



INTERNATIONAL STANDARD ISO 10303-235:2009

TECHNICAL CORRIGENDUM 1

Published 2011-09-15

INTERNATIONAL ORGANIZATION FOR STANDARDIZATION • МЕЖДУНАРОДНАЯ ОРГАНИЗАЦИЯ ПО СТАНДАРТИЗАЦИИ • ORGANISATION INTERNATIONALE DE NORMALISATION

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

Part 235:

Application protocol: Engineering properties for product design and verification

TECHNICAL CORRIGENDUM 1

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

RECTIFICATIF TECHNIQUE 1

Technical Corrigendum 1 to ISO 10303-235:2009 was prepared by Technical Committee ISO/TC 184, *Automation systems and integration*, Subcommittee SC 4, *Industrial data*.

This Technical Corrigendum addresses omissions in the AIM EXPRESS short listing of the engineering_properties_schema in 5.2 and the resulting omissions in the AIM EXPRESS expanded listing in Annex A. This Technical Corrigendum also corrects errors and omissions in the AIM EXPRESS-G drawings in Annex H.

ICS 25.040.40

Ref. No. ISO 10303-235:2009/Cor.1:2011(E)

© ISO 2011 – All rights reserved

Published in Switzerland

Page xiv, Introduction

Add an information element “Process_properties” to Figure 2. Change the figure title from “Generic model for a process” to “Information model for a process”. Remove the current Figure 2 and replace with:

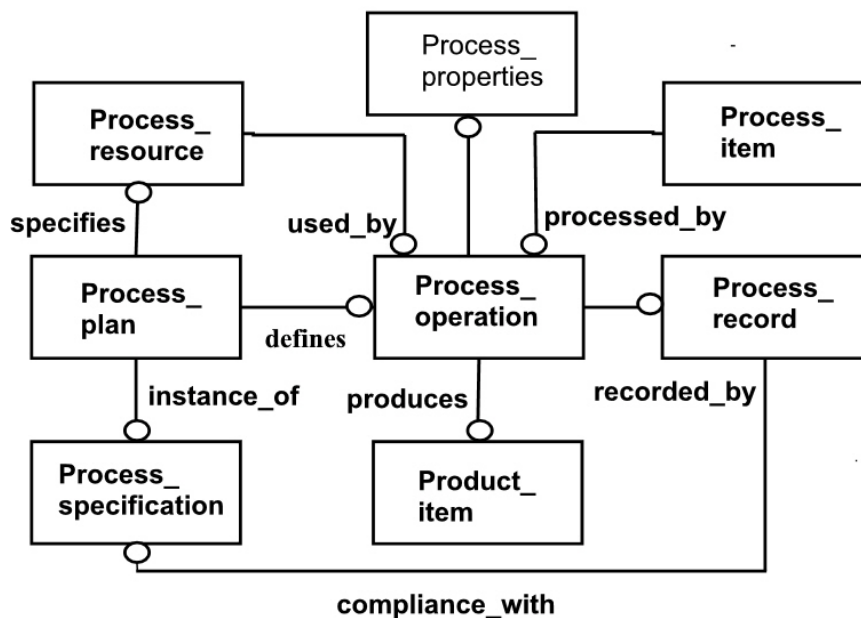


Figure 2 — Information model for a process

Page 239, 5.2, AIM EXPRESS short-listing

References to additional SI units that were specified in ISO 10303-41:2005/Cor 1:2008 were omitted from the AIM EXPRESS short listing.

Add:

```

si_absorbed_dose_unit,
si_capacitance_unit,
si_conductance_unit,
si_dose_equivalent_unit,
si_electric_charge_unit,
si_electric_potential_unit,
si_energy_unit,
si_force_unit,
si_frequency_unit,
si_illuminance_unit,
si_inductance_unit,
si_magnetic_flux_unit,
si_power_unit,
si_pressure_unit,
si_radioactivity_unit,
si_resistance_unit,

```

between **resistance_measure_with_unit** and **si_prefix** in the EXPRESS specification of 5.2.

Page 240, 5.2, AIM EXPRESS short-listing

The reference to **velocity_unit** was omitted and **vleocity_measure** was included by mistake. Replace **vleocity_measure** with **velocity_unit**.

Change:

```
velocity_measure,  
vleocity_measure,  
velocity_measure_with_unit,
```

to:

```
velocity_measure,  
velocity_unit,  
velocity_measure_with_unit,
```

in the EXPRESS specification.

Page 243, 5.2, AIM EXPRESS short-listing

Remove references to the functions: **item_correlation** and **get_multi_language** from the AIM EXPRESS short listing. References to functions are not allowed in the AIM EXPRESS short listing.

Delete the following lines from the EXPRESS specification:

```
REFERENCE FROM cutting_tools_schema (item_correlation); -- ISO 13399-1
```

```
REFERENCE FROM automotive_design_schema (get_multi_language); -- ISO 10303-214
```

Pages 284 to 527, Annex A, AIM EXPRESS expanded listing

Replace the entire contents of Annex A with the revised schema resulting from the changes to the short listing in 5.2.

