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

**Part 50:
Integrated generic resource: Mathematical
constructs**

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

Partie 50: Ressources génériques intégrées: Constructions mathématiques



Reference number
ISO 10303-50:2002(E)

AVA Documents
©ISO 2002
Document Sharing Hub

PDF disclaimer

This PDF file may contain embedded typefaces. In accordance with Adobe's licensing policy, this file may be printed or viewed but shall not be edited unless the typefaces which are embedded are licensed to and installed on the computer performing the editing. In downloading this file, parties accept therein the responsibility of not infringing Adobe's licensing policy. The ISO Central Secretariat accepts no liability in this area.

Adobe is a trademark of Adobe Systems Incorporated.

Details of the software products used to create this PDF file can be found in the General Info relative to the file; the PDF-creation parameters were optimized for printing. Every care has been taken to ensure that the file is suitable for use by ISO member bodies. In the unlikely event that a problem relating to it is found, please inform the Central Secretariat at the address given below.

© ISO 2002

All rights reserved. Unless otherwise specified, no part of this publication may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying and microfilm, without permission in writing from either ISO at the address below or ISO's member body in the country of the requester.

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

Printed in Switzerland

Contents

Page

1	Scope	1
2	Normative references	1
3	Terms, definitions, and symbols	2
3.1	Terms defined in ISO 10303-1	2
3.2	Other terms and definitions	2
3.3	Symbols	5
4	Mathematical functions	6
4.1	Introduction	8
4.2	Fundamental concepts and assumptions	8
4.2.1	Mathematical function	8
4.2.2	Mathematical object or value	9
4.2.3	Mathematical expression	10
4.2.4	Mathematical space	10
4.2.5	Mathematical tuple	11
4.2.6	Function domains and ranges	11
4.2.7	Spaces of one-tuples	12
4.2.8	Array function	12
4.2.9	Table function	12
4.2.10	Matrix	12
4.2.11	Inputs and Parameters	13
4.2.12	Function evaluation	14
4.2.13	Function application	14
4.3	Mathematical functions schema constant definitions	14
4.3.1	schema_prefix	15
4.3.2	the elementary spaces	15
4.3.3	the empty space	15
4.3.4	real intervals	15
4.3.5	tuple spaces	16
4.3.6	empty values	16
4.4	Mathematical functions schema type definitions	17
4.4.1	nonnegative_integer	17
4.4.2	positive_integer	17
4.4.3	zero_or_one	17
4.4.4	one_or_two	18
4.4.5	local names for simple types	18
4.4.6	maths_simple_atom	18
4.4.7	maths_atom	19
4.4.8	atom_based_tuple	19
4.4.9	atom_based_value	19
4.4.10	maths_tuple	20
4.4.11	maths_value	20
4.4.12	maths_expression	20
4.4.13	maths_function_select	21
4.4.14	input_selector	21
4.4.15	elementary_space_enumerators	22
4.4.16	ordering_type	23
4.4.17	lower_upper	24
4.4.18	symmetry_type	24

4.4.19	elementary_function_enumerators	25
4.4.20	open_closed	31
4.4.21	space_constraint_type	31
4.4.22	repackage_options	32
4.4.23	extension_options	32
4.4.24	maths_enum_atom	33
4.4.25	dotted_express_identifier	34
4.4.26	express_identifier	34
4.4.27	product_space	34
4.4.28	tuple_space	35
4.4.29	maths_space_or_function	35
4.4.30	real_interval	35
4.5	Mathematical functions schema entity definitions	36
4.5.1	quantifier_expression	36
4.5.2	dependent_variable_definition	37
4.5.3	bound_variable_semantics	37
4.5.4	free_variable_semantics	38
4.5.5	complex_number_literal	38
4.5.6	logical_literal	39
4.5.7	binary_literal	39
4.5.8	maths_enum_literal	39
4.5.9	real_tuple_literal	40
4.5.10	integer_tuple_literal	40
4.5.11	atom_based_literal	40
4.5.12	maths_tuple_literal	41
4.5.13	maths_variable	41
4.5.14	maths_real_variable	42
4.5.15	maths_integer_variable	42
4.5.16	maths_boolean_variable	43
4.5.17	maths_string_variable	43
4.5.18	function_application	44
4.5.19	maths_space	45
4.5.20	elementary_space	46
4.5.21	finite_integer_interval	46
4.5.22	integer_interval_from_min	47
4.5.23	integer_interval_to_max	47
4.5.24	finite_real_interval	47
4.5.25	real_interval_from_min	48
4.5.26	real_interval_to_max	49
4.5.27	cartesian_complex_number_region	49
4.5.28	polar_complex_number_region	50
4.5.29	finite_space	51
4.5.30	uniform_product_space	52
4.5.31	listed_product_space	53
4.5.32	extended_tuple_space	54
4.5.33	function_space	55
4.5.34	maths_function	56
4.5.35	finite_function	57
4.5.36	constant_function	58
4.5.37	selector_function	59
4.5.38	elementary_function	60
4.5.39	restriction_function	60

4.5.40	repackaging_function	61
4.5.41	reindexed_array_function	63
4.5.42	series_composed_function	64
4.5.43	parallel_composed_function	64
4.5.44	explicit_table_function	67
4.5.45	listed_real_data	68
4.5.46	listed_integer_data	69
4.5.47	listed_logical_data	69
4.5.48	listed_string_data	70
4.5.49	listed_complex_number_data	71
4.5.50	listed_data	72
4.5.51	externally_listed_data	72
4.5.52	linearized_table_function	73
4.5.53	standard_table_function	75
4.5.54	regular_table_function	76
4.5.55	triangular_matrix	78
4.5.56	strict_triangular_matrix	78
4.5.57	symmetric_matrix	79
4.5.58	symmetric_banded_matrix	80
4.5.59	banded_matrix	81
4.5.60	basic_sparse_matrix	83
4.5.61	homogeneous_linear_function	85
4.5.62	general_linear_function	86
4.5.63	b_spline_basis	87
4.5.64	b_spline_function	88
4.5.65	rationalize_function	89
4.5.66	partial_derivative_function	91
4.5.67	partial_derivative_expression	92
4.5.68	definite_integral_function	94
4.5.69	definite_integral_expression	95
4.5.70	abstracted_expression_function	97
4.5.71	expression_denoted_function	98
4.5.72	imported_point_function	99
4.5.73	imported_curve_function	100
4.5.74	imported_surface_function	100
4.5.75	imported_volume_function	101
4.5.76	application_defined_function	102
4.5.77	mathematical_description	103
4.6	Mathematical functions schema function definitions	104
4.6.1	all_members_of_es	104
4.6.2	any_space_satisfies	106
4.6.3	assoc_product_space	107
4.6.4	atan2	109
4.6.5	bool	110
4.6.6	check_sparse_index_domain	110
4.6.7	check_sparse_loc_range	111
4.6.8	check_sparse_index_to_loc	112
4.6.9	compare_basis_and_coef	113
4.6.10	compare_list_and_value	113
4.6.11	compare_values	114
4.6.12	compatible_complex_number_regions	115
4.6.13	compatible_es_values	117

4.6.14	compatible_intervals	118
4.6.15	compatible_spaces	118
4.6.16	composable_sequence	124
4.6.17	convert_to_literal	124
4.6.18	convert_to_maths_function	125
4.6.19	convert_to_maths_value	125
4.6.20	convert_to_operand	126
4.6.21	convert_to_operands	127
4.6.22	convert_to_operands_premfn	128
4.6.23	definite_integral_check	128
4.6.24	definite_integral_expr_check	129
4.6.25	derive_definite_integral_domain	130
4.6.26	derive_elementary_function_domain	132
4.6.27	derive_elementary_function_range	135
4.6.28	derive_finite_function_domain	137
4.6.29	derive_finite_function_range	138
4.6.30	derive_function_domain	138
4.6.31	derive_function_range	141
4.6.32	domain_from	144
4.6.33	dot_count	145
4.6.34	dotted_identifiers_syntax	145
4.6.35	drop_numeric_constraints	146
4.6.36	enclose_cregion_in_pregion	147
4.6.37	enclose_pregion_in_cregion	151
4.6.38	enclose_pregion_in_pregion	154
4.6.39	equal_cregion_pregion	160
4.6.40	equal_maths_functions	162
4.6.41	equal_maths_spaces	163
4.6.42	equal_maths_values	166
4.6.43	es_subspace_of_es	168
4.6.44	expression_is_constant	169
4.6.45	extract_factors	169
4.6.46	extremal_position_check	170
4.6.47	factor1	171
4.6.48	factor_space	172
4.6.49	free_variables_of	172
4.6.50	function_applicability	173
4.6.51	function_is_1d_array	174
4.6.52	function_is_1d_table	175
4.6.53	function_is_2d_table	176
4.6.54	function_is_array	177
4.6.55	function_is_table	177
4.6.56	has_values_space	178
4.6.57	list_selected_components	180
4.6.58	make_abstracted_expression_function	180
4.6.59	make_atom_based_literal	181
4.6.60	make_b_spline_basis	181
4.6.61	make_b_spline_function	182
4.6.62	make_banded_matrix	182
4.6.63	make_basic_sparse_matrix	183
4.6.64	make_binary_literal	184
4.6.65	make_boolean_literal	184

4.6.66	make_cartesian_complex_number_region	185
4.6.67	make_complex_number_literal	185
4.6.68	make_constant_function	186
4.6.69	make_cos_expression	186
4.6.70	make_definite_integral_expression	187
4.6.71	make_definite_integral_function	187
4.6.72	make_elementary_function	188
4.6.73	make_elementary_space	188
4.6.74	make_environment	189
4.6.75	make_expression_denoted_function	189
4.6.76	make_extended_tuple_space	190
4.6.77	make_finite_function	190
4.6.78	make_finite_integer_interval	191
4.6.79	make_finite_real_interval	191
4.6.80	make_finite_space	192
4.6.81	make_function_application	192
4.6.82	make_function_space	193
4.6.83	make_general_linear_function	194
4.6.84	make_int_literal	194
4.6.85	make_integer_interval_from_min	195
4.6.86	make_listed_complex_number_data	195
4.6.87	make_listed_data	196
4.6.88	make_listed_integer_data	196
4.6.89	make_listed_product_space	197
4.6.90	make_listed_real_data	197
4.6.91	make_logical_literal	198
4.6.92	make_maths_enum_literal	198
4.6.93	make_maths_real_variable	199
4.6.94	make_maths_tuple_literal	199
4.6.95	make_mult_expression	200
4.6.96	make_parallel_composed_function	200
4.6.97	make_partial_derivative_expression	201
4.6.98	make_partial_derivative_function	201
4.6.99	make_polar_complex_number_region	202
4.6.100	make_rationalize_function	202
4.6.101	make_real_interval_from_min	203
4.6.102	make_real_interval_to_max	203
4.6.103	make_real_literal	204
4.6.104	make_regular_table_function	204
4.6.105	make_reindexed_array_function	205
4.6.106	make_repackaging_function	205
4.6.107	make_selector_function	206
4.6.108	make_series_composed_function	207
4.6.109	make_sin_expression	207
4.6.110	make_standard_table_function	208
4.6.111	make_strict_triangular_matrix	208
4.6.112	make_string_literal	209
4.6.113	make_unary_minus_expression	210
4.6.114	make_uniform_product_space	210
4.6.115	max_exists	211
4.6.116	max_included	211
4.6.117	member_of	212

