ФЕДЕРАЛЬНОЕ АГЕНТСТВО

ПО ТЕХНИЧЕСКОМУ РЕГУЛИРОВАНИЮ И МЕТРОЛОГИИ

НАЦИОНАЛЬНЫЙ СТАНДАРТ РОССИЙСКОЙ ФЕДЕРАЦИИ

ГОСТ Р ИСО 10303-11— 2009

Системы автоматизации производства и их интеграция

ПРЕДСТАВЛЕНИЕ ДАННЫХ ОБ ИЗДЕЛИИ И ОБМЕН ЭТИМИ ДАННЫМИ

Часть 11

Методы описания. Справочное руководство по языку EXPRESS

ISO 10303-11:2004
Industrial automation systems and integration — Product data representation and exchange — Part 11: Description methods. The EXPRESS language reference manual
(IDT)

Издание официальное

БЗ 10—2009/745

Москва
Стандартинформ
2010

gost.e-ecolog.ru

Предисловие

Цели и принципы стандартизации в Российской Федерации установлены Федеральным законом от 27 декабря 2002 г. № 184-ФЗ «О техническом регулировании», а правила применения национальных стандартов Российской Федерации — ГОСТ Р 1.0 — 2004 «Стандартизация в Российской Федерации. Основные положения»

Сведения о стандарте

1 ПОДГОТОВЛЕН Государственным научным учреждением «Центральный научно-исследовательский и опытно-конструкторский институт робототехники и технической кибернетики» на основе собственного аутентичного перевода на русский язык стандарта, указанного в пункте 4

2 ВНЕСЕН Техническим комитетом по стандартизации ТК 459 «Информационная поддержка жизненного цикла изделий»

3 УТВЕРЖДЕН И ВВЕДЕН В ДЕЙСТВИЕ Приказом Федерального агентства по техническому регулированию и метрологии от 14 сентября 2009 г. № 366-ст

4 Настоящий стандарт идентичен международному стандарту ИСО 10303-11:2004 «Системы автоматизации производства и их интеграция. Представление данных об изделии и обмен этими данными. Часть 11. Методы описания. Справочное руководство по языку EXPRESS» (ISO 10303-11:2004 «Industrial automation systems and integration — Product data representation and exchange — Part 11: Description methods. The EXPRESS language reference manual»).

При применении настоящего стандарта рекомендуется использовать вместо ссылочных международных стандартов соответствующие им национальные стандарты Российской Федерации, сведения о которых приведены в дополнительном приложении ДА

5 ВЗАМЕН ГОСТ Р ИСО 10303-11—2000

Информация об изменениях к настоящему стандарту публикуется в ежегодно издаваемом информационном указателе «Национальные стандарты», а текст изменений и поправок — в ежемесячно издаваемых информационных указателях «Национальные стандарты». В случае пересмотра (замены) или отмены настоящего стандарта соответствующее уведомление будет опубликовано в ежемесячно издаваемом информационном указателе «Национальные стандарты». Соответствующая информация, уведомление и тексты размещаются также в информационной системе общего пользования — на официальном сайте Федерального агентства по техническому регулированию и метрологии в сети Интернет

© Стандартинформ, 2010

Настоящий стандарт не может быть полностью или частично воспроизведен, тиражирован и распространен в качестве официального издания без разрешения Федерального агентства по техническому регулированию и метрологии

Содержание

1 Область применения . . . 1
2 Нормативные ссылки . . . 1
3 Термины и определения . . . 2
3.1 Термины, определенные в ИСО 10303-1 . . . 2
3.2 Термины, определенные в ИСО/МЭК 10646-1 . . . 2
3.3 Другие определения . . . 2
4 Требования соответствия . . . 3
4.1 Формальные спецификации, написанные на языке EXPRESS . . . 3
4.1.1 Лексический язык . . . 3
4.1.2 Графическая форма . . . 4
4.2 Реализации языка EXPRESS . . . 4
4.2.1 Синтаксический анализатор языка EXPRESS . . . 4
4.2.2 Графический редактор . . . 4
5 Фундаментальные принципы . . . 4
6 Синтаксис спецификации языка . . . 5
6.1 Синтаксис спецификации . . . 5
6.2 Обозначение специальных символов . . . 6
7 Основные элементы языка . . . 6
7.1 Набор символов . . . 7
7.1.1 Цифры . . . 7
7.1.2 Буквы . . . 7
7.1.3 Специальные символы . . . 7
7.1.4 Подчеркивание . . . 7
7.1.5 Пустое пространство . . . 8
7.1.6 Комментарии . . . 8
7.2 Зарезервированные слова . . . 10
7.2.1 Ключевые слова . . . 10
7.2.2 Зарезервированные слова, обозначающие операторы . . . 10
7.2.3 Встроенные константы . . . 11
7.2.4 Встроенные функции . . . 11
7.2.5 Встроенные процедуры . . . 11
7.3 Знаки . . . 11
7.4 Идентификаторы . . . 12
7.5 Литералы . . . 12
7.5.1 Двоичный литерал . . . 12
7.5.2 Целочисленный литерал . . . 12
7.5.3 Действительный литерал . . . 12
7.5.4 Строковый литерал . . . 13
7.5.5 Логический литерал . . . 14
8 Типы данных . . . 14
8.1 Простые типы данных . . . 14
8.1.1 Числовой тип данных . . . 14
8.1.2 Действительный тип данных . . . 15
8.1.3 Целочисленный тип данных . . . 15
8.1.4 Логический тип данных . . . 15
8.1.5 Булев тип данных . . . 15
8.1.6 Строковый тип данных . . . 16
8.1.7 Двоичный тип данных . . . 16
8.2 Агрегированные типы данных . . . 17
8.2.1 Тип данных ARRAY . . . 17
8.2.2 Тип данных LIST . . . 18
8.2.3 Тип данных BAG . . . 19
8.2.4 Тип данных SET . . . 19
8.2.5 Уникальность значений в агрегированных структурах . . . 20

8.3 Именованные типы данных . . . 21
8.3.1 Объектный тип данных . . . 21
8.3.2 Определенный тип данных . . . 21
8.4 Конструкционные типы данных . . . 21
8.4.1 Перечисляемый тип данных . . . 22
8.4.2 Выбираемый тип данных . . . 24
8.5 Обобщенные типы данных . . . 25
8.6 Классификация применения типов данных . . . 26
8.6.1 Конкретизирующие типы данных . . . 26
8.6.2 Параметрические типы данных . . . 27
8.6.3 Базисные типы данных . . . 27
9 Объявления . . . 27
9.1 Объявление типа . . . 28
9.2 Объявление объекта . . . 29
9.2.1 Атрибуты . . . 29
9.2.2 Локальные правила . . . 33
9.2.3 Подтипы и супертипы . . . 35
9.2.4 Абстрактный объектный тип данных . . . 40
9.2.5 Ограничения подтипов/супертипов . . . 41
9.2.6 Неявные объявления . . . 44
9.2.7 Конкретизация . . . 45
9.3 Схема . . . 46
9.4 Константа . . . 47
9.5 Алгоритмы . . . 47
9.5.1 Функция . . . 47
9.5.2 Процедура . . . 48
9.5.3 Параметры . . . 48
9.5.4 Локальные переменные . . . 53
9.6 Правило . . . 53
9.7 Ограничения подтипов . . . 55
9.7.1 Ограничение абстрактного супертипа . . . 56
9.7.2 Подтипы полного покрытия . . . 56
9.7.3 Перекрывающиеся подтипы и их спецификация . . . 57
10 Область видимости и видимость . . . 58
10.1 Правила области видимости . . . 58
10.2 Правила видимости . . . 59
10.3 Правила для явных элементов . . . 60
10.3.1 Оператор альтернативных имен ALIAS . . . 61
10.3.2 Атрибут . . . 61
10.3.3 Константа . . . 61
10.3.4 Элемент перечисления . . . 61
10.3.5 Объект . . . 61
10.3.6 Функция . . . 62
10.3.7 Параметр . . . 62
10.3.8 Процедура . . . 62
10.3.9 Выражение QUERY . . . 62
10.3.10 Оператор цикла . . . 63
10.3.11 Правило . . . 63
10.3.12 Метка правила . . . 63
10.3.13 Схема . . . 63
10.3.14 Ограничение подтипа . . . 64
10.3.15 Тип . . . 64
10.3.16 Метка типа . . . 64
10.3.17 Переменная . . . 64

11 Спецификация интерфейсов . . . 65
11.1 Спецификация интерфейса USE . . . 65
11.2 Спецификация интерфейса REFERENCE . . . 65
11.3 Взаимодействие интерфейсов USE и REFERENCE . . . 66
11.4 Импорт объектов посредством неявных интерфейсов . . . 66
11.4.1 Импорт констант . . . 67
11.4.2 Импорт определенных типов данных . . . 67
11.4.3 Импорт объектных типов данных . . . 67
11.4.4 Импорт функций . . . 68
11.4.5 Импорт процедур . . . 68
11.4.6 Импорт правил . . . 68
11.4.7 Импорт ограничений подтипов . . . 68
12 Выражения . . . 68
12.1 Арифметические операторы . . . 69
12.2 Операторы отношений . . . 70
12.2.1 Операторы сравнения значений . . . 71
12.2.2 Операторы сравнения экземпляров . . . 74
12.2.3 Оператор принадлежности . . . 75
12.2.4 Интервальные выражения . . . 76
12.2.5 Оператор сопоставления строк . . . 76
12.3 Двоичные операторы . . . 77
12.3.1 Индексирование двоичных чисел . . . 77
12.3.2 Оператор двоичной конкатенации . . . 78
12.4 Логические операторы . . . 78
12.4.1 Оператор NOT . . . 78
12.4.2 Оператор AND . . . 78
12.4.3 Оператор OR . . . 79
12.4.4 Оператор XOR . . . 79
12.5 Строковые операторы . . . 80
12.5.1 Индексирование строк . . . 80
12.5.2 Оператор конкатенации строк . . . 80
12.6 Операторы агрегированных структур . . . 80
12.6.1 Индексирование агрегированных структур . . . 81
12.6.2 Оператор пересечения . . . 81
12.6.3 Оператор объединения . . . 82
12.6.4 Оператор различия . . . 83
12.6.5 Оператор подмножества . . . 84
12.6.6 Оператор супермножества . . . 84
12.6.7 Оператор запроса . . . 84
12.7 Ссылки . . . 85
12.7.1 Простые ссылки . . . 85
12.7.2 Префиксные ссылки . . . 86
12.7.3 Ссылки на атрибуты . . . 86
12.7.4 Групповые ссылки . . . 87
12.8 Вызов функции . . . 88
12.9 Инициализатор агрегированных структур . . . 89
12.10 Оператор построения экземпляра сложного объекта . . . 90
12.11 Совместимость типов . . . 90
12.12 Выбираемые типы данных в выражениях . . . 91
12.12.1 Выбираемые типы данных в унарных выражениях . . . 91
12.12.2 Выбираемые типы данных в бинарных выражениях . . . 91
12.12.3 Выбираемые типы данных в тернарных выражениях . . . 92
13 Исполняемые операторы . . . 92
13.1 Пустой оператор . . . 92
13.2 Оператор ALIAS . . . 92

13.3 Присваивание . . . 93
13.3.1 Оператор присваивания . . . 93
13.3.2 Совместимость по присваиванию . . . 93
13.4 Оператор CASE . . . 95
13.5 Составной оператор . . . 96
13.6 Оператор ESCAPE . . . 96
13.7 Оператор IF...THEN...ELSE . . . 97
13.8 Оператор вызова процедуры . . . 97
13.9 Оператор REPEAT . . . 97
13.9.1 Инкрементное управление . . . 98
13.9.2 Управляющее условие WHILE . . . 99
13.9.3 Управляющее условие UNTIL . . . 99
13.10 Оператор RETURN . . . 99
13.11 Оператор SKIP . . . 99
14 Встроенные константы . . . 100
14.1 Константа e . . . 100
14.2 Неопределенность . . . 100
14.3 Константа FALSE . . . 100
14.4 Константа PI . . . 100
14.5 SELF . . . 100
14.6 Константа TRUE . . . 100
14.7 Константа UNKNOWN . . . 100
15 Встроенные функции . . . 101
15.1 Арифметическая функция ABS . . . 101
15.2 Арифметическая функция ACOS . . . 101
15.3 Арифметическая функция ASIN . . . 101
15.4 Арифметическая функция ATAN . . . 101
15.5 Двоичная функция BLENGTH . . . 101
15.6 Арифметическая функция COS . . . 102
15.7 Универсальная функция EXISTS . . . 102
15.8 Арифметическая функция EXP . . . 102
15.9 Универсальная функция FORMAT . . . 102
15.9.1 Символьное представление . . . 102
15.9.2 Представление шаблоном . . . 103
15.9.3 Стандартное представление . . . 104
15.10 Арифметическая функция HIBOUND . . . 104
15.11 Арифметическая функция HIINDEX . . . 104
15.12 Строковая функция LENGTH . . . 105
15.13 Арифметическая функция LOBOUND . . . 105
15.14 Арифметическая функция LOG . . . 105
15.15 Арифметическая функция LOG2 . . . 106
15.16 Арифметическая функция LOG10 . . . 106
15.17 Арифметическая функция LOINDEX . . . 106
15.18 Функция пустого значения NVL . . . 106
15.19 Арифметическая функция ODD . . . 106
15.20 Универсальная функция ROLESOF . . . 107
15.21 Арифметическая функция SIN . . . 107
15.22 Агрегированная функция SIZEOF . . . 108
15.23 Арифметическая функция SQRT . . . 108
15.24 Арифметическая функция TAN . . . 108
15.25 Универсальная функция TYPEOF . . . 108
15.26 Универсальная функция USEDIN . . . 110
15.27 Арифметическая функция VALUE . . . 111
15.28 Функция принадлежности VALUE_IN . . . 111
15.29 Функция уникальности VALUE_UNIQUE . . . 111

16 Встроенные процедуры . . . 112
16.1 Процедура INSERT . . . 112
16.2 Процедура REMOVE . . . 112
Приложение А (обязательное) Синтаксис языка EXPRESS . . . 113
А.1 Лексические элементы . . . 113
А.1.1 Ключевые слова . . . 113
А.1.2 Классы символов . . . 115
А.1.3 Лексические элементы . . . 116
А.1.4 Комментарии . . . 116
А.1.5 Интерпретированные идентификаторы . . . 116
А.2 Грамматические правила . . . 116
А.3 Список перекрестных ссылок . . . 121
Приложение В (обязательное) Определение допустимых реализаций объектов . . . 127
В.1 Формализованный подход . . . 127
В.2 Операторы ограничения супертипов и подтипов . . . 128
В.2.1 ONEOF . . . 128
В.2.2 AND . . . 128
В.2.3 ANDOR . . . 128
В.2.4 Приоритет операторов . . . 128
В.3 Интерпретация возможных типов данных сложных объектов . . . 128
Приложение С (обязательное) Ограничения на экземпляры, налагаемые спецификацией интерфейса 138
Приложение D (обязательное) Графическое подмножество языка EXPRESS — EXPRESS-G . . . 141
D.1 Введение и обзор . . . 141
D.2 Обозначения определений . . . 141
D.2.1 Обозначения простых типов данных . . . 141
D.2.2 Обозначения конструкционных типов данных . . . 142
D.2.3 Обозначение определенных типов данных . . . 143
D.2.4 Обозначение объектных типов данных . . . 144
D.2.5 Обозначение ограничений подтипов . . . 144
D.2.6 Обозначение функций и процедур . . . 144
D.2.7 Обозначение правил . . . 144
D.2.8 Обозначение схем . . . 144
D.3 Обозначение взаимосвязей . . . 144
D.4 Обозначение компоновки диаграмм . . . 145
D.4.1 Ссылки между страницами . . . 146
D.4.2 Ссылки между схемами . . . 146
D.5 Диаграммы уровня объектов . . . 146
D.5.1 Имена ролей . . . 146
D.5.2 Мощности множеств . . . 146
D.5.3 Ограничения . . . 147
D.5.4 Конструкционные и определенные типы данных . . . 147
D.5.5 Объектные типы данных . . . 148
D.5.6 Ссылки между схемами . . . 151
D.6 Диаграммы уровня схем . . . 151
D.7 Полные EXPRESS-G диаграммы . . . 152
D.7.1 Полная диаграмма уровня объектов . . . 152
D.7.2 Полная диаграмма уровня схем . . . 153
Приложение Е (обязательное) Заявка о соответствии реализации протоколу (ЗСРП) . . . 154
Е.1 Синтаксический анализатор языка EXPRESS . . . 154
Е.2 Средство редактирования EXPRESS-G . . . 154
Приложение F (обязательное) Регистрация информационного объекта . . . 156
F.1 Обозначение документа . . . 156
F.2 Обозначение синтаксиса . . . 156
Приложение G (обязательное) Генерация одной схемы из нескольких схем . . . 157
G.1 Введение . . . 157
G.2 Основные понятия . . . 157

G.3 Изменение имен . . . 158
G.3.1 Конфликты имен . . . 158
G.3.2 Идентификаторы, представленные строками . . . 158
G.4 Этап 1 — преобразование нескольких схем в промежуточную схему . . . 158
G.4.1 Введение . . . 158
G.4.2 Первичное содержимое . . . 159
G.4.3 Вторичное содержимое . . . 160
G.4.4 Сокращение . . . 165
G.4.5 Имена и версии схем . . . 169
G.5 Этап 2 — преобразование промежуточной схемы в схему по ИСО 10303-11:1994 . . . 170
G.5.1 Введение . . . 170
G.5.2 Инициализация . . . 170
G.5.3 Преобразование наращиваемых конструкционных типов данных . . . 170
G.5.4 Преобразование ограничений подтипов . . . 173
G.5.5 Преобразование абстрактных объектных и обобщенных типов данных . . . 175
G.5.6 Преобразование атрибутов, переименованных при повторном объявлении . . . 176
Приложение H (справочное) Взаимосвязи . . . 178
H.1 Взаимосвязи через атрибуты . . . 178
H.1.1 Простая взаимосвязь . . . 179
H.1.2 Групповая взаимосвязь . . . 180
H.1.3 Дистрибутивная взаимосвязь . . . 181
H.1.4 Инверсный атрибут . . . 182
H.2 Взаимосвязи подтип/супертип . . . 182
Приложение J (справочное) Модели на языке EXPRESS для примеров, иллюстрирующих EXPRESS-G . . . 183
J.1 Пример модели единой схемы . . . 183
J.2 Модель взаимосвязей . . . 184
J.3 Простое дерево подтипов/супертипов . . . 184
J.4 Повторное объявление атрибутов . . . 185
J.5 Модели, состоящие из нескольких схем . . . 185
Приложение K (справочное) Возможность языка EXPRESS, не рекомендуемые к использованию . . . 187
Приложение L (справочное) Пример использования новых конструкций языка EXPRESS . . . 188
L.1 Пример управления разработкой изделий . . . 188
Приложение ДА (справочное) Сведения о соответствии ссылочных международных стандартов ссылочным национальным стандартам Российской Федерации . . . 190
Библиография . . . 191

Введение

Стандарты комплекса ИСО 10303 распространяются на компьютерное представление информации об изделиях и обмен данными об изделиях. Их целью является обеспечение нейтрального механизма, способного описывать изделия на всем протяжении их жизненного цикла. Этот механизм применим не только для нейтрального обмена файлами, но является также основой для реализации и совместного доступа к базам данных об изделиях и организации архивирования.

Стандарты комплекса ИСО 10303 представляют собой набор отдельно издаваемых стандартов (частей). Структура комплекса ИСО 10303 определена в ИСО 10303-1. Стандарты данного комплекса относятся к одной из следующих тематических групп: методы описания, методы реализации, методология и основы аттестационного тестирования, интегрированные обобщенные ресурсы, интегрированные прикладные ресурсы, прикладные протоколы, комплекты абстрактных тестов, прикладные интерпретированные конструкции и прикладные модули. Настоящий стандарт входит в тематическую группу методов описания.

Полный перечень стандартов комплекса ИСО 10303 доступен в Интернете на сайте

gost.e-ecolog.ru

.

Настоящий стандарт определяет элементы языка EXPRESS. Каждый элемент языка представлен в собственном контексте с примерами. Сначала представлены простые элементы, а далее с нарастающей сложностью определяются более сложные конструкции. Настоящая вторая редакция ИСО 10303-11 включает в себя небольшой пересмотр положений первой редакции (ИСО 10303-11:1994), за которой временно сохранен статус действующего стандарта для поддержки основанных на ней реализаций языка EXPRESS и нормативных ссылок в других стандартах комплекса ИСО 10303. Во вторую редакцию включена также Техническая поправка ИСО 10303-11:1994/Кор.1:1999.

Изменения, которые привели к необходимости выпуска настоящей второй редакции, были вызваны требованиями мультисхемных спецификаций. Новые положения определяют архитектуру расширяемых моделей данных. В настоящий стандарт были добавлены следующие ключевые слова:

- BASED_ON;
- END_SUBTYPE_CONSTRAINT;
- EXTENSIBLE;
- GENERIC_ENTITY;
- RENAMED;
- SUBTYPE_CONSTRAINED;
- TOTAL_OVER;
- WITH.

Схемы, содержащие данные слова в качестве идентификаторов языка EXPRESS, становятся недействительными для области применения настоящего стандарта. Кроме того, изменения, включенные в настоящую редакцию стандарта, являются совместимыми снизу вверх по отношению к предыдущей редакции.

Обзор языка

EXPRESS — это название формального языка спецификации информационных требований. Язык EXPRESS применяется для определения информационных требований других стандартов комплекса ИСО 10303. Язык EXPRESS создавался для решения следующих задач:

- объем и сложность стандартов комплекса ИСО 10303 требуют наличия языка, обеспечивающего восприятие содержащейся в них информации как компьютерами, так и людьми. Представление информационных объектов из стандартов комплекса ИСО 10303 в нестрого формализованном виде исключило бы возможность применения вычислительной техники для проверки несоответствий в представлениях информационных объектов или создания произвольного числа их вторичных представлений, включая представления реализаций информационных объектов;

- язык EXPRESS разработан для обеспечения возможности структурирования разнообразных данных, относящихся к стандартам комплекса ИСО 10303. В данном языке EXPRESS-схема является основой для структурирования и взаимосвязи элементов представления данных об изделии;

- посредством языка определяются логические объекты, представляющие объекты реального мира. Определение объекта дается через его свойства, задаваемые областью их значений и накладываемыми на нее ограничениями;

- язык, насколько это возможно, не должен быть привязан к конкретным реализациям. Тем не менее, имеется возможность создания представлений реализации (например, обмен статическими файлами) автоматическим и прямым способом.

В языке EXPRESS объекты определяются через их атрибуты, характеризующие признаки или характеристики объектов, имеющие большое значение для их понимания и использования. Атрибуты могут быть представлены простыми данными (например, целым числом) или другими объектами. Геометрическая точка может быть определена тремя действительными числами. Имена, присвоенные атрибутам, включаются в определение объекта. Так, для геометрической точки три действительных числа могут быть названы X, Y и Z. В языке установлена взаимосвязь между определяемым объектом и определяющими его атрибутами, а также между атрибутом и его представлением.

Примечания

1 При разработке языка EXPRESS были учтены особенности ряда языков, в частности, Ada, Algol, C, C++, Euler, Modula-2, Pascal, PL/I и SQL. Некоторые возможности были добавлены в язык EXPRESS, чтобы сделать его более пригодным для описания информационных моделей.

2 В настоящем стандарте примеры текстов на языке EXPRESS не соответствуют правилам какого-либо конкретного стиля. Действительно, в примерах иногда используется плохой стиль для того, чтобы сэкономить место или показать гибкость языка. Приведенные примеры не предназначены для того, чтобы отразить содержание информационных моделей, установленных в других стандартах комплекса ИСО 10303. Примеры приведены для того, чтобы продемонстрировать конкретные особенности языка EXPRESS. Не следует придавать значение какому-либо сходству между примерами, приведенными в настоящем стандарте, и нормативными информационными моделями, установленными в других стандартах комплекса ИСО 10303.

НАЦИОНАЛЬНЫЙ СТАНДАРТ РОССИЙСКОЙ ФЕДЕРАЦИИ

Системы автоматизации производства и их интеграция

ПРЕДСТАВЛЕНИЕ ДАННЫХ ОБ ИЗДЕЛИИ И ОБМЕН ЭТИМИ ДАННЫМИ

Часть 11

Методы описания.
Справочное руководство по языку EXPRESS

Industrial automation systems and integration. Product data representation and exchange.
Part 11. Description methods. The EXPRESS language reference manual

Дата введения — 2010 — 07— 01

1 Область применения

В настоящем стандарте определен язык, посредством которого могут быть описаны данные об изделии. Данный язык называется EXPRESS.

В настоящем стандарте также определено графическое представление для подмножества конструкций языка EXPRESS. Данное графическое представление называется EXPRESS-G.

EXPRESS является языком определения данных, как это установлено в ИСО 10303-1. Данный язык состоит из элементов, которые позволяют однозначно определять данные и устанавливать ограничения на эти данные.

Область применения настоящего стандарта распространяется на:
- типы данных;
- ограничения на экземпляры типов данных.

Область применения настоящего стандарта не распространяется на:
- определение форматов баз данных;
- определение форматов файлов;
- определение форматов передачи;
- управление процессами;
- обработку информации;
- обработку исключительных ситуаций.

Язык EXPRESS не является языком программирования.

2 Нормативные ссылки

В настоящем стандарте использованы ссылки на следующие международные стандарты:

ИСО 10303-1:1994 Системы автоматизации производства и их интеграция. Представление данных об изделии и обмен этими данными. Часть 1. Общие представления и основополагающие принципы (ISO 10303-1:1994, Industrial automation systems and integration — Product data representation and exchange — Part 1: Overview and fundamental principles)

ИСО/МЭК 8824-1:2002 Информационные технологии. Взаимосвязь открытых систем. Абстрактная синтаксическая нотация версии один (АСН.1). Часть 1. Спецификация основной нотации (ISO/IEC 8824-1:2002, Information technology — Abstract Syntax Notation One (ASN.1): Specification of basic notation)

ИСО/МЭК 10646:2003 Информационные технологии. Универсальный многооктетный набор закодированных символов (UCS) [ISO/IEC 10646:2003, Information technology — Universal Multiple-Octet Coded Character Set (UCS)]

Издание официальное

3 Термины и определения

3.1 Термины, определенные в ИСО 10303-1

В настоящем стандарте применены следующие термины:

- **требование соответствия** (conformance requirement);
- **контекст** (context);
- **данные** (data);
- **язык определения данных** (data specification language);
- **информация** (information);
- **информационная модель** (information model);
- **форма ЗСРП** (PICS proforma).

3.2 Термины, определенные в ИСО/МЭК 10646-1

В настоящем стандарте применен следующий термин:

графический символ (graphic character).

Примечание — Данное определение охватывает только те символы из ИСО/МЭК 10646, которые имеют установленное визуальное представление; тем самым исключаются любые пустые или заштрихованные позиции в таблице символов.

3.3 Другие определения

В настоящем стандарте также применены следующие термины с соответствующими определениями:

3.3.1 **сложный объектный тип данных** (complex entity data type): Представление объекта. Сложный объектный тип данных устанавливает область значений, определяемую общими атрибутами и ограничениями допустимой комбинации объектных типов данных в конкретном графе подтипов/супертипов.

3.3.2 **экземпляр сложного объекта (сложного объектного типа данных)** [complex entity (data type) instance]: Именованное значение сложного объектного типа данных. Имя экземпляра сложного объекта используется для ссылок на данный экземпляр.

3.3.3 **значение сложного объекта (сложного объектного типа данных)** [complex entity (data type) value]: Элемент данных, представляющий элемент информации в рамках класса, определяемого сложным объектным типом данных. Данный элемент принадлежит области определения, установленной данным сложным объектным типом данных.

3.3.4 **константа** (constant): Именованный элемент данных из заданной области определения, значение которого не может быть изменено.

3.3.5 **тип данных** (data type): Область значений.

3.3.6 **объект** (entity): Класс информации, определенный общими свойствами.

3.3.7 **объектный тип данных** (entity data type): Представление объекта. Объектный тип данных устанавливает область значений, определяемую общими атрибутами и ограничениями.

3.3.8 **экземпляр объекта (объектного типа данных)** [entity (data type) instance]: Именованное значение объектного типа данных. Имя экземпляра объекта используется для ссылок на данный экземпляр.

3.3.9 **значение (отдельного) объекта (объектного типа данных)** [(single) entity (data type) value]: Элемент данных, представляющий элемент информации в рамках класса, определенного объектным типом данных. Данный элемент принадлежит области определения, установленной данным объектным типом данных.

3.3.10 **экземпляр** (instance): Именованное значение.

3.3.11 **многолепестковый сложный объект (многолепестковый сложный объектный тип данных)** [multi-leaf complex entity (data type)]: Сложный объектный тип данных, состоящий из нескольких объектных типов данных, которые не имеют последующих подтипов в рамках данного сложного объектного типа данных.

3.3.12 **экземпляр многолепесткового сложного объекта (многолепесткового сложного объектного типа данных)** [multi-leaf complex entity (data type) instance]: Именованное значение многолепесткового сложного объектного типа данных. Имя экземпляра многолепесткового сложного объекта используется для ссылок на данный экземпляр.

3.3.13 **значение многолепесткового сложного объекта (многолепесткового сложного объектного типа данных)** [multi-leaf complex entity (data type) value]: Элемент данных, представляющий элемент информации в рамках класса, определяемого многолепестковым сложным объектным типом данных.

Данный элемент принадлежит области определения, установленной данным многолепестковым сложным объектным типом данных.

3.3.14 **частичный сложный объектный тип данных** (partial complex entity data type): Потенциальное представление объекта. Частичный сложный объектный тип данных является группировкой объектных типов данных в графе подтипов/супертипов, которая может частично или полностью формировать сложный объектный тип данных.

3.3.15 **значение частичного сложного объекта** (partial complex entity value): Значение частичного сложного объектного типа данных. Данное значение само по себе не имеет смысла и должно объединяться с другими значениями частичного сложного объекта и с именем для формирования экземпляра сложного объекта.

3.3.16 **совокупность** (population): Множество экземпляров объектного типа данных.

3.3.17 **основная схема** (primary schema): Схема в группе взаимосвязанных схем, образующая ориентированный граф, возможно, циклический. Основная схема является предметом интереса. В графе могут существовать одна или несколько основных схем, тогда как остальные схемы графа служат только для поддержки основных схем. Основная схема играет особую роль в преобразовании из короткой формы схемы в длинную форму (см. приложение G).

3.3.18 **корневая схема** (root schema): Схема в группе взаимосвязанных схем, образующая ориентированный граф, возможно, циклический. Корневая схема не является целью в какой-либо спецификации интерфейса, но все другие схемы должны быть достижимы из корневой схемы. Корневая схема может рассматриваться в качестве представителя графа. Корневая схема играет особую роль в преобразовании из короткой формы схемы в длинную форму (см. приложение G).

3.3.19 **экземпляр простого объекта (простого объектного типа данных)** [simple entity (data type) instance]: Именованный элемент данных, представляющий блок информации в рамках класса, определенного объектом. Данный элемент принадлежит области определения, установленной отдельным объектным типом данных.

3.3.20 **граф подтипов/супертипов** (subtype/supertype graph): Объявленная совокупность объектных типов данных. Объектные типы данных, объявленные в графе подтипов/супертипов, связаны через формулировку подтипов. Граф подтипов/супертипов определяет один или несколько сложных объектных типов данных.

3.3.21 **лексема** (token): Не подлежащий декомпозиции лексический элемент языка.

3.3.22 **значение** (value): Элемент данных.

4 Требования соответствия

4.1 Формальные спецификации, написанные на языке EXPRESS

4.1.1 Лексический язык

Формальная спецификация, написанная на языке EXPRESS, должна быть согласована с заданным уровнем, как определено ниже. Формальная спецификация считается согласованной с заданным уровнем, если все проверки, установленные для данного уровня и всех более низких уровней, верифицированы для данной спецификации.

Уровни проверки

Уровень 1 — проверка ссылок. Данный уровень состоит из проверки формальной спецификации для подтверждения ее синтаксической и ссылочной корректности. Формальная спецификация синтаксически корректна, если она соответствует синтаксису, сформированному посредством расширения основного синтаксического правила **(syntax),** установленного в приложении А. Формальная спецификация корректна в отношении ссылок, если все ссылки на элементы языка EXPRESS соответствуют области применения и правилам видимости, установленным в разделах 10 и 11.

Уровень 2 — проверка типов. Данный уровень включает в себя проверку формальной спецификации для подтверждения ее соответствия следующим требованиям:

- выражения должны удовлетворять правилам, установленным в разделе 12;
- присваивания должны удовлетворять правилам, установленным в 13.3;
- объявления инверсных атрибутов должны удовлетворять правилам, установленным в 9.2.1.3;
- повторные объявления атрибутов должны удовлетворять правилам, установленным в 9.2.3.4.

Уровень 3 — проверка значений. Данный уровень состоит из проверки формальной спецификации для подтверждения ее соответствия утверждениям типа «А должно быть больше В», установленным

в разделах 7 — 16. Данная проверка ограничена случаями, когда значения A и B могут быть выражены литералами и/или константами.

Уровень 4 — полная проверка. Данный уровень включает в себя проверку формальной спецификации для подтверждения ее соответствия формулировкам требований, установленных в настоящем стандарте.

Пример — В настоящем стандарте установлено, что функции должны содержать оператор возврата для каждой из возможных ветвей, по которым может пойти процесс при вызове данной функции, что и должно быть проверено.

4.1.2 Графическая форма

Формальная спецификация, представленная в формате EXPRESS-G, должна быть согласована с заданным уровнем, как определено ниже. Формальная спецификация считается согласованной с заданным уровнем, если все проверки, установленные для данного уровня и всех более низких уровней, верифицированы для данной спецификации.

Уровни проверки

Уровень 1 — проверка символов и области видимости. Данный уровень включает в себя проверку формальной спецификации для подтверждения ее соответствия спецификации уровня объекта или спецификации уровня схемы, которые определены в приложении D, разделы D.5 и D.6, соответственно. Данная проверка предусматривает проверку использования в формальной спецификации символов в соответствии с приложением D, разделы D.2, D.3 и D.4. Формальная спецификация также должна быть проверена на соответствие страничных ссылок и повторно объявленных атрибутов требованиям приложения D, подразделы D.4.1 и D.5.5, соответственно.

Уровень 2 — полная проверка. Данный уровень включает в себя проверку формальной спецификации на предмет установления в ней мест, не соответствующих требованиям уровня полного объекта или уровня полной схемы, установленным в приложении D, а также требованиям, установленным в разделах 7 — 16.

4.2 Реализации языка EXPRESS

4.2.1 Синтаксический анализатор языка EXPRESS

Реализация синтаксического анализатора языка EXPRESS должна обеспечивать синтаксический разбор любой формальной спецификации, написанной на языке EXPRESS, в соответствии с ограничениями, установленными в приложении E и связанными с данной реализацией. Синтаксический анализатор языка EXPRESS должен считаться соответствующим конкретному уровню проверки (см. 4.1.1), если он может выполнять все требуемые для данного (и любого нижележащего) уровня проверки формальной спецификации, написанной на языке EXPRESS.

Разработчик синтаксического анализатора языка EXPRESS должен точно определить все ограничения, которые реализация накладывает на число и длину идентификаторов, диапазон обрабатываемых чисел и максимальную точность представления действительных чисел. Данные ограничения должны быть документально оформлены в виде, установленном в приложении E, необходимом для проведения аттестационного тестирования.

4.2.2 Графический редактор

Реализация редактора для графической нотации EXPRESS-G должна обеспечивать создание и отображение формальных спецификаций, представленных в формате EXPRESS-G, в соответствии с ограничениями, установленными в приложении E и связанными с данной реализацией. Редактор EXPRESS-G должен считаться соответствующим конкретному уровню проверки, если он может создавать и отображать формальные спецификации в формате EXPRESS-G, соответствующие заданному (и любому нижележащему) уровню проверки.

Разработчик редактора EXPRESS-G должен точно определить любые ограничения, которые реализация накладывает на число и длину идентификаторов, число доступных символов на странице модели и максимальное число страниц. Данные ограничения должны быть документально оформлены в виде, установленном в приложении E, необходимом для проведения аттестационного тестирования.

5 Фундаментальные принципы

Для использования настоящего стандарта необходимо знание представленных ниже понятий.

Схема, написанная на языке EXPRESS, описывает совокупность условий, устанавливающих область ее определения. Экземпляры объектов могут быть оценены на их принадлежность к данной области определения. Если экземпляры объектов соответствуют всем условиям, то они объявляются принадлежащими данной области определения. Если экземпляры объектов не соответствуют каким-либо из условий, то они

нарушают данные условия и поэтому не принадлежат данной области определения. В случае если экземпляры объектов не содержат значения для необязательных атрибутов, а при некоторых условиях данные необязательные атрибуты используются, то может оказаться невозможным определить, соответствуют ли экземпляры объектов всем условиям. В таком случае считается, что экземпляры объектов принадлежат данной области определения.

Многим элементам языка EXPRESS присвоены имена. Имя позволяет другим элементам языка ссылаться на связанное с этим именем представление. Использование имени в определении других элементов языка создает ссылку на базовое представление. Хотя в соответствии с синтаксисом языка для обозначения имени используется идентификатор, базовое представление должно быть изучено для понимания его структуры.

Спецификация объектного типа данных в языке EXPRESS описывает область определения. Предполагается, что отдельные элементы области определения различаются некоторыми связанными с ними уникальными идентификаторами. Язык EXPRESS не определяет содержание или представление этих идентификаторов.

Объявление постоянного экземпляра объекта определяет идентифицируемый элемент области определения, представленный объектным типом данных. Такие экземпляры объектов не должны изменяться или уничтожаться операциями, выполняемыми в данной области определения.

Процедурные описания ограничений в языке EXPRESS могут объявлять или делать ссылки на дополнительные экземпляры объекта как на локальные переменные, которые принимаются как временные идентифицируемые элементы области определения. Данные процедурные описания могут изменять дополнительные экземпляры объекта, но не могут изменять постоянные элементы области определения. Такие временные элементы области определения доступны только в процессе выполнения процедуры, в которой они объявлены, и прекращают свое существование после завершения ее выполнения.

Язык EXPRESS не описывает среду реализации. В частности язык EXPRESS не определяет:

- как реализуются ссылки на имена;
- какие другие схемы становятся известными;
- как и когда проверяются ограничения;
- что должна делать реализация, если ограничение нарушено;
- имеют или не имеют право на существование в реализации экземпляры объектов, которые не соответствуют EXPRESS-схеме;
- когда и как в реализации создаются, изменяются и удаляются экземпляры объектов.

6 Синтаксис спецификации языка

В настоящем разделе определена нотация, используемая для представления синтаксиса языка EXPRESS.

Полный синтаксис языка EXPRESS приведен в приложении А. Части этих синтаксических правил воспроизведены в различных разделах настоящего стандарта для иллюстрации синтаксиса конкретных операторов. Эти части не всегда полны. Поэтому иногда необходимо руководствоваться приложением А в отношении недостающих в данном примере правил. Части синтаксических правил в тексте настоящего стандарта представлены в рамках. Каждое синтаксическое правило внутри рамки обозначено слева уникальным номером для использования его в перекрестных ссылках в других синтаксических правилах.

6.1 Синтаксис спецификации

Синтаксис языка EXPRESS определен как производная от синтаксической нотации Вирта (СНВ) [3].

Соглашения об обозначениях и самоопределенная СНВ приведены ниже.

```
syntax          = { production } .
production      = identifier '=' expression '.' .
expression      = term { '|' term } .
term            = factor { factor } .
factor          = identifier | literal | group | option | repetition .
identifier      = character { character } .
literal         = '"' character { character } '"' .
group           = '(' expression ')' .
option          = '[' expression ']' .
repetition      = '{' expression '}' .
```

Знак равенства '=' обозначает порождающее правило. Элемент слева от знака равенства определяется как комбинация элементов, расположенных справа от него. Любые пробелы между элементами правой части не имеют значения, если только они не входят в состав литерала. В конце порождающего правила ставится точка '.'.

Использование идентификатора в любом элементе обозначает нетерминальный символ, который присутствует в левой части другого порождающего правила. Идентификатор состоит из букв, цифр и символа подчеркивания. Ключевые слова языка представлены порождающими правилами, идентификаторы которых состоят только из прописных букв.

Литерал используется для обозначения терминального символа, который не может быть раскрыт в дальнейшем. Литерал представляется последовательностью не зависящих от регистра символов, заключенной в апострофы. Под символом в данном случае понимается любой символ, определенный в ИСО/МЭК 10646 в позициях 21 — 7E группы 00, плоскости 00, строки 00. Чтобы апостроф был включен в литерал, он должен быть записан дважды.

Семантика разных видов скобок определена следующим образом:

- фигурные скобки '**{ }**' обозначают ни одного или несколько повторений;
- квадратные скобки '**[ ]**' обозначают необязательные параметры;
- круглые скобки '**( )**' обозначают, что группа порождающих правил, заключенная в круглые скобки, должна использоваться как единое порождающее правило;
- вертикальная линия '**|**' обозначает, что в выражении должен использоваться только один из элементов, разделенных вертикальными линиями.

Примеры

1 Синтаксис строкового типа данных определяется следующим образом:

```
Синтаксис:
  311 string_type = STRING [ width_spec ] .
  341 width_spec = ' ( ' width ' ) ' [ FIXED ] .
  340 width = numeric_expression .
```

***Полное определение синтаксиса, представленное в приложении А, содержит определения лексем* STRING, numeric_expression *и* FIXED.**

2 В соответствии с синтаксисом, приведенном в примере 1, возможны следующие варианты:

string;

string (22);

string (19) fixed.

Правило для* numeric_expression *является достаточно сложным и позволяет представить много других вариантов.

6.2 Обозначение специальных символов

Следующая нотация используется для представления полных наборов символов и некоторых специальных символов, которые трудно визуально отобразить:

\a — представляет символы в позициях 21-7E строки 00, плоскости 00, группы 00 из ИСО/МЭК 10646;

\n — представляет символ новой строки (newline), зависящий от системы (см. 7.1.5.2);

\q — представляет символ одиночной кавычки (апострофа) (') и входит в \a;

\s — представляет символ пробела;

\x9, **\xA** и **\xD** — представляют символы, расположенные соответственно в позициях 9, 10 и 13 строки 00, плоскости 00, группы 00 из ИСО/МЭК 10646.

7 Основные элементы языка

В данном разделе определены основные элементы, из которых формируется EXPRESS-схема: набор символов, комментарии, знаки, зарезервированные слова, идентификаторы и литералы.

Из основных элементов языка формируется текстовая структура, обычно разделяемая на физические строки. Физическая строка представляет собой последовательность из любого числа (включая ни одного) символов, заканчивающуюся символом новой строки (см. 7.1.5.2).

Примечание — Схема более удобна для восприятия, если операторы разбиты на строки, а для выделения разных конструкций использованы пробелы.

Пример — Следующие форматы записи эквивалентны:

```
entity point; x, y, z : real; end_entity;
ENTITY point;
   x,
   y,
   Z : REAL;
END_ENTITY;
```

7.1 Набор символов

В схемах, представленных на языке EXPRESS, должны использоваться только символы из набора, включающего символы, расположенные в позициях 09, 0A, 0D, графические символы, лежащие в диапазоне от 20 до 7E из ИСО/МЭК 10646, а также специальный символ **\n**, обозначающий новую строку. Данный набор символов называется набором символов языка EXPRESS. На символы данного набора можно ссылаться по позиции, на которой расположен данный символ; номера этих позиций определены в шестнадцатеричной системе. Символы из данного набора, которые могут быть воспроизведены при печати (позиции 21–7E из ИСО/МЭК 10646), комбинируются для формирования лексем языка EXPRESS. Лексемами EXPRESS являются ключевые слова, идентификаторы, знаки или литералы. Более подробно классификация набора символов языка EXPRESS рассмотрена ниже.

Таким образом, данный набор символов определен как абстрактный набор символов; он не зависит от его представления в конкретной реализации.

Примечания

1 В ИСО/МЭК 6429 [5] установлена семантика символов, расположенных в позициях 09, 0A, 0D из ИСО/МЭК 10646. Для настоящего стандарта семантика, установленная в ИСО/МЭК 6429, не требуется, но она и не противоречит ему.

2 В данном разделе приводятся только ссылки на символы, используемые для определения EXPRESS-схемы, но не определяется область значений символов, допустимых для строкового типа данных.

7.1.1 Цифры

В языке EXPRESS используются арабские цифры 0 — 9 (позиции 30 — 39 из набора символов языка EXPRESS).

Синтаксис:

```
124 digit = '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9'.
```

7.1.2 Буквы

В языке EXPRESS используются буквы верхнего и нижнего регистров английского алфавита (позиции 41— 5A и 61— 7A из набора символов языка EXPRESS). Регистр букв имеет значение только в явных строковых литералах.

Примечание — В структурах языка EXPRESS могут использоваться буквы верхнего, нижнего или обоих регистров (см. пример в 7).

Синтаксис:

```
128 letter = 'a' | 'b' | 'c' | 'd' | 'e' | 'f' | 'g' | 'h' | 'i' | 'j' | 'k' |
             'l' | 'm' | 'n' | 'o' | 'p' | 'q' | 'r' | 's' | 't' | 'u' | 'v' |
             'w' | 'x' | 'y' | 'z'.
```

7.1.3. Специальные символы

Специальные символы (воспроизводимые при печати символы, не являющиеся буквами и цифрами) используются главным образом для пунктуации и в качестве операторов. Специальные символы соответствуют позициям 21 — 2F, 3A — 3F, 40, 5B — 5E, 60 и 7B — 7E набора символов языка EXPRESS.

Синтаксис:

```
137 special = not_paren_star_quote_special | '(' | ')' | '*' | ''''.
132 not_paren_star_quote_special = '!' | '"' | '#' | '$' | '%' | '&' | '+' |
                                   ',' | '-' | '.' | '/' | ':' | ';' | '<' |
                                   '=' | '>' | '?' | '@' | '[' | '\' | ']' |
                                   '^' | '_' | '`' | '{' | '|' | '}' | '~'.
```

7.1.4 Подчеркивание

Символ подчеркивания ('_', позиция 5F из набора символов языка EXPRESS) может использоваться в идентификаторах и ключевых словах, но не в качестве первого символа.

7.1.5 Пустое пространство

Пустое пространство в тексте образуется посредством символов, определенных в 7.1.5.1 — 7.1.5.3 и 7.1.6. Пустое пространство должно использоваться для разделения лексем в EXPRESS-схемах.

Примечание — Свободное и последовательное использование пустого пространства может улучшить структуру и удобочитаемость схемы.

7.1.5.1 Символ пробела

Один или несколько пробелов (позиция 20 из набора символов EXPRESS) могут располагаться между двумя лексемами. Обозначение **\s** может использоваться для представления символа пробела в синтаксических структурах языка.

7.1.5.2 Новая строка

Символом новой строки заканчивается каждая строка в формальной спецификации на языке EXPRESS. Обычно символ новой строки трактуется как пробел, но он является значимым символом, если им заканчивается комментарий или анормально заканчивается строковый литерал. В синтаксических структурах языка символ новой строки представляется обозначением **\n**.

Представление символа новой строки зависит от конкретной реализации.

7.1.5.3 Другие символы

Символы из позиций 09, 0A и 0D должны трактоваться как пустое пространство, кроме случаев, когда они встречаются в строковом литерале. Для представления этих символов в синтаксических структурах языка должно использоваться обозначение **\xn**, где **n** является одним из символов **9**, **A** или **D**.

7.1.6 Комментарии

Комментарий используется для документирования и должен интерпретироваться синтаксическим анализатором языка EXPRESS как пустое пространство. Существуют две формы комментария – встроенный комментарий и заключительный комментарий. Обе формы комментария могут быть ассоциированы с идентифицированной конструкцией посредством метки комментария.

7.1.6.1 Встроенный комментарий

Пара символов **(*** обозначает начало встроенного комментария, а пара символов ***)** обозначает его окончание. Встроенный комментарий может располагаться между любыми двумя лексемами.

Любой символ из набора символов языка EXPRESS может присутствовать между началом и концом встроенного комментария, включая символ новой строки, поэтому встроенные комментарии могут содержать несколько физических строк.

```
Синтаксис:
  145 embedded_remark = '(*' [ remark_tag ] { (not_paren_star { not_paren_star } ) |
                        lparen_then_not_lparen_star | ('*' {'*'} ) |
                        not_rparen_star_then_rparen | embedded_remark } '*)'.
  147 remark_tag = '"' remark_ref {'.' remark_ref } '"'.
  148 remark_ref = attribute_ref | constant_ref | entity_ref | enumeration_ref |
                   function_ref | parameter_ref | procedure_ref | rule_label_ref |
                   rule_ref | schema_ref | subtype_constraint_ref | type_label_ref |
                   type_ref | variable_ref .
  131 not_paren_star = letter | digit | not_paren_star_special .
  128 letter = 'a' | 'b' | 'c' | 'd' | 'e' | 'f' | 'g' | 'h' | 'i' | 'j' | 'k' |
               'l' | 'm' | 'n' | 'o' | 'p' | 'q' | 'r' | 's' | 't' | 'u' | 'v' |
               'w' | 'x' | 'y' | 'z'.
  124 digit = '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9'.
  133 not_paren_star_special = not_paren_star_quote_special | '''.
  132 not_paren_star_quote_special = '!' | '"' | '#' | '$' | '%' | '&' | '+' | ',' |
                                     '-' | '.' | '/' | ':' | ';' | '<' | '=' | '>' |
                                     '?' | '@' | '[' | '\' | ']' | '^' | '_' | '`' |
                                     '{' | '|' | '}' | '~'.
  129 lparen_then_not_lparen_star = '(' { '(' } not_lparen_star { not_lparen_star }.
  130 not_lparen_star = not_paren_star | ')'.
  138 not_rparen_star_then_rparen = not_rparen_star { not_rparen_star } ')' { ')' }.
  135 not_rparen_star = not_paren_star | '('.
```

Встроенные комментарии могут быть вложенными.

Примечание — При формировании вложенных комментариев необходимо обратить внимание на наличие согласованных пар символов, обозначающих начало и конец комментария.

Пример — Следующая строка демонстрирует пример встроенного комментария:
(* Символы '(*' начинают комментарий, а символы '*)' его заканчивают *)

7.1.6.2 Заключительный комментарий

Заключительный комментарий записывается в конце физической строки. Два последовательных дефиса «– –» начинают заключительный комментарий, а первый встреченный символ «новая строка» заканчивает его.

```
Синтаксис:
  149 tail_remark =  '--' [ remark_tag ] { \a | \s | \x9 | \xA | \xD } \n .
  147 remark_tag =  ' "' remark_ref { ' . ' remark_ref } ' "' .
  148 remark_ref =  attribute_ref | constant_ref | entity_ref | enumeration_ref |
                    function_ref | parameter_ref | procedure_ref | rule_label_ref |
                    rule_ref | schema_ref | subtype_constraint_ref | type_label_ref |
                    type_ref | variable_ref .
```

Пример — -- это комментарий, заканчивающийся символом «новая строка».

7.1.6.3 Метка комментария

Комментарий может быть соотнесен с именованным элементом, то есть с элементом, обозначенным идентификатором, посредством размещения метки комментария в качестве первой последовательности символов. Метка комментария должна следовать непосредственно за парой символов, идентифицирующих комментарий. Сама метка комментария состоит из ссылки на идентификатор, определенный последовательностью символов, заключенной в кавычки.

```
Синтаксис:
  147 remark_tag =  ' "' remark_ref { '.' remark_ref } ' "' .
  148 remark_ref =  attribute_ref | constant_ref | entity_ref | enumeration_ref |
                    function_ref | parameter_ref | procedure_ref | rule_label_ref |
                    rule_ref | schema_ref | subtype_constraint_ref | type_label_ref |
                    type_ref | variable_ref .
```

Правила и ограничения:

a) Элемент **remark_ref** должен соответствовать правилам видимости, определенным в 10.2.

b) В составной ссылке комментария правила видимости, определенные в 10.2, должны использоваться следующим образом: ссылка слева от символа '.' должна идентифицировать область видимости, в которой определена ссылка, расположенная справа от символа '.'.

Примечание — Составная ссылка комментария представляет собой ссылку комментария, в которой используется нотация с символом '.' (см. синтаксическое правило 147).

c) Если ссылка комментария не найдена в соответствии с указанными выше правилами видимости, то комментарий не должен ассоциироваться с каким-либо элементом.

d) Помеченный комментарий, содержащий другие помеченные комментарии (посредством вложенности), должен ассоциироваться целиком (включая вложенные комментарии) с указанным элементом.

e) Если вложенный комментарий и комментарий, в который он вложен, оба ссылаются на один и тот же идентифицированный элемент, то вложенный комментарий должен быть связан с этим элементом дважды: один раз в составе комментария, в который он вложен, и второй раз непосредственно.

Примеры

1 Помеченный комментарий в данном примере ссылается на атрибут **attr** ***в области видимости объекта*** **ent***:*

```
ENTITY ent;
   attr: INTEGER;
END_ENTITY;
(* "ent.attr" Атрибут attr . . . *)
```

2 За ссылкой на схему my_second_schema *в помеченном комментарии может следовать любой идентификатор, объявленный непосредственно в области видимости данной схемы, например, имя функции* a_complicated_function, *как в данном примере:*

```
SCHEMA my_second_schema;
...
FUNCTION a_complicated_function;
...
END_FUNCTION;
(* "my_second_schema.a_complicated_function" Данная сложная функция ... *)
...
END_SCHEMA;
```

7.2 Зарезервированные слова

Зарезервированными словами языка EXPRESS являются ключевые слова и имена встроенных констант, функций и процедур. Зарезервированные слова не должны использоваться в качестве идентификаторов. Зарезервированные слова языка EXPRESS описаны ниже.

7.2.1 Ключевые слова

Ключевые слова языка EXPRESS представлены в таблице 1.

Примечание — Ключевые слова представляются литералом, состоящим из заглавных букв. Это сделано для облегчения чтения синтаксических конструкций.

Таблица 1 — Ключевые слова языка EXPRESS

ABSTRACT	AGGREGATE	ALIAS	ARRAY
AS	BAG	BASED_ON	BEGIN
BINARY	BOOLEAN	BY	CASE
CONSTANT	DERIVE	ELSE	END
END_ALIAS	END_CASE	END_CONSTANT	END_ENTITY
END_FUNCTION	END_IF	END_LOCAL	END_PROCEDURE
END_REPEAT	END_RULE	END_SCHEMA	END_SUBTYPE_CONSTRAINT
END_TYPE	ENTITY	ENUMERATION	ESCAPE
EXTENSIBLE	FIXED	FOR	FROM
FUNCTION	GENERIC	GENERIC_ENTITY	IF
INTEGER	INVERSE	LIST	LOCAL
LOGICAL	NUMBER	OF	ONEOF
OPTIONAL	OTHERWISE	PROCEDURE	QUERY
REAL	RENAMED	REFERENCE	REPEAT
RETURN	RULE	SCHEMA	SELECT
SET	SKIP	STRING	SUBTYPE
SUBTYPE_CONSTRAINT	SUPERTYPE	THEN	TO
TOTAL_OVER	TYPE	UNIQUE	UNTIL
USE	VAR	WHERE	WHILE
WITH			

7.2.2 Зарезервированные слова, обозначающие операторы

Операторы, обозначенные зарезервированными словами, представлены в таблице 2. Определения этих операторов даны в разделе 12.

Таблица 2 — Зарезервированные слова, обозначающие операторы языка EXPRESS

AND	ANDOR	DIV	IN
LIKE	MOD	NOT	OR
XOR			

7.2.3 Встроенные константы

Имена встроенных констант представлены в таблице 3. Определения этих констант даны в разделе 14.

Таблица 3 — Зарезервированные слова, обозначающие константы языка EXPRESS

?	SELF	CONST_E	PI
FALSE	TRUE	UNKNOWN	

7.2.4 Встроенные функции

Имена встроенных функций представлены в таблице 4. Определения этих функций даны в разделе 15.

Таблица 4 — Зарезервированные слова, являющиеся именами функций языка EXPRESS

ABS	ACOS	ASIN	ATAN
BLENGTH	COS	EXISTS	EXP
FORMAT	HIBOUND	HIINDEX	LENGTH
LOBOUND	LOG	LOG2	LOG 10
LOINDEX	NVL	ODD	ROLESOF
SIN	SIZEOF	SQRT	TAN
TYPEOF	USEDIN	VALUE	VALUE_IN
VALUE_UNIQUE			

7.2.5 Встроенные процедуры

Имена встроенных процедур представлены в таблице 5. Определения этих процедур даны в разделе 16.

Таблица 5 — Зарезервированные слова, являющиеся именами процедур языка EXPRESS

INSERT	REMOVE

7.3 Знаки

Знаки являются специальными символами или группами специальных символов, имеющими особое значение в языке EXPRESS. Знаки используются в языке EXPRESS как разделители и операторы. Разделители используются для начала, разделения или завершения смежных лексических или синтаксических элементов. Интерпретация этих элементов была бы невозможна без разделителей. Операторы обозначают выполнение действий над операндами, связанными с оператором. Описание операторов дано в разделе 12. Знаки языка EXPRESS представлены в таблице 6.

Таблица 6 — Знаки языка EXPRESS

.	'	,	:
*	+	-	=
%	'	\	/
<	>	[	]
{	}	|	@
(	)	<=	< >
>=	<*	:=	||
**	– –	(*	*)
:=:	:<>:		

7.4 Идентификаторы

Идентификаторы являются именами, присвоенными объявленным в схеме элементам (см. 9.3), включая саму схему. Идентификатор не должен совпадать с зарезервированными словами языка EXPRESS.

```
Синтаксис:
  143 simple_id = letter { letter | digit | '_' } .
  128 letter = 'a' | 'b' | 'c' | 'd' | 'e' | 'f' | 'g' | 'h' | 'i' | 'j' | 'k' |
               'l' | 'm' | 'n' | 'o' | 'p' | 'q' | 'r' | 's' | 't' | 'u' | 'v' |
               'w' | 'x' | 'y' | 'z'.
  124 digit = '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9'.
```

Первый символ идентификатора должен быть буквой. Остальные символы (при их наличии) могут являться любой комбинацией букв, цифр и символа подчеркивания.

При разработке синтаксического анализатора языка EXPRESS необходимо задать максимальное число символов в идентификаторе, распознаваемых данным анализатором, используя в качестве руководства приложение Е.

7.5 Литералы

Литерал является самоопределяемой константой. Тип литерала зависит от композиции символов, формирующих лексему. Литералы могут быть следующих типов: двоичный, целочисленный, действительный, строковый и логический.

```
Синтаксис:
  251 literal = binary_literal | integer_literal | real_literal |
                string_literal | logical_literal.
```

7.5.1 Двоичный литерал

Двоичный литерал представляет значение двоичного типа данных и состоит из символа «%», за которым следует один или более битов (0 или 1).

```
Синтаксис:
  139 binary_literal = '%' bit { bit } .
  123 bit = '0' | '1'.
```

При разработке синтаксического анализатора языка EXPRESS необходимо задать максимальное число битов в двоичном литерале, распознаваемое данным анализатором, используя в качестве руководства приложение Е.

Пример — Правильная запись двоичного литерала:

%0101001100

7.5.2 Целочисленный литерал

Целочисленный литерал представляет значение целого типа данных и состоит из одной или более цифр.

```
Синтаксис:
  141 integer_literal = digits .
  125 digits = digit { digit } .
  124 digit = '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9'.
```

Примечание — Знак целочисленного литерала не моделируется в синтаксисе, так как в языке EXPRESS в синтаксисе выражений используется концепция унарных операторов.

При разработке синтаксического анализатора языка EXPRESS необходимо задать максимальное целое значение для целочисленного литерала, распознаваемое данным анализатором, используя в качестве руководства приложение Е.

Пример — Правильная запись целочисленных литералов:

4016

38

7.5.3 Действительный литерал

Действительный литерал представляет значение действительного типа данных и состоит из мантиссы и необязательного показателя степени; мантисса должна содержать десятичную точку.

Примечание — Знак действительного литерала не моделируется в синтаксисе, так как в языке EXPRESS в синтаксисе выражений используется концепция унарных операторов.

Синтаксис:

```
142 real_literal = integer_literal |
                   ( digits '.' [ digits ] [ 'e' [ sign ] digits ] ) .
125 digits = digit { digit } .
124 digit = '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9' .
304 sign = '+' | '–' .
```

При разработке синтаксического анализатора языка EXPRESS необходимо задать максимальную точность и максимальный показатель степени действительного литерала, распознаваемые данным анализатором, используя в качестве руководства приложение Е.

Примеры

1 Правильная запись действительных литералов:

1.E6 — ***Символ «E» может быть введен на верхнем или на нижнем регистре.***

3.5e-5

359.62

2 Неправильная запись действительных литералов:

.001 — ***По крайней мере одна цифра должна предшествовать десятичной точке.***

1e10 — ***Десятичная точка должна быть частью литерала.***

1. e10 — ***Пробел не является частью действительного литерала.***

7.5.4 Строковый литерал

Строковый литерал представляет значение строкового типа данных. Существуют две формы строкового литерала: простой строковый литерал и кодированный строковый литерал. Простой строковый литерал состоит из последовательности символов из набора символов языка EXPRESS (см. 7.1), заключенной в апострофы ('). Апостроф в составе простого строкового литерала представляется двумя последовательными апострофами. Кодированный строковый литерал состоит из четырехоктетного кодированного представления каждого символа в последовательности символов ИСО/МЭК 10646-1, заключенной в кавычки (").

Кодирование определяется следующим образом:

- первый октет — группа ИСО/МЭК 10646-1, в которой определен символ;
- второй октет — плоскость ИСО/МЭК 10646-1, в которой определен символ;
- третий октет — строка ИСО/МЭК 10646-1, в которой определен символ;
- четвертый октет — позиция ИСО/МЭК 10646-1, в которой определен символ.

Последовательность октетов должна определять один из допустимых символов ИСО/МЭК 10646-1.

Строковый литерал никогда не должен выходить за границу физической строки, то есть символ «новая строка» не должен встречаться между апострофами, ограничивающими строковый литерал.

Синтаксис:

```
310 string_literal = simple_string_literal | encoded_string_literal .
144 simple_string_literal = \q { ( \q \q ) | not_quote | \s | \x9 | \xA | \xD } \q .
134 not_quote = not_paren_star_quote_special | letter | digit | '(' | ')' | '*' .
132 not_paren_star_quote_special =  '!' | '"' | '#' | '$' | '%' | '&' | '+' | ',' |
                                    '-' | '.' | '/' | ':' | ';' | '<' | '=' | '>' |
                                    '?' | '@' | '[' | '\' | ']' | '^' | '_' | '`' |
                                    '{' | '|' | '}' | '~'.
128 letter = 'a' | 'b' | 'c' | 'd' | 'e' | 'f' | 'g' | 'h' | 'i' | 'j' | 'k' | 'l' |
             'm' | 'n' | 'o' | 'p' | 'q' | 'r' | 's' | 't' | 'u' | 'v' | 'w' | 'x' |
             'y' | 'z'.
124 digit = '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9' .
140 encoded_string_literal = '"' encoded_character { encoded_character } '"' .
126 encoded_character = octet octet octet octet .
136 octet = hex_digit hex_digit .
127 hex_digit = digit | 'a' | 'b' | 'c' | 'd' | 'e' | 'f' .
```

При разработке синтаксического анализатора языка EXPRESS необходимо задать максимальное число символов простого строкового литерала, распознаваемое данным анализатором, используя в качестве руководства приложение Е.

При разработке синтаксического анализатора языка EXPRESS необходимо задать максимальное число октетов (которое должно быть кратно четырем) кодированного строкового литерала, распознаваемое данным анализатором, используя в качестве руководства приложение Е.

Примеры
1 Правильная запись простых строковых литералов:
'Baby needs a new pair of shoes!'
Значением данного литерала является **Baby needs a new pair of shoes!**
'Ed''s Computer Store'
Значением данного литерала является **Ed's Computer Store**
2 Неправильная запись простых строковых литералов:
'Ed's Computer Store'
Литерал всегда должен содержать четное число апострофов.
'Ed" s Computer
Store'
Литерал выходит за границы физической строки.
3 Правильная запись кодированных строковых литералов:
"00000041"
Значением данного литерала является **A.**
"000000C5"
Значением данного литерала является **Å.**
"0000795E00006238"
Значением данного литерала являются японские иероглифы **神戸** ***, обозначающие*** **Kobe.**
4 Неправильная запись кодированных строковых литералов:
"000041"
Октеты должны быть сгруппированы по четыре.
"00000041 000000C5"
Между кавычками разрешены только шестнадцатеричные символы.

7.5.5 Логический литерал

Логический литерал представляет значение логического или булевого типа данных и является одной из встроенных констант **TRUE**, **FALSE** или **UNKNOWN.**

Примечание — Константа **UNKNOWN** несовместима с булевым типом данных.

Синтаксис:
255 logical_literal = FALSE | TRUE | UNKNOWN .

8 Типы данных

В данном разделе определены типы данных, используемые в языке EXPRESS. Каждый атрибут, локальная переменная или формальный параметр имеет связанный с ним тип данных.

Типы данных подразделяются на простые, агрегированные, именованные, конструкционные и обобщенные. Кроме того, типы данных подразделяются, в соответствии с их применением, на конкретизирующие, параметрические, базисные и именованные типы данных. Взаимосвязь между этими двумя классификациями определена в 8.6.

Операции, которые могут выполняться над значениями этих типов данных, определены в разделе 12.

8.1 Простые типы данных

Простые типы данных определяют области определения элементарных единиц данных в языке EXPRESS. То есть они не могут быть разделены на элементы, распознаваемые в языке EXPRESS. Простыми типами данных являются **NUMBER** (числовой), **REAL** (действительный), **INTEGER** (целочисленный), **STRING** (строковый), **BOOLEAN** (булев), **LOGICAL** (логический) и **BINARY** (двоичный).

8.1.1 Числовой тип данных

Областью определения типа данных **NUMBER** являются все числовые значения в языке EXPRESS. Числовой тип данных должен использоваться, когда не важно более конкретное представление чисел.

Синтаксис:
261 number_type = NUMBER .

Пример — Поскольку контекст параметра* size *(размер) может быть неизвестен, то и не известно как его правильно представить. Например, численность толпы на футбольном матче может выражаться целым числом (тип данных* INTEGER*), а площадь поля — действительным (тип данных* REAL*).

size : NUMBER ;

Примечание — В последующих редакциях настоящего стандарта могут быть введены новые конкретизации типа данных NUMBER, например, комплексные числа.

8.1.2 Действительный тип данных

Областью определения типа данных **REAL** являются все рациональные, иррациональные и экспоненциально представленные числа. Данный тип данных является конкретизацией типа данных **NUMBER.**

Синтаксис:
278 real_type = REAL ['(' precision_spec ')'] .
268 precision_spec = numeric_expression .

Рациональные и иррациональные числа имеют неограниченное разрешение и являются точными. Числа в экспоненциальной форме представляют величины, известные лишь с определенной точностью. Объект **precision_spec** выражается в терминах значащих цифр.

Действительное число представляется мантиссой и необязательным показателем степени. Число цифр, составляющих мантиссу после удаления всех стоящих впереди нулей, является числом значащих цифр. Известная точность значения определяется числом первых цифр, необходимых для конкретного приложения.

Правила и ограничения:

a) Объект **precision_spec** задает необходимое минимальное число цифр разрешения. Значением соответствующего этому объекту выражения должно быть положительное целое число.

b) Если спецификация разрешения не задана, то точность действительного числа не ограничена.

8.1.3 Целочисленный тип данных

Областью определения типа данных **INTEGER** являются все целые числа. Данный тип данных является конкретизацией типа данных **REAL.**

Синтаксис:
241 integer_type = INTEGER .

Пример — В данном примере целочисленный тип данных использован для представления атрибута с именем* nodes. *Областью определения данного атрибута являются все целые числа без каких-либо ограничений.

```
ENTITY foo;
   nodes : INTEGER;
   ...
END_ENTITY;
```

8.1.4 Логический тип данных

Областью определения типа данных **LOGICAL** являются три литерала: **TRUE**, **FALSE** и **UNKNOWN.**

Синтаксис:
256 logical_type = LOGICAL .

Значения логического типа данных упорядочены следующим образом: **FALSE < UNKNOWN < TRUE**. Тип данных **LOGICAL** совместим с типом данных **BOOLEAN**, за исключением того, что булевой переменной не может быть присвоено значение **UNKNOWN**.

8.1.5 Булев тип данных

Областью определения типа данных **BOOLEAN** являются два литерала: **TRUE** и **FALSE**. Тип данных **BOOLEAN** является конкретизацией типа данных **LOGICAL**.

Синтаксис:
182 boolean_type = BOOLEAN .

Для значений типа данных **BOOLEAN** установлен тот же порядок, что и для значений типа данных **LOGICAL: FALSE < TRUE**.

Пример — В данном примере атрибут с именем **planar**, ***представлен типом данных*** **BOOLEAN**. ***Значением для*** **planar**, ***связанным с экземпляром*** **surface**, ***может быть либо*** **TRUE**, ***либо*** **FALSE**.

```
ENTITY surface;
 planar : BOOLEAN;
 ...
END_ENTITY;
```

8.1.6 Строковый тип данных

Областью определения типа данных **STRING** являются последовательности символов. Символами, допустимыми в строковых значениях, являются символы ИСО/МЭК 10646-1 из позиций 09, 0A, 0D и графические символы, расположенные в позициях от 20 до 7E и от A0 до 10FFFE.

Синтаксис:

```
311 string_type = STRING [ width_spec ] .
341 width_spec = '(' width ')' [ FIXED ] .
340 width = numeric_expression .
```

Тип данных **STRING** может иметь как фиксированную, так и переменную длину (число символов). Если фиксированная длина строки конкретно не указана (посредством зарезервированного слова **FIXED** в определении), то строка имеет переменную длину.

Областью определения типа данных **STRING** фиксированной длины является множество всех последовательностей символов длины, точно указанной в определении типа.

Областью определения типа данных **STRING** переменной длины является множество всех последовательностей символов длины, меньшее или равное максимальной длине, указанной в определении типа.

Если длина не указана, то областью определения является множество всех последовательностей символов без ограничения на длину этих последовательностей.

Адресация к подстрокам и отдельным символам может осуществляться посредством подстрочных индексов, как описано в 12.5.

В пределах строки имеет значение регистр ввода букв (верхний или нижний).

Правило: значением выражения **width** должно быть положительное целое число.

Примеры

1 В данном примере определена строка переменной длины, значения которой не имеют заданной максимальной длины:

```
string1 : STRING;
```

2 В данном примере определена строка с максимальной длиной равной десяти символам, значения которой могут иметь длину от нуля до десяти символов:

```
string2 : STRING(10);
```

3 В данном примере определена строка фиксированной длины равной десяти символам,значения которой должны содержать ровно десять символов:

```
string3 : STRING(10) FIXED;
```

8.1.7 Двоичный тип данных

Областью определения типа данных **BINARY** являются последовательности битов, каждый из которых представляется 0 или 1.

Синтаксис:

```
181 binary_type = BINARY [ width_spec ] .
341 width_spec  = '(' width ')' [ FIXED ] .
340 width = numeric_expression .
```

Тип данных **BINARY** может иметь как фиксированную, так и переменную длину (число битов). Если конкретно не указана фиксированная длина (посредством зарезервированного слова **FIXED** в определении), то тип данных **BINARY** имеет переменную длину.

Областью определения типа данных **BINARY** фиксированной длины является множество всех последовательностей битов длины, точно указанной в определении типа.

Областью определения типа данных **BINARY** переменной длины является множество всех последовательностей битов длины меньшей или равной максимальной длине, указанной в определении типа. Если длина не указана, то областью определения является множество всех последовательностей битов без ограничения на длину этих последовательностей.

Адресация к частям последовательности битов и отдельным битам может осуществляться посредством подстрочных индексов, как описано в 12.3.

Правило: значением выражения **width** должно быть положительное целое число.

Пример — Данный фрагмент может быть использован для хранения информации о шрифте символов:

```
ENTITY character;
  representation : ARRAY [1:20] OF BINARY (8) FIXED ;
END_ENTITY;
```

8.2 Агрегированные типы данных

Областями определения агрегированных типов данных являются совокупности значений заданного базисного типа данных (см. 8.6.1). Эти значения базисного типа данных называются элементами агрегированной совокупности. В языке EXPRESS определены четыре вида агрегированных типов данных: **ARRAY** (массив), **LIST** (список), **BAG** (пакет) и **SET** (набор). Значения каждого вида агрегированного типа данных имеют разные свойства. Тип данных **AGGREGATE** является обобщением этих четырех видов агрегированных типов данных (см. 9.5.3.1).

Тип данных **ARRAY** представляет упорядоченное множество фиксированного размера, индексированное последовательностью целых чисел.

Пример — Матрица преобразования (в геометрии) может быть определена как массив массивов (чисел).

Тип данных **LIST** представляет последовательность элементов, доступ к которым осуществляется по их позициям. Число элементов в списке может изменяться и быть ограничено в определении типа данных.

Пример — Операции технологического маршрута могут быть представлены списком. Операции упорядочены и могут быть добавлены или удалены из технологического маршрута.

Тип данных **BAG** представляет неупорядоченное множество, в котором разрешены повторяющиеся элементы. Число элементов в пакете может изменяться и быть ограничено в определении типа данных.

Пример — Совокупность крепежных деталей, используемых при сборке, может быть представлено пакетом. В состав его элементов может входить некоторое число одинаковых болтов, но неважно, который из них используется в конкретном отверстии.

Тип данных **SET** представляет неупорядоченное множество элементов, в котором нет двух одинаковых элементов. Число элементов в наборе может изменяться и быть ограничено в определении типа данных.

Пример — Множество людей во всем мире является набором.

Примечание — Агрегированные типы данных в языке EXPRESS являются одномерными. Объекты, обычно рассматриваемые как многомерные (например, математические матрицы), могут быть представлены агрегированным типом данных, базисным типом которого является другой агрегированный тип данных. Таким образом, агрегированные типы данных могут быть вложенными на произвольную глубину, обеспечивая представление структур данных любой размерности.

Пример — Можно определить структуру **LIST [1:3] OF ARRAY [5:10] OF INTEGER,** ***которая в действительности является двумерной.***

8.2.1 Тип данных ARRAY

Областью определения типа данных **ARRAY** являются индексированные совокупности подобных элементов фиксированного размера. Нижняя и верхняя границы, задаваемые выражениями, имеющими целочисленные значения, определяют диапазон значений индекса и, следовательно, размер массива. В определении типа данных **ARRAY** может факультативно устанавливаться, что в массиве не могут присутствовать одинаковые элементы. Кроме того, может быть установлено, что элементы массива могут не присутствовать на всех индексированных позициях.

