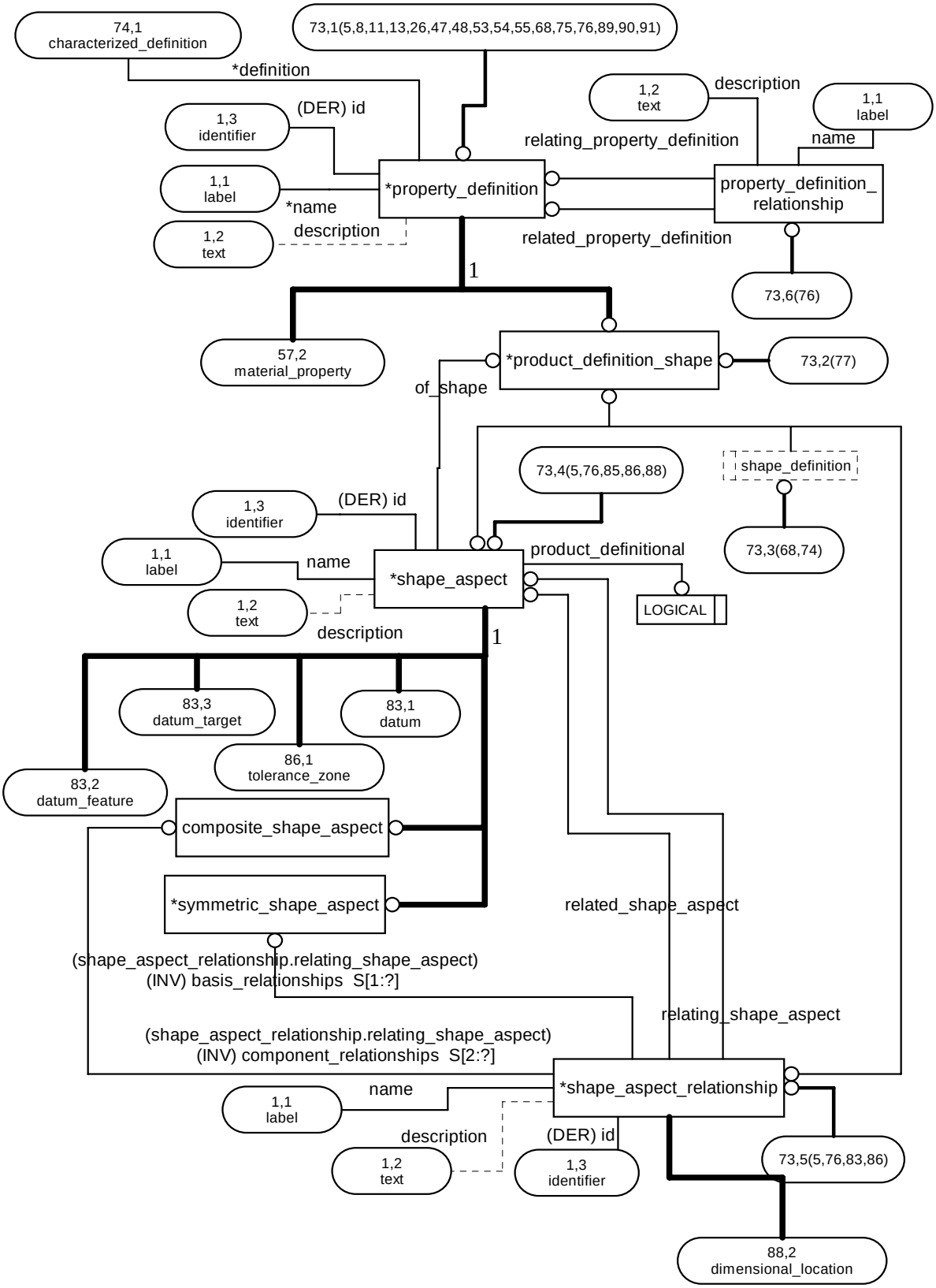


Figure H.73 — AIM EXPRESS-G diagram: property_definition and shape_aspect (73 of 91)

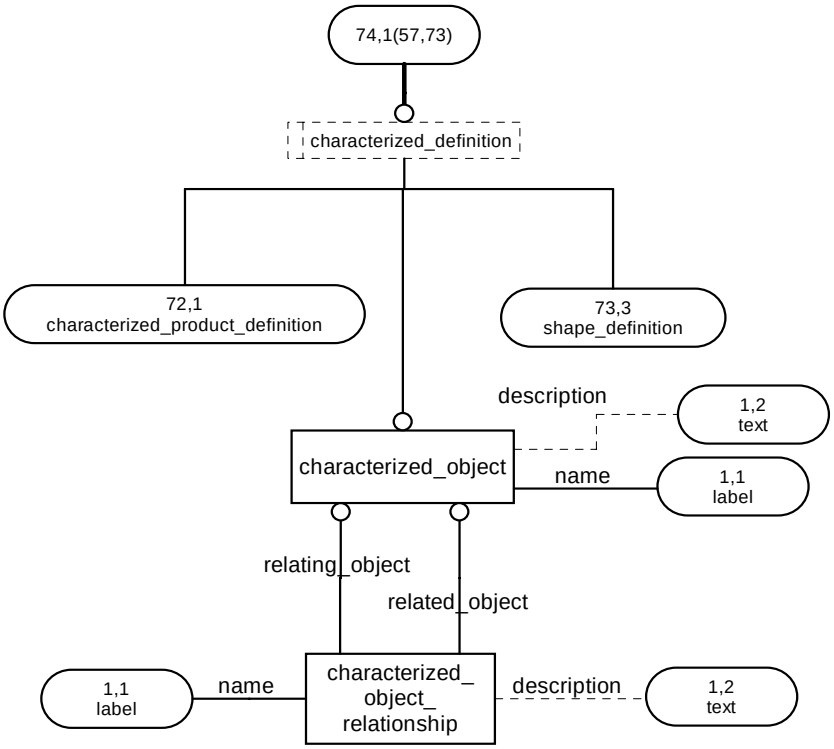


Figure H.74 — AIM EXPRESS-G diagram: characterized_object (74 of 91)

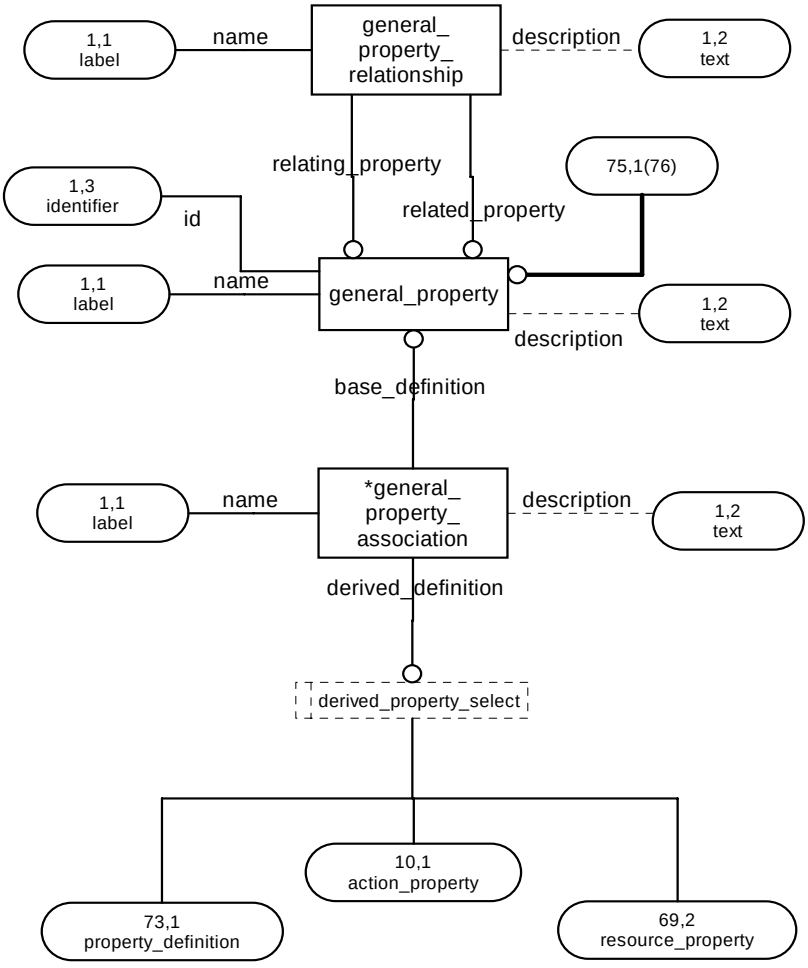


Figure H.75 — AIM EXPRESS-G diagram: general_property (75 of 91)