4.6.118	min_exists	217
4.6.119	min_included	217
4.6.120	no_cyclic_domain_reference	218
4.6.121	no_cyclic_space_reference	219
4.6.122	nondecreasing	220
4.6.123	number_superspace_of	220
4.6.124	number_tuple_subspace_check	221
4.6.125	one_tuples_of	221
4.6.126	parallel_composed_function_composability_check	222
4.6.127	parallel_composed_function_domain_check	222
4.6.128	parse_express_identifier	223
4.6.129	partial_derivative_check	224
4.6.130	real_max	225
4.6.131	real_min	225
4.6.132	regular_indexing	226
4.6.133	remove_first	227
4.6.134	repackage	227
4.6.135	shape_of_array	228
4.6.136	simplify_function_application	229
4.6.137	simplify_generic_expression	243
4.6.138	simplify_maths_space	249
4.6.139	simplify_maths_value	250
4.6.140	singleton_member_of	251
4.6.141	space_dimension	252
4.6.142	space_is_continuum	252
4.6.143	space_is_singleton	253
4.6.144	stripped_typeof	254
4.6.145	subspace_of	254
4.6.146	subspace_of_es	260
4.6.147	substitute	261
4.6.148	values_space_of	263
Annex A (normative)	Short names of entities	266
Annex B (normative)	Information object registration	269
Annex C (informative)	Computer-interpretable listings	270
Annex D (informative)	EXPRESS-G diagrams	271
Bibliography		282
Index		283

Figures

Figure 1 — Schema relationships of the mathematical_functions_schema	xi
Figure D.1 — EXPRESS-G diagram of the mathematical_functions_schema (1 of 10) . . .	272
Figure D.2 — EXPRESS-G diagram of the mathematical_functions_schema (2 of 10) . . .	273
Figure D.3 — EXPRESS-G diagram of the mathematical_functions_schema (3 of 10) . . .	274
Figure D.4 — EXPRESS-G diagram of the mathematical_functions_schema (4 of 10) . . .	275
Figure D.5 — EXPRESS-G diagram of the mathematical_functions_schema (5 of 10) . . .	276
Figure D.6 — EXPRESS-G diagram of the mathematical_functions_schema (6 of 10) . . .	277

Figure D.7 — EXPRESS-G diagram of the mathematical_functions_schema (7 of 10) . . . 278

Figure D.8 — EXPRESS-G diagram of the mathematical_functions_schema (8 of 10) . . . 279

Figure D.9 — EXPRESS-G diagram of the mathematical_functions_schema (9 of 10) . . . 280

Figure D.10 — EXPRESS-G diagram of the mathematical_functions_schema (10 of 10) . . 281

Tables

Table 1 — Mathematical symbology 6

Table 2 — Orderings indicated by ordering_type 23

Table A.1 — Short names of entities 266

10303-50:2002(E)

Foreword

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

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 3.

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

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

ISO 10303-50 was prepared by Technical Committee ISO/TC 184, *Industrial automation systems and integration*, Subcommittee SC4, *Industrial data*.

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

Each part of this International Standard is a member of one of the following series: description methods, implementation methods, conformance testing methodology and framework, integrated generic resources, integrated application resources, application protocols, abstract test suites, application interpreted constructs, and application modules. This part is a member of the integrated generic resources series. The integrated generic resources and the integrated application resources specify a single conceptual product data model.

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

[<http://www.nist.gov/sc4/editing/step/titles/>](http://www.nist.gov/sc4/editing/step/titles/)

Annexes A and B form a normative part of this part of ISO 10303. Annexes C and D are for information only.

Introduction

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

This part of ISO 10303 is a member of the integrated resource series. This part of ISO 10303 specifies the **mathematical_functions_schema**.

This part of ISO 10303 specifies EXPRESS data representations for a large class of mathematical functions, expressions, and arrays. They are intended to be used to communicate product property data and related engineering analysis data. Familiarity with the branches of mathematics commonly used in engineering applications is assumed. The central concept is that mathematical functions and arrays are the abstract data objects of interest for expressing any property which requires more than a single number as a value, for describing deterministic relationships between properties, and for documenting behavioural responses of products to varying conditions.

The relationships of the schema in this part of ISO 10303 to other schemas that define the integrated resources of this International Standard are illustrated in Figure 1 using the EXPRESS-G notation. EXPRESS-G is defined in annex D of ISO 10303-11. The documents containing the specifications for these related schemas are identified in note 1 at the beginning of clause 4.

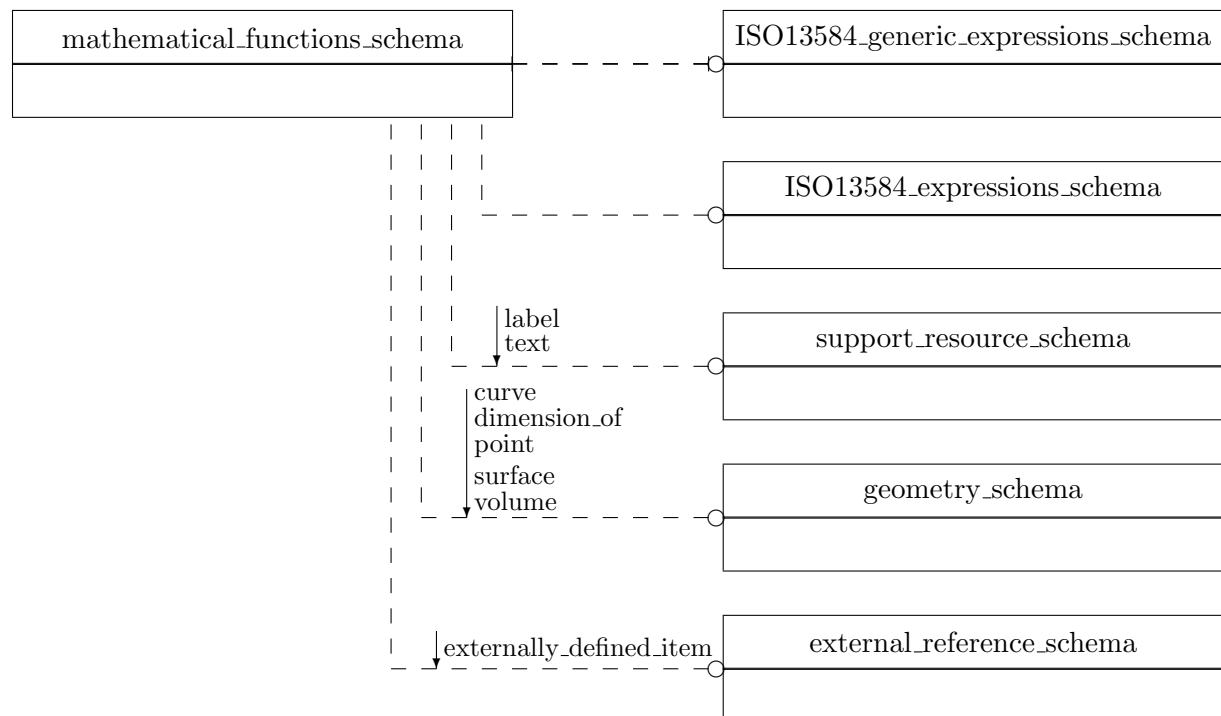


Figure 1 – Schema relationships of the mathematical_functions_schema

Copyright International Organization for Standardization

Industrial automation systems and integration — Product data representation and exchange — Part 50 : Integrated generic resource: Mathematical constructs

1 Scope

This part of ISO 10303 specifies the resource constructs for the explicit representation of mathematical structures and data related to properties of a product.

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

- multi-dimensional tables;
- mathematical expressions;
- mathematical functions;
- mathematical spaces.

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

- context of application;
- physical units;
- non-mathematical semantics.

2 Normative references

The following normative documents contain provisions which, through reference in this text, constitute provisions of this part of ISO 10303. For dated references, subsequent amendments to, or revisions of, any of these publications do not apply. However, parties to agreements based on this part of ISO 10303 are encouraged to investigate the possibility of applying the most recent editions of the normative documents indicated below. For undated references, the latest edition of the normative document referred to applies. Members of ISO and IEC maintain registers of currently valid International Standards.

ISO/IEC 8824-1:1998, *Information technology — Abstract Syntax Notation One (ASN.1): Specification of basic notation*

ISO 10303-1:1994, *Industrial automation systems and integration — Product data representation and exchange — Part 1: Overview and fundamental principles*

© ISO 2002 — All rights reserved

ISO 10303-50:2002(E)

ISO 10303-11:1994, *Industrial automation systems and integration — Product data representation and exchange — Part 11: Description methods: The EXPRESS language reference manual*

ISO 10303-41:2000, *Industrial automation systems and integration — Product data representation and exchange — Part 41: Integrated generic resource: Fundamentals of product description and support*

ISO 10303-42:2000, *Industrial automation systems and integration — Product data representation and exchange — Part 42: Integrated generic resource: Geometric and topological representation*

ISO 13584-20:1998, *Industrial automation systems and integration — Parts Library — Part 20: Logical resource: Logical model of expressions*

3 Terms, definitions, and symbols

3.1 Terms defined in ISO 10303-1

For the purpose of this part of ISO 10303, the following terms defined in ISO 10303-1 apply:

— integrated resource

3.2 Other terms and definitions

For the purposes of this part of ISO 10303, the following definitions apply:

3.2.1

actual function domain

mathematical space containing precisely the tuples of inputs to the function which are related to some tuple of outputs

NOTE See the definitions for “mathematical function” and “function domain”. See also 4.2.1 and 4.2.6.

3.2.2

actual function range

mathematical space containing precisely the tuples of outputs from the function which are related to some tuple of inputs

NOTE See the definitions for “mathematical function” and “function range”. See also 4.2.1 and 4.2.6.

3.2.3

array function

function whose domain is either a Cartesian product of finite intervals of integers or the one-tuples from such a Cartesian product

NOTE An element of such a domain is a subscript tuple. Evaluation of the function at such an element produces the “array entry” for that subscript tuple.

3.2.4

bound variable

variable which has been specifically referenced by a quantifier in a quantifier expression

NOTE A bound variable is no longer available for substitution by a constant value. The semantics of the quantifier expression uses all possible values of the variable rather than one (perhaps undetermined) value.

EXAMPLE 1 The variable x in the statement $\forall x(x = x)$.

EXAMPLE 2 The variable x in the set declaration $\{x | x^2 = x\}$.