Синтаксис:

```
175 array_type = ARRAY bound_spec OF [ OPTIONAL ] [ UNIQUE ] instantiable_type .
185 bound_spec = '[' bound_1 ':' bound_2 ']' .
183 bound_1 = numeric_expression .
184 bound_2 = numeric_expression .
```

Пусть *m* является нижней границей, а *n* — верхней, тогда в массиве имеется ровно *n* — *m* + 1 элементов. Эти элементы проиндексированы подстрочными индексами от *m* до *n* включительно (см. 12.6.1).

Примечание — Границы массива могут быть положительными, отрицательными или равными нулю, но не могут быть неопределенными (?) (см. 14.2).

Правила и ограничения:

a) Оба выражения в спецификации границ (**bound_1** и **bound_2**) должны иметь целочисленные значения. Ни одно из них не должно иметь неопределенного (?) значения.

b) Выражение **bound_1** задает нижнюю границу массива. Оно задает наименьшее значение индекса, допустимое для элементов массива этого типа данных.

c) Выражение **bound_2** задает верхнюю границу массива. Оно задает наибольшее значение индекса, допустимое для элементов массива этого типа данных.

d) Значение выражения **bound_1** должно быть меньше или равно значению выражения **bound_2**.

e) Если указано ключевое слово **OPTIONAL**, то массив этого типа данных может иметь неопределенное (?) значение в одной или нескольких индексированных позициях.

f) Если ключевое слово **OPTIONAL** не указано, то массив этого типа данных не должен содержать неопределенных (**?**) значений ни в одной индексированной позиции.

g) Если указано ключевое слово **UNIQUE**, то каждый элемент массива этого типа данных должен отличаться (то есть не быть эквивалентным экземпляром) от любого другого элемента того же массива.

Примечание — Оба ключевых слова **OPTIONAL** и **UNIQUE** могут использоваться в одном и том же определении типа данных **ARRAY**. Это не исключает многократного появления неопределенных (**?**) значений элементов массива, поскольку сравнение неопределенных (**?**) значений дает результат **UNKNOWN** и, следовательно, условие уникальности не нарушается.

Пример — Данный пример показывает, как объявляется многомерный массив.

```
sectors : ARRAY [ 1 : 10 ] OF     -- первое измерение
              ARRAY [ 11 : 14 ] OF    -- второе измерение
              UNIQUE something;
```

***Первый массив содержит 10 элементов типа данных* ARRAY [11:14] OF UNIQUE something. *Атрибут с именем* sectors *содержит всего 40 элементов типа данных* something. *В каждом из массивов* ARRAY [11:14] *не может быть повторяющихся элементов. Однако один и тот же экземпляр объекта* something *может присутствовать в двух разных массивах* ARRAY [11:14] *одного экземпляра атрибута* sectors.**

8.2.2 Тип данных LIST

Областью определения типа данных **LIST** являются последовательности подобных элементов. Необязательные нижняя и верхняя границы, задаваемые выражениями, имеющими целочисленные значения, определяют минимальное и максимальное число элементов в совокупности, определенной типом данных **LIST**. В определении типа данных **LIST** может быть факультативно установлено, что в списке не могут присутствовать одинаковые элементы.

Синтаксис:

```
250 list_type = LIST [ bound_spec ] OF [ UNIQUE ] instantiable_type .
185 bound_spec = '[' bound_1 ':' bound_2 ']' .
183 bound_1 = numeric_expression .
184 bound_2 = numeric_expression .
```

Правила и ограничения:

a) Выражение **bound_1** должно иметь целочисленное значение, большее или равное нулю. Оно задает нижнюю границу, определяющую минимальное число элементов, которое может содержаться в списке этого типа данных. Выражение **bound_1** не должно иметь неопределенного (**?**) значения.

b) Выражение **bound_2** должно иметь целочисленное значение, большее или равное значению **bound_1**, либо неопределенное (**?**) значение. Оно задает верхнюю границу, определяющую максимальное число элементов, которое может содержаться в списке этого типа данных. Если значение данного выражения является неопределенным (**?**), то число элементов в списке этого типа данных не ограничено сверху.

c) Если элемент **bound_spec** опущен, то границы списка определяются как **[0:?]**.

d) Если указано ключевое слово **UNIQUE**, то каждый элемент в списке этого типа данных должен отличаться (то есть не являться эквивалентным экземпляром) от любого другого элемента того же списка.

Пример — В данном примере определен список массивов. Список может содержать от нуля до десяти массивов. Каждый массив из десяти целых чисел должен отличаться от других массивов в данном списке.

complex_list : LIST[0:10] OF UNIQUE ARRAY[1:10] OF INTEGER;

8.2.3 Тип данных BAG

Областью определения типа данных **BAG** являются неупорядоченные совокупности подобных элементов. Необязательные нижняя и верхняя границы, задаваемые выражениями, имеющими целочисленные значения, определяют минимальное и максимальное число элементов в совокупности, определенной типом данных **BAG**.

Синтаксис:

```
180 bag_type = BAG [ bound_spec ] OF instantiable_type .
185 bound_spec = '[' bound_1 ':' bound_2 ']'.
183 bound_1 = numeric_expression .
184 bound_2 = numeric_expression .
```

Правила и ограничения:

a) Выражение **bound_1** должно иметь целочисленное значение, большее или равное нулю. Оно задает нижнюю границу, определяющую минимальное число элементов, которое может содержаться в экземпляре пакета этого типа данных. Выражение **bound_1** не должно иметь неопределенного (**?**) значения.

b) Выражение **bound_2** должно иметь целочисленное значение, большее или равное значению **bound_1**, либо неопределенное (**?**) значение. Оно задает верхнюю границу, определяющую максимальное число элементов, которое может содержаться в экземпляре пакета этого типа данных. Если значение данного выражения является неопределенным (**?**), то число элементов в экземпляре пакета этого типа данных не ограничено сверху.

c) Если элемент **bound_spec** опущен, то границы пакета определяются как **[0:?]**.

Пример — В данном примере атрибут **a_bag_of_points** ***определен как пакет объектов*** **point** ***(объекты*** **point** ***относятся к именованному типу данных, который объявлен в другом месте).***

a_bag_of_points : BAG OF point;

Экземпляр атрибута **a_bag_of_points** ***может содержать ни одного или несколько объектов*** **point**. ***Один и тот же экземпляр объекта*** **point** ***может появиться несколько раз в экземпляре*** **a_bag_of_points.**

Если требуется, чтобы экземпляр атрибута **a_bag_of_points** ***содержал, по крайней мере, один элемент, то в спецификации нижняя граница должна быть определена следующим образом:***

a_bag_of_points : BAG [l:?] OF point;

Экземпляр атрибута **a_bag_of_points** ***теперь должен содержать, по крайней мере, один объект*** **point.**

8.2.4 Тип данных SET

Областью определения типа данных **SET** являются неупорядоченные совокупности подобных элементов. Тип данных **SET** является конкретизацией типа данных **BAG**. Необязательные нижняя и верхняя границы, задаваемые выражениями, имеющими целочисленные значения, определяют минимальное и максимальное число элементов в совокупности, определенной типом данных **SET**. Определенная типом данных **SET** совокупность не должна содержать двух или более одинаковых элементов.

Синтаксис:

```
303 set_type = SET [ bound_spec ] OF instantiable_type .
185 bound_spec = '[' bound_1 ':' bound_2 ']'.
183 bound_1 = numeric_expression .
184 bound_2 = numeric_expression .
```

Правила и ограничения:

a) Выражение **bound_1** должно иметь целочисленное значение, большее или равное нулю. Оно задает нижнюю границу, определяющую минимальное число элементов, которое может содержаться в экземпляре набора этого типа данных. Выражение **bound_1** не должно иметь неопределенного (**?**) значения.

b) Выражение **bound_2** должно иметь целочисленное значение, большее или равное значению **bound_1**, либо неопределенное (**?**) значение. Оно задает верхнюю границу, определяющую максимальное число элементов, которое может содержаться в экземпляре набора этого типа данных. Если

значение данного выражения является неопределенным (**?**), то число элементов в экземпляре набора этого типа данных не ограничено сверху.

c) Если элемент **bound_spec** опущен, то границы набора определяются как [**0:?**].

d) Каждый элемент в экземпляре типа данных **SET** должен отличаться (то есть не являться эквивалентным экземпляром) от любого другого элемента того же экземпляра набора.

Пример — В данном примере атрибут **a_set_of_points** ***определен как набор объектов*** **point** ***(объекты*** **point** ***относятся к именованному типу данных, который объявлен в другом месте).***

a_set_of_points : SET OF point;

Атрибут **a_set_of_points** ***может содержать ни одного или несколько объектов*** **point.** ***Каждый экземпляр объекта*** **point** ***(в экземпляре набора) должен отличаться от любого другого объекта*** **point** ***в наборе.***

Если требуется, чтобы набор содержал не более 15 объектов **point,** ***то в спецификации верхняя граница должна быть определена следующим образом:***

a_set_of_points : SET [0:15] OF point;

Теперь экземпляр атрибута **a_set_of_points** ***может содержать не более 15 точек.***

8.2.5 Уникальность значений в агрегированных структурах

Уникальность среди элементов агрегированных структур основана на сравнении экземпляров (см. 12.2.2). К агрегированным структурам может быть предъявлено требование уникальности значений их элементов посредством использования функции **VALUE_UNIQUE** (см. 15.29).

Пример — Определение набора с уникальными значениями:

```
TYPE value_unique_set = SET OF a;
WHERE
    wrl : value_unique(SELF);
END_TYPE;
```

Примечание — Определяемая разработчиком уникальность значений может быть задана посредством двух функций с именами, например, **my_equal** и **my_unique**, как показано в следующем псевдокоде:

```
FUNCTION my_equal (v1,v2: GENERIC: gen): LOGICAL;
  (*Функция my_equal возвращает значение TRUE, если v1 «равно» v2 *)
END_FUNCTION;
FUNCTION my_unique (c: AGGREGATE OF GENERIC): LOGICAL;
  (* Функция my_unique возвращает значение FALSE, если два элемента из с имеют одинаковое «значение»,
    или возвращает значение UNKNOWN, если любой из сравниваемых элементов имеет значение
    UNKNOWN, в противном случае возвращает значение TRUE *)
LOCAL
    result   : LOGICAL;
    unknownp : BOOLEAN := FALSE;
END_LOCAL;
IF (SIZEOF(c) = 0) THEN
    RETURN(TRUE); END_IF;
REPEAT i := LOINDEX(c) TO (HIINDEX(c) - 1);
    REPEAT j := (i+1) TO HIINDEX(c);
        result := my_equal(c[i], c[j]);
        IF (result = TRUE) THEN
          RETURN(FALSE); END_IF;
        IF (result = UNKNOWN) THEN
          unknownp := TRUE; END IF;
    END REPEAT;
END_REPEAT;
IF unknownp THEN
    RETURN(UNKNOWN);
ELSE
    RETURN(TRUE);
END_IF;
END_FUNCTION;
```

Функция **my_equal** должна иметь следующие свойства, позволяющие формировать классы эквивалентности. Ниже через **S** обозначен рассматриваемый набор объектов, а **my_equal(i, j)**, где **i** и **j** принадлежат **S**, возвращает одно из значений **[FALSE, UNKNOWN, TRUE]**:

my_equal(i, i) имеет значение **TRUE** для всех **i** из **S** (так как в S нет неопределенных (**?**) значений, то не требуется, чтобы значением **my_equal (?, ?)** было **TRUE**);

my_equal(i, j) = my_equal(j, i) для всех **i** и **j** из **S**;

(my_equal(i, j) = TRUE) AND (my_equal(j, k) = TRUE) влечет за собой **(my_equal(i, k) = TRUE)** для всех **i, j, k** из **S.**

8.3 Именованные типы данных

Именованными типами данных являются типы данных, которые могут быть объявлены в формальной спецификации. Существуют два вида именованных типов данных: объектный и определенный. В данном подразделе определено обращение к именованным типам данных; объявление этих типов данных определено в разделе 9.

8.3.1 Объектный тип данных

Объектные типы данных устанавливаются объявлениями посредством ключевого слова **ENTITY** (см. 9.2). Объектный тип данных задается назначаемым пользователем идентификатором объекта. Обращение к объектному типу данных осуществляется посредством данного идентификатора.

Синтаксис:
152 entity_ref = entity_id .

Правило: элемент **entity_ref** должен быть ссылкой на объект, видимый в текущей области видимости (см. раздел 10).

Пример — В данном примере объектный тип данных **point** ***использован для представления атрибута.***

```
ENTITY point;
    x, y, z : REAL;
END_ENTITY;
ENTITY line;
    p0, p1 : point;
END_ENTITY;
```

Объект **line** ***имеет два атрибута с именами*** **p0** ***и*** **p1.** ***Типом данных каждого из этих атрибутов является*** **point.**

8.3.2 Определенный тип данных

Определенные типы данных устанавливаются объявлениями посредством ключевого слова **TYPE** (см. 9.1). Определенный тип данных задается назначаемым пользователем идентификатором типа. Обращение к определенному типу данных осуществляется посредством данного идентификатора.

Синтаксис:
162 type_ref = type_id .

Правило: элемент **type_ref** должен быть именем определенного типа данных, видимым в текущей области видимости (см. раздел 10).

Пример — В данном примере определенный тип данных использован для указания единиц измерения, связанных с атрибутом.

```
TYPE volume = REAL;
END_TYPE;
ENTITY PART;
...
  bulk : volume;
END_ENTITY;
```

Атрибут с именем **bulk** ***представлен действительным числом, но использование определенного типа данных*** **volume** ***помогает пояснить смысл и контекст данного действительного числа. В данном примере действительное число определяет объем, а не какое-либо другое понятие, значение которого может также определяться действительным числом.***

8.4 Конструкционные типы данных

Существуют два вида конструкционных типов данных в языке EXPRESS: перечисляемый (**ENUMERATION**) и выбираемый (**SELECT**). Эти типы данных имеют сходную синтаксическую структуру и могут использоваться только для задания базисных представлений определенных типов данных (см. 9.1).

8.4.1 Перечисляемый тип данных

Областью определения перечисляемого типа данных является множество имен. Размер данного множества имен определяется в зависимости от типа перечисляемого типа данных. Различают следующие перечисляемые типы данных:

- наращиваемый перечисляемый тип данных;
- перечисляемый тип данных, расширяющий наращиваемый перечисляемый тип данных, другими словами, основанный на наращиваемом перечисляемом типе данных;
- перечисляемый тип данных, который не является ни наращиваемым, ни расширяющим.

Имена, объявляемые посредством ключевого слова **ENUMERATION**, могут быть только допустимыми значениями перечисляемого типа данных. Каждое имя из данной области определения называется элементом перечисления и обозначается **enumeration_id**.

Областью определения перечисляемого типа данных, который не является ни наращиваемым, ни расширяющим, задаваемой при его объявлении, является упорядоченное множество элементов перечисления, указанных при его объявлении.

Областью определения наращиваемого перечисляемого типа данных является множество элементов перечисления, указанных при его объявлении, плюс объединение множеств элементов перечислений, заключающее в себе области определения всех расширяющих перечисляемых типов данных. Наращиваемый перечисляемый тип данных является обобщением основанных на нем перечисляемых типов данных. Наращиваемый перечисляемый тип данных задается посредством ключевого слова **EXTENSIBLE**.

Областью определения расширяющего перечисляемого типа данных является множество элементов перечисления, указанных при его объявлении, плюс элементы перечисления, указанные непосредственно (а не через расширение) в наращиваемом перечисляемом типе данных, на котором он основан. Расширяющий перечисляемый тип данных задается посредством ключевого слова **BASED_ON**.

Перечисляемый тип данных может быть как наращиваемым, так и расширяющим перечисляемым типом данных. Наращиваемый перечисляемый тип данных может быть задан без элементов перечисления, а также основан на другом наращиваемом перечисляемом типе данных без задания элементов перечисления, расширяющих данное базисное перечисление. Взаимосвязь «основан на» является транзитивной, то есть расширяющее перечисление остается основанным на наращиваемом перечислении самого верхнего уровня даже при наличии нескольких уровней взаимосвязей «основан на»; расширяющее перечисление включает все элементы, как промежуточных наращиваемых перечислений, так и наращиваемого перечисления самого верхнего уровня.

Примечания

1 Наращиваемое перечисление, расширенное два или более раза в одном контексте, может иметь большую область определения, чем его расширения, и в этом случае оно действительно является обобщением.

2 В первой редакции настоящего стандарта упорядочение элементов перечисления определяло упорядочение значений. В настоящем стандарте не определено никакого упорядочения, за исключением указанного ниже в правиле по перечислению d). Это сделано, чтобы обеспечить существование наращиваемых перечисляемых типов данных, в которых упорядочение расширений не может быть определено.

Синтаксис:

```
213 enumeration_type = [ EXTENSIBLE ] ENUMERATION [ ( OF
                          enumeration_items ) | enumeration_extension ] .
211 enumeration_items = '(' enumeration_id { ',' enumeration_id } ')' .
209 enumeration_extension = BASED_ON type_ref [ WITH enumeration_items ] .
```

Правила и ограничения:

a) Перечисляемый тип данных должен использоваться только в качестве базисного типа данных для определенного типа данных.

b) Перечисляемый тип данных может быть расширен, только если в его определении указано ключевое слово **EXTENSIBLE**.

c) Элемент **type_ref** в **enumeration_extension** должен быть ссылкой на наращиваемый перечисляемый тип.

d) В целях сравнения, упорядочение значений перечисляемого типа данных, который не является ни наращиваемым, ни расширяющим, может быть определено их относительной позицией в списке **enumeration_id**; первый встретившийся элемент должен быть меньше второго, второй — меньше третьего и т.д.

e) Не существует упорядочения значений для наращиваемого перечисления или расширяющего перечисления.

f) Для перечисления, не являющегося ни наращиваемым, ни расширяющим, в качестве его области определения должны быть заданы элементы перечисления.

g) Для перечисления, не являющегося наращиваемым, но являющегося расширяющим, должны быть заданы элементы перечисления, расширяющие область определения наращиваемого перечисления, на котором оно основано.

h) Два разных перечисляемых типа данных могут содержать одинаковый элемент **enumeration_id** в своих множествах имен. Если перечисляемые типы данных не являются расширениями одного наращиваемого типа данных, то их элементы **enumeration_id** относятся к разным понятиям, даже если их локальные имена совпадают. В данном случае, любая ссылка на элемент **enumeration_id** (например, в выражении) должна быть уточнена посредством идентификатора типа данных, чтобы обеспечить однозначность ссылок. Ссылка при этом выглядит следующим образом: **type_id.enumeration_id**.

Примечание — Элемент **type_id**, используемый для уточнения элемента **enumeration_id**, всегда определяется как перечисляемый тип данных.

Пример — В данном примере перечисляемые типы данных использованы для демонстрации того, как могут двигаться различные виды транспортных средств.

```
TYPE car_can_move = ENUMERATION OF
    (left, right, backward, forward);
END_TYPE;
TYPE plane_can_move = ENUMERATION OF
    (left, right, backward, forward, up, down);
END_TYPE;
```

***Элемент перечисления* left *имеет два независимых определения, задаваемых каждым типом данных, компонентом которого он является. Не существует никакой связи между этими двумя определениями идентификатора* left. *Сама по себе ссылка на* left *или* right *является неоднозначной. Для того, чтобы разрешить данную неоднозначность, ссылка на любое из этих значений должна быть уточнена посредством указания имени типа данных, например,* car_can_move.left.

i) Наращиваемое перечисление и его расширения задают область определения, состоящую из элементов **enumeration_id**. В пределах данной области определения все появления одного и того же элемента **enumeration_id** обозначают одно и то же значение, даже если элемент **enumeration_id** определен в нескольких перечисляемых типах данных, относящихся к данной области определения.

***Пример — В данном примере один элемент перечисления с именем* red *используется в двух расширениях* stop_light *и* canadian_flag *области определения перечисляемого типа данных* colour.

```
TYPE colour = EXTENSIBLE ENUMERATION; END_TYPE;
TYPE stop_light = ENUMERATION BASED_ON colour WITH (red, yellow, green); END_TYPE;
TYPE canadian_flag = ENUMERATION BASED_ON colour WITH (red, white); END_TYPE;
```

j) Объявление типа, в котором объявляется перечисляемый тип данных, не должно содержать правила для области определения (**WHERE**).

Примечание — Приведенные выше правила обеспечивают то, что определенный тип данных дает имя перечисляемому типу данных, и определенный тип данных не является конкретизацией перечисляемого типа данных.

Пример — В данном примере показано, как наращиваемое перечисление может быть использовано для моделирования контекстно-зависимого понятия «одобрение». Элемент* general_approval *представляет наиболее общее понятие одобрения, однозначно определяя только два значения. Объявление* general_approval *как наращиваемого перечисления позволяет ему принимать контекстно-зависимые значения в схемах, в которых объявляются его расширения. При использовании данного элемента для представления области определения атрибута, допустимые значения атрибута становятся контекстно зависимыми.

```
SCHEMA s1;
TYPE general_approval = EXTENSIBLE ENUMERATION OF (approved, rejected);
END_TYPE;
END_SCHEMA;
SCHEMA s2;
USE FROM s1 (general_approval);
```

```
TYPE domain2_approval = EXTENSIBLE ENUMERATION BASED_ON general_approval WITH (pending);
END_TYPE;
END_SCHEMA;
SCHEMA s3;
USE FROM s1 (general_approval);
TYPE domain3_approval = EXTENSIBLE ENUMERATION BASED_ON general_approval WITH (cancelled);
END_TYPE;
END_SCHEMA;
SCHEMA s4;
USE FROM s2 (domain2_approval);
REFERENCE FROM s3 (domain3_approval);
TYPE specific_approval = ENUMERATION BASED_ON domain2_approval WITH (rework);
END_TYPE;
END_SCHEMA;
SCHEMA s5;
USE FROM s1 (general_approval);
TYPE redundant_approval = ENUMERATION BASED_ON general_approval WITH (approved);
END_TYPE;
END_SCHEMA;
```

В контексте схемы **s1: general_approval** ***имеет область определения*** **(approved, rejected).**

В контексте схемы **s2: general_approval** ***имеет область определения*** **(approved, rejected, pending); domain2_approval** ***имеет область определения*** **(approved, rejected, pending).**

В контексте схемы **s3: general_approval** ***имеет область определения*** **(approved, rejected, cancelled); domain3_approval** ***имеет область определения*** **(approved, rejected, cancelled).**

В контексте схемы **s4: general_approval** ***имеет область определения*** **(approved, rejected, pending, cancelled, rework); domain2_approval** ***имеет область определения*** **(approved, rejected, pending, rework); domain3_approval** ***имеет область определения*** **(approved, rejected, cancelled); specific_approval** ***имеет область определения*** **(approved, rejected, pending, rework).**

В контексте схемы **s5: general_approval** ***имеет область определения*** **(approved, rejected); redundant_approval** ***имеет область определения*** **(approved, rejected).**

8.4.2 Выбираемый тип данных

Выбираемый тип данных определяет тип данных, позволяющий выбирать среди нескольких именованных типов данных. Выбираемый тип данных является обобщением именованных типов данных в своей области определения. Определенный тип данных, для которого выбираемый тип данных является базисным представлением, может добавлять ограничения на свою область определения посредством объявления локальных правил. Выбираемый тип данных может быть или не быть наращиваемым.

Областью определения выбираемого типа данных, который не является ни наращиваемым, ни расширяющим, является объединение областей определения именованных типов данных из его списка выбора.

Областью определения наращиваемого выбираемого типа данных является объединение областей определения именованных типов данных из его собственного списка выбора плюс объединение областей определения всех расширяющих выбираемых типов данных. Наращиваемый выбираемый тип данных задается посредством ключевого слова **EXTENSIBLE**.

Областью определения расширяющего выбираемого типа данных являются именованные типы данных из его собственного списка выбора плюс именованные типы данных, указанные непосредственно (а не через расширение) в наращиваемом выбираемом типе данных, на котором он основан. Расширяющий выбираемый тип данных задается посредством ключевого слова **BASED_ON**.

Выбираемый тип данных может быть как наращиваемым, так и расширяющим выбираемым типом данных. Наращиваемый выбираемый тип данных может быть задан без списка выбора, а также основан на другом наращиваемом выбираемом типе данных без задавания списка выбора, расширяющего данный базисный список выбора.

Только наращиваемый выбираемый тип данных может быть ограничен наличием в своей области определения только экземпляров объектов посредством ключевого слова **GENERIC_ENTITY**. В данном случае все элементы списка выбора должны быть элементами обобщенного объекта, где элемент обобщенного объекта определяется как относящийся к объектному типу данных либо к списку выбора из элементов обобщенного объекта. Все расширения данного выбираемого типа данных должны быть выбираемыми типами данных обобщенного объекта и должны задаваться посредством ключевого слова **GENERIC_ENTITY**.

Синтаксис:

```
302 select_type = [ EXTENSIBLE [ GENERIC_ENTITY ] ] SELECT
                  [ select_list | select_extension ] .
301 select_list = '(' named_types { ',' named_types } ')' .
300 select_extension = BASED_ON type_ref [ WITH select_list ] .
```

Правила и ограничения:

a) Все элементы в списке выбора должны относиться к объектному или определенному типу данных.

b) Только выбираемый тип данных должен использоваться в качестве базисного типа для определенного типа данных.

c) Выбираемый тип данных может быть расширен, только если в его определении указано ключевое слово **EXTENSIBLE**.

d) Элемент **type_ref** в **select_extension** должен быть ссылкой на наращиваемый выбираемый тип.

e) Для выбираемого типа данных, не являющегося ни наращиваемым, ни расширяющим, должен быть задан в качестве его области определения непустой список выбора.

f) Для выбираемого типа данных, не являющегося наращиваемым, но являющегося расширяющим, должен быть задан непустой список выбора, расширяющий область определения наращиваемого выбираемого типа данных, на котором он основан.

Примечание — Значение выбираемого типа данных может быть значением нескольких именованных типов данных, указанных в списке выбора для данного выбираемого типа данных.

Примеры

1 Если a и b являются подтипами c, и они связаны выражением **ANDOR**, ***и существует тип данных, определенный как*** **SELECT (a, b)**, ***то может оказаться, что значением выбираемого типа данных будут одновременно a и b.***

2 Должен быть сделан выбор среди нескольких типов предметов в заданном контексте:

```
TYPE attachment_method = EXTENSIBLE SELECT(nail, screw);
END_TYPE;
TYPE permanent_attachment = SELECT BASED_ON attachment_method WITH (glue, weld);
END_TYPE;
ENTITY nail;
    length      : REAL;
    head_area : REAL;
END_ENTITY;
ENTITY screw;
    length  : REAL;
    pitch   : REAL;
END ENTITY;
ENTITY glue;
    composition  : material_composition;
    solvent      : material_composition;
END_ENTITY;
ENTITY weld;
    composition  : material_composition;
END_ENTITY;
ENTITY wall_mounting;
    mounting  : product;
    on        : wall;
    using     : attachment_method;
END ENTITY;
```

Элемент **wall_mounting** ***описывает соединение изделия*** **product** ***со стеной*** **wall** ***с использованием способа соединения*** **attachment_method**. ***Исходный способ соединения описывает способы разборного соединения. Эти способы затем расширяются, добавляя способы неразборного соединения*** **permanent_attachment**. ***При определении значения элемента*** **wall_mounting** ***будет использоваться значение атрибута*** **using**, ***то есть*** **nail**, **screw**, **glue** ***или*** **weld**.

8.5 Обобщенные типы данных

Синтаксис:

```
223 generalized_types = aggregate_type | general_aggregation_types | generic_entity_type | generic_type .
```

Обобщенные типы данных используются для определения обобщения некоторых других типов данных и могут быть использованы только в некоторых очень специфичных контекстах. Тип **GENERIC** является обобщением всех типов данных. Тип данных **AGGREGATE** является обобщением всех агрегированных типов данных. Общий агрегированный тип данных является обобщением агрегированных типов данных, смягчающим некоторые ограничения, обычно применяемые к агрегированным типам данных. Все эти типы данных определены в 9.5.3.

8.6 Классификация применения типов данных

В разделе 8 типы данных языка EXPRESS систематизированы по их сущности: простые типы данных, агрегированные типы данных, конструкционные типы данных, именованные типы данных и обобщенные типы данных. В настоящем подразделе определена классификация типов данных в соответствии с их применением.

В языке EXPRESS типы данных применяются шестью разными способами в качестве:

- типов данных элементов агрегированных типов данных;
- элементов списка выбора при определении или расширении выбираемого типа данных;
- базисных типов для типов данных;
- типов данных атрибутов объектных типов данных;
- типов данных констант;
- типов данных формальных параметров и локальных переменных в функциях и процедурах.

Кроме того, существует несколько специальных применений объектных типов данных, определенных в разделе 9, которые применяются не к другому классу типов данных, и поэтому в настоящем подразделе не рассматриваются.

Типы данных классифицируются в соответствии с их применением следующим образом:

- конкретизирующие типы данных применяются для представления элементов агрегированных структур и типов данных констант;
- параметрические типы данных применяются для представления явных и производных атрибутов, формальных параметров, результатов функций и локальных переменных в функциях и процедурах;
- базисные типы данных применяются для представления определенных типов данных;
- именованные типы данных применяются в качестве элементов списка выбора, то есть для возможных представлений значения выбираемого типа данных.

Некоторые классы типов данных могут применяться разными способами, в то время как другие могут применяться только в определенных контекстах. Данные различия представлены в таблице 7.

Таблица 7 — Применение типов данных

Типы данных	a	b	c	d
Простые	•	•	•	
Агрегированные	•	•	•	
Именованные	•	•	•*)	•
Конструкционные			•	
Обобщенные		•		

a) Конкретизирующие типы данных — представление элементов агрегированных структур и констант.

b) Параметрические типы данных — представление явных и производных атрибутов, формальных параметров, локальных переменных или результата функции.

c) Базисные типы данных — представление определенного типа (см. 9.1).

d) Именованные типы данных — возможные представления выбираемого типа данных.

*) Из именованных типов данных только определенный тип может применяться в качестве базисного типа данных.

Именованные типы данных определены в 8.3. Конкретизирующие, параметрические и базисные типы данных определены в 8.6.1 — 8.6.3.

8.6.1 Конкретизирующие типы данных

Конкретизирующие типы данных используются для представления констант, элементов агрегированных типов данных и атрибутов неабстрактных объектных типов данных (см. 9.2.1).

К конкретизирующим типам данных относятся простые, агрегированные и именованные типы данных.

Синтаксис:

```
240 instantiable_type = concrete_types | entity_ref .
193 concrete_types = aggregation_types | simple_types | type_ref .
```

Правила и ограничения:

a) Тип данных константы не должен быть абстрактным объектным типом данных (см. 9.4).

b) Тип данных любого атрибута неабстрактного объектного типа данных должен быть конкретизирующим типом данных или переобъявлен как таковой (см. 9.2.1).

8.6.2 Параметрические типы данных

Параметрические типы данных применяются для представления атрибутов объектных типов данных или формальных параметров алгоритмов (функций и процедур). Параметрические типы данных могут также применяться для представления результатов функций и локальных переменных, объявленных в алгоритмах.

К параметрическим типам данных относятся конкретизирующие и обобщенные типы данных. Другими словами, все типы данных языка EXPRESS являются параметрическими типами данных (при этом конструкционные типы данных могут применяться только в качестве основанных на них определенных типах данных).

Синтаксис:

266 parameter_type = generalized_types | named_types | simple_types .

223 generalized_types = aggregate type | general_aggregation_types | generic_entity_type | generic_type .

Правила и ограничения:

a) Любой параметрический тип данных, соответствующий спецификациям конкретизирующего типа данных, считается конкретизирующим типом данных для применений, в которых требуется конкретизирующий тип данных.

b) Общий агрегированный тип данных (см. 9.5.3.5), базовым типом которого является конкретизирующий тип данных, считается конкретизирующим типом данных.

Примечание — Синтаксическая конструкция, такая как **ARRAY[1:3] OF REAL**, соответствует двум синтаксическим порождениям — **aggregation_type** и **general_aggregation_type**. Она считается конкретизирующей независимо от того, какое из порождений требуется для соответствия синтаксису.

8.6.3 Базисные типы данных

Базисные типы данных применяются для представления определенных типов данных.

К базисным типам данных относятся простые, агрегированные, конструкционные и определенные типы данных.

Синтаксис:

332 underlying_type = concrete_types | constructed_ types .

193 concrete_types = aggregation_types | simple_types | type_ref.

9 Объявления

В данном разделе определены объявления, установленные в языке EXPRESS. Объявление в языке EXPRESS создает новый элемент языка EXPRESS и связывает с ним некоторый идентификатор. На элемент языка EXPRESS можно ссылаться в любом месте посредством имени связанного с ним идентификатора (см. раздел 10).

Основные возможности языка EXPRESS обеспечиваются посредством следующих объявлений:

- типа;
- объекта;
- ограничения на подтипы;
- схемы;
- константы;
- функции;
- процедуры;
- правила.

Объявления могут быть явными или неявными. В данном разделе описаны явные объявления. Неявные объявления описаны в данном разделе и последующих подразделах, наряду с элементами и условиями, при которых они устанавливаются.

9.1 Объявление типа

Объявление типа создает определенный тип данных (см. 8.3.2) и задает идентификатор для ссылки на данный тип. В частности, имя определенного типа данных объявляется как **type_id**. Представлением данного типа данных является **underlying_type**. Область определения определенного типа данных совпадает с областью определения базисного типа **underlying_type**, но может быть дополнительно ограничена конструкцией **where_clause** (при ее наличии). Определенный тип данных является конкретизацией базисного типа данных и, следовательно, совместимым с базисным типом. Исключением являются конструкционные типы данных, когда определенный тип данных используется для задания имени конструкционного типа данных, и фактически не является конкретизацией конструкционного типа данных даже в случае, если выбираемый тип данных **SELECT** ограничен правилом **WHERE**.

Примечание — Несколько определенных типов данных могут быть связаны с одним и тем же представлением. Имена могут помочь пользователю в понимании назначения (или контекста) применения базисного типа данных **underlying_type**.

Синтаксис:

```
327 type_decl = TYPE type_id '=' underlying_type ';' [ where_clause ] END_TYPE ';' .
332 underlying_type = concrete_types | constructed_types .
```

Правило: результатом объявлений типа **TYPE** не должны быть циклические определения типа.

Пример — Следующее объявление задает определенный тип данных с именем **person_name** ***и базовым представлением*** **STRING.** ***Определенный тип*** **person_name** ***после данного объявления доступен для использования в качестве представления атрибутов, локальных переменных и формальных параметров. Это придает ему большую осмысленность, чем простое использование типа*** **STRING.**

```
TYPE person_name = STRING;
END TYPE;
```

Правила области определения (условие **WHERE**).

Правила области определения устанавливают ограничения для области определения определенного типа данных. Область определения определенного типа данных является областью определения его базисного представления, ограниченного правилом (или правилами) области определения. Правила области определения указываются после ключевого слова **WHERE**.

Синтаксис:

```
338 where_clause = WHERE domain_rule ';' { domain rule ';' } .
```

Для каждого объекта **domain_rule** может быть задана метка правила.

Примечание — Если метки правил заданы, то они могут использоваться в метках комментария (см. 7.1.6.3) или для идентификации правил в реализациях, например в документации, сообщениях об ошибках и спецификациях применения. Задание меток правил для указанных целей является желательным.

Правила и ограничения:

a) Результатом оценки каждого правила области определения должно быть логическое (**TRUE**, **FALSE** или **UNKNOWN**) или неопределенное (?) значение.

b) Ключевое слово **SELF** (см. 14.5) должно присутствовать в каждом правиле области определения, по крайней мере, один раз. Правило области определения должно оцениваться для конкретного значения из области определения базисного типа путем подстановки данного значения вместо каждого ключевого слова **SELF**, присутствующего в правиле.

c) Правило области определения должно быть применено, если оценкой выражения является **TRUE**; правило должно быть отклонено, если оценкой выражения является **FALSE**; правило не должно быть ни применено, ни отклонено, если оценкой выражения является **UNKNOWN** или неопределенное (?) значение.

d) Область определения определенного типа данных состоит из всех значений области определения базисного типа, не нарушающих ни одного правила области определения.

e) Метки правил области определения должны быть уникальными в рамках данного объявления типа **TYPE**.

Пример — Может быть создан определенный тип данных, ограничивающий базисный целочисленный тип данных только положительными целыми числами.

```
TYPE positive = INTEGER;
WHERE
   notnegative : SELF > 0;
END_TYPE;
```

Любой атрибут, локальная переменная или формальный параметр, объявленный принадлежащим к типу* positive, *при этом будет ограничен только положительными целочисленными значениями.

9.2 Объявление объекта

Объявление объекта **ENTITY** создает объектный тип данных и задает идентификатор для ссылок на него.

Каждый атрибут представляет свойство объекта и может быть ассоциирован со значением в каждом экземпляре объекта. Тип данных атрибута устанавливает область определения его возможных значений.

Каждое ограничение представляет одно из следующих свойств объекта:

a) Ограничения на число, вид и структуру значений атрибутов, задаваемые в объявлениях атрибутов.

b) Необходимые взаимосвязи между значениями атрибута или ограничения на допустимые значения атрибута для данного экземпляра, задаваемые условием **WHERE** и рассматриваемые как правила области определения.

c) Необходимые взаимосвязи между значениями атрибута для всех экземпляров объектного типа данных, задаваемые в следующих конструкциях:

1) в условии уникальности, где они рассматриваются как ограничения на уникальность;

2) в условии инверсии, где они рассматриваются как ограничения на мощность множества;

3) в глобальных правилах (см. 9.6).

d) Необходимые взаимосвязи между экземплярами нескольких объектных типов, задаваемые не в самом объявлении объекта, а в форме глобальных правил (см. 9.6).

Экземпляр объекта в языке EXPRESS может быть создан только посредством конструктора объекта (см. 9.2.6) или оператора построения сложного объекта (см. 12.10).

Синтаксис:

```
206 entity_decl = entity_head entity_body END_ENTITY ';' .
207 entity_head = ENTITY entity_id subsuper ';' .
204 entity_body = { explicit_attr } [ derive_clause ] [ inverse_clause ]  [ unique_clause ] [ where_clause ] .
```

Правила и ограничения:

a) Каждый идентификатор атрибута и метка, указанная в объявлении объекта, должны быть уникальными в рамках объявления.

b) Подтип не должен объявлять атрибут, имеющий такой же идентификатор, что и атрибут одного из супертипов, за исключением случая, когда подтип повторно объявляет атрибут, унаследованный от одного из его супертипов (см. 9.2.3.4).

9.2.1 Атрибуты

Атрибуты типа данных **ENTITY** представляют особенности, качества или свойства присущие объекту. Объявление атрибутов устанавливает взаимосвязь между типом данных **ENTITY** и типом данных, представляемым атрибутом.

Имя атрибута представляет роль, исполняемую ассоциированным с ним значением в контексте объекта, в котором оно используется.

Существуют три вида атрибутов:

- явный атрибут, то есть атрибут, значение которого должно быть определено реализацией при создании экземпляра объекта;

- вычисляемый атрибут, то есть атрибут, значение которого вычисляется некоторым способом;

- инверсный атрибут, то есть атрибут, значение которого состоит из экземпляров объекта, использующих данный объект в определенной роли.

Каждый атрибут устанавливает взаимосвязь между экземпляром объявляющего объектного типа данных и некоторым другим экземпляром или экземплярами. Атрибут, представленный не агрегированным типом данных, устанавливает простую взаимосвязь с этим типом данных. Атрибут, представленный агрегированным типом данных, устанавливает как групповые взаимосвязи со значениями агрегированной структуры, так и дистрибутивные взаимосвязи с элементами данных значений агрегированной структуры. Кроме того, каждый атрибут устанавливает неявную инверсную взаимосвязь между основным типом данных и объявляющим объектным типом данных.

Примечание — Дальнейшее обсуждение данных взаимосвязей приведено в приложении G.

9.2.1.1 Явный атрибут

Явный атрибут представляет свойство, значение которого должно быть обеспечено реализацией при создании экземпляра. Каждый явный атрибут определяет отдельное свойство. Объявление явного атрибута создает один или несколько явных атрибутов, имеющих указанную область определения, и назначает каждому из них идентификатор.

Синтаксис:

```
215 explicit_attr = attribute_decl { ',' attribute_decl } ':' [ OPTIONAL ] parameter_type ';' .
177 attribute_decl = attribute_id | redeclared_attribute .
266 parameter_type = generalized_types | named_types | simple_types .
```

Примечание — Синтаксис объекта **redeclared_attribute** обеспечивает повторное определение атрибута, установленное в 9.2.3.4.

Правила и ограничения:

a) Если явный атрибут не объявлен как **OPTIONAL**, то каждый экземпляр объектного типа данных должен иметь значение для данного атрибута.

Примечание — Если типом данных явного атрибута является наращиваемый перечисляемый тип данных, для которого не заданы элементы перечисления, то такой объект не может быть реализован, если только какое-нибудь расширение перечисляемого типа, содержащее, по крайней мере, один элемент перечисления, не будет объявлено. Если типом данных явного атрибута является наращиваемый выбираемый тип данных, для которого не заданы элементы списка выбора, то такой объект не может быть реализован, если только какое-нибудь расширение выбираемого типа, содержащее, по крайней мере, один именованный тип, не будет объявлено.

b) Ключевое слово **OPTIONAL** указывает на то, что в конкретном экземпляре объекта данный атрибут не обязательно должен иметь значение. Если атрибут не имеет значения, то считается, что он имеет неопределенное (**?**) значение.

c) Явный атрибут не должен объявляться ни явно, ни косвенно как имеющий тип данных **GENERIC**.

Примечания

1 Данное правило справедливо, несмотря на то, что такое объявление допускается синтаксисом.

2 Ключевое слово **OPTIONAL** указывает, что данный атрибут всегда является значимым для экземпляров данного объектного типа, но при этом для некоторых экземпляров он может не иметь значения, соответствующего роли, определяемой данным атрибутом. Ключевое слово **OPTIONAL** не указывает на то, что атрибут является значимым только для некоторых экземпляров объектного типа данных. Ситуация, при которой атрибут не является значимым для некоторых экземпляров, должным образом моделируется путем определения подтипов (см. 9.2.3).

3 Необходимо обратить внимание на ссылки на необязательные атрибуты, особенно в правилах, поскольку такие атрибуты могут не иметь значения. Встроенная функция **EXISTS** может использоваться для определения существования значения атрибута, а встроенная функция **NVL** позволяет обеспечить значение по умолчанию для вычислений. Если ни одна из данных функций не используется, то могут получиться непредсказуемые результаты.

Пример — Следующие объявления эквивалентны:

```
ENTITY point;
   x, y, z : REAL;
END ENTITY;
ENTITY point;
   x : REAL;
   y : REAL;
   z : REAL;
END ENTITY;
```

9.2.1.2 Вычисляемый атрибут

Вычисляемый атрибут представляет свойство, значение которого определяется посредством вычисления значения выражения. Вычисляемые атрибуты объявляются после ключевого слова **DERIVE**. Объявление состоит из идентификатора атрибута, типа его представления и выражения, которое должно использоваться для вычисления значения атрибута.

Синтаксис:
200 derived_attr = attribute_decl ':' parameter_type ':=' expression ';' .
177 attribute_decl = attribute_id | redeclared_attribute .
266 parameter_type = generalized_types | named_types | simple_types .

Примечание — Синтаксис элемента **qualified_attribute** обеспечивает повторное объявление атрибута, установленное в 9.2.3.4.

Выражение может ссылаться на любой атрибут, константу (включая **SELF**) или идентификатор функции, принадлежащий к области видимости.

Правила и ограничения:

a) Элемент **expression** должен быть совместимым с типом данных атрибута, которому присваивается его вычисленное значение (см. 13.3).

b) Для конкретного экземпляра объекта значение вычисляемого атрибута определяется путем вычисления значения выражения с заменой каждого ключевого слова **SELF** данным экземпляром и каждой ссылки на атрибут — значением соответствующего атрибута.

c) Вычисляемый атрибут не должен объявляться ни явно, ни косвенно как имеющий тип данных **GENERIC**.

Примечание — Данное правило справедливо, несмотря на то, что такое объявление допускается синтаксисом.

Пример — В данном примере, круг* (circle) *определяется центром* (centre), *осью* (axis) *и радиусом* (radius). *Помимо этих явных атрибутов, необходимо знать значения таких важных параметров, как площадь* (area) *и периметр* (perimeter). *Данное требование может быть реализовано посредством определения данных параметров как вычисляемых атрибутов, значения которых определены выражениями.

```
ENTITY circle;
   centre : point;
   radius : REAL;
   axis   : vector;
DERIVE
   area      : REAL := PI*radius**2;
   perimeter : REAL := 2.0*PI*radius;
END_ENTITY;
```

9.2.1.3 Инверсный атрибут

Если другой объект установил взаимосвязь с данным объектом посредством явного атрибута, то инверсный атрибут может быть использован для описания этой взаимосвязи в контексте данного объекта. Данный инверсный атрибут может также быть использован для того, чтобы ограничить эту взаимосвязь в дальнейшем.

Инверсные атрибуты объявляются после ключевого слова **INVERSE**. Каждый инверсный атрибут должен быть определен отдельно.

Ограничения мощности множества, применяемые к инверсной взаимосвязи, устанавливаются спецификацией границ для инверсного атрибута так же, как и для явных атрибутов.

Примечание — Более подробная информация о взаимосвязи между явными и инверсными атрибутами приведена в приложении H.

Инверсный атрибут представляется объектным типом данных либо типами данных **BAG** или **SET**, базисным типом которых является объектный тип данных. Обращение к объектному типу данных осуществляется как на ссылочный объект.

В объявлении инверсного атрибута также указывается явный атрибут ссылочного объекта. Для конкретного экземпляра данного объектного типа данных значение инверсного атрибута состоит из экземпляра или экземпляров типов данных ссылочного объекта, использующих данный экземпляр в определенной роли. В случае неоднозначности, возникшей из-за идентичности имен атрибутов в графе подтипов/супертипов ссылочного объекта, имени явного атрибута должно предшествовать имя объекта, который первоначально объявляет данный атрибут.

Каждый из трех возможных типов данных для представления инверсного атрибута устанавливает некоторые ограничения на взаимосвязь между двумя объектами, представленные ниже.

Тип данных **BAG**:

Спецификация границ, если она задана, определяет минимальное и максимальное число экземпляров ссылочного объекта, которое может использовать экземпляр данного объекта. Поскольку неупорядоченное множество (пакет), представляемое типом данных **BAG**, может содержать отдельный экземпляр несколько раз, то один или несколько экземпляров могут ссылаться на данный экземпляр, а конкретный экземпляр может ссылаться на данный экземпляр несколько раз.

Примечания

1 Если инвертированный атрибут представлен неуникальным агрегированным типом данных, то есть списком или массивом, для которого не задано ключевое слово **UNIQUE**, либо пакетом, то конкретный экземпляр данного объекта может быть использован конкретным экземпляром ссылочного объекта несколько раз.

2 Если инвертированный атрибут представлен уникальным агрегированным типом данных, то есть списком или массивом, для которого задано ключевое слово **UNIQUE**, либо набором, то конкретный экземпляр данного объекта может быть использован конкретным экземпляром ссылочного объекта только один раз.

Представление инверсного атрибута с нижней границей равной нулю указывает на то, что на данный экземпляр данного объекта не обязательно должен ссылаться любой экземпляр ссылочного объекта.

Тип данных **SET**:

Справедливо все сказанное выше для типа данных **BAG**, но с дополнительным ограничением, определяющим уникальность ссылочных экземпляров. Данное ограничение также означает, что конкретный ссылочный экземпляр может использовать данный экземпляр в инвертированной роли только один раз.

Примечание — Если инвертированный атрибут представлен уникальным агрегированным типом данных, то есть списком или массивом, для которого задано ключевое слово **UNIQUE**, либо набором, то инверсия не добавляет новых ограничений относительно уникальности.

Объектный тип данных:

Инверсный атрибут содержит точно тот один экземпляр ссылочного объектного типа данных, который использует данный экземпляр в определенной роли. В данном случае мощность множества инверсной взаимосвязи имеет соотношение 1:1.

Синтаксис:

```
248 inverse_attr = attribute_decℓ ':' [ ( SET | BAG ) [ bound_spec ] OF ]
                   entity_ref FOR [ entity_ref '.' ] attribute_ref ';' .
177 attribute_decℓ = attribute_id | rediclared_attribute .
185 bound_spec = '[' bound_1 ':' bound_2 ']'.
183 bound_1 = numeric_expression .
184 bound_2 = numeric_expression .
```

Правила и ограничения:

a) Объект, в котором определяется объявление прямой взаимосвязи с данным объектом, должен реализовать данное объявление в форме явного атрибута.

b) Типом данных явного атрибута в объекте, определяющем прямую взаимосвязь, должен быть один из следующих:

1) тип объявляемого объекта;

2) супертип объявляемого объекта;

3) определенный тип данных, основанный на выбираемом типе данных, содержащем один из перечисленных выше типов;

4) агрегированный тип данных, основным типом которого является один из перечисленных выше.

c) Объект, на который имеется ссылка в объявлении инверсного атрибута, может быть подтипом объекта, объявившего прямую взаимосвязь. В данном случае инверсный атрибут содержит только экземпляры данного подтипа. Подтипы, на которые даются ссылки подобным образом, не должны повторно объявлять явный атрибут как вычисляемый атрибут.

d) Если имя явного атрибута в объекте, определяющем прямую взаимосвязь, не уникально в графе подтипов/супертипов данного объекта, то для уточнения имени данного явного атрибута после ключевого слова **FOR** должно быть использовано имя объектного типа данных.

e) Инверсный атрибут не должен объявляться ни явно, ни косвенно как имеющий тип данных **GENERIC**.

Примечание — Данное правило справедливо, несмотря на то, что такое объявление допускается синтаксисом.

Пример — Предположим, что имеется следующее объявление для определения двери (объект door):

```
ENTITY door;
   handle : knob;
   hinges : SET [1 : ?] OF hinge;
END_ENTITY;
```

Мы можем ограничить объявление шарообразной ручки (объект knob) *так, чтобы шарообразные ручки могли существовать только, если они используются в роли поворотной ручки (объект* handle) *в одном экземпляре двери.*

```
ENTITY knob;
. . .
INVERSE
   opens : door FOR handle;
END ENTITY;
```

С другой стороны, мы можем просто указать, что шарообразная ручка используется в одной двери или не используется (например, она уже установлена на двери или еще только должна быть к ней присоединена).

```
ENTITY knob;
. . .
INVERSE
   opens : SET [0 : 1] OF door FOR handle;
END ENTITY;
```

9.2.2 Локальные правила

Локальные правила являются формальными утверждениями для области определения экземпляров объекта и, таким образом, применимы ко всем экземплярам данного объектного типа данных. Существуют два вида локальных правил. Правила уникальности контролируют уникальность значений атрибутов среди всех экземпляров заданного объектного типа данных. Правила области определения описывают другие ограничения на значения или между значениями атрибутов каждого экземпляра заданного объектного типа данных.

Каждому из локальных правил может быть присвоена метка правила.

Примечание — Метки правил, если они заданы, могут быть использованы в метках комментария (см. 7.1.6.3) или для идентификации правил в реализациях, например, в документации, сообщениях об ошибках и спецификациях применения. Задание меток правил для указанных целей является желательным.

9.2.2.1 Правило уникальности

В правиле уникальности может быть определено ограничение уникальности для отдельных атрибутов или комбинаций атрибутов. Правила уникальности записывают после ключевого слова **UNIQUE**, указывая имя одиночного атрибута или список имен атрибутов. Правило, в котором указано имя одиночного атрибута, называется правилом простой уникальности и устанавливает, что никакие два экземпляра объектного типа данных из области определения не должны использовать один и тот же экземпляр именованного атрибута. Правило, в котором указаны два или более имен атрибутов, называется правилом совместной уникальности и устанавливает, что никакие два экземпляра объектного типа данных не должны иметь одинаковую комбинацию экземпляров именованных атрибутов.

Примечание — При сравнении оценивается равенство экземпляров, а не равенство значений (см. 12.2.2).

Правило: если явный атрибут, который отмечен как **OPTIONAL** (см. 9.2.1.1), появляется в правиле уникальности, и если при этом атрибут не имеет значения для конкретного экземпляра объекта, то правило уникальности не является ни нарушенным, ни доказанным, и поэтому данный экземпляр объекта принадлежит области определения.

Синтаксис:

```
333 unique_clause = UNIQUE unique_rule ';' { unique rule ';' } .
334 unique_rule = [ rule_label_id ':' ] referenced_attribute { ','
        referenced_attribute } .
280 referenced_attribute = attribute_ref | qualified_attribute .
```

Примеры

1 Если объект не *имеет три атрибута* a, b *и* c, *то следующий фрагмент на языке EXPRESS означает, что два экземпляра объявленного объектного типа данных не могут иметь одинаковые значения для* a, b *или* c.

```
ENTITY e;
a, b, c : INTEGER;
UNIQUE
ur1 : a;
ur2 : b;
ur3 : c;
END_ENTITY;
```

2 Объект person_name *может быть определен следующим образом:*

```
ENTITY person name;
   last       : STRING;
   first      : STRING;
   middle     : STRING;
   nickname   : STRING;
END ENTITY;
```

Данный объект может быть использован следующим образом:

```
ENTITY employee;
   badge : NUMBER;
   name  : person_name;
   . . .
UNIQUE
   ur1 : badge, name;
   . . .
END_ENTITY;
```

В данном примере два экземпляра объекта person_name ***могли бы иметь одинаковый набор значений для четырех атрибутов. Однако для объекта*** employee ***введено требование, чтобы совместное использование атрибутов*** badge ***и*** name ***было уникальным. Таким образом, два экземпляра объекта*** employee ***могут иметь одинаковое значение атрибута*** badge ***и одинаковое значение атрибута*** name. ***Однако никакие два экземпляра объекта*** employee ***не могут иметь одинаковый экземпляр атрибута*** badge ***и одинаковый экземпляр атрибута*** name, ***поскольку при совместном использовании данная комбинация экземпляров должна быть уникальной (см. 9.6 по поводу способа описания уникальности значений атрибутов).***

9.2.2.2 Правила области определения (условие **WHERE**)

Правила области определения ограничивают значения отдельных атрибутов или комбинаций атрибутов для каждого экземпляра объекта. Все правила области определения указывают после ключевого слова **WHERE**.

Синтаксис:

338 where_clause = WHERE domain_rule ';' { domain_rule ';' } .

Правила и ограничения:

a) Результатом оценки выражения, описывающего правило области определения, должно быть логическое значение (**TRUE**, **FALSE** или **UNKNOWN**) или неопределенное (**?**) значение.

b) Каждое выражение, описывающее правило области определения, должно содержать ключевое слово **SELF** или атрибуты, объявленные в определении объекта или любого его супертипа.

c) Появление ключевого слова **SELF** должно означать ссылку на экземпляр объявляемого объекта.

d) Правило области определения должно считаться доказанным, если оценкой выражения является значение **TRUE**; правило должно считаться нарушенным, если оценкой выражения является значение **FALSE**; правило будет считаться ни доказанным, ни нарушенным, если оценкой выражения является неопределенное (**?**) значение или значение **UNKNOWN**.

e) Все правила должны выполняться для допустимого экземпляра объекта (в данной области определения).

Пример — Для объекта unit_vector ***требуется, чтобы его длина была равна точно единице. Это ограничение может быть определено следующим образом:***

```
ENTITY unit_vector;
   a, b, c : REAL;
```

```
WHERE
   length_1 : a**2 + b**2 + c**2 = 1.0;
END ENTITY;
```

Необязательные атрибуты в правилах области определения:

Правило области определения, содержащее необязательный атрибут, должно трактоваться в соответствии с описанными ниже правилами и ограничениями.

Правила и ограничения:

a) Если атрибут имеет значение, то правило области определения должно оцениваться как любое другое правило области определения.

b) Если атрибут не имеет значения, то в качестве значения атрибута при оценке выражения, описывающего правило области определения, используется неопределенное (**?**) значение. Оценка выражений, содержащих неопределенное (**?**) значение, рассмотрена в разделе 12.

Пример — Рассмотрим следующий вариант предыдущего примера:

```
ENTITY unit_vector;
   a, b : REAL;
   c    : OPTIONAL REAL;
WHERE
   length_1 : a**2 + b**2 + c**2 = 1.0;
END ENTITY;
```

Целью правила области определения является обеспечение того, чтобы объект **unit_vector** ***сделать единичным. Однако если атрибут*** **c** ***имеет неопределенное (?) значение, то оценкой правила области определения всегда является значение*** **UNKNOWN**, ***независимо от значений аргументов*** **a** ***и*** **b**.

Стандартная функция **NVL** ***(см. 15.18) может быть использована для обеспечения приемлемого значения в случае, если необязательный атрибут имеет неопределенное (?) значение. Если необязательный атрибут имеет значение, то функция*** **NVL** ***возвращает данное значение; в противном случае она возвращает подстановочное значение.***

```
ENTITY unit_vector;
   a, b : REAL;
   c    : OPTIONAL REAL;
WHERE
   length_1 : a**2 + b**2 + NVL(c, 0.0)**2 = 1.0;
END_ENTITY;
```

9.2.3 Подтипы и супертипы

Язык EXPRESS допускает определение объектов как подтипов других объектов, где подтип объекта является конкретизацией его супертипа. Тем самым устанавливается наследственная взаимосвязь (подтип/супертип) между объектами, в которых подтип наследует свойства (то есть атрибуты и ограничения) своего супертипа. Последовательные взаимосвязи подтип/супертип определяют граф наследования, в котором каждый экземпляр подтипа является экземпляром его супертипа (или супертипов).

Граф наследования, определенный взаимосвязями подтип/супертип, должен быть ациклическим.

Если в объявлении объекта полностью определены все значимые свойства данного объекта, то это означает, что объявлен простой объектный тип данных. Если в объявлении объекта установлены наследственные взаимосвязи с супертипами, то это означает, что объявлен сложный объектный тип данных. Сложный объектный тип данных в рамках графа наследования использует характеристики своего супертипа (или супертипов). Сложный объектный тип данных может иметь дополнительные характеристики, не содержащиеся в его супертипе (или супертипах).

Синтаксис:

312 subsuper = [supertype_constraint] [subtype_declaration] .

Следующие положения относятся к взаимосвязям подтип/супертип и основываются на графе подтипов/супертипов. Граф подтипов/супертипов является многокорневым направленным ациклическим графом, в котором узлы представляют объектные типы данных, а ребра представляют взаимосвязи подтип/супертип. Следующие за ключевыми словами **SUBTYPE OF** связи ведут к супертипам, тогда как следующие за ключевыми словами **SUBTYPE OF** связи ведут от супертипов к подтипам.

Правила и ограничения:

a) Ограничение супертипа, при его наличии, должно предшествовать ограничению подтипа, если оно присутствует.

b) Подтип может иметь более одного супертипа.

c) Супертип может иметь более одного подтипа.

d) Супертип может сам быть подтипом одного или нескольких других объектных типов данных. То есть пути в графе подтипов/супертипов могут проходить через несколько узлов.

e) Взаимосвязь подтип/супертип должна быть транзитивной. То есть, если A является подтипом B, а B является подтипом C, то A является подтипом C. Объекты, являющиеся супертипами некоторого объектного типа данных, должны быть такими объектами, к которым можно найти путь на графе, начиная с данного объектного типа данных и следуя по связям **SUBTYPE OF**.

f) Подтип не должен быть супертипом любого из типов, указанных в списке всех его супертипов, то есть граф подтипов/супертипов должен быть ациклическим.

9.2.3.1 Определение подтипов

Объект является подтипом, если он содержит объявление с ключевым словом **SUBTYPE**. В объявлении подтипа должны указываться все непосредственные супертипы объекта. Экземпляр объектного типа данных, определенный как подтип, является экземпляром каждого из его супертипов.

Синтаксис:

318 subtype_declaration = SUBTYPE OF '(' entity_ref { ',' entity_ref } ')' .

9.2.3.2 Определение супертипов

Объект может быть объявлен супертипом посредством явного или неявного определения. Объект объявлен супертипом в явной форме, если он содержит объявление с ключевым словом **ABSTRACT SUPERTYPE**. Объект объявлен супертипом в неявной форме, если его имя задано в объявлении **SUBTYPE**, по крайней мере, одного другого объекта.

Синтаксис:

319 supertype_constraint = abstract_entity_declaration | abstract_supertype_declaration | supertype_rule .
164 abstract_entity_declaration = ABSTRACT .
166 abstract_supertype_declaration = ABSTRACT SUPERTYPE [subtype_constraint] .
313 subtype_constraint = OF '(' supertype_expression ')' .
320 supertype_expression = supertype_factor { ANDOR supertype_factor } .
321 supertype_factor = supertype_term { AND supertype_term } .
323 supertype_term = entity_ref | one of | '(' supertype_expression ')' .
263 one_of = ONEOF '(' supertype_expression { ',' supertype_expression } ')' .
322 supertype_rule = SUPERTYPE subtype_constraint .

Правило: все подтипы, указанные в выражении для супертипа, должны содержать объявление подтипа, идентифицирующее данный объект как супертип.

Пример — Нечетные числа являются подтипом целых чисел, следовательно, целые числа являются супертипом нечетных чисел.

```
ENTITY integer_number;
   val : INTEGER;
END_ENTITY;
ENTITY odd_number
   SUBTYPE OF (integer_number);
WHERE
   not_even : ODD(val);
END_ENTITY;
```

9.2.3.3 Наследование атрибутов

Идентификаторы атрибутов в супертипе определены в рамках области видимости подтипа (см. раздел 10). Таким образом, подтип наследует все атрибуты своего супертипа. Это позволяет подтипам определять ограничения или свои собственные атрибуты с использованием унаследованного атрибута. Если подтип имеет несколько супертипов, то подтип наследует все атрибуты от всех своих супертипов. Это называется множественным наследованием.

Правила и ограничения:

a) Объект не должен объявлять атрибут с таким же именем, как у атрибута, унаследованного от одного из его супертипов, если только он не объявляет унаследованный атрибут повторно (см. 9.2.3.4).

b) Если подтип наследует атрибуты от двух супертипов, не имеющих общих элементов, то допускается, чтобы они имели отличающиеся атрибуты с одинаковыми идентификаторами. Неоднозначность имен

должна разрешаться посредством добавления к идентификатору префикса с именем супертипа, от которого унаследован каждый из атрибутов.

Пример — В данном примере показано, как объект* e12 *наследует два атрибута с одинаковыми именами* attr, *а для того, чтобы указать, для какого из двух атрибутов задается ограничение, к его имени добавляется префикс.

```
ENTITY e1;
   attr : REAL;
   ...
END_ENTITY;

ENTITY e2;
   attr : BINARY;
   ...
END_ENTITY;

ENTITY e12
SUBTYPE OF (e1,e2);
   ...
WHERE
   positive : SELF\e1.attr > 0.0 ; -- атрибут attr, объявленный в e1
END_ENTITY;
```

Подтип может наследовать один и тот же атрибут от разных супертипов, которые в свою очередь унаследовали его от одного супертипа. Это называется повторным наследованием. В данном случае подтип наследует атрибут только один раз, то есть существует только одно значение для данного атрибута в экземпляре данного объектного типа данных.

9.2.3.4 Повторное объявление атрибута

Атрибут, объявленный в супертипе, может быть повторно объявлен в подтипе. Атрибут остается в супертипе, но допустимая область значений для данного атрибута определяется повторным объявлением, заданным в подтипе.

Первоначальное объявление может быть изменено тремя основными способами:

- атрибуту может быть присвоено другое имя;
- тип данных атрибута может быть изменен конкретизацией исходного типа данных (см. 9.2.7).

***Пример — Атрибут типа данных* NUMBER *может быть изменен на тип данных* INTEGER *или* REAL;**

- если исходным типом данных атрибута является определенный тип данных, основанный на выбираемом типе данных, то он может быть изменен другим списком выбора, определяющим подмножество или конкретизацию элементов исходного списка выбора, либо конкретизацией одного из элементов исходного типа выбора;
- необязательный атрибут в супертипе может быть изменен на обязательный атрибут в подтипе;
- явный атрибут в супертипе может быть изменен на вычисляемый атрибут в подтипе;
- атрибуту в супертипе может быть присвоен новый идентификатор в подтипе. Новый идентификатор подчиняется всей области видимости и правилам видимости, определенным в разделе 10, для идентификатора атрибута подтипа, объявление которого содержит данное повторное объявление; но данный идентификатор всегда относится также и к исходному атрибуту в супертипе.

Примечание — Объявление нового идентификатора не удаляет старый идентификатор из области видимости имен. Старый идентификатор остается доступным в своем объектном типе данных и в любых подтипах, объявленных для данного объектного типа данных.

Синтаксис:

```
279 redeclared_attribute = qualified_attribute [ RENAMED attribute_id ] .
275 qualified_attribute = SELF group_qualifier attribute_qualifier .
232 group_qualifier = '\' entity_ref .
179 attribute_qualifier = '.' attribute_ref .
```

Правила и ограничения:

а) Тип данных в повторном объявлении должен совпадать или являться конкретизацией типа данных атрибута, объявленного в супертипе. Применяются правила конкретизации по 9.2.7.

b) Имя повторно объявленного атрибута должно быть задано с использованием синтаксиса элемента **qualified_attribute**.

c) Если тип данных, использованный при определении исходного атрибута, был ограничен правилом **WHERE**, то тип данных, используемый для определения повторно объявленного атрибута, должен быть ограничен так, чтобы область определения повторно объявленного атрибута являлась подмножеством области определения исходного атрибута.

d) Элемент **group_qualifier** в синтаксисе элемента **qualified_attribute** должен идентифицировать объектный тип данных, в котором данный атрибут был первоначально объявлен, либо объектный тип данных, который повторно объявляет атрибут из другого супертипа.

e) Если атрибут супертипа повторно объявлен в двух не взаимоисключающих подтипах, то экземпляр, содержащий оба подтипа, должен иметь единственное значение для атрибута, являющегося допустимым для обоих повторных объявлений.

f) Если атрибуту присвоен новый идентификатор, то данный идентификатор не должен совпадать с идентификатором любого атрибута в любом супертипе данного объектного типа данных.

Примечание — Правило **WHERE**, заданное для исходного атрибута, остается в силе и для повторных объявлений данного атрибута (см. 9.2.3.5).

Примеры

1 В некоторых геометрических системах используются координаты с плавающей точкой, в то время как другие системы работают в целочисленном координатном пространстве. Понятия GENERIC_ENTITY и RENAMED обеспечивают спецификации общей применимости и конкретизации для специфической области применения.

```
ENTITY point;
   x : NUMBER;
   y : NUMBER;
END_ENTITY;
ENTITY integer_point
   SUBTYPE OF (point);
   SELF\point.x RENAMED integer_x  :  INTEGER;
   SELF\point.y RENAMED integer_y  :  INTEGER;
END_ENTITY;
ENTITY line ABSTRACT;
   start : GENERIC_ENTITY;
   end  : GENERIC_ENTITY;
END_ENTITY;
ENTITY integer_point_line;
   SUBTYPE OF (line);
   SELF\line.start RENAMED integer_start  :  INTEGER_point;
   SELF\line.end  RENAMED integer_end    :  INTEGER_point;
END_ENTITY;
```

2 Данный пример демонстрирует изменение элементов агрегированного типа данных на уникальные, уменьшение числа элементов агрегированного типа данных и изменение необязательного атрибута на обязательный.

```
ENTITY super;
   things    :  LIST [3 : ?] OF thing;
   items     :  BAG [0 : ?] OF widget;
   may_be  :  OPTIONAL stuff;
END_ENTITY;
ENTITY sub
   SUBTYPE OF (super);
   SELF\super.things    : LIST [3 : ?] OF UNIQUE thing;
   SELF\super.items     : SET [1 : 10] OF widget;
   SELF\super.may_be  : stuff;
END_ENTITY;
```

3 В данном примере круг задан центром, осью и радиусом. Вариант круга задан центром и двумя точками, через которые он проходит. Эти три точки представляют данные, посредством которых задан данный вариант круга. В дополнение к этим данным необходимо учесть и другие важные параметры — радиус и ось. Это осуществляется посредством их повторного объявления как вычисляемых атрибутов с заданием их значений выражениями.

```
FUNCTION distance(p1, p2 : point) : REAL;
   (* Вычисляет кратчайшее расстояние между двумя точками *)
END_FUNCTION;
FUNCTION normal (p1, p2, p3 : point) : vector;
   (*Вычисляет нормаль к плоскости, заданной тремя точками на ней *)
END_FUNCTION;
ENTITY circle;
   centre : point;
   radius : REAL;
   axis   : vector;
DERIVE
   area   : REAL := PI*radius**2;
END_ENTITY;
ENTITY circle_by_points
   SUBTYPE OF (circle);
   p2 : point;
   p3 : point;
DERIVE
   SELF\circle.radius : REAL   := distance (centre, p2);
   SELF\circle.axis   : vector := normal (centre, p2, p3);
WHERE
   not coincident : (centre <> p2) AND
                    (p2 <> p3) AND
                    (p3 <> centre);
 is_circle        : distance (centre,p3) =
                    distance (centre,p2);
END_ENTITY;
```

***В подтипе три определяющие точки* (centre, p2 *и* p3) *являются явными атрибутами, а объекты* radius, axis *и* area *являются вычисляемыми атрибутами. Значения вычисляемых атрибутов определяются выражением, следующим за оператором присваивания. Значения объектов* radius *и* axis *получаются посредством вызова функции; заодно вычисляется значение объекта* area.**

9.2.3.5 Наследование правил

Каждое локальное или глобальное правило, относящееся ко всем экземплярам супертипа, применимо и ко всем экземплярам его подтипов. Таким образом, подтип наследует все правила своего супертипа. Если подтип имеет несколько супертипов, то подтип должен наследовать все правила, ограничивающие супертипы.

Нельзя изменить или удалить какое-либо из правил, связанных с подтипом, через наследование правил, но можно добавить новые правила, еще более ограничивающие подтип.

Правило: на экземпляр объекта распространяются все ограничения, установленные для каждого из его объектных типов данных.

Примечание — Если ограничения, установленные в двух (или более) объектных типах данных, противоречат друг другу, то не может существовать допустимого экземпляра, содержащего эти объектные типы данных.

Пример — В данном примере выпускником (объект* graduate*) является лицо (объект* person*), которое и учит, и учится. Объект* graduate *наследует атрибуты и ограничения от своих супертипов (объектов* teacher *(преподаватель) и* student *(студент)) вместе с атрибутами и ограничениями из их общего супертипа (объекта* person*). Но выпускнику, в отличие от преподавателя, не разрешено преподавать на старших курсах.

```
SCHEMA s;
ENTITY person;
   ss_no : INTEGER;
   born  : date;
   ...
DERIVE
   age : INTEGER := years_since (born);
UNIQUE
   un1 : ss_no;
END_ENTITY;
```

```
ENTITY teacher
   SUBTYPE OF (person);
   teaches : SET [1 : ?] OF course;
   ...
WHERE
   old : age >= 21;
END_ENTITY;
ENTITY student
   SUBTYPE OF (person);
   takes : SET [1 : ?] OF course;
   ...
WHERE
   young : age >= 5;
END_ENTITY;
ENTITY graduate
   SUBTYPE OF (student, teacher);
   ...
WHERE
   limited : NOT (GRAD_LEVEL IN teaches);
END_ENTITY;
TYPE course = ENUMERATION OF (..., GRAD_LEVEL, ...);
END TYPE;
   ...
END_SCHEMA; -- конец схемы S
```

Примечание — Если подтип наследует взаимно противоречащие ограничения от своих супертипов, то не может существовать соответствующий экземпляр данного подтипа, поскольку любой экземпляр будет нарушать одно из ограничений.

9.2.4 Абстрактный объектный тип данных

Язык EXPRESS позволяет объявлять объектные типы данных, не предназначенные для непосредственной реализации, а только для реализации через свои подтипы. Абстрактный объектный тип данных может объявить явные или вычисляемые атрибуты, типами данных которых являются обобщенные типы данных (см. 8.5). Эти обобщенные типы данных могут затем быть повторно объявлены как реализуемые типы данных в подтипах абстрактного объектного типа данных. Если подтип абстрактного объектного типа данных сам является абстрактным объектным типом данных, то он не должен повторно объявлять нереализуемые унаследованные атрибуты как реализуемые типы данных. В подтипе абстрактного объектного типа данных, который не является абстрактным объектным типом данных, никакие унаследованные или явно объявленные атрибуты не должны иметь нереализуемый тип данных.

Метки типа (см. 9.5.3.4) могут быть использованы для обеспечения того, что два или более атрибута, типами данных которых являются обобщенные типы данных, имеют одинаковые типы данных на момент обращения.

Правила и ограничения:

a) Объявление абстрактного объектного типа данных содержит ключевое слово **ABSTRACT** в объявлении объектного типа данных **ENTITY**, но не содержит ключевое слово **SUPERTYPE** (понятие **ABSTRACT SUPERTYPE** определено в 9.5.2.1).

b) Абстрактный объектный тип данных не является реализуемым, если только он не является частью сложного объектного типа данных, все атрибуты которого, имеющие обобщенные типы данных, были объявлены повторно как имеющие реализуемый тип данных.

Примечания

1 Правило по перечислению b) обеспечивает соответствие любого абстрактного объектного типа данных ограничению абстрактного супертипа (см. 9.2.5.1).

2 Повторное объявление может быть осуществлено непосредственно в реализуемом подтипе или в одном из его супертипов (см. 9.2.4).

Пример — В обобщенной модели согласования может потребоваться определить, что может быть согласована группа объектов. Данная модель затем может быть использована в ряде других схем и уточнена в части определения согласования реальных объектов.

```
ENTITY general_approval ABSTRACT;
   approved_items : BAG OF GENERIC_ENTITY;
   status         : approval_status;
END_ENTITY;
```

9.2.5 Ограничения подтипов/супертипов

Экземпляр объектного типа данных, объявленный явно или неявно как супертип (см. 9.2.3.2), может также являться экземпляром одного или нескольких из его подтипов (см. Н.2).

Синтаксис:

```
319 supertype_constraint = abstract_entity_declaration |
                           abstract_supertype_declaration | supertype_rule .
164 abstract_entity_declaration = ABSTRACT .
166 abstract_supertype_declaration = ABSTRACT SUPERTYPE [ subtype_constraint ] .
313 subtype_constraint = OF '(' supertype_expression ')' .
320 supertype_expression = supertype_factor { ANDOR supertype_factor } .
321 supertype_factor = supertype_term { AND supertype_term } .
323 supertype_term = entity_ref | one_of | '(' supertype_expression ')' .
263 one_of = ONEOF '(' supertype_expression {',' supertype_expression } ')' .
322 supertype_rule = SUPERTYPE subtype_constraint .
```

Имеется возможность определить ограничения, в соответствии с которыми могут быть реализованы графы подтипов/супертипов. Данные ограничения могут быть определены в объявлении супертипа посредством условия **SUPERTYPE**. Они могут также быть определены как отдельные правила посредством объявлений **SUBTYPE_CONSTRAINT** (см. 9.7).