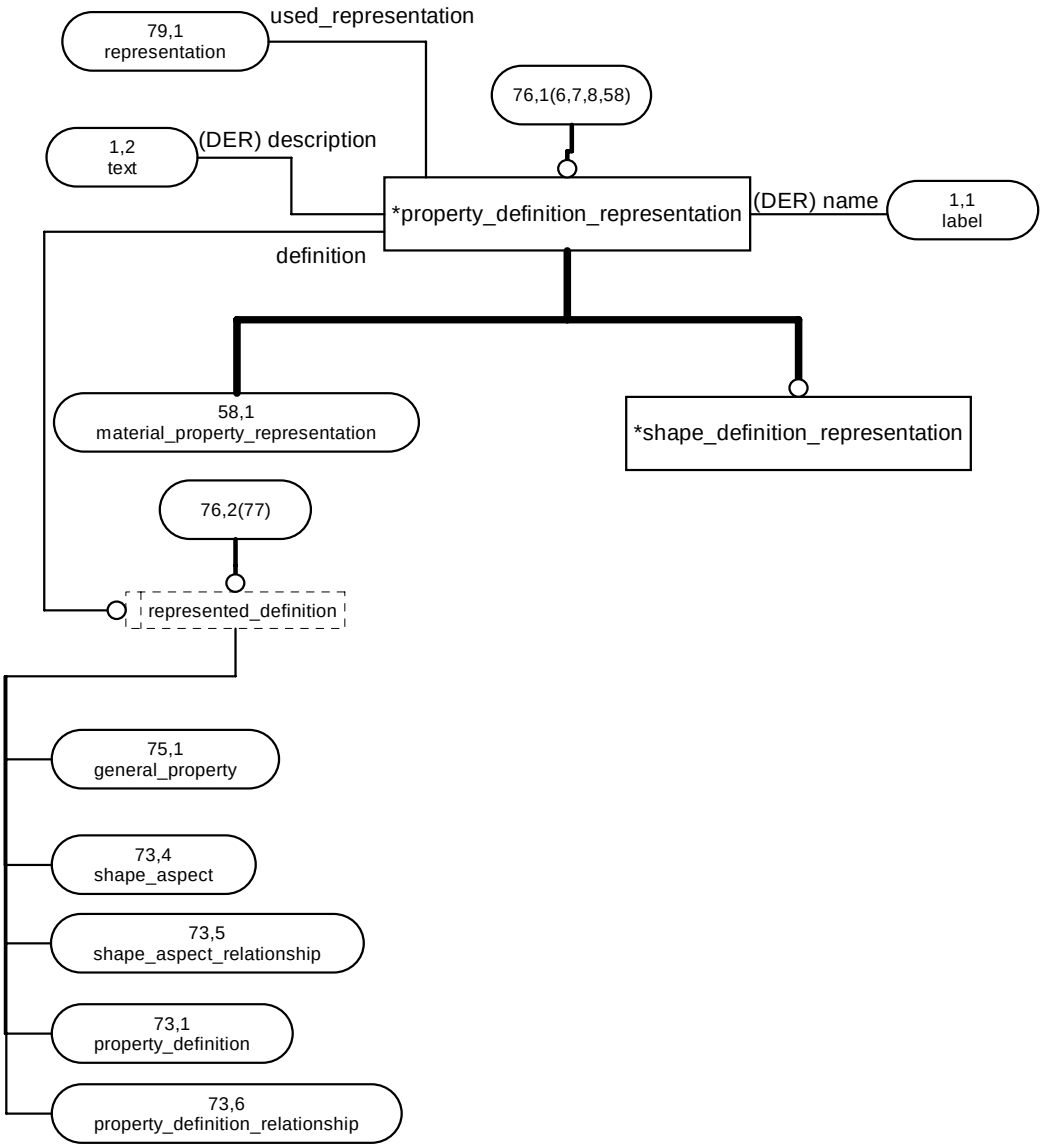


Figure H.76 — AIM EXPRESS-G diagram: property_definition_representation (76 of 91)

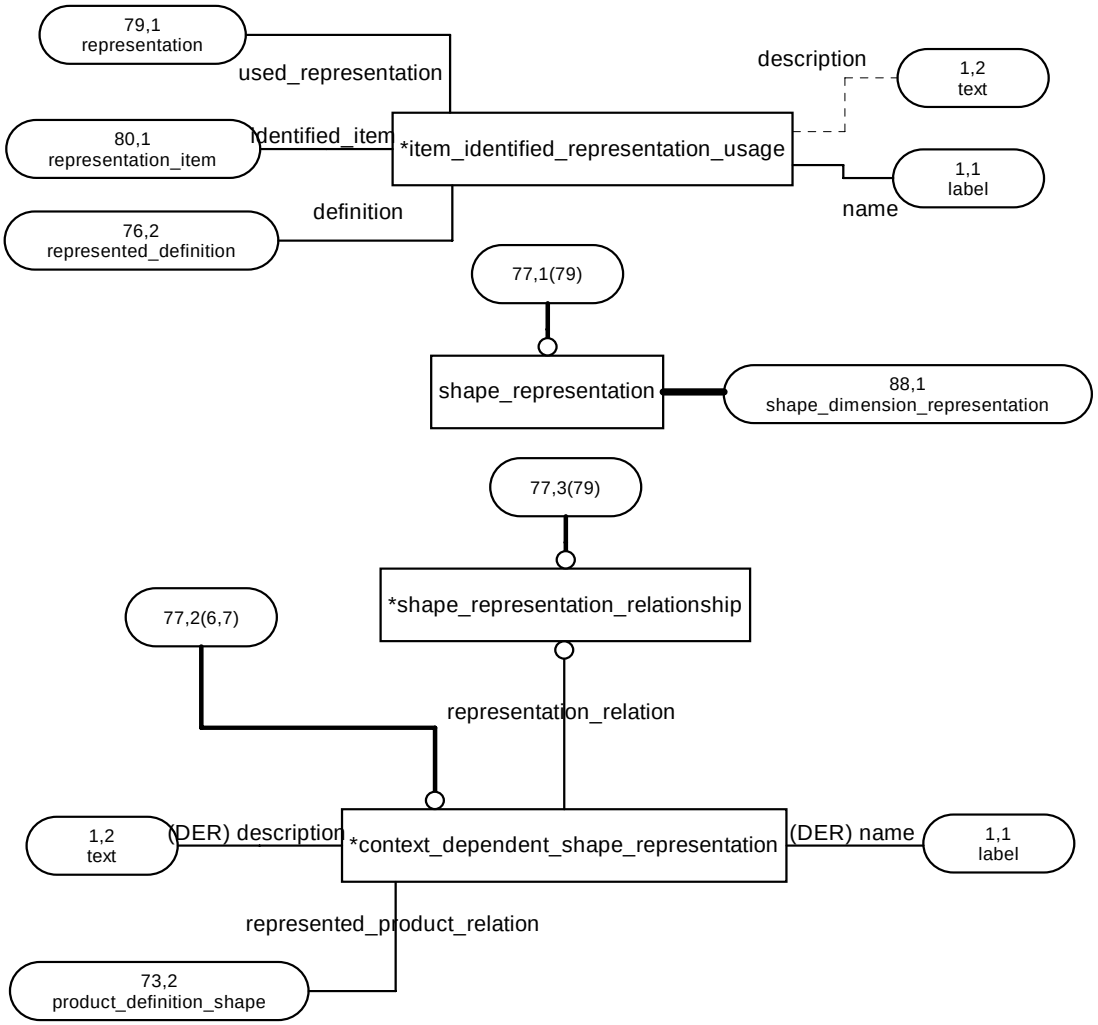


Figure H.77 — AIM EXPRESS-G diagram: shape_representation and item_identified_representation_usage (77 of 91)