EXAMPLE 3 The variable x in the definite integral $\int_1^2 \ln(x) dx$.

EXAMPLE 4 The variable x in the function definition $f(x) \equiv x + 2$.

3.2.5

Cartesian product space

mathematical space consisting of all ordered tuples whose components are members of the corresponding factor spaces of the product

NOTE Some care must be taken to be explicit about which spaces are the factors entering into the Cartesian product. For example, $R^3 \times R^3$ is the space of ordered pairs of ordered triples from space R , not the space of ordered sextuples from space R . When the factor spaces of the explicit factors are to be treated as the factor spaces of the Cartesian product, that is, the member tuples are to be concatenated rather than entupled, a subscript ‘ a ’ (indicating the “associative” product) is appended to the Cartesian product symbol. Thus, $R^3 \times_a R^3 \equiv R^6$.

3.2.6

compatible spaces

mathematical spaces whose intersection is not determined to be empty by the algorithm represented by the EXPRESS function **compatible_spaces**

NOTE Two spaces are compatible if their intersection is not “obviously” empty. In practice, “obvious” is defined by what is detected by the algorithm. The purpose is to rule out some obvious nonsense while permitting ignorance.

3.2.7

computable function

mathematical function for which the relationship between tuples of inputs and tuples of outputs is expressible by means of an algorithm which takes the inputs and produces the related outputs

NOTE All the mathematical functions of interest in the intended applications of this part of ISO 10303 are computable. Nevertheless, the possibility of representing non-computable functions is not ruled out.

3.2.8

expression

language construct composed of constants, variables, operators, quantifiers, and grouping markers, organized in accordance with the language’s grammar and denoting some object in the domain of the language

3.2.9

free variable

variable which has not been bound by a quantifier in the expression in which it is used

NOTE A free variable denotes an undetermined member of some set of possible values. There is an implicit assumption that it may be substituted by any constant denoting one of those values.

3.2.10

function abstraction

quantifier operation which creates an expression denoting a mathematical function from another expression and a list of variables

NOTE The variables become bound and identify the inputs to the function. The algorithm to be used to produce an output is implied by the expression. The "abstraction" comes from the change in the semantics of what the new expression denotes. The expression going into the operation denotes some object. The resulting expression denotes a higher-order object, namely a function which produces objects of the original type from inputs of the types associated with the variables named as inputs.

3.2.11

function application

operation which creates an expression by applying a function to appropriate inputs

NOTE The resulting expression denotes the object which would be the first (and, usually, only) output of the function when it is applied to the given inputs and evaluated.

3.2.12

function domain

mathematical space specifying all nominally valid tuples of inputs to the mathematical function

NOTE See the definitions for "mathematical function" and "actual function domain". See also 4.2.1 and 4.2.6.

3.2.13

function evaluation

process of ascertaining either the outputs a function associates to given inputs, or an error indication if the function does not associate any outputs to the given inputs

NOTE In the case of a computable function, the process amounts to carrying out the algorithm on the inputs to produce the outputs, or an error indication.

3.2.14

function range

mathematical space specifying all permissible tuples of outputs from the mathematical function

NOTE See the definitions for "mathematical function" and "actual function range". See also 4.2.1 and 4.2.6.

3.2.15

mathematical function

relationship between tuples of mathematical objects called inputs and tuples of mathematical objects called outputs in which each tuple of inputs is related to at most one tuple of outputs, together with a mathematical space specifying the function domain and a mathematical space specifying the function range

NOTE The phrase "mathematical function" in this part of ISO 10303 implies a specification of the domain and range in addition to the pairing of inputs and outputs. See 4.2.1 and 4.2.6 for the reasons this definition was chosen for this part of ISO 10303.

3.2.16**mathematical object; mathematical value**

abstract object from the domain of mathematics

NOTE The terms mathematical object and mathematical value are formally synonymous in this part of ISO 10303. There is a tendency to favor the use of mathematical *value* for simple objects like numbers, characters, and enumeration values, and to favor *object* for complex objects like spaces, functions and expressions, but no formal distinction is being made.

3.2.17**mathematical space; mathematical set**

instance of the fundamental aggregate object type in the domain of mathematics

NOTE No topological, metric, ordering, or vector structure is implied or required. In this part of ISO 10303, mathematical space is synonymous with mathematical set. See 4.2.4.

3.2.18**matrix**

table function whose domain contains pairs of integers

3.2.19**table function**

array function with the additional constraint that all the integer interval factor spaces of the domain either start at zero or at one

NOTE The implication of the constraint is that only the relative values of the subscripts are significant. Hence, one can use pure relative positions (all subscript ranges start from zero), or pure ordinal positions (all subscript ranges start from one), depending on the usual practice in the application area.

3.2.20**tuple**

linearly ordered, aggregate mathematical object which has finitely many component mathematical objects

NOTE A tuple contains a first component, a second component, et cetera, up to some maximum which is the length of the tuple. The tuple of zero length exists and has no components.

3.2.21**variable**

atomic expression component which has the semantics of denoting an undetermined value from some space of possible values

NOTE See *free* variable and *bound* variable.

3.3 Symbols

For the purposes of this part of ISO 10303, the following symbols and definitions apply.

The mathematical symbol convention used in the mathematical functions schema is given in Table 1.

Table 1 – Mathematical symbology

Symbol	Definition
$\times$	Cartesian product of spaces
$\times_a$	associative Cartesian product of spaces
$\cup$	set union of spaces
B^m	The Cartesian product of m factors of space B
R^m	m-dimensional real space
Σ	summation of numerical terms
Π	product of numerical factors

4 Mathematical functions

The following EXPRESS declaration begins the **mathematical_functions_schema** and identifies the necessary external references.

EXPRESS specification:

*)

```
SCHEMA mathematical_functions_schema;
-- Master document: ISO 10303-50:2002
-- EXPRESS last modified: 2001-09-07
```

```
REFERENCE FROM ISO13584_generic_expressions_schema      -- ISO 13584-20
(binary_generic_expression,
 environment,
 generic_expression,
 generic_literal,
 generic_variable,
 multiple_arity_generic_expression,
 simple_generic_expression,
 unary_generic_expression,
 variable_semantics);
```

```
REFERENCE FROM ISO13584_expressions_schema              -- ISO 13584-20
(abs_function      AS abs_expression,
 acos_function     AS acos_expression,
 and_expression,
 asin_function     AS asin_expression,
 atan_function     AS atan_expression,
 binary_boolean_expression,
 binary_function_call AS binary_numeric_call_expression,
 binary_numeric_expression,
 boolean_defined_function AS boolean_defined_expression,
 boolean_expression,
 boolean_literal,
 boolean_variable,
 comparison_equal,
 comparison_expression,
 comparison_greater,
 comparison_greater_equal,
 comparison_less,
 comparison_less_equal,
```

comparison_not_equal,	
concat_expression,	
cos_function	AS cos_expression,
defined_function	AS defined_expression,
div_expression,	
equals_expression,	
exp_function	AS exp_expression,
expression,	
format_function	AS format_expression,
index_expression,	
int_literal,	
int_numeric_variable,	
int_value_function	AS int_value_expression,
integer_defined_function	AS integer_defined_expression,
interval_expression,	
length_function	AS length_expression,
like_expression,	
literal_number,	
log_function	AS log_expression,
log10_function	AS log10_expression,
log2_function	AS log2_expression,
maximum_function	AS maximum_expression,
minimum_function	AS minimum_expression,
minus_expression,	
minus_function	AS unary_minus_expression,
mod_expression,	
mult_expression,	
multiple_arity_boolean_expression,	
multiple_arity_function_call	AS multiple_arity_numeric_call_expression,
multiple_arity_numeric_expression,	
not_expression,	
numeric_defined_function	AS numeric_defined_expression,
numeric_expression,	
numeric_variable,	
odd_function	AS odd_expression,
or_expression,	
plus_expression,	
power_expression,	
real_defined_function	AS real_defined_expression,
real_literal,	
real_numeric_variable,	
simple_boolean_expression,	
simple_numeric_expression,	
simple_string_expression,	
sin_function	AS sin_expression,
slash_expression,	
sql_mappable_defined_function	AS sql_mappable_defined_expression,
square_root_function	AS square_root_expression,
string_defined_function	AS string_defined_expression,
string_expression,	
string_literal,	
string_variable,	
substring_expression,	
tan_function	AS tan_expression,
unary_boolean_expression,	
unary_function_call	AS unary_numeric_call_expression,
unary_numeric_expression,	
value_function	AS value_expression,
variable,	

```

xor_expression);

REFERENCE FROM support_resource_schema          -- ISO 10303-41
(label,
text);

REFERENCE FROM external_reference_schema        -- ISO 10303-41
(externally_defined_item);

REFERENCE FROM geometry_schema                 -- ISO 10303-42
(curve,
dimension_of,
point,
surface,
volume);

(*)

```

NOTE 1 The schemas referenced above can be found in the following parts of ISO 10303 and ISO 13584:

ISO13584_generic_expressions_schema	ISO 13584-20
ISO13584_expressions_schema	ISO 13584-20
support_resource_schema	ISO 10303-41
external_reference_schema	ISO 10303-41
geometry_schema	ISO 10303-42

NOTE 2 All entity types defined in the two ISO 13584-20 schemas are explicitly referenced in this schema. The identifiers in the reference statement are arranged in alphabetic order in order to facilitate identification of the defining schema.

NOTE 3 All entity types from **ISO13584_expressions_schema** whose names contained the word “function” have been renamed using the word “expression” in this part of ISO 10303. This change reflects the fact that instances of those types do not represent mathematical functions, but rather the application of certain functions to operands to form mathematical expressions.

NOTE 4 The references to **geometry_schema** are required only for the definition of the **imported_-[point,curve,surface,volume]_function** entity types.

NOTE 5 See annex D, Figures D.1 to D.10, for a graphical presentation of this schema.

4.1 Introduction

The subject of the **mathematical_functions_schema** is the representation of mathematical functions. The context for these mathematical functions is provided by other parts of ISO 10303 and the applications which use this schema. Only the unitless, purely mathematical aspects of mathematical functions are considered in this schema.

4.2 Fundamental concepts and assumptions

4.2.1 Mathematical function

As used in this schema, the phrase mathematical function refers to an identified functional relationship between mathematical objects called inputs and mathematical objects called outputs, together with a specification of the number and kinds of inputs, and a specification of the number and kinds of outputs.

A relationship is a functional relationship when the inputs determine the outputs. Usually, a functional relationship exists because the outputs associated with given inputs can be computed from the inputs by means of an algorithm.

In this schema, the specification of the number and kinds of inputs is accomplished by specifying the domain of the function as a mathematical tuple space, where the number of factors of the tuple space indicates the number of individual inputs to the function and the corresponding factor spaces of the tuple space identify the kinds of the inputs.