Примечание — Для того, чтобы существующие схемы, разработанные в соответствии с первой редакцией справочного руководства по языку EXPRESS, оставались допустимыми, объявление ограничений на подтипы/супертипы, в котором при объявлении объекта используются ключевые слова **ONEOF**, **ANDOR** или **AND**, остается допустимым по отношению к настоящему стандарту. Однако, данный способ объявления нежелателен, поскольку в последующих редакциях стандарта планируется его запрет. Рекомендуется использовать объявление **SUBTYPE_CONSTRAINT**.

Объявление **SUBTYPE_CONSTRAINT** содержит совокупность ограничений, установленных в выражении **supertype_expression**. Объявление **SUBTYPE_CONSTRAINT** может содержать любое число ограничений **AND** и **ONEOF**, каждое из которых интерпретируется как отдельное ограничение.

Кроме того, при включении в формулировку какого-либо более сложного ограничения каждое выражение **ONEOF**, **AND** и **ANDOR** интерпретируется как совокупность экземпляров супертипа. При интерпретации выражения **supertype_expression** применяют следующие правила:

- имя объектного типа данных, встретившееся где-либо в выражении **supertype_expression**, интерпретируется как множество экземпляров объекта, образующих полную совокупность данного типа данных, как и в глобальном правиле (см. 9.6);

- результат вычисления выражения supertype_expression интерпретируется как множество экземпляров супертипа в соответствии с определенными ниже ограничениями **ONEOF**, **AND** и **ANDOR**.

Несмотря на то, что конечным результатом вычисления выражения **supertype_expression** для объявления **SUBTYPE_CONSTRAINT** является множество экземпляров объекта, данное множество не имеет значения. То есть результат всего выражения **supertype_expression** не устанавливает никакого ограничения, поскольку он не обязательно содержит все экземпляры супертипа и может содержать экземпляры, к которым не применяется ни одно из установленных ограничений.

Примечание — Следовательно, независимые ограничения могут быть связаны оператором **ANDOR**, который только добавляет экземпляры к общему (не имеющему значения) результату выражения **supertype_expression**.

Формальный подход к определению возможных комбинаций подтипов/супертипов, которые могут быть реализованы с учетом некоторых возможных ограничений, определенных ниже, представлен в приложении В.

9.2.5.1 Абстрактные супертипы

Язык EXPRESS допускает объявление супертипов, не предназначенных для непосредственной реализации. Для этого объектный тип данных должен содержать ключевые слова **ABSTRACT SUPERTYPE** в ограничении супертипа. Абстрактный супертип не должен быть реализован, кроме как в сочетании, по крайней мере, с одним из его подтипов.

Примечание — Это означает, что схема, содержащая определение абстрактного супертипа без каких-либо подтипов, является неполной и не может быть реализована, если только подтипы не объявлены в ссылочной схеме.

Пример — В модели перевозок транспортное средство (объект* vehicle*) может быть представлено абстрактным супертипом, поскольку все экземпляры данного объектного типа данных предназначены быть его подтипами (например, наземное или водное транспортное средство). Объектный тип данных для транспортного средства не должен реализовываться независимо.

```
ENTITY vehicle
  ABSTRACT SUPERTYPE;
END ENTITY;
ENTITY land_based
  SUBTYPE OF (vehicle);
  ...
END ENTITY;
ENTITY water_based
  SUBTYPE OF (vehicle);
  ...
END ENTITY;
```

9.2.5.2 **ONEOF**

Ограничение **ONEOF** устанавливает, что совокупности операндов из списка **ONEOF** являются взаимоисключающими. Никакой экземпляр любой совокупности операндов из списка **ONEOF** не должен присутствовать в совокупности любого другого операнда из списка **ONEOF**.

Синтаксис:

```
263 one_of = ONEOF '(' supertype_expression {',' supertype_expression } ')'.
320 supertype_expression = supertype_factor { ANDOR supertype_factor }.
321 supertype_factor = supertype_term { AND supertype_term }.
323 supertype_term = entity_ref | one_of | '(' supertype_expression ')'.
```

Ограничение **ONEOF** может комбинироваться с другими ограничениями супертипов, чтобы дать возможность записывать сложные ограничения. Когда ограничение **ONEOF** присутствует как операнд в другом ограничении, оно представляет множество экземпляров объектов, являющееся объединением совокупностей операндов из списка **ONEOF**.

Примечание — На естественном языке фраза **ONEOF** (**a**, **b**, **c**) означает, что «экземпляр объекта должен состоять только из одного из объектных типов данных **a**, **b**, **c**».

Пример — Экземпляр супертипа может быть установлен посредством реализации только одного из его подтипов. Данное ограничение объявляется с использованием ключевого слова* ONEOF*. Существует множество видов домашних животных (объект* pet*), но ни одно домашнее животное не может одновременно принадлежать к двум или более видам.

```
ENTITY pet
  ABSTRACT SUPERTYPE OF (ONEOF (cat,
                                rabbit,
                                dog,
                                ...));
  name : pet_name;
  ...
END ENTITY;
ENTITY cat
  SUBTYPE OF (pet);
  ...
END ENTITY;
ENTITY rabbit
  SUBTYPE OF (pet);
  ...
END ENTITY;
ENTITY dog
  SUBTYPE OF (pet);
  ...
END ENTITY;
```

9.2.5.3 **ANDOR**

Экземпляр совокупности, определяемой выражением с ключевым словом **ANDOR**, может быть экземпляром совокупности любого из операндов или обоих. Таким образом, **ANDOR** не определяет огра-

ничение. В сложном ограничении **ANDOR** представляет множество экземпляров объектов, являющееся объединением совокупностей выражений, определяющих операнды.

Примечание — **ANDOR** используется только при построении совокупностей для более сложного ограничения. На естественном языке фраза «**b ANDOR c**» означает «все экземпляры типа **b** и все экземпляры типа **c**, включая те, которые являются экземплярами обоих типов».

Пример — Человек (объект* person*) может быть служащим, посещающим вечерние занятия, и поэтому являться одновременно и служащим (объект* employee*), и студентом (объект* student*).

```
ENTITY person
   SUPERTYPE OF (employee ANDOR student);
   ...
END_ENTITY;
ENTITY employee
   SUBTYPE OF (person);
   ...
END ENTITY;
ENTITY student
   SUBTYPE OF (person);
   ...
END ENTITY;
```

9.2.5.4 **AND**

Ключевое слово **AND** определяет ограничение, что совокупности, заданные двумя операндами, должны быть идентичными. То есть любой экземпляр совокупности левого операнда должен также быть экземпляром совокупности правого операнда, а любой экземпляр совокупности правого операнда должен также быть экземпляром совокупности левого операнда.

Когда выражение с ключевым словом **AND** присутствует в качестве операнда в сложном ограничении, оно представляет любую из совокупностей своих операндов, поскольку они идентичны.

Примечание — На естественном языке фраза «**b AND c**» означает, что «экземпляр должен одновременно состоять из типов **b** и **c**».

Пример – Человек (объект* person*) может быть классифицирован как лицо мужского (объект* male*) или женского (объект* female*) пола, либо как гражданин (объект* citizen*) или иностранец (объект* alien*).

```
ENTITY person
   SUPERTYPE OF (ONEOF (male, female) AND
                 ONEOF (citizen, alien));
   ...
END_ENTITY;
ENTITY male
   SUBTYPE OF (person);
   ...
END_ENTITY;
ENTITY female
   SUBTYPE OF (person);
   ...
END_ENTITY;
ENTITY citizen
   SUBTYPE OF (person);
   ...
END_ENTITY;
ENTITY alien
   SUBTYPE OF (person);
   ...
END_ENTITY;
```

9.2.5.5 Приоритеты операторов супертипов

Оценка выражений супертипов проводится слева направо, при этом сначала определяются значения операторов с наивысшим приоритетом. Правила приоритетов для операторов выражений супертипов

представлены в таблице 8. Операторы, расположенные в одной строке, имеют равный приоритет, а строки упорядочены по уменьшению приоритета.

Таблица 8 — Приоритет оператора выражения супертипа

Приоритет	Операторы
1	**( ) ONEOF**
2	**AND**
3	**ANDOR**

Пример — Следующие два выражения не эквивалентны:

```
ENTITY x
   SUPERTYPE OF (a ANDOR b AND c);
END_ENTITY;
ENTITY x
   SUPERTYPE OF ((a ANDOR b) AND c);
END_ENTITY;
```

9.2.5.6 Ограничения между подтипами по умолчанию

Если в объявлении объекта не указано никаких ограничений на супертипы, то подтипы (при их наличии) должны быть взаимно инклюзивными, то есть, как если бы все подтипы были бы неявно представлены в конструкции **ANDOR**.

Если ограничение на супертипы задано для подмножества подтипов данного объекта, то оно должно включать в себя ограничения, установленные для этих подтипов, и ограничение **ANDOR** для других подтипов.

Пример — Модель из примера в 9.2.5.3 эквивалентна использованию следующей конструкции по умолчанию:

```
ENTITY person
   ...
END ENTITY;
ENTITY employee
   SUBTYPE OF (person);
   ...
END ENTITY;
ENTITY student
   SUBTYPE OF (person);
   ...
END ENTITY;
```

9.2.6 Неявные объявления

При объявлении объекта, одновременно неявным образом объявляется конструктор. Идентификатор конструктора совпадает с идентификатором объекта и область видимости объявления конструктора совпадает с областью видимости объявления объекта.

При запуске конструктора он должен возвращать в точку вызова значение частичного сложного объекта для данного объектного типа данных. Каждый атрибут данного значения частичного сложного объекта задается фактическим параметром, заданным в вызове конструктора мнимой копией. Мнимая копия представляет собой объект, в который экземпляр объекта копируется посредством ссылки, то есть атрибут является ссылкой на экземпляр, используемый в качестве фактического параметра, простой тип данных имеет скопированное значение, а агрегированные структуры имеют элементы, скопированные в мнимую копию. Конструктор должен обеспечивать только явные атрибуты из объявления конкретного объекта.

Синтаксис:

```
205 entity_constructor = entity_ref '(' [ expression { ',' expression } ] ')' .
```

При создании экземпляра сложного объекта (экземпляра объекта, присутствующего в графе подтипов/супертипов) конструкторы всех составляющих его объектов должны объединяться посредством оператора **||** (см. 12.10).

Правила и ограничения:

a) Конструктор должен иметь один формальный параметр для каждого явного атрибута, объявленного в данном объектном типе данных. Это не относится к атрибутам, унаследованным от супертипов и повторно объявленным в данном объектном типе данных.

b) Порядок формальных параметров должен быть идентичен порядку объявления явных атрибутов в объекте.

c) Параметрический тип данных каждого из формальных параметров должен быть идентичен типу данных соответствующего атрибута.

d) Если объект не имеет явных атрибутов, то должен быть указан пустой список параметров (то есть круглые скобки всегда должны присутствовать).

Примечание — Обязательное присутствие круглых скобок отличается от явно объявленных функций.

e) Необязательные атрибуты могут быть заданы неопределенным (**?**) значением при вызове неявно определенного конструктора. Это указывает на то, что явное значение не было задано.

f) Если в экземпляре сложного объекта имеется подтип, содержащий вычисляемые атрибуты, являющиеся повторно объявленными явными атрибутами супертипа, то конструктор супертипа должен задавать значения для данных повторно объявленных атрибутов. Вместо этих значений используется вычисленное значение.

Пример — Допустим, имеется следующее объявление объекта:

```
ENTITY point;
   x, y, z : REAL;
END_ENTITY;
```

Тогда неявно объявленный конструктор объекта* point *может быть представлен следующим образом:

```
FUNCTION point (x,y,z : REAL) : point;
```

Данный конструктор затем может использоваться при присваивании значений экземпляру данного объектного типа данных:

```
CONSTANT
   origin : point := point(0.0, 0.0, 0.0);
END_CONSTANT;
```

9.2.7 Конкретизация

Конкретизация является более ограниченной формой исходного объявления. Существуют следующие варианты определения конкретизации:

- объект, относящийся к подтипу данных, является конкретизацией любого из своих супертипов;
- тип данных **ENTITY** является конкретизацией типа данных **GENERIC_ENTITY**;
- тип данных **EXTENSIBLE GENERIC_ENTITY SELECT** является конкретизацией типа данных **GENERIC_ENTITY**;
- тип данных **SELECT**, содержащий только типы данных **ENTITY**, является конкретизацией типа данных **GENERIC_ENTITY**;
- агрегированные типы данных являются конкретизациями типа данных **AGGREGATE**;
- тип данных **SELECT**, состоящий из объектов **a**, **b**, **c**, является конкретизацией типа данных **SELECT**, состоящего из объектов **d**, **e**, **f**, если объекты **a**, **b**, **c** являются конкретизациями объектов **d**, **e**, **f**;
- тип данных **SELECT**, состоящий из объектов **a**, **b**, **c**, является конкретизацией супертипа, если **a**, **b**, **c** являются подтипами данного супертипа;
- типы данных **INTEGER** и **REAL** оба являются конкретизациями типа данных **NUMBER**;
- тип данных **INTEGER** является конкретизацией типа данных **REAL**;
- тип данных **BOOLEAN** является конкретизацией типа данных **LOGICAL**;
- выражение **LIST OF UNIQUE item** является конкретизацией выражения **LIST OF item**;
- выражение **ARRAY OF UNIQUE item** является конкретизацией выражения **ARRAY OF item**;
- выражение **ARRAY OF item** является конкретизацией выражения **ARRAY OF OPTIONAL item**;
- выражение **SET OF item** является конкретизацией выражения **BAG OF item**;
- пусть **AGG** обозначает один из типов данных **ARRAY**, **BAG**, **LIST** или **SET**, тогда выражение **AGG OF item** является конкретизацией выражения **AGG OF original** при условии, что **item** является конкретизацией **original**;

- пусть **AGG** обозначает один из типов данных **BAG**, **LIST** или **SET**, тогда выражение **AGG [b : t]** является конкретизацией выражения **AGG [l : u]** при условии, что **b ≤ t** и **l ≤ b ≤ u** и **l ≤ t ≤ u**;

- пусть **BSR** обозначает один из типов данных **BINARY**, **STRING** или **REAL**, тогда **BSR(length)** является конкретизацией **BSR**;

- **BSR(short)** является конкретизацией **BSR(long)** при условии, что **short** меньше чем **long**;

- тип данных **BINARY**, в котором используется ключевое слово **FIXED**, является конкретизацией типа данных **BINARY** переменной длины;

- тип данных **STRING**, в котором используется ключевое слово **FIXED**, является конкретизацией типа данных **STRING** переменной длины;

- конструкционный тип данных, основанный на расширяемом конструкционном типе данных, является конкретизацией данного расширяемого конструкционного типа данных;

- определенный тип данных является конкретизацией базисного типа данных, использованного для объявления данного определенного типа данных.

9.3 Схема

Объявление схемы (**SCHEMA**) определяет общую область видимости для совокупности относящихся к нему объявлений объектных и других типов данных. Схема может подвергаться изменениям в рамках среды разработки или стандартизации. Для поддержки возможности идентификации конкретной версии схемы определяется идентификатор версии схемы. В настоящем стандарте не определяется формат идентификатора версии схемы, за исключением того, что он должен быть строковым литералом. Ни в одной конструкции, определенной в настоящем стандарте, нет ссылок на идентификатор версии схемы. Кроме того, в настоящем стандарте не определен способ управления изменениями в схемах с использованием идентификаторов версии схемы. Если две схемы с одинаковым именем имеют разные идентификаторы версии схемы, то они не должны рассматриваться как одна и та же схема.

Примечание — Для схем, установленных в стандартах комплекса ИСО 10303, определено использование идентификатора информационного объекта, включающего в себя идентификатор версии. Смысл идентификатора объекта установлен в ИСО/МЭК 8824-1 и описан в ИСО 10303-1. Данный идентификатор объекта рекомендуется использовать в качестве идентификатора версии схемы.

Примеры

1 Идентификатор **geometry** ***может быть именем схемы, содержащей объявления точек, кривых, поверхностей и других, связанных с ними, типов данных.***

2 Может существовать множество версий схем, идентификаторы которых могут включать также идентификатор версии языка. В данном примере в схеме **support_resource_schema** ***используется идентификатор информационного объекта, соответствующий определению ИСО/МЭК 8842-1 и описанию в ИСО 10303-1.***

```
SCHEMA geometry_schema 'version_1';
END_SCHEMA;
SCHEMA geometry_schema 'version_2';
END_SCHEMA;
SCHEMA support_resource_schema '{ISO standard 10303 part(41) object(1)
        version(8)}';
END_SCHEMA;
SCHEMA support_resource_schema '{ISO standard 10303 part(41) object(1)
        version(9)}';
END_SCHEMA;
```

Порядок, в котором объявления появляются в объявлении схемы, значения не имеет.

Объявления, сделанные в одной схеме, могут быть сделаны видимыми в области видимости другой схемы посредством спецификации интерфейса в соответствии с разделом 11.

Синтаксис:

```
296 schema_decl = SCHEMA schema_id [ schema_version_id ] ';' schema _body
                  END_SCHEMA ';'.
298 schema_version_id = string_literal .
295 schema_body = { interface_specification } [ constant_decl ]
                  { declaration | rule_decl } .
242 interface_specification = reference_clause | use_clause .
199 declaration = entity_decl | function_decl | procedure_decl |
                  subtype_constraint_decl | type_decl .
```

9.4 Константа

Объявление константы используется для объявления именованных констант. Областью видимости идентификатора константы должна быть функция, процедура, правило или схема, в которых происходит объявление константы. Именованная константа, появляющаяся в объявлении **CONSTANT**, должна иметь явную инициализацию, значение которой вычисляется посредством оценивания выражения. Именованная константа может присутствовать в объявлении другой именованной константы. Объявление констант должно быть ациклическим.

Примечание — Требование ацикличности объявлений констант необходимо для обеспечения достоверности инициализации в любом случае, поскольку инициализация не обязательно реализуется в порядке объявления.

Синтаксис:

```
195 constant_decl = CONSTANT constant_body { constant_body} END_CONSTANT ';' .
194 constant_body = constant_id ':' instantiable_type ':=' expression ';' .
240 instantiable_type = concrete_types | entity_ref .
```

Правила и ограничения:

a) Значение константы не должно изменяться после инициализации.

b) Любое появление именованной константы вне ее объявления должно быть эквивалентно появлению ее инициализированного значения.

c) Выражение **expression** в синтаксическом правиле 194 должно возвращать значение соответствующее указанному базовому типу.

Пример — Ниже приведены допустимые объявления констант:

```
CONSTANT
   thousand : NUMBER := 1000;
   million  : NUMBER := thousand**2;
   origin   : point  := point(0.0, 0.0, 0.0);
END_CONSTANT;
```

9.5 Алгоритмы

Алгоритм является последовательностью операторов, выполнение которых приводит к некоторому желаемому конечному состоянию. Существуют два вида алгоритмов — функции и процедуры.

Формальные параметры определяют исходные данные для алгоритма. Обращение к алгоритму осуществляется с фактическими параметрами, обеспечивающими фактические значения или экземпляры. Фактические параметры должны соответствовать формальным параметрам по типу данных, порядку следования и числу.

Если в алгоритме необходимо сделать локальные объявления, то они приводятся непосредственно после заголовка алгоритма. Локальными объявлениями могут быть объявления типов данных, локальных переменных, других алгоритмов и т.д.

После локальных объявлений следует тело алгоритма.

9.5.1 Функция

Функция является алгоритмом, который обрабатывает параметры и выдает единственное результирующее значение установленного типа данных. При обращении к функции (см. 12.8) в выражении вычисляется результирующее значение в точке вызова.

Функция должна завершаться выполнением оператора **RETURN**. Значение выражения, связанного с оператором **RETURN**, определяет результат, полученный при вызове функции.

Синтаксис:

```
220 function_decl = function_head algorithm_head stmt { stmt } END_FUNCTION ';' .
221 function_head = FUNCTION function_id [ '(' formal_parameter
                        { ';' formal_parameter } ')' ] ':' parameter_type ';' .
218 formal_parameter = parameter_id { ',' parameter_id } ':' parameter_type .
266 parameter_type = generalized_types | named_types | simple_types .
173 algorithm_head = { declaration } [ constant_decl ] [ local_decl ] .
199 declaration = entity_decl | function_decl | procedure_decl |
                    subtype_constraint_decl | type_decl .
```

Правила и ограничения:

a) Оператор **RETURN** должен быть определен в теле функции для каждой из возможных ветвей вычислительного процесса, активизируемого при обращении к данной функции.

b) Для каждого оператора **RETURN**, присутствующего в функции, должно быть задано выражение, по которому вычисляется значение, возвращаемое в точку вызова.

c) Выражения, заданные для операторов **RETURN**, должны соответствовать объявленному типу данных возвращаемого функцией значения.

d) Функции не должны иметь побочных эффектов. Поскольку формальные параметры функции не должны определяться ключевым словом **VAR**, изменения этих параметров внутри функции не показываются в точке вызова функции.

Примечание — Локальным переменным, объявленным объектными типами данных, могут быть назначены экземпляры формальных параметров. Изменение этих локальных переменных будет влиять на формальный параметр, поскольку назначение осуществляется по ссылке. В соответствии с приведенным выше правилом, изменения формальных параметров не показываются в точке вызова функции, поэтому особое внимание необходимо обратить на попытку возврата этих локальных переменных.

e) Функции могут модифицировать локальные переменные или параметры, объявленные во внешней области видимости, то есть если данная функция объявлена в заголовке (элемент **algorithm_head** в определении синтаксиса) оператора **FUNCTION**, **PROCEDURE** или **RULE**.

9.5.2 Процедура

Процедура является алгоритмом, который получает параметры в точке вызова и обрабатывает их некоторым образом для получения желаемого конечного состояния. Изменения параметров внутри процедуры показываются в точке вызова только в том случае, если формальному параметру предшествует ключевое слово **VAR**.

Синтаксис:

```
271 procedure_decl = procedure_head algorithm_head { stmt } END_PROCEDURE ';' .
272 procedure head = PROCEDURE procedure_id  [ '(' [ VAR ] formal_parameter
                        { ';' [ VAR ] formal_parameter } ')' ] ';' .
218 formal_parameter = parameter_id { ',' parameter_id } ':' parameter_type .
266 parameter_type = generalized_types | named_types | simple_types .
173 algorithm_head =  { declaration }  [ constant_decl ]  [ local_decl ] .
199 declaration = entity_decl | function_decl | procedure_decl |
                  subtype_constraint_decl | type_decl .
```

Правило: процедуры могут модифицировать локальные переменные или параметры, объявленные во внешней области видимости, то есть если данная функция объявлена в заголовке (элемент **algorithm_head** в определении синтаксиса) оператора **FUNCTION**, **PROCEDURE** или **RULE**.

9.5.3 Параметры

Функция или процедура может иметь формальные параметры. Каждый формальный параметр устанавливает имя и тип параметра. Имя является идентификатором, который должен быть уникальным в области видимости функции или процедуры. Формальный параметр процедуры может, кроме того, быть объявлен с ключевым словом **VAR** (изменяемый), которое означает, что если данный параметр изменяется внутри процедуры, то данное изменение должно быть передано в точку вызова процедуры. Параметры, не объявленные как **VAR**, также могут изменяться, но такое изменение не будет видимым после возврата управления вызвавшей структуре.

Синтаксис:

```
218 formal_parameter = parameter_id  { ',' parameter_id } ':' parameter_type .
266 parameter_type = generalized_types | named_types | simple_types .
```

Пример — Следующие объявления показывают, как могут быть объявлены формальные параметры.

```
FUNCTION dist(p1, p2 : point) : REAL;
...
PROCEDURE midpt(p1, p2 : point; VAR result : point);
```

Обобщенные типы данных (**AGGREGATE**, общий агрегированный тип данных, **GENERIC** и **GENERIC_ENTITY**) (см. 8.5) используются для того, чтобы обеспечить обобщение типов данных, использу-

емых для представления формальных параметров функций и процедур. Тип данных **AGGREGATE**, общий агрегированный тип данных (см. 9.5.3.5) и родовой объектный тип данных (**GENERIC_ENTITY**) могут, кроме того, использоваться для объявления явных или вычисляемых атрибутов абстрактных объектных типов данных. Общий агрегированный тип данных может также использоваться, чтобы обеспечить обобщение базисных типов данных, разрешенное для конкретных агрегированных типов данных.

9.5.3.1 Тип данных **AGGREGATE**

Тип данных **AGGREGATE** является обобщением всех агрегированных типов данных.

При вызове процедуры или функции, формальный параметр которой определен типом данных **AGGREGATE**, передаваемый фактический параметр должен иметь тип данных **ARRAY**, **BAG**, **LIST** или **SET**. При этом операции, которые могут быть выполнены, должны зависеть от типа данных фактического параметра.

Примечание — Метки типов (см. 9.5.3.4) могут использоваться для обеспечения того, чтобы при вызове два или более параметров, представленных типом данных **AGGREGATE**, имели одинаковый тип данных, или чтобы тип данных возвращаемого результата был таким же, как и у одного из переданных параметров, независимо от переданных фактических типов данных.

Если явный или вычисляемый атрибут в объявлении типа данных **ABSTRACT ENTITY** представлен типом данных **AGGREGATE**, то данный атрибут должен быть объявлен в подтипах данного абстрактного объекта как **ARRAY**, **BAG**, **LIST** или **SET**.

Синтаксис:

```
171 aggregate_type = AGGREGATE [ ':' type_label ] OF parameter_type .
329 type_label = type_label_id | type_label_ref .
266 parameter_type = generalized_types | named_types | simple_types .
```

Правила и ограничения:

a) Тип данных **AGGREGATE** должен использоваться только в качестве типа формального параметра функции или процедуры, либо типа результата функции, либо типа локальной переменной внутри функции или процедуры, либо представления явного или вычисляемого атрибута в объявлении типа данных **ABSTRACT ENTITY**.

b) Если тип данных **AGGREGATE** используется в качестве типа данных результата функции или типа данных локальной переменной внутри функции или процедуры, то для такого применения требуются ссылки меток типов. Ссылки меток типов должны ссылаться на метки типов, объявленные формальными параметрами (см. 9.5.3.4).

c) Если тип данных **AGGREGATE** используется в качестве представления явного или вычисляемого атрибута в объявлении типа данных **ABSTRACT ENTITY**, то данный атрибут должен быть повторно объявлен как **ARRAY**, **BAG**, **LIST** или **SET** в неабстрактных подтипах данного объектного типа данных.

Пример — Данная функция получает агрегированную структуру чисел. Функция должна возвращать результат того же типа, что и у переданной ей агрегированной структуры, содержащий масштабированные числа.

```
FUNCTION scale (input : AGGREGATE : intype OF REAL;
                scalar : REAL) : AGGREGATE : intype OF REAL;
  LOCAL
    result : AGGREGATE : intype OF REAL := input;
  END_LOCAL;
  IF SIZEOF ( ['BAG, 'SET'] * TYPEOF(input)) > 0 THEN
    REPEAT i := LOINDEX(input) TO HIINDEX(input);
      result := result – input[i];              -- удалить исходный
      result := result + scalar *input[i];      -- вставить масштабированный
    END_REPEAT;
  ELSE
    REPEAT i := LOINDEX(input) TO HIINDEX(input);
      result [I] := scalar*input[i];
    END_REPEAT;
  END_IF
  RETURN (result);
END_FUNCTION;
```

9.5.3.2 Обобщенный тип данных

Тип данных **GENERIC** является обобщением всех других типов данных.

При вызове процедуры или функции с типом данных формального параметра **GENERIC**, передаваемый фактический параметр может не иметь тип данных **GENERIC**. Операции, которые при этом могут быть выполнены, зависят от типа данных фактического параметра.

Примечание — Метки типов (см. 9.5.3.4) могут быть использованы для обеспечения того, чтобы при вызове два или более параметров, представленных типом данных **GENERIC**, имели одинаковый тип данных, или чтобы тип данных возвращаемого результата был таким же, как и у одного из переданных параметров, независимо от переданных фактических типов данных.

Синтаксис:

```
231 generic_type = GENERIC [ ':' type_label ] .
329 type_label = type_label_id | type_label_ref .
```

Правила и ограничения:

a) Тип данных **GENERIC** должен использоваться только в качестве типа формального параметра функции или процедуры, типа результата функции либо типа локальной переменной внутри функции или процедуры.

b) Если тип данных **GENERIC** используется в качестве типа данных результата функции или типа данных локальной переменной внутри функции или процедуры, то для такого применения требуются ссылки меток типов. Ссылки меток типов должны ссылаться на метки типов, объявленные формальными параметрами (см. 9.5.3.4).

Пример — В данном примере показана универсальная функция, осуществляющая сложение чисел или векторов:

```
FUNCTION add(a,b: GENERIC:intype): GENERIC:intype;
  LOCAL
    nr : NUMBER; --целое или действительное число
    vr : vector;
  END_LOCAL;
  IF ('NUMBER' IN TYPEOF(a)) AND ('NUMBER' IN TYPEOF(b)) THEN
     nr := a+b;
     RETURN(nr);
  ELSE
    IF ('THIS_SCHEMA.VECTOR' IN TYPEOF(a)) AND
       ('THIS_SCHEMA.VECTOR' IN TYPEOF(b)) THEN
      vr := vector(a.i + b.i,
                   a.j + b.j,
                   a.k + b.k);
      RETURN(vr);
    END_IF;
  END_IF;
  RETURN(?); --если получен неправильный входной параметр, то
             --возвращается неопределенное значение
END FUNCTION;
```

9.5.3.3 Обобщенный объектный тип данных

Тип данных **GENERIC_ENTITY** является обобщением всех объектных типов данных.

При вызове процедуры или функции с формальным параметром, имеющим тип данных **GENERIC_ENTITY**, передаваемый фактический параметр должен быть экземпляром объекта. Операции, которые при этом могут быть выполнены, зависят от типа данных фактического параметра.

Примечание — Метки типов (см. 9.5.3.4) могут быть использованы для обеспечения того, чтобы при вызове двух или более параметров, представленных типом данных **GENERIC_ENTITY**, имели одинаковый тип данных или чтобы тип данных возвращаемого результата был таким же, как и у одного из переданных параметров, независимо от переданных фактических типов данных.

Если в объявлении типа данных **ABSTRACT ENTITY** явный или вычисляемый атрибут представлен типом данных **GENERIC_ENTITY**, то данный атрибут должен быть объявлен в подтипах данного объекта конкретным объектным типом данных.

Синтаксис:
230 generic_entity_type = GENERIC_ENTITY [':' type_label] .
329 type_label = type_label_id | type_label_ref .

Правила и ограничения:

a) Тип данных **GENERIC_ENTITY** должен использоваться только в качестве типа формального параметра функции или процедуры либо типа результата функции, либо типа локальной переменной внутри функции или процедуры, либо представления явного или вычисляемого атрибута в объявлении типа данных **ABSTRACT ENTITY**.

b) Если тип данных **GENERIC_ENTITY** используется в качестве типа данных результата функции или типа данных локальной переменной внутри функции или процедуры, то для такого применения требуются ссылки меток типов. Ссылки меток типов должны ссылаться на метки типов, объявленные формальными параметрами (см. 9.5.3.4).

Пример — Приведенная ниже функция проверяет, имеется ли ссылка на конкретный экземпляр* sample *от двух экземпляров* type1 *и* type2 *известных объектных типов данных. Объявление формального параметра* sample, *как имеющего тип данных* GENERIC_ENTITY, *позволяет рассматривать экземпляры любых объектных типов данных как допустимые входные параметры данной функции.

```
FUNCTION check_relating (type1    : instance_of_type_1;
                         type2    : instance_of_type_2;
                         sample : GENERIC_ENTITY) : BOOLEAN;
  RETURN ((type1 IN USEDIN (sample, '')
           AND
           (type2 IN USEDIN (sample, '') ));
END_FUNCTION;
```

9.5.3.4 Метки типов

Метки типов должны использоваться для установления связи между типом данных атрибута или фактического параметра в момент обращения с типами данных других атрибутов или фактических параметров, локальных переменных или возвращаемого значения функции. Метки объявляются для следующих типов данных:

- для типов данных **AGGREGATE, GENERIC_ENTITY** и **GENERIC** метки объявляются в объявлении формальных параметров функции или процедуры. Впоследствии на них могут ссылаться типы данных **AGGREGATE**, **GENERIC_ENTITY** и **GENERIC** в объявлении формальных параметров или локальных переменных функции или процедуры либо в объявлении типа данных возвращаемого значения функции;
- для типов данных **AGGREGATE** и **GENERIC_ENTITY** метки объявляются в объявлении явных или вычисляемых атрибутов абстрактных объектов. Впоследствии на них могут ссылаться типы данных **AGGREGATE** и **GENERIC_ENTITY** в оставшейся части объявления объектных типов данных.

Синтаксис:
329 type_label = type_label_id | type_label_ref .

Правила и ограничения:

a) Первое появление метки типа в объявлении формального параметра либо явного или вычисляемого атрибута объявляет метку данного типа; последующие случаи употребления данной метки типа являются ссылками на ее первое употребление.

b) Параметры, переданные функции или процедуре, в которых используется ссылка на метку типа, должны быть совместимы с типом данных переданного параметра, в котором объявлена данная метка типа.

c) Типы данных локальных переменных и возвращаемых значений функций, которые ссылаются посредством метки типа на тип данных параметра, должны быть идентичны типу данных параметра, в котором объявлена данная метка типа.

d) Типы данных атрибутов, которые ссылаются посредством метки типа на тип данных атрибута, должны быть идентичны типу данных атрибута, в котором объявлена данная метка типа.

Пример — В данном примере показано, как метки типов могут использоваться для проверки совместимости типов данных при вызове функции.

```
ENTITY a;
...
END ENTITY;
```

```
ENTITY b SUBTYPE OF (a);
...
END_ENTITY;
ENTITY c SUBTYPE OF (b);
...
END_ENTITY;
...
FUNCTION test ( pi: GENERIC:x; p2: GENERIC:x) : GENERIC:x;
...        --                  ^              ^          ^
...        --                  объявление     ссылка     ссылка
END_FUNCTION;
...
LOCAL
   v_a : a := a(...);
   v_b : b := a(...)||b(...); --оператор || определен в 12.10
   v_c : c := a(...)||b(...)||c(...);
   v_x : b;
END LOCAL;
v_x := test(v_b, v_a); --неверный v_a, не совместимый с типом b
v_x := test(v_a, v_b); --неверное присваивание, функция вернет тип a
```

Другие примеры использования меток типов приведены в разделе 15.

9.5.3.5 Общие агрегированные типы данных

Общие агрегированные типы данных образуют часть класса типов данных, называемых обобщенными типами данных. Общий агрегированный тип данных представляют обобщение соответствующих агрегированных типов данных (**ARRAY**, **BAG**, **LIST** и **SET**), позволяющее представлять тип данных элемента схемы обобщенным типом данных. То есть элемент **general_list_type** в приведенном ниже определении синтаксиса является обобщением элемента **list_type** так же, как и для типов данных **ARRAY**, **BAG** и **SET**.

Синтаксис:

```
224 general_aggregation_types = general_array_type | general_bag_type |
                                general_list_type | general_set_type .
225 general_array_type = ARRAY [ bound_spec ] OF [ OPTIONAL ] [ UNIQUE ]
                         parameter_type .
185 bound_spec = '[' bound_1 ':' bound_2 ']' .
183 bound_1 = numeric_expression .
184 bound_2 = numeric_expression .
266 parameter_type = generalized_types | named_types | simple_types .
226 general_bag_type = BAG [ bound_spec ] OF parameter_type .
227 general_list_type = LIST [ bound_spec ] OF [ UNIQUE ] parameter_type .
229 general_set_type = SET [ bound_spec ] OF parameter_type .
```

Общий агрегированный тип данных обобщает соответствующий агрегированный тип следующим образом:

- обобщенный массив может быть определен без указания значений индексов. При этом в спецификации формального параметра не указываются границы массива (элемент синтаксиса **bound_spec**).

Примечание — Для определения реальной системы индексирования массива в алгоритмической части должны использоваться функции **HIINDEX** и **LOINDEX**;

- базисным типом данных может быть **GENERIC**, **GENERIC_ENTITY**, **AGGREGATE** или общий агрегированный тип данных при использовании в качестве формального параметра функции или процедуры либо **GENERIC_ENTITY**, **AGGREGATE** или общий агрегированный тип данных при использовании в абстрактном объектном типе данных. Это определено в формализованной форме ниже.

Пусть **G** является общим агрегированным типом данных, а **EG** — (обобщенным) типом данных его элементов. Пусть для формального параметра функции или процедуры **A** является типом данных соответствующего фактического параметра; пусть для атрибута абстрактного объектного типа данных **A** является типом данных повторно объявленного атрибута неабстрактного подтипа и пусть для любого случая **EA** является типом данных элементов **A**.

Если **G** является обобщенным массивом, то **A** должно быть типом данных **ARRAY**. Если для **G** задан диапазон индексов, то диапазон индексов **A** должен быть таким же.

Если **G** является обобщенным пакетом, то **A** должно быть типом данных **BAG**. Если для **G** заданы границы, то границы **A** должны быть такими же.

Если **G** является обобщенным списком, то **A** должно быть типом данных **LIST**. Если для **G** заданы границы, то границы **A** должны быть такими же.

Если **G** является обобщенным набором, то **A** должно быть типом данных **SET**. Если для **G** заданы границы, то границы **A** должны быть такими же.

Если **G** является любым обобщенным типом данных, то **EA** должно соответствовать **EG**, как определено в 9.5.

Если **EG** не является обобщенным типом данных, то **EA** должно быть присваиванием, совместимым с **EG**, как определено в 13.3.

***Пример — В данном примере показано, как набор (тип данных* SET*) может быть описан в объявлении формального параметра. Он не может быть описан в объявлении атрибута, поскольку базисный тип данных для* SET *не включает тип данных* GENERIC.**

```
FUNCTION dimensions(input: SET [2:3] OF GENERIC):INTEGER;
```

9.5.4 Локальные переменные

Переменные, локальные по отношению к данному алгоритму, объявляются после ключевого слова **LOCAL**. Локальная переменная видима только в области видимости алгоритма, в котором она объявлена. Локальным переменным могут быть присвоены значения, и они могут присутствовать в выражениях.

```
Синтаксис:
  252 local_decl = LOCAL local_variable {local_variable} END_LOCAL ';'.
  253 local_variable = variable_id {',' variable_id} ':' parameter_type
                       [':=' expression] ';'.
  266 parameter_type = generalized_types | named_types | simple_types.
```

Инициализация локальных переменных:

Локальная переменная может появиться при инициализации другой зависимой локальной переменной. Объявление зависимых локальных переменных должно быть ациклическим. Если никакого начального значения не задано, то локальной переменной присваивается неопределенное (**?**) значение.

Примечания

1 Требование ацикличности объявлений локальных переменных необходимо, чтобы всегда обеспечивать существование допустимого начального значения, присваиваемого в порядке следования объявления.

2 Поскольку неопределенное (**?**) значение совместимо со всеми типами данных, то допустима явная инициализация с неопределенным (**?**) значением.

***Пример — Переменная* r_result *инициализируется со значением* 0.0:**

```
LOCAL
   r_result : REAL := 0.0;
   i_result : INTEGER;
END_LOCAL;
...
EXISTS(r_result)  -- TRUE
EXISTS(i_result)  -- FALSE поскольку никакого значения не присвоено
```

9.6 Правило

Правила позволяют определить ограничения, применяемые к одному или нескольким типам данных в пределах области видимости схемы. Локальные правила (к которым относятся ограничения уникальности и правила области видимости в объявлении объектов) объявляют ограничения, применяемые индивидуально к каждому экземпляру объектного типа данных. Объявление **RULE** позволяет определить ограничения, применяемые в совокупности ко всей области определения объектного типа данных или к экземплярам нескольких объектных типов данных. Одним из применений объявления **RULE** является скоординированное ограничение значений атрибутов разных объектов.

В объявлении правила ему присваивается имя и указываются объекты, на которые оно распространяется.

Тело правила состоит из локальных объявлений, исполняемых операторов и правил области определения. Конечное состояние правила показывает, удовлетворяется или нет некоторое глобальное

ограничение. Правило оценивается посредством выполнения операторов с последующей оценкой каждого из правил области определения. Если правило нарушено для совокупности экземпляров объектных типов данных, переданных в качестве параметров, то данные экземпляры не соответствуют EXPRESS-схеме.

Синтаксис:

```
291 rule_decl = rule_head algorithm_head { stmt } where_clause END_RULE ';' .
292 rule_head = RULE rule_id FOR '(' entity_ref { ',' entity_ref } ')' ';' .
173 algorithm_head = { declaration } [ constant_decl ] [ local_decl ] .
199 declaration = entity_decl | function_decl | procedure_decl |
                  subtype_constraint_decl | type_decl .
```

Правила и ограничения:

a) Результатом оценки любого правила области определения должно быть логическое (**LOGICAL**) или неопределенное (**?**) значение.

b) Выражение принимается, если оценкой его значения является **TRUE**; выражение отвергается, если оценкой его значения является **FALSE**; и выражение ни отвергается, ни принимается, если оценкой выражения является неопределенное (**?**) значение или значение **UNKNOWN**.

c) Ни одно из правил области определения не должно быть отвергнуто для допустимой совокупности экземпляров объектов объектных типов данных, указанных в заголовке правила.

d) Для совокупности экземпляров, принадлежащих допустимой области определения, все глобальные правила, заданные для данной области определения, должны быть приняты. Данное правило относится и к принятию правил для объектных типов данных, для которых не существует экземпляров в совокупности тестируемых экземпляров.

Примечание — Глобальное правило может быть задано для обеспечения существования, по крайней мере, одного экземпляра указанного типа данных. Данное правило не проверяет отсутствие экземпляров указанного объектного типа данных, поддерживающих необходимую семантику.

Примеры

1 Следующее правило устанавливает, что в первом и седьмом октантах должно быть одинаковое число точек:

```
RULE point_match FOR (point);
LOCAL
   first_oct ,
   seventh_oct : SET OF POINT := [ ]; -- пустой набор точек (см. 12.9)
END LOCAL
   first_oct := QUERY(temp <* point | (temp.x > 0) AND
                                      (temp.y > 0) AND
                                      (temp.z > 0) );
   seventh_oct := QUERY(temp <* point | (temp.x < 0) AND
                                        (temp.y < 0) AND
                                        (temp.z < 0) );
WHERE
   SIZEOF(first_oct) = SIZEOF(seventh_oct);
END RULE;
```

2 Правило может быть использовано для задания совокупной уникальности значений атрибутов объекта:

```
ENTITY b;
   a1 : c;
   a2 : d;
   a3 : f;
UNIQUE
   ur1 : a1, a2;
END_ENTITY;
```

Ограничение совместной уникальности в b ***применяется к экземплярам*** c ***и*** d. ***Следующее правило вводит дополнительное ограничение, что совокупная уникальность должна основываться на значениях:***

```
RULE vu FOR (b);
   ENTITY temp;
      a1 : c;
      a2 : d;
   END ENTITY;
```

```
LOCAL
   s : SET OF temp := [ ];
END LOCAL;
REPEAT i := 1 TO SIZEOF(b);
   s := s + temp(b[i].a1, b[i].a2);
END REPEAT;
WHERE
   wrl : VALUE_UNIQUE(s);
END_RULE;
```

Неявное объявление.

В объявлении **RULE** каждый синтаксический элемент **population** неявно объявлен локальной переменной, содержащей множество всех экземпляров именованного объектного типа данных из области определения. Данное множество экземпляров объекта подчиняется данному правилу.

Синтаксис:

267 population = entity_ref .

Правило: ссылки на конкретный элемент **population** могут быть сделаны только в глобальном правиле, которое ссылается на соответствующий объектный тип данных в заголовке данного правила.

Пример — При наличии следующего объявления:

```
RULE coincident FOR (point);
```

неявно объявленная переменная может выглядеть следующим образом:

```
LOCAL
   point : SET OF point;
END_LOCAL;
```

9.7 Ограничения подтипов

Понятия подтипа и супертипа определены в 9.2.3. Понятие ограничений подтипов/супертипов определено в 9.2.5. Существует возможность определить ограничения, в соответствии с которыми графы подтипов/супертипов могут быть реализованы вне объявления объекта. Данная возможность реализуется посредством объявления **SUBTYPE_CONSTRAINT**.

Синтаксис:

```
315 subtype_constraint_decl   = subtype_constraint_head subtype_constraint_body
                                END_SUBTYPE_CONSTRAINT ';' .
316 subtype_constraint_head   = SUBTYPE_CONSTRAINT subtype_constraint_ID FOR
                                entity_ref ';' .
314 subtype_constraint_body   = [ abstract_supertype ] [total_over ]
                                [ supertype_expression ';' .
165 abstract_supertype  = ABSTRACT SUPERTYPE ';' .
326 total_over  = TOTAL_OVER '(' entity_ref { ',' entity_ref } ')' ';' .
320 supertype_expression  = supertype_factor { ANDOR supertype_factor } .
321 supertype_factor  = supertype_term { AND supertype_term } .
323 supertype_term – entity_ref | one_of | '(' supertype_expression ')' .
263 one_of  = ONEOF '(' supertype_expression { ',' supertype_expression } ')' .
```

Объявление **SUBTYPE_CONSTRAINT** используется для определения следующих ограничений на возможную реализацию подтипов/супертипов:

- ограничение, что супертип является абстрактным и должен реализовываться только через свои подтипы;

- ограничение, что совокупность подтипов данного супертипа обеспечивает полное покрытие; то есть если полное покрытие задано, то экземпляр любого подтипа данного супертипа должен также быть экземпляром, по крайней мере, одного из подтипов, определенных в спецификации **TOTAL_OVER**;

- ограничение взаимосвязи между некоторыми подтипами.

Каждый из этих видов ограничений более детально рассмотрен в последующих пунктах. Формальный подход к определению потенциальных комбинаций подтип/супертип, которые могут быть реализованы при нескольких возможных ограничениях, описанных ниже, представлен в приложении В.

9.7.1 Ограничение абстрактного супертипа

Объявление **ABSTRACT SUPERTYPE**, определенное в 9.2.5.1, может также присутствовать в объявлении **SUBTYPE_CONSTRAINT**.

Правило: абстрактный супертип определяется объявлением **SUBTYPE_CONSTRAINT** в супертипе с использованием ключевых слов **ABSTRACT SUPERTYPE**.

Пример — В общей классификационной модели может потребоваться идентифицировать объект с именем **class**, ***который в данном контексте является реализуемым. В более специфичной модели может потребоваться использовать объект*** **class**, ***но ограничить его так, чтобы он мог быть реализован только через свои локально объявленные подтипы.***

```
SCHEMA general_classification_model;
ENTITY class;
   name : class_name;
END_ENTITY;
END_SCHEMA;
SCHEMA specific_classification_model;
USE FROM general_classification_model;
ENTITY class_of_facility;
   SUBTYPE OF (class);
END_ENTITY;
ENTITY class_of_organization
   SUBTYPE OF (class);
END_ENTITY;
SUBTYPE_CONSTRAINT independent_classification FOR class;
   ABSTRACT SUPERTYPE;
   ONEOF(class_of_facility, class_of_organization);
END_SUBTYPE_CONSTRAINT;
END_SCHEMA;
```

9.7.2 Подтипы полного покрытия

Ограничение полного покрытия **TOTAL_OVER** устанавливает, что каждый экземпляр супертипа должен быть экземпляром одного или нескольких заданных множеств подтипов. Другими словами, для заданного контекста область определения супертипа в точности совпадает с объединением множеств областей определения именованных подтипов.

Пример — Понятие личность полностью покрывается понятиями мужчина и женщина. Могут существовать и другие понятия, но любая личность является либо мужчиной, либо женщиной. Поэтому можно сказать, что супертип **person** ***(личность) полностью покрывается подтипами*** **male** ***(мужчина) и*** **female** ***(женщина).***

Если в двух или более ограничениях подтипов заданы ограничения **TOTAL_OVER** для одного и того же объектного типа данных, то такие ограничения **TOTAL_OVER** рассматриваются совместно. Это означает, что оба ограничения **TOTAL_OVER (a,b)** и **TOTAL_OVER (c,d)** должны выполняться.

Правила и ограничения:

a) Все подтипы, указанные в одном или нескольких ограничениях **TOTAL_OVER** для заданного супертипа, должны быть непосредственными подтипами данного супертипа.

b) Экземпляры других подтипов, как бы эти подтипы ни были определены или ограничены, также должны быть экземплярами одного или нескольких подтипов, указанных в спецификации **TOTAL_OVER**.

c) Поскольку супертип может иметь несколько контекстов, то он также может иметь несколько ограничений **TOTAL_OVER**.

Пример — В данном примере определяется, что объект **person** ***может быть объектом*** **male** ***либо объектом*** **female**. ***В примере ничего не говорится о взаимосвязи между объектами*** **male** ***и*** **female**, ***поэтому можно создать экземпляр, который является одновременно как*** **male**, ***так и*** **female**. ***Подтип*** **employee** ***всегда должен сочетаться с понятиями*** **male** ***и*** **female** ***и не может быть реализован независимо от них.***

```
ENTITY person;
   name : personal_name;
END_ENTITY;
ENTITY male
   SUBTYPE OF (person);
   . . .
END_ENTITY;
```

```
ENTITY female
  SUBTYPE OF (person);
  ...
END_ENTITY;
ENTITY employee
  SUBTYPE OF (person);
  ...
END_ENTITY;
SUBTYPE_CONSTRAINT person_sex FOR person;
  ABSTRACT SUPERTYPE;
  TOTAL_OVER (male, female);
END_SUBTYPE_CONSTRAINT;
```

9.7.3 Перекрывающиеся подтипы и их спецификация

Два или несколько непосредственных подтипов конкретного супертипа могут иметь перекрывающиеся реализации для конкретного контекста. Спецификация **SUBTYPE_CONSTRAINT** может быть использована для определения того, какие отношения установлены в конкретной группе непосредственных подтипов.

9.7.3.1 **ONEOF**

Ограничение **ONEOF**, определенное в 9.2.5.2, может быть объявлено в ограничении **SUBTYPE_CONSTRAINT**.

Пример — Экземпляр супертипа может быть порожден посредством реализации только одного из своих подтипов. Данное ограничение объявляется с использованием ограничений **ABSTRACT** ***и*** **ONEOF**. ***Существует много видов животных (объект*** **pet**), ***но ни один объект*** **pet** ***не может быть одновременно двумя или более видами животных.***

```
ENTITY pet
  name : pet_name;
  ...
END_ENTITY;
SUBTYPE_CONSTRAINT separate_species FOR pet;
  ABSTRACT SUPERTYPE;
  ONEOF(cat, rabbit, dog, ... );
END_SUBTYPE_CONSTRAINT;
ENTITY cat
  SUBTYPE OF (pet);
  ...
END_ENTITY;
ENTITY rabbit
  SUBTYPE OF (pet);
  ...
END_ENTITY;
ENTITY dog
  SUBTYPE OF (pet);
  ...
END_ENTITY;
```

9.7.3.2 **ANDOR**

Ограничение **ANDOR**, определенное в 9.2.5.3, может быть объявлено в ограничении **SUBTYPE_CONSTRAINT**.

Пример — Личность (объект **person**) ***может быть работником (объект*** **employee**), ***посещающим вечерние курсы, и поэтому может быть одновременно и работником и студентом (объект*** **student**).

```
ENTITY person
  ...
END_ENTITY;
SUBTYPE_CONSTRAINT employee_may_be_student FOR person;
  employee ANDOR student;
END_SUBTYPE_CONSTRAINT;
ENTITY employee
  SUBTYPE OF (person);
  ...
END_ENTITY;
```

```
ENTITY student
  SUBTYPE OF (person);
  ...
END_ENTITY;
```

9.7.3.3 **AND**

Ограничение **AND**, определенное в 9.2.5.4, может быть объявлено в ограничении **SUBTYPE_CONSTRAINT**.

Пример — Личность (объект **person)** ***может быть классифицирована как мужчина (объект*** **male)** ***или как женщина (объект*** **female)***, но также может быть классифицирована как гражданин (объект*** **citizen)** ***или иностранец (объект*** **alien)***.

```
ENTITY person
  ...
END_ENTITY;
SUBTYPE_CONSTRAINT no_mixing FOR person;
  SUPERTYPE OF
     (ONEOF(male, female) AND
      ONEOF(citizen, alien));
END_SUBTYPE_CONSTRAINT;
ENTITY male
  SUBTYPE OF (person);
  ...
END_ENTITY;
ENTITY female
  SUBTYPE OF (person);
  ...
END_ENTITY;
ENTITY citizen
  SUBTYPE OF (person);
  ...
END_ENTITY;
ENTITY alien
  SUBTYPE OF (person);
  ...
END_ENTITY;
```

10 Область видимости и видимость

В языке EXPRESS объявление создает идентификатор, который может быть использован для ссылок на объявленный элемент в других частях данной схемы (или в других схемах). В некоторых конструкциях языка EXPRESS неявно объявляются элементы языка с присваиванием им идентификаторов. Считается, что в тех местах, где может указываться ссылка на идентификатор объявленного элемента, данный объявленный элемент является видимым. На элемент может быть дана ссылка только там, где его идентификатор является видимым. Правила видимости определены в 10.2. Более подробная информация о ссылках на элементы с использованием их идентификаторов приведена в 12.7.

Совокупность элементов языка EXPRESS образует область (блок) текста, называемый областью видимости данного элемента. Данная область видимости ограничивает видимость объявленных в ней идентификаторов. Области видимости могут быть вложенными, то есть элемент языка EXPRESS, имеющий свою область видимости, может быть включен в область видимости другого элемента. Существуют ограничения, в соответствии с которыми элементы могут появляться в области видимости конкретного элемента языка EXPRESS. Данные ограничения, как правило, определяются синтаксисом языка EXPRESS (см. приложение А).

Для каждого из элементов языка, представленных в таблице 9, в последующих подразделах установлены границы его области видимости (при ее существовании) и видимость объявленного идентификатора как в общих терминах, так и с конкретными деталями.

10.1 Правила области видимости

Ниже приведены общие правила, применимые ко всем формам определения области видимости, допустимым в языке EXPRESS. Перечень элементов языка, определяющих области видимости, приведен в таблице 9.

Правила и ограничения:

a) Все объявления должны существовать в области видимости.

b) В одной области видимости идентификатор может быть объявлен или в явной форе включен (см. раздел 11) только один раз. Идентификатор объекта или типа данных, который был в явной форме включен в данную схему по двум или более маршрутам, основанным на одном исходном объявлении, учитывается только один раз.

c) Области видимости должны быть вложены корректно, то есть области видимости не должны перекрываться (это диктуется синтаксисом языка).

Максимально допустимая глубина вложения не устанавливается настоящим стандартом, но в реализациях синтаксических анализаторов языка EXPRESS может задаваться максимальная глубина вложения областей видимости.

Таблица 9 — Области действия и идентификаторы, определяющие элементы языка

Элемент языка	Область видимости	Идентификатор
Оператор альтернативных имен	•	• [1]
Атрибут		•
Константа		•
Перечисление		•
Объект	•	•
Функция	•	•
Параметр		•
Процедура	•	•
Выражение **QUERY**	•	• [1]
Оператор цикла	•	• [1], [2]
Правило	•	• [3]
Метка правила		•
Схема	•	•
Ограничение подтипа	•	•
Тип	•	•
Метка типа		•
Переменная		•

[1] Идентификатором является неявно объявленная переменная в определенной области видимости объявления.

[2] Переменная объявляется неявно только тогда, когда установлен инкрементный контроль.

[3] Неявное объявление переменной осуществляется для всех объектов, ограниченных данным правилом.

10.2 Правила видимости

Ниже определены правила видимости для идентификаторов. Перечень элементов языка EXPRESS, объявляющих идентификаторы, представлен в таблице 9.

Правила и ограничения:

a) Идентификатор виден в области видимости, в которой он объявлен. Эта область видимости называется локальной областью видимости идентификатора.

b) Если идентификатор виден в некоторой области видимости, то он также виден во всех областях видимости, определенных внутри данной области, с учетом правила по перечислению d).

c) Идентификатор не виден в любой области видимости вне его локальной области видимости, с учетом правила по перечислению f).

d) Если идентификатор ***i***, видимый в области видимости ***P***, повторно объявлен в некоторой внутренней области видимости ***Q***, вложенной в ***P***, то:

- если ***i***, объявленный в ***P***, ссылается на именованный тип данных или на метку типа, а ***i***, объявленный в ***Q***, не ссылается на именованный тип данных или на метку типа, то, как ***i***, объявленный в ***P***, так и ***i***, объявленный в ***Q***, являются видимыми в ***Q***;
- в противном случае, только ***i***, объявленный в ***Q***, является видимым в ***Q*** и в любых областях видимости, объявленных в ***Q***. Идентификатор ***i***, объявленный в ***P***, будет виден в ***P*** и во всех внутренних областях видимости, в которых ***i*** не объявляется повторно.

e) Встроенные константы, функции, процедуры и типы данных языка EXPRESS считаются объявленными в виртуальной всеобщей области видимости. Все схемы являются вложенными в данную область видимости. Идентификаторы, ссылающиеся на встроенные константы, функции, процедуры, типы данных языка EXPRESS и схемы, являются видимыми во всех областях видимости, определенных в языке EXPRESS.

f) Идентификаторы элементов перечисления, объявленные в области видимости определенного типа данных, являются видимыми там, где данный определенный тип данных является видимым, за исключением случая, когда внешняя область видимости содержит объявление того же идентификатора для какого-либо другого элемента.

Примечание — Если следующая внешняя область видимости содержит объявление того же идентификатора, то элементы перечисления остаются доступными, но к ним должен быть добавлен в качестве префикса идентификатор определенного типа данных (см. 12.7.2).

g) Объявления из одной схемы становятся видимыми для элементов другой схемы посредством спецификации интерфейса (см. раздел 11).

Пример — В следующей схеме показаны примеры идентификаторов и ссылок, являющихся допустимыми в соответствии с приведенными выше правилами.

```
SCHEMA example;
CONSTANT
   b : INTEGER  := 1 ;
   c : BOOLEAN := TRUE ;
END_CONSTANT;
TYPE enum = ENUMERATION OF ( e, f, g );
END_TYPE;
ENTITY entity1;
   a : INTEGER;
WHERE
   wr1: a > 0 ;   -- "entity1.wr1" подчиняется правилу по перечислению a):
                  -- идентификатор "a" виден в локальной области видимости
   wr2: a <> b ;  -- "entity1.wr2" подчиняется правилу по перечислению b):
                  -- идентификатор "b" виден из внешней области видимости
END_ENTITY;
ENTITY entity2;
c : REAL;         -- "entity2.c" подчиняется правилу по перечислению c):
                  -- константа "c" не видима здесь
END_ENTITY;
ENTITY d;
   attr1 : INTEGER;
   attr2 : enum;
WHERE
   wr1: ODD(attr1); -- "d.wr1" подчиняется правилу по перечислению d):
                    -- функция ODD видима везде
   wr2: attr2 <> e; -- "d.wr2" подчиняется правилу по перечислению e):
                    -- идентификатор "e" виден вне области видимости,
                    -- определенной типом данных enum
   END_ENTITY;
END SCHEMA;
```

10.3 Правила для явных элементов

В данном разделе более подробно определено, как общие правила областей действия и видимости применяются к разным элементам языка EXPRESS.

10.3.1 Оператор альтернативных имен ALIAS

Определение оператора **ALIAS** дано в 13.2.

Видимость: идентификатор, неявно объявленный в операторе **ALIAS**, является видимым в области видимости, определенной данным оператором.

Область видимости: оператор **ALIAS** определяет новую область видимости. Данная область видимости размещается от ключевого слова **ALIAS** до ключевого слова **END_ALIAS**, которым завершается данный оператор альтернативных имен.

10.3.2 Атрибут

Видимость: идентификатор атрибута является видимым в областях видимости объекта, в которой он объявлен, и всех подтипов данного объекта.

10.3.3 Константа

Видимость: идентификатор константы является видимым в области видимости функции, процедуры, правила или схемы, в которой он объявлен.

10.3.4 Элемент перечисления

Видимость: идентификатор элемента перечисления является видимым во всех областях видимости, в которых является видимым определенный тип данных, в котором объявлен данный элемент перечисления, за исключением случая, когда такая внешняя область видимости содержит объявление такого же идентификатора для какого-либо другого элемента.

10.3.5 Объект

Видимость: идентификатор объекта является видимым в области видимости функции, процедуры, правила или схемы, в которой он объявлен. Идентификатор объекта остается видимым при условиях, определенных в 10.2, во внутренних областях видимости, в которых данный идентификатор объявлен повторно.

Область видимости: объявление объекта определяет новую область видимости. Данная область видимости размещается от ключевого слова **ENTITY** до ключевого слова **END_ENTITY**, которым завершается объявление данного объекта. Атрибуты, объявленные в супертипе объекта, являются видимыми в объекте подтипа по принципу наследования.

Примечание — Область видимости объекта подтипа не считается вложенной в область видимости супертипа.

Объявления: следующие элементы языка EXPRESS могут объявлять идентификаторы, видимые в области видимости объявления объекта:

- атрибут (явный, вычисляемый и инверсный);
- метка правила (правил уникальности и области определения).

Примеры

1 Идентификаторы атрибута **batt** ***в двух объектах не конфликтуют, поскольку они объявлены в двух разных областях видимости.***

```
ENTITY entity1;
   aatt : INTEGER;
   batt : INTEGER;
END_ENTITY;
ENTITY entity2;
   a    : entity1;
   batt : INTEGER;
END_ENTITY;
```

2 Приведенная ниже спецификация недопустима, поскольку идентификатор атрибута **aatt** ***одновременно и наследуется, и объявляется в области видимости объекта*** **illegal** ***(см. 9.2.3.3). Метки правила*** **lab** ***в двух объектах не конфликтуют, поскольку они объявлены в разных областях видимости; допустимый экземпляр объекта*** **illegal**, ***игнорирующий ошибку с атрибутом*** **aatt**, ***подчиняется обоим правилам области определения.***

```
ENTITY may_be_ok;
   quantity : REAL;
   aatt     : REAL;
WHERE
   lab : quantity >= 0.0;
END_ENTITY;
```

```
ENTITY illegal
   SUBTYPE OF (may_be_ok);
   aatt : INTEGER;
   batt : INTEGER;
WHERE
   lab : batt < 0;
END_ENTITY;
```

10.3.6 Функция

Видимость: идентификатор функции является видимым в области видимости функции, процедуры, правила или схемы, в которой он объявлен.

Область видимости: объявление функции определяет новую область видимости. Данная область видимости размещается от ключевого слова **FUNCTION** до ключевого слова **END_FUNCTION**, которым завершается объявление данной функции.

Объявления: следующие элементы языка EXPRESS могут объявлять идентификаторы, видимые в области видимости объявления функции:

- константа;
- объект;
- перечисление;
- функция;
- параметр;
- процедура;
- тип;
- метка типа;
- переменная.

Пример — Приведенный ниже фрагмент некорректен, поскольку идентификатор формального параметра parm используется одновременно и как идентификатор локальной переменной.

```
FUNCTION illegal(parm : REAL) : LOGICAL;
LOCAL
   parm : STRING;
END_LOCAL;
   ...
END_FUNCTION;
```

10.3.7 Параметр

Видимость: идентификатор формального параметра является видимым в области видимости функции или процедуры, в которой он объявлен.

10.3.8 Процедура

Видимость: идентификатор процедуры является видимым в области видимости функции, процедуры, правила или схемы, в которой он объявлен.

Область видимости: объявление процедуры определяет новую область видимости. Данная область видимости размещается от ключевого слова **PROCEDURE** до ключевого слова **END_PROCEDURE**, которым завершается объявление процедуры.

Объявления: следующие элементы языка EXPRESS могут объявлять идентификаторы в области видимости объявления процедуры:

- константа;
- объект;
- перечисление;
- функция;
- параметр;
- процедура;
- тип;
- метка типа;
- переменная.

10.3.9 Выражение QUERY

Выражение **QUERY** определено в 12.6.7.

Видимость: идентификатор, неявно объявленный в выражении **QUERY**, является видимым в области видимости, определенной данным выражением.

Область видимости: выражение **QUERY** определяет новую область видимости. Данная область видимости размещается от открывающей круглой скобки '**(**', следующей за ключевым словом **QUERY**, до закрывающей круглой скобки '**)**', которой завершается данное выражение **QUERY**.

10.3.10 Оператор цикла

Оператор цикла **REPEAT** определен в 13.9.

Видимость: идентификатор, неявно объявленный в инкрементно управляемом операторе цикла, является видимым в области видимости данного оператора цикла.

Область видимости: оператор цикла определяет новую область видимости. Данная область видимости размещается от ключевого слова **REPEAT** до ключевого слова **END_REPEAT**, которым завершается оператор цикла.

10.3.11 Правило

Видимость: идентификатор правила является видимым в области видимости схемы, в которой он объявлен.

Примечание — Идентификатор правила может использоваться в реализациях или в метке комментария (см. 7.1.6.3).

Область видимости: объявление правила определяет новую область видимости. Данная область видимости размещается от ключевого слова **RULE** до ключевого слова **END_RULE**, которым завершается объявление данного правила.

Объявления: следующие элементы языка EXPRESS могут объявлять идентификаторы в области видимости объявления правила:

- константа;
- объект;
- перечисление;
- функция;
- процедура;
- метка правила;
- тип;
- переменная.

Пример — Приведенный ниже фрагмент некорректен, поскольку идентификатор* point, *относящийся к объекту, на который распространяется правило, неявно объявлен как переменная в данном правиле и, кроме того, явно объявлен как локальная переменная.

```
RULE illegal FOR (point);
LOCAL
   point : STRING;
END_LOCAL;
   . . .
END_RULE;
```

10.3.12 Метка правила

Видимость: метка правила является видимой в области видимости объекта, правила или типа, в которой она объявлена.

Примечание — Метка правила может использоваться в реализациях и в метке комментария (см. 7.1.6.3).

10.3.13 Схема

Видимость: идентификатор схемы является видимым для всех других схем.

Примечание — Совместимая реализация может обеспечивать механизм обобщения областей видимости, позволяющий рассматривать совокупность схем как область видимости.

Область видимости: объявление схемы определяет новую область видимости. Данная область видимости размещается от ключевого слова **SCHEMA** до ключевого слова **END_SCHEMA**, которым завершается объявление данной схемы.

Объявления: следующие элементы языка EXPRESS могут объявлять идентификаторы в области видимости объявления схемы:

- константа;
- объект;

- перечисление;
- функция;
- процедура;
- правило;
- ограничение подтипа;
- тип.

Пример — Приведенный ниже фрагмент некорректен по двум причинам. Во-первых, идентификатор* adef *импортирован в схему посредством оператора* USE, *но повторно объявлен как имя типа. Во-вторых, имя* fdef *использовано как идентификатор в двух объявлениях (относящихся к объекту и к функции, хотя тип элемента является несущественным).

```
SCHEMA incorrect;
USE FROM another_schema (adef);
  FUNCTION fdef(parm : NUMBER) : INTEGER;
  ...
  END_FUNCTION;
  TYPE adef = STRING;
  END_TYPE;
  ENTITY fdef;
  ...
  END_ENTITY;
END_SCHEMA;
```

10.3.14 Ограничение подтипа

Видимость: идентификатор ограничения подтипа является видимым в области видимости схемы, в которой он объявлен.

Примечание — Идентификатор ограничения подтипа может использоваться в реализациях и в метке комментария (см. 7.1.6.3).

Область видимости: ограничение подтипа расширяет область видимости объекта, для которого он объявлен. Данное расширение области видимости размещается от ключевого слова **SUBTYPE_CONSTRAINT** до ключевого слова **END_SUBTYPE_CONSTRAINT**, которым завершается объявление данного ограничения подтипа.

10.3.15 Тип

Видимость: идентификатор типа является видимым в области видимости функции, процедуры, правила или схемы, в которой он объявлен. Идентификатор типа остается видимым при условиях, определенных в 10.2, во внутренних областях видимости, в которых данный идентификатор объявляется повторно.

Область видимости: объявление типа создает новую область видимости. Данная область видимости размещается от ключевого слова **TYPE** до ключевого слова **END_TYPE**, которым завершается объявление данного типа.

Объявления: следующие элементы языка EXPRESS могут объявлять идентификаторы, видимые в области видимости объявления типа:

- перечисление;
- метка правила (правило области определения).

10.3.16 Метка типа

Видимость: метка типа является видимой в области видимости объекта и всех подтипов данного объекта, функции или процедуры, в которой она объявлена. Метка типа неявно объявляется при первом ее появлении в области видимости. В функциях и процедурах первое появление метки типа должно иметь место в спецификации формального параметра. На объявленную таким образом метку типа можно ссылаться где-либо в другом месте в спецификации формального параметра или в локальных объявлениях функции или процедуры. Если метка типа объявлена в функции, то на метку типа можно ссылаться в спецификации типа результата функции.

10.3.17 Переменная

Видимость: идентификатор переменной является видимым в области видимости функции, процедуры или правила, в которой он объявлен.

11 Спецификация интерфейсов

В данном разделе определены конструкции, позволяющие элементам, объявленным в одной схеме, быть видимыми в другой схеме. Существуют две спецификации интерфейсов (**USE** и **REFERENCE**), обеспечивающие видимость элементов. Спецификация интерфейса **USE** позволяет элементам, объявленным в одной схеме, получить независимую реализацию в схеме, в которой определена конструкция **USE**.

Экземпляр объекта считается независимым, если он не играет роль, предписанную атрибутом любого другого экземпляра объекта, то есть функция **ROLESOF** (см. 15.20), примененная к независимому экземпляру объекта, выдаст в качестве результата пустое множество. Тип данных объекта, объявленного в схеме локально или с использованием интерфейса **USE**, может быть реализован независимо или играть роль, предписанную атрибутом объекта в данной схеме.

Объект, объявленный посредством интерфейса **REFERENCE** или неявного интерфейса, должен быть реализован только для выполнения роли, предписанной атрибутом реализации объекта в схеме.

Синтаксис:

```
242 interface_specification = reference_clause | use_clause .
```

Внешним объявлением является любое объявление (например, объекта), присутствующее во внешней схеме (в любой схеме, отличной от данной схемы).

Другое различие между двумя формами интерфейса состоит в том, что интерфейс **USE** применим только к именованным типам данных (объектным типам данных и определенным типам данных), а интерфейс **REFERENCE** применим ко всем объявлениям, за исключением правил и схем.

Внешнему элементу языка EXPRESS в данной схеме может быть присвоено новое имя. На внешний элемент языка EXPRESS в данной схеме следует ссылаться по его новому имени, если оно задано после ключевого слова **AS**.

11.1 Спецификация интерфейса USE

Объектный или определенный тип данных, объявленный во внешней схеме, может быть сделан доступным в данной схеме посредством интерфейса **USE**. Спецификация интерфейса **USE** задает имя внешней схемы и, факультативно, объявленные в ней имена объектных или определенных типов данных. Если не указан элемент **named_types**, то все именованные типы данных, объявленные явно или посредством интерфейса **USE** во внешней схеме, трактуются как объявленные локально в данной схеме.

Синтаксис:

```
336 use_clause = USE FROM schema_ref [ '(' named_type_or_rename
                 { ',' named_type_or_rename } ')' ] ';' .
259 named_type_or_rename = named_types [ AS ( entity_id | type_id ) ] .
```

11.2 Спецификация интерфейса REFERENCE

Спецификация интерфейса **REFERENCE** позволяет сделать видимыми в данной схеме следующие элементы языка EXPRESS, объявленные во внешней схеме:

- константа;
- объект;
- функция;
- процедура;
- тип.

Спецификация интерфейса **REFERENCE** задает имя внешней схемы и, факультативно, объявленные в ней имена элементов языка EXPRESS. Если имена не заданы, то все элементы языка EXPRESS, объявленные явно или посредством интерфейса **USE** во внешней схеме, являются видимыми в данной схеме.

Синтаксис:

```
281 reference_clause = REFERENCE FROM schema_ref [ '(' resource_or_rename
                       { ',' resource_or_rename } ')' ] ';' .
288 resource_or_rename = resource_ref [ AS rename_id ] .
289 resource_ref = constant_ref | entity_ref | function_ref | procedure_ref | type_ref .
284 rename_id = constant_id | entity_id | function_id | procedure_id | type_id .
```

Внешние объявления, введенные посредством интерфейса **REFERENCE**, не считаются локальными объявлениями и поэтому не могут быть реализованы независимо, но могут быть реализованы для выполнения роли, предписанной атрибутом объекта в данной схеме.

11.3 Взаимодействие интерфейсов USE и REFERENCE

Если объектный или определенный тип данных в данной схеме одновременно указан в спецификациях интерфейсов **USE** и **REFERENCE**, то спецификация интерфейса **USE** имеет приоритет.

Пример — В следующем фрагменте **a1** ***трактуется как локальное объявление:***

```
USE FROM s1 (a1);
REFERENCE FROM s1 (a1);
```

Если именованный тип данных импортирован в данную схему посредством интерфейса **USE**, то данный именованный тип данных может быть импортирован другой схемой из данной схемы посредством интерфейсов **USE** или **REFERENCE** (это означает, что спецификации интерфейсов **USE** могут связывать схемы в цепочки).

Пример — Пусть заданы следующие объявления двух схем:

```
SCHEMA s1;
  ENTITY e1;
  END_ENTITY;
END SCHEMA;
SCHEMA s2;
USE FROM s1 (e1 AS e2);
END_SCHEMA;
```

тогда следующие спецификации эквивалентны:

```
SCHEMA s3;                    SCHEMA s3;
USE FROM s1 (e1 AS e2);       USE FROM s2 (e2);
END_SCHEMA;                   END_SCHEMA;
```

Поскольку элементы языка EXPRESS, импортированные посредством интерфейса **REFERENCE**, не трактуются как локально объявленные элементы, то связывание схем в цепочки посредством интерфейсов **REFERENCE** невозможно.

11.4 Импорт объектов посредством неявных интерфейсов

Внешнее объявление может ссылаться на идентификаторы, которые не являются видимыми в данной схеме. Такие элементы языка EXPRESS, на которые даны неявные ссылки, требуются для полного понимания данной схемы, но они не видимы для элементов языка EXPRESS, объявленных в данной схеме. Каждый импортированный неявно элемент может в свою очередь ссылаться на другие элементы языка EXPRESS, которые не являются видимыми в данной схеме; такие элементы языка EXPRESS также требуются для полного понимания данной схемы.