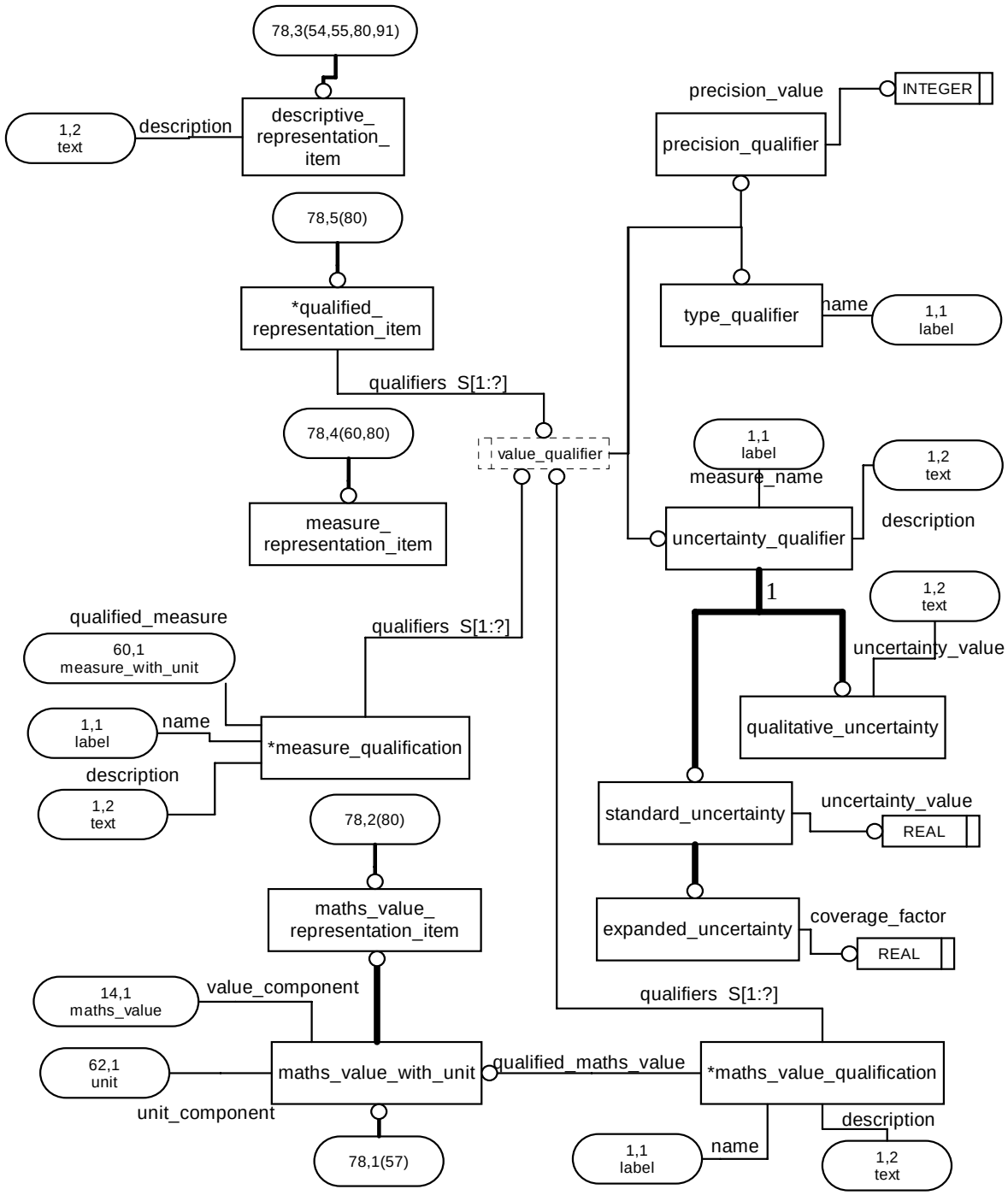


Figure H.78 — AIM EXPRESS-G diagram: measure_qualification (78 of 91)

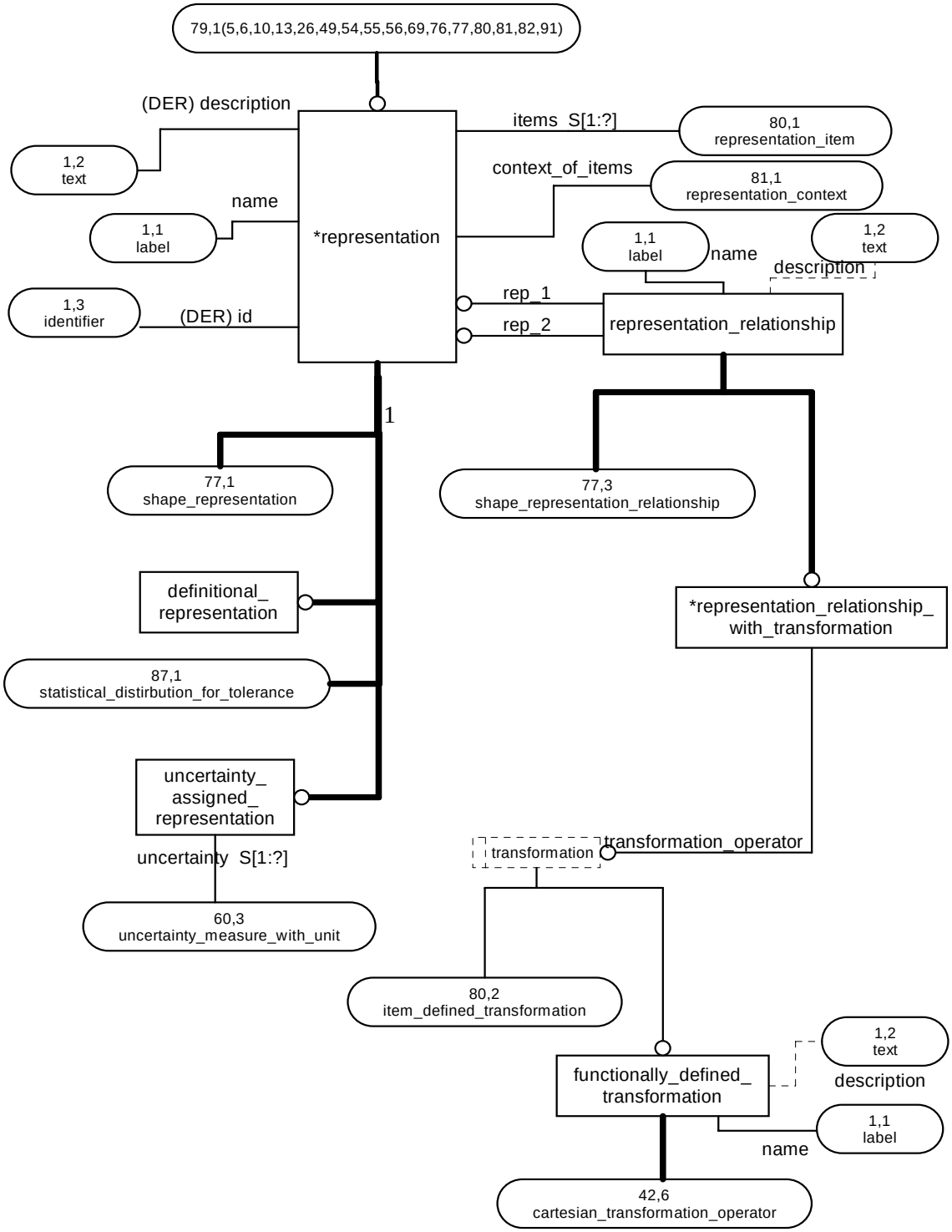


Figure H.79 — AIM EXPRESS-G diagram: representation (79 of 91)

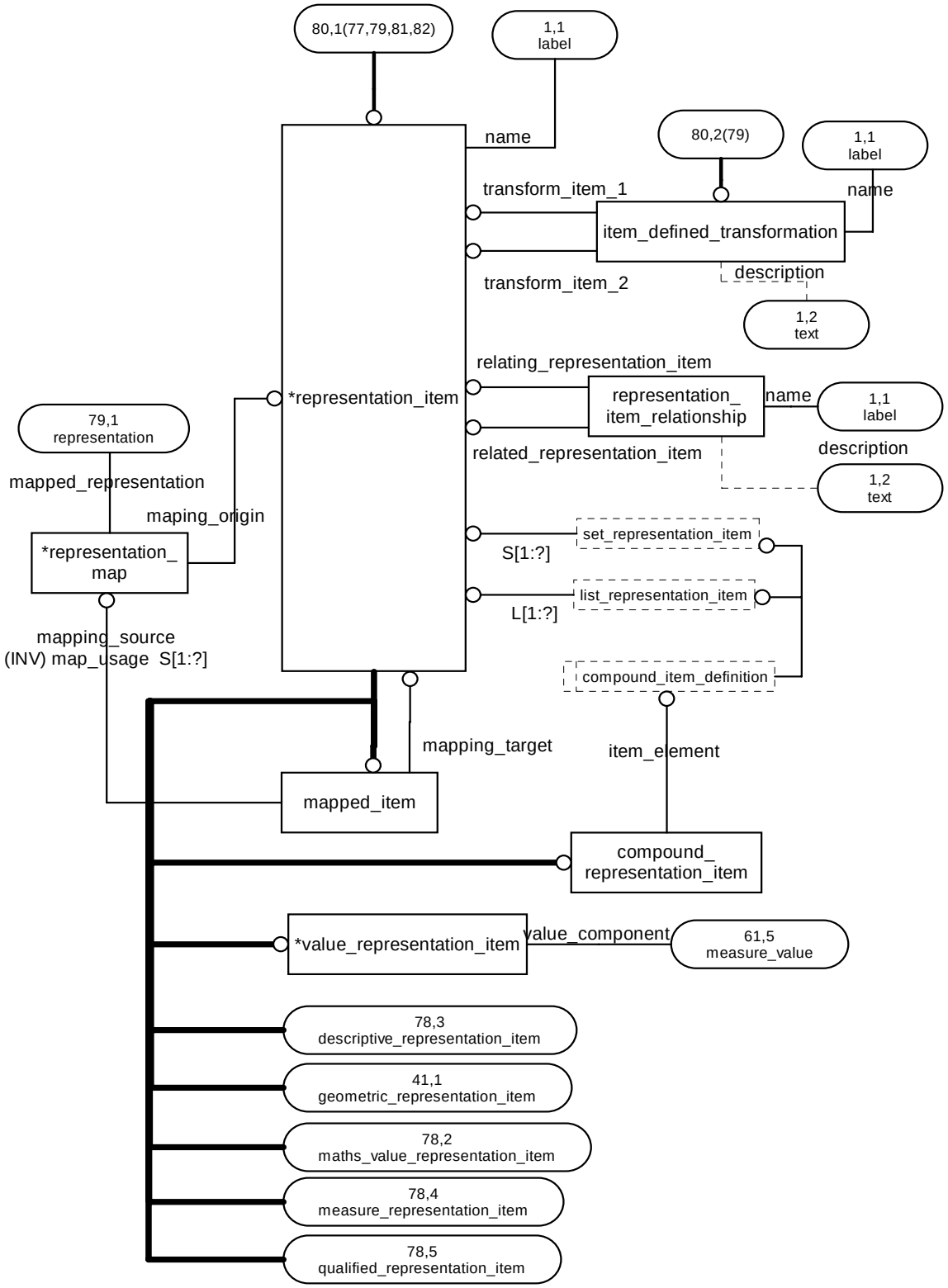


Figure H.80 — AIM EXPRESS-G diagram: representation_item (80 of 91)