Analogously, the specification of the number and kinds of outputs is accomplished by specifying the range of the function as a mathematical tuple space, where the number of factors of the tuple space indicates the number of individual outputs from the function and the corresponding factor spaces of the tuple space identify the kinds of the outputs.

NOTE 1 The concept of mathematical function defined in this part of ISO 10303 is a blend of the computer science concept and the mathematical concepts. The exact form it assumes was driven by the requirements of data exchange using the existing structure of ISO 10303.

NOTE 2 Mathematics has two basic views of functions. In the set theoretic view, a function is a set of ordered pairs no two of which have the same first member. In the algorithmic view, a function is a well-formed expression in a formal language with a semantics equivalent to a programming language. Neither of these views is appropriate for the present purpose. The present purpose requires only that sufficient information be recorded so that a function can be reliably reproduced on a different system than that in which it originated. It is assumed that certain very general algorithms are sufficiently well known and understood that it is sufficient to associate them with a named subtype and supply as attributes the parameters necessary to determine a particular instance of that type of function. Examples of such general algorithms are algebraic expression evaluation, matrix multiplication, B-spline evaluation, partial differentiation, and tensor contraction.

NOTE 3 Some of the simplest and most general algorithms are those that combine existing functions to make new functions. Examples are composition of functions (that is, applying a second function to the output of a first function), concatenation of the outputs of two or more functions, and taking linear combinations of functions. In order to identify which functions it makes sense to combine in such ways, functions must be classified according to the kinds of inputs they require and the kinds of outputs they produce. This consideration leads to the inclusion of representations of mathematical spaces, mathematical values and mathematical ordered tuples in the schema.

NOTE 4 Theoretically, the functional relationship precisely determines an actual domain and an actual range and the nominal domain and range required above are redundant. Unfortunately, it is generally impossible to derive this information reliably, as it is equivalent to solving the Turing Machine Halting Problem. Even in simple cases, it is impractical. Consider a function defined as the reciprocal of a fifth degree polynomial. Its actual domain is all real numbers except the one to five real roots of the polynomial. There is no algebraic formula for solving fifth degree (or higher) polynomials. Rather than trying to deal with the complexity of actual domains or ranges, it is far better to specify the nominal domain of such a function as all real numbers, and then accept that the function may fail to produce outputs for some inputs. Similarly, it is better to specify the nominal range of a function and accept that not all values in that range actually appear as outputs.

4.2.2 Mathematical object or value

The phrases “mathematical object” and “mathematical value” are synonymous in this part of ISO 10303. Both are used for any abstract object from the domain of mathematics. As a consequence of providing representations of mathematical functions, there is need to provide a representation of any mathematical object which could be an input to a mathematical function, an output from a mathematical function, or used in a definition of a mathematical function. The EXPRESS language “select” type **maths_value** (see 4.4.11) is defined in this schema

with the intent of encompassing all EXPRESS language representations of mathematical values.

NOTE 1 The **maths_value** type is highly recursive and includes values isomorphic to any instance or value constructed using this schema except expressions containing free variables and instances of **mathematical_description**.

NOTE 2 There is one fundamental philosophical mismatch between mathematical values and their EXPRESS representations. EXPRESS instances can be created, modified, and destroyed. Mathematical values merely exist. In mathematics there is exactly one object which is the integer three, one object which is the space of all real numbers, and one object which is the (set theoretic) trigonometric sine function taking a real number as input and producing a real number as output. In the context of a language whose constructs represent mathematical objects, however, there will necessarily be many instances representing the same object (for example, 3 , $2 + 1$, $5 - 2$, $|-3|$, $e^{\ln 3}$), so no attempt is made in this schema to try to prevent this redundancy even at the simple level of multiple instances of the same entity type with the same attribute values. With the notable exception of instances of **maths_variable**, only value equality or inequality has any significance within this schema.

4.2.3 Mathematical expression

A mathematical expression is a linguistic construct, possibly containing free variables, which denotes a mathematical object once any free variables have been substituted by appropriate constants. An expression is composed of symbols representing constants, variables, operators, quantifiers, and grouping marks and is constructed in accordance with a grammar.

NOTE 1 In effect, a mathematical expression is a template with some missing parts indicated by free variables. There is a close relationship between mathematical expressions and mathematical functions. Either can be used to construct instances of the other. The difference is that an expression denotes an undetermined object of a certain sort and a corresponding function denotes a whole functional relationship among objects of that sort.

EXAMPLE If n is an integer variable, then $n + 2$ is an expression denoting an (indeterminate) integer, and it makes sense to ask whether it is, for example, an even integer, even though no answer can be given in the absence of further information.

Corresponding to this expression, there is a function, which might be described in English as "add two", which could be expressed in the formalism of the Lambda Calculus (see, for example, [4]) as $\lambda n(n + 2)$, where lambda is the quantifier indicating "function abstraction" with respect to the associated variable. The variable n in this latter expression is now a bound variable rather than a free variable. This latter expression has no free variables and therefore denotes a mathematical constant. However, the constant denoted is of a higher order than an integer, namely that of functions from integers to integers. It does not make any sense to ask whether this object is an even integer, since it is not an integer.

NOTE 2 The designation of a specific formal language, grammar, and symbology for mathematics is outside the scope of this part of ISO 10303. It is expected that the correspondence between the EXPRESS representations for mathematical expressions in this part of ISO 10303 and any formal or informal language for mathematics will be clear and unambiguous.

4.2.4 Mathematical space

In this part of ISO 10303, mathematical space is a synonym for mathematical set, the fundamental aggregate of mathematics.

NOTE 1 In many contexts the word "space" implies a set with some additional structure, such as in "topological space", "vector space", "Euclidean space", "projective space", or "Hilbert space". Here it has no such implication. The term "space" is preferred in this part of ISO 10303 to avoid confusion with the EXPRESS concept of set. In actual implementations, the EXPRESS set construct can only be used

to represent finite sets. Almost all the mathematical spaces of interest have infinitely many members. Consequently, they cannot be directly represented by an EXPRESS set value.

NOTE 2 The present edition provides representations for only the simplest and most useful mathematical spaces. The addition of further subtypes to **maths_space** awaits specific application needs.

4.2.5 Mathematical tuple

The second most important aggregate in mathematics after mathematical set is the ordered tuple. An ordered tuple has a length, a first component, a second component, et cetera, up to the length of the tuple. The components need not have any common type.

The EXPRESS list aggregate type is chosen to represent mathematical ordered tuples in this schema.

Mathematically, there is a unique zero-tuple object. As with many vacuous or trivial or extremal objects in mathematics it plays a useful simplifying role in some applications. It is represented by the zero-length list and such a list is explicitly included in **maths_tuple**, but will usually be excluded in the related tuple types.

4.2.6 Function domains and ranges

The classification of mathematical functions by the “kinds of inputs they require” and the “kinds of outputs they produce” is accomplished by explicitly representing the mathematical spaces which contain all legitimate inputs and all possible outputs. In mathematical terminology, these spaces are the domains and ranges of the functions, respectively.

In order to talk about *the* domain of a function of several inputs, the several “natural” inputs are implicitly combined into an ordered tuple and the space of all such ordered tuples is the space which is *the* domain of the function. Similarly, a function or procedure which might naturally be conceived as having multiple outputs is described as having a single output which is an ordered tuple of the “natural” outputs. The “dimension” of the tuple space which is the domain or range of a function indicates the number of individual natural inputs or outputs, respectively. The top level of tupling in a domain or range can be decomposed to obtain the individual input domains or output ranges. (See also 4.2.7.)

The range of a mathematical function is nominal to the extent that any mathematical space which contains all the actual output values may be specified as the value for the range attribute. Since mathematical function in this context means mathematical partial function, the mathematical space supplied as the value for the domain attribute is also technically nominal. However, this is somewhat deceptive. To the greatest extent that is practical, the schema requires a function domain to be the space of all “valid” inputs or some subset thereof. The cases of valid inputs for which the associated algorithm fails should generally be rare.

The domain and range attributes of the abstract supertype **maths_function** are derived attributes. For each subtype, the schema derives appropriate values from the nature of the implied algorithm and explicit attributes of that subtype. For some subtypes, the explicit attributes directly determine the domain and/or range. In any case, the effective domain or range may be further restricted by explicitly composing the function with a restriction function.

The principal effect of the values of domain and range attributes is in determining to which classes of functions the function will belong and, consequently, which roles it might fill in constructing additional functions.

The actual range space of a given mathematical function on an explicit domain space is the unique mathematical space which contains all and only the output values actually produced by the function from members of that domain. The actual range space is a subspace of every legitimate range space that might be specified for the function. Although mathematically well-defined, the image space is often ill-defined computationally.

The actual domain space of a given mathematical function refers to the unique mathematical space which contains all and only the input values for which the function's algorithm actually produces an output other than the indeterminate value '?'. It, too, is often ill-defined computationally.

4.2.7 Spaces of one-tuples

Mathematicians typically identify a space X and the space X^1 of all one-tuples of elements of X via the obvious canonical natural isomorphism. However, when this is done it becomes impossible to distinguish between the domains of a function $f(x, y)$ which takes two real numbers as input and a function $g((x, y))$ which takes a single input which is a pair of real numbers. These two cases have quite distinct domains for computer programming purposes. Consequently, this schema does maintain the distinction between the domain of f , which is R^2 , and the domain of g , which is $(R^2)^1$.

4.2.8 Array function

An array function is a mathematical function whose domain is either a subscript space or the space of one-tuples from a subscript space. A subscript space is a finite Cartesian product of finite intervals of integers. The set of valid subscript tuples for a multi-dimensional array is a subscript space. Given an ordered tuple of subscripts as input, an array function produces a corresponding value. However, an array function need not have a storage location allocated for every possible subscript tuple. Moreover, varying numbers of dimensions can be accommodated in a uniform manner without resorting to recursive constructs such as "ARRAY [1:4] OF ARRAY [0:3] OF ...". Array functions generalize, unify, and support efficient representations of the many specializations of the multi-dimensional array concept.

NOTE Since array functions which are not table functions are uncommon, this part of ISO 10303 primarily provides facilities for representing table functions. Any array function can be represented by reindexing the corresponding table function using the type **reindexed_array_function**. For an example of the representation of a general array function, see example 3 in 4.5.53.

4.2.9 Table function

A table function is an array function in which all the finite intervals of integers appearing in the domain start at zero, or all start at one. In practice, almost all instances of array functions are table functions. Table functions occur when the subscript tuple inputs have no significance beyond establishing relative position. In such cases it is most convenient to use ordinal positions (all subscript ranges start at one) or relative positions (all subscript ranges start at zero). Since both choices are widely used by applications, both are supported.