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 si_absorbed_dose_unit
SUBTYPE OF (absorbed_dose_unit, si_unit);
WHERE
    WR1: SELF\si_unit.name = si_unit_name.gray;
    WR2: NOT EXISTS(SELF\derived_unit.name);
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_constraint_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);
        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_ld_table(loc);
WR3:
        function_is_ld_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 si_capacitance_unit
SUBTYPE OF (capacitance_unit, si_unit);
WHERE
  WR1: SELF\si_unit.name = si_unit_name.farad;
  WR2: NOT EXISTS(SELF\derived_unit.name);
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, 1.000000);
    WHERE
        WR1:
            scl > 0.000000;
    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 si_conductance_unit
  SUBTYPE OF (conductance_unit, si_unit);
WHERE
  WR1: SELF\si_unit.name = si_unit_name.siemens;
  WR2: NOT EXISTS (SELF\derived_unit.name);
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.relate_method :=:
related_relationship.relate_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.relate_action :=:
related_relationship.relate_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_component);
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.relatiing_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.operands,
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 si_dose_equivalent_unit
SUBTYPE OF (dose_equivalent_unit, si_unit);
WHERE
    WR1: SELF\si_unit.name = si_unit_name.sievert;
    WR2: NOT EXISTS(SELF\derived_unit.name);
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 si_electric_charge_unit
  SUBTYPE OF (electric_charge_unit, si_unit);
WHERE
  WR1: SELF\si_unit.name = si_unit_name.coulomb;
  WR2: NOT EXISTS(SELF\derived_unit.name);
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 si_electric_potential_unit
  SUBTYPE OF (electric_potential_unit, si_unit);
  WHERE
    WR1: SELF\si_unit.name = si_unit_name.volt;
    WR2: NOT EXISTS(SELF\derived_unit.name);
  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 si_energy_unit
  SUBTYPE OF (energy_unit, si_unit);
  WHERE
    WR1: SELF\si_unit.name = si_unit_name.joule;
    WR2: NOT EXISTS(SELF\derived_unit.name);
  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 si_force_unit
SUBTYPE OF (force_unit, si_unit);
WHERE
  WR1: SELF\si_unit.name = si_unit_name.newton;
  WR2: NOT EXISTS(SELF\derived_unit.name);
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_measure_with_unit
SUBTYPE OF (measure_with_unit);
WHERE
  WR1:
    'ENGINEERING_PROPERTIES_SCHEMA.FREQUENCY_UNIT' IN
TYPEOF(SELF\measure_with_unit.unit_component);
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 si_frequency_unit
  SUBTYPE OF (frequency_unit, si_unit);
WHERE
  WR1: SELF\si_unit.name = si_unit_name.hertz;
  WR2: NOT EXISTS(SELF\derived_unit.name);
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 si_illuminance_unit
  SUBTYPE OF (illuminance_unit, si_unit);
WHERE
  WR1: SELF\si_unit.name = si_unit_name.lux;
  WR2: NOT EXISTS(SELF\derived_unit.name);
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 si_inductance_unit
  SUBTYPE OF (inductance_unit, si_unit);
WHERE
  WR1: SELF\si_unit.name = si_unit_name.henry;
  WR2: NOT EXISTS(SELF\derived_unit.name);
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 si_magnetic_flux_density_unit
  SUBTYPE OF (magnetic_flux_density_unit, si_unit);
WHERE
  WR1: SELF\si_unit.name = si_unit_name.tesla;
  WR2: NOT EXISTS(SELF\derived_unit.name);
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 si_magnetic_flux_unit
  SUBTYPE OF (magnetic_flux_unit, si_unit);
WHERE
  WR1: SELF\si_unit.name = si_unit_name.weber;
  WR2: NOT EXISTS(SELF\derived_unit.name);
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 si_power_unit
    SUBTYPE OF (power_unit, si_unit);
    WHERE
        WR1: SELF\si_unit.name = si_unit_name.watt;
        WR2: NOT EXISTS(SELF\derived_unit.name);
    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 si_pressure_unit
    SUBTYPE OF (pressure_unit, si_unit);
    WHERE
        WR1: SELF\si_unit.name = si_unit_name.pascal;
        WR2: NOT EXISTS(SELF\derived_unit.name);
    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_component) =
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 si_radioactivity_unit
    SUBTYPE OF (radioactivity_unit, si_unit);
    WHERE
        WR1: SELF\si_unit.name = si_unit_name.becquerel;
        WR2: NOT EXISTS(SELF\derived_unit.name);
    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 : repack_options;
    output_repack : repack_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 si_resistance_unit
    SUBTYPE OF (resistance_unit, si_unit);
    WHERE
        WR1: SELF\si_unit.name = si_unit_name.ohm;
        WR2: NOT EXISTS(SELF\derived_unit.name);
    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.relate_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 velocity_unit
SUBTYPE OF (derived_unit);
WHERE
    WR1: derive_dimensional_exponents(SELF) =
        dimensional_exponents ( 1.0, 0.0, -1.0, 0.0, 0.0, 0.0, 0.0 );
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,
     product_material_composition_relationship);
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,
        repack_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 repack_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;
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;

```