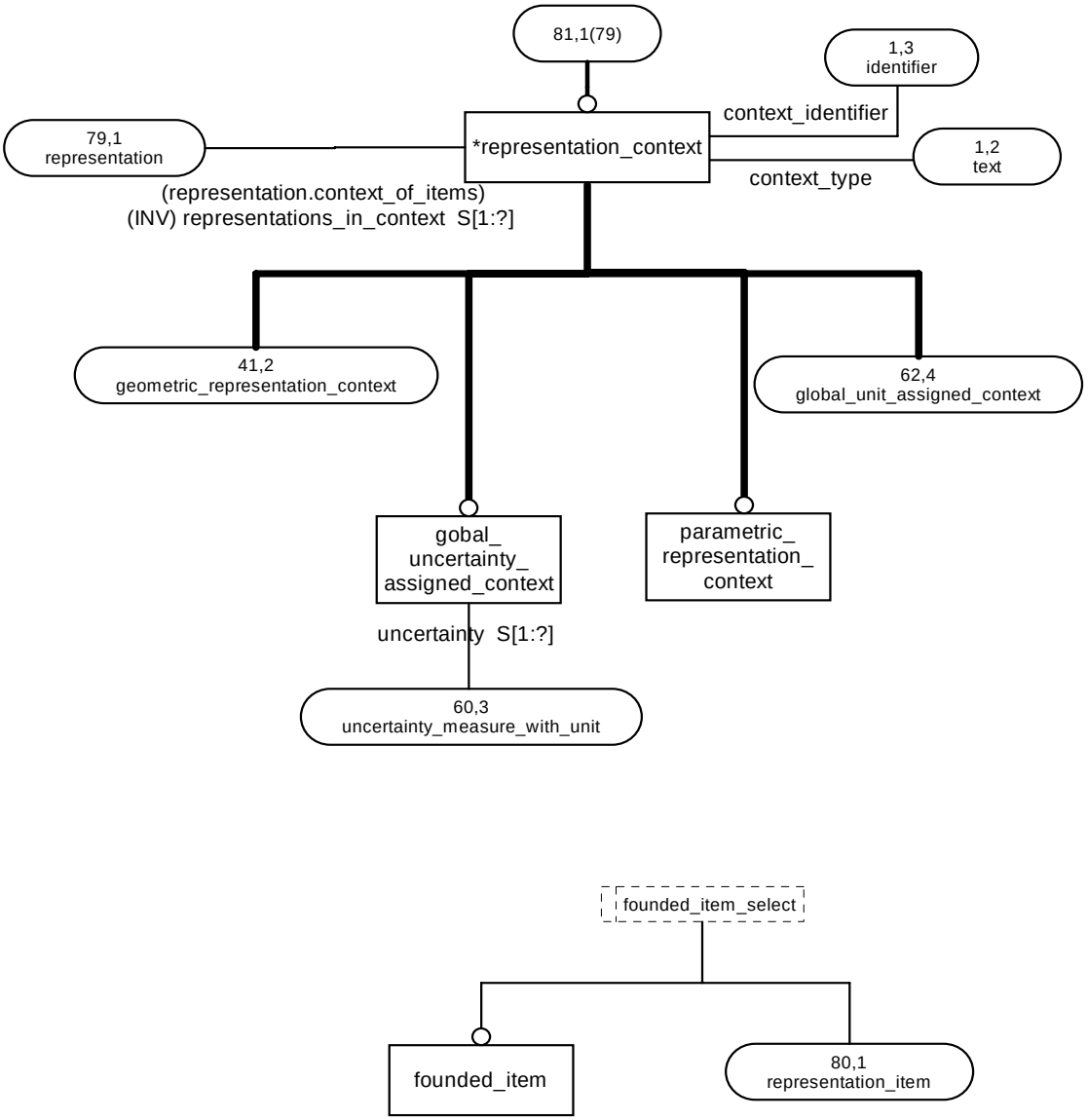


Figure H.81 — AIM EXPRESS-G diagram: representation_context (81 of 91)

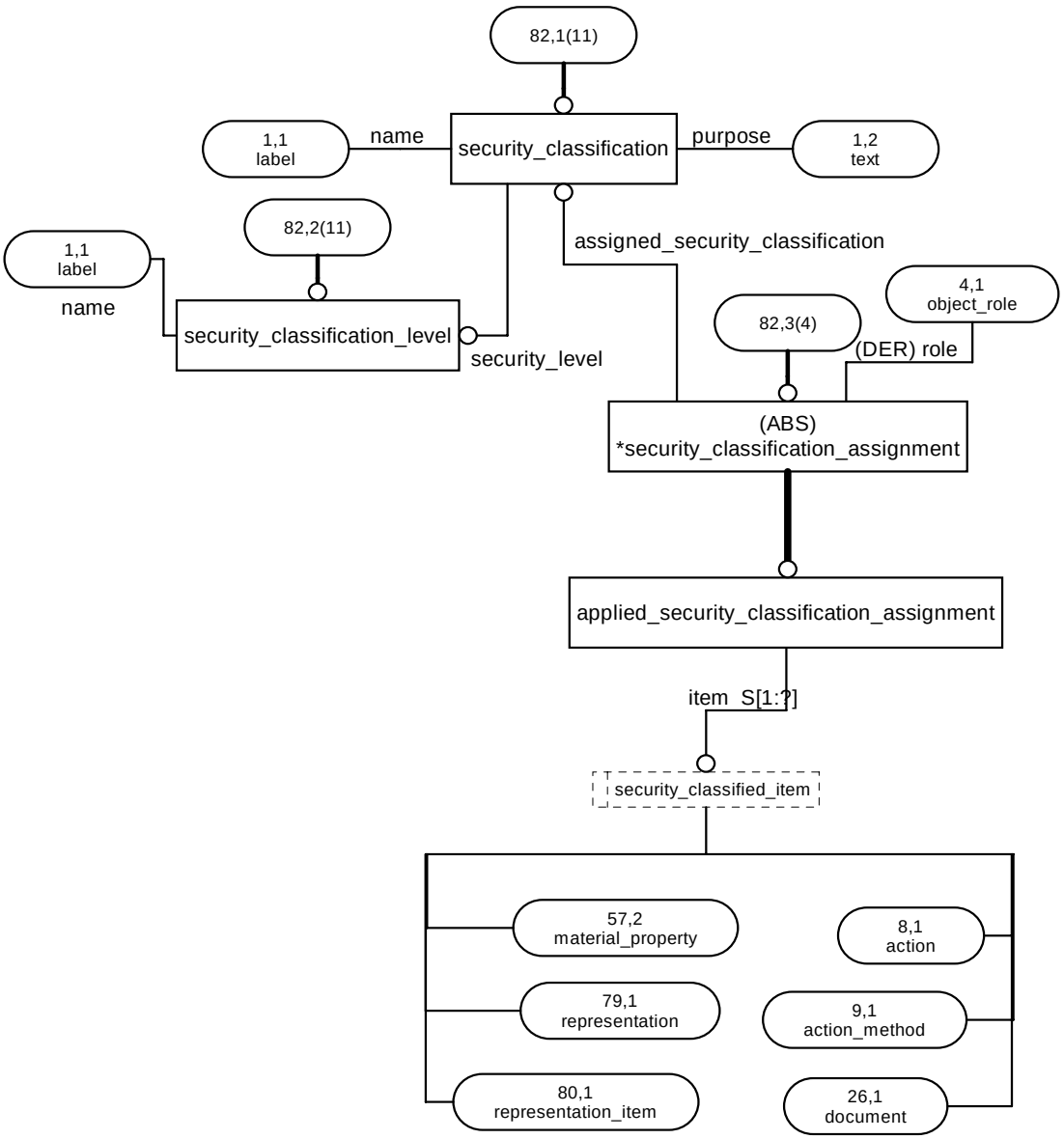


Figure H.82 — AIM EXPRESS-G diagram: security_classification (82 of 91)