4.2.10 Matrix

Within this schema, matrix is synonymous with table function defined on a two-dimensional subscript space. Matrices are visualized as rectangular tables of values and much of the terminology is based on this visualization. A position in a matrix is a pair of integers belonging to the subscript domain. The first integer of the pair that constitute a matrix position is the

row position and the second is the column position. The matrix entry at a position is the value produced as output by the table function when it is applied to the position. Terms for important subsets of positions are also applied to the corresponding subsets of entries. A row is the set of all positions with the same row position. A column is the set of all positions with the same column position. The main diagonal is the set of all positions for which the row position equals the column position. The upper triangle is the set of positions for which the row position is less than or equal to the column position. The strictly upper triangle is the set of positions for which the row position is less than the column position. The lower triangle is the set of positions for which the row position is greater than or equal to the column position. The strictly lower triangle is the set of positions for which the row position is greater than the column position. A diagonal is a set of positions for which the signed difference between the column position and the row position is a given integer constant. This signed difference is the diagonal “position” and is referenced when discussing diagonals “above” or “below” the main diagonal.

NOTE The upper or lower “triangle” could, in the case of a non-square matrix, actually look more like a trapezoid. The word “triangle” will however still be used.

4.2.11 Inputs and Parameters

When a mathematical function is defined by means of an expression, it is sometimes necessary to view the variables as being of two types: function inputs and function parameters. The function parameters are conceived as having been given specific (but, so far, undetermined) values from their corresponding domains, and the function is viewed as having only the remaining inputs as its inputs.

In informal mathematics one might define a function f as follows:

Let $f(x) = a \cdot x + b$, where a and b are constants.

One might then describe the function f as a general linear function from R^1 to R^1 . However, if one examines the defining expression out of context, it appears to describe a quadratic function of three variables, not a linear function of one variable. The expression alone does not provide enough information to capture the mathematical intent. In the informal language of inputs and parameters used above, one would say that x is being viewed as an input while a and b are viewed as parameters.

The formalism which does capture what is occurring here is the Lambda Calculus, which would express the function definition above as:

$$f \equiv \lambda x(a \cdot x + b)$$

This notation uses a logical quantifier, λ , to make clear that the role of the variable x has changed to that of a bound variable. It is now just a placeholder in the expression for the actual input to the function. At the same time, the semantics of the expression has changed and it now denotes a real-valued function of one real variable rather than a real number. This function object is undetermined in that it still depends on the values chosen for the free variables (parameters) a and b . But the function object does not depend on x and it makes no sense to talk about choosing a value for x . Alternatively, one might say that the function object depends on all the possible values for x simultaneously.

The **abstracted_expression_function** subtype explicitly represents the quantifier operation which converts an expression into a function by identifying the variables which are to be treated as function inputs and by specifying their order. Any other variables in the expression are function parameters.

4.2.12 Function evaluation

A mathematical function is evaluated at inputs to produce the corresponding outputs. The details of the implementation of this process are out of the scope of this standard. However, when such a process is to be evaluated for conformance with this part of ISO 10303, the results shall be equivalent to the following:

First the tuple of inputs is tested to determine whether or not it is in the domain space of the function. If it is not, then the evaluation process ceases and the indeterminate value is produced as output. Second, the functional relationship is invoked to find the output tuple corresponding to the input tuple, if any. If the functional relationship is defined by an algorithm, this means that the algorithm is carried out to produce the corresponding tuple of outputs. If this process fails, the evaluation process ceases and the indeterminate value is produced as output. Third, the tuple of outputs is tested to determine whether or not it is in the range space of the function. If it is not, the indeterminate value is produced as output. Otherwise, the values produced by the second stage are the outputs.

NOTE 1 The price of explicitly providing information about domains and ranges of functions is the need to ensure its validity by incorporating it in the evaluation process.

NOTE 2 Testing a value for membership in a mathematical space may itself require the evaluation of explicit or implicit functions in a recursive manner.

NOTE 3 The general problem of detecting evaluation processes which will not terminate is known to be unsolvable. It is in the self interest of all users to avoid creating or communicating functions whose evaluation does not terminate.

4.2.13 Function application

Function application is the process of forming an expression from a function and an appropriate set of inputs. The term may also be used for an expression resulting from the process. A function application expression denotes the first output from the evaluation of the function on those inputs.

NOTE In most contexts, the term “function” is restricted to algorithms which only have one output, and “first output” is synonymous with “the output”. This part of ISO 10303 uses the term in the wider sense which includes algorithms with multiple outputs, and must therefore adopt this refinement to harmonize the computer programming situation with the more traditional one.

4.3 Mathematical functions schema constant definitions

The following constants are used repeatedly in the remainder of the formal specification of this part of ISO 10303.

NOTE Implementor information: Since schema constants cannot be referenced from ISO 10303-21 files, these constants are only useful within the formal specifications. Instantiations of product data models must reproduce these constant instances as needed. Only one instance of any constant is ever required in a model, although it does no harm if an implementation generates several.

EXPRESS specification:

*)
CONSTANT

(*

4.3.1 schema_prefix

This string constant defines a frequently needed prefix for type names.

EXPRESS specification:

```
*)
  schema_prefix : STRING := 'MATHEMATICAL_FUNCTIONS_SCHEMA.';
(*)
```

4.3.2 the elementary spaces

These instances of **elementary_space** represent unique and frequently referenced mathematical spaces.

EXPRESS specification:

```
*)
  the_integers      : elementary_space := make_elementary_space(es_integers);
  the_reals         : elementary_space := make_elementary_space(es_reals);
  the_complex_numbers : elementary_space := make_elementary_space(es_complex_numbers);
  the_numbers       : elementary_space := make_elementary_space(es_numbers);
  the_logicals      : elementary_space := make_elementary_space(es_logicals);
  the_booleans      : elementary_space := make_elementary_space(es_booleans);
  the_strings       : elementary_space := make_elementary_space(es_strings);
  the_binarys       : elementary_space := make_elementary_space(es_binarys);
  the_maths_spaces  : elementary_space := make_elementary_space(es_maths_spaces);
  the_generics      : elementary_space := make_elementary_space(es_generics);
(*)
```

4.3.3 the empty space

An instance of **finite_space** representing the unique mathematical space with no elements is frequently referenced.

EXPRESS specification:

```
*)
  the_empty_space : finite_space := make_finite_space([]);
(*)
```

4.3.4 real intervals

These instance of **maths_space** represent unique and frequently referenced intervals of real numbers.

EXPRESS specification:

```

*)
the_nonnegative_reals      : real_interval_from_min :=
  make_real_interval_from_min(0.0, closed);
the_zero_one_interval      : finite_real_interval := make_finite_real_interval(
  0.0, closed, 1.0, closed);
the_zero_pi_interval       : finite_real_interval := make_finite_real_interval(
  0.0, closed, pi, closed);
the_neg1_one_interval      : finite_real_interval := make_finite_real_interval(
  -1.0, closed, 1.0, closed);
the_neghalfpi_halfpi_interval : finite_real_interval := make_finite_real_interval(
  -0.5*pi, closed, 0.5*pi, closed);
the_negpi_pi_interval      : finite_real_interval := make_finite_real_interval(
  -pi, open, pi, closed);
(*

```

4.3.5 tuple spaces

These instances of **tuple_space** represent unique and frequently referenced mathematical tuple spaces.

EXPRESS specification:

```

*)
the_zero_tuple_space : listed_product_space := make_listed_product_space([]);
the_tuples           : extended_tuple_space := make_extended_tuple_space(
  the_zero_tuple_space, the_generics);
the_integer_tuples   : extended_tuple_space := make_extended_tuple_space(
  the_zero_tuple_space, the_integers);
the_real_tuples      : extended_tuple_space := make_extended_tuple_space(
  the_zero_tuple_space, the_reals);
the_complex_tuples   : extended_tuple_space := make_extended_tuple_space(
  the_zero_tuple_space, the_complex_numbers);
(*

```

4.3.6 empty values

These special “empty” values of **maths_value** are frequently referenced.

EXPRESS specification:

```

*)
the_empty_maths_tuple      : maths_tuple := [];
the_empty_maths_value      : maths_value := the_empty_maths_tuple;
the_empty_atom_based_tuple : atom_based_tuple := [];
the_empty_atom_based_value : atom_based_value := the_empty_atom_based_tuple;
(*

```

EXPRESS specification:

```
*)
END_CONSTANT;
(*
```

NOTE Since entity instance identity has almost no significance in this schema, the repeated use of the same constant entity instance is acceptable and reduces overhead.

4.4 Mathematical functions schema type definitions

4.4.1 nonnegative_integer

This type provides a named type for the nonnegative integers.

EXPRESS specification:

```
*)
TYPE nonnegative_integer = INTEGER;
WHERE nonnegativity: SELF >= 0;
END_TYPE;
(*
```

4.4.2 positive_integer

This type provides a named type for the positive integers.

EXPRESS specification:

```
*)
TYPE positive_integer = nonnegative_integer;
WHERE positivity: SELF > 0;
END_TYPE;
(*
```

4.4.3 zero_or_one

This subset type of the nonnegative integers is used to indicate whether the derived standard indexing for a table function should start from zero or from one.

EXPRESS specification:

```
*)
TYPE zero_or_one = nonnegative_integer;
WHERE in_range: (SELF = 0) OR (SELF = 1);
END_TYPE;
(*
```

4.4.4 one_or_two

This subset type of the positive integers is used to indicate whether the summation index for a linear function is one or two.

EXPRESS specification:

```
*)
TYPE one_or_two = positive_integer;
WHERE in_range: (SELF = 1) OR (SELF = 2);
END_TYPE;
(*
```

4.4.5 local names for simple types

These type definitions provide named types for the EXPRESS simple types.

NOTE In order to create the “generic” type **maths_value** (see 4.4.11) and other similar “supertypes” using the EXPRESS select construct, named types must be defined for the EXPRESS simple types.

EXPRESS specification:

```
*)
TYPE maths_number = NUMBER;
END_TYPE;
TYPE maths_real = REAL;
END_TYPE;
TYPE maths_integer = INTEGER;
END_TYPE;
TYPE maths_logical = LOGICAL;
END_TYPE;
TYPE maths_boolean = BOOLEAN;
END_TYPE;
TYPE maths_string = STRING;
END_TYPE;
TYPE maths_binary = BINARY;
END_TYPE;
(*
```

4.4.6 maths_simple_atom

This type encompasses any EXPRESS simple value.

EXPRESS specification:

```
*)
TYPE maths_simple_atom = SELECT
  (maths_number,
   maths_real,
   maths_number,
   maths_logical,
```

```

    maths_boolean,
    maths_string,
    maths_binary);
END_TYPE;
(*)

```

4.4.7 maths_atom

This type encompasses any EXPRESS simple value and any enumeration item of any EXPRESS enumeration type defined in this schema. It is a foundation type for the recursive constructions for types **atom_based_value** and **maths_value**.

EXPRESS specification:

```

*)
TYPE maths_atom = SELECT
    (maths_simple_atom,
     maths_enum_atom);
END_TYPE;
(*)

```

4.4.8 atom_based_tuple

This type provides representations for arbitrary finite tuples of **atom_based_value** values.