```

        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;
    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.base,
            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;
    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
    efunum : elementary_function_enumerators;
    mthfun : maths_function;
END_LOCAL;
IF schema_prefix + 'MATHS_FUNCTION' IN TYPEOF(func) THEN
    mthfun := func;
ELSE
    efunum := func;
    mthfun := make_elementary_function(efunum);
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;

```

```

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
    (igr1 : 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;
END_IF;

```

```

        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, 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;
    END_REPEAT;
END_IF;
END_FUNCTION;

```

```

        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_e
xponent;
        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_e
xponent;
        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.electr
ic_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.thermo
dynamic_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.lumino
us_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;
    idxs : 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);
    END_IF;

```

```

        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
n.mat.range),
func\homogeneous_linear_function.mat\explicit_table_function.shape[func\homogeneo
us_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_function.ma
t.range),
func\general_linear_function.mat\explicit_table_function.shape[func\general_linea
r_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_knots[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.variables[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.domain_arg
ument);
    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_function.oper
ands)].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
n.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_function.ma
t.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(tspace)),
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.range_argu
ment);
    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_space.ba
se), 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 > 0.000000 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 := 0.000000;
        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 > 0.000000 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 := 0.000000;
                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;
END_IF;

```

```

        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;
    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, aitr, 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( -0.50000 * 3.14159, aitr, 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( 0.50000 * 3.14159, aitr, 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;
    END_IF;
    xitr := 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;

```

```

    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;
    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;
      END_IF;
    END_IF;
  END_IF;
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;
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 + rxp;
    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 := 0.000000;
            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(rm xp / rc);
        IF strictly_in(apc - a, aitp) THEN
            amin := ab - ASIN(rm xp / rc);
            amin_in := max_included(ritp);
        END_IF;
        IF strictly_in(apc + a, aitp) THEN
            amax := ab + ASIN(rm xp / 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_region_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(yitv) 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_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
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;
    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;
  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;
    END_IF;
END_IF;

```

```

        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;
    END_IF;
END_FUNCTION;

```

```

        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);
    nargs : positive_integer := SIZEOF(arguments);
    arg : generic_expression;
END_LOCAL;
    IF schema_prefix + 'PRODUCT_SPACE' IN domain_types THEN
        IF space_dimension(domain) <> nargs THEN
            RETURN (FALSE);
        END_IF;
    ELSE
        IF schema_prefix + 'EXTENDED_TUPLE_SPACE' IN domain_types THEN
            IF space_dimension(domain) > nargs THEN
                RETURN (FALSE);
            END_IF;
        ELSE
            RETURN (FALSE);
        END_IF;
    END_IF;
    REPEAT i := 1 TO nargs;
        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;
  prepfuns : LIST [2:?] OF maths_function;
  finfunc : maths_function_select ) : parallel_composed_function;
  RETURN (parallel_composed_function(srcdom, prepfuns, finfunc) ||
maths_function() || generic_expression() ||
multiple_arity_generic_expression(convert_to_operands_prcmfn(srcdom, prepfuns,
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;

```

```

        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;

```

```

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;

```

```

        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_mathspaces(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_mathspaces(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;
    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_domain,
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;
    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;

```

```

    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 repack
    (tspace : tuple_space;
     repckg : repack_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_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;
    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;
    END_IF;

```

```

        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);
    opl := convert_to_operand(v1);
END_IF;
IF 'BINARY_GENERIC_EXPRESSION' IN types THEN
    v1 :=
simplify_generic_expression(expr\binary_generic_expression.operands[1]);
    opl := 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;
    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;
  END_IF;

```

```

    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;
  aitiv : 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;
    END_IF;
END_IF;

```

```

        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;
    END;
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_mathspaces));
    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_expression.derivand)));
END_IF;
IF schema_prefix + 'DEFINITE_INTEGRAL_EXPRESSION' IN typenames THEN
    RETURN
(drop_numeric_constraints(values_space_of(expr\definite_integral_expression.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 := 1.000000;
      vec1 := arg1;
    END_IF;
    IF 'ENGINEERING_PROPERTIES_SCHEMA.VECTOR' IN TYPEOF(arg2) THEN
      mag2 := arg2.magnitude;
      vec2 := arg2.orientation;
    ELSE
      mag2 := 1.000000;
      vec2 := arg2;
    END_IF;
    vec1 := normalise(vec1);
    vec2 := normalise(vec2);
    ndim := SIZEOF(vec1.direction_ratios);
    mag := 0.000000;
    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 > 0.000000 THEN
      result := dummy_gri || vector(res, SQRT(mag));
    ELSE
      result := dummy_gri || vector(vec1, 0.000000);
    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;

```

Page 604, Figure H.6

The diagram shown in Figure H.6 in ISO 10303-235:2009 is a repeat of the diagram in Figure H.5. The caption for Figure H.6 was correct in the original version and is unchanged for this replacement. The diagram for Figure H.6 should be replaced by the diagram shown below.