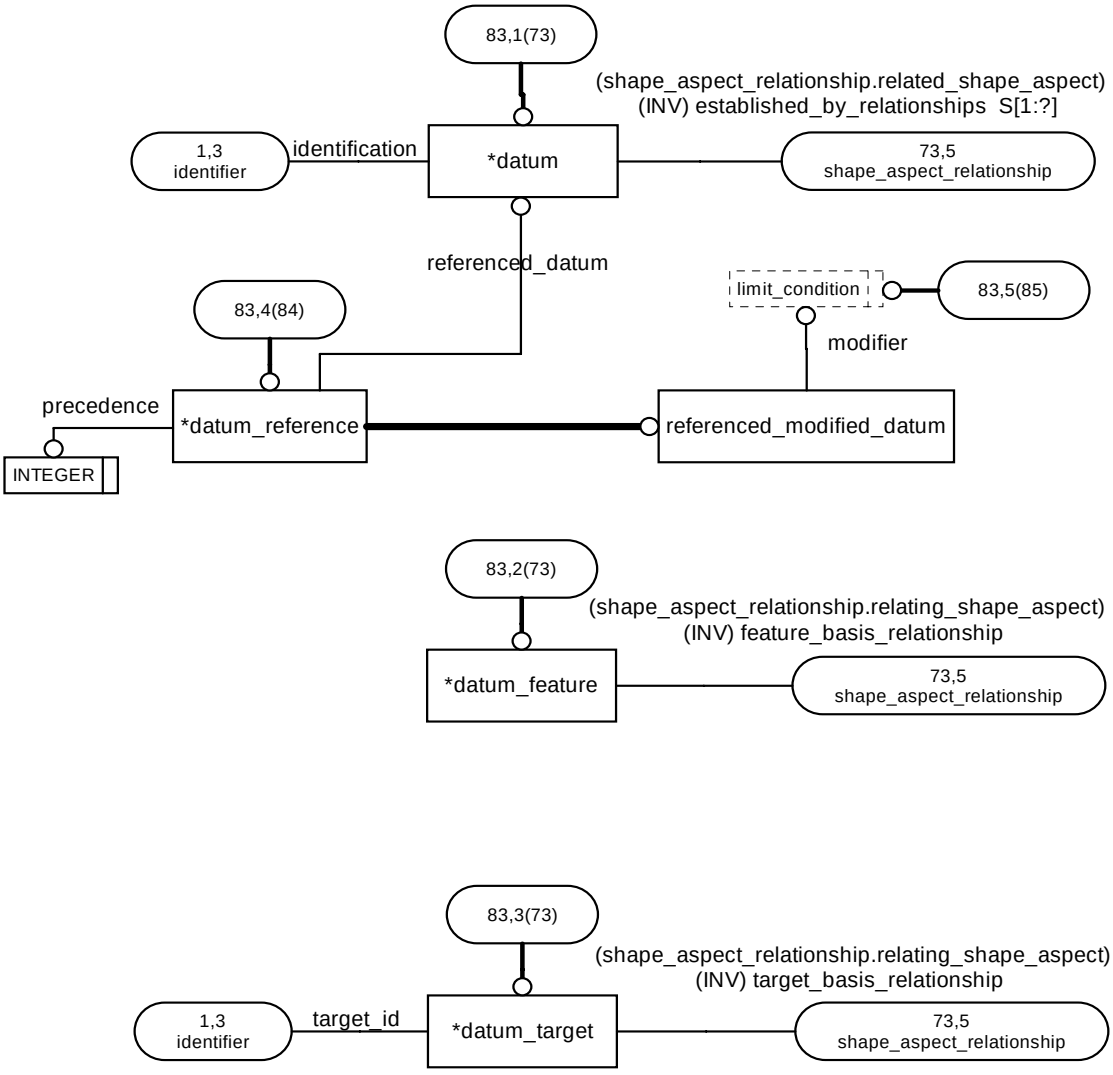


Figure H.83 — AIM EXPRESS-G diagram: tolerance_datum (83 of 91)

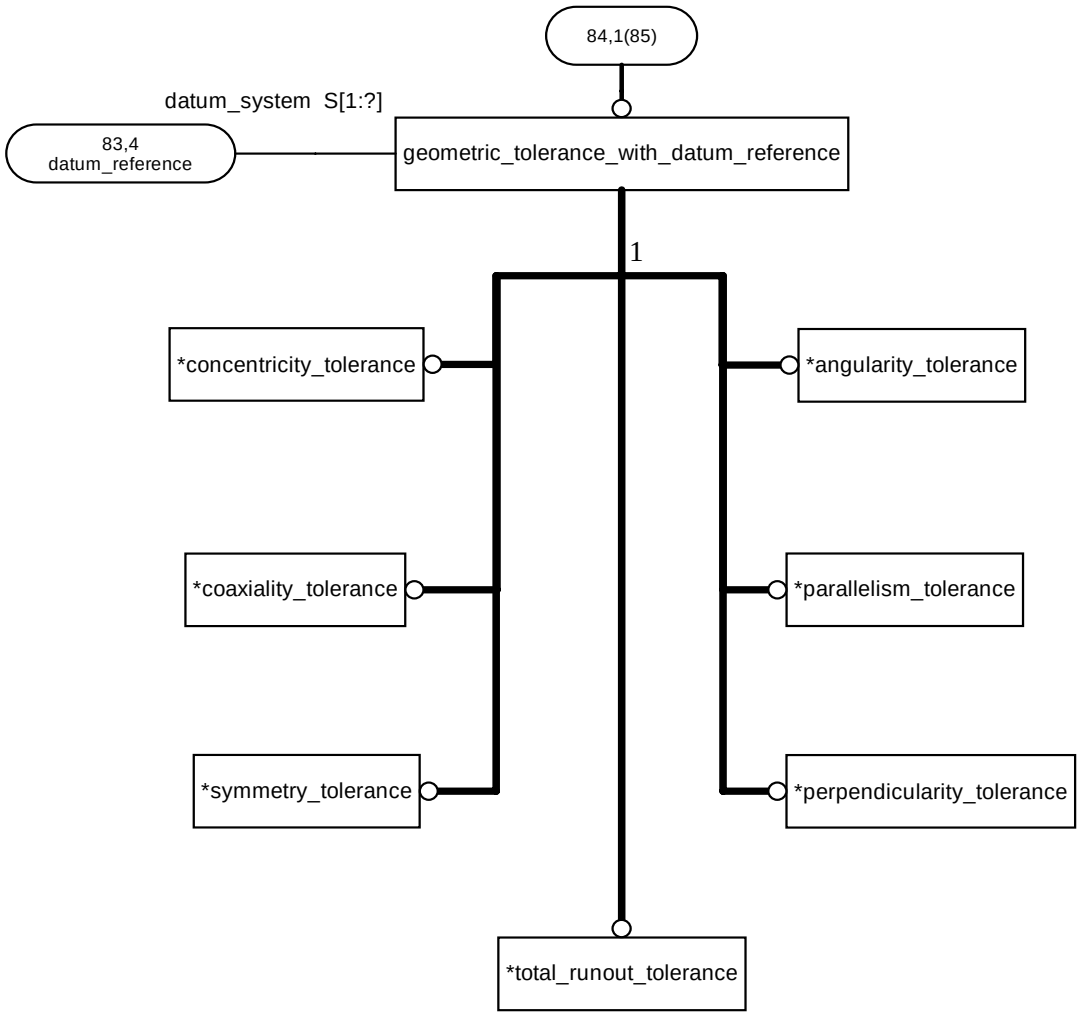


Figure H.84 — AIM EXPRESS-G diagram: geometric_tolerance_with_datum (84 of 91)