EXPRESS specification:

```

*)
TYPE atom_based_tuple = LIST OF atom_based_value;
END_TYPE;
(*)

```

4.4.9 atom_based_value

This type represents all values constructed from **maths_atom** values by recursive formations of tuples. The EXPRESS representation is arbitrarily nested lists of atoms or the atoms themselves.

EXPRESS specification:

```

*)
TYPE atom_based_value = SELECT
    (maths_atom,
     atom_based_tuple);
END_TYPE;
(*)

```

NOTE Since there is no mechanism for self reference, the values are necessarily finite and acyclic. Note, in particular, that no entity instances can participate in values of this type.

4.4.10 maths_tuple

This type represents the mathematical ordered tuple.

NOTE 1 The semantics of mathematical ordered tuple differs subtly from that of the EXPRESS list aggregate in that there are no notions of creation or destruction or element insertion or removal operations.

NOTE 2 In the context of formal mathematics, there is a single object which can be called the zero-tuple. It is naturally represented by a zero-length list. This object is permitted in the type **maths_tuple**, but not in the other tuple types.

EXPRESS specification:

```
*)
TYPE maths_tuple = LIST [0:?] OF maths_value;
END_TYPE;
(*
```

4.4.11 maths_value

This type is intended to encompass any data value which could be an input to a mathematical function, an output from a mathematical function, or useful in defining a mathematical function.

NOTE The EXPRESS type "generic" would be appropriate if it were allowed. Instead, applications needing to use data values not belonging to this type will have to define additional subtypes of the supertypes provided in this schema.

EXPRESS specification:

```
*)
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;
(*
```

Formal propositions:

constancy: If a **maths_value** is a **generic_expression** then it shall denote a constant mathematical object.

4.4.12 maths_expression

This type is intended to encompass any expression recognizable as mathematical.

EXPRESS specification:

```

*)
TYPE maths_expression = SELECT
    (atom_based_value,
     maths_tuple,
     generic_expression);
END_TYPE;
(*

```

4.4.13 maths_function_select

This type enables the use of either **elementary_function_enumerators** values or **maths_function** instances to reference a mathematical function in certain attributes of **parallel_composed_function** and **function_application**.

NOTE This select type enables a more efficient representation of an enormous variety of mathematical functions. In particular, it reduces the number of entity instances required.

EXPRESS specification:

```

*)
TYPE maths_function_select = SELECT
    (maths_function,
     elementary_function_enumerators);
END_TYPE;
(*

```

4.4.14 input_selector

This type of **positive_integer** adds the special semantics of denoting a particular input of a function of several inputs, or a particular component of the first input when there is only one input and that input is a tuple. The positive integer value indicates the ordinal position of the selected input in the list of inputs or in the list of components of the sole tuple input. Values larger than the number of inputs or number of components, respectively, shall be prevented by rules in each context using this type.

NOTE 1 It is normally the case that either all the inputs to a function have been prepackaged as a single input tuple, or the function has multiple inputs, or the single input is not a tuple. If there is a single input that is a tuple and it is desired to select that single input rather than one of its components, then the desired selector function is just an identity function on that domain, and an appropriate instance of **restriction_function** may be used instead.

NOTE 2 This type is used by **selector_function**, **partial_derivative_function**, and **definite_integral_function**.

EXPRESS specification:

```

*)
TYPE input_selector = positive_integer;
END_TYPE;
(*

```

4.4.15 elementary_space_enumerators

This enumeration type provides unique nominal values to represent the mathematical spaces associated with the EXPRESS simple types, the complex numbers and the generic type.

EXPRESS specification:

```
*)
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;
(*
```

Enumerated item definitions:

es_numbers: Enumeration item used to identify the mathematical space of all numbers.

es_complex_numbers: Enumeration item used to identify the mathematical space of all complex numbers.

es_reals: Enumeration item used to identify the mathematical space of all real numbers.

es_integers: Enumeration item used to identify the mathematical space of all integers.

es_logicals: Enumeration item used to identify the mathematical space of all EXPRESS type LOGICAL values.

es_booleans: Enumeration item used to identify the mathematical space of all EXPRESS type BOOLEAN values.

es_strings: Enumeration item used to identify the mathematical space of all EXPRESS type STRING values.

es_binarys: Enumeration item used to identify the mathematical space of all EXPRESS type BINARY values.

es_maths_spaces: Enumeration item used to identify the mathematical space of all mathematical spaces representable by instances of **maths_space**.

es_maths_functions: Enumeration item used to identify the mathematical space of all mathematical functions representable by instances of **maths_function**.

es_generics: Enumeration item used to identify the mathematical space of all mathematical values representable using this schema.

NOTE The “space of all numbers” consists of all mathematical values corresponding to EXPRESS type NUMBER values and the complex numbers.

4.4.16 ordering_type

This enumeration type indicates which of the two most natural linear "ascending" orderings on the members of a subscript space is to be used. The enumeration identifier **by_rows** indicates a lexicographic order in which the first subscript is most significant and the last subscript varies most rapidly. The enumeration identifier **by_columns** indicates an order in which the last subscript is most significant and the first subscript varies most rapidly.

NOTE The enumeration identifiers were chosen to be meaningful in the most common case, that of subscript pairs for matrices. Traditionally, the first index indicates the row position and the second the column position. Languages which define multi-dimensional arrays by a recursive application of a one-dimensional array construct imply the **by_rows** ordering. The FORTRAN programming language specified the **by_columns** ordering.

EXPRESS specification:

```
*)
TYPE ordering_type = ENUMERATION OF
    (by_rows,
     by_columns);
END_TYPE;
(*
```

Enumerated item definitions:

by_rows: This enumeration value indicates that the subscript tuples are in ascending left-to-right-lexicographic order with the last subscript varying most rapidly.

by_columns: This enumeration value indicates that the subscript tuples are in ascending right-to-left-lexicographic order with the first subscript varying most rapidly.

EXAMPLE Consider the twelve subscript tuples for a 2 by 2 by 3 table indexed from 1. Table 2 shows the two linear orderings of these triples indicated by values **by_rows** and **by_columns**.

Table 2 – Orderings indicated by ordering_type

Ordinal	by_rows	by_columns
1	[1,1,1]	[1,1,1]
2	[1,1,2]	[2,1,1]
3	[1,1,3]	[1,2,1]
4	[1,2,1]	[2,2,1]
5	[1,2,2]	[1,1,2]
6	[1,2,3]	[2,1,2]
7	[2,1,1]	[1,2,2]
8	[2,1,2]	[2,2,2]
9	[2,1,3]	[1,1,3]
10	[2,2,1]	[2,1,3]
11	[2,2,2]	[1,2,3]
12	[2,2,3]	[2,2,3]

4.4.17 lower_upper

This enumeration type is used to indicate whether the elements below the main diagonal of a matrix are being considered or the elements above. In other words, whether the ordinal positions with the row ordinal greater than or equal to the column ordinal are being considered or the ordinal positions with the row ordinal less than or equal to the column ordinal.

When used in a higher-dimensional context, the enumeration value **lower** indicates the set of ordinal positions for which the ordinal indices are non-increasing, and the enumeration value **upper** indicates the set of ordinal positions for which the ordinal indices are non-decreasing.

NOTE Other context information in a given use of this type may further restrict the set of positions by changing all the inequalities to strict inequalities. Such restrictions are described by using the terms “strictly lower” and “strictly upper”.

EXPRESS specification:

```
*)
TYPE lower_upper = ENUMERATION OF
  (lower,
   upper);
END_TYPE;
(*
```

Enumerated item definitions:

lower: The lower triangle of a matrix is to be considered. That is, the elements which have their row ordinal greater than or equal to their column ordinal.

upper: The upper triangle of a matrix is to be considered. That is, the elements which have their row ordinal less than or equal to their column ordinal.

4.4.18 symmetry_type

This enumeration type is used to indicate the kind of symmetry present in a symmetric matrix. Values indicating identity symmetry, skew symmetry, Hermitian symmetry and skew-Hermitian symmetry are defined.

EXPRESS specification:

```
*)
TYPE symmetry_type = ENUMERATION OF
  (identity,
   skew,
   hermitian,
   skew_hermitian);
END_TYPE;
(*
```

Enumerated item definitions:

identity: The element in position $[j, k]$ is the same as the element in position $[k, j]$.

skew: The element in position $[j, k]$ is the numeric negative of the element in position $[k, j]$.

hermitian: The element in position $[j, k]$ is the complex conjugate of the element in position $[k, j]$.

skew_hermitian: The element in position $[j, k]$ is the negative complex conjugate of the element in position $[k, j]$.

4.4.19 elementary_function_enumerators

This enumeration type provides unique nominal values identifying a wide collection of elementary mathematical functions and operators. This type is used in constructing expressions and **elementary_function** instances. The enumeration items denoting trigonometric functions and their inverses all assume angle measures in radians.

The naming convention is to use a suffix to distinguish the argument type(s) for those functions or operations which occur in more than one type domain. The suffix “_i” indicates the operation is applied to integers, the suffix “_r” indicates the operation is applied to real numbers, the suffix “_c” indicates the operation is applied to complex numbers, the suffix “_s” indicates the operation is applied to strings, the suffix “_b” indicates the operation is applied to binaries, the suffix “_t” indicates the operation is applied to tuples, the suffix “_it” indicates the operation is applied to integer tuples, the suffix “_rt” indicates the operation is applied to real tuples, and the suffix “_ct” indicates the operation is applied to complex number tuples. In the case of exponentiation, the integer exponent case, which can be interpreted purely in terms of repeated multiplication and division, has been distinguished using suffixes “_i”, “_ri” and “_ci”, for the cases of integer, real and complex numbers raised to integral powers, with results in the same space as the base.

The six comparison operators are abbreviated in the enumeration item names as follows: “eq” abbreviates “equal” (=), “ne” abbreviates “not equal” (<> or $\neq$), “gt” abbreviates “greater than” (>), “lt” abbreviates “less than” (<), “ge” abbreviates “greater than or equal” (>= or $\geq$), and “le” abbreviates “less than or equal” (<= or $\leq$).

NOTE 1 The only difference between functions and operators is in the notation that is usually used. For the purposes of this schema, the two terms are synonymous.

NOTE 2 Many of these enumeration items correspond to EXPRESS language functions, procedures or operations. Many of the EXPRESS operators are “overloaded”, that is, they are unions of several mathematical functions defined on different domains. For the purposes of this schema, the individual operators are identified.

NOTE 3 Consider the following differences in the exponentiation operations: Two to the power minus two, using the function indicated by **ef.exponentiate_i**, is the integer zero obtained from the integer division of one by four. Two to the power minus two, using the function indicated by **ef.exponentiate_r**, is the real number 0.25 exactly. Two to the power minus two, using the function indicated by **ef.exponentiate_r**, is a real number which is very close to 0.25, but which might not be exactly 0.25 on all systems because of computational round-off error.