Пример — Неявно импортированные элементы и связывание неявных интерфейсов в цепочки.

```
SCHEMA s1;
  TYPE t1 = REAL;
  END_TYPE;
  ENTITY e1;
    a : t1;
  END_ENTITY;
  ENTITY e2;
    a1 : e1;
  END_ENTITY;
END_SCHEMA;
SCHEMA s2;
  REFERENCE FROM s1 (e2);
  ENTITY e3;
    a3 : e2;
  END_ENTITY;
END_SCHEMA;
```

Объект **e2** ***используется как тип данных атрибута*** **a3**. ***Поскольку в определении объекта*** **e2** ***требуется*** **e1**, ***то объект*** **e1** ***неявно импортируется схемой*** **s2**. ***Однако поскольку*** **e1** ***не был импортирован в схему*** **s2** ***в явной форме, то*** **e1** ***не может использоваться в схеме*** **s2**. ***Аналогично, в определении объекта*** **e1** ***требуется*** **t1**; ***поэтому*** **t1** ***неявно импортируется схемой*** **s2**.

В последующих пунктах термин «импортирован» используется для обозначения понятий «импортирован посредством интерфейса **USE**», «импортирован посредством интерфейса **REFERENCE**» или «неявно импортирован».

11.4.1 Импорт констант

При импорте константы неявно импортируются:

- любые определенные типы данных, использованные в объявлении импортируемой константы;
- любые объектные типы данных, использованные в объявлении импортируемой константы;
- любые константы, использованные в объявлении импортируемой константы;
- любые функции, использованные в объявлении импортируемой константы.

11.4.2 Импорт определенных типов данных

При импорте определенного типа данных неявно импортируются:

- любые определенные типы данных, использованные в объявлении импортируемого типа, включая наращиваемые определенные типы данных, которые данный импортируемый тип может расширять, используя ключевое слово **BASED_ON**, но исключая любой из выбираемых элементов, если импортируемым типом является тип данных **SELECT**, а также исключая те выбираемые элементы выбираемых типов данных, на которых может основываться данный импортируемый тип;
- любые константы или функции, использованные в объявлении представления импортируемого определенного типа данных;
- любые константы или функции, использованные в правилах области определения импортируемого определенного типа данных;
- любые определенные типы данных, представленные типом данных **SELECT**, список выбора которого содержит импортируемый определенный тип данных.

Пример — Неявный импорт определенного типа данных через тип данных **SELECT.**

```
SCHEMA s1;
   TYPE sel1 = SELECT (e1, t1);
   END_TYPE;
   TYPE t1 = INTEGER;
   END_TYPE;
   ENTITY e1;
   ...
   END_ENTITY;
END_SCHEMA;
   SCHEMA s2;
   REFERENCE FROM s1 (t1);
   END_SCHEMA;
```

Схема **s2** ***содержит явную ссылку на*** **t1**, ***а поскольку*** **sel1** ***представлен типом данных*** **SELECT**, ***содержащим*** **t1**, ***то на*** **sel1** ***делается неявная ссылка.***

11.4.3 Импорт объектных типов данных

При импорте объектного типа данных неявно импортируются:

- все объектные типы данных, являющиеся супертипами данного импортируемого объектного типа данных.

Примечание — Подтипы импортируемого объектного типа данных, независимо от того, присутствуют ли они в выражении **SUPERTYPE OF**, не будут импортированы неявно в результате установления данного интерфейса для импорта;

- все правила, ссылающиеся на импортируемый объектный тип данных и, возможно, на другие объектные типы данных, которые все явно или неявно импортированы в данную схему;
- все ограничения подтипов для импортируемого объектного типа данных;
- любые константы, определенные типы данных, объектные типы данных или функции, использованные в объявлении атрибутов импортируемого объектного типа данных;
- любые константы, определенные типы данных, объектные типы данных или функции, использованные в правилах области определения импортируемого объектного типа данных;
- любые определенные типы данных, представленные типом данных **SELECT**, содержащем импортируемый объектный тип данных в своем списке выбора.

В графах подтипов/супертипов могут быть отсечены ветви только в результате следования по связям, задаваемым ключевым словом **SUBTYPE OF**, при группировке неявных интерфейсов импортируемого

объектного типа данных. Алгоритм, используемый для вычисления допустимых реализаций усеченного графа подтипов/супертипов, приведен в приложении С.

11.4.4 Импорт функций

При импорте функций неявно импортируются:

- любые определенные типы данных или объектные типы данных, использованные в объявлении параметров для импортируемой функции;
- любые определенные типы данных или объектные типы данных, использованные в определении возвращаемого типа данных для импортируемой функции;
- любые определенные типы данных или объектные типы данных, использованные в объявлении локальных переменных в импортируемой функции;
- любые константы, функции или процедуры, использованные в импортируемой функции.

11.4.5 Импорт процедур

При импорте процедуры неявно импортируются:

- любые определенные типы данных или объектные типы данных, использованные в объявлении параметров для импортируемой процедуры;
- любые определенные типы данных или объектные типы данных, использованные в объявлении локальных переменных в импортируемой процедуре;
- все константы, функции или процедуры, использованные в импортируемой процедуре.

11.4.6 Импорт правил

При импорте правила неявно импортируются:

- любые определенные типы данных или объектные типы данных, использованные в объявлении локальных переменных в импортируемом правиле;
- все константы, функции или процедуры, использованные в импортируемом правиле.

11.4.7 Импорт ограничений подтипов

При импорте ограничения подтипов объекты не импортируются неявным образом.

Ограничения, заданные в импортируемом ограничении подтипов, преобразовываются после импортирования так, чтобы не допустить реализацию в данной схеме какого-либо сложного объектного типа данных, который не был разрешен в исходной внешней схеме (см. приложение С).

12 Выражения

Выражения являются комбинациями операторов, операндов и вызовов функций, которые вычисляются для получения некоторого значения.

Синтаксис:

```
216 expression = simple_expression [ rel_op_extended simple_expression ] .
283 rel_op_extended = rel_op | IN | LIKE .
282 rel_op = '<' | '>' | '<=' | '>=' | '<>' | '=' | ':<>:' | ':=:' .
305 simple_expression = term { add_like_op term } .
325 term = factor { multiplication_like_op factor } .
217 factor = simple_factor [ '**' simple_factor ] .
306 simple_factor = aggregate_initializer | entity_constructor |
                    enumeration_reference | interval | query_expression |
                    ( [ unary_op ] ( '(' expression ')' | primary ) ) .
331 unary_op = '+' | '-' | NOT .
269 primary = literal | ( qualifiable_factor { qualifier } ) .
257 multiplication_like_op = '*' | '/' | DIV | MOD | AND | '||' .
168 add_like_op = '+' | '-' | OR | XOR .
```

Некоторым операторам требуется один операнд, а другим — два операнда. Оператор, которому требуется только один операнд, должен располагаться перед своим операндом. Оператор, которому требуются два операнда, должен располагаться между своими операндами. В данном разделе определены операторы и установлены типы данных операндов, которые могут использоваться каждым оператором.

Существует семь классов операторов:

а) Арифметические операторы принимают числовые операнды и выдают числовые результаты. Тип данных значения результата арифметического оператора зависит от оператора и типов данных операндов (см. 12.1).

b) Операторы отношения принимают в качестве операндов разные типы данных и выдают результаты, имеющие тип данных **LOGICAL** (**TRUE**, **FALSE** или **UNKNOWN**).

c) Двоичные операторы принимают операнды типа данных **BINARY** и выдают результаты, имеющие тоже тип данных **BINARY**.

d) Логические операторы принимают операнды типа данных **LOGICAL** и выдают результаты, имеющие тоже тип данных **LOGICAL**.

e) Строковые операторы принимают операнды типа данных **STRING** и выдают результаты, имеющие тоже тип данных **STRING**.

f) Агрегированные операторы комбинируют агрегированные значения с другими агрегированными значениями или с отдельными элементами разными способами и выдают результаты агрегированного типа.

g) Ссылочные и индексные операторы извлекают компоненты из экземпляров объектов и агрегированных значений.

Вычисление выражений осуществляется в соответствии с приоритетом входящих в выражение операторов.

Значение выражения, заключенного в круглые скобки, вычисляется до того, как оно будет трактоваться как единый операнд.

Процесс вычисления осуществляется слева направо, при этом операторы с более высоким приоритетом вычисляются первыми. Правила приоритетов для всех операторов языка EXPRESS установлены в таблице 10. Операторы в одной строке имеют одинаковый приоритет, а строки упорядочены по уменьшению приоритета.

Операнд, расположенный между двумя операторами, имеющими разные приоритеты, относится к оператору, имеющему более высокий приоритет. Операнд, расположенный между двумя операторами с одинаковым приоритетом, относится к оператору, расположенному слева.

Таблица 10 — Приоритет операторов

Приоритет	Описание	Операторы
1	Ссылки на элементы	**[ ] . **
2	Унарные операторы	**+ – NOT**
3	Возведение в степень	**
4	Умножение/деление	*** / DIV MOD AND ||**
5	Сложение/вычитание	**– + OR XOR**
6	Отношение	**= <> <= >= < > :=: :<>: IN LIKE**
Примечание — **||** является оператором построения сложного объекта.		

Пример — Выражение **–10**2** ***вычисляется как*** **(–10)**2,** ***давая в результате значение*** **100. *Выражение*** **10/20*30** ***вычисляется как*** **(10/20)*30,** ***давая в результат значение*** **15.0.**

12.1 Арифметические операторы

Арифметическими операторами, которым требуется один операнд, являются тождество (**+**) и отрицание (**–**). Операнд должен иметь числовой тип (**NUMBER**, **INTEGER** или **REAL**). Результат оператора (**+**) равен операнду, результат оператора (**–**) имеет знак, противоположный знаку операнда. Если операнд имеет неопределенное (**?**) значение, то результат также будет иметь неопределенное (**?**) значение для обоих операторов.

Арифметическими операторами, которым требуются два операнда, являются сложение (**+**), вычитание (**–**), умножение (*****), деление (**/**), возведение в степень (******), целочисленное деление (**DIV**) и деление по модулю (**MOD**). Операнды должны иметь числовой тип (**NUMBER**, **INTEGER** или **REAL**).

Операторы сложения, вычитания, умножения, деления и возведения в степень выполняют одноименные математические операции. За исключением деления, они выдают целочисленный результат, если оба операнда имеют тип данных **INTEGER**, и результат типа **REAL** — в остальных случаях [если при этом ни один из операндов не имеет неопределенного (**?**) значения]. Результатом оператора деления (**/**) является действительное число [если при этом ни один из операндов не имеет неопределенного (**?**) значения].

Деление по модулю (**MOD**) и целочисленное деление (**DIV**) дают целочисленный результат [если при этом ни один из операндов не имеет неопределенного (**?**) значения]. Если какой-либо операнд имеет тип данных **REAL**, то перед выполнением данного оператора его значение преобразуется в значение типа **INTEGER** усечением, то есть его дробная часть отбрасывается. Для любых целых чисел **a** и **b** всегда справедливо равенство **(a DIV b)*b + c*(a MOD b) = a**, где **c =1** для **b>=0** и **c = –1** для **b<0**. Абсолютное значение выражения **a MOD b** должно быть меньше, чем абсолютное значение **b**, а знак выражения **a MOD b** должен совпадать со знаком **b**.

Если какой-либо из операндов арифметического оператора имеет неопределенное (**?**) значение, то результат оператора должен иметь неопределенное (**?**) значение.

Округление действительных чисел.

Когда требуется округление, оно осуществляется с точностью *p* (либо заданной в явном виде для типа данных **REAL**, либо определяемой ограничением для конкретной реализации, устанавливаемым в соответствии с приложением E) по следующему алгоритму:

a) преобразовать представление числа в экспоненциальный формат с удалением всех предшествующих нулей;

b) установить указатель разряда *k* на *p*-ю позицию справа от десятичной точки;

c) если действительное число является положительным, то выполняются следующие действия:

- если цифра, расположенная на позиции *k*, принадлежит к диапазону от 5 до 9, то добавить 1 к цифре, расположенной на позиции *k*–1, а цифры, начиная с позиции *k* и далее, отбросить. Перейти к шагу по перечислению e);
- если цифра, расположенная на позиции *k*, принадлежит к диапазону от 0 до 4, то цифры, начиная с позиции *k* и далее, отбросить. Перейти к шагу по перечислению h);

d) если действительное число является отрицательным, то выполняются следующие действия:

- если цифра, расположенная на позиции *k*, принадлежит к диапазону от 6 до 9, то добавить 1 к цифре, расположенной на позиции *k*–1, а цифры, начиная с позиции *k* и далее, отбросить. Перейти к шагу по перечислению e);
- если цифра, расположенная на позиции *k*, принадлежит к диапазону от 0 до 5, то цифры, начиная с позиции *k* и далее, отбросить. Перейти к шагу по перечислению h);

e) присвоить указателю разряда *k* значение *k*–1;

f) если цифра, расположенная на позиции *k*, принадлежит к диапазону от 0 до 9, то перейти к шагу по перечислению h);

g) если цифра, расположенная на позиции *k*, имеет значение 10, то добавить 1 к цифре, расположенной на позиции *k*–1, и установить цифру, расположенную на позиции *k*, в 0. Перейти к шагу по перечислению e);

h) округление действительного числа завершено.

Примечание — В результате действия данного алгоритма число 0,5 округляется до 1, а число –0,5 округляется до 0.

Пример – Данный пример показывает результат задания числа значащих цифр в дробной части действительного числа, то есть его точности.

```
LOCAL
   distance   : REAL(6);
   x1, y1, z1 : REAL;
   x2, y2, z2 : REAL;
END_LOCAL;
...
x1 := 0.;  y1 := 0.;  z1 := 0.;
x2 := 10.;  y2 := 11.;  z2 := 12.;
...
distance := SQRT( (x2–x1)**2 + (y2–y1)**2 + (z2–z1)**2);
```

Вычисленное значение объекта* distance *равно* 1.9104973 . . . e+1, *но, его реальным значением будет* 1.91050e+1, *поскольку в спецификации данного объекта задана точность, равная шести значащим цифрам, поэтому будут оставлены только шесть значащих цифр.

12.2 Операторы отношений

К операторам отношений относятся операторы сравнения значений, сравнения экземпляров, принадлежности (**IN**) и сопоставления строк (**LIKE**). Результатом вычисления выражения отношения является значение типа **LOGICAL** (**TRUE**, **FALSE** или **UNKNOWN**). Если хотя бы один из операндов имеет неопределенное (**?**) значение, то выражению присваивается значение **UNKNOWN**.

12.2.1 Операторы сравнения значений

К операторам сравнения значений относятся:

- равно (**=**);
- не равно (**<>**);
- больше чем (**>**);
- меньше чем (**<**);
- больше или равно (**>=**);
- меньше или равно (**<=**).

Данные операторы могут применяться к числовым, логическим, строковым и двоичным операндам. Кроме того, данные операторы могут применяться к элементам перечислений, объявленным в перечислениях, не являющихся наращиваемыми перечислениями и не основанными на наращиваемых перечислениях. Помимо этого, операторы **=** и **<>** могут применяться к значениям агрегированного и объектного типов данных и к элементам перечислений, объявленным в наращиваемых перечислениях или в перечислениях, основанных на наращиваемых перечислениях (см. 12.11).

Для двух заданных значений **a** и **b** выражения **a <> b** и **NOT (a = b)** эквивалентны для всех типов данных. Кроме того, если **a** и **b** не являются агрегированными или объектными типами данных, то справедливы следующие утверждения:

- одно из следующих выражений имеет значение **TRUE**: **a < b**, **a = b** или **a > b**;
- выражение **a <= b** эквивалентно выражению **(a < b) OR (a = b)**;
- выражение **a >= b** эквивалентно выражению **(a > b) OR (a = b)**.

12.2.1.1 Сравнение чисел

Операторы сравнения значений, примененные к числовым операндам, должны соответствовать математическому упорядочению действительных чисел.

Примечание — При сравнении двух действительных чисел спецификация их точности не учитывается.

Пример — Пусть задано:

a : REAL(3) := 1.23
b : REAL(5) := 1.2300;

тогда значением выражения **a = b** ***будет*** **TRUE.**

12.2.1.2 Сравнение двоичных чисел

При сравнении двух двоичных чисел сравниваются биты, расположенные в одинаковых позициях каждого числа, начиная с первой (самой левой) пары битов, затем — биты во второй позиции и так далее до тех пор, пока не встретится пара несовпадающих битов или не будут проверены все пары. Если встретилась пара несовпадающих битов, то меньшим считается двоичное число, бит которого равен 0. Никакого дополнительного сравнения не требуется. Если пара несовпадающих битов не встретилась, то меньшим считается более короткое двоичное число (длина двоичного числа определяется с помощью функции **BLENGTH**). Если сравниваемые двоичные числа имеют одинаковую длину и все пары их битов совпадают, то данные двоичные числа равны.

12.2.1.3 Сравнение логических значений

При сравнении двух значений типа **LOGICAL** (или **BOOLEAN**) должен соблюдаться следующий порядок значений:

FALSE < UNKNOWN < TRUE.

12.2.1.4 Сравнение строковых значений

При сравнении двух строковых значений сравниваются символы, расположенные в одинаковых позициях каждого строкового значения, начиная с первой (самой левой) пары символов, затем — символы, расположенные во второй позиции, и так далее до тех пор, пока не встретится пара несовпадающих символов или не будут проверены все пары символов. Если встретилась пара несовпадающих символов, то меньшим считается строковое значение, содержащее символ с меньшим значением кода (в соответствии с определением значений октетов для символов по ИСО/МЭК 10646). Никакого дополнительного сравнения не требуется. Если пара несовпадающих символов не встретилась, то меньшим считается более короткое строковое значение (длина строкового значения определяется с помощью функции **LENGTH**). Если сравниваемые строковые значения имеют одинаковую длину, и все пары их символов совпадают, то данные строковые значения равны.

12.2.1.5 Сравнение элементов перечисления

Сравнение значений элементов перечисления, которое не является наращиваемым и не основано на наращиваемом перечислении, основано на их относительных позициях в объявлении перечисляемого типа данных. См. правило по перечислению d) в 8.4.1.

Для значений, тип данных которых является наращиваемым перечисляемым типом или перечисляемым типом, основанном на наращиваемом перечисляемом типе, определено только сравнение на равенство или неравенство. Два таких значения равны, если они представляют один и тот же элемент перечисления, и не равны — в противном случае.

12.2.1.6 Сравнение агрегированных значений

Операторами сравнения значений, установленными для агрегированных значений, являются операторы «равно» (**=**) и «не равно» (**<>**). Два агрегированных значения могут сравниваться только в том случае, если их типы данных совместимы (см. 12.11).

При сравнении агрегированных структур должно проверяться число элементов в каждом из операндов: если справедливо выражение **SIZEOF (a) <> SIZEOF (b)**, то агрегированные структуры не равны. При сравнении агрегированных структур сравниваются элементы агрегированного значения путем сравнения значений. Если результатом какого-либо сравнения элементов является **FALSE**, то результатом сравнения агрегированных структур является также **FALSE**. Если результатом одного или нескольких сравнений элементов при сравнении агрегированных структур является **UNKNOWN**, а результатом остальных сравнений является **TRUE**, то результатом сравнения агрегированных структур является **UNKNOWN**. Во всех других случаях результатом сравнения агрегированных структур является **TRUE**.

Определение равенства агрегированных структур зависит от их агрегированных типов данных:

- два массива **a** и **b** равны тогда и только тогда, когда значение каждого элемента из **a** равно значению элемента из **b**, расположенного в той же позиции, то есть **a[i] = b[i]** (см. 12.6.1);

- два списка **a** и **b** равны тогда и только тогда, когда значение каждого элемента из **a** равно значению элемента из **b**, расположенного в той же позиции;

- два пакета или набора **a** и **b** равны тогда и только тогда, когда каждый элемент **VALUE_IN a** встречается в **VALUE_IN b** равное число раз, а каждый элемент **VALUE_IN b** также встречается в **VALUE_IN a** равное число раз.

12.2.1.7 Сравнение значений объектных типов данных

Значения двух экземпляров объектного типа данных являются равными, если равны значения их соответствующих атрибутов. Поскольку экземпляры объектного типа данных могут иметь атрибуты, представленные также объектными типами данных, для таких экземпляров существует возможность ссылаться на самих себя. При этом значения экземпляров объектного типа данных являются равными, если все атрибуты, представленные простыми типами данных, имеют одинаковые значения, и одни и те же атрибуты в обоих экземплярах объектного типа данных ссылаются сами на себя.

Для более точного определения предположим, что необходимо сравнить два экземпляра ***ℓ*** и **r**. Если ***ℓ*** **:=: r**, то ***ℓ*** **= r**. Иначе вводят следующие определения:

- определют упорядочение на совокупности рассматриваемых экземпляров. На практике такая совокупность конечна, поэтому упорядочение может быть осуществлено;

- для целей данного рассмотрения определяют оператор индексирования агрегированной структуры, соблюдающий данное упорядочение так, чтобы для любой агрегированной структуры **agg** и для любых индексов **i** и **j** условие **i < j** было эквивалентно условию **agg[i] < agg[j]**;

- определяют ссылочный путь как последовательность одной или нескольких ссылок на атрибуты или индексы. Применение ссылочного пути **s** к экземпляру **i** будет записано как **s(i)**. Тогда **s(i)** является вычисляемым, если ни одна из ссылок, за исключением последней, не приводит к неопределенности (**?**).

Тогда значение выражения ***ℓ*** **= r** определяют по первому выполненному из следующих условий:

a) если **TYPEOF(*ℓ*)<>TYPEOF(r)**, то ***ℓ*** **= r** имеет значение **FALSE**;

b) если существует такой ссылочный путь **s**, что только один из **s(*ℓ*)** и **s(r)** является вычисляемым, то ***ℓ*** **= r** имеет значение **FALSE**;

c) если существует такой ссылочный путь **s**, что результатами как **s(*ℓ*)**, так и **s(r)** являются значения простого типа данных, и если **s(*ℓ*)<>s(r)**, то ***ℓ*** **= r** имеет значение **FALSE**;

d) если существует такой ссылочный путь **s**, что результатами как **s(*ℓ*)**, так и **s(r)** являются либо значения объектного типа данных, либо объявляется, что они должны быть выбираемого типа данных, и если **TYPEOF(s(*ℓ*))<>TYPEOF(s(r))**, то ***ℓ*** **= r** имеет значение **FALSE**;

e) если существует такой ссылочный путь **s**, что **NOT EXISTS(s(*ℓ*))** или **NOT EXISTS(s(r))**, то ***ℓ*** **= r** имеет значение **UNKNOWN**;

f) в ином случае ***ℓ*** **= r** имеет значение **TRUE**.

Примеры

1 Представленный ниже алгоритм является одной из возможных реализаций описанной выше проверки сравнения значений. Данный алгоритм приведен для иллюстрации и не предназначен для представления какой-либо конкретной реализации.

Пусть в данном алгоритме ℓ и **r** ***являются переменными типа*** **GENERIC:**

a) инициализируем ℓ в качестве экземпляра объекта левой части, а **r** ***в качестве экземпляра объекта правой части;***

b) если ℓ и **r** ***представляют один и тот же экземпляр, то есть ℓ*** **:=: r,** ***то выражение имеет значение*** **TRUE;**

c) инициализируем пустой список **plist,** ***который будет содержать упорядоченные пары идентификаторов экземпляров объектного типа данных.***

Примечание — Представление идентификаторов экземпляров определяется реализацией;

d) сравним ℓ и **r,** ***используя определенный ниже алгоритм глубинного равенства;***

e) выражение будет иметь значение, возвращаемое алгоритмом глубинного равенства,

Алгоритм глубинного равенства:

a) если ℓ, **r** ***или оба экземпляра имеют неопределенное*** **(?)** ***значение, то алгоритм возвращает значение*** **UNKNOWN;**

b) если **TYPEOF(*ℓ*)<>TYPEOF(r),** ***то алгоритм возвращает значение*** **FALSE;**

c) если ℓ и **r** ***не являются экземплярами объектного типа данных, то алгоритм возвращает результат выражения ℓ=r, используя соответствующую проверку равенства;***

d) если ℓ и **r** ***представляют один и тот же экземпляр объектного типа данных, то есть ℓ*** **:=: r,** ***то алгоритм возвращает значение*** **TRUE;**

e) если пара экземпляров (ℓ, **r)** ***присутствует в списке*** **plist,** ***то алгоритм возвращает значение*** **TRUE;**

f) если пара экземпляров (ℓ, **r)** ***не присутствует в списке*** **plist,** ***то выполняются следующие действия:***

1) в список **plist** ***добавляется пара (ℓ,*** **r),**

2) для каждого атрибута **a,** ***определенного для ℓ и*** **r,** ***сравниваются ℓ.a и*** **r.a** ***с использованием алгоритма глубинного равенства, полагая, что ℓ=l,a и*** **r=r.a.**

Примечание — При этом используется рекурсивный вызов,

3) если результатом алгоритма глубинного равенства для какого-либо из атрибутов на шаге по пункту 2) перечисления e) будет **FALSE,** ***то в целом алгоритм также возвращает значение*** **FALSE.** ***В противном случае, если алгоритм возвращает значение*** **UNKNOWN** ***для какого-либо из атрибутов, то общим результатом также будет значение*** **UNKNOWN.** ***Иначе, алгоритм возвращает значение*** **TRUE.**

Примечание — Данный алгоритм обеспечивает, что если хотя бы один из результатов сравнения имеет значение **FALSE**, то общий результат имеет значение **FALSE**. Если все результаты сравнения имеют значение **TRUE**, то общий результат также имеет значение **TRUE**. Если результат какого-либо сравнения имеет значение **UNKNOWN**, а все другие результаты имеют значение **TRUE**, то общий результат имеет значение **UNKNOWN**.

2 Локальные переменные **i1** ***и*** **i2** ***имеют тип*** **loop_of_integer;** ***при выполнении указанных в данном примере операторов присваивания, значения данных переменных не равны.***

```
ENTITY loop_of_integer;
   int    : INTEGER;
   next   : loop_of_integer ;
END ENTITY;
. . .
LOCAL
   i1, i2 : loop_of_integer ;
END LOCAL;
. . .
i1 := loop_of_integer(5,loop_of_integer(3,SELF));
i2 := loop_of_integer(3,loop_of_integer(5,SELF));
IF i1 = i2 THEN  -- результатом сравнения является FALSE
```

Сравнение значений объектов может быть применено к экземплярам объектов и экземплярам сгруппированных объектов (см. 12.7.4). Для экземпляров объектов должны сравниваться атрибуты всех

подтипов и супертипов сравниваемых экземпляров. Для экземпляров сгруппированных объектов должны сравниваться только атрибуты, объявленные как атрибуты в объявлении объектного типа данных, указанного в квалификаторе группы (это не относится к унаследованным атрибутам, которые повторно объявлены в указанном объектном типе данных).

12.2.2 Операторы сравнения экземпляров

Операторами сравнения экземпляров являются:

- равенство экземпляров (**:=:**);
- неравенство экземпляров (**:<>:**).

Данные операторы могут применяться к операндам числового, логического, строкового, двоичного, перечисляемого, агрегированного и объектного типов данных. Оба операнда оператора сравнения экземпляров должны быть совместимы по типу данных (см. 12.11).

Для двух заданных операндов **a** и **b**, выражение **(a:<>:b)** эквивалентно выражению **NOT (a:=:b)** для всех типов данных.

Применение операторов сравнения экземпляров к числовым, логическим, строковым, двоичным и перечисляемым типам данных эквивалентно применению соответствующих операторов сравнения значений. То есть **(a:=:b)** эквивалентно **(a=b)**, а **(a:<>:b)** эквивалентно **(a<>b)** для указанных типов данных.

12.2.2.1 Сравнение экземпляров агрегированных структур

Операторами сравнения экземпляров, определенными для значений агрегированных структур, являются операторы равенства (**:=:**) и неравенства (**:<>:**). Два значения агрегированных структур могут сравниваться только в том случае, если совместимы их типы данных (см. 12.11).

Все сравнения агрегированных структур должны проверять число элементов в каждом из операндов: если **SIZEOF(a)<>SIZEOF(b)**, то агрегированные структуры не равны. При сравнении агрегированных структур проводится сравнение элементов значений агрегированных структур посредством сравнения экземпляров. Если результатом сравнения каких-либо элементов является значение **FALSE**, то результат сравнения агрегированных структур также будет иметь значение **FALSE**. Если результатом одного или нескольких сравнений элементов для данного сравнения агрегированных структур является значение **UNKNOWN**, а результатом всех остальных сравнений является значение **TRUE**, то результатом сравнения агрегированных структур будет значение **UNKNOWN**. В противном случае результат сравнения агрегированных структур будет иметь значение **TRUE**.

Определение равенства экземпляров агрегированных структур зависит от сравниваемых агрегированных типов данных:

- два массива **a** и **b** равны тогда и только тогда, когда каждый элемент массива **a** представлен тем же экземпляром, что и элемент массива **b**, расположенный на той же позиции, то есть **a[i] :=: b[i]** (см. 12.6.1);
- два списка **a** и **b** равны тогда и только тогда, когда каждый элемент списка **a** представлен тем же экземпляром, что и элемент списка **b**, расположенный на той же позиции;
- экземпляры двух пакетов **a** и **b** равны тогда и только тогда, когда каждый элемент из пакета **a** присутствует такое же число раз в пакете **b**, а каждый элемент из пакета **b** также присутствует такое же число раз в пакете **a**;
- экземпляры двух наборов **a** и **b** равны тогда и только тогда, когда каждый элемент из набора **a** присутствует в наборе **b**, а каждый элемент из набора **b** присутствует в наборе **a**;
- экземпляр пакета равен экземпляру набора тогда и только тогда, когда каждый элемент из набора присутствует в пакете только один раз, а пакет не содержит элементов, которых нет в наборе.

Пример — Сравнение экземпляров двух массивов:

```
LOCAL
   a1, a2 : ARRAY [1:10] OF b;
END LOCAL;
...
IF (a1 :=: a2) THEN ...
```

12.2.2.2 Сравнение экземпляров объектного типа данных

Операторы равенства (**:=:**) и неравенства (**:<>:**) экземпляров объектного типа данных сравнивают два совместимых экземпляра объектного типа данных и выдают результат типа **LOGICAL**.

Результатом сравнения **a:=:b** является значение **TRUE**, если **a** представлен тем же экземпляром объектного типа данных, что и **b**, то есть их зависящие от реализации идентификаторы одинаковы. Результат сравнения будет иметь значение **FALSE**, если **a** представлен другим экземпляром объектного типа данных, чем **b**. Результат сравнения будет иметь значение **UNKNOWN**, если хотя бы один из операндов имеет неопределенное (**?**) значение.

Если не оговорено иное, то сравнение экземпляров объектного типа данных должно использоваться для сравнения двух экземпляров объектного типа данных, например, при сравнении агрегированных структур и проверке правила уникальности **UNIQUE**.

Пример — Все дети имеют матерей, но некоторые дети могут иметь братьев или сестер. Это моделируется следующим образом:

```
ENTITY child
SUBTYPE OF (person);
   mother : female;  -- мы не рассматриваем более одного поколения
   father  : male;
END ENTITY;
ENTITY sibling
SUBTYPE OF (child);
   siblings : SET [1:?] sibling;
WHERE
                -- установим, что текущий экземпляр не является
                -- одним из своих братьев или сестер
not_identical : SIZEOF ( QUERY ( i <* siblings | i :=: SELF ) ) = 0;
                -- установим, что каждый из братьев или сестер
                -- имеет общего отца или мать с текущим экземпляром
same_parent : SIZEOF (QUERY ( i <* siblings |
                                  ( i.mother :=: SELF.mother ) OR
                                  ( i.father :=: SELF.father ) ) =
                SIZEOF ( siblings ) );
END_ENTITY;
```

12.2.3 Оператор принадлежности

Оператор принадлежности **IN** осуществляет проверку, принадлежит ли данный элемент к какой-либо агрегированной структуре и возвращает ли результат типа **LOGICAL**. Операнд, расположенный справа от оператора, должен иметь значение агрегированного типа данных, а операнд, расположенный слева, должен быть совместим с базисным типом данного значения агрегированного типа данных. Результат выражения **e IN agg** определяется следующим образом:

a) если любой из операндов имеет неопределенное (**?**) значение, то выражение имеет значение **UNKNOWN**;

b) если существует такой элемент **agg[i]**, для которого **e:=:agg[i]**, то выражение имеет значение **TRUE**;

c) если существует элемент **agg[i]**, имеющий неопределенное (**?**) значение, то выражение имеет значение **UNKNOWN**;

d) в противном случае, выражение имеет значение **FALSE**.

Примечание — Для того, чтобы проверить, существует ли в агрегированной структуре элемент, имеющий конкретное значение, может быть использована функция **VALUE_IN** (см. 15.28).

Проверка принадлежности, определенной разработчиком модели, может быть осуществлена посредством пары функций, названных для примера **my_equal** (см. примечание в 8.2.5) и **my_in** в следующем псевдокоде:

```
FUNCTION my_in(c:AGGREGATE OF GENERIC:gen; v:GENERIC:gen) : LOGICAL;
   (*"my_in" возвращает значение UNKNOWN, если v или с имеет неопределенное (?) значение, иначе,
     возвращает значение TRUE, если любой элемент из с имеет 'значение' v, иначе, возвращает
     значение UNKNOWN, если результатом любого сравнения является UNKNOWN, иначе возвращает
     значение FALSE *)
LOCAL
   result      : LOGICAL;
   unknownp  : BOOLEAN := FALSE;
END_LOCAL
IF ( (NOT EXISTS (v) ) OR (NOT EXISTS (c) ) THEN
   RETURN (UNKNOWN); END_IF;
REPEAT i  :=  LOINDEX(c) TO HIINDEX(c);
   result   :=  my_equal(v, c[i]);
   IF (result = TRUE) THEN
     RETURN (result); END_IF;
   IF (result = UNKNOWN) THEN
     unknownp := TRUE; END_IF;
END_REPEAT;
```

```
IF (unknownp) THEN
   RETURN (UNKNOWN);
ELSE
   RETURN(FALSE);
END_IF;
END_FUNCTION;
```

Это может быть использовано, например, следующим образом:

```
LOCAL
   v : a;
   c : SET OF a;
END_LOCAL;
. . .
IF my_in(c, v) THEN . . .
```

12.2.4 Интервальные выражения

Интервальное выражение проверяет, находится ли значение в заданном интервале. Выражение содержит три операнда, которые должны быть совместимыми (см. 12.11). Операнды должны принадлежать к типу данных, имеющему установленное упорядочение, то есть к простым типам (см. 8.1) и определенным типам данных, базисными типами которых являются простые либо перечисляемые типы данных.

Синтаксис:

```
243 interval = '{' interval_low interval_op interval_item interval_op interval_high '}'.
246 interval_low = simple_expression .
247 interval_op = '<' | '<='.
245 interval_item = simple_expression .
244 interval_high = simple_expression .
```

Примечание — Интервальное выражение:
{ interval_low interval_op interval_item interval_op interval_high }
семантически эквивалентно следующему:
(interval_low interval_op interval_item) AND
(interval_item interval_op interval_high)
Предполагается, что во втором выражении **interval_item** вычисляется только один раз.

Результатом интервального выражения является значение типа **LOGICAL**, которое имеет значение **TRUE**, если результатом обоих операторов отношения является **TRUE**. Интервальное выражение имеет значение **FALSE**, если результатом любого из операторов отношения является **FALSE**, и значение **UNKNOWN**, если какой-либо из операндов имеет неопределенное (**?**) значение.

Пример — В данном примере проверяется, имеет ли* b *значение большее, чем* 5.0, *и меньшее или равное* 100.0*:

```
LOCAL
   b : REAL := 20.0;
END_LOCAL;
. . .
IF { 5.0 < b <= 100.0 } THEN -- результатом является TRUE
. . .
```

12.2.5 Оператор сопоставления строк

Оператор сопоставления строк **LIKE** сравнивает два строковых значения, используя описанный ниже алгоритм сопоставления с образцом. Результатом оператора **LIKE** является значение типа **LOGICAL**. Операнд, расположенный слева от оператора, представляет исследуемую строку. Операнд, расположенный справа от оператора, является эталонной строкой.

Алгоритм сопоставления с образцом определяется следующим образом. Каждый символ эталонной строки сравнивается с соответствующим символом (символами) исследуемой строки. Если любая пара соответствующих символов не совпадает, то сопоставление дало отрицательный результат и результатом выражения является значение **FALSE**.

Некоторые специальные символы в эталонной строке могут сопоставляться с несколькими символами в исследуемой строке. Данные символы определены в таблице 11. Для того, чтобы результатом выражения было значение **TRUE**, все соответствующие символы должны быть идентичными или совпадать (в соответствии с таблицей 11). Если какой-либо из операндов имеет неопределенное (**?**) значение, то выражение имеет значение **UNKNOWN**.

Если какой-либо из специальных символов сопоставления с образцом сам подлежит сопоставлению, то образец должен содержать эталонную управляющую последовательность. Эталонная управляющая последовательность должна содержать символ начала управляющей последовательности (****), за которым следует специальный символ, подлежащий сравнению.

Пример — Для сопоставления с символом @ используется управляющая последовательность \@.

В приведенных ниже примерах показаны разные символы, сравниваемые с образцом.

Примеры

1 Если **a := '\AAAA'**, ***то справедливо следующее:***

```
a LIKE '\\AAAA'  --> TRUE
a LIKE '\AAAA'   --> FALSE
a LIKE '\\A?AA'  --> TRUE
a LIKE '\\!\\AAA' --> TRUE
a LIKE '\\&'     --> TRUE
a LIKE '\$'      --> FALSE
```

2 Если **a := 'The quick red fox';** ***то справедливо следующее:***

```
a LIKE '$$$$'    --> TRUE
```

3 Если **a := 'Page 407';** ***то справедливо следующее:***

```
a LIKE '$*'      --> TRUE
```

Примечание — Символ отрицания (**!**) может использоваться перед любым символом, а не только перед символами сопоставления с образцом, чтобы сопоставить любой символ, отличающийся от данного символа.

Таблица 11 — Символы сопоставления с образцом

Символ	Значение
@	Сопоставляет любую букву
^	Сопоставляет любую букву на верхнем регистре
?	Сопоставляет любой символ
&	Сопоставляет остальную часть строки
#	Сопоставляет любую цифру
$	Сопоставляет подстроку, заканчивающуюся символом пробела или конца строки
*	Сопоставляет любое число символов
\	Начинает эталонную управляющую последовательность
!	Символ отрицания (используется с другими символами)

12.3 Двоичные операторы

Помимо операторов отношений, определенных в 12.2.1.2, для типа данных **BINARY** определены еще два оператора — индексирования (**[ ]**) и конкатенации (**+**).

12.3.1 Индексирование двоичных чисел

Оператор индексирования двоичных чисел принимает два операнда — индексируемое двоичное число и спецификацию индексов, а его результатом является двоичное число длиной, определяемой выражением **(index_2 – index_1 + 1)**. Полученное в качестве результата двоичное число эквивалентно последовательности битов, расположенных в индексируемом двоичном числе на позициях от **index_1** до **index_2** включительно. Если требуется двоичное число единичной длины, то необходимо указать только **index_1**. Значение индекса, равное 1, указывает на самый левый бит индексируемого двоичного числа.

Синтаксис:

```
239 index_qualifier = '[' index_1 ':' index_2 ] ']' .
237 index_1 = index .
236 index = numeric_expression .
238 index_2 = index .
```

Правила и ограничения:

a) Параметр **index_1** должен быть представлен положительным целым числом или неопределенным (**?**) значением.

b) Должно выполняться условие **1≤index_1≤BLENGTH(двоичное число)**, в противном случае, будет возвращено неопределенное (**?**) значение.

c) Если параметр **index_2** задан, то он должен быть представлен положительным целым числом или неопределенным (**?**) значением.

d) Должно выполняться условие **index_1≤index_2≤BLENGTH(двоичное число)**, в противном случае будет возвращено неопределенное (**?**) значение.

e) Если **index_1** или **index_2** имеет неопределенное (**?**) значение, то результатом также будет неопределенное (**?**) значение.

f) Если индексируемое выражение имеет неопределенное (**?**) значение, то результатом также будет неопределенное (**?**) значение.

Примеры

1 Четвертый бит двоичной переменной **image** ***может быть проверен следующим образом:***

```
image := %01010101
IF image[4] = %1 THEN ... -- результатом является TRUE
IF image[4 : 4] = %1 THEN ... -- эквивалентное выражение
```

2 Биты с четвертого по десятый двоичной переменной **image** ***могут быть проверены следующим образом:***

```
IF image[4:10] = %1011110 THEN ...
```

12.3.2 Оператор двоичной конкатенации

Оператор двоичной конкатенации (**+**) является двоичным оператором, который последовательно соединяет два двоичных числа. Оба операнда должны быть двоичными числами, а результатом выполнения оператора является двоичное число, содержащее конкатенацию двух операндов, при этом первый операнд расположен слева. Если какой-либо из операндов имеет неопределенное (**?**) значение, то результатом будет также иметь неопределенное (**?**) значение.

Пример — Двоичные числа могут быть соединены следующим образом:

```
image := %101000101 + %101001 ;
(* переменная image теперь содержит двоичное число %101000101101001 *)
```

12.4 Логические операторы

К логическими операторами относятся **NOT**, **AND**, **OR** и **XOR**. Результатом каждого из данных операторов является логическое значение. Операторам **AND**, **OR** и **XOR** требуются два логических операнда, а оператору **NOT** — один логический операнд.

12.4.1 Оператор NOT

Оператору **NOT** требуется один логический операнд, помещаемый справа от оператора **NOT**. Результатом является логическое значение, формируемое в соответствии с таблицей 12.

Таблица 12 — Оператор **NOT**

Операнд	Результат
TRUE	**FALSE**
UNKNOWN	**UNKNOWN**
FALSE	**TRUE**

12.4.2 Оператор AND

Оператору **AND** требуются два логических операнда, а результатом является логическое значение, формируемое в соответствии с таблицей 13. Оператор **AND** является коммутативным.

Таблица 13 — Оператор **AND**

Операнд 1	Операнд 2	Результат
TRUE	**TRUE**	**TRUE**
TRUE	**UNKNOWN**	**UNKNOWN**
TRUE	**FALSE**	**FALSE**
UNKNOWN	**TRUE**	**UNKNOWN**
UNKNOWN	**UNKNOWN**	**UNKNOWN**
UNKNOWN	**FALSE**	**FALSE**
FALSE	**TRUE**	**FALSE**
FALSE	**UNKNOWN**	**FALSE**
FALSE	**FALSE**	**FALSE**

12.4.3 Оператор OR

Оператору **OR** требуются два логических операнда, а результатом является логическое значение, формируемое в соответствии с таблицей 14. Оператор **OR** является коммутативным.

Таблица 14 — Оператор **OR**

Операнд 1	Операнд 2	Результат
TRUE	**TRUE**	**TRUE**
TRUE	**UNKNOWN**	**TRUE**
TRUE	**FALSE**	**TRUE**
UNKNOWN	**TRUE**	**TRUE**
UNKNOWN	**UNKNOWN**	**UNKNOWN**
UNKNOWN	**FALSE**	**UNKNOWN**
FALSE	**TRUE**	**TRUE**
FALSE	**UNKNOWN**	**UNKNOWN**
FALSE	**FALSE**	**FALSE**

12.4.4 Оператор XOR

Оператору **XOR** требуются два логических операнда, а результатом является логическое значение, формируемое в соответствии с таблицей 15. Оператор **XOR** является коммутативным.

Таблица 15 — Оператор **XOR**

Операнд 1	Операнд 2	Результат
TRUE	**TRUE**	**FALSE**
TRUE	**UNKNOWN**	**UNKNOWN**
TRUE	**FALSE**	**TRUE**
UNKNOWN	**TRUE**	**UNKNOWN**
UNKNOWN	**UNKNOWN**	**UNKNOWN**
UNKNOWN	**FALSE**	**UNKNOWN**
FALSE	**TRUE**	**TRUE**
FALSE	**UNKNOWN**	**UNKNOWN**
FALSE	**FALSE**	**FALSE**

12.5 Строковые операторы

Помимо операторов отношений, определенных в 12.2.1.4 и 12.2.5, для строкового типа данных определены еще два оператора — индексирования (**[ ]**) и конкатенации (**+**).

12.5.1 Индексирование строк

Оператор индексирования строк принимает два операнда — индексируемую строку и спецификацию индексов, а его результатом является строка с длиной, определяемой выражением **(index_2–index_1+1)**. Полученная в качестве результата строка эквивалентна последовательности символов, расположенных в индексируемой строке на позициях от **index_1** до **index_2** включительно. Если требуется строка единичной длины, то необходимо указать только **index_1**. Значение индекса, равное 1, указывает на самый левый символ индексируемой строки.

Синтаксис:

```
239 index_qualifier = '[' index_1 [ ':' index_2 ] ']' .
237 index_1 = index .
236 index = numeric_expression .
238 index_2 = index .
```

Правила и ограничения:

a) Параметр **index_1** должен быть представлен положительным целым числом или неопределенным (**?**) значением.

b) Должно выполняться условие **1≤index_1≤LENGTH (строковое значение)**, в противном случае будет возвращено неопределенное (**?**) значение.

c) Если параметр **index_2** задан, то он должен быть представлен положительным целым числом или неопределенным (**?**) значением.

d) Должно выполняться условие **index_1≤index_2≤LENGTH (строковое значение)**, в противном случае будет возвращено неопределенное (**?**) значение.

e) Если **index_1** или **index_2** имеет неопределенное (**?**) значение, то результатом также будет неопределенное (**?**) значение.

f) Если индексируемое выражение имеет неопределенное (**?**) значение, то результатом также будет неопределенное (**?**) значение.

Примеры

1 Седьмой символ строковой переменной **name** ***может быть проверен следующим образом:***

```
IF name[7] = "00125FEI" THEN ... -- используется кодировка по ИСО 10646
IF name[7:7] = "00125FEI" THEN ... -- эквивалентное выражение
```

2 Символы с седьмого по десятый строковой переменной **name** ***могут быть проверены следующим образом:***

```
IF name[7:10] = 'Some' THEM ...
```

12.5.2 Оператор конкатенации строк

Оператор конкатенации строк (**+**) является строковым оператором, соединяющим две строки вместе. Оба операнда должны иметь строковое значение, а результатом является строковое значение, содержащее конкатенацию двух операндов, при этом содержимое первого операнда расположено слева. Если какой-либо из операндов имеет неопределенное (**?**) значение, то результат будет также иметь неопределенное (**?**) значение.

Пример — Строковые значения могут быть соединены следующим образом:

```
name := 'ABC' + ' ' + 'DEF' ;
(* переменная name теперь содержит строку 'ABC DEF' *)
```

12.6 Операторы агрегированных структур

К операторам агрегированных структур относятся операторы индексирования (**[ ]**), пересечения (*), объединения (**+**), различия (**–**), подмножества (**<=**), супермножества (**>=**) и запроса (**QUERY**). Определения данных операторов установлены ниже. Ко всем агрегированным значениям применимы также определенные в 12.2 операторы отношений — «равно» (**=**), «не равно» (**<>**), «равенство экземпляров» (**:=:**), «неравенство экземпляров» (**:<>:**) и **IN**.

Примечание — Для некоторых операций над агрегированными структурами требуются неявные сравнения элементов агрегированных структур; при этом используется сравнение экземпляров.

12.6.1 Индексирование агрегированных структур

Оператор индексирования агрегированных структур принимает два операнда – индексируемую агрегированную структуру и спецификацию индексов, а результатом является единственный элемент из агрегированной структуры. Типом данных выбранного элемента является базисный тип данных индексируемой агрегированной структуры.

Синтаксис:

```
239 index_qualifier = '[' index_1 [ ':' index_2 ] ']' .
237 index_1 = index .
236 index = numeric_expression .
238 index_2 = index .
```

Правила и ограничения:

a) Параметр **index_2** не должен присутствовать за исключением случая, когда должен быть проиндексирован единственный элемент из агрегированной структуры.

b) Параметр **index_1** должен быть представлен целым числом

c) Должно выполняться условие **LOINDEX (агрегированное значение) ≤index_1≤HIINDEX (агрегированное значение)**, в противном случае будет возвращено неопределенное (**?**) значение.

d) Если типом агрегированной структуры является **ARRAY** или **LIST**, то результатом будет элемент агрегированной структуры, расположенный на позиции, указанной параметром **index_1**.

e) Если типом агрегированной структуры является **BAG** или **SET**, то для каждого значения параметра **index_1**, находящегося в диапазоне от **LOINDEX (агрегированное значение)** до **HIINDEX (агрегированное значение)**, результатом должны быть разные элементы агрегированной структуры.

f) При повторном применении данного оператора к той же агрегированной структуре с тем же значением **index_1** результатом должен стать тот же элемент, если только агрегированная структура не была модифицирована. Если агрегированная структура была модифицирована, то для агрегированных типов данных **BAG** или **SET** результат повторного применения данного оператора к модифицированной агрегированной структуре непредсказуем.

g) Если **index_1** или **index_2** имеет неопределенное (**?**) значение, то результатом также будет неопределенное (**?**) значение.

h) Если индексируемое выражение имеет неопределенное (**?**) значение, то результатом также будет неопределенное (**?**) значение.

Пример — Применение оператора индексирования к пакетам и наборам может быть использовано для итерационного обращения ко всем значениям в данной агрегированной структуре.

```
FUNCTION set_product (a_set : SET OF INTEGER) : INTEGER;
LOCAL
   result : INTEGER := 1;
END_LOCAL;
   REPEAT index := LOINDEX(a_set) TO HIINDEX(a_set);
      result := result * a_set[index];
   END_REPEAT;
   RETURN (result);
END_FUNCTION;
```

***После выхода из оператора* REPEAT *переменная* result *будет содержать произведение всех целых чисел из агрегированной структуры* a_set.**

12.6.2 Оператор пересечения

Оператор пересечения (*) принимает два операнда агрегированного типа данных и выдает результат также агрегированного типа данных. Допустимые типы данных операндов и соответствующие им типы данных результата приведены в таблице 16. Результирующая агрегированная структура является неявно объявленной агрегированной структурой с типом данных, соответствующим таблице 16, и с границами **[0 . . ?]**. Базисные типы данных операндов должны быть совместимыми (см. 12.11). Если пересечение двух операндов не содержит элементов, то размер значения результирующей агрегированной структуры должен быть нулевым.

Если одним из операндов является набор, то результат должен быть набором, содержащим все элементы, присутствующие в обоих операндах.

Если оба операнда являются пакетами и некоторый элемент **e** присутствует в одном пакете **m** раз, а в другом пакете — **n** раз (где **m** меньше или равно **n**), то результат должен **m** раз содержать элемент **e**. Если какой-либо из операндов имеет неопределенное (**?**) значение, то результат будет также иметь неопределенное (**?**) значение.

Таблица 16 — Оператор пересечения: типы данных операндов и результата

Первый операнд	Второй операнд	Результат
BAG	**BAG**	**BAG**
BAG	**SET**	**SET**
SET	**SET**	**SET**
SET	**BAG**	**SET**

12.6.3 Оператор объединения

Оператор объединения (**+**) принимает два операнда, один из которых должен быть агрегированной структурой, и выдает результат агрегированного типа данных. Допустимые типы данных операндов и соответствующие им типы данных результата приведены в таблице 17. Результат оператора объединения определяется по первому выполненному из следующих условий:

a) Если левый операнд является пакетом, а правый операнд — пакетом, списком или набором, элементы которого совместимы с базисным типом данных левого операнда, то результатом является левый операнд плюс все элементы правого операнда.

b) Если левый операнд является набором, а правый операнд — пакетом, списком или набором, элементы которого совместимы с базисным типом данных левого операнда, то результат формируется следующим образом: сначала результату присваивается значение левого операнда, затем по очереди рассматриваются элементы правого операнда и, если очередного элемента нет в результирующем наборе, то данный элемент добавляется в результирующий набор.

c) Если оба операнда являются совместимыми списками, то результирующий список представляет собой левый операнд с добавленным к его концу правым операндом.

Примечание — Результирующий список может содержать повторяющиеся элементы даже если оба операнда были объявлены как **LIST OF UNIQUE**.

d) Если тип данных одного из операндов (**E**) совместим с базисным типом данных другого операнда (**A**), то операнд **E** добавляется к **A** следующим образом:

- если **A** является набором, то результирующим набором является **A**, к которому добавлен набор **E**, если только **E** уже не содержится в **A**;
- если **A** является списком, то результирующим списком является **A** со списком **E**, вставленным на позицию 1, если **E** был левым операндом, или на позицию **SIZEOF(A+1)**, если **E** был правым операндом.

Примечание — Результирующий список может содержать повторяющиеся элементы, даже если операнд, являющийся списком, был объявлен как **LIST OF UNIQUE**;

- если **A** является пакетом, то результат будет пакетом, содержащим **A** и **E**.

e) Если какой-либо из операндов имеет неопределенное (**?**) значение, то результат будет также иметь неопределенное (**?**) значение.

Таблица 17 — Оператор объединения: типы данных операндов и результата

Первый операнд	Второй операнд	Результат
BAG	**BAG**	**BAG**
BAG	Элемент	**BAG**
Элемент	**BAG**	**BAG**
BAG	**SET**	**BAG**

Окончание таблицы 17

Первый операнд	Второй операнд	Результат
BAG	**LIST**	**BAG**
SET	**SET**	**SET**
SET	Элемент	**SET**
Элемент	**SET**	**SET**
SET	**BAG**	**SET**
SET	**LIST**	**SET**
LIST	**LIST**	**LIST**[1)]
Элемент	**LIST**	**LIST**[2)]
LIST	Элемент	**LIST**[3)]

[1)] Первый элемент второго списка следует за последним элементом первого списка.
[2)] Новый элемент становится первым в результирующем списке.
[3)] Новый элемент становится последним в результирующем списке.

12.6.4 Оператор различия

Оператор различия (**–**) принимает два операнда, левый из которых должен быть агрегированной структурой, и выдает результат агрегированного типа данных. Допустимые типы данных операндов и соответствующие им типы данных результата приведены в таблице 18. Результирующая агрегированная структура содержит элементы первого операнда за исключением элементов, совпадающих с элементами второго операнда. Другими словами, каждый элемент второго операнда, который присутствует и в первом операнде, удаляется из первого операнда. Результирующая агрегированная структура является неявно объявленной агрегированной структурой с типом данных, соответствующим таблице 18, и границами **[0 . . ?]**. Базисные типы операндов должны быть совместимы (см. 12.11). Тип данных возвращаемой агрегированной структуры должен совпадать с типом данных первого операнда. Если оба операнда являются пакетами, и некоторый элемент **e** присутствует **m** раз в первом операнде и **n** раз во втором операнде, то элемент **e** должен присутствовать в результирующей агрегированной структуре **m–n** раз, если **m** больше **n**, и ни одного раза, если **m** меньше или равно **n**. Если второй операнд содержит элементы, которых нет в первом операнде, то такие элементы игнорируются и не включаются в результирующую агрегированную структуру. Если какой-либо из операндов имеет неопределенное (**?**) значение, то результат будет также иметь неопределенное (**?**) значение.

Таблица 18 — Оператор различия: типы данных операндов и результата

Первый операнд	Второй операнд	Результат
BAG	**BAG**	**BAG**
BAG	**SET**	**BAG**
BAG	Элемент	**BAG**
SET	**SET**	**SET**
SET	**BAG**	**SET**
SET	Элемент	**SET**

***Пример — Если* A *является пакетом целых чисел* [1,2,1,3], *то выражение* A – 1 *имеет значение* [1,2,3], *которое эквивалентно значению* [2,1,3].

12.6.5 Оператор подмножества

Оператор подмножества (**<=**) принимает два операнда, определенных в таблице 19, и выдает результат типа **LOGICAL**. Результат принимает значение **TRUE** тогда и только тогда, когда какой-либо элемент **e**, присутствующий **n** раз в первом операнде, присутствует не менее **n** раз во втором операнде. Результат принимает значение **UNKNOWN**, если какой-либо из операндов имеет неопределенное (**?**) значение. В противном случае результат принимает значение **FALSE**.

Типы данных операндов должны быть совместимы (см. 12.11).

Таблица 19 — Операторы подмножества и супермножества: типы данных операндов

Первый операнд	Второй операнд
BAG	**BAG**
BAG	**SET**
SET	**BAG**
SET	**SET**

12.6.6 Оператор супермножества

Оператор супермножества (**>=**) принимает два операнда, определенных в таблице 19, и выдает результат типа **LOGICAL**. Результат принимает значение **TRUE** тогда и только тогда, когда какой-либо элемент **e**, присутствующий **n** раз во втором операнде, присутствует не менее **n** раз в первом операнде. Результат принимает значение **UNKNOWN**, если какой-либо из операндов имеет неопределенное (**?**) значение. В противном случае результат принимает значение **FALSE**.

Типы данных операндов должны быть совместимы (см. 12.11).

Выражение **b >= a** должно быть полностью эквивалентно выражению **a <= b**.

12.6.7 Оператор запроса

Оператор запроса **QUERY** применяет логическое выражение **logical_expression** по отдельности к каждому элементу агрегированной структуры и формирует в качестве результата агрегированную структуру, содержащую элементы, для которых значением **logical_expression** было **TRUE**. В результате формируется подмножество исходной агрегированной структуры, все элементы которого соответствуют условию, представленному логическим выражением.

```
Синтаксис:
  277 query_expression = QUERY '(' variable_id '<*' aggregate_source '|'
            logical_expression ')' .
  170 aggregate_source = simple_expression .
  254 logical_expression = expression .
```

Правила и ограничения:

a) Элемент **variable_id** является неявно объявленной переменной в области видимости оператора запроса.

Примечание — Данная переменная не должна быть объявлена где-либо еще и не существует вне оператора запроса.

b) Элемент **aggregate_source** должен быть представлен агрегированной структурой (**ARRAY**, **BAG**, **LIST** или **SET**).

c) Если элемент **aggregate_source** имеет неопределенное (**?**) значение, то оператор возвращает также неопределенное (**?**) значение.

d) Третий операнд (**logical_expression**) должен быть выражением, результат которого имеет тип данных **LOGICAL**.

Элементы поочередно извлекают из исходной агрегированной структуры, подставляют в **logical_expression** вместо **variable_id** и вычисляют значение **logical_expression**. Если **logical_expression** имеет значение **TRUE**, то данный элемент добавляют к результату; в противном случае — не добавляют. Если **logical_expression** принимает неопределенное (**?**) значение, то данный элемент не включают в результирующую агрегированную структуру. Данные действия повторяют для каж-

дого элемента исходной агрегированной структуры. Содержимое результирующей агрегированной структуры зависит от конкретного вида агрегированного типа данных:

a) Тип данных **ARRAY**: результирующий массив имеет такие же границы и базисный тип, что и исходный массив, но элементы данного массива объявлены как **OPTIONAL**. Изначально каждый элемент имеет неопределенное (**?**) значение. Каждый элемент исходного массива, для которого **logical_expression** имеет значение **TRUE**, помещается на такую же индексную позицию в результирующем массиве.

b) Тип данных **BAG**: результирующий пакет имеет такие же базисный тип и верхнюю границу, что и исходный пакет. Нижняя граница равна нулю. Изначально результирующий пакет является пустым. Каждый элемент исходного пакета, для которого **logical_expression** имеет значение **TRUE**, добавляется в результирующий пакет.

c) Тип данных **LIST**: результирующий список имеет такие же базисный тип и верхнюю границу, что и исходный список. Нижняя граница равна нулю. Изначально результирующий список является пустым. Каждый элемент из исходного списка, для которого **logical_expression** имеет значение **TRUE**, добавляется в конец результирующего списка. Порядок следования элементов исходного списка сохраняется.

d) Тип данных **SET**: результирующий набор имеет такие же базисный тип и верхнюю границу, что и исходный набор. Нижняя граница равна нулю. Изначально результирующий набор является пустым. Каждый элемент из исходного набора, для которого **logical_expression** имеет значение **TRUE**, добавляется в результирующий набор.

Примечание — Если исходная агрегированная структура является пустой, то результатом будет пустая агрегированная структура.

Примеры

1 Пусть **colour** ***является определенным типом данных, имеющим в качестве базисного типа*** **ENUMERATION**, ***включающий элементы*** **pink** ***и*** **scarlet**. ***В приведенном ниже фрагменте показано, как извлечь из массива*** **colour** ***элементы, являющиеся либо*** **pink**, ***либо*** **scarlet**.

```
LOCAL
   colours : ARRAY OF colour;
   reds    : ARRAY OF OPTIONAL colour;
END_LOCAL;
...
reds := QUERY ( element <* colours | ( element = pink ) OR
                                     ( element = scarlet ) ) ;
...
```

2 В данном примере оператор запроса используется для проверки всех экземпляров объектного типа данных **point**. ***Результирующий набор содержит все экземпляры объекта*** **point**, ***расположенные в начале координат.***

```
RULE two_points_at_origin FOR (point);
WHERE
   SIZEOF (QUERY(temp <* point | temp = point(0.0,0.0,0.0))) = 2;
END_RULE;
```

В данном примере показаны три неявных объявления. Первым является переменная **point**, ***которая неявно объявлена в заголовке*** **RULE** ***как набор всех экземпляров*** **point**. ***Вторым является переменная*** **temp**, ***в которую собираются последовательные элементы агрегированной структуры*** **point** ***при выполнении оператора запроса. Третьим является конструктор*** **point**, ***объявление которого следует из объявления его объекта.***

12.7 Ссылки

Когда элемент, видимый в локальной области видимости, должен использоваться локально, ссылки на данный элемент должны осуществляться по идентификатору, объявленному для данного элемента.

12.7.1 Простые ссылки

Простая ссылка представляет собой просто имя (идентификатор), присвоенное элементу в текущей области видимости.

Данным способом можно ссылаться на следующие элементы, причем на элементы, помеченные одной звездочкой (*), данным способом можно ссылаться внутри выражения, а на объекты, помеченные двумя звездочками (**), можно ссылаться как на конструктор (см. 9.2.6) или как на локальную переменную в глобальном правиле (см. 9.6):

- атрибуты в объявлении объекта*;
- константы *;

- элементы из перечисляемого типа данных*;
- объекты**;
- функции*;
- локальные переменные в теле алгоритма*;
- параметры в теле алгоритма*;
- процедуры;
- правила;
- схемы в спецификации интерфейсов;
- типы данных.

Пример — Допустимые простые ссылки:

```
line      (* объектный тип данных *)
Circle    (* объектный тип данных *)
RED       (* элемент перечисления *)
z_depth   (* атрибут *)
```

12.7.2 Префиксные ссылки

В случае, когда одно и то же имя элемента перечисления объявлено в нескольких определенных типах данных, видимых в одной и той же области видимости (см. раздел 10), для обеспечения однозначной идентификации элемента перечисления его имя должно иметь префикс, представляющий собой идентификатор определенного типа данных, соответствующего данному элементу. Префиксная ссылка представляет собой имя определенного типа данных, за которым следует точка (.), за которой следует имя элемента перечисления.

Пример — В данном примере показано, как элемент перечисления **red** ***может быть однозначно идентифицирован для использования в объекте*** **stop_signal.**

```
TYPE traffic_light = ENUMERATION OF (red, amber, green);
END_TYPE;
TYPE rainbow = ENUMERATION OF
                  (red, orange, yellow, green, blue, indigo, violet);
END_TYPE;
stop_signal : traffic light := traffic light.red;
ink_colour : rainbow := blue;
```

12.7.3 Ссылки на атрибуты

Ссылка на атрибут (.) является ссылкой на отдельный атрибут экземпляра объекта. Выражение, расположенное слева от ссылки на атрибут, должно представлять экземпляр объекта или значение частичного сложного объекта. Идентификатор атрибута, на который дается ссылка, указывают после точки (.).

Синтаксис:
179 attribute_qualifier = '.' attribute_ref .

Ссылка на атрибут, используемая в выражении, возвращает значение указанного атрибута экземпляра объекта или частичного сложного объекта. Если выражение, расположенное слева от ссылки на атрибут, имеет неопределенное (?) значение, то выражение, в котором использована ссылка на атрибут, также имеет неопределенное (?) значение. Если выражение, расположенное слева от ссылки на атрибут, представляет значение частичного сложного объекта, то имя атрибута, указанное справа от ссылки на атрибут, должно присутствовать в объявлении объекта для данного частичного сложного объектного типа данных. Если объявленный тип данных выражения, расположенного слева от ссылки на атрибут, является объектным типом данных, то имя атрибута, указанное справа от ссылки на атрибут, должно быть объявлено в данном объектном типе данных либо в супертипе или подтипе данного объектного типа данных. Если объявленный тип данных выражения, расположенного слева от ссылки на атрибут, является выбираемым типом данных, то имя атрибута, указанное справа, должно быть объявлено в объекте, присутствующем в списке выбора в супертипе или подтипе объекта, присутствующего в списке выбора. Если указанного атрибута нет в экземпляре объекта или в значении частичного сложного объекта, то возвращается неопределенное (?) значение. Если два или более атрибутов имеют одно и то же имя, то данная ссылка является неоднозначной, и возвращается неопределенное (?) значение.

Примечание — В ситуации, когда возможно возникновение неоднозначности, рекомендуется использовать спецификатор групповой ссылки, чтобы ограничить область видимости ссылки.

Пример — Данный пример демонстрирует использование ссылки на атрибут.

```
ENTITY point;
   x, y, z, : REAL;
END_ENTITY;
ENTITY coloured_point
SUBTYPE OF (point);
   colour : colour;
END_ENTITY;
...
PROCEDURE foo;
LOCAL
   first      : point := point(1.0, 2.0, 3.0);
   second  : coloured_point := point(1.0,2.0,3.0)||coloured_point(red);
   x_coord : REAL;
END_LOCAL;
...
x_coord := first.x;  -- "foo" имеет значение 1.0
IF first.colour = red THEN  (*colour в "foo" является допустимой ссылкой, так как данный атрибут
                            присутствует в подтипе coloured_point, однако, в данном случае ссылка на
                            атрибут вернет неопределенное (?) значение, поскольку он не присутствует
                            в данном экземпляре. *)
IF second.colour = red THEN  -- Значением "foo" является TRUE, так как
                             -- colour является допустимой ссылкой
```

12.7.4 Групповые ссылки

Групповая ссылка (****) обеспечивает ссылку на значение частичного сложного объекта в экземпляре сложного объекта. Выражение, расположенное слева от групповой ссылки, должно представлять экземпляр сложного объекта. Объектный тип данных значения частичного сложного объекта, на который делается ссылка, указывается после обратной косой черты (****).

Синтаксис:

232 group_qualifier = '\' entity_ref .

Групповая ссылка, используемая в выражении, возвращает значение частичного сложного объекта, соответствующее именованному объектному типу данных в экземпляре сложного объекта, на который делается ссылка. Если выражение, расположенное слева от групповой ссылки, имеет неопределенное (**?**) значение, то выражение, содержащее групповую ссылку, также имеет неопределенное (**?**) значение. Если объявленный тип данных выражения, расположенного слева от групповой ссылки, является объектным типом данных, то имя объекта, указанное справа от групповой ссылки, должно соответствовать объекту из того же графа подтипов/супертипов, что и данный объектный тип данных. Если объявленный тип данных выражения, расположенного слева от групповой ссылки, является выбираемым типом данных, то имя объекта, указанное справа, должно присутствовать в списке выбора или соответствовать объекту из того же графа подтипов/супертипов объектного типа данных, представленного в списке выбора. Если указанный объектный тип данных не представлен в экземпляре сложного объекта, на который делается ссылка, то возвращается неопределенное (**?**) значение. Групповая ссылка может быть далее уточнена посредством ссылки на атрибут. В этом случае групповая ссылка определяет область видимости ссылки на атрибут.

Примечание — Данное использование групповой ссылки требуется тогда, когда тип данных экземпляра сложного объекта имеет несколько атрибутов с одинаковым именем или когда выбираемый тип данных содержит несколько объектов с атрибутами, имеющими одинаковое имя.

Ограничение: групповая ссылка, которая не уточнена ссылкой на атрибут, должна присутствовать в качестве операнда либо оператора сравнения значений объектов (**=**), либо конструктора экземпляра сложного объекта (**||**).

Примеры

1 В данном примере показано использование групповой ссылки при сравнении значений.

```
ENTITY E1
ABSTRACT SUPERTYPE;
   attrib1 : REAL;
   attrib2 : REAL;
   attrib3 : REAL;
END_ENTITY;
```

```
ENTITY E2
SUBTYPE OF (E1);
   attribA : INTEGER;
   attribB : INTEGER;
   attribC : INTEGER;
END_ENTITY;
LOCAL
   a : E1;
   b : E2;
END_LOCAL;
   -- построим экземпляры сложных объектов a и b,
   -- используя оператор конструирования экземпляра сложного объекта
a := E1(0.0,1.0,2.0)||E2(1,2,3);
b := E1(0.0,1.0,2.0)||E2(3,2,1);
   -- проверим значения в a и b атрибутов,
   -- объявленных в E1
a\E1 = b\E1 -- TRUE
(*
    это эквивалентно следующему:
        (a.attrib1 = b.attrib1) AND
        (a.attrib2 = b.attrib2) AND
        (a.attrib3 = b.attrib3)
*)
```

2 В данном примере показано использование групповой ссылки для указания конкретного объектного типа данных, который может быть использован для имени атрибута.

```
ENTITY foo1;
   attr : REAL;
END_ENTITY;
ENTITY foo2
   SUBTYPE OF (foo1);
   attr2 : BOOLEAN;
END_ENTITY;
ENTITY t;
   attr : BINARY;
END_ENTITY;
TYPE crazy=SELECT(foo2,t);
END_TYPE;
...
LOCAL
   v : crazy;
END_LOCAL;
...
IF 'THIS.F002' IN TYPEOF(v) THEN   -- этим обеспечивается отсутствие
                                   -- непредсказуемых результатов
                                   -- (иногда это называется «защита»).
  v\foo1.attr := 1.5;              -- присваивает 1,5 атрибуту v attr,
                -- так как attr определен в foo1, в групповой ссылке
                -- должен использоваться foo1.
END_IF;
```

12.8 Вызов функции

Вызов функции активизирует данную функцию. Вызов функции состоит из идентификатора функции, за которым может следовать список фактических параметров. Число, тип и порядок следования фактических параметров должны соответствовать формальным параметрам, определенным для данной функции. Вызов функции возвращает значение функции при подстановке в объявлении функции фактических параметров вместо формальных параметров.

Примечание — Фактические параметры функции могут иметь неопределенное (?) значение. Функция должна корректно обрабатывать такие значения и может сама возвращать неопределенное значение.

Активизация функции расширяет пространство экземпляров. Любые экземпляры, созданные в процессе выполнения функции, должны быть однозначно идентифицируемыми во всей совокупности извест-

ных экземпляров. Как правило, созданный таким образом экземпляр недоступен вне создающей его функции и, в частности, не является частью рассматриваемой совокупности экземпляров. Исключением является случай, когда такой экземпляр возвращается в качестве результата или в составе результата вызова функции. В данном случае экземпляр остается доступным в точке вызова функции. Если экземпляр возвращается подобным образом на уровень схемы (то есть как значение вычисляемого атрибута или константы), он рассматривается как часть общей совокупности экземпляров.

Синтаксис:

```
219 function_call = ( built_in_function | function_ref )
                    [ actual_parameter_list ] .
167 actual_parameter_list = '(' parameter { ',' parameter } ')' .
264 parameter = expression .
```

Ограничение: передаваемые фактические параметры должны быть совместимы по назначению с формальными параметрами.

Пример — Пример использования вызова функции:

```
ENTITY point;
   x, y, z : number;
END_ENTITY;
FUNCTION midpoint_of_line(l:line):point;
...
END_FUNCTION;
IF midpoint_of_line(L506).x = 9.0 THEN ...
                        -- применяя оператор ссылки на атрибут
                        -- непосредственно к результату функции
END_IF;
```

12.9 Инициализатор агрегированных структур

Инициализатор агрегированных структур используется, чтобы установить значение типа **AGGREGATE OF GENERIC**, которое может быть задано массиву, пакету, списку или набору. В квадратных скобках может быть не заключено ни одного или заключено несколько выражений, представляющих значения, принадлежащие к типу данных, совместимому с базисным типом данных агрегированной структуры. При наличии двух или более значений, они должны разделяться запятыми. Разреженный массив может быть инициализирован посредством использования неопределенности (**?**) вместо отсутствующих символов. Результатом выражения инициализатора агрегированной структуры является агрегированное значение, содержащее значения, определенные как его элементы. Число инициализированных элементов должно соответствовать границам, заданным для данного агрегированного типа данных.

Инициализатор агрегированной структуры, не содержащий ни одного элемента, устанавливает пустой пакет, список или набор (данная конструкция не может быть использована для инициализации пустых массивов).

Синтаксис:

```
169 aggregate_initializer = '[' [ element { ',' element } ] ']' .
203 element = expression [ ':' repetition ] .
287 repetition = numeric_expression .
```

Пример — Задано объявление:

```
a : SET OF INTEGER;
```

его значение может быть задано следующим образом:

```
a := [1, 3, 6, 9*8, –12]; -- 9*8 является выражением со значением 72
```

Если несколько последовательных значений является одинаковыми, то может быть использовано выражение повторения, которое представляется двумя выражениями, разделенными символом двоеточия (**:**). Выражение слева от двоеточия представляет значение, которое должно повторяться. Выражение справа от двоеточия задает число повторений левого значение. Данное выражение вычисляется один раз, перед инициализацией, и должно иметь неотрицательное целочисленное значение.

Пример — Задано следующее объявление:

```
a : BAG OF BOOLEAN;
```

Следующие два оператора эквивалентны:

```
a := [TRUE : 5];
a := [TRUE, TRUE, TRUE, TRUE, TRUE];
```

12.10 Оператор построения экземпляра сложного объекта

Оператор построения экземпляра сложного объекта (||) создает экземпляр сложного объекта посредством объединения частичных значений сложного объекта. Частичные значения сложного объекта могут объединяться в произвольном порядке. Результатом выражения оператора построения экземпляра сложного объекта является частичное значение сложного объекта либо экземпляр сложного объекта. Частичный сложный объектный тип данных может присутствовать только один раз на одном уровне выражения оператора построения экземпляра сложного объекта. Частичное значение сложного объекта может присутствовать на разных уровнях, если они являются вложенными, то есть если частичное значение сложного объекта используется для построения экземпляра сложного объекта, являющегося атрибутом частичного значения сложного объекта, объединяемого с другими элементами при построении экземпляра сложного объекта. Если какой-либо из операндов имеет неопределенное (**?**) значение, то результатом выражения будет также неопределенное (**?**) значение. Дополнительная информация об экземплярах сложных объектов представлена в приложении В.