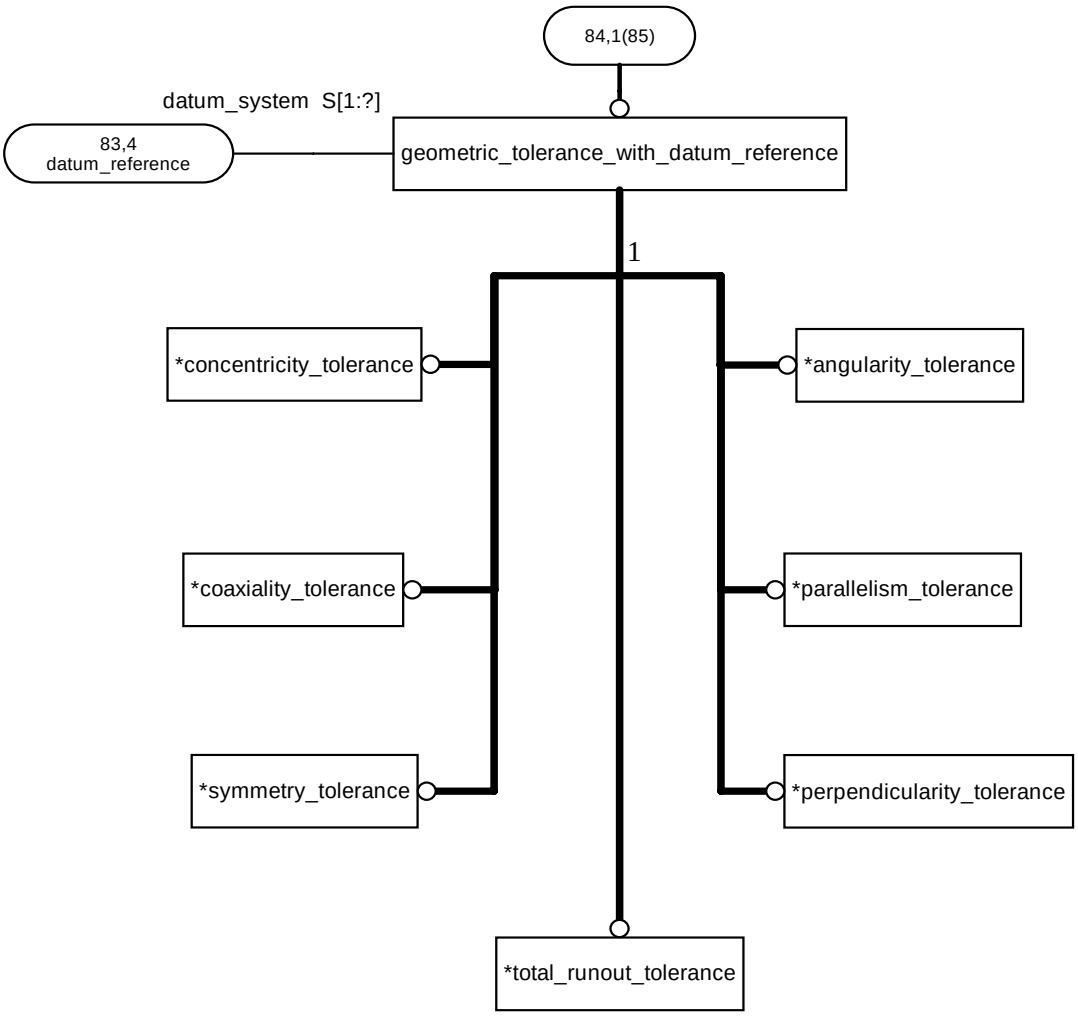


Figure H.85 — AIM EXPRESS-G diagram: geometric_tolerance (85 of 91)

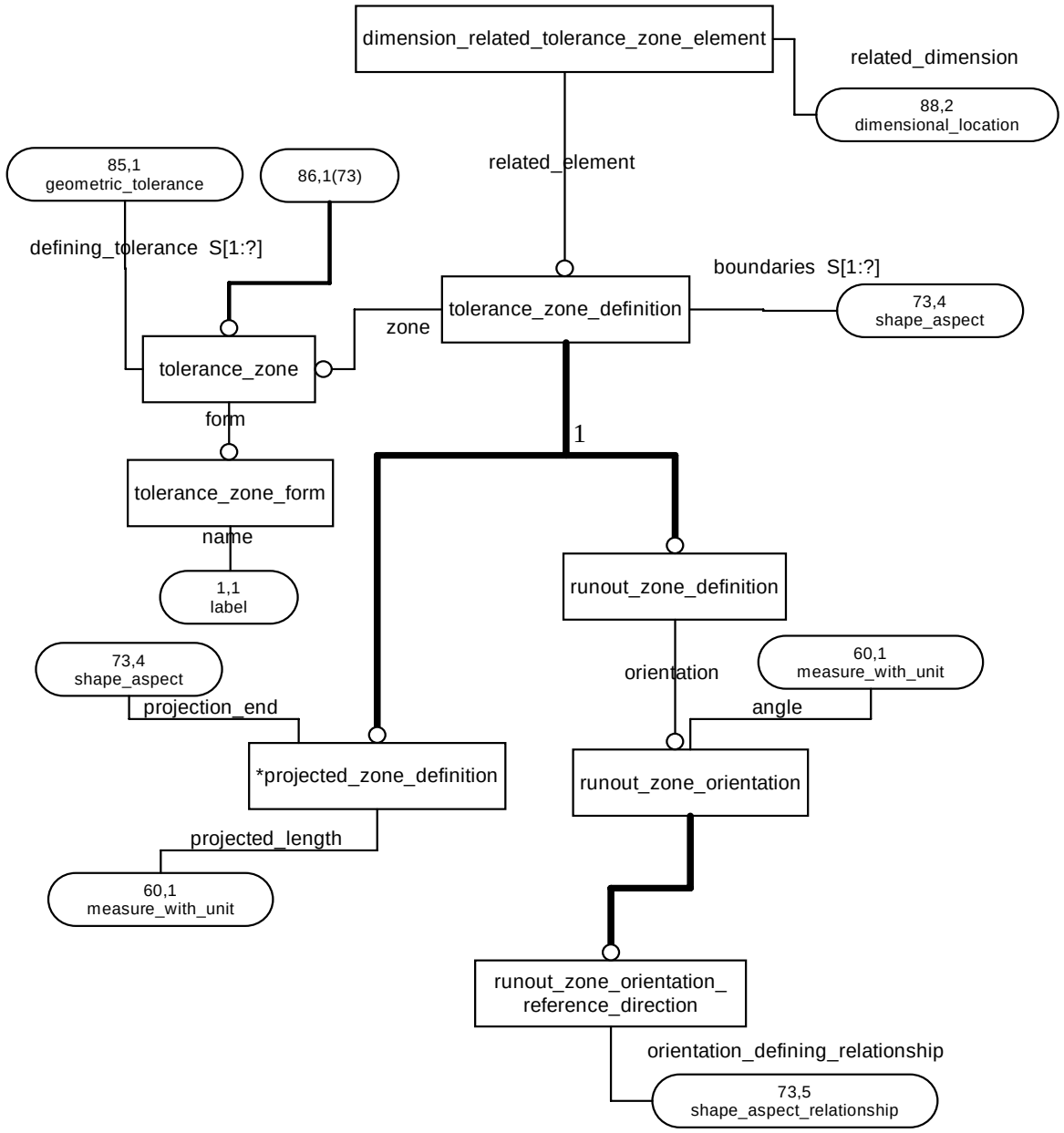


Figure H.86 — AIM EXPRESS-G diagram: tolerance_zone_definition (86 of 91)

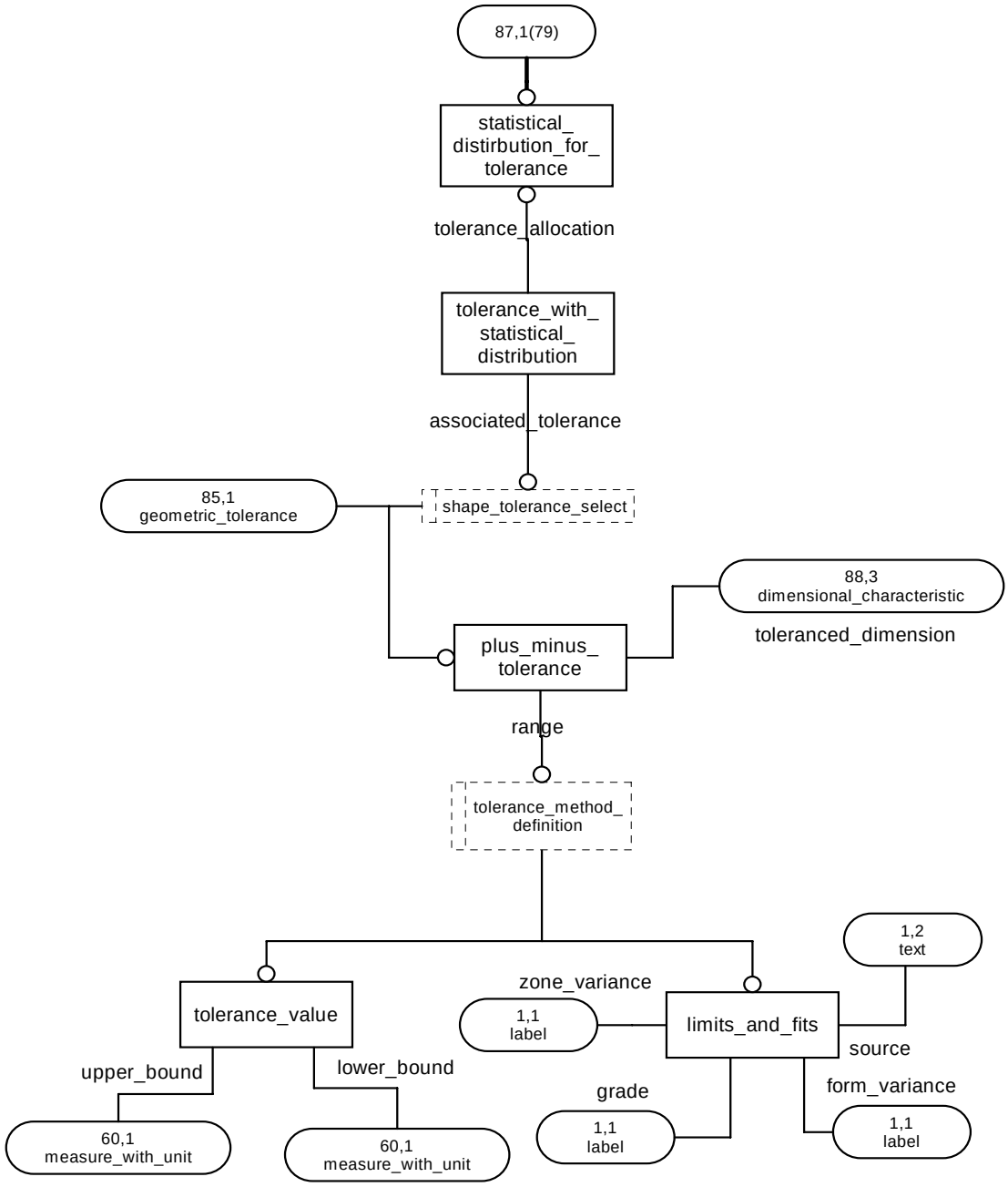


Figure H.87 — AIM EXPRESS-G diagram: tolerance_value (87 of 91)