NOTE 4 The EXPRESS procedures **insert** and **remove** each have only one VAR parameter, so they correspond directly to functions. The notion that there is some sense in which the input and output aggregates are the same “instance” is not recognized in the mathematical function.

NOTE 5 The EXPRESS aggregate operators - intersection, union, difference, subset, and superset -

rely in part on the EXPRESS “in” operator, which relies on instance equality rather than value equality. Similarly, the “rolesof” function and the “usedin” function depend explicitly on instance equality, and on an implicit current population of instances. Consequently, they do not correspond to mathematical functions (in the sense used in this schema). The “hibound” and “lobound” functions depend on the type definitions in a schema rather than on the intrinsic structure of a data value, and have also been omitted.

NOTE 6 A useful extension to this part of ISO 10303 would describe a mathematical model of an EXPRESS-based computational environment and an EXPRESS representation thereof. Such an extension would support notions of entity instances and instance identity within a specified schema context and model state and would support the communication of functions defined on model states.

EXPRESS specification:

*)

```

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_r_i,
   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_c_i, 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;
(*)

```

Enumerated item definitions:

ef_and: Enumeration member denoting the EXPRESS logical and operation.

ef_or: Enumeration member denoting the EXPRESS logical or operation.

ef_not: Enumeration member denoting the EXPRESS logical not operation.

ef_xor: Enumeration member denoting the EXPRESS logical xor operation.

ef_negate_i: Enumeration member denoting the integer negation operation.

ef_add_i: Enumeration member denoting the integer addition operation.

ef_subtract_i: Enumeration member denoting the integer subtraction operation.

ef_multiply_i: Enumeration member denoting the integer multiplication operation.

ef_divide_i: Enumeration member denoting the integer division operation. This is the same as the EXPRESS “div” operation.

ef_mod_i: Enumeration member denoting the integer modulo operation. This is the same as the EXPRESS “mod” operation.

ef_exponentiate_i: Enumeration member denoting the integer exponentiation operation implemented using integer multiplication and division.

ef_eq_i: Enumeration member denoting the integer equality relation.

ef_ne_i: Enumeration member denoting the integer inequality relation.

ef_gt_i: Enumeration member denoting the integer greater than relation.

ef_lt_i: Enumeration member denoting the integer less than relation.

ef_ge_i: Enumeration member denoting the integer greater than or equal relation.

ef_le_i: Enumeration member denoting the integer less than or equal relation.

ef_abs_i: Enumeration member denoting the integer absolute value function.

ef_max_i: Enumeration member denoting the integer maximum value function.

ef_min_i: Enumeration member denoting the integer minimum value function.

ef_if_i: Enumeration member denoting the integer ternary conditional operation (as in the C programming language).

ef_negate_r: Enumeration member denoting the real negation operation.

ef_reciprocal_r: Enumeration member denoting the real reciprocal operation.

ef_add_r: Enumeration member denoting the real addition operation.

ef_subtract_r: Enumeration member denoting the real subtraction operation.

ef_multiply_r: Enumeration member denoting the real multiplication operation.

ef_divide_r: Enumeration member denoting the real division operation.

ef_mod_r: Enumeration member denoting the real modulo operation. For real numbers a and $b \neq 0$, $a \bmod b \equiv a - \lfloor a/b \rfloor * b$, where $\lfloor x \rfloor$ is the greatest integer less than or equal to x .

ef_exponentiate_r: Enumeration member denoting the real exponentiation operation.

ef_exponentiate_ri: Enumeration member denoting exponentiation operation restricted to reals raised to integer powers.

ef_eq_r: Enumeration member denoting the real equality relation.

ef_ne_r: Enumeration member denoting the real inequality relation.

ef_gt_r: Enumeration member denoting the real greater than relation.

ef_lt_r: Enumeration member denoting the real less than relation.

ef_ge_r: Enumeration member denoting the real greater than or equal relation.

ef_le_r: Enumeration member denoting the real less than or equal relation.

ef_abs_r: Enumeration member denoting the real absolute value function.

ef_max_r: Enumeration member denoting the real maximum value function.

ef_min_r: Enumeration member denoting the real minimum value function.

ef_acos_r: Enumeration member denoting the real inverse cosine function.

ef_asin_r: Enumeration member denoting the real inverse sine function.

ef_atan2_r: Enumeration member denoting the real four-quadrant inverse tangent function. Its two arguments y and x , which shall not both be zero, determine a point (x, y) in the real plane. The signed angle θ between the positive x -axis and the ray from the origin through that point, measured in radians and lying in the range $-\pi < \theta \leq \pi$ shall be returned. When the first input, y , is positive, the output is positive.

ef_cos_r: Enumeration member denoting the real cosine function.

ef_exp_r: Enumeration member denoting the real exponential function.

ef_ln_r: Enumeration member denoting the real natural logarithm function.

ef_log2_r: Enumeration member denoting the real logarithm base two function.

ef_log10_r: Enumeration member denoting the real logarithm base ten function.

ef_sin_r: Enumeration member denoting the real sine function.

ef_sqrt_r: Enumeration member denoting the real square root function.

ef_tan_r: Enumeration member denoting the real tangent function.

ef_if_r: Enumeration member denoting the real ternary conditional operation (as in the C programming language).

ef_form_c: Enumeration member denoting the function which forms a complex number from two reals representing the real and imaginary parts.

ef_rpart_c: Enumeration member denoting the function which returns the real part of a complex number.

ef_ipart_c: Enumeration member denoting the function which returns the imaginary part of a complex number.

ef_negate_c: Enumeration member denoting the complex negation operation.

ef_reciprocal_c: Enumeration member denoting the complex reciprocal operation.

ef_add_c: Enumeration member denoting the complex addition operation.

ef_subtract_c: Enumeration member denoting the complex subtraction operation.

ef_multiply_c: Enumeration member denoting the complex multiplication operation.

ef_divide_c: Enumeration member denoting the complex division operation.

ef_exponentiate_c: Enumeration member denoting the complex exponentiation operation.

ef_exponentiate_ci: Enumeration member denoting exponentiation operation restricted to complexes raised to integer powers.

ef_eq_c: Enumeration member denoting the complex equality relation.

ef_ne_c: Enumeration member denoting the complex inequality relation.

ef_conjugate_c: Enumeration member denoting the complex conjugate operation.

- ef_abs_c:** Enumeration member denoting the complex absolute value function.
- ef_arg_c:** Enumeration member denoting the complex argument function, which shall be defined by $arg(x + yi) \equiv atan2(y, x)$.
- ef_cos_c:** Enumeration member denoting the complex cosine function.
- ef_exp_c:** Enumeration member denoting the complex exponential function.
- ef_ln_c:** Enumeration member denoting the complex natural logarithm function.
- ef_sin_c:** Enumeration member denoting the complex sine function.
- ef_sqrt_c:** Enumeration member denoting the complex square root function producing the square root with positive real part, or, if the real parts are zero, the root with nonnegative imaginary part.
- ef_tan_c:** Enumeration member denoting the complex tangent function.
- ef_if_c:** Enumeration member denoting the complex ternary conditional operation (analogous to the corresponding real operation).
- ef_subscript_s:** Enumeration member denoting the EXPRESS string indexing operation with one index.
- ef_eq_s:** Enumeration member denoting the string equality relation.
- ef_ne_s:** Enumeration member denoting the string inequality relation.
- ef_gt_s:** Enumeration member denoting the string greater than relation.
- ef_lt_s:** Enumeration member denoting the string less than relation.
- ef_ge_s:** Enumeration member denoting the string greater than or equal relation.
- ef_le_s:** Enumeration member denoting the string less than or equal relation.
- ef_subsequence_s:** Enumeration member denoting the EXPRESS string indexing operation with two indices.
- ef_concat_s:** Enumeration member denoting the string concatenation operation.
- ef_size_s:** Enumeration member denoting the string size operation.
- ef_format:** Enumeration member denoting the EXPRESS formatting function.
- ef_value:** Enumeration member denoting the EXPRESS string to number conversion function.
- ef_like:** Enumeration member denoting the EXPRESS string pattern matching operation.
- ef_if_s:** Enumeration member denoting the string ternary conditional operation (analogous to the corresponding real operation).
- ef_subscript_b:** Enumeration member denoting the EXPRESS binary indexing operation with one index.
- ef_eq_b:** Enumeration member denoting the binary equality relation.
- ef_ne_b:** Enumeration member denoting the binary inequality relation.
- ef_gt_b:** Enumeration member denoting the binary greater than relation.
- ef_lt_b:** Enumeration member denoting the binary less than relation.
- ef_ge_b:** Enumeration member denoting the binary greater than or equal relation.
- ef_le_b:** Enumeration member denoting the binary less than or equal relation.

ef_subsequence_b: Enumeration member denoting the EXPRESS binary indexing operation with two indices.

ef_concat_b: Enumeration member denoting the binary concatenation operation.

ef_size_b: Enumeration member denoting the binary size operation.

ef_if_b: Enumeration member denoting the binary ternary conditional operation (analogous to the corresponding real operation).

ef_subscript_t: Enumeration member denoting the tuple indexing operation.

ef_eq_t: Enumeration member denoting the tuple equality relation.

ef_ne_t: Enumeration member denoting the tuple inequality relation.

ef_concat_t: Enumeration member denoting the tuple concatenation operation.

ef_size_t: Enumeration member denoting the tuple size operation.

ef_entuple: Enumeration member denoting the entuple function, that is, the function which collects its input arguments into an ordered tuple and delivers that tuple as its single output.

ef_detuple: Enumeration member denoting the operation returning the all the members of an ordered tuple as explicit independent outputs.

ef_insert: Enumeration member denoting the function version of the EXPRESS insert in list procedure.

ef_remove: Enumeration member denoting the function version of the EXPRESS remove from list procedure.

ef_if_t: Enumeration member denoting the tuple ternary conditional operation (analogous to the corresponding real operation).

ef_sum_it: Enumeration member denoting the operation which sums the members of an integer tuple to produce an integer result.

ef_product_it: Enumeration member denoting the operation which multiplies the members of an integer tuple to produce an integer result.

ef_add_it: Enumeration member denoting the integer tuple addition operation.

ef_subtract_it: Enumeration member denoting the integer tuple subtraction operation.

ef_scalar_mult_it: Enumeration member denoting the integer tuple scalar multiplication operation.

ef_dot_prod_it: Enumeration member denoting the integer tuple dot product operation.

ef_sum_rt: Enumeration member denoting the operation which sums the members of a real tuple to produce a real result.

ef_product_rt: Enumeration member denoting the operation which multiplies the members of a real tuple to produce a real result.

ef_add_rt: Enumeration member denoting the real tuple addition operation.

ef_subtract_rt: Enumeration member denoting the real tuple subtraction operation.

ef_scalar_mult_rt: Enumeration member denoting the real tuple scalar multiplication operation.

ef_dot_prod_rt: Enumeration member denoting the real tuple dot product operation.

ef_norm_rt: Enumeration member denoting the real tuple norm operation.

ef_sum_ct: Enumeration member denoting the operation which sums the members of a complex tuple to produce a complex number result.

ef_product_ct: Enumeration member denoting the operation which multiplies the members of a complex tuple to produce a complex number result.

ef_add_ct: Enumeration member denoting the complex tuple addition operation.

ef_subtract_ct: Enumeration member denoting the complex tuple subtraction operation.

ef_scalar_mult_ct: Enumeration member denoting the complex tuple scalar multiplication operation.

ef_dot_prod_ct: Enumeration member denoting the complex tuple dot product operation.

ef_norm_ct: Enumeration member denoting the complex tuple norm operation.

ef_if: Enumeration member denoting the generic ternary conditional operation (analogous to the corresponding real operation).

ef_ensemble: Enumeration member denoting the operation which forms a **finite_space** from its operands.

ef_member_of: Enumeration member denoting the operation which tests a mathematical value for membership in a mathematical space.