Пример — Задано:

```
ENTITY a
ABSTRACT SUPERTYPE;
   a1 : INTEGER;
END_ENTITY;
ENTITY b SUBTYPE OF (a);
   b1 : STRING;
END_ENTITY;
ENTITY c SUBTYPE OF (a);
   c1 : REAL;
END_ENTITY;
```

Тогда могут быть простроены следующие экземпляры сложных объектов:

```
LOCAL
   v1 = a ;
   v2 = c ;
END_LOCAL;
   v2 := a(2) || c(7.998e-5); -- это экземпляр типа a&c
   v1 := v2 || b('abc');      -- это экземпляр типа a&b&c
   v1 := v2\a || b("00002639"); -- это экземпляр типа a&b
   v1 := v1 || v2;            -- недопустимо, т.к. тип был бы a&b&a&c
```

Примечание — Назначение **v1** копирует экземпляр, созданный оператором построения экземпляра сложного объекта; данный экземпляр содержит значение **v2**, но не экземпляр **v2**.

12.11 Совместимость типов

Операнды оператора должны быть совместимы с типом (типами) данных, необходимых оператору. Типы данных обоих операндов некоторых операторов также должны быть совместимы друг с другом, что было определено выше в данном разделе. Типы данных могут быть совместимыми, не будучи идентичными. Типы данных являются совместимыми при выполнении одно из следующих условий:

- типы данных совпадают;
- один тип данных является подтипом или конкретизацией другого (включая определенные типы данных, использующие определенный тип данных в качестве базисного типа данных, и конструкционные типы данных, основанные на наращиваемых типах данных);
- оба типа данных являются типами данных **ARRAY** с совместимыми базисными типами данных и одинаковыми границами;
- оба типа данных являются типами данных **LIST** с совместимыми базисными типами данных;
- оба типа данных являются типами данных **BAG** или **SET** с совместимыми базисными типами данных.

Пример — Заданы следующие определения:

```
TYPE natural = REAL;
WHERE SELF >= 0.0;
END_TYPE;
TYPE positive = natural;
WHERE SELF > 0.0;
END_TYPE;
```

```
TYPE bag_of_natural = BAG OF natural;
END_TYPE;
TYPE set_of_up_to_five_positive = SET [0 : 5] OF positive;
END_TYPE;
```

При этом совместимыми являются следующие типы данных:

Тип данных	*Совместим с типами данных*
REAL	**INTEGER, REAL, NUMBER, natural, positive**
natural	**REAL, NUMBER, natural, positive**
positive	**REAL, NUMBER, natural, positive**
bag_of_natural	**BAG OF REAL, BAG OF NUMBER, BAG OF natural, BAG OF positive, SET OF REAL, SET OF NUMBER, SET OF natural, SET OF positive, bag_of_natural, set_of_up_to_five_positive**
set_of_up_to_five_positive	**BAG OF REAL, BAG OF NUMBER, BAG OF natural, BAG OF positive, SET OF REAL, SET OF NUMBER, SET OF natural, SET OF positive, bag_of_natural,**

12.12 Выбираемые типы данных в выражениях

При проверке схемы парсер уровня 2 должен идентифицировать совместимость типов операндов и операторов в выражениях. Выражение, содержащее тип данных **SELECT**, может быть допустимым только для некоторых типов данных из списка выбора и не допустимым для остальных типов данных из списка выбора. Ранее в данном разделе были определены допустимые типы данных в выражениях, кроме типов данных **SELECT**; установленные ниже правила относятся конкретно к данным типам данных.

Тип данных, возвращаемый выражением, содержащим операнды, объявленный тип данных которых является выбираемым типом данных, является выбираемым типом данных, содержащим все возможные типы данных, возвращенные допустимыми выражениями из указанных операндов.

Невыбираемые типы данных в выбираемом типе данных – это невыбираемые типы данных каждого типа данных из списка выбора выбираемого типа данных; невыбираемым типом данных в типе данных, который не является выбираемым, является сам данный тип данных.

12.12.1 Выбираемые типы данных в унарных выражениях

В данном пункте определена обработка выбираемых типов данных в выражениях с одним операндом, к которым относятся операторы: **–**, **+**, **NOT** и **QUERY**:

a) Если все невыбираемые типы данных в списке выбора объявленного типа операнда допустимы в контексте данного выражения, то выражение является допустимым и должно возвращать допустимый результат.

b) Если некоторые, но не все невыбираемые типы данных в списке выбора объявленного типа операнда допустимы в контексте данного выражения, то выражение является допустимым, но может выдать ошибку, если в выражении вычисляются значения типов данных, являющихся недопустимыми.

c) Если ни один из невыбираемых типов данных в списке выбора объявленного типа операнда не является допустимым в контексте данного выражения, то выражение является недопустимым и всегда будет возвращать недопустимый результат.

12.12.2 Выбираемые типы данных в бинарных выражениях

В данном пункте определена обработка выбираемых типов данных в выражениях с двумя операндами:

a) Если для каждого невыбираемого типа данных в списке выбора объявленного типа левого операнда существует допустимое выражение с каждым невыбираемым типом данных в списке выбора объявленного типа правого операнда, то данное выражение является допустимым и должно возвращать допустимый результат.

b) Если некоторые, но не все невыбираемые типы данных в списке выбора объявленного типа левого операнда и, по крайней мере, один невыбираемый тип данных правого операнда допустимы в контексте данного выражения, то выражение является допустимым, но может выдать ошибку, если в выражении вычисляются значения типов данных, являющихся недопустимыми.

c) Если ни один из невыбираемых типов данных в списке выбора объявленного типа левого операнда и какой-либо из невыбираемых типов данных правого операнда не является допустимым в контексте данного выражения, то выражение является недопустимым и всегда будет возвращать недопустимый результат.

12.12.3 Выбираемые типы данных в тернарных выражениях

В данном пункте определена обработка выбираемых типов данных в выражениях с тремя операндами.

Единственным выражением в языке EXPRESS, содержащим три операнда, является интервальное выражение. Оно рассматривается в контексте выбираемых типов данных, как если бы существовало два отдельных выражения, связанных оператором **AND**.

13 Исполняемые операторы

Исполняемые операторы определяют действия функций, процедур и правил. Данные операторы воздействуют только на переменные, локальные по отношению к **FUNCTION**, **PROCEDURE** или **RULE**. Данные операторы используются для определения логики действий, необходимых для поддержки определения ограничений, которые задаются условиями **WHERE** и правилами **RULE**. Данные операторы не оказывают влияния на экземпляры объектов в области определения, как установлено в разделе 5. К исполняемым операторам относятся: пустой оператор, **ALIAS**, оператор присваивания, **CASE**, составной оператор, **ESCAPE**, **IF**, вызов процедуры, **REPEAT**, **RETURN** и **SKIP**.

Исполняемые операторы могут присутствовать только внутри **FUNCTION**, **PROCEDURE** или **RULE**.

```
Синтаксис:
  309 stmt = alias_stmt | assignment_stmt | case_stmt | compound_stmt |
             escape_stmt | if_stmt | null_stmt | procedure_call_stmt |
             repeat_stmt | return_stmt | skip_stmt .
```

13.1 Пустой оператор

Исполняемый оператор, состоящий только из точки с запятой (;), называется пустым оператором. Никаких действий пустой оператор не выполняет.

```
Синтаксис:
  260 null_stmt = ';' .
```

Пример — В данном примере показано возможное использование пустого оператора.

```
IF a = 13 THEN
  ; -- это пустой оператор.
ELSE
  b := 5;
END_IF;
```

13.2 Оператор ALIAS

Оператор **ALIAS** обеспечивает возможность локального переименования уточненных переменных и параметров.

```
Синтаксис:
  174 alias_stmt = ALIAS variable_id FOR general_ref { qualifier } ';' stmt { stmt }
                   END_ALIAS ';' .
  228 general_ref = parameter_ref | variable_ref .
```

В области видимости оператора **ALIAS** переменная **variable_id** неявно объявлена имеющей надлежащий тип данных и содержит значение, на которое ссылается уточняющий идентификатор, следующий за ключевым словом **FOR**.

Примечание — Правила видимости для **variable_id** описаны в 10.3.1.

Пример — Предположим, что существует объектный тип данных **point** ***с атрибутами*** **x,y,z;** ***тогда оператор*** **ALIAS** ***может быть использован в функции*** **calculate_length** ***для сокращения длины возвращаемого выражения.***

```
ENTITY line;
  start_point,
  end_point : point;
END_ENTITY;
```

```
FUNCTION calculate_length (the_line : line) : real;
ALIAS s FOR the_line.start_point;
  ALIAS e FOR the_line.end_point;
    RETURN (SQRT( (s.x – e.x)**2 + (s.y – e.y)**2 + (s.z – e.z)**2) );
  END ALIAS;
END_ALIAS;
END_FUNCTION;
```

13.3 Присваивание

13.3.1 Оператор присваивания

Оператор присваивания используется для задания экземпляра локальной переменной или параметру. Если выражение, расположенное справа от оператора присваивания, является экземпляром объекта, то оператор присваивания задает локальной переменной или параметру ссылку на данный экземпляр объекта. После исполнения оператора присваивания изменения локальной переменной или параметра отражаются в исходном экземпляре. Оператор присваивания может также использоваться для копирования значений в локальную переменную или экземпляр, когда они объявляются принадлежащими к необъектным типам данных. Тип данных значения, присвоенного переменной, должен быть совместим по присваиванию с переменной или параметром.

Примечание — Оператор присваивания не может использоваться для создания копии значения экземпляра в локальной переменной или параметре.

Синтаксис:

```
176 assignment_stmt = general_ref { qualifier } ':=' expression ';' .
228 general_ref = parameter_ref | variable_ref .
```

Пример — Следующие фрагменты демонстрируют допустимые присваивания:

```
LOCAL
  a, b : REAL;
  p    : point;
END LOCAL;
...
  a   := 1.1;
  b   := 2.5 * a;
  p.x := b;
```

13.3.2 Совместимость по присваиванию

Для значения, присваиваемого вычисляемому атрибуту, локальной переменной или параметру, должны выполняться два условия.

Примечание — В приведенном ниже тексте термин «переменная» использован для обозначения вычисляемого атрибута, локальной переменной или параметра.

a) результирующий тип данных присваиваемого выражения должен быть совместимым с типом данных переменной;

b) результирующее значение, полученное при вычислении выражения, должно удовлетворять всем ограничениям, установленным для данного типа данных переменной.

Считается, что тип данных присваиваемого выражения и тип данных переменной являются совместимыми, если выполняется одно из следующих условий:

a) типы данных являются одинаковыми;

b) результат выражения принадлежит к типу данных, который является подтипом или конкретизацией типа данных, объявленного для переменной, которой он присваивается;

c) объявленный тип данных переменной, которой присваивается значение, является определенным типом данных, основным типом данных которого является выбираемый тип данных, а результат выражения принадлежит к типу данных, совместимому по присваиванию с одним или более типами данных, установленными в области определения выбираемого типа данных (включая элементы, добавленные к данной области определения другими выбираемыми типами данных, основанными на данном выбираемом типе данных).

Основным типом определенного типа данных является основный тип базисного типа данных, а основным типом типа данных, не являющегося определенным типом данных, является сам данный тип данных;

d) переменная представлена определенным типом данных, основным типом которого является простой тип данных, а результатом выражения — значение данного простого типа данных;

e) переменная представлена агрегированным типом данных, а выражение является инициализатором агрегированной структуры, элементы которой, при их наличии, совместимы по присваиванию с базисным типом данного агрегированного типа данных;

f) если присваиваемый объект является уточненным, то следующие условия должны выполняться для разных видов уточнения:

1) уточнение атрибутом:

- объявленный тип выражения, расположенного слева от ссылки на атрибут, должен быть объектным типом данных или выбираемым типом данных, определенным с использованием, по крайней мере, одного объектного типа данных. Атрибут, имя которого указано справа от ссылки на атрибут, должен присутствовать в объектном типе данных или в объекте, присутствующем в том же графе подтипов/супертипов, что и объектный тип данных,

- если результатом выражения, расположенного слева от ссылки на атрибут, является экземпляр, содержащий указанный атрибут, имеющий некоторое значение, то исходное значение заменяется выражением, расположенным справа от оператора присваивания, если только уточнение объекта, которому присваивается значение, не продолжается дальше; в последнем случае используется данное дальнейшее уточнение,

- если результатом выражения, расположенного слева от ссылки на атрибут, является экземпляр, содержащий указанный атрибут, а данный атрибут имеет неопределенное (**?**) значение (если он является необязательным или еще не инициализирован), то выражение, расположенное справа от оператора присваивания, назначается данному атрибуту, если только уточнение объекта, которому присваивается значение, не продолжается дальше; в последнем случае фиксируется ошибка;

2) уточнение группой:

- объявленный тип выражения, расположенного слева от ссылки на группу, должен быть объектным типом данных или выбираемым типом данных, определенным с использованием, по крайней мере, одного объектного типа данных. Объект, имя которого указано справа от ссылки на группу, должен присутствовать в том же графе подтипов/супертипов, что и объектный тип данных,

- если результатом выражения, расположенного слева от ссылки на группу, является экземпляр, содержащий имя объекта, указанное справа от ссылки на группу, то исходное частичное значение сложного объекта заменяется выражением, расположенным справа от оператора присваивания, если только уточнение объекта, которому присваивается значение, не продолжается дальше; в последнем случае используется данное дальнейшее уточнение;

3) уточнение элементом:

- объявленный тип выражения, расположенного слева от квалификатора элемента, должен быть одним из (**ARRAY**, **BINARY**, **LIST** или **STRING**), либо выбираемым типом данных, определенным с использованием одного из указанных типов данных. Параметр **index_1** должен иметь целочисленное значение,

- выражение, расположенное слева от квалификатора элемента, должно быть инициализировано, то есть иметь значение до того, как его элементам могут быть присвоены значения,

- если результат выражения, расположенного слева от квалификатора элемента, принадлежит к типу данных **ARRAY** и выполняется **{LOINDEX (left) <=index_1<=HIINDEX (left)}**, то:

если в массиве уже имеется элемент на данной позиции, то выражение, расположенное справа от оператора присваивания, замещает исходное значение, расположенное в массиве на данной позиции, если только уточнение объекта, которому присваивается значение, не продолжается дальше; в последнем случае для исходного элемента используется данное дальнейшее уточнение,

если в массиве на данной позиции присутствует неопределенное (**?**) значение, которое далее не уточняется, то выражение, расположенное справа от оператора присваивания вставляется в массив на данную позицию,

- если результат выражения, расположенного слева от квалификатора элемента, принадлежит к типу данных **BINARY** и выполняется **{1<=index_1<=BLENGTH (left)}**, то выражение, расположенное справа от оператора присваивания, заменяет в двоичном числе бит, расположенный на данной позиции,

- если результат выражения, расположенного слева от квалификатора элемента, принадлежит к типу данных **LIST** и выполняется **{1<=index_1<=SIZEOF(left)}**, то выражение, расположенное справа от оператора присваивания, заменяет в списке элемент, расположенный на данной позиции, если только объект, которому присваивается значение, не уточняется; в последнем случае используется данное дальнейшее уточнение,

- если результат выражения, расположенного слева от квалификатора элемента, принадлежит к типу данных **STRING** и выполняется **{1<=index_1<=LENGTH(left)}**, то выражение, расположенное справа от оператора присваивания, заменяет в строке символ, расположенный на данной позиции;

4) уточнение диапазоном:

- объявленный тип выражения, расположенного слева от квалификатора диапазона, должен быть **BINARY** или **STRING** либо выбираемым типом данных, определенным с использованием одного из указанных типов данных. Параметры **index_1** и **index_2** должны иметь целочисленное значение,

- выражение, расположенное слева от квалификатора диапазона, должно быть инициализировано, то есть иметь значение до того, как его элементам могут быть присвоены значения,

- если результат выражения, расположенного слева от квалификатора диапазона, принадлежит к типу данных **BINARY** и выполняется **{1<=index_1<=index_2}AND (index_2<=BLENGTH (left))**, то выражение, расположенное справа от оператора присваивания замещает элементы, расположенные в исходном двоичном числе между позициями **index_1** и **index_2**.

Примечание — Если **BLENGTH(right)<>(index_2–index_1+1)**, то данным присваиванием будет заменен элемент **BLENGTH(left)**,

- если результат выражения, расположенного слева от квалификатора диапазона, принадлежит к типу данных **STRING** и выполняется **{1<=index_1<=index_2}AND(index_2<=LENGTH(left))**, то выражение, расположенное справа от оператора присваивания, замещает элементы, расположенные в исходной строке между позициями **index_1** и **index_2**.

Примечание — Если **LENGTH (right) <> (index_2– index_1+1)**, то данным присваиванием будет заменен элемент **LENGTH (left)**;

g) если объект, которому присваивается значение уточнен, но не соответствует ни одному из перечисленных выше случаев, то фиксируется ошибка.

Если синтаксический анализатор, обеспечивающий проверку уровня 4 (см. 4.1.1) определит, что результат выражения принадлежит к типу данных, являющемуся обобщением типа данных, объявленного для переменной, которой присваивается значение, то данный оператор присваивания считается недопустимым. При этом могут существовать допустимые присваивания с использованием данного оператора, если фактические значения, возвращаемые выражением, соответствуют установленным выше условиям, однако, может быть получен непредсказуемый результат, если фактические значения, возвращаемые выражением, не совместимы с установленными выше условиями.

Частичные экземпляры, являющиеся недопустимыми экземплярами сложного объекта (см. приложение В), не могут быть присвоены параметрам или переменным сложного объекта и переданы в качестве фактических параметров функциям или процедурам. Данное требование не ограничивает присваивание допустимых экземпляров сложного объекта.

13.4 Оператор CASE

Оператор выбора CASE обеспечивает механизм для выборочного исполнения операторов на основе значения некоторого выражения. Оператор исполняется в зависимости от значения переключателя (элемент **selector**). Оператор выбора состоит из выражения, являющегося переключателем блоков, и списка альтернативных действий, каждому из которых предшествует одно или несколько выражений, являющихся метками блоков. Результирующий тип данных метки блока должен быть совместим с типом данных переключателя блоков. Выбирается первая встретившаяся метка блока, имеющая значение, равное переключателю блоков, и исполняются операторы, связанные с данной меткой. Если результатом сравнения метки и переключателя является **UNKNOWN** или **FALSE**, то выбор не проводится. Исполняется не более одного из блоков выбора. Если переключатель блоков имеет неопределенное (**?**) значение, то исполняется блок, которому предшествует ключевое слово **OTHERWISE**, при его наличии. Если метка блока имеет неопределенное (**?**) значение, то результатом сравнения является **UNKNOWN**, и оператор не должен

исполняться. Если значение ни одной из меток блоков не совпадает со значением переключателя блоков, то возможны следующие альтернативы:

- если ключевое слово **OTHERWISE** присутствует, то исполняется связанный с ним оператор;
- если ключевое слово **OTHERWISE** отсутствует, то ни один из операторов, связанных с оператором выбора, не исполняется.

Синтаксис:

```
191 case_stmt = CASE selector OF { case_action } [ OTHERWISE ':' stmt ]
                END_CASE ';' .
299 selector = expression .
189 case_action = case_label { ',' case_label } ':' stmt .
190 case_label = expression .
```

Ограничение: тип данных вычисленного значения меток блоков должен быть совместим с типом данных вычисленного значения переключателя блоков.

Пример — Простой оператор выбора, использующий целочисленные метки блоков.

```
LOCAL
   a : INTEGER;
   x : REAL;
END_LOCAL;
...
a := 3;
x := 34.97;
CASE a OF
   1      : x := SIN(x);
   2      : x := EXP(x);
   3      : x := SQRT(x); -- этот оператор исполняется!
   4, 5   : x := LOG(x);
OTHERWISE : x := 0.0;
END_CASE;
```

13.5 Составной оператор

Составной оператор представляет собой последовательность операторов, ограниченную ключевыми словами **BEGIN** и **END**. Составной оператор действует как единый оператор.

Примечание — Составной оператор не определяет новую область видимости.

Синтаксис:

```
192 compound_stmt = BEGIN stmt { stmt } END ';' .
```

Пример — Простой составной оператор:

```
BEGIN
   a = a+1;
   IF a > 100 THEN
      a := 0;
   END_IF;
END;
```

13.6 Оператор ESCAPE

Оператор **ESCAPE** вызывает немедленный переход к оператору, следующему непосредственно за оператором **REPEAT**, в котором встретился данный оператор **ESCAPE**.

Примечание — Применение оператора **ESCAPE** является единственным способом выхода из оператора **REPEAT** в случае задания бесконечного цикла.

Синтаксис:

```
214 escape_stmt = ESCAPE ';' .
```

Ограничение: оператор **ESCAPE** должен присутствовать только в пределах области видимости оператора **REPEAT**.

***Пример — Оператор* ESCAPE *передает управление оператору, следующему за* END_REPEAT, *если* a < 0:**

```
REPEAT UNTIL (a=1);
  . . .
  IF (a < 0) THEN
    ESCAPE;
  END_IF;
  . . .
END_REPEAT; -- управление передается в точку после END_REPEAT
```

13.7 Оператор IF. . .THEN. . .ELSE

Оператор **IF. . .THEN. . .ELSE** обеспечивает условное выполнение операторов на основе значения выражения типа **LOGICAL**. Если значением **logical_expression** является **TRUE**, то исполняется оператор, следующий за ключевым словом **THEN**. Если значением **logical_expression** является **FALSE**, **UNKNOWN** или неопределенность (**?**), то исполняется оператор, следующий за ключевым словом **ELSE**, если данное ключевое слово присутствует. Если значением **logical_expression** является **FALSE**, **UNKNOWN** или неопределенность (**?**), а ключевое слово **ELSE** отсутствует, то управление передается следующему оператору.

Синтаксис:

```
233 if_stmt = IF logical_expression THEN stmt { stmt } [ ELSE stmt { stmt } ]
              END_IF ';' .
254 logical_expression = expression .
```

***Пример — Простой оператор* IF:**

```
IF a < 10 THEN
  c := c + 1;
ELSE
  c := c – 1;
END_IF;
```

13.8 Оператор вызова процедуры

Оператор вызова процедуры активизирует процедуру. Фактические параметры, передаваемые при вызове процедуры, должны соответствовать по числу, порядку и типу данных с формальными параметрами, установленными для данной процедуры.

Синтаксис:

```
270 procedure_call_stmt = ( built_in_procedure | procedure_ref)
                          [ actual_parameter_list ] ';' .
167 actual_parameter_list = '(' parameter { ',' parameter } ')' .
264 parameter = expression .
```

Ограничение: передаваемые фактические параметры должны быть совместимы по присваиванию с формальными параметрами.

***Пример — Вызов встроенной процедуры* INSERT:**

```
INSERT (point_list, this_point, here );
```

13.9 Оператор REPEAT

Оператор цикла **REPEAT** используется для зависящего от условия повторения исполнения последовательности операторов. Начало или продолжение повторения определяется по вычисленному значению управляющего условия (или условий). Управляющими условиями являются:

- конечное число итераций (инкрементное управление);
- пока условие имеет значение **TRUE** (управляющее условие **WHILE**);
- до тех пор, пока условие имеет значение **TRUE** (управляющее условие **UNTIL**).

Синтаксис:

```
286 repeat_stmt = REPEAT repeat_control ';' stmt { stmt } END_REPEAT ';' .
285 repeat_control = [ increment_control ] [ while_control ] [ until_control ] .
235 increment_control = variable_id ':=' bound_1 TO bound_2 [ BY increment ] .
183 bound_1 = numeric_expression .
184 bound_2 = numeric_expression .
234 increment = numeric_expression .
```

Для задания условий окончания цикла может использоваться комбинация управляющих условий. Для управления итерациями значения данных условий определяются следующим образом:

a) Если выполняется оператор **REPEAT** и при этом присутствует инкрементное управляющее условие, то вычисляется выражение, определяющее данное инкрементное условие в соответствии с 13.9.1.

b) Если присутствует управляющее условие **WHILE**, то вычисляется значение указанного в нем выражения. Если значением выражения является **TRUE** (либо если управляющее условие **WHILE** отсутствует), то исполняется тело оператора **REPEAT**. Если значением выражения является **FALSE**, **UNKNOWN** или неопределенность (**?**), то исполнение оператора **REPEAT** заканчивается.

c) Когда заканчивается исполнение тела оператора **REPEAT**, вычисляют значение выражения управляющего условия **UNTIL**. Если значением выражения является **TRUE**, то дальнейшее исполнение итераций прекращается, и исполнение оператора **REPEAT** завершается. Если значением выражения является **FALSE** или **UNKNOWN**, то исполнение оператора **REPEAT** возвращается к проверке инкрементного управляющего условия. Если управляющее условие **UNTIL** отсутствует, то исполнение оператора **REPEAT** возвращается к проверке инкрементного управляющего условия.

d) Если инкрементное управляющее условие присутствует, то значение переменной цикла изменяется на значение, задаваемое элементом **increment**. Если значение переменной цикла находится в пределах от **bound_1** до **bound_2**, включая данные границы, то управление передается на шаг по перечислению b), в противном случае, исполнение оператора **REPEAT** заканчивается.

Пример — В данном примере показано, как в операторе* REPEAT *могут использоваться несколько управляющих условий. Повтор исполнения операторов тела цикла осуществляется до тех пор, пока не выполнится одно из двух условий, то есть пока не будет достигнута заданная точность или выполнено сто циклов; то есть итерационный процесс прекращается, если решение не сходится достаточно быстро.

```
REPEAT i := 1 TO 100 UNTIL epsilon < 1.E–6;
  ...
  epsilon := ... ;
END_REPEAT;
```

13.9.1 Инкрементное управление

При инкрементном управлении тело оператора цикла исполняется для следующих одно за другим значений из некоторой последовательности. При входе в оператор цикла неявно объявленной переменной числового типа **variable_id** присваивается значение **bound_1**. После каждой итерации переменной **variable_id** присваивается значение **variable_id + increment**. Если элемент **increment** не задан, то по умолчанию используется значение равное единице (1). Если значение **variable_id** находится в пределах между **bound_1** и **bound_2** (включая случай, когда **variable_id=bound_2**), то выполнение оператора цикла продолжается.

Синтаксис:

```
235 increment_control = variable_id ':=' bound_1 TO bound_2 [ BY increment ] .
183 bound_1 = numeric_expression .
184 bound_2 = numeric_expression .
234 increment = numeric_expression .
```

Правила и ограничения:

a) Выражения **numeric_expression**, представляющие элементы **bound_1**, **bound_2** и **increment**, должны иметь числовые значения.

b) Выражения **numeric_expression**, представляющие границы и приращение, вычисляют один раз при входе в оператор **REPEAT**.

c) Если какое-либо из выражений **numeric_expression**, представляющих границы или приращение, имеет неопределенное (**?**) значение, то оператор **REPEAT** не исполняется.

d) Перед первым вычислением оператора инкрементного управления проверяют следующие условия:

- если приращение (элемент **increment**) положительно и **bound_1 > bound_2**, то оператор **REPEAT** не исполняется,
- если приращение (элемент **increment**) отрицательно и **bound_1<bound_2**, то оператор **REPEAT** не исполняется,
- если приращение (элемент **increment**) равно нулю (**0**), то оператор **REPEAT** не исполняется;

- во всех других случаях оператор **REPEAT** исполняется до тех пор, пока значение переменной **variable_id** не выйдет за заданные границы или один из других управляющих операторов в теле оператора **REPEAT** не завершит его исполнение.

e) Переменная цикла инициализируется со значением **bound_1** в начале первого цикла итераций, и изменяется на значение, определяемое элементом **increment** в начале каждого последующего цикла.

f) Значение переменной цикла не должно изменяться в теле оператора **REPEAT**.

g) Оператор **REPEAT** устанавливает локальную область видимости, в которой переменная цикла **variable_id** неявно объявляется как числовая переменная. Поэтому любое использование переменной **variable_id** для внешней области видимости скрыто в операторе **REPEAT**, и значение переменной цикла недоступно вне оператора **REPEAT**.

13.9.2 Управляющее условие WHILE

Управляющее условие **WHILE** обеспечивает инициализацию и продолжение исполнения тела оператора **REPEAT**, пока значением управляющего выражения является **TRUE**. Значение управляющего выражения вычисляется перед каждой итерацией.

Если управляющее условие присутствует **WHILE**, и значением управляющего выражения является **FALSE**, **UNKNOWN** или неопределенность (**?**), то тело оператора **REPEAT** не исполняется.

Синтаксис:

```
339 while_control = WHILE logical_expression .
254 logical_expression = expression .
```

Правила и ограничения:

a) элемент **logical_expression** должен иметь значение типа **LOGICAL**;

b) элемент **logical_expression** вычисляют заново в начале каждой итерации.

13.9.3 Управляющее условие UNTIL

Управляющее условие **UNTIL** обеспечивает продолжение выполнения тела оператора **REPEAT** до тех пор, пока управляющее выражение не примет значение **TRUE**. Значение выражения должно вычисляться после каждой итерации.

Если управляющее условие **UNTIL** является единственным управляющим условием, то всегда должна быть исполнена, по крайней мере, одна итерация.

Синтаксис:

```
335 until_control = UNTIL logical_expression .
254 logical_expression = expression .
```

Правила и ограничения:

a) элемент **logical_expression** должен иметь значение типа **LOGICAL**;

b) элемент **logical_expression** вычисляют заново в конце каждой итерации.

13.10 Оператор RETURN

Оператор возврата **RETURN** завершает исполнение функции или процедуры. В функции оператор **RETURN** должен определять выражение. Значение, полученное при вычислении данного выражения, является результатом функции и возвращается в точку вызова. Выражение должно быть совместимо по присваиванию с объявленным типом данных, возвращаемым функцией. В процедуре оператор **RETURN** не должен определять выражение.

Синтаксис:

```
290 return_stmt = RETURN [ '(' expression ')' ] ';' .
```

Ограничение: оператор **RETURN** должен присутствовать только в функциях или процедурах.

Пример — Допустимые операторы **RETURN.**

```
RETURN(50);              (* возврат из функции *)
RETURN(work_point);      (* возврат из функции *)
RETURN;                  (* возврат из процедуры *)
```

13.11 Оператор SKIP

Оператор **SKIP** вызывает немедленный переход в конец тела оператора **REPEAT**, в котором он присутствует. Затем вычисляют значения управляющих условий в соответствии с 13.9.

Синтаксис:
308 skip_stmt = SKIP ';' .

Ограничение: оператор **SKIP** должен встречаться только в области видимости оператора **REPEAT**.

***Пример — Оператор* SKIP *передает управление оператору* END_REPEAT, *который инициирует вычисление значения управляющего условия* UNTIL.**

```
REPEAT UNTIL (a=1);
  ...
  IF (a < 0) THEN
    SKIP;
  END_IF;
  ... -- эти операторы будут пропущены, если a < 0
END_REPEAT;
```

14 Встроенные константы

В языке EXPRESS имеется несколько встроенных констант, которые определены в настоящем разделе.

Примечание — Считается, что встроенные константы имеют точные значения, даже если такое значение не может быть представлено на компьютере.

14.1 Константа e

CONST_E является константой типа **REAL**, представляющей математическое значение числа **e**, являющегося основанием функции натурального логарифма (*ln*). Значение данной константы задается следующей математической формулой

$$e = \sum_{i=0}^{\infty} i!^{-1} .$$

14.2 Неопределенность

Символ неопределенности (**?**) обозначает неоднозначное значение. Он совместим со всеми типами данных.

Примечание — Наиболее часто неопределенность (**?**) используется в качестве указателя верхней границы пакета, списка или набора. Это обозначает, что размер агрегированного значения, определенного агрегированным типом данных, является неограниченным.

14.3 Константа FALSE

Константа **FALSE** является константой типа **LOGICAL**, представляющей логическое понятие «ложь». Данная константа совместима с типами данных **BOOLEAN** и **LOGICAL**.

14.4 Константа PI

Константа **PI** является константой типа **REAL**, представляющей математическую величину π, представляющую отношение длины окружности к ее диаметру.

14.5 SELF

Ключевое слово **SELF** обозначает ссылку на текущий экземпляр объекта или значение типа данных. Ключевое слово **SELF** может присутствовать в объявлении объекта, объявлении типа данных или конструкторе объекта.

Примечание — **SELF** не является константой, но играет роль константы в любом контексте, в котором она может использоваться.

14.6 Константа TRUE

Константа **TRUE** является константой типа **LOGICAL**, представляющей логическое понятие «истина». Данная константа совместима с типами данных **BOOLEAN** и **LOGICAL**.

14.7 Константа UNKNOWN

Константа **UNKNOWN** является константой типа **LOGICAL**, обозначающей недостаточность имеющейся информации для оценки логического условия. Данная константа совместима с типом данных **LOGICAL**, но несовместима с типом данных **BOOLEAN**.

15 Встроенные функции

Предполагается, что все функции (и математические операции вообще) вычисляются с точными результатами. Все встроенные функции возвращают неопределенный (**?**) результат, если им был передан параметр с неопределенным (**?**) значением, за исключением случаев, когда это в явном виде не устанавливается в определении функции.

Для каждой встроенной функции установлен прототип с целью демонстрации типов данных формальных параметров и результата.

15.1 Арифметическая функция ABS

FUNCTION ABS (V:NUMBER) : NUMBER;

Функция **ABS** возвращает абсолютное значение числа.

Параметр: **V** — число.

Результат: абсолютное значение **V**. Тип данных результата идентичен типу данных параметра **V**.

Пример **— ABS (–10) – –> 10**

15.2 Арифметическая функция ACOS

FUNCTION ACOS (V:NUMBER) : REAL;

Функция **ACOS** возвращает величину угла, заданного значением косинуса.

Параметр: **V** — число, представляющее значение косинуса угла.

Результат: значение угла в радианах ($0 \le$ результат $\le \pi$), косинус которого равен значению **V**.

Условие: **–1.0 ≤ V ≤ 1.0**.

Пример **— ACOS (0.3) – –> 1.266103. . .**

15.3 Арифметическая функция ASIN

FUNCTION ASIN (V:NUMBER) : REAL;

Функция **ASIN** возвращает величину угла, заданного значением синуса.

Параметр: **V** — число, представляющее значение синуса угла.

Результат: значение угла в радианах ($-\pi/2 \le$ результат $\le \pi/2$), синус которого равен значению **V**.

Условие: **–1.0 ≤ V ≤ 1.0**.

Пример **— ASIN(0.3) – –> 3.04692. . .e –1**

15.4 Арифметическая функция ATAN

FUNCTION ATAN (V1:NUMBER; V2:NUMBER) : REAL;

Функция **ATAN** возвращает величину угла, заданного значением тангенса **V**, где **V** задано выражением **V = V1/V2**.

Параметры:

a) **V1** — число;

b) **V2** — число.

Результат: значение угла в радианах ($-\pi/2 \le$ результат $\le \pi/2$), тангенс которого равен значению **V**. Если значение параметра **V2** равно нулю, то результат равен $\pi/2$ или $-\pi/2$ в зависимости от знака **V1**.

Условие: **V1** и **V2** не должны одновременно иметь нулевое значение.

Пример **— ATAN(–5.5, 3.0) – –> –1.071449. . .**

15.5 Двоичная функция BLENGTH

FUNCTION BLENGTH (V:BINARY) : INTEGER;

Функция **BLENGTH** возвращает число битов в двоичном числе.

Параметр: **V** — двоичное число.

Результат: возвращаемым значением является реальное число битов в переданном функции двоичном числе.

Пример — Использование функции **BLENGTH:**

```
LOCAL
   n : NUMBER;
   x : BINARY := %01010010 ;
END_LOCAL;
   . . .
   n := BLENGTH ( x );  – – n присваивается значение 8
```

15.6 Арифметическая функция COS

FUNCTION COS (V:NUMBER) : REAL;

Функция **COS** возвращает значение косинуса угла.

Параметр: **V** — число, представляющее значение угла в радианах.

Результат: косинус угла **V** (–1.0 ≤ результат ≤ 1.0).

Пример **— COS (0.5) – –> 8.77582. . .E–1**

15.7 Универсальная функция EXISTS

FUNCTION EXISTS (V : GENERIC) : BOOLEAN;

Функция **EXISTS** возвращает значение **TRUE**, если у входного параметра существует значение, или значение **FALSE**, если у входного параметра не существует значения. Функция **EXISTS** полезна для проверки, задано ли значение для необязательных (**OPTIONAL**) атрибутов или инициализированы ли переменные.

Параметр: **V** — выражение, результат которого может иметь любой тип данных.

Результат: **TRUE** или **FALSE**, в зависимости от того, имеет ли **V** конкретное или неопределенное (**?**) значение.

Пример **— IF EXISTS (a) THEN . . .**

15.8 Арифметическая функция EXP

FUNCTION EXP (V : NUMBER) : REAL;

Функция **EXP** возвращает число e (основание системы натуральных логарифмов), возведенное в степень **V**.

Параметры: **V** — число.

Результат: значение e^{V}.

Пример **— EXP (10) – –> 2.202646. . .E+4**

15.9 Универсальная функция FORMAT

FUNCTION FORMAT(N : NUMBER; F : STRING) : STRING;

Функция **FORMAT** возвращает форматированное строковое представление числа.

Параметры:

а) **N** — число (целое или действительное);

б) **F** — строка, содержащая команды форматирования.

Результат: строковое представление числа **N**, отформатированное в соответствии с **F**. При необходимости строковое представление округляется.

Строка форматирования содержит специальные символы, определяющие форму отображения результата. Строка форматирования может быть представлена тремя способами:

a) строка форматирования может задавать символьное описание представления результата;

b) строка форматирования может задавать описание шаблона представления результата;

c) если строка форматирования пуста, то производится стандартное представление результата.

15.9.1 Символьное представление

Общая форма символьного формата имеет вид: **[sign] width [.decimals] type**.

a) Элемент **sign** определяет представление знака числа. Если элемент **sign** не задан или задан знаком минус (**–**), то первым возвращаемым символом будет минус для отрицательных чисел и пробел для положительных чисел (включая ноль). Если элемент **sign** задан знаком плюс (**+**), то первым возвращаемым символом будет минус для отрицательных чисел, плюс — для положительных чисел и пробел — для нуля.

b) Элемент **width** задает общее число символов в возвращаемой строке. Он должен быть целым числом больше двух. Если элемент **width** задан с предшествующим нулем, то возвращаемая строка будет содержать предшествующие нули, в противном случае предшествующие нули опускаются. Если для форматируемого числа требуется больше символов, чем задано элементом **width**, то возвращается строка с необходимым числом символов.

c) Элемент **decimals** задает число цифр в возвращаемой строке справа от десятичной точки. Число данный элемент задан, то он должен быть положительным целым числом. Если элемент **decimals** не задан, то в возвращаемой строке не будет десятичной точки и следующих за ней цифр.

d) Элемент **type** является буквой, определяющей вид числа, представленного в возвращаемой строке:

1) если в качестве элемента **type** задана буква **I**, то результат должен быть представлен в виде целого числа; при этом:

- элемент **decimals** не должен быть задан,
- значение элемента **width** должно быть не менее двух,

2) если в качестве элемента **type** задана буква **F**, то результат должен быть представлен в виде действительного числа с фиксированной десятичной точкой; при этом:

- значение элемента **decimals**, если он задан, должно быть не менее единицы,
- если элемент **decimals** не задан, то используется его значение по умолчанию равное двум,
- значение элемента **width** должно быть не менее четырех,

3) если в качестве элемента **type** задана буква **E**, то результат должен быть представлен в виде действительного числа в экспоненциальной форме, при этом:

- элемент **decimals** должен быть задан обязательно,
- значение элемента **decimals** должно быть не менее единицы,
- значение элемента **width** должно быть не менее семи,
- если в элементе **width** задан предшествующий ноль, то первыми двумя символами мантиссы будут **0.**,
- экспоненциальная часть должна содержать, по меньшей мере, два символа с обязательным знаком,
- отображаемый символ '**E**' должен быть прописной буквой (символом верхнего регистра).

Примечание — В таблице 20 показано, как форматирование влияет на вид отображения разных значений.

Таблица 20 — Пример влияния символьного форматирования

Число	Формат	Отображение	Комментарий
10	**+7I**	**' +10'**	Нули опущены
10	**+07I**	**'+000010'**	Нули не опущены
10	**10.3E**	**'1.000E+01'**	
123.456789	**8.2F**	**' 123.46'**	
123.456789	**8.2E**	**'1.23E+02'**	
123.456789	**08.2E**	**'0.12E+02'**	Влияет предшествующий ноль
9.876E123	**8.2E**	**'9.88E+123'**	Экспоненциальная часть содержит три символа, значение **width** игнорируется
32.777	**6I**	**' 33'**	Округлено

15.9.2 Представление шаблоном

При форматировании посредством шаблона, каждый символ шаблона соответствует символу в возвращаемой строке. Используемые в шаблоне символы представлены в таблице 21.

Таблица 21 — Символы форматирования шаблоном

Символ	Значение
# (решетка)	Представляет цифру
, (запятая)	Разделитель
. (точка)	Разделитель
+ – (плюс и минус)	Представляет знак
( ) (круглые скобки)	Представляет отрицание

Разделители '.' и ',' используются следующим образом:

- если запятая встречается в строке форматирования до точки, то запятая представляет символ группирования, а точка представляет десятичный символ;
- если точка встречается в строке форматирования до запятой, то точка представляет символ группирования, а запятая — десятичный символ;
- если в строке форматирования присутствует один разделитель, то он представляет десятичный символ.

Все остальные символы отображаются без изменения.

Примечание — В таблице 22 показано, как форматирование влияет на вид отображения разных значений.

Таблица 22 — Пример влияния форматирования шаблоном

Число	Формат	Отображение	Комментарий
10	**###**	**' 10 '**	
10	**(###)**	**' 10 '**	Круглые скобки игнорируются
–10	**(###)**	**' (10) '**	
7123.456	**###,###.##**	**' 7,123.46 '**	Нотация США
7123.456	**###.###,##**	**' 7.123,46 '**	Европейская нотация

15.9.3 Стандартное представление

Стандартным представлением для целых чисел является **'7I'**. Стандартным представлением для действительных чисел является **'10.1E'**. Символьные представления форматов чисел определены в 15.9.1.

15.10 Арифметическая функция HIBOUND

FUNCTION HIBOUND (V:AGGREGATE OF GENERIC) : INTEGER;

Функция **HIBOUND** возвращает объявленный верхний индекс объекта типа **ARRAY** или объявленную верхнюю границу объекта типа **BAG**, **LIST** или **SET**.

Параметр: **V** — агрегированное значение.

Результат:

a) если типом данных **V** является **ARRAY**, то возвращаемым значением является объявленный верхний индекс;

b) если типом данных **V** является **BAG**, **LIST** или **SET**, то возвращаемым значением является объявленная верхняя граница; если границы не объявлены или верхняя граница объявлена неопределенной (**?**), то возвращается неопределенное (**?**) значение.

Пример — Использование функции* HIBOUND *для вложенных агрегированных значений:

```
LOCAL
  a : ARRAY[–3:19] OF SET[2:4] OF LIST [0:?] OF INTEGER;
  h1, h2, h3 : INTEGER;
END_LOCAL;
  . . .
  a[–3] [1] [1] := 2;    – – помещает значение в список
  . . .
  h1 := HIBOUND(a);          – – =19  (верхний индекс массива)
  h2 := HIBOUND(a[–3]);      – – = 4  (верхняя граница набора)
  h3 := HIBOUND(a[–3] [1]); – – = ?  (верхняя граница списка (не ограничен))
```

15.11 Арифметическая функция HIINDEX

FUNCTION HIINDEX (V:AGGREGATE OF GENERIC) : INTEGER;

Функция **HIINDEX** возвращает верхний индекс объекта типа **ARRAY** или число элементов в объекте типа **BAG**, **LIST** или **SET**.

Параметр: **V** — агрегированное значение.

Результат:

a) если типом данных **V** является **ARRAY**, то возвращаемым значением является объявленный верхний индекс;

b) если типом данных **V** является **BAG**, **LIST** или **SET**, то возвращаемым значением является фактическое число элементов в агрегированном значении.

Пример — Использование функции **HIINDEX** ***для вложенных агрегированных значений:***

```
LOCAL
   a : ARRAY [–3 : 19] OF SET[2 : 4] OF LIST[0 : ?] OF INTEGER;
   h1, h2, h3 : INTEGER;
END_LOCAL;
   a[–3][1][1] := 2;              – – помещает значение в список
   h1 := HIINDEX(a);              – – = 19 (верхний индекс массива)
   h2 := HIINDEX(a[–3]);          – – =  1 (размер набора) – – это неверно для
                                                            – – заданных границ
                                                            – – набора
h3 := HIINDEX(a[–3] [1]); – – = 1 (размер списка)
```

15.12 Строковая функция LENGTH

FUNCTION LENGTH (V : STRING) : INTEGER;

Функция **LENGTH** возвращает число символов в строке.

Параметр: **V** — строковое значение.

Результат: возвращаемым значением является число символов в строке; возвращаемое значение должно быть больше или равно нулю.

Пример — Использование функции **LENGTH:**

```
LOCAL
   n : NUMBER;
   x1 : STRING := 'abc';
   x2 : STRING := "0000795E00006238";
END_LOCAL;
   . . .
   n := LENGTH ( x1 ); – – n присваивается значение 3
   n := LENGTH ( x2 ); – – n присваивается значение 2
```

15.13 Арифметическая функция LOBOUND

FUNCTION LOBOUND (V : AGGREGATE OF GENERIC) : INTEGER;

Функция **LOBOUND** возвращает объявленный нижний индекс объекта типа **ARRAY** или объявленную нижнюю границу объекта типа **BAG**, **LIST** или **SET**.

Параметр: **V** — агрегированное значение.

Результат:

a) если типом данных **V** является **ARRAY**, то возвращаемым значением является объявленный нижний индекс;

b) если типом данных **V** является **BAG**, **LIST** или **SET**, то возвращаемым значением является объявленная нижняя граница; если нижняя граница не объявлена, то возвращается ноль (**0**).

Пример — Использование функции **LOBOUND** ***для вложенных агрегированных значений:***

```
LOCAL
   a : ARRAY [–3 : 19] OF SET [2 : 4] OF LIST [0 : ?] OF INTEGER;
   h1, h2, h3 : INTEGER;
END_LOCAL;
   . . .
   h1 := LOBOUND (a);             - - = –3 (нижний индекс массива)
   h2 := LOBOUND (a[–3]);         - - =  2 (нижняя граница набора)
   h3 := LOBOUND (a[–3] [1]);     - - =  0 (нижняя граница списка)
```

15.14 Арифметическая функция LOG

FUNCTION LOG (V : NUMBER) : REAL;

Функция **LOG** возвращает натуральный логарифм числа.

Параметр: **V** — число.

Результат: действительное число, являющееся натуральным логарифмом **V**.

Условие: **V** > 0.0

Пример — **LOG (4.5) – –> 1.504077. . .E0**

15.15 Арифметическая функция LOG2

FUNCTION LOG2 (V : NUMBER) : REAL;

Функция **LOG2** возвращает логарифм числа по основанию два.

Параметр: **V** — число.

Результат: действительное число, являющееся логарифмом **V** по основанию два.

Условие: **V** > 0.0

Пример **— LOG2 (8) – –> 3.00. . .E0**

15.16 Арифметическая функция LOG10

FUNCTION LOG10 (V : NUMBER) : REAL;

Функция **LOG 10** возвращает десятичный логарифм числа.

Параметр: **V** — число.

Результат: действительное число, являющееся десятичным логарифмом **V**.

Условие: **V** > 0.0

Пример **— LOG10 (10) – –> 1.00. . .E0**

15.17 Арифметическая функция LOINDEX

FUNCTION LOINDEX (V : AGGREGATE OF GENERIC) : INTEGER;

Функция **LOINDEX** возвращает нижний индекс агрегированного значения.

Параметр: **V** — агрегированное значение.

Результат:

a) если типом данных **V** является **ARRAY**, то возвращаемым значением является объявленный нижний индекс;

b) если типом данных **V** является **BAG**, **LIST** или **SET**, то возвращаемым значением является единица (**1**).

Пример — Использование функции **LOINDEX** ***для вложенных агрегированных значений:***

```
LOCAL
   a : ARRAY[–3:19] OF SET[2:4] OF LIST[0 : ?] OF INTEGER;
   h1, h2, h3 : INTEGER;
END_LOCAL;
   . . .
   h1 := LOINDEX(a);          – – =–3  (нижний индекс массива)
   h2 := LOINDEX(a[–3]);      – – =  1  (для набора)
   h3 := LOINDEX(a[–3] [1]);  – – =  1  (для списка)
```

15.18 Функция пустого значения NVL

FUNCTION NVL (V : GENERIC : GEN1; SUBSTITUTE : GENERIC : GEN1) : GENERIC : GEN1;

Функция **NVL** возвращает исходное значение либо альтернативное значение, если входной параметр имеет неопределенное (**?**) значение.

Параметры:

a) **V** — выражение любого типа данных;

b) **SUBSTITUTE** — выражение, которое не должно иметь неопределенное (**?**) значение.

Результат: если **V** не имеет неопределенного (**?**) значение, то возвращается значение **V**. В противном случае возвращается **SUBSTITUTE**.

Пример — Функция **NVL** ***используется, чтобы подставить ноль (0.0) в качестве значения Z, если Z имеет неопределенное (?) значение:***

```
ENTITY unit_vector;
   x, y : REAL;
   z : OPTIONAL REAL;
WHERE
   x**2 + y**2 + NVL(z, 0.0)**2 = 1.0;
END_ENTITY;
```

15.19 Арифметическая функция ODD

FUNCTION ODD (V : INTEGER) : LOGICAL;

Функция **ODD** возвращает значение **TRUE** или **FALSE**, в зависимости от того, является ли заданное число нечетным или четным.

Параметр: **V** — целое число.

Результат: если **V MOD 2 = 1**, то возвращается **TRUE**, в противном случае, возвращается **FALSE**.

Условие: ноль не считается нечетным числом.

Пример — **ODD (121) – –> TRUE**

15.20 Универсальная функция ROLESOF

FUNCTION ROLESOF (V : GENERIC_ENTITY) : SET OF STRING;

Функция **ROLESOF** возвращает набор строк, содержащих полные уточненные имена ролей, исполняемых указанным экземпляром объекта. Полное уточненное имя определяется как имя атрибута, уточненное именем схемы и объекта, в которых объявлен данный атрибут (то есть '**SCHEMA.ENTITY.ATTRIBUTE**').

Параметр: **V** — любой экземпляр объектного типа данных.

Результат: набор строковых значений (на верхнем регистре), содержащих полные уточненные имена атрибутов экземпляров объектов, использующих экземпляр **V**.

Если именованный тип данных импортирован посредством операторов **USE** или **REFERENCE**, то возвращаются также имя исходной схемы и имя именованного типа данных в этой схеме, если имело место переименование. Поскольку операторы **USE** могут быть связаны в цепочку, то возвращаются имена всех связанных схем и имена именованного типа данных во всех схемах.

Пример — Данный пример демонстрирует возможность использования точки в качестве центра окружности. Функция **ROLESOF** ***определяет, какие роли экземпляр объекта исполняет фактически.***

```
SCHEMA that_schema;
ENTITY point;
   x, y, z : REAL;
END_ENTITY;
ENTITY line;
   start,
   end : point;
END_ENTITY;
END SCHEMA;
SCHEMA this_schema;
USE FROM that_schema (point, line);
CONSTANT
   origin : point := point(0.0, 0.0, 0.0);
END_CONSTANT;
ENTITY circle;
   center : point;
   axis   : vector;
   radius : REAL;
END_ENTITY;
   . . .
LOCAL
   p : point := point(1.0, 0.0, 0.0);
   c : circle := circle(p, vector(1, 1, 1), 1.0);
   l : line := line(p, origin);
END_LOCAL;
   . . .
IF 'THIS_SCHEMA.CIRCLE.CENTRE' IN ROLESOF(p) THEN  – – true
   . . .
IF 'THIS_SCHEMA.LINE.START' IN ROLESOF(p) THEN     – – true
   . . .
IF 'THAT_SCHEMA.LINE.START' IN ROLESOF(p) THEN     – – true
   . . .
IF 'THIS SCHEMA.LINE.END' IN ROLESOF(p) THEN       – – false
```

15.21 Арифметическая функция SIN

FUNCTION SIN (V : NUMBER) : REAL;

Функция **SIN** возвращает значение синуса угла.

Параметр: **V** — число, представляющее значение угла в радианах.

Результат: синус угла **V** ($-1.0 \le$ результат ≤ 1.0).

Пример — **SIN (PI) – –> 0.0**

15.22 Агрегированная функция SIZEOF

FUNCTION SIZEOF (V : AGGREGATE OF GENERIC) : INTEGER;

Функция **SIZEOF** возвращает число элементов в агрегированном значении.

Параметр: **V** — агрегированное значение.

Результат:

a) если типом данных **V** является **ARRAY**, то возвращаемым значением является объявленное число элементов в данном агрегированном типе данных;

b) если типом данных **V** является **BAG**, **LIST** или **SET**, то возвращаемым значением является фактическое число элементов в агрегированном значении.

Пример — Использование функции **SIZEOF:**

```
LOCAL
   n : NUMBER;
   y : ARRAY[2:5] OF b;
END_LOCAL;
   ...
   n := SIZEOF (y);  -- n присваивается значение 4
```

15.23 Арифметическая функция SQRT

FUNCTION SQRT (V : NUMBER) : REAL;

Функция **SQRT** возвращает неотрицательное значение квадратного корня числа.

Параметр: **V** — любое неотрицательное число.

Результат: неотрицательное значение квадратного корня числа **V**.

Условие: **V** ≥ 0.0

Пример **— SQRT (121) – –> 11.0**

15.24 Арифметическая функция TAN

FUNCTION TAN (V : NUMBER) : REAL;

Функция **TAN** возвращает значение тангенса угла.

Параметр: **V** — число, представляющее значение угла в радианах.

Результат: тангенс угла. Если угол равен $n\pi/2$, где n — нечетное целое число, то возвращается неопределенное (**?**) значение.

Пример **— TAN (0.0) – –> 0.0**

15.25 Универсальная функция TYPEOF

FUNCTION TYPEOF (V : GENERIC) : SET OF STRING;

Функция **TYPEOF** возвращает набор строк, содержащих имена всех типов данных, к которым принадлежит данный параметр. За исключением простых типов данных (**BINARY**, **BOOLEAN**, **INTEGER**, **LOGICAL**, **NUMBER**, **REAL** и **STRING**) и агрегированных типов данных (**ARRAY**, **BAG**, **LIST**, **SET**), данные имена уточняются именем схемы, содержащей определение данного типа данных.

Примечание — Основным назначением данной функции является проверка, может ли данное значение (переменной или атрибута) использоваться для определенной цели, например, чтобы убедиться в совместимости по присваиванию двух значений. Данная функция может также использоваться, если разные подтипы или конкретизации заданного типа данных должны по-разному трактоваться в некотором контексте.

Параметр: **V** — значение любого типа данных.

Результат: содержимым возвращаемого набора строковых значений являются имена (на верхнем регистре) всех типов данных, к которым принадлежит **V**. Данные имена уточняются именем схемы, содержащей определение данного типа данных (**'SCHEMA.TYPE'**), если они не относятся к простому или агрегированному типу данных. Возвращаемый набор строковых значений может быть определен посредством следующего алгоритма (данный алгоритм приводится здесь в целях пояснения, а не в качестве предписания какого-либо конкретного вида реализации):

a) возвращаемый набор строковых значений набор инициализируется включением как имени типа данных, к которому принадлежит **V**, так и именем типа данных, которое представляет экземпляр **V** (если они различаются), включая имена их схем, если данные типы данных являются именованными типами данных; при этом применяются следующие правила:

Примечание — Если фактический параметр, передаваемый функции **TYPEOF**, был формальным параметром некоторой вычисляемой функции, то «типом данных, к которому принадлежит **V** (в соответствии

с объявлением)» является тип данных, объявленный для исходного фактического параметра, или тип данных результата вычисления выражения, определяющего фактический параметр, в соответствии с разделом 12, а не тип данных, объявленный для формального параметра, вместо которого он подставлен,

1) если **V** является агрегированным значением, то имя типа данных является просто именем агрегированного типа данных (**ARRAY**, **BAG**, **LIST**, **SET**), а не каким-либо другим.

2) если **V** является перечисляемым типом данных, базирующимся на другом перечисляемом типе данных, то добавляют имена перечисляемых типов данных, получаемых при прослеживании взаимосвязей **BASED_ON**, начиная от данного перечисляемого типа данных,

3) если **V** является наращиваемым перечисляемым типом данных, то рекурсивно добавляются имена перечисляемых типов данных, являющихся расширениями **V**.

Примечание — Две последние из перечисленных выше позиций справедливы для расширяемого перечисляемого типа данных, который базируется на другом перечисляемом типе данных,

4) если **V** имеет неопределенное (**?**) значение, то возвращается пустой набор типа данных **SET**;

b) повторяют следующие действия до тех пор, пока возвращаемый набор не перестанет расширяться:

1) выполняют следующие действия для всех имен в возвращаемом наборе:

- если текущее имя является именем простого типа данных, то пропускают,
- если текущее имя является именем агрегированного типа данных (**ARRAY**, **BAG**, **LIST**, **SET**), то пропускают,
- если текущее имя является именем перечисляемого типа данных, то пропускают,
- если текущее имя является именем выбираемого типа данных, то к возвращаемому набору добавляются имена всех типов данных (с именем схемы) из списка выбора, которые действительно конкретизируются посредством **V** (данных имен может быть несколько, так как список выбора может содержать имена типов данных, являющихся совместимыми подтипами общего супертипа или конкретизациями одного общего обобщенного типа),
- если текущее имя является именем любого другого вида определенного типа данных, то имя типа данных, на который ссылается определение данного типа данных, включая (при необходимости) имя схемы, добавляется к возвращаемому набору. Если ссылка делается на агрегированный тип данных, то добавляется имя данного агрегированного типа данных,
- если текущее имя является именем объекта, то к возвращаемому набору добавляются имена всех тех подтипов (включая, при необходимости, имя схемы), которые действительно конкретизируются посредством **V**,

2) выполняют следующие действия для всех имен в возвращаемом наборе:

- если текущее имя является именем подтипа, то к возвращаемому набору добавляются имена всех его супертипов,
- если текущее имя является именем конкретизации, то к возвращаемому набору добавляются имена всех ее обобщений,

3) выполняют следующие действия для всех имен в возвращаемом наборе и для каждого типа данных **SELECT**, у которого текущее имя присутствует в списке выбора:

- добавляют имя выбираемого типа данных к списку,
- если выбираемый тип данных базируется на другом выбираемом типе данных, то добавляют имена выбираемых типов данных, получаемых при прослеживании взаимосвязей **BASED_ON**, начиная с текущего выбираемого типа данных,
- если выбираемый тип данных является наращиваемым выбираемым типом данных, то рекурсивно добавляют имена выбираемых типов данных, являющихся расширениями текущего выбираемого типа данных,

4) выполняют следующие действия для всех имен в возвращаемом наборе: если текущее имя импортировано в схему посредством операторов **USE** или **REFERENCE**, то в возвращаемый набор добавляют имя из схемы, откуда был осуществлен импорт, уточненное именем данной схемы. Поскольку операторы **USE** могут быть связаны в цепочки, в возвращаемый набор также добавляют имена из всех связанных схем, уточненные именами соответствующих схем;

c) результатом функции является сформированный возвращаемый набор.

Если **V** имеет неопределенное (**?**) значение, то функция **TYPEOF** возвращает пустой набор.

Примечание — Функция **TYPEOF** завершает свою работу, когда встречается агрегированный тип данных. Функция не дает информации относительно базисного типа данных агрегированного значения. При

необходимости данная информация может быть получена при применении функции **TYPEOF** к допустимым элементам агрегированного значения.

Примеры

1 В контексте следующей схемы:

```
SCHEMA this_schema;
   TYPE
      mylist = LIST [1 : 20] OF REAL;
   END_TYPE;
   . . .
   LOCAL
      lst : mylist;
   END_LOCAL;
   . . .
END_SCHEMA;
```

следующие условия имеют значение **TRUE:**

```
TYPEOF (lst)      = [ 'THIS_SCHEMA.MYLIST', 'LIST' ]
TYPEOF (lst [17]) = [ 'REAL', 'NUMBER' ]
```

2 Действие операторов **USE** ***или*** **REFERENCE** ***показано на основе предыдущего примера:***

```
SCHEMA another_schema;
   REFERENCE FROM this_schema (mylist AS hislist);
   . . .
     lst : hislist;
   . . .
END_SCHEMA;
```

В данном контексте следующее выражение имеет значение **TRUE:**

```
TYPEOF (lst) = [ 'ANOTHER_SCHEMA.HISLIST', 'THIS_SCHEMA.MYLIST', 'LIST' ]
```

15.26 Универсальная функция USEDIN

```
FUNCTION USEDIN (T : GENERIC ; R : STRING) : BAG OF GENERIC_ENTITY;
```

Функция **USEDIN** возвращает все экземпляры объекта, в которых используется указанный экземпляр объекта в указанной роли.

Параметры:

a) **T** — любой экземпляр любого объектного типа данных.

b) **R** — строка, содержащая полностью уточненное имя атрибута (роли), в соответствии с 15.20.

Результат: все экземпляры объекта, в которых используется указанный экземпляр объекта в указанной роли, возвращаются в форме пакета (типа данных **BAG**).

Если экземпляр **T** не исполняет никаких ролей или роль **R** не указана, то возвращается пустой пакет.

Если **R** представлен пустой строкой, то документируется каждое использование **T**. Проверяют все взаимосвязи, направленные к **T**. Если взаимосвязь исходит от атрибута с именем **R**, то экземпляр объекта, содержащий данный атрибут, добавляется к возвращаемому пакету. Отметим, что, если **T** не используется, то возвращается пустой пакет.

Пример — Данный пример показывает, как может быть использовано правило для проверки того, что должна существовать точка (объект **point)** ***в начале координат, используемая как центр окружности. Заметим, что в данном примере выражение*** **QUERY** ***(см. 12.6.7) используется в качестве параметра функции*** **SIZEOF.**

```
ENTITY point;
   x, y, z : REAL;
END_ENTITY;
ENTITY circle;
   centre : point;
   axis   : vector;
   radius : REAL;
END_ENTITY; . . .
( *Правило "example" находит каждую точку, используемую в качестве центра окружности, и затем
   проверяет, что, по крайней мере, одна из этих точек расположена в начале координат *)
. . .
RULE example FOR (point);
LOCAL
   circles : SET OF circle := [ ]; -- пустой набор окружностей
END_LOCAL;
```

```
REPEAT i := LOINDEX(point) TO HIINDEX(point);
  circles := circles +
             USEDIN (point [i], 'THIS_SCHEMA.CIRCLE.CENTRE' );
END_REPEAT;
WHERE R1 : SIZEOF (
             QUERY(                                    -- начало запроса
                at zero <* circles |                   -- взять все точки
                (at_zero.centre = point (0.0, 0.0, 0.0)) -- в 0, 0, 0
             )
          ) >= 1;                                      -- по крайней мере, одна
END_RULE;
```

15.27 Арифметическая функция VALUE

FUNCTION VALUE (V : STRING) : NUMBER;

Функция **VALUE** возвращает число, представленное строкой.

Параметр: **V** — строка, содержащая действительный или целочисленный литерал (см. 7.5).

Результат: число, соответствующее содержанию строки. Если строка не может быть интерпретирована как действительный или целочисленный литерал, то возвращается неопределенное (**?**) значение.

Пример — В данном примере представлены результаты вызова функции* VALUE *с разными параметрами:

```
VALUE ( '1.234' ) --> 1.234 (действительное число)
VALUE ( '20' )    --> 20 (целое число)
VALUE ( 'abc' )   --> ? (неопределенное значение)
```

15.28 Функция принадлежности VALUE_IN

FUNCTION VALUE_IN (C : AGGREGATE OF GENERIC : GEN; V : GENERIC : GEN) : LOGICAL;

Функция **VALUE_IN** возвращает логическое значение в зависимости от того, принадлежит ли конкретное значение к агрегированной структуре.

Параметры:

a) **C** — агрегированная структура любого типа;

b) **V** — выражение, совместимое по присваиванию с базисным типом **C**.

Результат:

a) если **V** или **C** имеет неопределенное (**?**) значение, то возвращается значение **UNKNOWN**;

b) если значение какого-либо элемента из **C** равно значению **V**, то возвращается значение **TRUE**;

c) если какой-либо элемент из **C** имеет неопределенное (**?**) значение, то возвращается значение **UNKNOWN**;

d) во всех других случаях возвращается значение **FALSE**.

Пример — Следующая проверка обеспечивает, что существует хотя бы одна точка (объект* point*), расположенная в начале координат:

```
LOCAL
  points : SET OF point;
END_LOCAL;
...
IF VALUE_IN(points, point(0.0, 0.0, 0.0)) THEN ...
```

15.29 Функция уникальности VALUE_UNIQUE

FUNCTION VALUE_UNIQUE (V : AGGREGATE OF GENERIC) : LOGICAL;

Функция **VALUE_UNIQUE** возвращает логическое значение в зависимости от того, являются ли значения элементов агрегированной структуры уникальными.

Параметр: **V** — агрегированная структура любого типа данных.

Результат:

a) если **V** имеет неопределенное (**?**) значение, то возвращается значение **UNKNOWN**;

b) если значения любых двух элементов из **V** равны, то возвращается значение **FALSE**;

c) если любой элемент из **V** имеет неопределенное (**?**) значение, то возвращается значение **UNKNOWN**;

d) во всех других случаях возвращается значение **TRUE**.

Пример — Следующая проверка обеспечивает, что все точки (объекты* points*) в наборе расположены на разных позициях (по определению, эти точки различны, то есть их экземпляры уникальны).

```
IF VALUE_UNIQUE (points) THEN ...
```

16 Встроенные процедуры

В языке EXPRESS имеется две встроенные процедуры, используемые для управления списками. В данном разделе приведено описание этих процедур. Встроенные процедуры не исполняются, если им передается неопределенный (**?**) параметр, если только иное не определено в явном виде в описании процедуры.

Для каждой процедуры задан заголовок, чтобы показать типы данных формальных параметров.

16.1 Процедура INSERT

PROCEDURE INSERT (VAR L : LIST OF GENERIC : GEN; E : GENERIC : GEN; P : INTEGER);

Процедура **INSERT** вставляет элемент на заданную позицию в списке.

Параметры:

a) **L** — значение списка, в который должен быть вставлен элемент;

b) **E** — экземпляр, вставляемый в список **L**. **E** должен быть совместим с базисным типом **L**, как указано метками типов в заголовке процедуры;

c) **P** — целое число, задающее позицию в **L**, на которую должен быть вставлен элемент **E**.

Результат: список **L** модифицируется вставлением **E** в **L** на указанную позицию. **E** вставляется непосредственно после существующего элемента, расположенного на позиции **P**, если **P** = 0, то **E** становится первым элементом списка.

Условие: **0 ≤ P ≤ SIZEOF(L)**.

16.2 Процедура REMOVE

PROCEDURE REMOVE (VAR L : LIST OF GENERIC; P : INTEGER);

Процедура **REMOVE** удаляет элемент из заданной позиции в списке.

Параметры:

a) **L** — значение списка, из которого должен быть удален элемент;

b) **P** — целое число, задающее позицию в **L**, из которой должен быть удален элемент.

Результат: список **L** модифицируется удалением элемента, находящегося на заданной позиции **P**.

Условие: **1 ≤ P ≤ SIZEOF (L).**

Приложение А
(обязательное)

Синтаксис языка EXPRESS

В настоящем приложении определены лексические элементы языка и грамматические правила, которым данные элементы должны подчиняться.

Примечание — Прямое применение данного определения синтаксиса приведет к неоднозначности при построении синтаксических анализаторов. Данное определение разработано для представления информации, относящейся к использованию идентификаторов. Интерпретированные идентификаторы определяют лексические элементы, являющиеся ссылками на объявленные идентификаторы, и поэтому не должны трактоваться как простые идентификаторы **(simple_id).** Разработчик синтаксического анализатора должен создать таблицу соответствия или что-то подобное ей с тем, чтобы обеспечить разрешение ссылок на идентификаторы и получение требуемого ссылочного лексического элемента для проверки грамматических правил. Такой подход был применен, чтобы помочь разработчикам синтаксических анализаторов обеспечить отсутствие неоднозначности при использовании идентификаторов.

А.1 Лексические элементы

Приведенные ниже правила определяют лексические элементы, используемые в языке EXPRESS. За исключением случаев, когда это явно установлено в синтаксических правилах, никакие пробелы или комментарии не должны присутствовать в тексте, относящемся к отдельному синтаксическому правилу, представленному в А.1.1 — А.1.3 и А.1.5.

А.1.1 Ключевые слова

В настоящем подразделе установлены правила, используемые для представления ключевых слов языка EXPRESS.

Примечание — Здесь используется соглашение, установленное в 6.1, по которому каждое ключевое слово представляется синтаксическим правилом, содержащим в левой части данное ключевое слово, записанное с использованием символов верхнего регистра (прописных букв). Поскольку строковые литералы в синтаксических правилах являются независящими от регистра, данные ключевые слова могут задаваться в формальных спецификациях с использованием символов верхнего, нижнего или обоих регистров.

```
 0   ABS = 'abs'.
 1   ABSTRACT = 'abstract'.
 2   ACOS = 'acos'.
 3   AGGREGATE = 'aggregate'.
 4   ALIAS = 'alias'.
 5   AND = 'and'.
 6   ANDOR = 'andor'.
 7   ARRAY = 'array'.
 8   AS = 'as'.
 9   ASIN = 'asin'.
10   ATAN = 'atan'.
11   BAG = 'bag'.
12   BASED_ON = 'based_on'
13   BEGIN = 'begin'.
14   BINARY = 'binary'.
15   BLENGTH = 'blength'.
16   BOOLEAN = 'boolean'.
17   BY = 'by'.
18   CASE = 'case'.
19   CONSTANT = 'constant'.
20   CONST_E = 'const_e'.
21   COS = 'cos'.
22   DERIVE = 'derive'.
23   DIV = 'div'.
24   ELSE = 'else'.
25   END = 'end'.
26   END_ALIAS = 'end_alias'.
27   END_CASE = 'end_case'.
```

```
28    END_CONSTANT = 'end_constant'.
29    END_ENTITY = 'end_entity'.
30    END_FUNCTION = 'end_function'.
31    END_IF = 'end_if'.
32    END_LOCAL = 'end_local'.
33    END_PROCEDURE = 'end_procedure'.
34    END_REPEAT = 'end_repeat'.
35    END_RULE = 'end_rule'.
36    END_SCHEMA = 'end_schema'.
37    END_SUBTYPE_CONSTRAINT = 'end_subtype_constraint'
38    END_TYPE = 'end_type'.
39    ENTITY = 'entity'.
40    ENUMERATION = 'enumeration'.
41    ESCAPE = 'escape'.
42    EXISTS = 'exists'.
43    EXTENSIBLE = 'extensible'.
44    EXP = 'exp'.
45    FALSE = 'false'.
46    FIXED = 'fixed'.
47    FOR = 'for'.
48    FORMAT = 'format'.
49    FROM = 'from'.
50    FUNCTION = 'function'.
51    GENERIC = 'generic'.
52    GENERIC_ENTITY = 'generic_entity'.
53    HIBOUND = 'hibound'.
54    HIINDEX = 'hiindex'.
55    IF = 'if'.
56    IN = 'in'.
57    INSERT = 'insert'.
58    INTEGER = 'integer'.
59    INVERSE = 'inverse'.
60    LENGTH = 'length'.
61    LIKE = 'like'.
62    LIST = 'list'.
63    LOBOUND = 'lobound'.
64    LOCAL = 'local'.
65    LOG = 'log'.
66    LOG10 = 'log10'.
67    LOG2 = 'log2'.
68    LOGICAL = 'logical'.
69    LOINDEX = 'loindex'.
70    MOD = 'mod'.
71    NOT = 'not'.
72    NUMBER = 'number'.
73    NVL = 'nvl'.
74    ODD = 'odd'.
75    OF = 'of'.
76    ONEOF = 'oneof'.
77    OPTIONAL = 'optional'.
78    OR = 'or'.
79    OTHERWISE = 'otherwise'.
80    PI = 'pi'.
81    PROCEDURE = 'procedure'.
82    QUERY = 'query'.
83    REAL = 'real'.
84    REFERENCE = 'reference'.
85    REMOVE = 'remove'.
86    RENAMED = 'renamed'.
87    REPEAT = 'repeat'.
88    RETURN = 'return'.
89    ROLESOF = 'rolesof'.
```

```
RULE = 'rule'.
SCHEMA = 'schema'.
SELECT = 'select'.
SELF = 'self'.
SET = 'set'.
SIN = 'sin'.
SIZEOF = 'sizeof'.
SKIP = 'skip'.
SQRT = 'sqrt'.
STRING = 'string'.
SUBTYPE = 'subtype'.
SUBTYPE_CONSTRAINT = 'subtype_constraint'.
SUPERTYPE = 'supertype'
TAN = 'tan'.
THEN = 'then'.
TO = 'to'.
TOTAL_OVER = 'total_over'.
TRUE = 'true'.
TYPE = 'type'.
TYPEOF = 'typeof'.
UNIQUE = 'unique'.
UNKNOWN = 'unknown'.
UNTIL = 'until'.
USE = 'use'.
USEDIN = 'usedin'.
VALUE = 'value'.
VALUE_IN = 'value_in'.
VALUE_UNIQUE = 'value_unique'.
VAR = 'var'.
WHERE = 'where'.
WHILE = 'while'.
WITH = 'with'.
XOR = 'xor'.
```

А.1.2 Классы символов

Представленные ниже правила определяют различные классы символов, используемых при построении лексических элементов в А.1.3.

```
bit = '0' | '1'.
digit = '0'| '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9'.
digits = digit { digit } .
encoded_character = octet octet octet octet .
hex_digit = digit | 'a' | 'b' | 'c' | 'd' | 'e' | 'f'.
letter = 'a'| 'b'|'c' | 'd'| 'e'|'f' | 'g'| 'h'| 'i' | 'j'|
         'k'| 'l'| 'm'| 'n'|'o' |'p' |'q' | 'r' |'s' | 't'|
         'u'| 'v'| 'w' | 'x' |'y' | 'z' .
lparen_then_not_lparen_star = '('{'('} not_lparen_star
                              { not_lparen_star } .
not_lparen_star = not_paren_star |')'.
not_paren_star = letter | digit | not_paren_star_special .
not_paren_star_quote_special = '!' |'"' | '#'| '$'| '%'| '&'|
                               '+' | ',' | '-'| '.' | '/' | ':'|
                               ';' |'<'| '='|'>' | '?'|'@'|
                               '[' | '\' | ']' | '^' | '_'| '`'|
                               '{' | '|' | '}'| '~'.
not_paren_star_special = not_paren_star_quote_special |''''.
not_quote = not_paren_star_quote_special | letter | digit | '(' |
            ')' | '*'.
not_rparen_star = not_paren_star | '('.
octet = hex_digit hex_digit .
special = not_paren_star_quote_special | '(' | ')'| '*' | '"'/.
not_rparen_star_then_rparen = not_rparen_star { not_rparen_star }
                              ')'{')'}.
```

А.1.3 Лексические элементы

Представленные ниже правила определяют, как определенные комбинации символов интерпретируются в качестве лексических элементов языка.

```
139  binary_literal = '%' bit { bit } .
140  encoded_string_literal = '"' encoded_character {
                              encoded_character } '"' .
141  integer_literal = digits .
142  real_literal = nteger_literal |
                    ( digits '.' [ digits ] [ 'e' [ sign ] digits ] ) .
143  simple_id = letter { letter | digit | '_' } .
144  simple_string_literal = \q { ( \q \q ) | not_quote | \s | \x9 |
                             \xA | \xD } \q .
```

А.1.4 Комментарии

Представленные ниже правила определяют синтаксис комментариев в языке EXPRESS.

```
145  embedded_remark = '(*' [ remark_tag ] { ( not_paren_star {
                       not_paren_star } ) |
                       lparen_then_not_lparen_star | ( '*' { '*' } ) |
                       not_rparen_star_then_rparen | embedded_remark }
                       '*)' .
146  remark = embedded_remark | tail_remark .
147  remark_tag = '"' remark_ref { '.' remark_ref } '"' .
148  remark_ref = attribute_ref | constant_ref | entity_ref |
     enumeration_ref | function_ref | parameter_ref | procedure_ref |
     rule_label_ref | rule_ref | schema_ref | subtype_constraint_ref |
     type_label_ref | type_ref | variable_ref .
149  tail_remark = '--' [ remark_tag ] { \a | \s | \x9 | \xA | \xD }
                   \n .
```

А.1.5 Интерпретированные идентификаторы

Представленные ниже правила определяют идентификаторы, для которых установлен конкретный смысл (то есть они объявляются как типы, функции и т. д.).