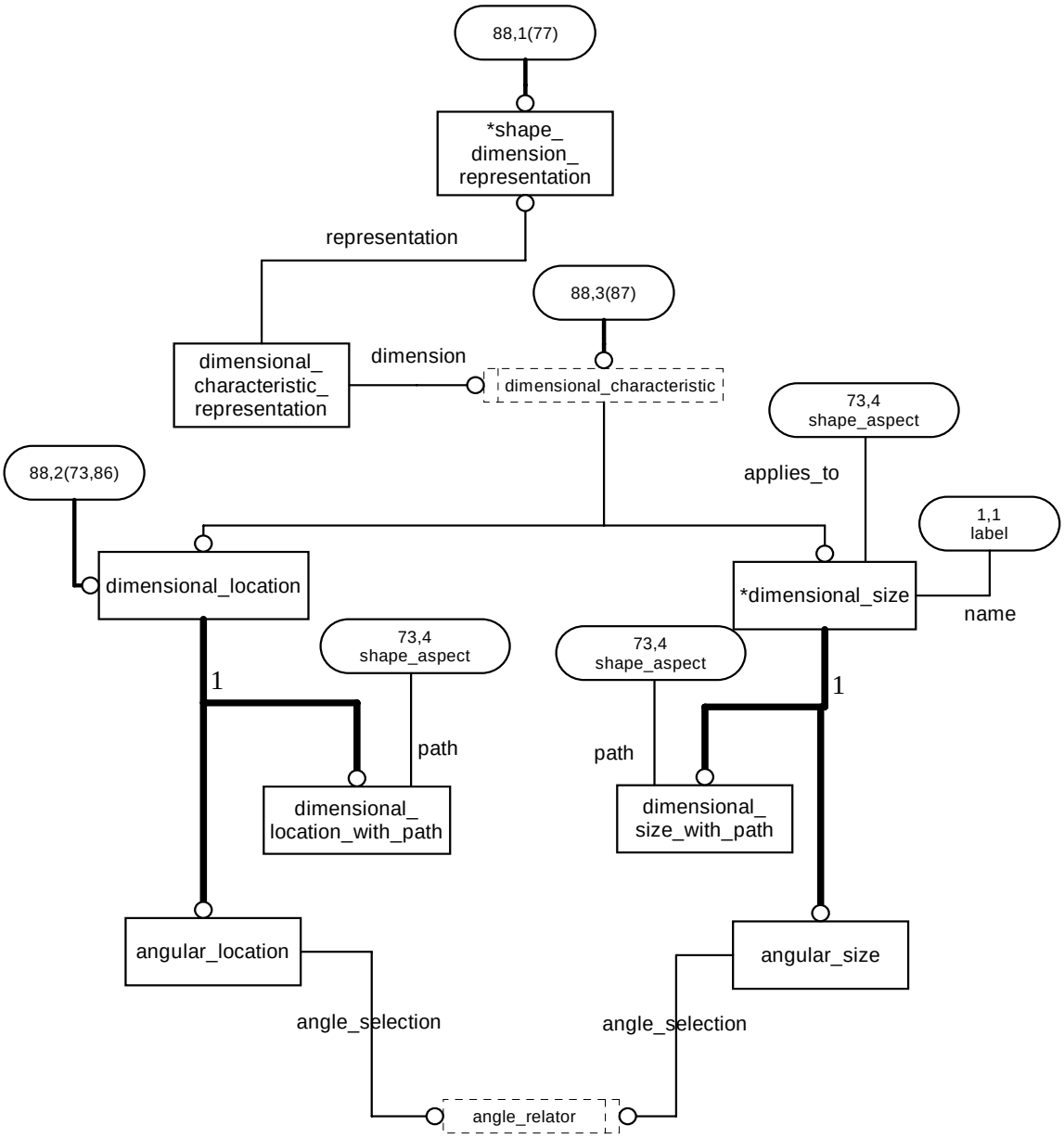


Figure H.88 — AIM EXPRESS-G diagram: dimensional_location and dimensional_size (88 of 91)

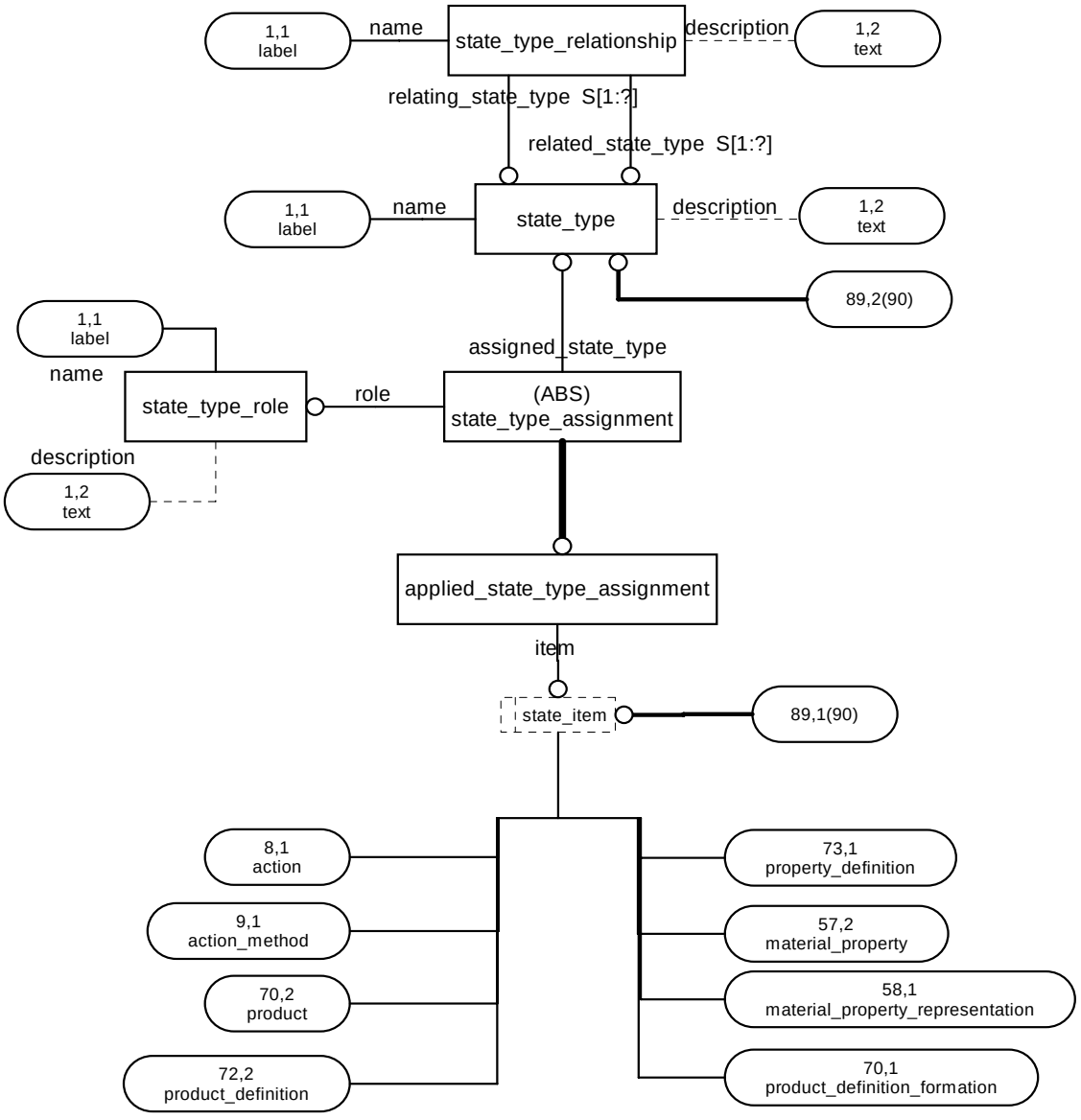


Figure H.89 — AIM EXPRESS-G diagram: state_type (89 of 91)