4.4.20 open_closed

This enumeration type is used to indicate whether real intervals are topologically open or closed at their endpoints. If the real interval is closed at an endpoint, the endpoint is a member of the real interval; otherwise, the endpoint is not a member of the real interval.

EXPRESS specification:

```
*)
TYPE open_closed = ENUMERATION OF
    (open,
     closed);
END_TYPE;
(*
```

Enumerated item definitions:

open: Indicator that the real interval is topologically open at the associated endpoint, that is, that the boundary point is not a member of the interval.

closed: Indicator that the real interval is topologically closed at the associated endpoint, that is, that the boundary point is a member of the interval.

4.4.21 space_constraint_type

This enumeration type is used to indicate which of three possible constraints on a mathematical space is to be used in a given context. The possibilities are that one space is equal to another, a subspace of another, or a member of another.

EXPRESS specification:

```
*)
TYPE space_constraint_type = ENUMERATION OF
  (sc_equal,
   sc_subspace,
   sc_member);
END_TYPE;
(*
```

Enumerated item definitions:

sc_equal: Indicator that the subject space shall be equal to the given argument space.

sc_subspace: Indicator that the subject space shall be a subspace, but not necessarily a proper subspace, of the given argument space.

sc_member: Indicator that the subject space shall be member of the given argument space.

4.4.22 repackaging_options

This enumeration type is used by **repackaging_function** instances to identify repackaging options for inputs and outputs.

EXPRESS specification:

```
*)
TYPE repackaging_options = ENUMERATION OF
  (ro_nochange,
   ro_wrap_as_tuple,
   ro_unwrap_tuple);
END_TYPE;
(*
```

Enumerated item definitions:

ro_nochange: Indicator that no repackaging is to occur.

ro_wrap_as_tuple: Indicator that multiple values are to be repackaged as a single tuple of values.

ro_unwrap_tuple: Indicator that a single tuple value is to be repackaged as multiple values.

4.4.23 extension_options

This enumeration type identifies certain common mathematical mechanisms for extending the exact domain of a function in an unambiguous way to boundary and other technically singular inputs.

NOTE This type is used by **partial_derivative_function** instances to identify which of certain common extensions to the function determined by the strictest notion of differentiability are to be made.

EXPRESS specification:

```

*)
TYPE extension_options = ENUMERATION OF
    (eo_none,
     eo_cont,
     eo_cont_right,
     eo_cont_left);
END_TYPE;
(*

```

Enumerated item definitions:

eo_none: Indicator that no extension shall occur.

eo_cont: Indicator that extension by continuity shall occur. Wherever an unambiguous value can be computed which preserves continuity, the function shall be defined there and output that value.

eo_cont_right: Indicator that extension by continuity shall occur and, when and where continuity to the right is meaningful, extension by continuity to the right shall occur.

eo_cont_left: Indicator that extension by continuity shall occur and, when and where continuity to the left is meaningful, extension by continuity to the left shall occur.

NOTE Extension by continuity to the right or left is only meaningful in the context of the real numbers. When extension by continuity occurs, the value produced is necessarily the same as the values produced by continuity to the right or continuity to the left.

4.4.24 maths_enum_atom

This select type collects the enumeration types defined in this schema which contain mathematical values and which are not otherwise included in the type used to represent mathematical values.

EXPRESS specification:

```

*)
TYPE maths_enum_atom = SELECT
    (elementary_space_enumerators,
     ordering_type,
     lower_upper,
     symmetry_type,
     elementary_function_enumerators,
     open_closed,
     space_constraint_type,
     repackage_options,
     extension_options);
END_TYPE;
(*

```

4.4.25 dotted_express_identifier

This type of **STRING** includes the strings which could syntactically be qualified type names or attribute names in EXPRESS. That is, it includes strings which are composed of one or more EXPRESS identifiers separated by period characters.

EXPRESS specification:

```
*)
TYPE dotted_express_identifier = STRING;
WHERE syntax: dotted_identifiers_syntax(SELF);
END_TYPE;
(*
```

Formal propositions:

syntax: The string shall have the form of a sequence of EXPRESS identifiers separated by full stops.

4.4.26 express_identifier

This type of **dotted_express_identifier** includes only the strings which could syntactically be EXPRESS identifiers. In particular, a value of this type shall begin with a letter and contain only letters, digits and low lines.

EXPRESS specification:

```
*)
TYPE express_identifier = dotted_express_identifier;
WHERE syntax: dot_count(SELF) = 0;
END_TYPE;
(*
```

Formal propositions:

syntax: The string shall begin with a letter and contain only letters, digits, and low lines.

NOTE In order that **express_identifier** be a subtype of **dotted_express_identifier**, the rule is actually defined by requiring the string to be an instance of **dotted_express_identifier** which contains no full stops.

4.4.27 product_space

A **product_space** is a selection from among the entity types representing finite Cartesian product spaces.

EXPRESS specification:

```

*)
TYPE product_space = SELECT
    (uniform_product_space,
     listed_product_space);
END_TYPE;
(*

```

4.4.28 tuple_space

A **tuple_space** is a selection from among the types representing mathematical spaces whose elements are ordered tuples. This values of this type are those which could occur as domains or ranges of mathematical functions.

NOTE Specifically, the values of this type represent finite Cartesian products of mathematical spaces, and certain infinite unions of such Cartesian products. The latter case arises for functions with an indefinite number of input or output arguments.

EXPRESS specification:

```

*)
TYPE tuple_space = SELECT
    (product_space,
     extended_tuple_space);
END_TYPE;
(*

```

4.4.29 maths_space_or_function

A **maths_space_or_function** is a selection of either a **maths_space** or a **maths_function**.

NOTE This type describes the two choices for the source of the information for determining the domain of a **constant_function**, **selector_function**, or **parallel_composed_function**.

EXPRESS specification:

```

*)
TYPE maths_space_or_function = SELECT
    (maths_space,
     maths_function);
END_TYPE;
(*

```

4.4.30 real_interval

A **real_interval** is a selection from among the entity types representing non-empty, non-trivial, real intervals.

NOTE The empty interval and the single point interval are not representable by this type. All other real intervals have a value-unique representation in this type.

EXPRESS specification:

```

*)
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;
(*)

```

Formal propositions:

WR1: If the select value is an **elementary_space**, it is the one representing the space of all real numbers.

4.5 Mathematical functions schema entity definitions

4.5.1 quantifier_expression

This abstract type of **multiple_arity_generic_expression** distinguishes those expressions which are quantifier expressions. Certain of the operands of a quantifier expression are restricted to variables. These variables are said to be bound by the quantifier and their semantics is determined by the quantifier. The variables which are bound by a quantifier are not available for substitution operations, that is, they may not be subsequently tied to particular constant values. The semantics of the quantifier expression depends on all the possible values of each of its bound variables rather than varying with the association of variables to particular values.

NOTE 1 This type together with entity type **bound_variable_semantics**, function **free_variables_of** and **expression_is_constant** generically extend the **ISO13584_generic_expressions_schema** to handle operations which involve logical quantifiers.

NOTE 2 Function **has_values_space** identifies which generic expressions pertain to the **mathematical_functions_schema** and, consequently, have associated spaces of possible values given by function **values_space_of**.

EXPRESS specification:

```

*)
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;
(*)

```

Attribute definitions:

variables: The list of variables bound by the quantifier operation.

Formal propositions:

WR1: The variables listed in attribute **variables** are among the operands of the operation.

WR2: The variables listed in attribute **variables** have the semantics of bound variables.

4.5.2 dependent_variable_definition

This subtype of **unary_generic_expression** defines a dependent variable. From the viewpoint of the formal language of mathematical expressions, a dependent variable is merely an abbreviation for a subexpression which is being given special attention because of its significance or frequency of occurrence in other expressions.

NOTE Observe that a dependent variable is not a variable at all! In particular, it is not a logical variable, may not be quantified, may not be assigned a value (other than that resulting from the defining expression), and will not appear in lists of used or free variables. It has significance for rendering operations, which should use the associated name instead of rendering the defining expression when a reference to the **dependent_variable_definition** instance is used in another expression, and it has significance for computational operations, which may improve efficiency of computation by saving and reusing its value under appropriate circumstances, but it has no effect on the meaning of the defining expression as a formal expression.

EXPRESS specification:

```
*)
ENTITY dependent_variable_definition
  SUBTYPE OF (unary_generic_expression);
  name      : label;
  description : text;
END_ENTITY;
(*
```

Attribute definitions:

SELF\unary_generic_expression.operand: The expression defining the dependent variable.

name: The label identifying the dependent variable.

description: Descriptive information for the dependent variable or defining expression.

NOTE The description may also include information about the preferred rendering of the dependent variable or the defining expression in some presentation system such as TeX or XML. Of course, only certain receiving systems would be able to make use of such information.

4.5.3 bound_variable_semantics

This type of **variable_semantics** indicates that a variable is a bound variable whose semantics is determined by the quantifier expression which references the variable in its **variables** attribute.

EXPRESS specification:

```
*)
ENTITY bound_variable_semantics
  SUBTYPE OF (variable_semantics);
END_ENTITY;
(*
```

Informal propositions:

IP1: Only those variables which are listed in the **variables** attribute of a quantifier expression may have this semantics.

4.5.4 free_variable_semantics

This type of **variable_semantics** indicates that a variable has no further semantics beyond what is implied by its use in an expression. This could occur when the expression itself is the object of interest.

EXPRESS specification:

```
*)
ENTITY free_variable_semantics
  SUBTYPE OF (variable_semantics);
END_ENTITY;
(*
```

4.5.5 complex_number_literal

This type of **generic_literal** represents mathematical complex number constants in expressions.

EXPRESS specification:

```
*)
ENTITY complex_number_literal
  SUBTYPE OF (generic_literal);
  real_part : REAL;
  imag_part : REAL;
END_ENTITY;
(*
```

Attribute definitions:

real_part: The value of the real part of the complex number which is to be represented by this literal in expressions.

imag_part: The value of the imaginary part of the complex number which is to be represented by this literal in expressions.