Примечание — Предполагается, что идентификаторы, определенные данными правилами, распознаются реализацией. Способ, которым реализация получает эту информацию, не связан с определением языка. Одним из способов получения данной информации является многопроходный синтаксический анализ, когда на первом проходе собираются идентификаторы из их объявлений, а на последующих проходах появляется возможность отличить, например, **variable_ref** от **function_ref.**

```
150  attribute_ref = attribute_id .
151  constant_ref = constant_id .
152  entity_ref = entity_id .
153  enumeration_ref = enumeration_id .
154  function_ref = function_id .
155  parameter_ref = parameter_id .
156  procedure_ref = procedure_id .
157  rule_label_ref = rule_label_id .
158  rule_ref = rule_id .
159  schema_ref = schema_id .
160  subtype_constraint_ref = subtype_constraint_id .
161  type_label_ref = type_label_id .
162  type_ref = type_id .
163  variable_ref = variable_id .
```

А.2 Грамматические правила

Представленные ниже правила определяют, как рассмотренные выше лексические элементы могут объединяться в конструкции языка EXPRESS. Пробелы и/или комментарии могут помещаться между любыми двумя лексемами в данных правилах. Первичным синтаксическим правилом языка EXPRESS является **syntax.**

```
164  abstract_entity_declaration = ABSTRACT .
165  abstract_supertype = ABSTRACT SUPERTYPE ';' .
166  abstract_supertype_declaration = ABSTRACT SUPERTYPE [
     subtype_constraint ] .
167  actual_parameter_list = '(' parameter { ',' parameter } ')' .
168  add_like_op = '+' | '-' | OR | XOR .
169  aggregate_initializer = '[' [ element { ',' element } ] ']' .
```

```
aggregate_source = simple_expression .
aggregate_type = AGGREGATE [ ':' type_label ] OF parameter_type .
aggregation_types = array_type | bag_type | list_type | set_type .
algorithm_head = { declaration } [ constant_decl ] [ local_decl ] .
alias_stmt = ALIAS variable_id FOR general_ref { qualifier } ';'
             stmt { stmt } END_ALIAS ';' .
array_type = ARRAY bound_spec OF [ OPTIONAL ] [ UNIQUE ]
             instantiable_type .
assignment_stmt = general_ref { qualifier } ':=' expression ';' .
attribute_decl = attribute_id | redeclared_attribute .
attribute_id = simple_id .
attribute_qualifier = '.' attribute_ref .
bag_type = BAG [ bound_spec ] OF instantiable_type .
binary_type = BINARY [ width_spec ] .
boolean_type = BOOLEAN .
bound_1 = numeric_expression .
bound_2 = numeric_expression .
bound_spec = '[' bound_1 ':' bound_2 ']' .
built_in_constant = CONST_E | PI | SELF | '?' .
built_in_function = ABS | ACOS | ASIN | ATAN | BLENGTH | COS |
                    EXISTS | EXP | FORMAT | HIBOUND | HIINDEX |
                    LENGTH | LOBOUND | LOINDEX | LOG | LOG2 |
                    LOG10 | NVL | ODD | ROLESOF | SIN | SIZEOF |
                    SQRT | TAN | TYPEOF | USEDIN | VALUE |
                    VALUE_IN | VALUE_UNIQUE .
built_in_procedure = INSERT | REMOVE .
case_action = case_label { ',' case_label } ':' stmt .
case_label = expression .
case_stmt = CASE selector OF { case_action } [ OTHERWISE ':' stmt ]
            END_CASE ';' .
compound_stmt = BEGIN stmt { stmt } END ';' .
concrete_types = aggregation_types | simple_types | type_ref .
constant_body = constant_id ':' instantiable_type ':='
                expression ';' .
constant_decl = CONSTANT constant_body { constant_body }
                END_CONSTANT ';' .
constant_factor = built_in_constant | constant_ref .
constant_id = simple_id .
constructed_types = enumeration_type | select_type .
declaration = entity_decl | function_decl | procedure_decl |
              subtype_constraint_decl | type_decl .
derived_attr = attribute_decl ':' parameter_type ':=' expression
               ';' .
derive_clause = DERIVE derived_attr { derived_attr } .
domain_rule = [ rule_label_id ':' ] expression .
element = expression [ ':' repetition ] .
entity_body = { explicit_attr } [ derive_clause ]
              [ inverse_clause ] [ unique_clause ]
              [ where_clause ] .
entity_constructor = entity_ref '(' [ expression { ',' 
                     expression } ] ')' .
entity_decl = entity_head entity_body END_ENTITY ';' .
entity_head = ENTITY entity_id subsuper ';' .
entity_id = simple_id .
enumeration_extension = BASED_ON type_ref [ WITH
                        enumeration_items ] .
enumeration_id = simple_id .
enumeration_items = '(' enumeration_id { ',' enumeration_id } ')' .
enumeration_reference = [ type_ref '.' ] enumeration_ref .
enumeration_type = [ EXTENSIBLE ] ENUMERATION [ ( OF
                   enumeration_items ) | enumeration_extension ] .
escape_stmt = ESCAPE ';' .
```

```
215  explicit_attr  = attribute_decl { ',' attribute_decl } ':'
                      [ OPTIONAL ] parameter_type ';' .
216  expression = simple_expression [ rel_op_extended
                     simple_expression ] .
217  factor = simple_factor [ '**' simple_factor ] .
218  formal_parameter  = parameter_id { ',' parameter_id } ':'
                         parameter_type .
219  function_call  = ( built_in_function  |  function_ref )
                      [ actual_parameter_list ] .
220  function_decl  = function_head algorithm_head stmt { stmt }
                      END_FUNCTION ';' .
221  function_head =  FUNCTION function_id [ '(' formal_parameter
                      { ';' formal_parameter } ')' ] ':'
                      parameter_type ';' .
222  function_id  = simple_id .
223  generalized_types  =  aggregate_type  |  general_aggregation_types  |
                           generic_entity_type  |  generic_type .
224  general_aggregation_types  =  general_array_type  |
                                    general_bag_type  |  general_list_type  |
                                    general_set_type .
225  general_array_type  =  ARRAY [ bound_spec ] OF [ OPTIONAL ]
                           [ UNIQUE ] parameter_type .
226  general_bag_type  =  BAG [ bound_spec ] OF parameter_type .
227  general_list_type =  LIST [ bound_spec ] OF [ UNIQUE ]
                          parameter_type .
228  general_ref  =  parameter_ref | variable_ref .
229  general_set_type  =  SET [ bound_spec ] OF parameter_type .
230  generic_entity_type  =  GENERIC_ENTITY [ ':' type_label ] .
231  generic_type  =  GENERIC [ ':' type_label ] .
232  group_qualifier  =  '\' entity_ref .
233  if_stmt  =  IF logical_expression THEN stmt { stmt } [ ELSE stmt
                { stmt } ] END_IF ';' .
234  increment  =  numeric_expression .
235  increment_control  =  variable_id ':=' bound_1 TO bound_2 [ BY
                           increment ] .
236  index  =  numeric_expression .
237  index_1  =  index .
238  index_2  =  index .
239  index_qualifier  =  '[' index_1 [ ':' index_2 ] ']' .
240  instantiable_type  =  concrete_types  |  entity_ref .
241  integer_type  =  INTEGER .
242  interface_specification  =  reference_clause  |  use_clause .
243  interval  =  '{' interval_low interval_op interval_item
                  interval_op interval_high '}' .
244  interval_high  =  simple_expression .
245  interval_item  =  simple_expression .
246  interval_low  =  simple_expression .
247  interval_op  =  '<'  |  '<='.
248  inverse_attr  =  attribute_decl ':' [ ( SET  |  BAG ) [ bound_spec ]
                      OF ] entity_ref FOR [ entity_ref '.' ]
                      attribute_ref ';' .
249  inverse_clause  =  INVERSE inverse_attr { inverse_attr } .
250  list_type  =  LIST [ bound_spec ] OF [ UNIQUE ] instantiable_type .
251  literal  =  binary_literal  |  logical_literal  |  real_literal  |
                string_literal .
252  local_decl  =  LOCAL local_variable { local_variable }
                    END_LOCAL ';' .
253  local_variable  =  variable_id { ',' variable_id } ':'
                        parameter_type [ ':=' expression ] ';' .
254  logical_expression  =  expression .
255  logical_literal  =  FALSE | TRUE | UNKNOWN .
256  logical_type  =  LOGICAL .
```

```
multiplication_like_op = '*' | '/' | DIV | MOD | AND | '||'.
named_types = entity_ref | type_ref.
named_type_or_rename = named_types [ AS ( entity_id | type_id ) ].
null_stmt = ';'.
number_type = NUMBER.
numeric_expression = simple_expression.
one_of = ONEOF '(' supertype_expression { ',' supertype_expression } ')'.
parameter = expression.
parameter_id = simple_id.
parameter_type = generalized_types | named_types | simple_types.
population = entity_ref.
precision_spec = numeric_expression.
primary = literal | ( qualifiable_factor { qualifier } ).
procedure_call_stmt = ( built_in_procedure | procedure_ref )
                      [ actual_parameter_list ] ';'.
procedure_decl = procedure_head algorithm_head { stmt }
                 END_PROCEDURE ';'.
procedure_head = PROCEDURE procedure_id [ '(' [ VAR ]
                 formal_parameter { ';' [ VAR ]
                 formal_parameter } ')' ] ';'.
procedure_id = simple_id.
qualifiable_factor = attribute_ref | constant_factor |
                     function_call | general_ref | population.
qualified_attribute = SELF group_qualifier attribute_qualifier.
qualifier = attribute_qualifier | group_qualifier |
            index_qualifier.
query_expression = QUERY '(' variable_id '<*' aggregate_source '|'
                   logical_expression ')'.
real_type = REAL [ '(' precision_spec ')' ].
redeclared_attribute = qualified_attribute [ RENAMED
                       attribute_id ].
referenced_attribute = attribute_ref | qualified_attribute.
reference_clause = REFERENCE FROM schema_ref [ '(' 
                   resource_or_rename { ',' resource_or_rename }
                   ')' ] ';'.
rel_op = '<' | '>' | '<=' | '>=' | '<>' | '=' | ':<>:' | ':=:'.
rel_op_extended = rel_op | IN | LIKE.
rename_id = constant_id | entity_id | function_id | procedure_id |
            type_id.
repeat_control = [ increment_control ] [ while_control ]
                 [ until_control ].
repeat_stmt = REPEAT repeat_control ';' stmt { stmt }
              END_REPEAT ';'.
repetition = numeric_expression.
resource_or_rename = resource_ref [ AS rename_id ].
resource_ref = constant_ref | entity_ref | function_ref |
               procedure_ref | type_ref.
return_stmt = RETURN [ '(' expression ')' ] ';'.
rule_decl = rule_head algorithm_head { stmt } where_clause
            END_RULE ';'.
rule_head = RULE rule_id FOR '(' entity_ref { ',' entity_ref }
            ')' ';'.
rule_id = simple_id.
rule_label_id = simple_id.
schema_body = { interface_specification } [ constant_decl ]
              { declaration | rule_decl }.
schema_decl = SCHEMA schema_id [ schema_version_id ] ';'
              schema_body END_SCHEMA ';'.
schema_id = simple_id.
schema_version_id = string_literal.
selector = expression.
```

```
300  select_extension = BASED_ON type_ref [ WITH select_list ] .
301  select_list = '(' named_types { ',' named_types } ')' .
302  select_type = [ EXTENSIBLE [ GENERIC_ENTITY ] ] SELECT
                   [ select_list | select_extension ] .
303  set_type = SET [ bound_spec ] OF instantiable_type .
304  sign = '+' | '-'.
305  simple_expression = term { add_like_op term } .
306  simple_factor = aggregate_initializer | entity_constructor |
                     enumeration_reference | interval |
                     query_expression | ( [ unary_op ] ( '(' 
                     expression ')' | primary ) ) .
307  simple_types = binary_type | boolean_type | integer_type |
                    logical_type | number_type | real_type |
                    string_type .
308  skip_stmt = SKIP ';'.
309  stmt = alias_stmt | assignment_stmt | case_stmt | compound_stmt |
          escape_stmt | if_stmt | null_stmt | procedure_call_stmt |
          repeat_stmt | return_stmt | skip_stmt .
310  string_literal = simple_string_literal | encoded_string_literal .
311  string_type = STRING [ width_spec ] .
312  subsuper = [ supertype_constraint ] [ subtype_declaration ] .
313  subtype_constraint = OF '(' supertype_expression ')' .
314  subtype_constraint_body = [ abstract_supertype ] [ total_over ]
                               [ supertype_expression ';' ] .
315  subtype_constraint_decl = subtype_constraint_head
                               subtype_constraint_body
                               END_SUBTYPE_CONSTRAINT ';'.
316  subtype_constraint_head = SUBTYPE_CONSTRAINT
                               subtype_constraint_id FOR entity_ref
                               ';'.
317  subtype_constraint_id = simple_id .
318  subtype_declaration = SUBTYPE OF '(' entity_ref { ',' entity_ref
                           } ')' .
319  supertype_constraint = abstract_entity_declaration |
                            abstract_supertype_declaration |
                            supertype_rule .
320  supertype_expression = supertype_factor { ANDOR
                            supertype_factor } .
321  supertype_factor = supertype_term { AND supertype_term } .
322  supertype_rule = SUPERTYPE subtype_constraint .
323  supertype_term = entity_ref | one_of | '(' supertype_expression
                      ')' .
324  syntax = schema_decl { schema_decl } .
325  term = factor { multiplication_like_op factor } .
326  total_over = TOTAL_OVER '(' entity_ref { ',' entity_ref } ')' ';'.
327  type_decl = TYPE type_id '=' underlying_type ';' [ where_clause ]
                 END_TYPE ';'.
328  type_id = simple_id .
329  type_label = type_label_id | type_label_ref .
330  type_label_id = simple_id .
331  unary_op = '+' | '-' | NOT .
332  underlying_type = concrete_types | constructed_types .
333  unique_clause = UNIQUE unique_rule ';' { unique_rule ';' } .
334  unique_rule = [ rule_label_id ':' ] referenced_attribute { ','
                   referenced_attribute } .
335  until_control = UNTIL logical_expression .
336  use_clause = USE FROM schema_ref [ '(' named_type_or_rename
                  { ',' named_type_or_rename } ')' ] ';'.
337  variable_id = simple_id .
338  where_clause = WHERE domain_rule ';' { domain_rule ';' } .
339  while_control = WHILE logical_expression .
340  width = numeric_expression .
341  width_spec = '(' width ')' [ FIXED ] .
```

А.3 Список перекрестных ссылок

Элементы, указанные слева, используются в правилах, указанных справа.

0	ABS	187
1	ABSTRACT	164,165,166
2	ACOS	187
3	AGGREGATE	171
4	ALIAS	174
5	AND	257,321
6	ANDOR	320
7	ARRAY	175,225
8	AS	259,288
9	ASIN	187
10	ATAN	187
11	BAG	180,226,248
12	BASED_ON	209,300
13	BEGIN	192
14	BINARY	181
15	BLENGTH	187
16	BOOLEAN	182
17	BY	235
18	CASE	191
19	CONSTANT	195
20	CONST_E	186
21	COS	187
22	DERIVE	201
23	DIV	257
24	ELSE	233
25	END	192
26	END_ALIAS	174
27	END_CASE	191
28	END_CONSTANT	195
29	END_ENTITY	206
30	END_FUNCTION	220
31	END_IF	233
32	END_LOCAL	252
33	END_PROCEDURE	271
34	END_REPEAT	286
35	END_RULE	291
36	END_SCHEMA	296
37	END_SUBTYPE_CONSTRAINT	315
38	END_TYPE	327
39	ENTITY	207
40	ENUMERATION	213
41	ESCAPE	214
42	EXISTS	187
43	EXTENSIBLE	213,302
44	EXP	187
45	FALSE	255
46	FIXED	341
47	FOR	174,248,292,316
48	FORMAT	187
49	FROM	281,336
50	FUNCTION	221
51	GENERIC	231
52	GENERIC_ENTITY	230,302
53	HIBOUND	187
54	HIINDEX	187
55	IF	233
56	IN	283
57	INSERT	188
58	INTEGER	241

59	INVERSE	249
60	LENGTH	187
61	LIKE	283
62	LIST	227,250
63	LOBOUND	187
64	LOCAL	252
65	LOG	187
66	LOG10	187
67	LOG2	187
68	LOGICAL	256
69	LOINDEX	187
70	MOD	257
71	NOT	331
72	NUMBER	261
73	NVL	187
74	ODD	187
75	OF	171,175,180,191,213,225,226,227,229, 248,250,303,313,318
76	ONEOF	263
77	OPTIONAL	175,215,225
78	OR	168
79	OTHERWISE	191
80	PI	186
81	PROCEDURE	272
82	QUERY	277
83	REAL	278
84	REFERENCE	281
85	REMOVE	188
86	RENAMED	279
87	REPEAT	286
88	RETURN	290
89	ROLESOF	187
90	RULE	292
91	SCHEMA	296
92	SELECT	302
93	SELF	186,275
94	SET	229,248,303
95	SIN	187
96	SIZEOF	187
97	SKIP	308
98	SQRT	187
99	STRING	311
100	SUBTYPE	318
101	SUBTYPE_CONSTRAINT	316
102	SUPERTYPE	165,166,322
103	TAN	187
104	THEN	233
105	TO	235
106	TOTAL_OVER	326
107	TRUE	255
108	TYPE	327
109	TYPEOF	187
110	UNIQUE	175,225,227,250,333
111	UNKNOWN	255
112	UNTIL	335
113	USE	336
114	USEDIN	187
115	VALUE	187
116	VALUE_IN	187
117	VALUE_UNIQUE	187
118	VAR	272
119	WHERE	338

120	WHILE	339
121	WITH	209,300
122	XOR	168
123	bit	139
124	digit	125,127,131,134,143
125	digits	141,142
126	encoded_character	140
127	hex_digit	136
128	letter	131,134,143
129	lparen_then_not_lparen_star	145
130	not_lparen_star	129
131	not_paren_star	130,135,145
132	not_paren_star_quote_special	133,134,137
133	not_paren_star_special	131
134	not_quote	144
135	not_rparen_star	138
136	octet	126
137	special	
138	not_rparen_star_then_rparen	145
139	binary_literal	251
140	encoded_string_literal	310
141	integer_literal	142
142	real_literal	251
143	simple_id	178,197,208,210,222,265,273,293,294,297,317,328,330,337
144	simple_string_literal	310
145	embedded_remark	145,146
146	remark	
147	remark_tag	145,149
148	remark_ref	147
149	tail_remark	146
150	attribute_ref	148,179,248,274,280
151	constant_ref	148,196,289
152	entity_ref	148,205,232,240,248,258,267,289,292, 316,318,323,326
153	enumeration_ref	148,212
154	function_ref	148,219,289
155	parameter_ref	148,228
156	procedure_ref	148,270,289
157	rule_label_ref	148
158	rule_ref	148
159	schema_ref	148,281,336
160	subtype_constraint_ref	148
161	type_label_ref	148,329
162	type_ref	148,193,209,212,258,289,300
163	variable_ref	148,228
164	abstract_entity_declaration	319
165	abstract_supertype	314
166	abstract_supertype_declaration	319
167	actual_parameter_list	219,270
168	add_like_op	305
169	aggregate_initializer	306
170	aggregate_source	277
171	aggregate_type	223
172	aggregation_types	193
173	algorithm_head	220,271,291
174	alias_stmt	309
175	array_type	172
176	assignment_stmt	309
177	attribute_decl	200,215,248
178	attribute_id	150,177,279
179	attribute_qualifier	275,276
180	bag_type	172

181 binary_type | 307
182 boolean_type | 307
183 bound_1 | 185,235
184 bound_2 | 185,235
185 bound_spec | 175,180,225,226,227,229,248,250,303
186 built_in_constant | 196
187 built_in_function | 219
188 built_in_procedure | 270
189 case_action | 191
190 case_label | 189
191 case_stmt | 309
192 compound_stmt | 309
193 concrete_types | 240,332
194 constant_body | 195
195 constant_decl | 173,295
196 constant_factor | 274
197 constant_id | 151,194,284
198 constructed_types | 332
199 declaration | 173,295
200 derived_attr | 201
201 derive_clause | 204
202 domain_rule | 338
203 element | 169
204 entity_body | 206
205 entity_constructor | 306
206 entity_decl | 199
207 entity_head | 206
208 entity_id | 152,207,259,284
209 enumeration_extension | 213
210 enumeration_id | 153,211
211 enumeration_items | 209,213
212 enumeration_reference | 306
213 enumeration_type | 198
214 escape_stmt | 309
215 explicit_attr | 204
216 expression | 176,190,194,200,202,203,205,253,254, 264,290,299,306
217 factor | 325
218 formal_parameter | 221, 272
219 function_call | 274
220 function_decl | 199
221 function_head | 220
222 function_id | 154,221,284
223 generalized_types | 266
224 general_aggregation_types | 223
225 general_array_type | 224
226 general_bag_type | 224
227 general_list_type | 224
228 general_ref | 174,176,274
229 general_set_type | 224
230 generic_entity_type | 223
231 generic_type | 223
232 group_qualifier | 275,276
233 if_stmt | 309
234 increment | 235
235 increment_control | 285
236 index | 237,238
237 index_1 | 239
238 index_2 | 239
239 index_qualifier | 276
240 instantiable_type | 175,180,194,250,303
241 integer_type | 307

242 interface_specification | 295
243 interval | 306
244 interval_high | 243
245 interval_item | 243
246 interval_low | 243
247 interval_op | 243
248 inverse_attr | 249
249 inverse_clause | 204
250 list_type | 172
251 literal | 269
252 local_decl | 173
253 local_variable | 252
254 logical_expression | 233,277,335,339
255 logical_literal | 251
256 logical_type | 307
257 multiplication_like_op | 325
258 named_types | 259,266,301
259 named_type_or_rename | 336
260 null_stmt | 309
261 number_type | 307
262 numeric_expression | 183,184,234,236,268,287,340
263 one_of | 323
264 parameter | 167
265 parameter_id | 155, 218
266 parameter_type | 171,200,215,218,221,225,226,227,229,253
267 population | 274
268 precision_spec | 278
269 primary | 306
270 procedure_call_stmt | 309
271 procedure_decl | 199
272 procedure_head | 271
273 procedure_id | 156,272,284
274 qualifiable_factor | 269
275 qualified_attribute | 279,280
276 qualifier | 174,176,269
277 query_expression | 306
278 real_type | 307
279 redeclared_attribute | 177
280 referenced_attribute | 334
281 reference_clause | 242
282 rel_op | 283
283 rel_op_extended | 216
284 rename_id | 288
285 repeat_control | 286
286 repeat_stmt | 309
287 repetition | 203
288 resource_or_rename | 281
289 resource_ref | 288
290 return_stmt | 309
291 rule_decl | 295
292 rule_head | 291
293 rule_id | 158,292
294 rule_label_id | 157,202,334
295 schema_body | 296
296 schema_decl | 324
297 schema_id | 159,296
298 schema_version_id | 296
299 selector | 191
300 select_extension | 302
301 select_list | 300,302
302 select_type | 198

303	set_type	172
304	sign	142
305	simple_expression	170,216,244,245,246,262
306	simple_factor	217
307	simple_types	193,266
308	skip_stmt	309
309	stmt	174,189,191,192,220,233,271,286,291
310	string_literal	251,298
311	string_type	307
312	subsuper	207
313	subtype_constraint	166,322
314	subtype_constraint_body	315
315	subtype_constraint_decl	199
316	subtype_constraint_head	315
317	subtype_constraint_id	160,316
318	subtype_declaration	312
319	supertype_constraint	312
320	supertype_expression	263,313,314,323
321	supertype_factor	320
322	supertype_rule	319
323	supertype_term	321
324	syntax	
325	term	305
326	total_over	314
327	type_decl	199
328	type_id	162,259,284,327
329	type_label	171,230,231
330	type_label_id	161,329
331	unary_op	306
332	underlying_type	327
333	unique_clause	204
334	unique_rule	333
335	until_control	285
336	use_clause	242
337	variable_id	163,174,235,253,277
338	where_clause	204,291,327
339	while_control	285
340	width	341
341	width_spec	181,311

Приложение В
(обязательное)

Определение допустимых реализаций объектов

В конкретном графе подтип/супертип может присутствовать большое число сложных и простых объектных типов данных, которые могут реализовываться в виде своих экземпляров. В данном приложении показано, как объектные типы данных идентифицируются в объявлении общего графа подтипов/супертипов.

Примечание — Для иллюстрации дальнейшего изложения рассмотрим множество натуральных чисел [1, 2, 3 , ...]. Данное множество может быть структурировано разными способами:

- разделено на четные и нечетные числа: [2, 4, 6, ...] и [1, 3, 5, 7, ...];
- выделены простые числа: [2, 3, 5, 7, ...];
- выделены числа с общим делителем 3: [3, 6, 9, 12, ...];
- выделены числа с общим делителем 4: [4, 8, 12, 16, ...].

В терминах языка EXPRESS натуральные числа могут быть представлены супертипом, а другие множества чисел — его подтипами. Очевидно, что некоторые из подтипов являются неперекрывающимися множествами (например, четные и нечетные числа), в то время как другие будут иметь какие-то общие элементы (например, подтипы «Числа с общим делителем 3» и «Числа с общим делителем 4»).

В.1 Формализованный подход

Любая группа объектных типов данных, связанная отношениями «подтип/супертип», может рассматриваться как потенциально реализуемый граф подтипов/супертипов. В данном формализованном подходе такие группы объектных типов данных называются частичными сложными объектными типами данных. Частичный сложный объектный тип данных может содержать единственный объектный тип данных, поскольку отдельный объектный тип данных является потенциально реализуемым. Частичный сложный объектный тип данных обозначается именами составляющих его объектных типов данных, разделенных символом ' & '. Частичные сложные объектные типы данных могут комбинироваться для образования новых частичных сложных объектных типов данных. Частичный сложный объектный тип данных может быть сложным объектным типом данных сложного объекта, но для того, чтобы в этом удостовериться, необходимо осуществить его полную оценку. Экземпляр сложного объекта, являющийся предметом рассмотрения в данном приложении, представляет собой экземпляр сложного объектного типа данных.

Для любого частичного сложного объектного типа данных справедливы следующие тождества:

$A\&A \equiv A$, то есть конкретный объектный тип данных может присутствовать в данном частичном сложном объектном типе данных только один раз;

$A\&B \equiv B\&A$, то есть группирование частичных сложных объектных типов данных является коммутативным;

$A\&(B\&C) \equiv (A\&B)\&C \equiv A\&B\&C$, то есть группирование частичных сложных объектных типов данных является ассоциативным; круглые скобки, указывающие на приоритет выполнения вычислений в данном случае не влияют на результат.

Результирующее множество определяется как математическое множество частичных сложных объектных типов данных, обозначенное частичными сложными объектными типами данных, разделенными запятыми (' , ') и заключенными в квадратные скобки. Пустое результирующее множество обозначается ' [ ] '.

Для объединения частичного сложного объектного типа данных и результирующего множества могут быть использованы два оператора:

$A + [B1, B2] \equiv [B1, B2] + A \equiv [A, B1, B2]$. Оператор ' + ' добавляет частичный сложный объектный тип данных к результирующему множеству в качестве нового элемента данного множества. Один и тот же частичный сложный объектный тип данных не должен присутствовать в одном результирующем множестве более одного раза;

$A\&[B1, B2] \equiv [B1, B2]\&A \equiv [A\&B1, A\&B2]$. Оператор ' & ' добавляет частичный сложный объектный тип данных ко всем частичным сложным объектным типам данных в результирующем множестве. Следовательно, он является дистрибутивным для результирующих множеств.

Результирующие множества могут объединяться посредством тех же двух механизмов:

$[A1, A2] + [B1, B2] \equiv [A1, A2, B1, B2]$. Может быть сформировано результирующее множество, содержащее все элементы двух объединяемых множеств. Данная операция является объединением двух множеств;

$[A1, A2]\&[B1, B2] \equiv [A1\&B1, A1\&B2, A2\&B1, A2\&B2]$. Результирующее множество может быть сформировано путем повторного применения правила дистрибутивности, касающегося оператора ' & ', для каждого элемента первого результирующего множества ко второму результирующему множеству.

Результирующие множества могут фильтроваться посредством оператора ' / ' с целью создания нового результирующего множества:

[*A*, *A*& *B*, *A*& *C*, *A*& *B*& *D*, *B*& *C*, *D*]/*A* ≡ [*A*, *A*& *B*, *A*& *C*, *A*& *B*& *D*]. Новое результирующее множество содержит только те элементы исходного результирующего множества, которые содержат данный частичный сложный объектный тип данных;

[*A*, *A*& *B*, *A*& *C*, *A*& *B*& *D*, *B*& *C*, *D*] / [*B*, *D*] ≡ [*A*& *B*, *A*& *B*& *D*, *B*& *C*, *D*]. Новое результирующее множество может быть сформировано путем повторной фильтрации первого результирующего множества каждым частичным сложным объектным типом данных из второго результирующего множества с последующим объединением результатов посредством оператора ' + '.

Может быть определена разность результирующих множеств посредством оператора ' — ' с целью создания нового результирующего множества:

[*A*1, *A*2, *B*1, *B*2] — [*A*2, *B*1] ≡ .[*A*1, *B*2]. Может быть сформировано результирующее множество, содержащее все элементы первого результирующего множества за исключением элементов, входящих во второе результирующее множество.

Следующие тождества справедливы для любого результирующего множества:

[*A*, *B*] ≡ [*B*, *A*]. Результирующие множества не зависят от порядка следования своих элементов;

[*A*, *A*, *B*] ≡ [*A*, *B*]. Конкретный частичный сложный объектный тип данных может присутствовать в любом результирующем множестве только один раз;

[*A*, [*B*, *C*]] ≡ [*A*, *B*, *C*]. Результирующие множества могут быть вложенными.

B.2 Операторы ограничения супертипов и подтипов

Используя вышеописанный формальный подход, можно переписать ограничения, определенные в выражениях для супертипов на языке EXPRESS, а также ограничения на подтипы в терминах результирующих множеств. Преобразования, представленные в B.2.1 — B.2.3, применяются рекурсивно до тех пор, пока не останется ни одного терма, представляющего супертип (**ONEOF**, **AND** или **ANDOR**).

Данные преобразования не описывают полное содержание выражения супертипов и ограничений подтипов, в частности, условий **ONEOF** и **TOTAL_OVER**. Для этого требуется полный алгоритм, представленный в B.3.

B.2.1 ONEOF

Список оператора **ONEOF** преобразуется в результирующее множество, содержащее варианты выбора оператора **ONEOF**, то есть:

ONEOF (***A***, ***B***,. . .) → [*A*, *B*, . . .]

B. 2.2 AND

Оператор **AND** эквивалентен оператору & и оперирует с частичными сложными объектными типами данных или с результирующими множествами с целью создания частичного сложного объектного типа данных или результирующего множества.

A **AND** ***B*** → [*A*& *B*]

A **AND ONEOF** ***(B1, B2)*** → *A*& [*B*1, *B*2] = [*A*& *B*1, *A*& *B*2]

ONEOF ***(A1, A2)*** **AND ONEOF** ***(B1,B2)*** → [*A*1, *A*2] & [*B*1, *B*2] = [*A*1& *B*1, *A*1& *B*2, *A*2& *B*1, *A*2& *B*2]

B. 2.3 ANDOR

Оператор **ANDOR** создает результирующее множество, содержащее все операнды по отдельности и операнды, объединенные оператором &. Оператор **ANDOR** оперирует с частичными сложными объектными типами данных или результирующими множествами.

A **ANDOR** ***B*** → [*A*, *B*, *A*& *B*]

A **ANDOR ONEOF**(***B1***, ***B2***) → [*A*, [*B*1, *B*2], *A*& [*B*1, *B*2]] = [*A*, *B*1, *B*2, *A*& *B*1, *A*& *B*2]

ONEOF (***A1***, ***A2***) **ANDOR ONEOF** (***B1***, ***B2***) → [[*A*1, *A*2] , [*B*1, *B*2] , [*A*1, *A*2] & [*B*1, *B*2]] =

[*A*1, *A*2, *B*1, *B*2, *A*1& *B*1, *A*1& *B*2, *A*2& *B*1, *A*2& *B*2]

B.2.4 Приоритет операторов

Вычисление результирующих множеств проводится слева направо. При этом операторы, имеющие высший приоритет, выполняются первыми в соответствии с 9.2.5.5.

Пример — Нижеприведенное выражение вычисляется следующим образом:

A ANDOR *B* and *C* → [*A*, [*B*& *C*], *A*& [*B*& *C*]] = [*A*, *B*& *C*, *A*& *B*& *C*]

B.3 Интерпретация возможных типов данных сложных объектов

Интерпретация выражений супертипов и ограничений подтипов с дополнительной информацией, имеющейся в объявленной структуре, позволяет разработчику EXPRESS-схемы определить сложные объектные типы данных, которые могут быть реализованы, исходя из данных объявлений. Для того, чтобы обеспечить данное определение может быть создано результирующее множество сложных объектных типов данных для графа подтипов/супертипов. Для этого определим следующие термины:

подтип с множественным наследованием (multiply inheriting subtype): Подтипом с множественным наследованием является подтип, в объявлении которого указаны два или более супертипов.

корневой супертип (root supertype): Корневым супертипом является супертип, не являющийся подтипом.

Результирующее множество *R* сложных объектных типов данных вычисляют по следующему алгоритму:

a) выявляют все объявления объектов, формирующие граф подтипов/супертипов.

Примечание — Для сложных графов подтипов/супертипов для выполнения данного действия может потребоваться несколько итераций;

b) для каждого супертипа *i* из графа подтипов/супертипов, в котором объявлено ограничение супертипа создается конструкция SUBTYPE_CONSTRAINT следующего вида:

SUBTYPE_CONSTRAINT i_superconstraint FOR i;
<supertype_constraint> ;
END_SUBTYPE_CONSTRAINT;

в которой элемент <supertype_constraint> заменяется ограничением супертипа, объявленным в объекте. Для целей данного алгоритма следует рассматривать данное ограничение как часть схемы. Кроме того, следует игнорировать выражение супертипа в объявлении объекта, послужившее основой для данного ограничения подтипа.

Примечание — На данном шаге ограничения супертипов, объявленные в объекте, преобразуются в эквивалентные объявления SUBTYPE_CONSTRAINT;

c) для каждого супертипа *i* из графа подтипов/супертипов выявляются все имеющиеся в данном графе типы данных *j*1, *j*2, ... *j*k, которые определены как подтипы *i*, но не встречаются в какой-либо конструкции SUBTYPE_CONSTRAINT, определенной для *i* в данной схеме или сгенерированной на шаге b), и создается конструкция SUBTYPE_CONSTRAINT следующего вида:

SUBTYPE_CONSTRAINT i_othersubtypes FOR i;
j1 ANDOR j2 ANDOR ... ANDOR jk;
END_SUBTYPE_CONSTRAINT;

Для целей данного алгоритма будем рассматривать данное ограничение как часть схемы;

d) для каждого супертипа *i* из графа подтипов/супертипов выявляются все конструкции SUBTYPE_CONSTRAINT *sc*1, *sc*2, ... *sck*, в условии FOR которых присутствует *i*. На данном шаге игнорируются части ограничений подтипов, содержащие полное покрытие или абстрактные ограничения. Выражения подтипов *sxi* данных ограничений объединяются в одну конструкцию SUBTYPE_CONSTRAINT *sti* следующего вида: (*sx*1 ANDOR *sx*2 ANDOR *sx*3 ... ANDOR *sxk*);

e) для каждого супертипа *i* из графа подтипов/супертипов генерируется результирующее множество, представляющее ограничения между его непосредственными подтипами, путем применения преобразований из B.2 и тождеств из B.1 к конструкции SUBTYPE_CONSTRAINT *sti*, сформированной на предыдущем шаге по перечислению d). Полученный результат объединяется с *i* посредством оператора & . Если *i* не определено как ABSTRACT SUPERTYPE в своем объявлении ENTITY или в какой-либо конструкции SUBTYPE_CONSTRAINT из *i*, то *i* добавляется к результату с использованием оператора + . Назовем полученное множество E_i ;

f) для каждого корневого супертипа *r* из графа подтипов/супертипов E_r раскрывается следующим образом:

1) для каждого подтипа *s* из *r* заменяется каждое вхождение (включая вхождения в сложные объектные типы данных) *s* в E_r на E_s, если это возможно, и применяются преобразования из B.2 и тождества из B.1,

2) рекурсивно повторяется шаг по пункту 1) перечисления f) для каждого *s*, раскрывая подтипы *s* до тех пор, пока не будут достигнуты концевые объекты (для которых не существует E_s).

Примечание — Данная рекурсивная процедура должна завершиться, поскольку в графе подтипов/супертипов нет циклов;

g) объединяют корневые множества. Создается $R = \Sigma_r E_r \equiv E_{r1} + E_{r2} + ...$, то есть *R* является объединением множеств, созданных на шаге по перечислению f);

h) для каждого супертипа *s* из *R* и для каждого ограничения подтипа полного покрытия $t_1, t_2, ... t_k$, определенного для *s*, выполняют следующие действия:

1) пусть *t* определяется как (t_1 ANDOR t_2 ...ANDOR t_k),

2) для всех непосредственных подтипов s_i из *s*, не входящих в $t_1, t_2, ... t_k$, каждое вхождение s_i в *R* заменяется выражением, полученным из (s_i и *t*) с использованием определений из B.2.2,

3) *R* преобразовывают в соответствии с преобразованиями, определенными в B.2, и тождествами, определенными в B.1;

i) для каждого подтипа с множественным наследованием *m* выполняют следующие действия:

1) для каждого из его непосредственных супертипов *s* формируют множество *R*/*m*/*s*, содержащее только те сложные типы данных из *R*, которые включают как *m*, так и *s*,

2) создают результирующее множество комбинаций супертипов, допустимых для m: $P_m = R/m/s1\&R/m/s2\&...$, то есть объединяются результирующие множества, сформированные на шаге по пункту 1) перечисления i1) с помощью оператора &,

3) формируют результирующее множество комбинаций супертипов, которое может не включать в себя все супертипы m: $X_m = \Sigma_s R/m/s$, то есть объединяются результирующие множества, полученные на шаге по пункту 1) перечисления i1),

4) выполняют действие: $R = (R — X_m) + P_m$;

j) для каждого k-го выражения SUBTYPE_CONSTRAINT [включая сформированные на шагах по перечислениям b) и h)], имеющего вид ONEOF(S_1, S_2, ...), выполняют следующие действия:

1) для каждой пары подвыражений S_i, S_j, управляемой k ($i < j$), вычисляют множество комбинаций, запрещенных оператором ONEOF(S_i, S_j): $D_k^{i,j} = [S_i \& S_j]$. $D_k^{i,j}$ преобразуется в соответствии с преобразованиями, определенными в В.2, и тождествами, определенными в В.1,

2) формируют $D_k = \Sigma_{i,j} D_k^{i,j}$, то есть D_k является объединением множеств, вычисленных на шаге по пункту 1) перечисления j1),

3) выполняют действие: $R = R — (R/D_k)$;

k) для каждого k-го выражения SUBTYPE_CONSTRAINT [включая сформированные на шагах по перечислениям b) и h)], имеющего вид S_1 AND S_2, выполняют следующие действия:

1) вычисляют множество требуемых комбинаций, задаваемых k: $Q_k = [S_1 \& S_2]$. Q_k преобразуется в соответствии с преобразованиями, определенными в В.2, и тождествами, определенными в В.1,

2) для каждого объектного типа данных объекта i, объявленного в k, вычисляют множество недопустимых комбинаций объектов, содержащих i, которые запрещены k: $D_k^i = R/i — R/(Q_k/i)$,

3) выполняют действие: $D_k = \Sigma_i D_k^i$, то есть D_k является объединением множеств, сформированных на шаге по пункту 2) перечисления k).

4) выполняют действие: $R = R — D_k$;

l) полученное результирующее множество R является результирующим множеством для исходного графа подтипов/супертипов.

Примеры

1 В данном примере заданы только объявления супертипов и подтипов объектов, поскольку только они требуются для выявления возможных сложных объектных типов данных.

```
SCHEMA example;
ENTITY p;
END_ENTITY;
SUBTYPE_CONSTRAINT p_subs FOR p;
   TOTAL_OVER(m, f);
   ONEOF(m, f) AND ONEOF(c, a);
END_SUBTYPE_CONSTRAINT;
ENTITY m SUBTYPE OF (p);
END_ENTITY;
ENTITY f SUBTYPE OF (p);
END_ENTITY;
ENTITY c SUBTYPE OF (p);
END_ENTITY;
ENTITY a SUBTYPE OF (p);
END_ENTITY;
SUBTYPE_CONSTRAINT no_li FOR a;
   ABSTRACT SUPERTYPE;
   ONEOF(l, i);
END_SUBTYPE_CONSTRAINT;
ENTITY l SUBTYPE OF (a);
END_ENTITY;
ENTITY i SUBTYPE OF (a);
END_ENTITY;
END_SCHEMA;
```

Данная схема в формате EXPRESS-G представлена на рисунке В.1.

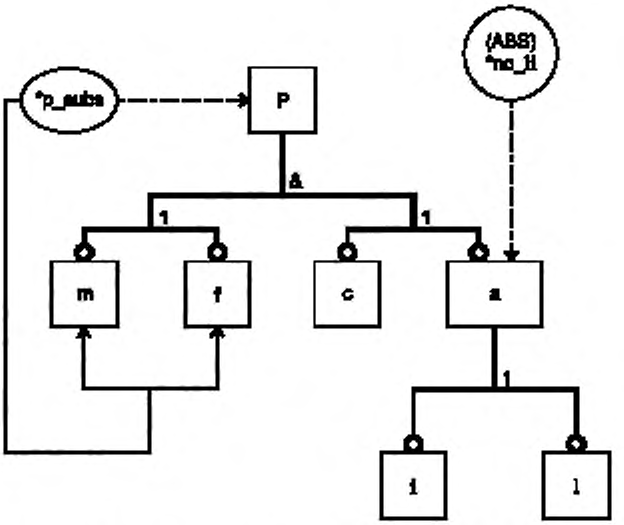

Рисунок В.1 — EXPRESS-G диаграмма схемы из примера 1

Возможные сложные объектные типы данных могут быть определены следующим образом:

- в представленной выше EXPRESS-схеме уже заданы все объявления объектов и полные выражения супертипов, которые требуются для алгоритма на шагах по перечислению a), b) и c);

- в результате выполнения шага по перечислению d) получаем:

```
SUBTYPE_CONSTRAINT stp FOR p;
TOTAL_OVER(m, f);
( (ONEOF(m, f) AND ONEOF(c, a)) );
END_SUBTYPE_CONSTRAINT;
SUBTYPE_CONSTRAINT sta FOR a;
(ONEOF(i,l));
END_SUBTYPE_CONSTRAINT;
```

- в результате выполнения шага по перечислению e) получаем:

$E_p \rightarrow$ [*p&m&c, p&m&a, p&f&c, p&f&a, p*];

$E_a \rightarrow$ [*a&l, a&i*];

- в результате выполнения шага по перечислению f) расширяются объявления корневых объектов, в данном случае p. В результате получаем следующее множество:

E_p = [*p&m&c, p&m&a&l, p&m&a&i, p&f&c, p&f&a&l, p&f&a&i, p*];

- в результате объединения корневых множеств на шаге по перечислению g), получаем:

R = [*p&m&c, p&m&a&l, p&m&a&i, p&f&c, p&f&a&l, p&f&a&i, p*];

- в результате применения действий на шаге по перечислению h) к ограничению TOTAL_OVER, получаем:

TOTAL_OVER(m, f): s_p = [*p&m, p&f*].

Заменяя все вхождения p, не содержащие m или f, получаем:

R = [*p&m&c, p&m&a&l, p&m&a&i, p&f&c, p&f&a&l, p&f&a&i, p&m, p&f*].

- подтипы с множественным наследованием отсутствуют, поэтому выполнения шага i) не требуется;

- выполняя действия на шаге по перечислению j) для каждого ограничения ONEOF, получаем:

ONEOF(m,f): $D_1^{1,2}$ = [*m&f*];

D_1 = [*m&f*].

После удаления D_1 *из R в соответствии с шагом по пункту 3) перечисления j) R остается неизменным. Следовательно, мы имеем следующее множество:*

R = [*p&m&c, p&m&a&l, p&m&a&i, p&f&c, p&f&a&l, p&f&a&i, p&m, p&f*].

ONEOF(c,a): $D_2^{1,2}$ = [*c&a*];

D_2 = [*c&a*].

После удаления D_2 из R в соответствии с шагом по пункту 3) перечисления j) R остается неизменным. Следовательно, мы имеем следующее множество:

R = [p&m&c, p&m&a&l, p&m&a&i, p&f&c, p&f&a&l, p&f&a&i, p&m, p&f].

ONEOF(l,i): $D_3^{1,2}$ = [*l&i*];

D_3 = [*l&i*].

После удаления D_3 из R в соответствии с шагом по пункту 3) перечисления j) R остается неизменным. Следовательно, мы имеем следующее множество:

R = [p&m&c, p&m&a&l, p&m&a&i, p&f&c, p&f&a&l, p&f&a&i, p&m, p&f];

- выполняя шаг по перечислению k) для каждого ограничения AND получаем:

ONEOF(m, f) AND ONEOF(c, a): Q_1 = [*m&c, m&a, f&c, f&a*];

D_1^m = [*p&m*];

D_1^f = [*p&f*];

D_1^c = [ ];

D_1^a = [ ];

D_1 = [*p&m, p&f*].

После удаления D_1 из R в соответствии с шагом по пункту 4) перечисления k) R остается неизменным. Следовательно, имеем следующее множество:

R = [p&m&c, p&m&a&l, p&m&a&i, p&f&c, p&f&a&l, p&f&a&i];

после выполнения шага по перечислению l) получаем результат:

R = [p&m&c, p&m&a&l, p&m&a&i, p&f&c, p&f&a&l, p&f&a&i].

Данный пример, хотя и являющийся произвольным, мог бы быть сделан более реалистичным, если объектам дать более содержательные имена. Например, если вместо p, m, f, c, a, i объекты были бы названы person *(личность),* male *(мужчина),* female *(женщина),* citizen *(гражданин),* alien *(иностранец),* legal_alien *(легальный иностранец) и* illegal_alien *(нелегальный иностранец), соответственно.*

Данная трактовка дает следующий смысл некоторым элементам из окончательного результирующего множества:

- личность (person), *которая является мужчиной* (male) *и гражданином* (citizen);

- личность (person), *которая является мужчиной иностранцем* (male alien) *и нелегальным иностранцем* (illegal_alien);

- личность (person), *которая ...*

Кроме того, ограничение TOTAL_OVER *обеспечивает то, что результирующие множества, определенные для какой-либо другой схемы, расширяющей данный граф подтипов/супертипов, также должны включать объекты* male *или* female *в число допустимых экземпляров объекта* person.

2 Данный пример демонстрирует, что ONEOF *является глобальным ограничением, которое не может быть аннулировано множественным наследованием.*

```
SCHEMA diamond;
ENTITY a;
END_ENTITY;
SUBTYPE_CONSTRAINT a_subs FOR a;
   ONEOF(b, c);
END_SUBTYPE_CONSTRAINT;
ENTITY b SUBTYPE OF (a);
END_ENTITY;
ENTITY c SUBTYPE OF (a);
END_ENTITY;
ENTITY d SUBTYPE OF (b, c);
END_ENTITY;
END_SCHEMA;
```

Данная схема в формате EXPRESS-G представлена на рисунке B.2.

В представленной выше EXPRESS-схеме уже заданы все объявления объектов и полные выражения супертипов, которые требуются для алгоритма на шагах по перечислениям a) и b).

В результате выполнения шага по перечислению c) получаем:

```
SUBTYPE_CONSTRAINT b_othersubtypes FOR b;
   d;
END_SUBTYPE_CONSTRAINT;
SUBTYPE_CONSTRAINT c_othersubtypes FOR c;
   d;
END_SUBTYPE_CONSTRAINT;
```

В результате выполнения шага по перечислению d), получаем:

```
SUBTYPE_CONSTRAINT sca FOR a;
   ONEOF(b, c);
END_SUBTYPE_CONSTRAINT;
SUBTYPE_CONSTRAINT scb FOR b;
   d;
END_SUBTYPE_CONSTRAINT;
SUBTYPE_CONSTRAINT scc FOR c;
   d;
END_SUBTYPE_CONSTRAINT;
```

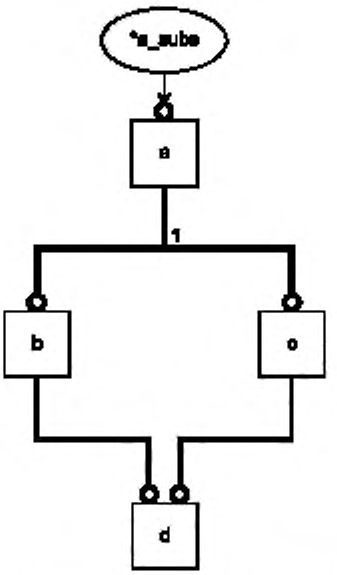

Рисунок В.2 — EXPRESS-G диаграмма схемы из примера 2

В результате выполнения шага по перечислению e) получаем:

$E_a \rightarrow$ [*a&b, a&c, a*];

$E_b \rightarrow$ [*b&d, b*];

$E_c \rightarrow$ [*c&d, c*];

$E_d \rightarrow$ [*d*].

В результате выполнения шага по перечислению f) расширяются объявления корневых объектов, в данном случае a. В результате получаем следующее множество:

E_a = [*a&b&d, a&b, a&c&d, a&c, a*].

В результате объединения корневых множеств на шаге по перечислению g) получаем:

R = [*a&b&d, a&b, a&c&d, a&c, a*].

Применение шага по перечислению i) к каждому подтипу с множественным наследованием дает следующие результаты:

Для объекта d: C_d^b = [*a&b&d*];

C_d^c = [*a&c&d*];

P_d = [*a&b&d&c*];

X_d = [*a&b&d, a&c&d*].

Новое множество R = (R — X_d) + P_d тогда определяется как [*a&b, a&c, a, a&b&d&c*].

Выполняя шаг по перечислению j) для каждого ограничения ONEOF, *получаем:*

ONEOF(b, c): $D_1^{1,2}$ = [*b&c*];

D_1 = [*b&c*].

При удалении D_1 из R в соответствии с шагом по пункту 3) перечисления j) из R удаляется следующий элемент:

[a&b&d&c].

Таким образом, получаем:

R = [a&b, a&c, a].

Выражения супертипов, использующие оператор AND, отсутствуют, поэтому выполнение шага по перечислению k) не требуется.

После выполнения шага по перечислению l) получаем результат:

R = [a&b, a&c, a].

3 Данный пример демонстрирует влияние применения ограничений к сложной структуре, содержащей, по крайней мере, одно ограничение каждого возможного типа. Назначением данного примера является не моделирование конкретной ситуации, а просто демонстрация алгоритма.

```
SCHEMA complex;
ENTITY a;
END_ENTITY;
ENTITY b SUBTYPE OF (a);
END_ENTITY;
ENTITY c SUBTYPE OF (a);
END_ENTITY;
ENTITY d SUBTYPE OF (a);
END_ENTITY;
ENTITY f SUBTYPE OF (a, z);
END_ENTITY;
ENTITY k SUBTYPE OF (d);
END_ENTITY;
ENTITY l SUBTYPE OF (d, y);
END_ENTITY;
ENTITY x SUBTYPE OF (z);
END_ENTITY;
ENTITY y SUBTYPE OF (z);
END_ENTITY;
ENTITY z;
END_ENTITY;
SUBTYPE_CONSTRAINT a_subs FOR a;
   ONEOF(b, c) AND d ANDOR f;
END_SUBTYPE_CONSTRAINT;
SUBTYPE_CONSTRAINT d_subs FOR d;
   ABSTRACT;
   ONEOF(k, l);
END_SUBTYPE_CONSTRAINT;
END_SCHEMA;
```

Данная схема в формате EXPRESS-G представлена на рисунке В.3.

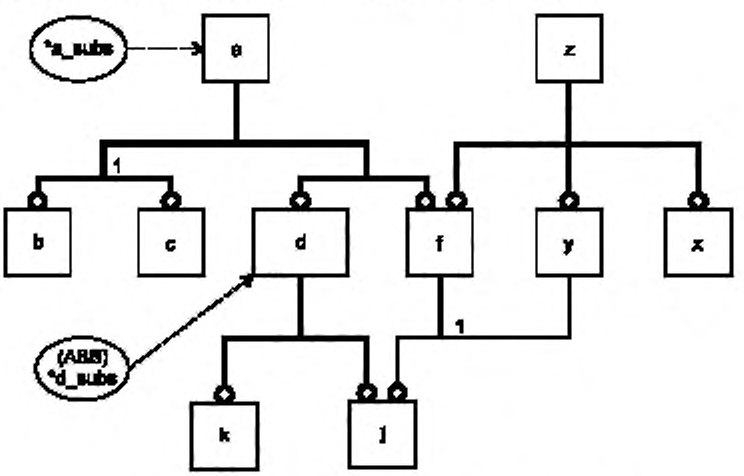

Рисунок В.3 — EXPRESS-G диаграмма схемы из примера 3

Возможные сложные объектные типы могут быть определены следующим образом:

В приведенной выше EXPRESS-схеме в явном виде представлены все объявления объектов и полные выражения супертипов, требуемые на шагах по перечислениям a) и b).

В результате выполнения шага по перечислению c) получаем:

```
SUBTYPE_CONSTRAINT z_othersubtypes FOR z;
   f ANDOR y ANDOR x;
END_SUBTYPE_CONSTRAINT;
SUBTYPE_CONSTRAINT y_othersubtypes FOR y;
   ℓ;
END_SUBTYPE_CONSTRAINT;
```

В результате выполнения шага по перечислению d), получаем:

```
SUBTYPE_CONSTRAINT stz FOR z;
   f ANDOR y ANDOR x;
END_SUBTYPE_CONSTRAINT;
SUBTYPE_CONSTRAINT sty FOR y;
   ℓ;
END_SUBTYPE_CONSTRAINT;
SUBTYPE_CONSTRAINT sta FOR a;
   ONEOF(b, c) AND d ANDOR f;
END_SUBTYPE_CONSTRAINT;
SUBTYPE_CONSTRAINT std FOR d;
   ABSTRACT;
   ONEOF(k, ℓ);
END_SUBTYPE_CONSTRAINT;
```

В результате выполнения шага по перечислению e), получаем:

$E_a \rightarrow$ [*a, a&b&d, a&b&d&f, a&c&d, a&c&d&f, a&f*];

$E_d \rightarrow$ [*d&k, d&l*];

$E_y \rightarrow$ [*l&y, y*];

$E_z \rightarrow$ [*f&x&y&z, f&x&z, f&y&z, f&z, x&y&z, x&z, y&z, z*].

В результате выполнения шага по перечислению f) преобразуются объявления корневых объектов a *и* z. *В результате получаем следующие множества:*

E_a = [*a, a&b&d&k, a&b&d&l, a&b&d&f&k, a&b&d&f&l, a&c&d&k, a&c&d&l, a&c&d&f&k, a&c&d&f&l, a&f*];

E_z = [*f&l&x&y&z, f&l&y&z, f&x&y&z, f&x&z, f&y&z, f&z, l&x&y&z, l&y&z, x&y&z, x&z, y&z, z*].

Объединяя корневые множества на шаге по перечислению g), получаем:

R = [*a, a&b&d&k, a&b&d&l, a&b&d&f&k, a&b&d&f&l, a&c&d&k, a&c&d&l, a&c&d&f&k, a&c&d&f&l, a&f, f&l&x&y&z, f&l&y&z, f&x&y&z, f&x&z, f&y&z, f&z, l&x&y&z, l&y&z, x&y&z, x&z, y&z, z*].

Ограничений подтипов полного покрытия не существует, поэтому выполнение шага по перечислению h) не требуется.

Применение шага по перечислению i) к каждому подтипу с множественным наследованием дает следующие результаты:

Для объекта f:

C_f^a = [*a&b&d&k&f, a&b&d&l&f, a&c&d&k&f, a&c&d&l&f, a&f*];

C_f^z = [*f&l&x&y&z, f&l&y&z, f&x&y&z, f&x&z, f&y&z, f&z*];

P_f = [*a&b&d&f&k&z, a&b&d&f&k&x&z, a&b&d&f&k&l&y&z, a&b&d&f&k&y&z, a&b&d&f&k&l&x&y&z, a&b&d&f&l&z, a&b&d&f&l&x&z, a&b&d&f&l&y&z, a&b&d&f&l&x&y&z, a&c&d&f&k&z, a&c&d&f&k&x&z, a&c&d&f&k&l&y&z, a&c&d&f&k&y&z, a&c&d&f&k&l&x&y&z, a&c&d&f&k&x&y&z, a&c&d&f&l&z, a&c&d&f&l&x&z, a&c&d&f&l&y&z, a&c&d&f&l&x&y&z, a&f&z, a&f&x&z, a&f&l&y&z, a&f&y&z, a&f&l&x&y&z, a&f&x&y&z*];

X_f = [*a&b&d&f&k, a&b&d&f&l, a&c&d&f&k, a&c&d&f&l, a&f, f&l&x&y&z, f&l&y&z, f&x&y&z, f&x&z, f&y&z, f&x*].

Новое множество $R = (R—X_f) + P_f$ *тогда равно:* [*a, a&b&d&f&k&z, a&b&d&f&k&x&z, a&b&d&f&k&l&y&z, a&b&d&f&k&y&z, a&b&d&f&k&l&x&y&z, a&b&d&f&k&x&y&z, a&b&d&f&l&z, a&b&d&f&l&x&z, a&b&d&f&l&y&z, a&b&d&f&l&x&y&z, a&b&d&k, a&b&d&l, a&c&d&f&k&z, a&c&d&f&k&x&z, a&c&d&f&k&l&y&z, a&c&d&f&k&y&z, a&c&d&f&k&l&x&y&z, a&c&d&f&k&x&y&z, a&c&d&f&l&z, a&c&d&f&l&x&z, a&c&d&f&l&y&z, a&c&d&f&l&x&y&z, a&c&d&k, a&c&d&l, a&f&z, a&f&x&z, a&f&l&y&z, a&f&y&z, a&f&l&x&y&z, a&f&x&y&z, l&x&y&z, l&y&z, x&y&z, x&z, y&z, z*].

Для объекта l:

C_l^d = [a&b&d&f&k&l&y&z, a&b&d&f&k&l?&x&y&z, a&b&d&f&l&z, a&b&d&f&l&x&z, a&b&d&f&l&y&z, a&b&d&f&l&x&y&z, a&b&d&l, a&c&d&f&k&l&y&z, a&c&d&f&k&l&x&y&z, a&c&d&f&l&z, a&c&d&f&l&x&z, a&c&d&f&l&y&z, a&c&d&f&l&x&y&z, a&c&d&l];

C_l^y = [a&b&d&f&k&l&y&z, a&b&d&f&k&l&x&y&z, a&b&d&f&l&y&z, a&b&d&f&l&x&y&z, a&c&d&f&k&l&y&z, a&c&d&f&k&l&x&y&z, a&c&d&f&l&y&z, a&c&d&f&l&x&y&z, a&f&l&y&z, a&f&l&x&y&z, l&x&y&z, l&y&z];

P_l = [a&b&c&d&f&k&l&y&z, a&b&c&d&f&k&l&x&y&z, a&b&c&f&k&l&y&z, a&b&c&f&l&x&y&z, a&b&d&f&k&l&y&z, a&b&d&f&k&l&x&y&z, a&b&d&f&l&y&z, a&b&d&f&l&x&y&z, a&b&d&l&x&y&z, a&b&d&l&y&z, a&c&d&f&k&l&y&z, a&c&d&f&k&l&x&y&z, a&c&d&f&l&y&z, a&c&d&f&l&x&y&z, a&c&d&l&x&y&z, a&c&d&l&y&z];

X_l = [a&b&d&f&k&l&y&z, a&b&d&f&k&l&x&y&z, a&b&d&f&l&z, a&b&d&f&l&x&z, a&b&d&f&l&y&z, a&b&d&f&l&x&y&z, a&b&d&l, a&c&d&f&k&l&y&z, a&c&d&f&k&l&x&y&z, a&c&d&f&l&z, a&c&d&f&l&x&z, a&c&d&f&l&y&z, a&c&d&f&l&x&y&z, a&c&d&l, a&f&l&y&z, a&f&l&x&y&z, l&x&y&z, l&y&z].

Новое множество $R = (R—X_l) + P_l$ *тогда равно:* [a, a&b&c&d&f&k&l&y&z, a&b&c&d&f&k&l&x&y&z, a&b&c&f&l&y&z, a&b&c&f&l&x&y&z, a&b&d&f&k&l&y&z, a&b&d&f&k&l&x&y&z, a&b&d&f&k&x&z, a&b&d&f&k&y&z, a&b&d&f&k&x&y&z, a&b&d&f&k&z, a&b&d&f&l&y&z, a&b&d&f&l&x&y&z, a&b&d&k, a&b&d&l&x&y&z, a&b&d&l&y&z, a&c&d&f&k&l&x&y&z, a&c&d&f&k&l&y&z, a&c&d&f&l&y&z, a&c&d&f&l&x&y&z, a&c&d&f&k&x&z, a&c&d&f&k&y&z, a&c&d&f&k&x&y&z, a&c&d&f&k&z, a&c&d&k, a&c&d&l&x&y&z, a&c&d&l&y&z, a&f&z, a&f&x&z, a&f&y&z, a&f&x&y&z, x&y&z, x&z, y&z, z].

Выполняя шаг по перечислению j) для каждого ограничения ONEOF, *получаем:*

ONEOF (b, c): $C_1^{1,2}$ = [b&c];

D_1 = [b&c].

В результате удаления D_1 *из R в соответствии с шагом по пункту 3) перечисления j) из R удаляются следующие элементы:*

[a&b&c&d&f&k&l&y&z, a&b&c&d&f&k&l&x&y&z, a&b&c&f&l&y&z, a&b&c&f&l&x&y&z].

Следовательно, мы имеем следующее множество:

R = [a, a&b&d&f&k&l&y&z, a&b&d&f&k&l&x&y&z, a&b&d&f&k&x&z, a&b&d&f&k&y&z, a&b&d&f&k&x&y&z, a&b&d&f&k&z, a&b&d&f&l&y&z, a&b&d&f&l&x&y&z, a&b&d&k, a&b&d&l&x&y&z, a&b&d&l&y&z, a&c&d&f&k&l&x&y&z, a&c&d&f&k&l&y&z, a&c&d&f&l&y&z, a&c&d&f&l&x&y&z, a&c&d&f&k&x&z, a&c&d&f&k&y&z, a&c&d&f&k&x&y&z, a&c&d&f&k&z, a&c&d&k, a&c&d&l&x&y&z, a&c&d&l&y&z, a&f&z, a&f&x&z, a&f&y&z, a&f&x&y&z, x&y&z, x&z, y&z, z].

ONEOF (k, *l*): $D_2^{1,2}$ = [k&l];

D_2 = [k&l].

В результате удаления D_2 *из R в соответствии с шагом по пункту 3) перечисления j) из R удаляются следующие элементы:*

[a&b&d&f&k&l&y&z, a&b&d&f&k&l&x&y&z, a&c&d&f&k&l&y&z, a&c&d&f&k&l&x&y&z].

Следовательно, мы имеем следующее множество:

R = [a, a&b&d&f&k&x&z, a&b&d&f&k&y&z, a&b&d&f&k&x&y&z, a&b&d&f&k&z, a&b&d&f&l&y&z, a&b&d&f&l&x&y&z, a&b&d&k, a&b&d&l&x&y&z, a&b&d&l&y&z, a&c&d&f&l&y&z, a&c&d&f&l&x&y&z, a&c&d&f&k&x&z, a&c&d&f&k&y&z, a&c&d&f&k&x&y&z, a&c&d&f&k&z, a&c&d&k, a&c&d&l&x&y&z, a&c&d&l&y&z, a&f&z, a&f&x&z, a&f&y&z, a&f&x&y&z, x&y&z, x&z, y&z, z].

Выполняя шаг по перечислению k) для каждого ограничения AND, *получаем:*

ONEOF (b, c) AND d: Q_1 = [b&d, c&d];

D_1^b = [ ];

D_1^c = [ ];

D_1^d = [ ];

D_1 = [ ].

После удаления D_1 из R в соответствии с шагом по пункту 4) перечисления k) R остается неизменным. Следовательно, имеем следующее множество:

R = [a, a&b&d&f&k&x&z, a&b&d&f&k&y&z, a&b&d&f&k&x&y&z, a&b&d&f&k&z, a&b&d&f&l&y&z, a&b&d&f&l&x&y&z, a&b&d&k, a&b&d&l&x&y&z, a&b&d&l&y&z, a&c&d&f&l&y&z, a&c&d&f&l&x&y&z, a&c&d&f&k&x&z, a&c&d&f&k&y&z, a&c&d&f&k&x&y&z, a&c&d&f&k&z, a&c&d&k, a&c&d&l&x&y&z, a&c&d&l&y&z, a&f&z, a&f&x&z, a&f&y&z, a&f&x&y&z, x&y&z, x&z, y&z, z].

После выполнения шага по перечислению i) получаем результат:

R = [a, a&b&d&f&k&x&z, a&b&d&f&k&y&z, a&b&d&f&k&x&y&z, a&b&d&f&k&z, a&b&d&f&l&y&z, a&b&d&f&l&x&y&z, a&b&d&k, a&b&d&l&x&y&z, a&b&d&l&y&z, a&c&d&f&l&y&z, a&c&d&f&l&x&y&z, a&c&d&f&k&x&z, a&c&d&f&k&y&z, a&c&d&f&k&x&y&z, a&c&d&f&k&z, a&c&d&k, a&c&d&l&x&y&z, a&c&d&l&y&z, a&f&z, a&f&x&z, a&f&y&z, a&f&x&y&z, x&y&z, x&z, y&z, z].

Приложение С
(обязательное)

Ограничения на экземпляры, налагаемые спецификацией интерфейса

При построении интерфейсов со сложными графами подтипов/супертипов, допустимые сложные объектные типы данных вычисляют посредством расширения правил, определенных в разделе 11 и приложении В. Граф подтипов/супертипов, определенный в одной или нескольких других схемах, может быть усечен для использования в данной схеме посредством указания только тех объектов, которые необходимы в данной схеме.

В настоящем приложении установлены правила, необходимые для интерпретации графов подтипов/супертипов в которых с одним или несколькими объектными типами данных, изначально имеющимися в графе, не установлены интерфейсы. Такие пропущенные объектные типы данных объектов оставляют пустые места в выражениях супертипов. В данном приложении такие пустые места обозначены как **<>**. Для удаления пустых мест из выражения супертипа используются следующие преобразования:

ONEOF (A, <>, . . .) → ONEOF(A, . . .);
ONEOF (<>) → <> ;
ONEOF (A) → A ;
A AND <> → ONEOF(A,A);
A ANDOR <> → A ;
TOTAL_OVER (A,<>, . . .) → TOTAL_OVER(A, . . .);
TOTAL_OVER (<>) → <> .

Интерпретация оператора **AND** должна обеспечить, чтобы те объектные типы данных, которые в исходной схеме должны объединяться, не могли бы существовать в данной схеме (обеспечивается оператором **ONEOF(A,A)**), если с объектными типами данных, с которыми они должны объединяться, не установлены интерфейсы.

Результирующее множество допустимых сложных объектных типов данных для схемы, у которой установлены интерфейсы с другими схемами, вычисляют по следующему алгоритму:

a) создается полный пул объектов для данной схемы. Полный пул включает в себя следующие объекты:
1) все объекты, определенных в данной схеме,
2) все объекты, импортированных в данную схему посредством операторов **USE** и **REFERENCE**,
3) все объекты, неявно импортированные в данную схему.

Примечание — Полный пул объектов может содержать несколько объектов с одинаковым именем (в случае их неявного импортирования из разных схем) или может включать в себя один и тот же объект под разными именами (в случае использования оператора **USE FROM . . . AS**). В первом случае пул будет содержать все объекты с одинаковыми именами, а во втором — только один объект, несмотря на наличие у него нескольких имен;

b) для каждого супертипа из пула объектов сокращается выражение супертипа, посредством удаления всех ссылок на объекты, отсутствующие в пуле объектов. Данное преобразование выполняется многократно, чтобы удалить образовавшиеся пустые места и получить истинное выражение супертипа, в котором присутствуют ссылки только на объекты из пула объектов;

c) вычисляется результирующее множество в соответствии с алгоритмом, установленным в приложении В, начиная с шага по перечислению b) и с применением преобразований, определенных в начале данного приложения, к ограничениям, полученным в результате выполнения шагов по перечислениям b), c) и h) алгоритма из приложения В, раздел В.3.

Сложный объектный тип данных из результирующего множества, содержащий, по крайней мере, один локально объявленный или импортированный посредством оператора **USE** объект, может быть реализован автономно. Сложный объектный тип данных, не содержащий таких объектов, не может быть реализован автономно в данной схеме.

Примечание — Если существует импортированный в явном виде объект, не присутствующий в какомлибо сложном объектном типе данных из результирующего множества, то такой объект вообще не может быть реализован. Вероятно, данный объект был импортирован по ошибке.

Примеры

1 Для демонстрации данного алгоритма используется схема* example *(см. пример 1 из приложения В, раздел В.3).

```
SCHEMA test;
USE FROM example (l);
REFERENCE FROM example (m, c);
END_SCHEMA;
```

Возможные сложные объектные типы данных определяются следующим образом:

Пул объектов состоит из l, m, c, a, p: l, m и c импортированы в явной форме, а и p импортированы в неявной форме, поскольку они входят в цепочку супертипов объекта l .

Сокращая выражение супертипа для p и преобразовывая его в соответствии с шагом по перечислению b), получаем:

```
ONEOF(m, f)   AND  ONEOF  (c, a)
ONEOF(m, <>)  AND  ONEOF  (c, a)
ONEOF (m)     AND  ONEOF  (c ,a)
      m       AND  ONEOF  (c ,a)
```

Аналогично получаем для a:

```
ONEOF (l, i )
ONEOF (l, <>)
ONEOF (l)
l
```

В данном случае выражения супертипов уже сформированы в виде, необходимом для шага по перечислению c).

Применяя алгоритм вычисления результирующего множества на шаге по перечислению c), получаем результирующее множество: R = [c&m&p, a&l&m&p].

Сложный объектный тип данных a&l&m&p содержит объект l, явно импортированный посредством оператора USE, *и поэтому он может быть реализован автономно. С другой стороны, сложный объектный тип данных c&m&p не может быть реализован автономно в данной схеме.*

2 Пусть имеются следующие схемы:

```
SCHEMA s1;
ENTITY e1 SUPERTYPE OF (e11 ANDOR e12); END_ENTITY;
ENTITY e11 SUBTYPE OF (e1); END_ENTITY;
ENTITY e12 SUBTYPE OF (e1); END_ENTITY;
END_SCHEMA;
SCHEMA s2;
USE FROM s1 (e11 AS f);
ENTITY e211 SUBTYPE OF (f); END_ENTITY;
ENTITY e212 SUBTYPE OF (f); END_ENTITY;
END_SCHEMA;
SCHEMA s3;
USE FROM s1 (e12 as g);
ENTITY e321 SUBTYPE OF (g); END_ENTITY;
ENTITY e322 SUBTYPE OF (g); END_ENTITY;
END_SCHEMA;
```

Результирующими множествами для данных схем являются:

s1 [e1, e1&e11, e1&e12, e1&e11&e12];

s2 [e1&f, e1&f&e211, e1&f&e212, e1&f&e211&e212];

s3 [e1&g, e1&g&e321, e1&g&e322, e1&g&e321&e322].

Если определена следующая схема test:

```
SCHEMA test;
USE FROM s2 (e211);
USE FROM s3 (e322);
END_SCHEMA;
```

то возможные сложные объектные типы данных для нее определяются следующим образом:

Пул объектов состоит из e211, e322, f, g, e1: e211 *и* e322 *импортированы в явной форме,* a f, g *и* e1 *импортированы в неявной форме, поскольку они входят в цепочку супертипов* e211 *и* e322. f *и* g *являются переименованиями* e11 *и* e12, *соответственно, поэтому* e11 *и* e12 *являются фактическими членами пула объектов.*

Сокращая выражение супертипа для e1 *и преобразовывая его в соответствии с шагом по перечислению b), получаем:*

e11 ANDOR e12;
f ANDOR *g*;

для f получаем:

e211 ANDOR e212;
e211 ANDOR <>;
e211;

для g получаем:

e321 ANDOR e322;
<> ANDOR e322;
e322.

В данном случае выражения супертипов уже сформированы в виде, необходимом для шага по перечислению c).