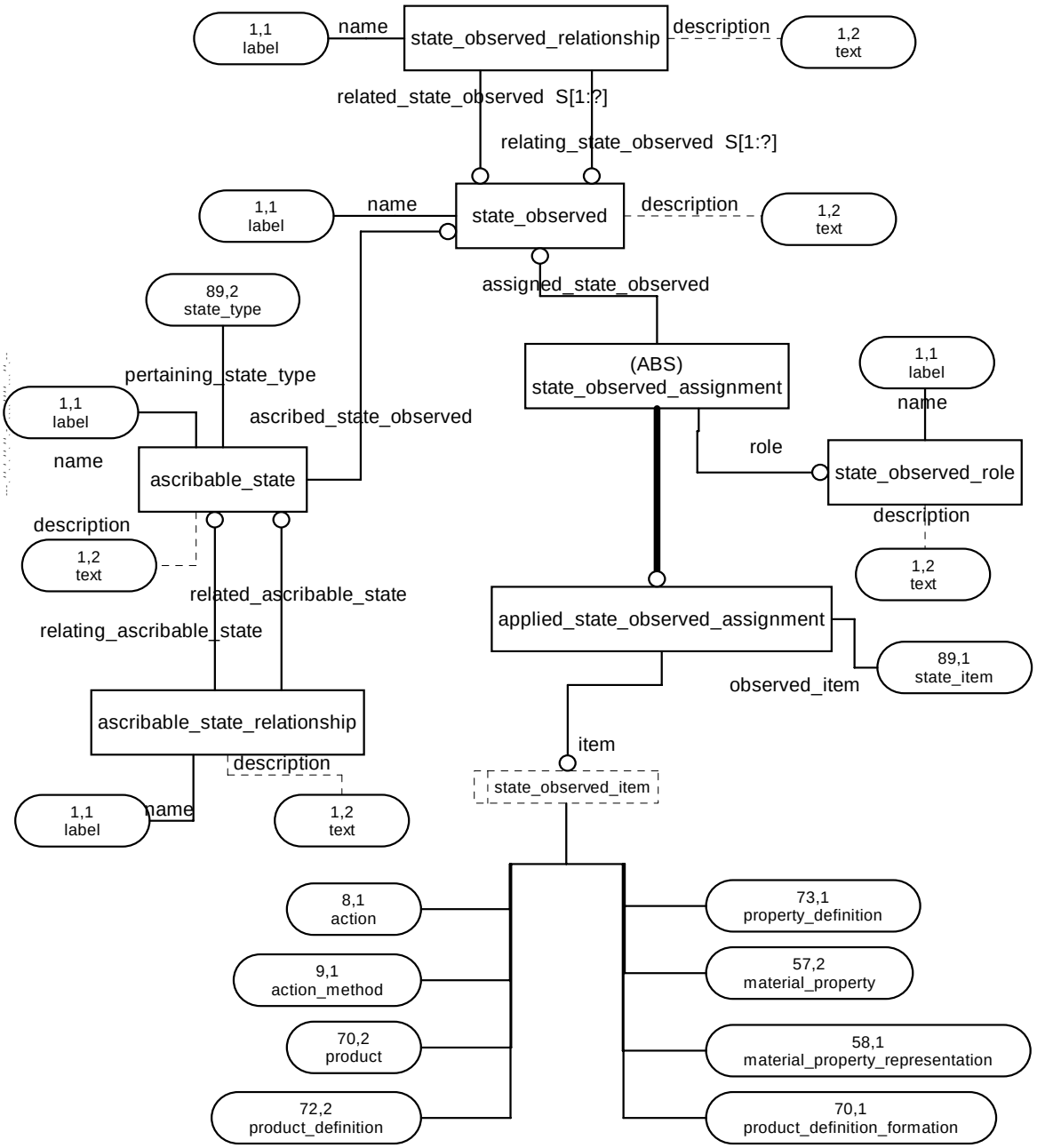


Figure H.90 — AIM EXPRESS-G diagram: state_observed (90 of 91)

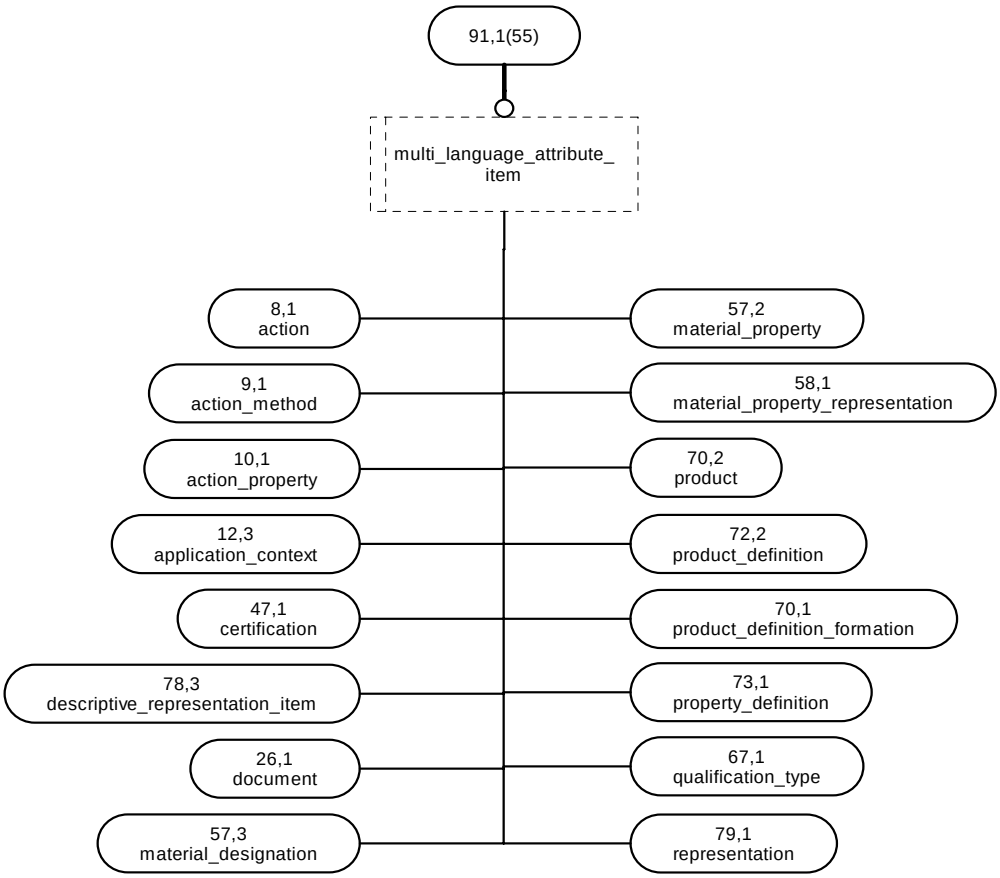


Figure H.91 — AIM EXPRESS-G diagram: multi_language_attribute_item (91 of 91)

Annex I
(informative)

Computer interpretable listings

This annex provides a listing of the complete EXPRESS shema specified in Annex A of this part of ISO 10303 without comments or other explanatory text. It also provides a listing of the EXPRESS entity names and corresponding short names as specified in Annex B of this part of ISO 10303. The content of of this annex is available in computer-interpretable form and can be found at the following URLs:

— Short names: http://www.tc184-sc4.org/short_names/

— EXPRESS: http://www.tc184-sc4.org/express/

If there is any difficulty accessing these sites contact the ISO Central Secretariat or contact the ISO TC184/SC4 Secretariat directly at: sc4sec@tc184-sc4.org

NOTE The information provided in computer-interpretable form at the above URLs is informative. The information that is contained in the body of this part of ISO 10303 is normative.

Bibliography

[1] ISO 1000, *SI units and recommendations for the use of their multiples and of certain other units.*

[2] ISO 10303-11, *Industrial automation systems and integration — Product data representation and exchange — Part 11: Description methods: The EXPRESS language reference manual.*

[3] ISO 10303-28, *Industrial automation systems and integration — Product data representation and exchange — Part 28: Implementation methods: XML representations of EXPRESS schemas and data, using XML schemas.*

[4] ISO 10303-214, *Industrial automation systems and integration — Product data representation and exchange — Part 214: Application protocol: Core data for automotive mechanical design processes.*

[5] ISO 10303-240, *Industrial automation systems and integration — Product data representation and exchange — Part 240: Application protocol: Process plans for machined products.*

[6] ISO/TS 10303-1011, *Industrial automation systems and integration — Product data representation and exchange — Part 1011: Application module: Person organization*.

[7] ISO/TS 10303-1051, *Industrial automation systems and integration — Product data representation and exchange — Part 1051: Application module: Geometric tolerance*.

[8] ISO/TS 10303-1233, *Industrial automation systems and integration — Product data representation and exchange — Part 1233: Application module: Requirement assignment*.

[9] ISO 80000 (all parts) *Quantities and units.*

[10] IEC 80000 (all parts) *Quantities and units.*

[11] ISO/IEC 8824-1, *Information technology — Abstract Syntax Notation One (ASN.1) — Part 1: Specification of basic notation.*

[12] IDEF0 Federal Information Processing Standards Publication 183, Integration Definition for Functional Modeling (IDEF0), FIPS Pub 183, National Institute of Standards and Technology, December 1993.

[13] SWINDELLS, N. Communicating Materials Information – Product Data Technology for Materials, *Int. Materials Reviews*, 2002, vol. 47, no. 1, p. 31-46.

Index

ICS 25.040.40

Price based on 283 pages