Применяя алгоритм вычисления результирующего множества на шаге по перечислению c), получаем результирующее множество:

R = [e1, e1&*f*, e1&*g*, e1&*f*&*g*, e1&*f*&e211, e1&*f*&*g*&e211, e1&*g*&e322, e1&*f*&*g*&e322, e1&*f*&*g*&e211&e322].

Сложные объектные типы данных e1&*f*&e211, e1&*f*&*g*&e211, e1&*g*&e322, e1&*f*&*g*&e322 *и* e1&*f*&*g*&e211&e322 *содержат один из явно импортированных посредством оператора* USE *объектов* e211 *или* e322 *и поэтому могут быть реализованы автономно. С другой стороны* e1, e1&*f*, e1&*g* *и* e1&*f*&*g* *не могут быть реализованы автономно в данной схеме.*

Приложение D
(обязательное)

Графическое подмножество языка EXPRESS — EXPRESS-G

D.1 Введение и обзор

EXPRESS-G является формальной графической нотацией, предназначенной для изображения спецификаций данных, определенных в языке EXPRESS. Данная нотация поддерживает подмножество языка EXPRESS.

EXPRESS-G поддерживает:

- разные уровни абстракции данных;
- диаграммы, размещаемые на нескольких страницах;
- диаграммы, использующие минимальные возможности компьютерной графики.

Нотация EXPRESS-G представлена графическими обозначениями, образующими диаграмму. В нотации используется три типа обозначений:

- обозначения определений — обозначения, представляющие простые типы данных, именованные типы данных, конструкционные типы данных и объявления схем;
- обозначения взаимосвязей — обозначения, представляющие взаимосвязи, существующие между определениями;
- обозначения компоновки — обозначения, позволяющие размещать диаграммы на нескольких страницах.

EXPRESS-G поддерживает простые типы данных, именованные типы данных, взаимосвязи и мощность множеств. Кроме того, EXPRESS-G поддерживает представление одной или нескольких схем. EXPRESS-G не поддерживает механизмы ограничений, предоставляемые языком EXPRESS.

Примечание — EXPRESS-G может использоваться как самостоятельный язык определения данных, поскольку не требуется иметь соответствующую спецификацию на языке EXPRESS.

Пример — На рисунках D.1 и D.2 представлена EXPRESS-G диаграмма для схемы на языке EXPRESS, определенной в примере из приложения J, раздел J.1. Диаграмма представлена на двух страницах, чтобы показать, как создаются многостраничные диаграммы.

Основные элементы диаграммы показывают, что личность (объект **person**) ***имеет некоторые определяющие характеристики, включая имя (*** **first_name**), ***фамилию (*** **last_name**), ***необязательный псевдоним (*** **nickname**), ***дату рождения (*** **birth_date**) ***и описание волос (*** **hair**). ***Личность может быть мужчиной (объект*** **male**) ***или женщиной (объект*** **female**). ***Мужчина может иметь жену (*** **wife**) ***женского пола; в этом случае женщина имеет мужа (*** **husband**) ***мужского пола. Личность может иметь детей (*** **children**), ***которые также являются личностями.***

D.2 Обозначения определений

Определения типов данных и схем на диаграмме обозначаются прямоугольниками, в которых указано имя определяемого элемента. Взаимосвязи между элементами обозначаются линиями, соединяющими прямоугольники. Для разных типов определений и взаимосвязей используются разные стили линий.

D.2.1 Обозначения простых типов данных

Простой тип данных языка EXPRESS обозначается прямоугольником, ограниченным сплошными линиями с трех сторон и двойной вертикальной сплошной линией справа. Имя типа данных указывается внутри прямоугольника, как показано на рисунке D.3.

D.2.1.1 Обозначения обобщенных типов данных

Для обозначения типа данных **GENERIC_ENTITY** используется такой же прямоугольник, что и для обозначения простого типа данных языка EXPRESS. Имя типа данных указывается внутри прямоугольника, как показано на рисунке D.4.

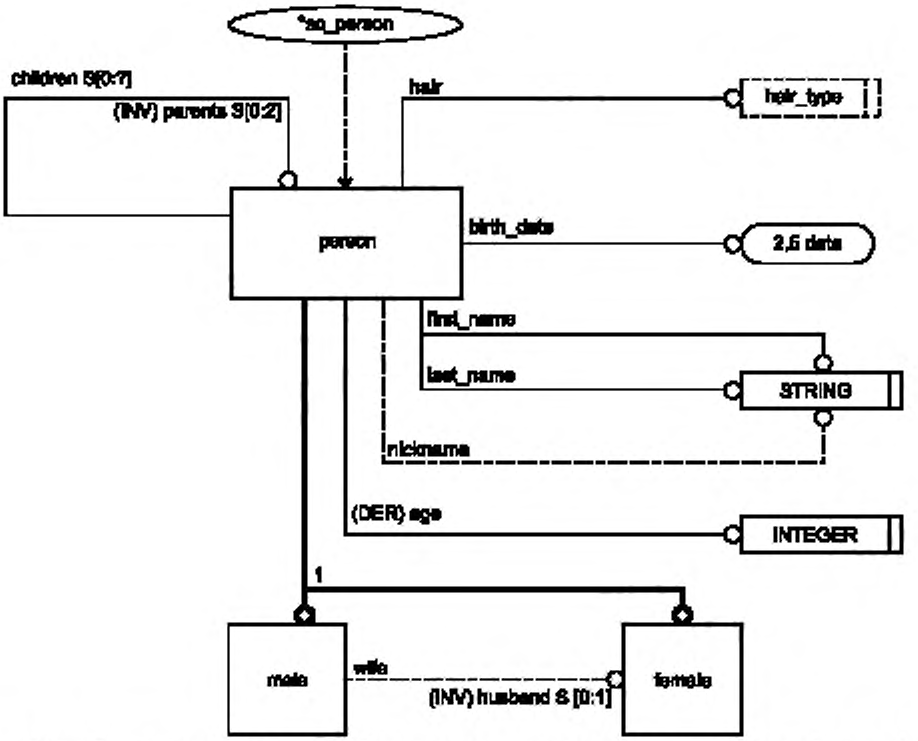

Рисунок D.1 — Полная диаграмма уровня объектов для примера из приложения J, раздел J.1 (лист 1 из 2)

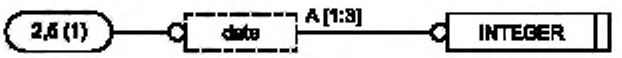

Рисунок D.2 — Полная диаграмма уровня объектов для примера из приложения J, раздел J.1 (лист 2 из 2)

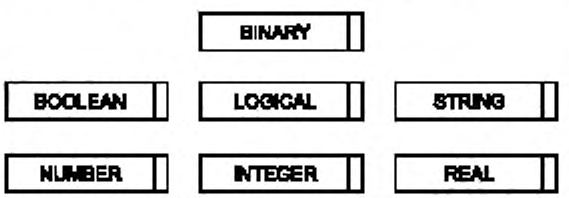

Рисунок D.3 — Обозначения простых типов данных языка EXPRESS



Рисунок D.4 — Обозначение типа данных **GENERIC_ENTITY** языка EXPRESS

D.2.2 Обозначения конструкционных типов данных

Конструкционные типы данных языка EXPRESS — **SELECT** и **ENUMERATION** обозначаются прямоугольниками, ограниченными пунктирными линиями. Имя типа данных указывается внутри прямоугольника, как показано на рисунке D.5.



Рисунок D.5 — Символы для конструкционных типов данных языка EXPRESS

Обозначение типа данных **SELECT** состоит из пунктирного прямоугольника с двойной вертикальной пунктирной линией слева. Если выбираемым типом данных является **GENERIC_ENTITY SELECT**, то перед именем типа данных ставится символ «звездочка» (*).

Обозначение типа данных **ENUMERATION** состоит из пунктирного прямоугольника с двойной вертикальной пунктирной линией справа. EXPRESS-G не обеспечивает представления списка с перечислением.

Примечание — Поскольку простые типы данных и тип данных **ENUMERATION** являются в EXPRESS-G элементарными типами данных, обозначение типа данных **ENUMERATION** похоже на обозначение простого типа данных, в котором также используется двойная вертикальная линия справа.

В языке EXPRESS допускается использование только типов данных **SELECT** и **ENUMERATION** для представления определенного типа данных. EXPRESS-G предоставляет сокращенную нотацию, в которой имя определенного типа данных помещается внутри пунктирного прямоугольника, обозначающего типы данных **SELECT** или **ENUMERATION**, вместо имени типа данных, а специального обозначения определенного типа данных не существует, как показано на рисунке D.6 (см. D.5.4).



Рисунок D.6 — Сокращенные обозначения конструкционных типов данных языка EXPRESS при их использовании для представления определенных типов данных

Пример — Две диаграммы на рисунке D.7 эквивалентны.

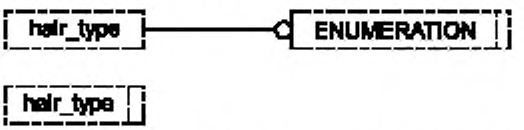

Рисунок D.7 — Пример альтернативных методов представления перечисляемого типа данных

При реализации инструментальных средств редактирования для EXPRESS-G может использоваться полная форма представления конструкционных типов данных, сокращенная форма или обе формы вместе. Разработчик инструментальных средств редактирования для EXPRESS-G должен указать, какая из этих форм применена, используя приложение E.

D.2.2.1 Наращиваемые конструкционные типы данных

Наращиваемые конструкционные типы данных обозначаются в EXPRESS-G посредством символов **EX**, заключенных в круглые скобки, то есть **(EX)**, размещенных перед именем конструкционного типа данных, как показано на рисунке D.8.

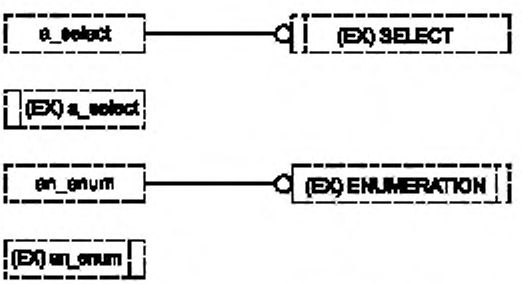

Рисунок D.8 — Обозначение наращиваемых конструкционных типов данных языка EXPRESS

D.2.3 Обозначение определенных типов данных

Определенный тип данных обозначается пунктирным прямоугольником, в котором указано имя типа, как показано на рисунке D.9.

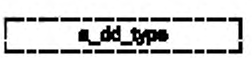

Рисунок D.9 — Обозначение определенного типа данных языка EXPRESS

D.2.4 Обозначение объектных типов данных

Объектный тип данных обозначается прямоугольником, ограниченным сплошными линиями, в котором указано имя объекта, как показано на рисунке D.10.

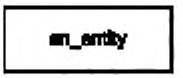

Рисунок D.10 — Обозначение объектного типа данных языка EXPRESS

D.2.5 Обозначение ограничений подтипов

Ограничение подтипа (конструкция **SUBTYPE_CONSTRAINT**) обозначается эллипсом, в котором указано имя ограничения, как показано на рисунке D.11.



Рисунок D.11 — Обозначение конструкции языка EXPRESS **SUBTYPE_CONSTRAINT**

D.2.6 Обозначение функций и процедур

В языке EXPRESS-G не поддерживаются обозначения операторов **FUNCTION** и **PROCEDURE**.

D.2.7 Обозначение правил

В языке EXPRESS-G не поддерживается обозначение оператора **RULE**. Имена объектов, являющихся параметрами оператора **RULE**, могут быть помечены звездочкой (см. D.5.3).

D.2.8 Обозначение схем

Обозначением схемы (см. рисунок D.12) является прямоугольник, ограниченный сплошными линиями и разделенный пополам горизонтальной линией. В верхней половине прямоугольника указывается имя схемы. Нижняя половина прямоугольника остается пустой.

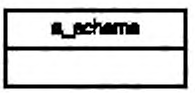

Рисунок D.12 — Обозначение схемы

D.3 Обозначение взаимосвязей

Обозначения определений соединяются линиями различных стилей, показанных на рисунке D.13.

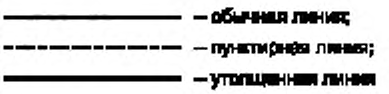

Рисунок D.13 — Стили линий, обозначающих взаимосвязи

Взаимосвязь с необязательным атрибутом объектного типа данных, определенным как **OPTIONAL**, представляется пунктирной линией. Ссылка между схемами представляется пунктирной линией. Пунктирная линия может также связывать эллипс, обозначающий ограничение подтипа, и прямоугольник, обозначающий ограниченный супертип. Отношение наследования (то есть взаимосвязь между подтипом и супертипом) представляется утолщенной линией. Расширение одного конструкционного типа данных другим также представляется утолщенной линией. Все прочие взаимосвязи представляются сплошными линиями обычной толщины.

Взаимосвязи являются двунаправленными, но одно из двух направлений является главным. Если объект A имеет явный атрибут, которым является объект B, то главным является направление от A к B. В EXPRESS-G взаимосвязь помечается незаштрихованным кружком в главном направлении, в данном случае в конце линии у объекта B. Для отношения наследования главным является направление к подтипу, то есть кружок располагается в конце линии со стороны подтипа. Для расширения конструкционных типов данных главным является направление к конструкционному типу данных, основанному на наращиваемом типе данных (то есть кружок располагается в конце линии со стороны конструкционного типа данных, основанного на наращиваемом конструкционном типе данных).

Пример — Направления взаимосвязей показаны на рисунке D.14, который является неполным представлением кода на языке EXPRESS из примера, приведенного в приложении J, раздел J.2. Диаграмма содержит шесть объектных типов данных, три определенных типа данных и несколько простых типов данных. Объект **super** ***имеет два подтипа с именами*** **sub_1** ***и*** **sub_2**. ***Объект*** **sub_2** ***имеет атрибут вы-***

бираемого типа данных с именем **choice**, ***представляющего выбор между объектным типом данных с именем*** **an_ent** ***и определенным типом данных*** **name**. ***Атрибутом объектного типа данных*** **an_ent** ***является целочисленный тип данных, а*** **name** ***является строковым типом данных.***

Атрибутом объектного типа данных объекта **sub_1** ***является объектный тип данных*** **from_ent**, ***необязательным атрибутом которого является*** **to_ent**, ***а обязательным — действительный тип данных. В свою очередь, обязательным атрибутом объектного типа данных*** **to_ent** ***является определенный тип данных с именем*** **strings**, ***а*** **strings** ***является списком (не показанным на диаграмме) строкового типа данных.***

Примечания

1 Хотя в приведенной диаграмме показаны только прямые линии взаимосвязей, линии могут иметь любую конфигурацию (например, быть кривыми).

2 Не всегда может оказаться удобным изобразить диаграмму без взаимного пересечения линий взаимосвязей. Способы различения точек пересечения определяются разработчиком диаграммы.

D.4 Обозначение компоновки диаграмм

Графические представления могут располагаться на нескольких страницах. Каждая страница нумеруется. Обозначение межстраничных ссылок показано на рисунке D.15.

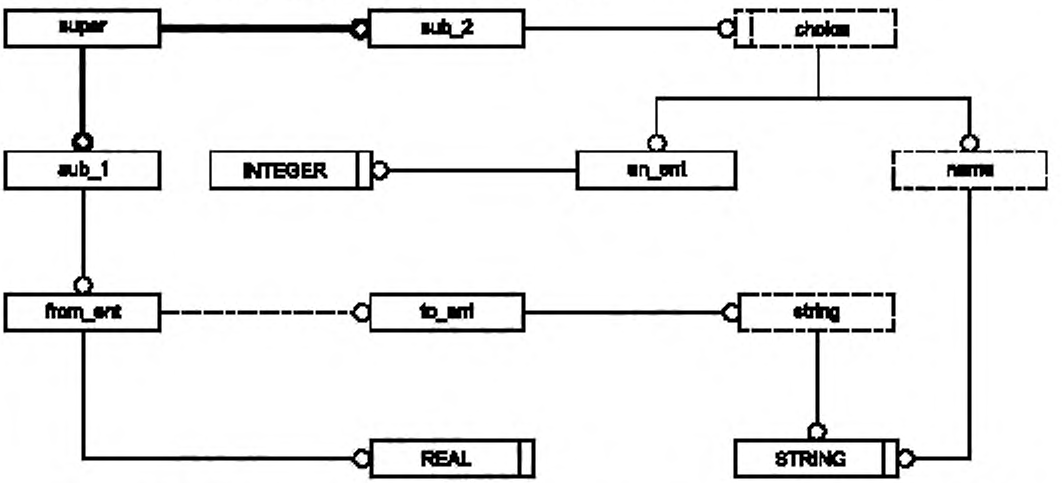

Рисунок D.14 — Частная диаграмма уровня объектов, иллюстрирующая направления взаимосвязей для примера из приложения J, раздел J.2 (лист 1 из 1)

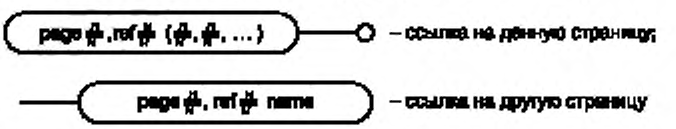

Рисунок D.15 — Обозначение компоновки диаграммы: ссылки между страницами

Схема может содержать ссылки на определения из других схем. Обозначение ссылок между схемами показано на рисунке D.16.

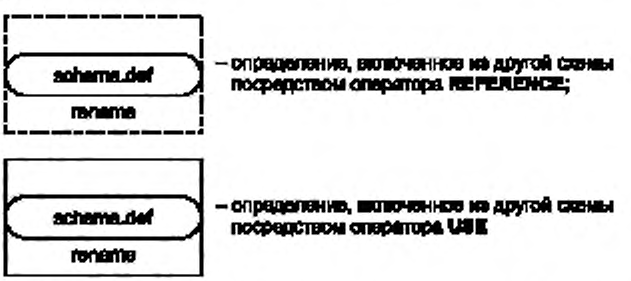

Рисунок D.16 — Обозначение компоновки диаграммы: ссылки между схемами

D.4.1 Ссылки между страницами

Если существует взаимосвязь между определениями на разных страницах, линия взаимосвязи, связывающая две страницы, заканчивается скругленным прямоугольником на каждой из них. Скругленный прямоугольник содержит номер страницы и номер ссылки, как показано на рисунке D.15. Номер страницы определяет номер страницы, содержащей определение, на которое дается ссылка. Номер ссылки используется для различения нескольких ссылок на странице. Обозначение компоновки диаграммы на странице, на которой размещена ссылка, содержит имя определения, на которое дана ссылка. Скругленный прямоугольник ссылки на странице, на которую дана ссылка, может содержать заключенный в круглые скобки список номеров страниц, из которых исходят ссылки.

Примечание — Использование межстраничных ссылок показано на рисунках D.1 и D.2. Скругленный прямоугольник, помеченный «**2, 5**», исходящий от определения объекта **person**, указывает, что определение, на которое дается ссылка, находится на странице диаграммы с номером 2 под ссылкой с номером 5. На странице диаграммы с номером 2, показанной на рисунке D.2, в скругленном прямоугольнике, направленном на определение объекта **date** указано, что на данное определение ссылка сделана из другого определения, расположенного на другой странице диаграммы. Число в круглых скобках указывает, что ссылающийся элемент расположен на странице диаграммы с номером 1.

D.4.2 Ссылки между схемами

Ссылки между схемами обозначаются скругленным прямоугольником, содержащим имя определения, уточненное именем схемы, как показано на рисунке D.16.

Скругленный прямоугольник с определением, к которому осуществляется доступ из другой схемы посредством оператора **REFERENCE**, помещается внутри прямоугольника, ограниченного пунктирными линиями. Если определение переименовывается при импорте, то новое имя может быть указано под скругленным прямоугольником.

Скругленный прямоугольник с определением, к которому осуществляется доступ из другой схемы посредством оператора **USE**, указывается в прямоугольнике, ограниченном сплошными линиями. Если определение переименовывается при импорте, то новое имя может быть указано под скругленным прямоугольником.

Примечание — Использование ссылок между схемами показано на рисунке D.24.

D.5 Диаграммы уровня объектов

Графическая нотация EXPRESS-G может быть использована для представления определений и взаимосвязей между ними в рамках одной схемы. Данный тип диаграммы может содержать обозначения простых типов данных, определенных типов данных, объектных типов данных, ограничений подтипов и взаимосвязей, а также информацию о ролях и мощности множеств, необходимую для представления содержания одной схемы.

D.5.1 Имена ролей

В языке EXPRESS для атрибута объектного типа данных указывается имя роли типа данных, на который дается ссылка, когда экземпляр участвует во взаимосвязи, установленной данным атрибутом. Текстовая строка с именем роли может быть размещена на линии взаимосвязи, соединяющей обозначение объектного типа данных с обозначением его атрибута. Данные имена ролей должны согласовываться с областью видимости и правилами видимости, определенными в разделе 10.

D.5.2 Мощности множеств

Атрибуты объектных и определенных типов данных могут быть представлены агрегированными типами данных (**LIST, SET, BAG и ARRAY**). В графической нотации EXPRESS-G агрегированная структура указывается на линии взаимосвязи для атрибута сразу после имени атрибута. Используется только первая буква названия агрегированного типа данных (то есть **A, B, L** или **S**), а ключевое слово **OF** опускают. Исключением является использование типа данных **AGGREGATE**, что указывают на линии взаимосвязи для атрибута посредством пустой пары квадратных скобок (**[ ]**), а не буквы. Если агрегированная структура не специфицирована, то мощность множества принимается равной единице для обязательной взаимосвязи и нулю или единице — для необязательного атрибута.

Примечание — Полное представление в графической нотации EXPRESS-G примера из приложения J, раздел J.2 представлено на рисунке D.17. Для компонентов типа данных **SELECT** имена ролей не присвоены.

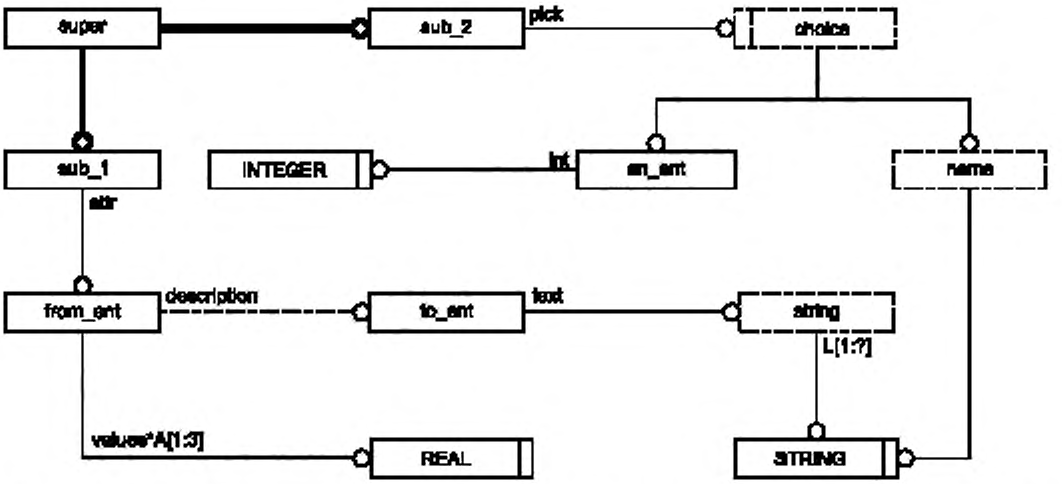

Рисунок D.17 — Полная диаграмма уровня объектов для примера из приложения J, раздел J.2 (лист 1 из 1)

D.5.3 Ограничения

Графическая нотация EXPRESS-G не предоставляет способов определения ограничений, кроме ограничений на мощность множеств. Тот факт, что некоторый элемент ограничен в спецификации данных на языке EXPRESS, может быть обозначен указанием символа звездочки (*) перед именем данного элемента. При этом применяют следующие правила:

- если объект является параметром оператора **RULE**, то имени объекта может предшествовать звездочка;
- если атрибут объекта ограничен в рамках объекта условием **UNIQUE** или **WHERE**, то имени атрибута может предшествовать звездочка;
- если определенный тип данных ограничен условием **WHERE**, то имени определенного типа данных может предшествовать звездочка;
- если агрегированный тип данных ограничен ключевым словом **UNIQUE**, то первому символу имени агрегированной структуры может предшествовать звездочка.

D.5.4 Конструкционные и определенные типы данных

Тип данных **SELECT** представляется обозначением выбираемого типа данных (см. рисунок D.5) с определением взаимосвязи и типа данных для каждого из элементов выбора. Для взаимосвязей не указывают ни мощность множества, ни имя роли.

Тип данных **ENUMERATION** представляется только своим обозначением (см. рисунок D.5).

Примечание — Графическая нотация EXPRESS-G не обеспечивает механизм для обозначения элементов перечисления.

Определенный тип данных представляется обозначением определения типа (см. рисунок D.9), содержащим имя определения, определением типа данных представления и линией взаимосвязи, направленной от определения определенного типа данных к определению типа данных представления. На линии взаимосвязи может быть указана мощность представления.

Примечание — Представление определенного типа данных показано на рисунке D.14 на примере типа данных **strings**.

Взаимосвязь расширения между наращиваемым конструкционным типом данных и основанным на нем конструкционным типом данных обозначается утолщенной линией, связывающей наращиваемый тип данных с его расширениями. Поскольку может существовать более одного расширения наращиваемого типа данных, то линия, связывающая наращиваемый тип данных с его расширениями, может иметь ответвления. Наращиваемый тип данных может иметь несколько линий, ведущих к его расширениям. Конец линии со стороны наращиваемого типа данных не имеет специального обозначения. Конец линии со стороны расширения обозначается незаштрихованным кружком.

Примечание — Два расширения наращиваемого выбираемого типа данных, один из которых сам является наращиваемым, показаны на рисунке D.18.

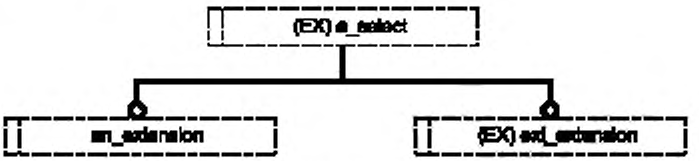

Рисунок D.18 — Диаграмма наращиваемого выбираемого типа данных

D.5.5 Объектные типы данных

Определения **ENTITY** обозначают в EXPRESS-G прямоугольником, ограниченным прямыми линиями (см. рисунок D.10). Имя объектного типа данных указывают внутри прямоугольника.

В графической нотации EXPRESS-G объект может:

- быть частью ациклического графа наследования;
- иметь явные атрибуты;
- иметь вычисляемые атрибуты;
- иметь инверсные атрибуты.

Каждый явный или вычисляемый атрибут объекта языка EXPRESS отображается взаимосвязью в соответствующей EXPRESS-G диаграмме. Имя роли атрибута вместе со следующей за ним мощностью могут быть указаны на линии взаимосвязи. Вычисляемый атрибут отличается от явного атрибута тем, что перед именем вычисляемого атрибута указываются символы **DER** в круглых скобках, то есть **(DER).**

В случае, когда для данного атрибута определен инверсный атрибут, имя и мощность инверсного атрибута указывают на линии взаимосвязи с противоположной стороны относительно имени атрибута, для которого он является инверсным. Перед именем инверсного атрибута указывают символы **INV** в круглых скобках, то есть **(INV).**

Инверсия может не быть прямой (простой) инверсией явного атрибута. Объект, на который дается ссылка в объявлении инверсного атрибута может быть подтипом объекта, объявившего прямую взаимосвязь. Другие виды непрямых инверсных взаимосвязей описаны в 9.2.1.3. К таким непрямым инверсным взаимосвязям применяют следующие правила:

- инверсный атрибут обозначается обычной линией, связывающей объект, в котором определен инверсный атрибут, с объектом (или с обозначениями компоновки диаграммы для ссылок между страницами или схемами, представляющими объект), представляющим цель инверсного атрибута. Целью инверсного атрибута может быть объект, объявляющий явный атрибут, для которого он является инверсией или подтипом данного объекта;
- конец линии со стороны объекта, содержащего инверсный атрибут, обозначается незаштрихованным кружком;
- конец линии со стороны целевого объекта не имеет стиля конца;
- имена исходного явного атрибута и объекта, в котором данный атрибут объявлен, размещаются рядом с линией в круглых скобках в виде (имя_объекта.имя_атрибута);
- символы **INV**, заключенные в круглые скобки, то есть **(INV)**, помещаются перед именем инверсного атрибута, которое также помещается рядом с линией;
- если инверсный атрибут ограничен правилом **WHERE** или **UNIQUE**, то имени атрибута предшествует звездочка в позиции верхнего индекса (*);
- если инверсный атрибут определен агрегированным типом данных, то данный агрегированный тип данных показывается в соответствии с D.5.2 после имени атрибута;
- если инверсный атрибут является повторно объявленным атрибутом, то у него перед символами **(INV)** должны быть указаны символы **RT**, заключенные в круглые скобки, то есть **(RT)**. Если в данном повторном объявлении атрибут переименовывается, то новое имя атрибута указывается после исходного имени, и эти имена разделяются символом «больше чем» (**>**).

Примечания

1 Типичные диаграммы уровня объектов показаны на рисунках D.1 и D.17.

2 Обозначение правил области определения, примененных к атрибутам, можно увидеть на рисунке D.1 на примере ролей **husband** и **maiden_name.**

3 Примерами объектов, ограниченных правилами, являются объекты **male** и **female** на рисунке D.1.

Подтипы и супертипы.

Объекты, формирующие граф наследования, соединяются утолщенными сплошными линиями. Кружок на конце линии взаимосвязи обозначает ее конец со стороны подтипа. Когда супертип является абстрактным супертипом, то есть определен как **ABSTRACT**, в прямоугольнике, обозначающем объект, перед именем объекта указываются символы **ABS** в круглых скобках, то есть **(ABS)**. Если в **SUBTYPE_CONSTRAINT** объявлено ограничение **ABSTRACT SUPERTYPE**, то обозначение **(ABS)** должно предшествовать имени соответствующего ограничения **SUBTYPE_CONSTRAINT** в эллипсе, обозначающем данное ограничение, как показано на рисунке D.19.

Если объект объявлен как **ABSTRACT** (не **ABSTRACT SUPERTYPE**, а **ABSTRACT ENTITY**), то символы **AE**, заключенные в круглые скобки, то есть **(AE)**, предшествуют имени объекта в прямоугольнике, обозначающем данный объект, как показано на рисунке D.20.

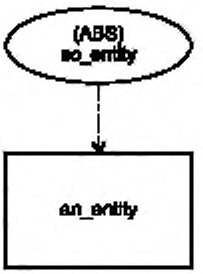

Рисунок D.19 — Обозначение ограничения **ABSTRACT SUPERTYPE**, если данное ограничение определено в конструкции **SUBTYPE_CONSTRAINT**



Рисунок D.20 — Обозначение ограничения **ABSTRACT ENTITY**

Графическая нотация EXPRESS-G обеспечивает ограниченные возможности для представления логической структуры графа наследования. Отношение **ONEOF** может быть представлено разветвляющейся линией взаимосвязи, направленной от супертипа к каждому из его подтипов, связанных друг с другом отношением **ONEOF**, и цифрой **1**, расположенной у точки разветвления. Если отношение **ONEOF** задано конструкцией **SUBTYPE_CONSTRAINT**, то перед именем **SUBTYPE_CONSTRAINT** в эллипсе, обозначающем данную конструкцию, должна стоять звездочка.

Отношение **AND** может быть представлено разветвляющейся линией взаимосвязи, направленной от супертипа к каждому из его подтипов, связанных друг с другом отношением **AND**, и символом **&**, расположенным у точки разветвления. Если отношение **AND** задано конструкцией **SUBTYPE_CONSTRAINT**, то перед именем **SUBTYPE_CONSTRAINT** в эллипсе, обозначающем данную конструкцию, должна стоять звездочка.

Представление ограничения **TOTAL_OVER** связано с обозначением ограничения **SUBTYPE_CONSTRAINT**, составной частью которого оно было определено (см. рисунок D.21). Между эллипсом, обозначающим ограничение **SUBTYPE_CONSTRAINT**, и прямоугольником, обозначающим ограниченный объектный супертип, должна быть проведена пунктирная линия; стиль этой линии не должен зависеть от существования ограничения **TOTAL_OVER**. Между эллипсом, обозначающим ограничение **SUBTYPE_CONSTRAINT**, и прямоугольниками, обозначающими объектные подтипы, которые обеспечивают полное покрытие супертипа, должны быть проведены линии обычной толщины. Все эти линии должны заканчиваться открытыми стрелками у прямоугольников, обозначающих объектный супертип и его подтипы, включенные в данное ограничение. Кроме того, линии, исходящие от **SUBTYPE_CONSTRAINT**, могут заканчиваться у обозначений компоновки диаграммы для ссылок между страницами и схемами.

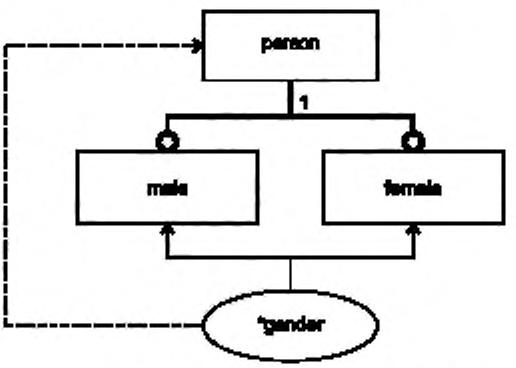

Рисунок D.21 — Пример ограничения покрытия **TOTAL_OVER**

Пример — На рисунке D.21 представлена следующая модель:

```
ENTITY person;
END_ENTITY;
ENTITY male SUBTYPE OF (person);
END_ENTITY;
ENTITY female SUBTYPE OF (person);
END_ENTITY;
SUBTYPE_CONSTRAINT gender FOR person;
   TOTAL_OVER (male, female);
   ONEOF (female, male);
END_SUBTYPE_CONSTRAINT;
```

Примечания

1 На рисунке D.22 представлена EXPRESS-G диаграмма для примера из приложения J, раздел J.3, изображающая объект **sub2** в качестве абстрактного супертипа.

2 На рисунке D.15 показано, что объекты **sub1**, **sub2** и **sub5** являются подтипами супертипа **super**. Экземпляр супертипа **super** может не иметь подтипов, поскольку **super** не является абстрактным супертипом. Объекты **sub3** и **sub4** являются подтипами супертипа **sub2**. Объекты **sub3** и **sub4** связаны друг с другом отношением **ONEOF**.

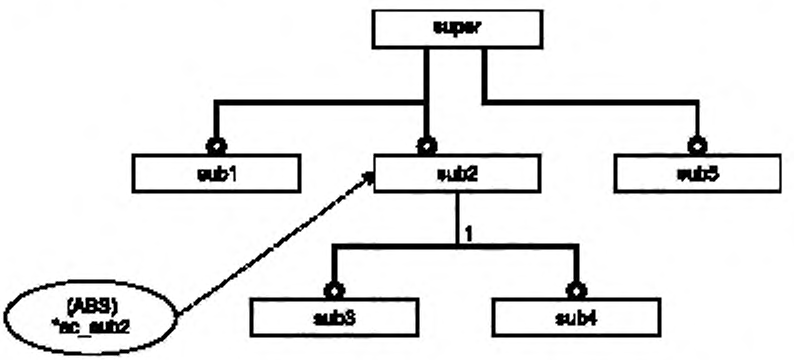

Рисунок D.22 — Полная диаграмма уровня объектов графа наследования для примера из приложения J, раздел J.3 (лист 1 из 1)

Язык EXPRESS допускает повторное объявление атрибутов супертипа в подтипе. При этом повторно объявленный атрибут является конкретизацией типа данных атрибута супертипа. Если повторное объявление атрибута включает в себя также его переименование, то новое имя указывается после исходного имени и отделяется от него символом «больше чем» (**>**). В графической нотации EXPRESS-G повторно объявленный атрибут представляется так же, как атрибут его супертипа, но с добавлением символов **RT** в круглых скобках, то есть **(RT)** перед именем атрибута.

Примечание — Некоторые формы повторного объявления атрибутов, представленных в примере на языке EXPRESS из приложения J, раздел J.4 показаны на рисунке D.23. Объект **sub_a** повторно объявляет атрибут **attr** из своего супертипа как подтип атрибута своего супертипа. Объект **sup_b** имеет необязательный атрибут типа **NUMBER**. В его подтипе данный атрибут повторно объявлен как обязательный атрибут типа **REAL**.

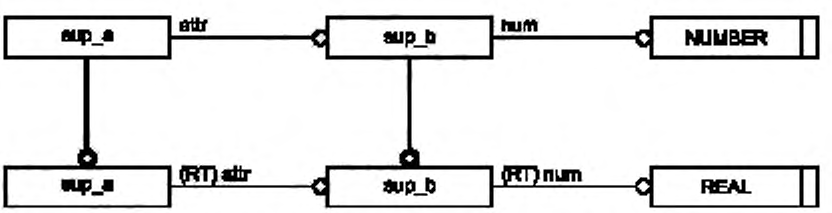

Рисунок D.23 — Полная диаграмма уровня объектов для примера из приложения J, раздел J.4, показывающая повторные объявления атрибутов в подтипах (лист 1 из 1)

D.5.6 Ссылки между схемами

Если определение из данной схемы ссылается на определение из другой схемы, используется обозначение ссылки между схемами, содержащее уточненное имя определения.

Примечание — Диаграмма уровня объектов для одной схемы показана на рисунке D.24. Исходный текст на языке EXPRESS для данной диаграммы приведен в примере 1 из приложения J, раздел J.5. Полная диаграмма состоит из двух схем — **top** и **geom** (см. рисунок D.25), а некоторые объекты схемы **top** имеют атрибуты, использующие определения из схемы **geom**. Поскольку диаграмма уровня объектов состоит только из элементов, определенных в одной схеме, для представления схемы **top** в данном примере необходимо использовать ссылки между схемами.

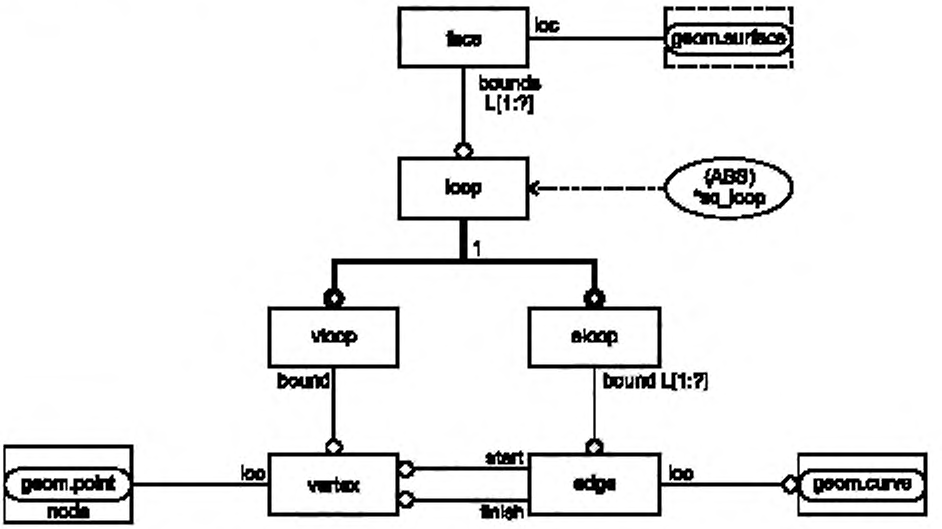

Рисунок D.24 — Полная диаграмма уровня объектов схемы top для примера 1 из приложения J, раздел J.5, иллюстрирующая ссылки между схемами (лист 1 из 1)

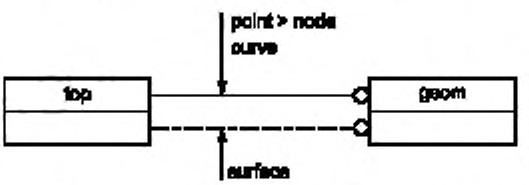

Рисунок D.25 — Полная диаграмма уровня схем для примера 1 из приложения J, раздел J.5 (лист 1 из 1)

D.6 Диаграммы уровня схем

Диаграмма уровня схем включает в себя представление нескольких схем и интерфейсов между ними.

Содержимое EXPRESS-G диаграммы уровня схем ограничено схемами, представленными в диаграмме, и интерфейсами между ними. Диаграмма уровня схем может содержать следующие элементы:

- схемы, ссылающиеся на другие схемы посредством оператора **USE**;
- схемы, ссылающиеся на другие схемы посредством оператора **REFERENCE**;
- имена элементов, импортированных посредством операторов **USE** или **REFERENCE**.

Интерфейс **USE** представляется линией взаимосвязи обычной толщины, направленной от импортирующей схемы к импортируемой схеме, с незаштрихованным кружком, обозначающим импортируемую схему. Интерфейс **REFERENCE** представляется пунктирной линией взаимосвязи, направленной от импортирующей схемы к импортируемой схеме, с незаштрихованным кружком, обозначающим импортируемую схему.

Импортированные определения могут быть показаны в виде списка имен, расположенного рядом с соответствующей линией взаимосвязи и соединенного с линией взаимосвязи посредством линии, заканчивающейся стрелкой, указывающей на линию взаимосвязи. Переименование определения представляется его исходным именем, за которым следуют символ «больше чем» (**>**) и новое имя определения.

Примечание — Диаграмма с двумя схемами показана на рисунке D.25. Схема **top** имеет интерфейс со схемой **geom**. В частности, схема **top** ссылается на объект **surface** и использует определения **curve** и **point** из схемы **geom**. Определение **point** в схеме **top** переименовано в **node**.

Если диаграмма уровня схем размещается на нескольких страницах, а интерфейсы между схемами пересекают границы страниц, то используются обозначения межстраничных ссылок.

Примечание — В примере 2 из приложения J, раздел J.5 представлен исходный текст на языке EXPRESS для сокращенной версии диаграммы уровня схем. EXPRESS-G диаграмма для данного примера показана на рисунке D.26.

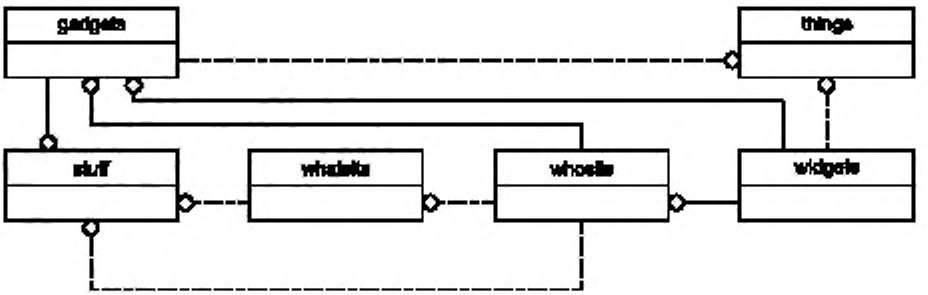

Рисунок D.26 — Полная диаграмма уровня схем для примера 2 из приложения J, раздел J.5 (лист 1 из 1)

D.7 Полные EXPRESS-G диаграммы

Полной EXPRESS-G диаграммой называется диаграмма, в которой с учетом ограничений графической нотации EXPRESS-G точно представлены все определения, взаимосвязи и ограничения посредством диаграммы уровня объектов или уровня схем.

D.7.1 Полная диаграмма уровня объектов

Содержание диаграмм, представляющих полную диаграмму одной схемы, определяется следующими правилами:

a) каждая страница должна иметь заголовок, начинающийся со слов: «Complete entity level diagram of...» (полная диаграмма уровня объектов для ...);

b) каждая страница нумеруется в виде «Page X of N» (страница X из N), где N — общее число страниц диаграммы, а X — номер данной страницы;

c) должны быть показаны обозначения всех объектных типов данных, определенных типов данных и простых типов данных, используемых в данной схеме;

d) не должно быть обозначений схем;

e) должны быть показаны все взаимосвязи, имена атрибутов и мощности множеств;

f) должны быть показаны все атрибуты, включая явные, вычисляемые и инверсные;

g) должны быть показаны все взаимосвязи наследования (между подтипами и супертипами);

h) должны быть помечены все ограничения **ABSTRACT SUPERTYPE**;

i) должны быть помечены все взаимосвязи подтипов **ONEOF**;

j) все определения, импортированные из других схем, должны быть обозначены скругленными прямоугольниками, заключенными в прямоугольники соответствующего стиля (то есть ограниченными сплошными линиями для определений, импортированных посредством оператора **USE**, и пунктирными линиями — для определений, импортированных посредством оператора **REFERENCE**);

k) любое переименование должно быть представлено в соответствующем обозначении ссылки между схемами;

l) все объекты, ограниченные оператором **RULE**, должны быть помечены звездочкой (*);

m) все атрибуты, на которые наложены ограничения, должны быть помечены звездочкой (*);

n) все определенные типы данных, на которые наложены ограничения, должны быть помечены звездочкой (*);

o) все агрегированные типы данных, на которые наложены ограничения, должны быть помечены звездочкой (*);

p) все объявления **ABSTRACT ENTITY** должны быть помечены;

q) все ограничения **TOTAL_OVER** должны быть помечены;

r) все взаимосвязи между наращиваемыми конструкционными типами данных и их расширениями должны быть показаны;

s) все новые и старые имена атрибутов, переименованных при повторном объявлении, должны быть показаны;

t) все ограничения **GENERIC_ENTITY SELECT** должны быть показаны;

u) все типы данных **GENERIC_ENTITY**, использованные в данной схеме, должны быть показаны.

Все взаимосвязи между объектами, не помеченные инверсным атрибутом, интерпретируются как имеющие мощность, равную нулю или больше нуля. Непомеченное отношение подтипов не может служить основанием для вывода о наличии логического структурирования, если только данное отношение не является отношением **ONEOF**.

D.7.2 Полная диаграмма уровня схем

Содержание диаграмм, представляющих полную диаграмму уровня схем, определяется следующими правилами:

a) каждая страница должна иметь заголовок, начинающийся со слов: «Complete schema level diagram of... » (полная диаграмма уровня схем для ...);

b) каждая страница нумеруется в виде «Page X of N» (страница X из N), где N — общее число страниц диаграммы, а X — номер данной страницы;

c) должны быть показаны все использованные схемы;

d) не должны показываться обозначения объектов, типов данных и простые символы;

e) должны быть показаны все отношения между схемами, представленные операторами **USE** и **REFERENCE**;

f) имена всех определений, импортированных посредством операторов **USE** или **REFERENCE**, должны быть привязаны к соответствующей линии взаимосвязи вместе с их переименованиями. Если к линии взаимосвязи не привязано никаких имен, то это интерпретируется как схема, целиком импортированная посредством операторов **USE** или **REFERENCE**.

Примечание — При разработке моделей или графических диаграмм полезно иметь возможность представлять диаграммы на разных уровнях абстракции. Например, на конкретной диаграмме могут быть представлены не все атрибуты или показаны не все имена ролей. Подобное представление находится вне области определения EXPRESS-G, но рекомендуется, чтобы уровень абстракции был согласован и документирован до начала разработки. Кроме того, рекомендуется, чтобы в заголовке диаграммы были отражены используемые уровни абстракции.

Приложение Е
(обязательное)

Заявка о соответствии реализации протоколу (ЗСРП)

Является ли данная реализация синтаксическим анализатором или верификатором языка EXPRESS? Если да, то следует ответить на вопросы из Е.1.

Является ли реализация средством редактирования EXPRESS-G? Если да, то следует ответить на вопросы из Е.2.

Е.1 Синтаксический анализатор языка EXPRESS

Для какого уровня заявляется поддержка:

☐ Уровень 1 — проверка ссылок;
☐ Уровень 2 — проверка типов данных;
☐ Уровень 3 — проверка значений;
☐ Уровень 4 — полная проверка.

Примечание — Для того, чтобы заявить о поддержке данного уровня, поддержка всех нижележащих уровней также должна быть обеспечена.

Каково максимальное целочисленное значение [integer_literal]?: .....:

Какова максимальная точность действительных чисел [real_literal]?: .....:

Каков максимальный показатель степени действительных чисел [real literal]?: .....:

Какова максимальная длина строки (в символах) [simple_string_literal]?: .....:

Какова максимальная длина строки (в октетах) [encoded_string_literal]?: .....:

Какова максимальная длина двоичных чисел (в битах) [binary_literal]?: .....:

Существует ли ограничение на число объявленных уникальных идентификаторов? Если да, то чему оно равно?: .....:

Существует ли ограничение на число символов в идентификаторе? Если да, то чему оно равно?: .....:

Существует ли ограничение на глубину вложения областей видимости? Если да, то чему оно равно?: .....:

Реализуется ли концепция нескольких областей видимости имен, в которых могут появляться имена схем? Если да, то как называются эти области видимости?: .....:

Как представлена стандартная константа ' ? ' [built_in_constant]?: .....:

Е.2 Средство редактирования EXPRESS-G

Для какого уровня заявляется поддержка:

☐ Уровень 1 — проверка обозначений;
☐ Уровень 2 — полная проверка.

Примечание – Для того, чтобы заявить о поддержке данного уровня, поддержка всех нижележащих уровней также должна быть обеспечена.

Существует ли ограничение на число объявленных уникальных идентификаторов? Если да, то чему оно равно?: .....:

Существует ли ограничение на число символов в идентификаторе? Если да, то чему оно равно?: .....:

Существует ли ограничение на число обозначений на странице модели? Если да, то чему оно равно?: .....:

Существует ли ограничение на число страниц в модели? Если да, то чему оно равно?: .....:

Реализуется ли концепция нескольких областей видимости имен, в которых могут появляться имена схем? Если да, то как называются эти области видимости?: .....:

Реализуется ли полная форма представления конструкционных типов данных, сокращенная форма или обе формы одновременно?: .....:

Приложение F
(обязательное)

Регистрация информационного объекта

F.1 Обозначение документа

Для обеспечения однозначного обозначения информационного объекта в открытой системе настоящему стандарту присвоен следующий идентификатор объекта:

{ iso standard 10303 part(11) version(4) }

Смысл данного обозначения установлен в ИСО/МЭК 8824-1 и описан в ИСО 10303-1.

F.2 Обозначение синтаксиса

Для обеспечения однозначного обозначения информационного объекта в открытой системе синтаксису языка EXPRESS присвоен следующий идентификатор объекта:

{ iso standard 10303 part(11) version(4) object(1) EXPRESS-syntax (1) }

Смысл данного обозначения установлен в ИСО/МЭК 8824-1 и описан в ИСО 10303-1.

Приложение G
(обязательное)

Генерация одной схемы из нескольких схем

G. 1 Введение

Модель данных на языке EXPRESS состоит, по крайней мере, из одной схемы, но может состоять и из нескольких схем. Существует много способов построения мультисхемной спецификации, некоторые из которых перечислены ниже.

Часть спецификации может быть определена в одной схеме, которая повторно может использоваться в другой схеме с целью конкретизации или расширения.

Схема верхнего уровня может импортировать несколько схем, чтобы построить полную спецификацию данных.

Полная спецификация может состоять из нескольких независимых схем.

В исходных методах реализации, включенных в стандарты комплекса ИСО 10303, предполагалась модель данных, состоящая из одной схемы, называемой схемой в длинной форме. До тех пор, пока методы реализации, включенные в стандарты комплекса ИСО 10303, не обновляются, чтобы соответствовать настоящей редакции ИСО 10303-11, или реализации не модифицируются, чтобы соответствовать новым методам реализации, может быть достаточно преобразовать спецификацию данных, соответствующую настоящему стандарту, в спецификацию данных, соответствующую предыдущей редакции 1994 г. В настоящем приложении определены правила выполнения данного преобразования. Эти правила обеспечивают преобразование мультисхемной спецификации, соответствующей требованиям настоящего стандарта, в схему в длинной форме, соответствующую предыдущей редакции 1994 г. Данные правила разработаны, чтобы получить в результате полную и непротиворечивую длинную форму и минимизировать потери семантики исходной модели данных.

G.2 Основные понятия

Объекты, состоящие во взаимосвязи супертип-подтип, образуют направленный граф, который должен быть ациклическим (многокорневым) деревом. Аналогично, схема, которая связана спецификациями интерфейсов **USE** или **REFERENCE**, образует направленный граф, который может быть циклическим, и узлами которого являются схемы, а ребрами – спецификации интерфейсов. В общем случае спецификация модели данных состоит из одного или нескольких графов схем. В частности, такой граф может иметь одну корневую схему, в которую не направлена какая-либо спецификация интерфейса, но из которой можно достичь все другие схемы. Корневая схема может рассматриваться в качестве представителя данного графа. В других случаях в графе может существовать одна или несколько основных схем, тогда как другие схемы в данном графе существуют только для поддержки основной схемы. Корневая и основная схемы играют особую роль в процессе преобразования.

Исходными данными для процесса генерации схемы в длинной форме являются корневая и основная схемы графа, заключающего в себе спецификацию модели данных.

Результатом данного процесса является одна схема, содержащая все конструкции из исходных схем плюс необходимые поддерживающие конструкции из других схем, присутствующих во множестве графов. К поддерживающим конструкциям относятся конструкции, которые явно или неявно импортированы в корневую и основную схему.

Результирующая схема в длинной форме почти семантически идентична модели без операторов **USE** и **REFERENCE**; объекты, интерфейс с которыми установлен посредством оператора **REFERENCE**, вносятся непосредственно в данную схему. Информация, описывающая исходные схемы, отбрасывается. Объекты в импортированных схемах, на которые непосредственно не ссылаются объекты из корневой или основной схемы, также отбрасываются. Проводится сокращение и перезапись некоторых конструкций из исходных схем, чтобы не включать объекты, объявленные первоначально, но не используемые в конечной схеме.

Процесс преобразования состоит из двух этапов, на каждом из которых происходит потеря семантики:

а) мультисхемная спецификация данных преобразовывается в спецификацию промежуточной единой схемы. При этом проводятся следующие основные преобразования:

1) выбираемые элементы типа данных **SELECT** сокращают за счет удаления элементов, не импортируемых в схему. Согласно 11.4.2 выбираемые элементы не являются неявно импортируемыми, поскольку импортируется сам тип данных **SELECT**. Если выбираемые элементы остались в списке выбора, а соответствующие им объекты не видимы в схеме, то результатом компиляции схемы в длинной форме будет ошибка,

2) сокращают ограничения **SUBTYPE_CONSTRAINT**, чтобы отразить сокращение графа подтипов/супертипов в соответствии с 11.4.3 и приложением С,

3) сокращают правила **RULE**, чтобы отразить сокращение графа подтипов/супертипов в соответствии с 11.4.3 и приложением С,

4) имена схемы в полностью уточненных ссылках на атрибуты заменяют именем схемы в длинной форме,

5) знания о том, как конструкции были импортированы, то есть их видимость и реализуемость, преобразовываются в правила. Информация, описывающая как объект был импортирован (учитывая различие между операторами **USE** и **REFERENCE**), влияет на его реализуемость и видимость; см. 11.2 и 11.3.

При этом может быть потеряна следующая семантическая информация:

теряются знания о схеме, из которой происходит каждая конструкция, комментарии, не имеющие меток комментария (см. 7.1.6.3), могут быть отброшены, регистр символов в идентификаторах пользователя может не сохраниться;

b) представление промежуточной единой схемы переписывают с использованием только конструкций из ИСО 10303-11:1994 для создания конечной схемы в длинной форме. Преобразование из сокращенной формы промежуточного представления в схему в длинной форме требует удаления или изменения конструкций языка EXPRESS, не поддерживаемых предыдущей редакцией ИСО 10303-11. В частности, проводят следующие основные действия с сопутствующими семантическими потерями:

1) типы данных **EXTENSIBLE SELECT** заменяют типами данных **SELECT** в соответствии с ИСО 10303-11:1994, не поддерживающим расширение,

2) типы данных **EXTENSIBLE ENUMERATION** заменяют типами данных **ENUMERATION** в соответствии с ИСО 10303-11:1994, не поддерживающим расширение,

3) операторы **SUBTYPE_CONSTRAINT** удаляют:

их ограничения **SUPERTYPE** и операторы **ABSTRACT** преобразовывают в операторы **SUPERTYPE**, соответствующие ИСО 10303-11:1994, и переписывают, чтобы удалить типы данных, которые в других обстоятельствах не появились бы в схеме в длинной форме,

ограничения **TOTAL_OVER** заменяются конструкциями **RULE**,

4) **ABSTRACT ENTITY** и **GENERIC_ENTITY** преобразовывают в ограничения **ABSTRACT SUPERTYPE**,

5) конструкции **RENAMED** преобразовывают в атрибуты **DERIVE**,

6) для пустых типов данных **SELECT** должно быть сформировано сообщение об ошибке.

Спецификация промежуточной единой схемы является артефактом процесса преобразования и не используется вне данного процесса.

Перечисленные выше действия более подробно описываются в последующих подразделах.

G.3 Изменение имен

G.3.1 Конфликты имен

Схема определяет область видимости имен, в которой имена, указанные в объявлениях объектов, уникальны. Имена, объявленные в других схемах, относятся к другим областям видимости и поэтому они не обязательно будут уникальными при формировании единой схемы из нескольких. Ниже представлен процесс объединения объявлений из разных схем при формировании единой схемы. Для того, чтобы обеспечить уникальность имен в такой единой схеме, любое имя, не являющееся уникальным в исходном наборе объединяемых схем, должно быть модифицировано с тем, чтобы избежать конфликта имен и обеспечить выполнение требования уникальности имен в объединенной схеме.

Перед каждым не уникальным именем добавляется имя схемы, в которой оно определено, и строка '**_dot_**'. Соответствующие изменения имен должны быть сделаны во всей исходной схеме.

Пример — Пусть объект **thing** ***объявлен в схеме*** **scha**, ***а тип данных*** **thing** ***объявлен в схеме*** **schb**. ***При объединении данных схем объект переименовывается в*** **scha_dot_thing**, ***а тип данных — в*** **schb_dot_thing**.

G.3.2 Идентификаторы, представленные строками

В схеме некоторые строки могут представлять полностью уточненные имена.

Пример — В следующем фрагменте кода:

IF 'THIS_SCHEMA.AN_ENTITY' IN TYPEOF(super)

строка 'THIS_SCHEMA.AN_ENTITY' ***представляет полностью уточненное имя.***

Если объявления, содержащие такие строки, перемещаются из своей исходной схемы в другую схему, то в данных строках часть, содержащая имя исходной схемы, должна быть модифицирована, чтобы представлять схему, в которую данные объявления перемещаются.

Пример — Пусть объявление с именем **whatsit** ***из схемы*** **scha** ***копируется в другую схему*** **schb**. ***Другое объявление, содержащее строку*** **' SCHA.WHATSIT '**, ***также копируется в*** **schb**. ***В схеме*** **schb** ***данная строка появится с именем*** **' SCHB.WHATSIT '**.

G.4 Этап 1 — преобразование нескольких схем в промежуточную схему

G.4.1 Введение

Ниже определен первый этап формирования спецификации модели данных единой схемы по ИСО 10303-11:1994 из спецификации моделей данных нескольких схем. Результатом данного этапа является спецификация модели данных в форме единой схемы по ИСО 10303-11:2003. Назовем данную схему **artifact**.

Исходными данными для преобразования на этапе 1 являются корневая и основная схемы для спецификации модели данных. Начальная спецификация модели данных должна быть ссылочно полной, то есть в ней не должно быть неопределенных ссылок.

Способ идентификации исходных схем не определяется в настоящем стандарте и остается на усмотрение разработчиков реализации.

G.4.2 Первичное содержимое

Создается новая промежуточная схема с именем **artifact**, в которую копируются все объявления и помеченные комментарии из корневой и основной схем. Схема **artifact** не должна иметь идентификатор версии **schema_version_id**. Любой конфликт имен между объявлениями и помеченными комментариями в схеме **artifact** должен быть разрешен и все представления строковых идентификаторов соответствующим образом модифицированы.

Удаляются все дубликаты в операторах **USE** и **REFERENCE**. Если какой-либо элемент появляется в обоих операторах **USE** и **REFERENCE**, то данный элемент удаляется из оператора **REFERENCE**, поскольку спецификация интерфейса посредством оператора USE имеет приоритет по отношению к оператору **REFERENCE** (см. 11.3).

Оператор **USE** позволяет рассматривать элементы из другой схемы, как объявленные локально в данной схеме (см. 11.1). Копируются все такие элементы и относящиеся к ним помеченные комментарии в схему **artifact**, при этом разрешаются все конфликты имен и модифицируются идентификаторы, представленные строками.

Если импортированный элемент должен быть переименован (см. 11.1 и 11.2), то он должен быть скопирован под своим исходным именем, а ссылки на идентификатор должны быть изменены соответствующим образом.

Затем из схемы **artifact** удаляются все операторы **USE**, поскольку все элементы, идентифицированные посредством операторов **USE**, уже скопированы в данную схему.

Примечание — Схема **artifact** теперь будет содержать все элементы, объявленные в исходных корневой и основной схемах, плюс элементы, импортированные в исходные схемы посредством операторов **USE**, плюс операторы (модифицированные) **REFERENCE** из исходных схем.

Примеры

1 Данный пример иллюстрирует копирование переименованных элементов. Дано:

```
SCHEMA sch;
    USE FROM second (alfred AS alf);
    REFERENCE FROM second (bert AS herbert);
    ENTITY joe;
        attr1 : alf;
        attr2 : herbert;
     END_ENTITY;
...
```

и

```
SCHEMA short;
    USE sch;
    ...
END_SCHEMA;
```

тогда, если схема **short** ***была для алгоритма исходной корневой схемой, а*** **alfred** ***и*** **bert** ***являются объявлениями объектов, то схема*** **artifact** ***будет иметь следующий вид:***

```
SCHEMA artifact;
    ENTITY joe;
        attr1 : alfred;
        attr2 : bert;
    END_ENTITY;
    ENTITY alfred ...
    ENTITY bert ...
...
```

2 Данный пример иллюстрирует изменение имен для разрешения конфликтов имен. Исходная спецификация содержит три схемы:

```
SCHEMA s1;
    ENTITY creature;
        – – атрибуты
    END_ENTITY;
    – – другие объявления
END_SCHEMA; – – конец схемы s1
SCHEMA farming;
    USE FROM s1 (creature);
    ENTITY dog SUBTYPE OF creature;
        – – атрибуты
    END_ENTITY;
```

```
        ENTITY shepherd;
            dogs : SET OF dog;
        END_ENTITY;
        -- другие объявления
    END_SCHEMA; -- конец схемы farming
    SCHEMA pet_shows;
        USE FROM s1 (creature);
        ENTITY pet SUBTYPE OF (creature);
            -- атрибуты
        END_ENTITY;
        ENTITY dog SUBTYPE OF pet;
            -- атрибуты
        END_ENTITY;
        ENTITY dog_show;
        dogs : SET [1:?] OF dog;
        -- другие атрибуты
        END_ENTITY;
        -- другие объявления
    END_SCHEMA; -- конец схемы pet_shows
```

Принимая схемы **farming** *и* **pet_shows** *в качестве основных схем, получаем следующую промежуточную схему:*

```
    SCHEMA artifact;
        ENTITY creature;
            -- атрибуты
        END_ENTITY;
        ENTITY farming_dot_dog SUBTYPE OF creature;
            -- атрибуты
        END_ENTITY;
        ENTITY shepherd;
            dogs : SET OF farming_dot_dog;
        END_ENTITY;
        ENTITY pet SUBTYPE OF (creature);
            -- атрибуты
        END_ENTITY;
        ENTITY pet_shows_dot_dog SUBTYPE OF pet;
            -- атрибуты
        END_ENTITY;
        ENTITY dog_show;
            dogs : SET [1 : ?] OF pet_shows_dot_dog;
            -- другие атрибуты
        END_ENTITY;
        -- другие объявления
    END_SCHEMA; — конец схемы artifact
```

G.4.3 Вторичное содержимое

Проверяют каждый элемент, идентифицированный в схеме **artifact** в операторе **REFERENCE**. Если данный элемент необходим для обеспечения ссылочной полноты, то в схему **artifact** из исходной схемы копируют объявление и относящиеся к нему помеченные комментарии. Если данный элемент принадлежит к объектному типу данных, то сохраняется семантика оператора **REFERENCE**, в соответствии с которой данный элемент должен быть реализован, если на него ссылается другой элемент, посредством следующей процедуры:

а) для первого подобного элемента в схеме **artifact** создают следующие правило и функцию:

```
RULE validate_dependently_instantiable_entity_data_types FOR
                (<здесь перечисляются данный элемент первым, а затем все относящиеся к нему объектные типы данных>);
LOCAL
    number_of_input_instances  : INTEGER;
    previous_in_chain          : LIST OF GENERIC := [ ];
    set_of_input_types         : SET OF STRING := [ ];
    all_instances              : SET OF GENERIC := [ ];
END_LOCAL;
all_instances := <создается объединение всего неявного содержимого условия FOR>;
number_of_input_instances := SIZEOF(all_instances);
```

```
(* Собираются строки всех типов для экземпляров FOR в один набор. *)
REPEAT i:=1 TO number_of_input_instances;
   set_of_input_types := set_of_input_types + TYPEOF(all_instances [ i ]);
END_REPEAT;
WHERE
   WR1: dependently_instantiated(all_instances, set_of_input_types, previous_in_chain);
END_RULE;
FUNCTION dependently_instantiated(
            set_of_input_instances : SET OF GENERIC:igen;
            set_of_input_types : SET OF STRING;
            previous_in_chain : LIST OF GENERIC:cgen): BOOLEAN;
(*Функция "dependently_instantiated" предназначена для проверки, на все ли экземпляры в исходном
    set_of_input_instances имеются ссылки от независимо реализуемых экземпляров. Если да, то функ-
    ция возвращает значение true. Set_of_input_types содержит строки типов для всех исходных экземп-
    ляров. Экземпляры в previous_in_chain используются для выявления циклических ссылок при рекур-
    сивном вызове данной функции. Параметр содержит список уже проверенных экземпляров в цепочке
    ссылок.
   *)
LOCAL
   number_of_input_instances      : INTEGER;
   number_of_referring_instances : INTEGER;
   bag_of_referring_instances     : BAG OF GENERIC:igen := [ ];
   dependently_instantiated_flag  : BOOLEAN;
   previous_in_chain_plus         : LIST OF GENERIC:cgen := [ ];
   result                         : BOOLEAN := true;
   set_of_types                   : SET OF STRING := [ ];
END_LOCAL;
IF EXISTS(set_of_input_instances) THEN
   number_of_input_instances := SIZEOF(set_of_input_instances);
   (* Объявленный тип bag_of_referring_instances добавляется к множеству типов импортированных
      посредством оператора REFERENCE экземпляров для последующего сравнения подмножеств.
    *)
   set_of_input_types := set_of_input_types + ' GENERIC ';
   REPEAT i:=1 TO number_of_input_instances;
      (* Определяются все ссылки на текущий исходный экземпляр. *)
      bag_of_referring_instances := USEDIN (set_of_input_instances [ i ] , ' ');
      IF EXISTS(bag_of_referring_instances) THEN
         number_of_referring_instances := SIZEOF(bag_of_referring_instances);
         dependently_instantiated_flag := false;
         REPEAT j:=1 TO number_of_referring_instances;
            (* Определяются строки типов текущего ссылающегося экземпляра.
             *)
            set_of_types := TYPEOF(bag_of_referring_instances [ j ]);
            (* Если ссылающийся экземпляр принадлежит к одному из типов только зависимо реализуе-
              мых выбираемых элементов, то текущий исходный экземпляр еще может быть недопустимо
              реализован. В противном случае все в порядке, и проверяется следующий исходный экземп-
              ляр.
             *)
            IF set_of_types <= set_of_input_types THEN – – оператор подмножества
               (* Ссылающийся экземпляр принадлежит к одному из ограниченных типов. Однако на него
                  самого может ссылаться допустимый экземпляр; тогда текущий экземпляр также мог бы
                  быть допустимым.
                  Таким образом, осуществляется рекурсивный вызов данной функции со ссылающимся эк-
                  земпляром в качестве входного параметра.
                  Чтобы избежать бесконечного цикла в случае, когда множество экземпляров ссылается
                  друг на друга в замкнутом цикле, сначала проверяется, присутствует ли текущий ссылаю-
                  щийся экземпляр в списке уже обработанных элементов цепочки.
             *)
           IF NOT (bag_of_referring_instances [ j ] IN previous_in_chain) THEN
            previous_in_chain_plus := previous_in_chain +
                                      set_of_input_instances [ i ];
            IF dependently_instantiated([bag_of_referring_instances [ j ]],
                                        set_of_input_types,
                                        previous_in_chain_plus) THEN
```

```
            dependently_instantiated_flag := true;
            ESCAPE; – – экземпляр, зависимо реализуемый; переход к
                    – – следующему исходному экземпляру
          ELSE
            (* Экземпляр, не зависимо реализуемый: переход к следующему ссылающемуся
              экземпляру. *)
            SKIP;
          END_IF;
        END_IF;
      ELSE
        dependently_instantiated_flag := true;
        ESCAPE; – – экземпляр, зависимо реализуемый; взять следующий
                – – исходный экземпляр
        END_IF;
    END_REPEAT;
    IF NOT dependently_instantiated_flag THEN
      RETURN  (false);
    END_IF;
  ELSE
    RETURN (false); – – на экземпляр нет ссылок => недопустимо реализован
  END_IF;
 END_REPEAT;
ELSE
 RETURN (false); – – нет исходных данных
END_IF;
RETURN (true);
END_FUNCTION; – – конец функции dependently_instantiated
```

b) к условию **FOR** и правой части оператора присваивания, в левой части которого имеется объект **all_instances** (оба эти места отмечены угловыми скобками **< >**), добавляют имена первого и всех последующих таких объектных типов данных;

c) к этим двум местам добавляют также такие объектные типы данных, которые требуются уже идентифицированным экземплярам для их ссылочной полноты, и которые не содержатся где-либо еще в длинной форме как независимо реализуемые объектные типы данных;

d) Если одна или несколько основных схем являются длинными формами, которые были созданы на основе данной процедуры, и, следовательно, уже содержатся в правиле **validate_dependently_instantiable_entity_data_types**, то добавляется содержимое их условий **FOR** к созданному содержимому. Значимым содержимым являются объектные типы данных, которые остаются зависимо реализуемыми также и после генерации текущей длинной формы, и которые уже не включены в новое правило.

Если данный элемент не требуется для ссылочной полноты, то он не должен копироваться. Конфликты имен должны быть разрешены, а строковые идентификаторы — модифицированы.

Примечание – Не скопированные объявления, которые не требуются для полноты, содержат семантику операторов **REFERENCE** данных объявлений.

Пример – В данном примере исходная модель состоит из двух схем, одна из которых не содержит спецификаций интерфейсов, а вторая, являющаяся специализированной схемой в длинной форме, содержит оператор **REFERENCE:**

```
SCHEMA export;
  ENTITY a;
    a1: STRING;
  END_ENTITY;
  ENTITY b;
    b1: STRING;
  END_ENTITY;
END_SCHEMA; – – конец схемы export
SCHEMA import;
  REFERENCE FROM export (a, b); – – только зависимо реализуемый!
  ENTITY ref;
    aref: a; – – реализация зависит от объектного типа данных ref
    bref: b; – – реализация зависит от объектного типа данных ref
  END_ENTITY;
END_SCHEMA; – – конец схемы import
```

Используя схему import *в качестве корневой схемы, получаем следующую промежуточную схему:*

```
SCHEMA artifact;
  ENTITY a;
    a1: STRING;
  END_ENTITY;
  ENTITY b;
    b1: STRING;
  END_ENTITY;
  ENTITY ref;
    aref: a; – – реализация зависит от объектного типа данных ref
    bref: b; – – реализация зависит от объектного типа данных ref
  END_ENTITY;
  RULE validate_dependently_instantiable_entity_data_types FOR
       (a, b); – – !! здесь добавлены a и b !!
  LOCAL
    number_of_input_instances : INTEGER;
    previous_in_chain       : LIST OF GENERIC := [ ];
    set_of_input_types      : SET OF STRING := [ ];
    all_instances           : SET OF GENERIC := [ ];
  END_LOCAL;
  all_instances := a+b; – – !! здесь добавлены a и b !!
  number_of_input_instances := SIZEOF(all_instances);
  (* Собираются все строки типов всех экземпляров FOR в одно множество. *)
  REPEAT i:=1 TO number_of_input_instances;
    set_of_input_types := set_of_input_types + TYPEOF(all_instances [ i ]);
  END_REPEAT;
  WHERE
    WR1: dependently_instantiated(all_instances, set_of_input_types, previous_in_chain);
  END_RULE;
  FUNCTION dependently_instantiated (
              set_of_input_instances  : SET OF GENERIC:igen;
              set_of_input_types      : SET OF STRING;
              previous_in_chain       : LIST OF GENERIC:cgen): BOOLEAN;
                 (*Функция «dependently_instantiated» предназначена для проверки, на все ли экземпляры
       в исходном set_of_input_instances имеются ссылки от независимо реализуемых экземпляров.
       Если да, то функция возвращает значение true. Set_of_input_types содержит строки типов для
       всех исходных экземпляров. Экземпляры в previous_in_chain используются для выявления цик-
       лических ссылок при рекурсивном вызове данной функции. Параметр содержит список уже про-
       веренных экземпляров в цепочке ссылок.
    *)
  LOCAL
    number_of_input_instances      : INTEGER;
    number_of_referring_instances : INTEGER;
    bag_of_referring_instances     : BAG OF GENERIC:igen := [ ];
    dependently_instantiated_flag  : BOOLEAN;
    previous_in_chain_plus         : LIST OF GENERIC:cgen := [ ];
    result                         : BOOLEAN := true;
    set_of_types                   : SET OF STRING := [ ];
  END_LOCAL;
  IF EXISTS(set_of_input_instances) THEN
    number_of_input_instances := SIZEOF(set_of_input_instances);
                  (* Объявленный тип bag_of_referring_instances добавляется к множеству типов
            импортированных посредством оператора REFERENCE экземпляров для последующего
            сравнения подмножеств.
      *)
    set_of_input_types := set_of_input_types + ' GENERIC ' ;
    REPEAT i:=1 TO number_of_input_instances;
                          (* Определяются все ссылки на текущий исходный экземпляр. *)
      bag_of_referring_instances := USEDIN (set_of_input_instances [ i ] , ' ');
      IF EXISTS(bag_of_referring_instances) THEN
        number_of_referring_instances := SIZEOF(bag_of_referring_instances);
        dependently_instantiated_flag := false;
        REPEAT j:=1 TO number_of_referring_instances;
```

```
            (* Определяются строки типов текущего ссылающегося экземпляра.
             *)
            set_of_types := TYPEOF(bag_of_referring_instances [ j ]);
                                (* Если ссылающийся экземпляр принадлежит к одному из типов только
                зависимо реализуемых выбираемых элементов, то текущий исходный экземпляр еще
                может быть недопустимо реализован. В противном случае все в порядке, и проверяет-
                ся следующий исходный экземпляр.
             *)
            IF set_of_types <= set_of_input_types THEN — оператор подмножества
                                    (* Ссылающийся экземпляр принадлежит к одному из ограничен-
                ных типов. Однако на него самого может ссылаться допустимый экземпляр; тогда
                текущий экземпляр также мог бы быть допустимым.
                    Таким образом, осуществляется рекурсивный вызов данной функции со ссы-
                лающимся экземпляром в качестве входного параметра.
                    Чтобы избежать бесконечного цикла в случае, когда множество экземпляров
                ссылается друг на друга в замкнутом цикле, сначала проверяется, присутствует ли
                текущий ссылающийся экземпляр в списке уже обработанных элементов цепочки.
            *)
            IF NOT (bag_of_referring_instances [ j ] IN previous_in_chain) THEN
              previous_in_chain_plus := previous_in_chain +
                                        set_of_input_instances [ i ];
              IF dependently_instantiated([bag_of_referring_instances [ j ]],
                                        set_of_input_types,
                                        previous_in_chain_plus) THEN
                dependently_instantiated_flag : = true;
                ESCAPE; – – экземпляр, зависимо реализуемый; переход к
                        – – следующему исходному экземпляру
              ELSE
                (* Экземпляр, не зависимо реализуемый: переход к следующему ссылающемуся
                  экземпляру. *)
                SKIP;
              END_IF;
            END_IF;
          ELSE
            dependently_instantiated_flag := true;
            ESCAPE; – – экземпляр, зависимо реализуемый; взять следующий
                    – – исходный экземпляр
          END_IF;
        END_REPEAT;
        IF NOT dependently_instantiated_flag THEN
          RETURN (false);
        END_IF;
      ELSE
        RETURN (false); – – на экземпляр нет ссылок => недопустимо реализован
      END_IF;
    END_REPEAT;
  ELSE
    RETURN (false); – – нет входных данных
  END_IF;
  RETURN (true);
  END_FUNCTION; – – конец функции dependently_instantiated
END_SCHEMA; – – конец схемы artifact
```

Когда внешнее объявление импортируется в схему, то другие объявления могут оказаться импортированными неявно (см. 11.4). В схему **artifact** копируются те неявно импортированные объявления, которые необходимы для ссылочной полноты схемы **artifact**, включая их помеченные комментарии. При этом разрешаются конфликты имен и модифицируются представления строковых идентификаторов.

Примечание — Копирование неявно импортированных элементов может быть рекурсивным процессом.

Примеры

1 В данном примере исходная модель состоит из двух схем:

```
SCHEMA export;
  TYPE colour = EXTENSIBLE ENUMERATION;
  END_TYPE;
```

```
   TYPE stop_light = ENUMERATION BASED_ON colour WITH (red, yellow, green);
   END_TYPE;
END_SCHEMA; — конец схемы export
SCHEMA import;
   USE FROM export (stop_light);
   REFERENCE FROM export (colour);
   TYPE canadian_flag = ENUMERATION BASED_ON colour WITH (red, white);
   END_TYPE;
   – – другие объявления, зависящие от canadian_flag и stop_light
   ...
END_SCHEMA; — конец схемы import
```

Используя схему import *в качестве корневой схемы, получаем следующую промежуточную схему:*

```
SCHEMA artifact;
   TYPE colour = EXTENSIBLE ENUMERATION;
   END_TYPE;
   TYPE stop_light = ENUMERATION BASED_ON colour WITH (red, yellow, green);
   END_TYPE;
   TYPE canadian_flag = ENUMERATION BASED_ON colour WITH (red, white);
   END_TYPE;
   – – другие объявления, зависящие от canadian_flag и stop_light
   ...
END_SCHEMA; – – конец схемы artifact
```

2 В данном примере исходная модель состоит из четырех схем:

```
SCHEMA s1;
   TYPE general_approval = EXTENSIBLE ENUMERATION OF (approved, rejected);
   END_TYPE;
END_SCHEMA; – – конец схемы s1
SCHEMA s2;
   USE FROM s1 (general_approval);
   TYPE domain2_approval = EXTENSIBLE ENUMERATION BASED_ON general_approval WITH (pending);
   END_TYPE;
END_SCHEMA; – – конец схемы s2
SCHEMA s3;
   USE FROM s1 (general_approval);
   TYPE domain3_approval = EXTENSIBLE ENUMERATION BASED_ON general_approval WITH (cancelled);
   END_TYPE;
END_SCHEMA; – – конец схемы s3
SCHEMA s4;
   USE FROM s2 (domain2_approval);
   REFERENCE FROM s3 (domain3_approval);
   TYPE specific_approval = ENUMERATION BASED_ON domain2_approval WITH (rework);
   END_TYPE;
   – – другие объявления, зависящие от данных типов
   ...
END_SCHEMA; — конец схемы s4
```

Используя схему s4 *в качестве корневой схемы, получаем следующую промежуточную схему:*

```
SCHEMA artifact;
   TYPE specific_approval = ENUMERATION BASED_ON domain2_approval WITH (rework);
   END_TYPE;
   TYPE domain3_approval = EXTENSIBLE ENUMERATION BASED_ON general_approval WITH (cancelled);
   END_TYPE;
   TYPE domain2_approval = EXTENSIBLE ENUMERATION BASED_ON general_approval WITH (pending);
   END_TYPE;
   TYPE general_approval = EXTENSIBLE ENUMERATION OF (approved, rejected);
   END_TYPE;
   – – другие объявления, зависящие от данных типов
   ...
END_SCHEMA; – – конец схемы artifact
```

G.4.4 Сокращение

Существуют ограничения на неявно импортированные объявления (см. 11.4).

Примеры

1 Тип данных **SUPERTYPE ENTITY** *не импортирует неявно любые подтипы данного объекта.*

2 Выбираемый тип данных не импортирует неявно любой из своих элементов выбора.

3 Правило не импортирует неявно любые объектные типы данных, указанные в его списке параметров.

Вследствие ограничений на неявные интерфейсы в схеме **artifact** могут существовать объявления, не являющиеся ссылочно полными.

Выражения ограничений супертипов должны быть сокращены в соответствии с приложением С.

Правило (оператор **RULE**), в котором не все параметры являются видимыми, должно быть сокращено посредством удаления всех невидимых параметров и относящихся к ним помеченных комментариев.

Правило может вызывать функции и процедуры. Любая функция или процедура в схеме **artifact**, которая не вызывается каким-либо объявлением из данной схемы, должна быть удалена, включая ее помеченные комментарии.

Выбираемый тип данных должен быть сокращен посредством удаления из списка выбора всех выбираемых элементов, невидимых в схеме **artifact**.

Примечание — Сокращение выбираемого типа данных может привести в результате к пустому списку выбора.

Удаляют все спецификации интерфейсов **REFERENCE**.

Примечание — На данном этапе схема **artifact** должна быть ссылочно полной. Все объявления, кроме объявлений из исходной схемы или объявлений, импортированных в исходную схему посредством оператора **USE**, должны быть необходимыми для обеспечения ссылочной полноты исходных объявлений.

Примеры

1 Данный пример иллюстрирует сокращение выбираемого типа данных. Исходная модель состоит из двух схем:

```
SCHEMA export;
  TYPE attachment_method = EXTENSIBLE SELECT (nail, screw);
  END_TYPE;
  ENTITY nail;
  END_ENTITY;
  ENTITY screw;
  END_ENTITY;
END_SCHEMA; — конец схемы export
SCHEMA import;
  USE FROM export (attachment_method,
                   nail);
  TYPE permanent_attachment = SELECT BASED_ON attachment_method WITH
      (glue, weld);
  END_TYPE;
  TYPE simple_attachment = SELECT BASED_ON attachment_method WITH
      (needle, tape);
  END_TYPE;
  – – объявления объектов glue и других
  ...
END_SCHEMA; — конец схемы import
```

Схема **import** *используется в качестве корневой схемы. Объект* **screw** *как явно, так и неявно не импортирован в схему* **import**, *поэтому он не присутствует в промежуточной схеме, которая имеет следующий вид:*

```
SCHEMA artifact;
  TYPE attachment_method = EXTENSIBLE SELECT (nail);
  END_TYPE;
  ENTITY nail;
  END_ENTITY;
  TYPE permanent_attachment = SELECT BASED_ON attachment_method WITH
      (glue, weld);
  END_TYPE;
  TYPE simple_attachment = SELECT BASED_ON attachment_method WITH
      (needle, tape);
  END_TYPE;
  – – объявления объектов glue и других
```

```
    ...
END_SCHEMA; — конец схемы artifact
```

Следует заметить, что исходный выбираемый тип данных attachment_method *из схемы* export *был сокращен посредством удаления объекта* screw *из списка выбора.*

2 Данный пример иллюстрирует случай, когда сокращение выбираемого типа данных приводит к пустому списку выбора. Исходная спецификация содержит три схемы:

```
SCHEMA s1;
  TYPE t11 = EXTENSIBLE SELECT
        (t12, t13);
  END_TYPE;
  – – объявления объектов t12, t13 и других
  ...
END_SCHEMA; – – конец схемы s1
SCHEMA s2;
  REFERENCE FROM s1 (t11);
  ENTITY e21;
    attr : t11;
  END_ENTITY;
  – – другие объявления
  ...
END_SCHEMA; – – конец схемы s2
SCHEMA s3;
  USE FROM s1 (t11);
  USE FROM s2 (e21);
  TYPE t31 = SELECT BASED_ON  t11  WITH
          (t32, t33);
  END_TYPE;
  – – объявления объектов t32, t33 и других
  ...
END_SCHEMA; – – конец схемы s3
```

Используя схему s2 *в качестве корневой схемы, получаем промежуточную схему следующего вида:*

```
SCHEMA artifact;
  ENTITY e21;
    attr : t11;
  END_ENTITY;
  TYPE t11 = EXTENSIBLE SELECT;
  END_TYPE;
  – – другие объявления
  ...
END_SCHEMA; – – конец схемы artifact
```

Схема s2 *импортирует объект* t11 *из схемы* s1, *но не импортирует какие-либо элементы из его списка выбора. После сокращения список выбора пуст и поэтому он не присутствует в окончательном представлении промежуточной схемы.*

3 Исходная модель состоит из двух схем. Данный пример иллюстрирует сокращение ограничения TOTAL_OVER *и правила* RULE.

```
SCHEMA s1;
  ENTITY e1;
  END_ENTITY;
  ENTITY e2 SUBTYPE OF (e1);
  END_ENTITY;
  ENTITY e3 SUBTYPE OF (e1);
  END_ENTITY;
  SUBTYPE_CONSTRAINT sc_total_over FOR e1;
    TOTAL_OVER (e2, e3);
  END_SUBTYPE_CONSTRAINT;
  RULE e2_and_e3 (e2, e3);
    – – тело оператора RULE
  END_RULE;
END_SCHEMA; – – конец схемы s1
SCHEMA import;
  USE FROM s1 (e1, e2);
END_SCHEMA; – – конец схемы import
```

Используя схему import *в качестве корневой схемы, получаем следующий вид промежуточной схемы до каких-либо сокращений:*

```
SCHEMA artifact;
  ENTITY e1;
  END_ENTITY;
  ENTITY e2 SUBTYPE OF (e1);
  END_ENTITY;
  SUBTYPE_CONSTRAINT sc_total_over FOR e1;
    TOTAL_OVER (e2, e3);
  END_SUBTYPE_CONSTRAINT;
  RULE e2_and_e3 (e2, e3);
    -- тело оператора RULE
  END_RULE;
END_SCHEMA; -- конец схемы artifact
```

Следует заметить, что объект e3 отсутствует.

После сокращения окончательный вид промежуточной схемы выглядит следующим образом:

```
SCHEMA artifact;
  ENTITY e1;
  END_ENTITY;
  ENTITY e2 SUBTYPE OF (e1);
  END_ENTITY;
  SUBTYPE_CONSTRAINT sc_total_over FOR e1;
    TOTAL_OVER (e2);
  END_SUBTYPE_CONSTRAINT;
END_SCHEMA; -- конец схемы artifact
```

Поскольку объекта e3 *нет в схеме* artifact, *сокращается и ограничение* SUBTYPE_CONSTRAINT. *Аналогично удаляется и правило* RULE.

4 Данный пример основан на схеме example из примера 1 из приложения B, раздел B.3 и иллюстрирует сокращение выражений супертипов. Исходными схемами по ИСО 10303-11:2003 являются следующие:

```
SCHEMA example;
  ENTITY p;
  END_ENTITY;
  SUBTYPE_CONSTRAINT p_subs FOR p;
    ONEOF(m, f) AND ONEOF(c, a);
  END_SUBTYPE_CONSTRAINT;
  ENTITY m SUBTYPE OF (p);
  END_ENTITY;
  ENTITY f SUBTYPE OF (p);
  END_ENTITY;
  ENTITY c SUBTYPE OF (p);
  END_ENTITY;
  ENTITY a ABSTRACT SUBTYPE OF (p);
  END_ENTITY;
  SUBTYPE_CONSTRAINT no_ℓi FOR a;
    ONEOF(ℓ, i);
  END_SUBTYPE_CONSTRAINT;
  ENTITY ℓ SUBTYPE OF (a);
  END_ENTITY;
   ENTITY i SUBTYPE OF (a);
  END_ENTITY;
END_SCHEMA; — конец схемы example
SCHEMA import;
  USE FROM example (ℓ);
  REFERENCE FROM example (m);
END_SCHEMA; -- конец схемы import
```

Используя схему import *в качестве корневой схемы, получают следующий вид промежуточной схемы до сокращения:*

```
SCHEMA artifact;
  ENTITY p;
  END_ENTITY;
  SUBTYPE_CONSTRAINT p_subs FOR p;
     ONEOF(m, f) AND ONEOF (c, a);
```

```
    END_SUBTYPE_CONSTRAINT;
    ENTITY m SUBTYPE OF (p);
    END_ENTITY;
    ENTITY a ABSTRACT SUBTYPE OF (p);
    END_ENTITY;
    SUBTYPE_CONSTRAINT no_ℓi FOR a;
      ONEOF (ℓ, i);
    END_SUBTYPE_CONSTRAINT;
    ENTITY ℓ SUBTYPE OF (a);
    END_ENTITY;
  END_SCHEMA; — конец схемы artifact
```

Объекты f, c *и* i *не присутствуют в данной схеме, поэтому выражения супертипов в* p_subs *и* no_ℓi *должны быть сокращены в соответствии с приложением С. Сначала берут выражение из* p_subs, *которое выглядит следующим образом:*

```
ONEOF (m, f) AND ONEOF (c, a);
```

Первое сокращение (ONEOF(A, <>) => ONEOF(A)) *в результате дает следующее сокращенное выражение:*

```
ONEOF(m) AND ONEOF(a);
```

Данное выражение может быть сокращено еще раз (посредством ONEOF(A) => A), *что приведет к следующему окончательному выражению:*

```
m AND a;
```

Таким образом, после сокращения p_subs *выглядит следующим образом:*

```
SUBTYPE_CONSTRAINT p_subs FOR p;
  m AND a;
END_SUBTYPE_CONSTRAINT;
```

Выражение для no_ℓi *выглядит следующим образом:*

```
ONEOF (ℓ, i);
```

Применяя сокращения, получаем следующий вид no_ℓi:

```
SUBTYPE_CONSTRAINT no_ℓi FOR a;
  ℓ;
END_SUBTYPE_CONSTRAINT;
```

Данное ограничение бессодержательно, поэтому данный оператор SUBTYPE_CONSTRAINT *в целом и потенциально относящиеся к нему помеченные комментарии должны быть удалены.*

Окончательным результатом после сокращения будет следующая схема:

```
SCHEMA artifact;
  ENTITY p;
  END_ENTITY;
  SUBTYPE_CONSTRAINT p_subs FOR p;
    m AND a;
  END_SUBTYPE_CONSTRAINT;
  ENTITY m SUBTYPE OF (p);
  END_ENTITY;
  ENTITY a ABSTRACT SUBTYPE OF (p);
  END_ENTITY;
  ENTITY ℓ SUBTYPE OF (a);
  END_ENTITY;
END_SCHEMA; – – конец схемы artifact
```

G.4.5 Имена и версии схем

Имена исходных схем, из которых копируются элементы в промежуточную схему, могут быть сохранены в виде встроенных комментариев. Если сохраняется имя какой-либо схемы, то и имена всех других схем также должны быть сохранены.

Встроенный комментарий должен иметь следующий формат (где **\n** обозначает конец строки, текст, заключенный в угловые скобки, является переменным, а квадратные скобки обозначают необязательный элемент):

```
(* Original 2003 schemas: \n
  schema = <schema_id> [schema_version_id = ' <version> '] ; \n
  ...
*)
```

Если схема, содержащая элемент **schema_version_id**, указывается во встроенном комментарии, то и сам элемент **schema_version_id** должен быть в нем указан.

Порядок, в соответствии с которым указываются схемы, значения не имеет.

Примеры

1 Если одной из исходных схем является **SCHEMA geometry_schema**, *то соответствующий ей встроенный комментарий должен выглядеть следующим образом:*

```
schema = geometry_schema;
```

2 Встроенный комментарий, содержащий имена следующих двух исходных схем:

```
SCHEMA schema_one;
...
SCHEMA schema_two ' version 4 ' ;
...
```

выглядит следующим образом:

```
(* Original 2003 schemas:
  schema = schema_two schema_version_id = ' version 4 ' ;
  schema = schema_one;
*)
```

G.5 Этап 2 — Преобразование промежуточной схемы в схему по ИСО 10303-11:1994

G.5.1 Введение

Ниже определен второй этап формирования спецификации модели данных единой схемы по ИСО 10303-11:1994 из многосхемного представления модели данных.

Исходными данными для данного этапа является спецификация модели данных в виде ссылочно полной единой схемы, сформированной в соответствии с G.4. Результатом данного этапа является единая ссылочно полная схема, не содержащая конструкций, не соответствующих ИСО 10303-11:1994.

Установленные ниже правила определяют, каким образом конструкции исходной схемы должны приводиться в соответствие с ИСО 10303-11:1994.

Назовем исходную схему **artifact**, а результирующую схему **longform**.

G.5.2 Инициализация

Создадим новую схему с именем **longform**. Скопируем все объявления и комментарии из промежуточной схемы в схему **longform**. Все представления строковых идентификаторов должны быть соответствующим образом модифицированы.

G.5.3 Преобразование наращиваемых конструкционных типов данных

Наращиваемые выбираемые типы данных и наращиваемые перечисляемые типы данных (см. 8.4) не определены в ИСО 10303-11:1994. Дерево, содержащее любой из этих типов данных, должно быть сокращено до одного нерасширяемого типа данных. Конструкционные типы данных, не являющиеся частью данного дерева, должны быть просто скопированы из схемы **artifact** в схему **longform**.

Для того, чтобы сократить дерево наращиваемых типов данных, выполняют следующие преобразования:

- каждый наращиваемый или расширяющий тип данных должен быть заменен типом данных с тем же именем, но который не является ни наращиваемым, ни расширением какого-либо типа данных;

- элементы списков, заданных в каждом конструкционном типе данных должны быть совокупностью элементов, принадлежащих области определения данного типа данных относительно схемы **longform**;

- для того, чтобы поддерживать отношения между наращиваемыми и расширяющими типами данных, их результирующие структуры должны быть связаны; зависимые типы данных, к которым относятся расширяющие типы данных, должны быть созданы в виде определенных типов данных, использующих расширяющий тип данных в качестве базисного типа; элементы базисного типа данных, не являющиеся допустимыми в определенном типе данных, должны быть ограничены посредством локальных правил;

- в случае, если определенный тип данных в схеме, видимой во внешней схеме, имеет в качестве своего базисного типа данных наращиваемый или расширяющий тип данных, то к данному определенному типу данных добавляется условие **WHERE**; это позволит исключить все элементы, появившиеся в результате преобразования, но не являющиеся допустимыми элементами списков в схемах, видимых во внешней схеме; поясняющие примеры приведены ниже;

- при выполнении данной процедуры в результирующую схему могут быть скопированы конструкционные типы данных, на которые не ссылается ни один тип данных в схеме в длинной форме. Такие типы данных и их помеченные комментарии должны быть удалены из длинной формы.

Ниже приведено подробное описание применения данных правил к типам данных **EXTENSIBLE ENUMERATION** и **EXTENSIBLE SELECT**.

G.5.3.1 Тип данных EXTENSIBLE ENUMERATION

Тип данных **EXTENSIBLE ENUMERATION** должен быть преобразован в соответствии с определенной выше процедурой в перечисляемый тип данных, не являющийся наращиваемым. Имена всех конструкций должны быть сохранены.

Если все элементы перечисления наращиваемого перечисляемого типа данных допустимы в контексте результирующей схемы в длинной форме, то наращиваемый перечисляемый тип данных должен быть преобразован в определенный тип данных, базисным типом которого является целевой перечисляемый тип данных

наращиваемого перечисления, на котором он основан. Именем определенного типа данных в целевой модели должно быть имя соответствующего перечисляемого типа данных в исходной схеме.

Если из целевого определенного типа данных необходимо исключить элементы перечисления, недопустимые в его контексте, но которые определены в его базисном перечислении, то необходимо выполнить следующие действия:

- создать промежуточный определенный тип данных и присвоить ему имя, являющееся уникальным в целевой длинной форме;
- использовать в качестве его базисного типа данных базисный тип данных наращиваемого перечисляемого типа данных;
- создать другой определенный тип данных и присвоить ему имя наращиваемого перечисляемого типа данных, подлежащего преобразованию;
- использовать промежуточный определенный тип данных как его базисный тип;
- исключить элементы перечисления, являющиеся недопустимыми в целевом контексте, но определенные в преобразуемом перечислении, из реализации в целевой схеме посредством локальных правил (см. пример 3);
- для каждого исключаемого элемента перечисления должно быть создано одно локальное правило.

Примечание – См. правило по перечислению j) в 8.4.1, касающееся использования локальных правил в объявлениях типов, в которых объявляются перечисляемые типы данных.

Примеры

1 Промежуточная схема, сформированная в соответствии с G.4, выглядит следующим образом:

```
SCHEMA artifact;
   TYPE gender = ENUMERATION OF (not-known, male, female);
   END_TYPE;
   TYPE general_approval = EXTENSIBLE ENUMERATION OF (approved, rejected);
   END_TYPE;
   – – другие объявления, зависящие от gender и general_approval
   ...
   END_SCHEMA; — конец схемы artifact
```

Результирующая схема по ИСО 10303-11:1994 выглядит следующим образом:

```
SCHEMA longform;
   TYPE gender = ENUMERATION OF (not-known, male, female);
   END_TYPE;
   TYPE general_approval = ENUMERATION OF (approved, rejected);
   END_TYPE;
   – – другие объявления, зависящие от gender и general_approval
   ...
END_SCHEMA; – – конец схемы longform
```

2 Промежуточная схема, сформированная в соответствии с G.4, выглядит следующим образом:

```
SCHEMA artifact;
   TYPE general_approval = EXTENSIBLE ENUMERATION OF (approved, rejected);
   END_TYPE;
   TYPE domain2_approval = EXTENSIBLE ENUMERATION BASED_ON general_approval
         WITH (pending);
   END_TYPE;
   – – другие объявления, зависящие от domain2_approval
   ...
END_SCHEMA; – – конец схемы artifact
```

Результирующая схема по ИСО 10303-11:1994 выглядит следующим образом:

```
SCHEMA longform;
   TYPE general_approval = ENUMERATION OF
         (approved, rejected, pending);
   END_TYPE;
   TYPE domain2_approval = general_approval;
   END_TYPE;
   – – другие объявления, зависящие от domain2_approval
   ...
END_SCHEMA; – – конец схемы longform
```

3 Промежуточная схема, сформированная в соответствии с G.4, выглядит следующим образом:

```
SCHEMA artifact;
   TYPE general_approval = EXTENSIBLE ENUMERATION OF
         (approved, rejected);
   END_TYPE;
```

```
    TYPE domain2_approval = EXTENSIBLE ENUMERATION BASED_ON general_approval
        WITH (pending);
    END_TYPE;
    TYPE specific_approval = ENUMERATION BASED_ON domain2_approval WITH
        (rework);
    END_TYPE;
    TYPE domain3_approval = EXTENSIBLE ENUMERATION BASED_ON general_approval
        WITH (cancelled);
    END_TYPE;
    -- другие объявления, зависящие от данных типов
    ...
END_SCHEMA; — конец схемы artifact
```

Результирующая схема по ИСО 10303-11:1994 выглядит следующим образом:

```
SCHEMA longform;
    TYPE general_approval = ENUMERATION OF
        (approved, rejected, pending, cancelled, rework);
    END_TYPE;
    TYPE domain2_approval = general_approval;
    WHERE
        wr1 : SELF <> cancelled;
    END_TYPE;
    TYPE specific_approval = domain2_approval;
    END_TYPE;
    TYPE domain3_approval = general_approval;
    WHERE
        wr1 : SELF <> pending;
        wr2 : SELF <> rework;
    END_TYPE;
    -- другие объявления, зависящие от данных типов
    ...
END_SCHEMA; -- конец схемы longform
```

Особое внимание следует уделить тому факту, что в перечисляемых типах данных по ИСО 10303-11:1994 учитывается порядок элементов перечисления. Данное понятие упорядочения не включено в ИСО 10303-11:2003.

G.5.3.2 Тип данных EXTENSIBLE SELECT

Тип данных **EXTENSIBLE SELECT** должен быть преобразован в соответствии с определенной выше процедурой в выбираемый тип данных, не являющийся наращиваемым. Имя исходной конструкции должно быть сохранено.

Выбираемый тип данных, основанный на наращиваемом выбираемом типе данных, должен быть преобразован в определенный тип данных, базисным типом которого является целевой выбираемый тип данных наращиваемого выбора, на котором он основан. Именем определенного типа данных в целевой модели должно быть имя соответствующего выбираемого типа данных в исходной схеме. Из определенного типа данных необходимо исключить элементы выбора, недопустимые в его контексте, но определенные в его базисном выборе, из реализации во внешней схеме посредством локальных правил. Для каждого исключаемого элемента выбора должно быть создано одно локальное правило.

Примеры

1 Промежуточная схема, сформированная в соответствии с G.4, выглядит следующим образом:

```
SCHEMA artifact;
    TYPE attachment_method = EXTENSIBLE SELECT
        (nail);
    END_TYPE;
    ENTITY nail;
    END_ENTITY;
    TYPE permanent_attachment = SELECT BASED_ON attachment_method WITH
        (glue, weld);
    END_TYPE;
    TYPE simple_attachment = SELECT BASED_ON attachment_method WITH
        (needle, tape);
     END_TYPE;
    -- объявления объектов glue и других
END_SCHEMA; -- конец схемы artifact
```

Результирующая схема по ИСО 10303-11:1994 выглядит следующим образом:

```
SCHEMA longform;
  TYPE attachment_method = SELECT
        (nail, glue, weld, needle, tape);
  END_TYPE;
  ENTITY nail;
  END_ENTITY;
  TYPE permanent_attachment = attachment_method;
  WHERE
      wr1 : NOT (' LONGFORM.NEEDLE ' IN TYPEOF(SELF));
      wr2 : NOT (' LONGFORM.TAPE ' IN TYPEOF(SELF));
  END_TYPE;
  TYPE simple_attachment = attachment_method;
  WHERE
      wr1 : NOT (' LONGFORM.GLUE ' IN TYPEOF(SELF));
      wr2 : NOT (' LONGFORM.WELD ' IN TYPEOF(SELF));
  END_TYPE;
  -- объявления объектов glue и других
END_SCHEMA; — конец схемы longform
```

2 Промежуточная схема, сформированная в соответствии с G.4 и названная **problem**, *выглядит следующим образом:*

```
SCHEMA problem;
   ENTITY e1;
      attr : t1;
  END_ENTITY;
  ENTITY e2;
      attr : t3;
  END_ENTITY;
  TYPE t1 = SELECT
       (t2, t3);
  END_TYPE;
  TYPE t2 = INTEGER;
  END_TYPE;
  TYPE t3 = EXTENSIBLE SELECT;
  END_TYPE;
END_SCHEMA;
```

Схема **problem** *является ссылочно полной спецификацией данных по ИСО 10303-11:2003. Однако ее невозможно преобразовать в спецификацию, соответствующую ИСО 10303-11:1994, поскольку тип данных* **SELECT** *не может иметь пустой список элементов выбора. Удаление* **t3** *не решает данную проблему, поскольку на него ссылается объект* **e2**.

G.5.4 Преобразование ограничений подтипов

Ключевое слово **SUBTYPE_CONSTRAINT** не определено в ИСО 10303-11:1994. Поэтому объявления **SUBTYPE_CONSTRAINT** удаляются в процессе преобразования. Однако семантика ограничения должна сохраниться в длинной форме по ИСО 10303-11:1994.

Преобразования ограничений **TOTAL_OVER** и ограничений на допустимые реализации графов подтипов/супертипов определены ниже.

G.5.4.1 Ограничение TOTAL_OVER

К ограничениям **TOTAL_OVER** применяются следующие правила преобразования:

- схема в длинной форме должна сохранять семантику ограничения **TOTAL_OVER**, даже если импортируется только одна из компонент ограничения;

- для каждого ограничения **TOTAL_OVER** из **SUBTYPE_CONSTRAINT** в схему должно быть добавлено глобальное правило **RULE**;

- именем **RULE** должно быть **total_over_<имя ограничения подтипа>**;

- правило **RULE** должно быть допустимым для объекта супертипа, для которого было задано ограничение **TOTAL_OVER**;

- правило **WHERE** в глобальном правиле **RULE** должно обеспечивать, чтобы каждый экземпляр целевого супертипа являлся типом/типами данных одного или нескольких подтипов, определенных в исходном ограничении **TOTAL_OVER** и импортированных в целевую схему в длинной форме;

- комментарии с метками, относящиеся к ограничению **SUBTYPE_CONSTRAINT**, должны быть переназначены новому правилу **RULE**; если создается несколько глобальных правил, то комментарий должен быть повторен для каждого правила.

Примечание — Рекомендуется вручную отредактировать переназначенные помеченные комментарии в конце процесса преобразования.

Пример — Промежуточная схема, сформированная в соответствии с G.4, выглядит следующим образом:

```
SCHEMA artifact;
  ENTITY e1;
  END_ENTITY;
  ENTITY e2 SUBTYPE OF (e1);
  END_ENTITY;
  SUBTYPE_CONSTRAINT sc_total_over FOR e1;
    TOTAL_OVER (e2);
  END_SUBTYPE_CONSTRAINT;
END_SCHEMA; — конец схемы artifact
```

Результирующая схема по ИСО 10303-11:1994 выглядит следующим образом:

```
SCHEMA longform;
  ENTITY e1;
  END_ENTITY;
  ENTITY e2 SUBTYPE OF (e1);
  END_ENTITY;
  RULE total_over_sc_total_over FOR (e1);
  WHERE
    (* "total_over_sc_total_over.wr1" Все экземпляры e1
      должны также быть объектного типа e2. *)
    wr1 : SIZEOF (QUERY(e1_i <* e1 |
        SIZEOF ([ ' LONGFORM.E2 ' ] * TYPEOF(e1_i)) = 0)) = 0;
  END_RULE;
END_SCHEMA; – – конец схемы longform
```

G.5.4.2 Ограничения реализаций подтипов/супертипов

Если **SUBTYPE_CONSTRAINT** включает в себя ограничения на реализацию графов подтипов/супертипов, то они должны быть преобразованы в ограничения **SUPERTYPE**. Данные ограничения обычно включают в себя ключевые слова **AND, ANDOR** и **ONEOF**. Ограничения **SUPERTYPE** должны быть построены по следующим правилам:

a) для каждого объекта, ограниченного ограничением **SUBTYPE_CONSTRAINT**, содержащим ограничение **SUPERTYPE**, ограничение **SUPERTYPE** заключается в круглые скобки и добавляется к ограничению **SUPERTYPE**, определенному в данном объекте;

b) каждое ограничение, вытекающее из **SUBTYPE_CONSTRAINT**, объединяется посредством оператора **ANDOR** со всеми другими ограничениями, добавленными из разных спецификаций **SUBTYPE_CONSTRAINT**.

Комментарии с метками, относящиеся к ограничению **SUBTYPE_CONSTRAINT**, содержащему указанные ограничения, должны быть переназначены объекту с ограничением **SUPERTYPE**.

Примечание — Рекомендуется вручную отредактировать переназначенные помеченные комментарии в конце процесса преобразования.

Пример — Промежуточная схема, сформированная в соответствии с G.4, выглядит следующим образом:

```
SCHEMA artifact;
  ENTITY p;
  END_ENTITY;
  SUBTYPE_CONSTRAINT p_subs FOR p;
    m AND a;
  END_SUBTYPE_CONSTRAINT;
  ENTITY m SUBTYPE OF (p);
  END_ENTITY;
  ENTITY a ABSTRACT SUBTYPE OF (p);
  END_ENTITY;
  ENTITY l SUBTYPE OF (a);
  END_ENTITY;
END_SCHEMA; – – конец схемы artifact
```

Результирующая схема по ИСО 10303-11:1994 выглядит следующим образом:

```
SCHEMA longform;
  ENTITY p
  SUPERTYPE OF (m AND a);
  END_ENTITY;
  ENTITY a ABSTRACT SUPERTYPE
    SUBTYPE OF (p);
  END_ENTITY;
  ENTITY l SUBTYPE OF (a);
  END_ENTITY;
  ENTITY m SUBTYPE OF (p);
  END_ENTITY;
END_SCHEMA; -- конец схемы longform
```

G.5.5 Преобразование абстрактных объектных и обобщенных типов данных

Объявления **ABSTRACT ENTITY** должны быть преобразованы в ограничения на объект **ABSTRACT SUPERTYPE** в схеме по ИСО 10303-11:1994.

Формальные параметры и локальные переменные функций и процедур, объявленные с типом данных **GENERIC_ENTITY**, должны быть преобразованы в тип данных **GENERIC**. Может потребоваться доработка алгоритма функции или процедуры, чтобы обеспечить совместимость по типам данных с типом данных **GENERIC**.

Объект может иметь атрибут, объявленный обобщенным типом данных, например, **AGGREGATE**, **GENERIC_ENTITY**, или типом данных **SELECT**, элементы списка выбора которого ограничены типом данных **GENERIC_ENTITY**. Для таких атрибутов у типа данных **ABSTRACT ENTITY**, областью определения которых является обобщенный тип данных, применяются следующие правила преобразования:

- если все подтипы повторно объявляют тот же самый тип области определения атрибута, то присваивают данный тип типу данных атрибута супертипа в схеме по ИСО 10303-11:1994;

- во всех остальных случаях в схеме по ИСО 10303-11:1994 создают тип данных **SELECT** и добавляют все именованные типы данных, являющиеся типами данных атрибутов во всех повторных объявлениях в любом подтипе или супертипе. Имя выбираемого типа данных должно представлять собой объединение имени объектного типа данных абстрактного супертипа с именем атрибута, имеющего выбираемый тип данных. Между двумя именами должен быть помещен символ подчеркивания (_). К полученному имени должен быть добавлен термин **_select** (см. **binary_relationship_end_one_select** в приведенном ниже примере). В подтипах данный новый атрибут супертипа должен быть повторно объявлен с типом данных, соответствующим типу данных элементу списка выбора, объявленному в исходной схеме.

Если типом данных повторно объявленного атрибута окажется неименованный тип данных, например, агрегированный, то создаются определенные типы данных, поддерживающие тип данных атрибута, которые используются в типе данных **SELECT**. Имена данным поддерживающим определенным типам данных присваивают по следующему правилу: к имени агрегированного типа данных добавляют термин **_of_** и имя базисного типа данных агрегированной структуры (данное правило может применяться рекурсивно).

***Пример — В данном примере промежуточная схема, сформированная в соответствии с G.4, названа* abstract_example.**

```
SCHEMA abstract_example;
  ENTITY person;
  END_ENTITY;
  ENTITY product;
  END_ENTITY;
  ENTITY organization;
  END_ENTITY;
  ENTITY nary_relationship ABSTRACT;
    end_one : AGGREGATE OF GENERIC_ENTITY;
    end_two : GENERIC_ENTITY;
  END_ENTITY;
  ENTITY product_of_organization
    SUBTYPE OF (nary_relationship);
    SELF\nary_relationship.end_one: SET OF product;
    SELF\nary_relationship.end_two: organization;
  END_ENTITY;
    ENTITY person_in_organization
    SUBTYPE OF (binary_relationship);
    SELF\nary_relationship.end_one: SET OF person;
    SELF\nary_relationship.end_two: organization;
  END_ENTITY;
```

```
END_SCHEMA; -- abstract_example
```

При формировании длинной формы, конструкция **ABSTRACT** *преобразуется в ограничение* **SUPERTYPE**. *Вводится тип данных* **SELECT**, *чтобы обеспечить разные типы данных, которые обобщенный атрибут супертипа создает в его подтипах. Поскольку типы данных, используемые в списке выбора, не обязательно являются именованными, то по мере необходимости создаются определенные типы данных.*

Результирующая схема по ИСО 10303-11:1994 выглядит следующим образом:

```
SCHEMA longform;
  ENTITY person;
  END_ENTITY;
  ENTITY product;
  END_ENTITY;
  ENTITY organization;
  END_ENTITY;
  TYPE set_of_product = SET OF product;
  END_TYPE;
  TYPE set_of_person = SET OF person;
  END_TYPE;
  TYPE nary_relationship_end_one_select = SELECT
       (set_of_person, set_of_product);
  END_TYPE;
  ENTITY nary_relationship
    ABSTRACT SUPERTYPE;
    end_one : nary_relationship_end_one_select;
    end_two : organization;
  END_ENTITY;
  ENTITY product_of_organization
    SUBTYPE OF (nary_relationship);
    SELF\nary_relationship.end_one: SET OF product;
  END_ENTITY;
  ENTITY person_in_organization
    SUBTYPE OF (nary_relationship);
    SELF\nary_relationship.end_one: SET OF person;
  END_ENTITY;
END_SCHEMA; — longform
```

G.5.6 Преобразование атрибутов, переименованных при повторном объявлении

Для каждого атрибута, переименованного в подтипе при повторном объявлении, применяют следующие правила преобразования:

- повторные объявления, изменяющие только имя атрибута, но не изменяющие его типа данных, должны быть удалены;
- в повторных объявлениях с переименованием должны быть удалены ключевое слово **RENAMED** и следующее за ним новое имя атрибута;
- в данном подтипе должны быть созданы вычисляемые атрибуты с новыми именами;
- вычисленные значения атрибутов должны иметь следующие вид:

SELF\<имя супертипа>.<старое имя атрибута>.

П р и м е р — В данном примере промежуточная схема, сформированная в соответствии с G.4, названа **renamed_example.**

```
SCHEMA renamed_example;
  ENTITY binary_relationship;
    end_one : being;
    end_two : structure;
  END_ENTITY;
  ENTITY being;
  END_ENTITY;
  ENTITY structure;
  END_ENTITY;
  ENTITY person
    SUBTYPE OF (being);
  END_ENTITY;
```

```
    ENTITY person_in_structure
      SUBTYPE OF (binary_relationship);
        SELF\binary_relationship.end_one RENAMED the_person : person;
        (*"person_in_structure.end_two" Следующий атрибут только
          переименовывается, но не конкретизируется. *)
        SELF\binary_relationship.end_two RENAMED the_structure : structure;
    END_ENTITY;
  END_SCHEMA; — конец схемы renamed_example
```

При формировании длинной формы конструкции **RENAMED** *преобразуют в атрибуты* **DERIVE**. *Объект* person_in_structure *модифицируется следующим образом:*

```
...
    ENTITY person_in_structure
      SUBTYPE OF (binary_relationship);
      SELF\binary_relationship.end_one : person;
    DERIVE
      the_person : person := SELF\binary_relationship.end_one;
      the_structure : structure := SELF\binary_relationship.end_two;
  END_ENTITY;
...
```

Примечание — Ошибки могут возникать в функциях, в которых имени атрибута, следующему за ключевым словом **RENAMED**, что-либо присвоено. В подобных случаях для исправления ошибок требуется ручная обработка схемы.

Приложение Н
(справочное)

Взаимосвязи

Н.1 Взаимосвязи через атрибуты

В языке EXPRESS объявление в объектном типе данных атрибута, областью определения которого является другой тип данных, явно устанавливает взаимосвязь между этими двумя типами данных. Данная взаимосвязь считается простой взаимосвязью, которая связывает экземпляр объявляющего объекта с одним экземпляром представляющего типа данных.

Для того, чтобы описать взаимосвязи, устанавливаемые атрибутами, имеющими агрегированные значения, определяют как основной базисный тип для типа данных неагрегированный тип данных, задаваемый следующим образом:

- основным базисным типом неагрегированного типа данных является сам данный тип данных;
- основным базисным типом агрегированного типа данных является основной базисный тип его базисного типа.

Если основным базисным типом атрибута А является Т, принимают, что А основан на Т.

Тогда объявление в объектном типе данных атрибута, областью определения которого является агрегированный тип данных, основанный на основном базисном типе, устанавливает два вида взаимосвязей:

- групповая взаимосвязь между объявляющим объектом и агрегированным типом данных, связывающая экземпляр объявляющего объекта с совокупностью экземпляров основного базисного типа;
- дистрибутивная взаимосвязь между объявляющим объектом и основным базисным типом, связывающая экземпляр объявляющего объекта с одним или несколькими экземплярами основного базисного типа по отдельности.

Примечание — Данный подход отличается от подходов, принятых в некоторых других языках моделирования. Например, в модели Сущность-связь (ER-модели) объекты и взаимосвязи моделируются разными конструкциями.

Как простые, так и дистрибутивные взаимосвязи направлены от объявляющего объекта к некоторому другому типу данных. Полезно рассмотреть мощность этих взаимосвязей (с точки зрения объявляющего объекта). Если мощность обозначить как $m : n$ (где $0 \le m \le n$), то каждый экземпляр объявляющего объекта связан не менее чем с m и не более с n экземплярами целевого типа данных. Если n имеет неопределенное (**?**) значение, то не существует верхнего ограничения на число экземпляров целевого типа данных, с которыми может быть связан экземпляр объявляющего объекта.

Полезно рассмотреть инверсную взаимосвязь, представляющую обратное направление простой или дистрибутивной взаимосвязи. Данная взаимосвязь неявно существует всегда и по умолчанию имеет мощность **0 : ?**. Она может быть явно поименована и возможно ограничена посредством объявления атрибута **INVERSE** в представляющем типе данных, если представляющий тип данных является объектным типом данных.

Пример — В данном примере существует простая взаимосвязь между объектными типами данных* first *и* second, *в котором* second *играет роль* ref. *Мощность данной взаимосвязи по отношению к* first *в данном случае равна 1 : 1 (то есть каждый экземпляр объекта* first *связан строго с одним экземпляром объекта* second*). Мощность данной взаимосвязи по отношению к* second *будет 0 : ? или неограниченной (то есть один экземпляр объекта* second *может быть либо не связан ни с одним или связан с несколькими экземплярами объекта* first*), что определяет значение мощности инверсной взаимосвязи по умолчанию.

```
ENTITY first;
   ref    : second;
   fattr  : STRING;
END_ENTITY;
ENTITY second;
   sattr : STRING;
END_ENTITY;
```

Если объектный тип данных **E** имеет взаимосвязь с типом данных **T**, установленным атрибутом **A**, то данная взаимосвязь может быть условно изображена следующим образом:

$$\mathbf{E.A} \xrightarrow[\{p : q\}]{\{m : n\}} \mathbf{T},$$

где $0 \le m \le n$ и $0 \le p \le q$. Здесь $m : n$ является мощностью прямой взаимосвязи от **E** к **T**, а $p : q$ является мощностью инверсной взаимосвязи от **T** к **E**.

Ниже в более формализованной форме описаны три вида взаимосвязей и соответствующие им мощности.

Н.1.1 Простая взаимосвязь

Простая взаимосвязь является взаимосвязью, установленной атрибутом, представление которого является другим объектным типом данных. Данная взаимосвязь устанавливается между двумя объектными типами данных.

Простая взаимосвязь всегда существует между экземпляром объявляющего объекта и не более чем одним экземпляром представляющего объекта. Используя установленное выше условное изображение, данная взаимосвязь может быть представлена следующим образом:

E. A $\xrightarrow{\{m : 1\}\ \ \{p : q\}}$ **T.**

где $0 \le m \le 1$ и $0 \le p \le q$.

Это означает, что для каждого экземпляра **E** роль **A** либо не исполняет ни один из экземпляров **T**, либо исполняет строго один экземпляр **T**. Для каждого экземпляра **T** должно существовать от ***p*** до ***q*** экземпляров **E**, в которых данный экземпляр **T** исполняет роль **A**.

Следующие варианты значений ***p*** и ***q*** представляют содержательные классы ограничений на простую взаимосвязь между **E** и **T**:

- если $q = 1$, то существует ограничение, заключающееся в том, что экземпляр **T** не может исполнять роль **A** более чем в одном экземпляре **E**;
- если $1 \le p$, то для **T** существует ограничение существования. То есть для каждого экземпляра **T** должно существовать не менее ***p*** (но не более ***q***) экземпляров **E**, использующих данный экземпляр **T** в роли **A**.

Для ограничения мощности простой взаимосвязи и ее инверсной взаимосвязи используются несколько разных конструкций на языке EXPRESS:

- вариант $T = 0$ обеспечивается объявлением атрибута **A** как необязательного **(OPTIONAL)**. Если **A** не объявлен как **OPTIONAL**, то $m = 1$;
- вариант $q = 1$ обеспечивается объявлением простого инверсного атрибута или присоединением к **E.A** правила уникальности, которое требует, чтобы для каждой из ролей **A** в совокупности объектов **E** использовались разные экземпляры, поэтому экземпляр **T** может быть использован не более чем одним **E.A**;
- другие ограничения на мощность инверсной взаимосвязи выражаются путем объявления в **T** инверсного атрибута в виде: **INVERSE I : SET [p : q] OF E FOR A.** Случай, когда $p = q = 1$, может быть сокращенно представлен в виде:

INVERSE I : E FOR A.

Ниже приведены примеры простых взаимосвязей и связанных с ними ограничений мощности.

Примеры

1 **CIRCLE.CENTRE** $\xrightarrow{\{1 : 1\}\ \ \{0 : ?\}}$ **POINT**

Каждая окружность **CIRCLE** ***имеет строго одну точку*** **POINT**, ***исполняющую роль ее центра*** **CENTRE**. ***Каждая точка*** **POINT** ***может исполнять роль центра в произвольном числе окружностей (в том числе ни в одной). Это может быть объявлено следующим образом:***

```
ENTITY point;
  ...
END_ENTITY;
ENTITY circle;
  centre : point;
  ...
END_ENTITY;
```

2 **PRODUCT_VERSION.BASE_PRODUCT** $\xrightarrow{\{1 : 1\}\ \ \{1 : ?\}}$ **PRODUCT**

Каждая версия изделия **PRODUCT_VERSION** ***имеет строго одно изделие*** **PRODUCT**, ***исполняющее роль базового изделия*** **BASE_PRODUCT**. ***Объект*** **PRODUCT** ***может играть роль*** **BASE_PRODUCT** ***в любом количестве версий*** **PRODUCT_VERSION**, ***но не менее чем в одной (зависимость существования). Это может быть объявлено следующим образом:***

```
ENTITY product_version;
  base_product : product; ...
END_ENTITY;
ENTITY product;
  ...
INVERSE
  versions : SET [1 : ?] OF product_version FOR base_product;
  ...
END_ENTITY;
```

3 **PERSON.LUNCH** $\xrightarrow{\{0:1\}\ \{0:?\}}$ **MEAL**

Каждая личность* PERSON *может иметь еду* MEAL *в роли завтрака* LUNCH. *Еда* MEAL *может играть роль* LUNCH *для любого числа личностей (предполагая, что еды достаточно много). Это может быть объявлено следующим образом:

```
ENTITY person;
   lunch : OPTIONAL meal;
   ...
END_ENTITY;
ENTITY meal;
   calories : energy_measure;
   amount : weight_measure;
   ...
END_ENTITY;
```

H.1.2 Групповая взаимосвязь

Атрибут объектного типа данных, имеющий агрегированное значение, устанавливает групповую взаимосвязь между объектным типом данных и агрегированным типом данных, используемым для представления атрибута.

Примечание — Групповая взаимосвязь не распространяется на экземпляры объектов, из которых агрегированные значения атрибута в итоге формируются. Вместо этого данные экземпляры участвуют в дистрибутивной взаимосвязи (см. H.1.3).

Групповая взаимосвязь подобна простой взаимосвязи в случае неагрегированных типов данных. Групповая взаимосвязь всегда существует между экземпляром объявляющего объекта и не более чем одним экземпляром представляющего агрегированного типа данных. Как и в случае простой взаимосвязи, это может быть представлено следующим образом:

$$\mathbf{E.A} \xrightarrow{\{m:1\}\ \{r:s\}} \mathbf{T},$$

где $0 \le m \le 1$ и $0 \le r \le s$.

Следующие варианты значений ***r*** и ***s*** представляют содержательные классы ограничений на групповую взаимосвязь между **E** и **T**:

- если $s = 1$, то существует ограничение уникальности для группового значения атрибута **A**;
- если $1 \le r$, то существует ограничение существования для **T**.

Как и в простой взаимосвязи значение ***m*** определяется объявлением атрибута **A** как **OPTIONAL** ($m = 0$). Ограничение уникальности, когда $s = 1$, может быть обеспечено как и в случае простой взаимосвязи записью правила уникальности для **A** в объявлении **E**. В противном случае ***r*** и ***s*** не могут быть ограничены.

Ниже приведены примеры групповых взаимосвязей и связанных с ними ограничений мощности.

Примеры

1 **POLY_CURVE.COEF** $\xrightarrow{\{1:1\}\ \ \{0:?\}}$ **LIST [1 : ?] OF REAL**

Каждый объект* POLY_CURVE *имеет список действительных чисел, играющий роль объекта* COEF. *Любой список* LIST [1 : ?] OF REAL *может играть роль объекта* COEF *в любом количестве объектов* POLY_CURVE *(включая ни одного). Это может быть объявлено следующим образом:

```
ENTITY poly_curve;
   coef :  LIST [1 : ?] OF REAL;
   ...
END_ENTITY;
```

2 **LOOP.EDGES** $\xrightarrow{\{0:1\}\ \ \{0:1\}}$ **LIST [1 : ?] OF EDGES**

Каждый объект* LOOP *может иметь список объектов* EDGE, *играющий роль объекта* EDGES. *Каждый список* LIST [1 : ?] OF EDGE *может играть роль объекта* EDGES *самое большее для одного экземпляра* LOOP. *Это может быть объявлено следующим образом:

```
ENTITY loop;
   edges : OPTIONAL LIST [1 : ?] OF edge;
   ...
UNIQUE
   un1 : edges;
END_ENTITY;
ENTITY edge;
      ...
END_ENTITY;
```

Н.1.3 Дистрибутивная взаимосвязь

Помимо групповой взаимосвязи, рассмотренной выше, атрибут, имеющий агрегированное значение, устанавливает дистрибутивную взаимосвязь между объектным типом данных и основным базисным типом агрегированного типа данных, используемым для представления атрибута.

Дистрибутивная взаимосвязь индивидуально связывает экземпляр объявляющего объекта с любым числом экземпляров представляющего основного базисного типа. Мощность данной взаимосвязи ограничена мощностью агрегированного типа (типов) данных, используемого для представления атрибута. Обозначив основной базисный тип типа данных атрибута как **FUND(T)**, дистрибутивная взаимосвязь может быть представлена следующим образом:

$$\mathbf{E.A} \xrightarrow{\{k:1\}\ \{p:q\}} \mathbf{FUND\ (T)},$$

где $0 \le k \le 1$ и $0 \le p \le q$.

Это означает, что для каждого экземпляра **E** атрибут **A** состоит из экземпляров **FUND (T)** числом от ***k*** до **1**. Экземпляр **FUND (T)** может появиться в роли **A** в от ***p*** до ***q*** экземплярах **E**.

Следующие варианты значений ***p*** и ***q*** представляют содержательные классы ограничений на дистрибутивную взаимосвязь между ***E*** и **FUND (T)**:

- если $q = 1$, то существует ограничение, что экземпляр **FUND (T)** не может появиться в роли **A** более чем в одном экземпляре **E**;
- если $1 \le p$, то существует ограничение существования для **FUND (T)**. То есть для каждого экземпляра **FUND (T)** должно существовать не менее ***p*** (но не более ***q***) экземпляров **E**, содержащих экземпляр **FUND (T)** в роли **A**.

Для ограничения мощности дистрибутивной взаимосвязи и ее инверсной взаимосвязи используются следующие конструкции на языке EXPRESS:

- значения **k** и **1** определяются спецификациями границ агрегированных типов данных, используемых для представления **A**. В простейшем случае типом данных атрибута будет просто **SET [k : I] OF FUND (T)** (или аналогичный тип данных **BAG** или **LIST**).

Примечание — При данном подходе к взаимосвязям не существует различия между одномерными и многомерными агрегированными значениями;

- для дистрибутивной взаимосвязи вариант $q = 1$ не может быть охвачен присоединением правила уникальности к **E. A**. Вместо этого инверсный атрибут должен быть объявлен и ограничен в **FUND (T)** как **INVERSE I : E FOR A**;
- другие ограничения на мощность инверсной взаимосвязи выражаются путем объявления инверсного атрибута в **FUND (T)**, как **INVERSE I : SET [p : q] OF E FOR A**. Случай, когда $p = q = 1$, может быть сокращенно представлен как **INVERSE I : E FOR A**.

Ниже приведены примеры дистрибутивных взаимосвязей и связанных с ними ограничений мощности.

Примеры

1 Сравните данный пример с примером 1 из Н.1.2.

$$\mathbf{POLY_CURVE.COEF} \xrightarrow{\{1:?\}\ \ \{0:?\}} \mathbf{REAL}$$

Объект* POLY_CURVE *имеет, по меньшей мере, одно действительное число, играющее роль объекта* COEF. *Конкретное действительное число может быть использовано в атрибуте* COEF *в неограниченном числе объектов* POLY_CURVE *(в том числе ни одного). Это может быть объявлено следующим образом:

```
ENTITY poly_curve;
   coef :  LIST [1 : ?] OF REAL;
   . . .
END_ENTITY;
```

2 Сравните данный пример с примером 2 из Н.1.2.

$$\mathbf{LOOP.EDGES} \xrightarrow{\{0:?\}\ \ \{2:2\}} \mathbf{EDGE}$$

Объект* LOOP *может состоять из любого числа объектов* EDGE *(в том числе ни одного). Объект* EDGE *должен использоваться точно в двух разных объектах* LOOP. *Это может быть объявлено следующим образом:

```
ENTITY loop;
edges   : OPTIONAL LIST [1 : ?] OF edge;
   . . .
UNIQUE
   un1 : edges;
END_ENTITY;
```

```
ENTITY edge;
  ...
INVERSE
  loops : SET [2:2] OF loop FOR edges;
  ...
END_ENTITY;
```

H.1.4 Инверсный атрибут

Любая взаимосвязь, установленная атрибутом, имеет неявную инверсную взаимосвязь. По умолчанию инверсное отношение фактически игнорируется, то есть на него не могут делаться ссылки, и ее мощность не ограничена. Правило уникальности для атрибута, объявляющего простую взаимосвязь, фактически ограничивает мощность инверсной взаимосвязи. Язык EXPRESS предоставляет конструкции, позволяющие присваивать имена и ограничивать инверсные взаимосвязи. Данные конструкции частично были описаны выше при рассмотрении других классов взаимосвязей, а ниже приводится их полное описание.

Идентификатор инверсной взаимосвязи задается в объявлении инверсного атрибута посредством ключевого слова **INVERSE**. Тип инверсного атрибута может ограничивать мощность данной инверсной взаимосвязи.

Ниже рассмотрены конкретные конструкции на языке EXPRESS и их влияние на мощность инверсной взаимосвязи. Предполагается, что объект **E** объявляет атрибут **A** типа данных **T**. Если **T** является агрегированным типом данных, то его основным базисным типом является **FUND (T)**. Представляющий объект **[FUND (T)** или **T]** обозначен как **R**. Предполагается, что инверсная взаимосвязь объявляется инверсным атрибутом **I** в **R**.

Если **A** является неагрегированным атрибутом, с которым связано правило уникальности, то простая взаимосвязь ограничена так, чтобы в совокупности экземпляров объекта **E** каждый атрибут **A** был уникален. Поэтому экземпляр **T** может исполнять роль **A** не более чем в одном экземпляре **E**, то есть $q = 1$. Это эквивалентно следующей конструкции: **INVERSE I : SET [0 : 1] OF E FOR A**.

Если **A** является агрегированным атрибутом, с которым связано правило уникальности, то групповая взаимосвязь ограничена так, чтобы $s = 1$. То есть экземпляр **T** (который является агрегированной структурой) может исполнять роль **A** не более чем в одном экземпляре **E**. На дистрибутивную взаимосвязь ограничений нет: экземпляр **R** может исполнять роль **A** в любом числе экземпляров **E**.

Если **I** объявлен как **INVERSE I : BAG [p:q] OF E FOR A**, то мощность инверсного направления простой или дистрибутивной взаимосвязи ограничена в соответствии со значениями p и q. То есть экземпляр **R** может исполнять роль **A** в экземплярах **E** числом от p до q. Поскольку тип данных **BAG** допускает существование элементов с одинаковыми экземплярами, конкретный экземпляр **R** может исполнять данную роль более одного раза в конкретном экземпляре **E**. Это имеет смысл только если **T** является агрегированным типом данных, допускающим существование одинаковых элементов.

Если **I** объявлен как **INVERSE I : SET [p:q] OF E FOR A**, то мощность инверсного направления простой или дистрибутивной взаимосвязи ограничена в соответствии со значениями p и q. То есть экземпляр **R** может исполнять роль **A** в экземплярах **E** числом от p до q. Поскольку тип данных **SET** не допускает существования одинаковых элементов, конкретный экземпляр **R** не может исполнять данную роль более одного раза в конкретном экземпляре **E**.

Если **I** объявлен как **INVERSE I : E FOR A**, то это равносильно его объявлению как **SET [1 : 1] OF E**. То есть каждый экземпляр **R** должен исполнять роль **A** точно в одном экземпляре **E**.

Любое объявление **I** как **BAG** или **SET** с $p \geq 1$ либо как не **BAG** и не **SET**, устанавливает ограничение существования для **R**, которое требует, чтобы любой экземпляр **R** исполнял роль **A** не менее чем в одном экземпляре **E**.

H.2 Взаимосвязи подтип/супертип

Объявление подтипа в объекте устанавливает взаимосвязь между объектом подтипа и указанными объектами супертипов.

Для заданного объекта супертипа **P**, имеющего подтип **C**, данная взаимосвязь может быть условно представлена следующим образом:

$$\mathbf{P} \xrightarrow{\{n:1\} \quad \{1:1\}} \mathbf{C},$$

где $0 \leq n \leq 1$.

Это означает, что для каждого экземпляра **P** не существует ни одного, либо существует один экземпляр **C**. Для каждого экземпляра **C** существует один экземпляр **P**.

В случае, когда **P** является абстрактным супертипом, данная взаимосвязь представляется следующим образом:

$$\mathbf{P} \xrightarrow{\{1:1\} \quad \{1:1\}} \mathbf{C}.$$

Это означает, что для каждого экземпляра **P** существует один экземпляр **C**, и для каждого экземпляра **C** существует один экземпляр **P**.

Приложение J
(справочное)

Модели на языке EXPRESS для примеров, иллюстрирующих EXPRESS-G

В данном приложении приведены преобразованные на язык EXPRESS копии нескольких примеров, использованных для иллюстрации моделирования посредством графической нотации EXPRESS-G.

Данные примеры не претендуют на то, чтобы считаться реалистичными или «хорошими». В частности, модели, использованные в данных примерах, не имеют никакой связи с какими-либо моделями из других стандартов комплекса ИСО 10303.

J.1 Пример модели единой схемы

Модель в данном примере в основном утверждает, что личность может быть мужчиной или женщиной. Каждая личность имеет некоторые определяющие характеристики, такие как имя и фамилия, дата рождения, тип волос, а также может иметь или не иметь детей (которые, конечно, также являются людьми). Мужчина может быть женат на женщине. В этом случае женщина имеет инверсную взаимосвязь с данным мужчиной.

Возраст (объект **age**) личности является вычисляемым атрибутом, который рассчитывается с помощью функции **years**, определяющей число лет между датой, вводимой в качестве параметра, и текущей датой.

Личность (объект **person**) имеет инверсный атрибут, связывающий детей со своими родителями. Нижней границей данного инверсного атрибута является **0**, чтобы обеспечить ненужность предоставления всей родословной.

Примечание — Если бы объект **parents** (родители) был необходимым явным атрибутом, а объект children (дети) — его инверсным атрибутом, то родословную надо было бы раскручивать назад во времени.

Пример — Модель единой схемы на языке EXPRESS.

```
SCHEMA example;
TYPE date    = ARRAY [1 : 3] OF INTEGER;
END_TYPE;
TYPE hair_type = ENUMERATION OF
       (blonde,
        brown,
        black,
        red,
        white,
        bald);
END_TYPE;
ENTITY person;
   first_name : STRING;
   last_name : STRING;
   nickname : OPTIONAL STRING;
   birth_date : date;
   children : SET [0 : ?] OF person;
   hair : hair_type;
DERIVE
   age : INTEGER : = years (birth_date);
INVERSE
   parents : SET [0 : 2] OF person FOR children;
END_ENTITY;
SUBTYPE_CONSTRAINT sc_person FOR person;
   ONEOF (female, male);
END_SUBTYPE_CONSTRAINT;
ENTITY female
   SUBTYPE OF (person);
INVERSE
   husband : SET [0 : 1] OF male FOR wife;  – – муж является необязательным!
END_ENTITY;
ENTITY male
   SUBTYPE OF (person);
   wife : OPTIONAL female;
END_ENTITY;
```

```
FUNCTION years (past : date): INTEGER;
   (* "years" Данная функция рассчитывает число лет
      между датой в прошлом и текущей датой*)
END_FUNCTION;
END_SCHEMA;
```

J.2 Модель взаимосвязей

В приведенном ниже примере представлена простая модель, демонстрирующая некоторые объявления и взаимосвязи на языке EXPRESS. Модель содержит объекты супертипов, объекты подтипов и объекты, не являющиеся ни тем, ни другим. Также показаны два определенных типа данных, выбираемый тип данных и несколько простых типов.

Пример — Модель, содержащая простой объект и взаимосвязь типов данных.

```
SCHEMA etr;
ENTITY super;
END_ENTITY;
ENTITY sub_1
   SUBTYPE OF (super);
   attr : from_ent;
END_ENTITY;
ENTITY sub_2
   SUBTYPE OF (super);
   pick : choice;
END_ENTITY;
ENTITY an_ent;
   int : INTEGER;
END_ENTITY;
ENTITY from_ent;
   description : OPTIONAL to_ent;
   values : ARRAY [1 : 3] OF UNIQUE REAL;
END_ENTITY;
ENTITY to_ent;
   text : strings;
END_ENTITY;
TYPE choice = SELECT
      (an_ent,
       name);
END_TYPE;
TYPE name = STRING;
END_TYPE;
TYPE strings = LIST [1 : ?] OF STRING;
END_TYPE;
END_SCHEMA;
```

J.3 Простое дерево подтипов/супертипов

Язык EXPRESS позволяет определять очень сложные деревья (и сети) подтипов/супертипов. Показанное в приведенном ниже примере дерево относительно простое.

Пример — Дерево подтипов/супертипов на языке EXPRESS.

```
SCHEMA simple_trees;
ENTITY super;
END_ENTITY;
ENTITY sub1
   SUBTYPE OF (super);
END_ENTITY;
ENTITY sub2;
   SUBTYPE OF (super);
END_ENTITY;
SUBTYPE_CONSTRAINT sc_sub2 FOR sub2;
   ABSTRACT;
   ONEOF(sub3, sub4);
END_SUBTYPE_CONSTRAINT;
```

```
ENTITY sub3
   SUBTYPE OF (sub2);
END_ENTITY;
ENTITY sub4
   SUBTYPE OF (sub2);
END_ENTITY;
ENTITY sub5
   SUBTYPE OF (super);
END_ENTITY;
END_SCHEMA;
```

J.4 Повторное объявление атрибутов

Язык EXPRESS допускает повторное объявление наследованных атрибутов, обеспечивающее совместимость новых типов данных атрибутов. В приведенном ниже примере показаны некоторые допустимые формы повторного объявления:

- типом данных повторно объявленного атрибута является подтип наследованного типа;
- типом данных повторно объявленного атрибута является совместимый простой тип данных;
- значение повторно объявленного атрибута является необходимым, хотя наследованное значение было необязательным.

Пример — Повторное объявление атрибута на языке EXPRESS.

```
ENTITY sup_a;
   attr  :  sup_b;
END_ENTITY;
ENTITY sub_a
   SUBTYPE OF (sup_a);
   SELF\sup_a.attr    : sub_b;
END_ENTITY;
ENTITY sup_b;
   num : OPTIONAL NUMBER;
END_ENTITY;
ENTITY sub_b
   SUBTYPE OF (sup_b);
   SELF\sup_b.num :  REAL;
END_ENTITY;
```

J.5 Модели, состоящие из нескольких схем

Модели на языке EXPRESS состоят, по меньшей мере, из одной схемы. В приведенном ниже примере показана модель, состоящая из двух схем.

Пример — Модель на языке EXPRESS, состоящая из двух схем.

```
SCHEMA geom;
   ENTITY lcs;
   END_ENTITY;
   ENTITY surface;
   END_ENTITY;
   ENTITY curve;
   END_ENTITY;
   ENTITY point;
   END_ENTITY;
END_SCHEMA; – – geom
SCHEMA top;
   USE FROM geom
        (curve,
        point AS node);
   REFERENCE FROM geom
        (surface);
   ENTITY  face;
      bounds : LIST [1 : ?] OF loop;
      loc : surface;
   END_ENTITY;
   ENTITY loop;
   END_ENTITY;
```

```
SUBTYPE_CONSTRAINT sc_loop FOR loop;
   ABSTRACT;
   ONEOF(eloop, vloop);
END_SUBTYPE_CONSTRAINT;
ENTITY eloop
   SUBTYPE OF (loop);
   bound : LIST [1 : ?] OF edge;
END_ENTITY;
ENTITY vloop
   SUBTYPE OF (loop);
   bound : vertex;
END_ENTITY;
ENTITY edge;
   start : vertex;
   end : vertex;
   loc : curve;
END_ENTITY;
ENTITY vertex;
   loc : node;
END_ENTITY;
END_SCHEMA; -- top
```

Более сложный набор схем показан в следующем примере. Следует иметь в виду, что в каждой из объявленных схем существуют объекты, типы данных и другие определения, которые здесь не показаны для экономии места.

Пример — Многосхемная модель на языке EXPRESS.

```
SCHEMA stuff;
END_SCHEMA;
SCHEMA whatsits;
   REFERENCE FROM stuff;
END_SCHEMA;
SCHEMA widgets;
   USE FROM whosits;
   USE FROM gadgets;
   REFERENCE FROM things;
END_SCHEMA;
SCHEMA things;
END_SCHEMA;
SCHEMA gadgets;
   USE FROM stuff;
   REFERENCE FROM things;
END_SCHEMA;
SCHEMA whosits;
   REFERENCE FROM stuff;
   REFERENCE FROM whatsits;
END_SCHEMA;
```

Приложение К
(справочное)

Возможности языка EXPRESS, не рекомендуемые к использованию

В настоящей редакции языка EXPRESS существует несколько понятий, которые могут быть определены посредством двух разных синтаксических структур. Данное двойственное представление обеспечивает допустимость существующих схем и в то же время позволяет использовать новые конструкции языка. В последующих редакциях языка EXPRESS данное дублирование будет исключено удалением синтаксических структур первой редакции языка EXPRESS из конструкций, для которых определен новый синтаксис. По этой причине не рекомендуется использование следующих синтаксических структур:

- обозначение **(ABS)** для объекта, являющегося абстрактным супертипом **(ABSTRACT SUPERTYPE)**, на EXPRESS-G диаграммах;

- синтаксис определения ограничения **SUPERTYPE** (см. 9.2.3.2 и 9.2.5), которое теперь может быть определено новым объявлением **SUBTYPE_CONSTRAINT**.

Кроме того, имеется семантика, связанная с понятиями, существующими в первой редакции языка EXPRESS, которые модифицированы или удалены из настоящей редакции языка. По этой причине не рекомендуется использование следующей семантики:

- упорядочение элементов перечисления, относящегося к ненаращиваемому, нерасширяющему типу данных **ENUMERATION**.

Приложение L
(справочное)

Пример использования новых конструкций языка EXPRESS

В данном приложении представлен пример использования новых конструкций, добавленных в язык EXPRESS настоящим стандартом. Приведенная ниже модель носит чисто иллюстративный характер и демонстрирует, как могут быть использованы новые конструкции языка.

L.1 Пример управления разработкой изделий

Данный пример демонстрирует использование следующих конструкций языка EXPRESS в схеме, которая может быть использована в качестве образца для схем, расширенных применительно к конкретным областям применения:

- наращиваемые конструкционные типы данных;
- ограничение **GENERIC_ENTITY** на тип данных **SELECT**;
- **ABSTRACT ENTITY**;
- переименование атрибутов;
- **SUBTYPE_CONSTRAINT**.

Пример — В данном примере используются наращиваемые конструкционные типы данных.

```
SCHEMA my_product_management;
USE FROM generic_product_management;
TYPE my_additional_categories = ENUMERATION BASED_ON product_category_names WITH
   ( document, drawing, electromechanical, mechanical, electrical, pump );
END_TYPE;
TYPE my_additional_values = ENUMERATION BASED_ON approval_status_values WITH
   ( approved, disapproved, pending );
END_TYPE;
TYPE my_approvable_objects = EXTENSIBLE SELECT BASED_ON approvable_objects WITH
   ( product, product_category, product_to_category_relationship );
END_TYPE;
ENTITY approval_by_person_in_organization
   SUBTYPE OF ( approval );
   SELF\approval.approved_by : person_in_organization_relationship;
END_ENTITY;
ENTITY approval_by_person
   SUBTYPE OF ( approval );
   SELF\approval.approved_by : person;
END_ENTITY;
SUBTYPE_CONSTRAINT not_both FOR approval;
   ONEOF ( approval_by_person, approval_by_person_in_organization );
END_SUBTYPE_CONSTRAINT;
END_SCHEMA;
SCHEMA generic_product_management;
TYPE product_category_names = EXTENSIBLE  ENUMERATION OF ( part, tool,
                                                           raw_material );
END_TYPE;
TYPE approval_status_values = EXTENSIBLE  ENUMERATION;
END_TYPE;
TYPE approvable_objects = EXTENSIBLE GENERIC_ENTITY SELECT;
END_TYPE;
ENTITY product;
   name : STRING;
END_ENTITY;
ENTITY product_category;
   name : product_category_names;
END_ENTITY;
ENTITY binary_entity_relationship ABSTRACT;
   end_one : GENERIC_ENTITY;
   end_two : GENERIC_ENTITY;
END_ENTITY;
```

```
ENTITY product_to_category_relationship
    SUBTYPE OF ( binary_entity_relationship );
    SELF\binary_entity_relationship.end_one RENAMED the_category : product_category;
    SELF\binary_entity_relationship.end_two RENAMED the_product : product;
END_ENTITY;
ENTITY approval ABSTRACT;
    approved_by : GENERIC_ENTITY;
    status : approval_status_values;
    approved_items : SET [1 : ?] OF approvable_objects;
END_ENTITY;
ENTITY person;
    name : STRING;
END_ENTITY;
ENTITY organization;
    name : STRING;
END_ENTITY;
ENTITY person_in_organization_relationship
    SUBTYPE OF ( binary_entity_relationship );
    role_of_person : STRING;
    SELF\binary_entity_relationship.end_one RENAMED the_person : person;
    SELF\binary_entity_relationship.end_two RENAMED the_organization : organization;
END_ENTITY;
END_SCHEMA;
```

Приложение ДА
(справочное)

Сведения о соответствии ссылочных международных стандартов ссылочным национальным стандартам Российской Федерации

Таблица ДА.1

Обозначение ссылочного международного стандарта	Степень соответствия	Обозначение и наименование соответствующего национального стандарта
ИСО 10303-1:1994	IDT	ГОСТ Р ИСО 10303-1—99 Системы автоматизации производства и их интеграция. Представление данных об изделии и обмен этими данными. Часть 1. Общие представления и основополагающие принципы
ИСО/МЭК 8824-1:2002	IDT	ГОСТ Р ИСО/МЭК 8824-1—2001 Информационная технология. Абстрактная синтаксическая нотация версии один (АСН.1). Часть 1. Спецификация основной нотации
ИСО/МЭК 10646:2003	—	*

* Соответствующий национальный стандарт отсутствует. До его утверждения рекомендуется использовать перевод на русский язык данного международного стандарта. Перевод данного международного стандарта находится в Федеральном информационном фонде технических регламентов и стандартов.

Примечание — В настоящей таблице использовано следующее условное обозначение степени соответствия стандартов:

- IDT — идентичные стандарты.

Библиография

[1] ISO TR/9007:1987 Information processing systems — Concepts and terminology for the conceptual schema and the information base

[2] Kamada, T. and Kawai, S. A General Framework for Visualizing Abstract Objects and Relations», ACM Transactions on Graphics, January 1991, vol. 10, no. 1, p. 1—39

[3] Wirth, N. What can we do about the unnecessary diversity of notation for syntactic definitions?, Communications of the ACM, November 1977, vol. 20, no. 11, p. 822

[4] Sanderson, D. The Proposed Amendment to EXPRESS — Its motivation, features and relationship to EXPRESS Edition 2

УДК 656.072:681.3:006.354 ОКС 25.040.40 П87 ОКСТУ 4002

Ключевые слова: автоматизация, системы автоматизации производства, прикладные автоматизированные системы, данные, представление данных, обмен данными, искусственные языки, языки моделирования, язык EXPRESS, язык EXPRESS-G

Редактор *В. Н. Колысов*
Технический редактор *В. Н. Прусакова*
Корректор *Е. Ю. Митрофанова*
Компьютерная верстка *Т. Ф. Кузнецовой*

Сдано в набор 19.07.2010. Подписано в печать 02.11.2010. Формат 60×84 1/8. Бумага офсетная. Гарнитура Ариал. Печать офсетная. Усл. печ. л. 23,25. Уч.-изд. л. 23,50. Тираж 104 экз. Зак. 1161

ФГУП «СТАНДАРТИНФОРМ», 123995 Москва, Гранатный пер., 4.
www.gostinfo.ru info@gostinfo.ru
Набрано и отпечатано в Калужской типографии стандартов, 248021 Калуга, ул. Московская, 256.

gost.e-ecolog.ru