ГОСУДАРСТВЕННЫЙ 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

Издание официальное

БЗ 7—2000/177

ГОССТАНДАРТ РОССИИ
Москва

Предисловие

1 РАЗРАБОТАН Всероссийским научно-исследовательским институтом стандартизации (ВНИИстандарт) совместно с НИЦ CALS-технологий «Прикладная логистика»

ВНЕСЕН Техническим комитетом по стандартизации ТК 431 «CALS-технологии»

2 ПРИНЯТ И ВВЕДЕН В ДЕЙСТВИЕ Постановлением Госстандарта России от 14 ноября 2000 г. № 293-ст

3 Настоящий стандарт представляет собой аутентичный текст международного стандарта ИСО 10303-11-94 «Системы автоматизации производства и их интеграция. Представление данных об изделии и обмен этими данными. Часть 11. Методы описания. Справочное руководство по языку EXPRESS» с учетом поправки 1—99

4 ВВЕДЕН ВПЕРВЫЕ

Содержание

Введение

Стандарты серии ГОСТ Р ИСО 10303 распространяются на машинно-ориентированное представление данных об изделии и обмен этими данными. Целью является создание механизма, позволяющего описывать данные об изделии на протяжении всего жизненного цикла изделия независимо от конкретной системы. Характер такого описания делает его пригодным не только для обмена инвариантными файлами, но также и для создания баз данных об изделиях, коллективного пользования этими базами и архивации соответствующих данных.

Стандарты серии ГОСТ Р ИСО 10303 представляют собой набор отдельно издаваемых стандартов (частей). Части данной серии стандартов относятся к одной из следующих тематических групп: методы описания, интегрированные ресурсы, прикладные протоколы, комплекты абстрактных тестов, методы реализации и аттестационное тестирование. Группы стандартов данной серии описаны в ГОСТ Р ИСО 10303-1. Настоящий стандарт входит в группу методов описания.

Настоящий стандарт описывает элементы языка EXPRESS. Каждый элемент языка представлен в собственном контексте с примерами. Сначала следуют простейшие примеры, далее с нарастающей сложностью раскрываются более сложные положения.

Обзор языка

EXPRESS — это название формального языка описания информационных требований. EXPRESS применятся для определения информационных требований других стандартов серии ГОСТ Р ИСО 10303. Цели создания языка EXPRESS:

- объем и сложность стандартов серии ГОСТ Р ИСО 10303 требуют наличия возможностей как для машинной интерпретации содержащейся в них информации, так и для интерпретации данной информации человеком. Представление элементов информации из стандартов серии ГОСТ Р ИСО 10303 в не строго формализованном виде позволит игнорировать возможности применяемых средств автоматизации для проверки несоответствий в процессе представления или создания вторичных отображений, включая отображения реализации;
- язык EXPRESS разрабатывался таким образом, чтобы обеспечить возможности структурирования различных материалов, относящихся к стандартам серии ГОСТ Р ИСО 10303. В данном языке EXPRESS-схема является основой для структурирования и взаимосвязи элементов представления данных об изделии;
- язык основан на определении объектов, представляющих формализованные описания моделируемых реальных объектов. Определение объекта дается через его свойства, которые характеризуются путем определения области их значений и ограничений, накладываемых на область значений;
- при создании языка ставилась задача избежать, насколько это возможно, влияния особенностей реализации. Тем не менее, имеется возможность создания отображений реализации (таких как статический файл обмена) как автоматически, так и непосредственно.

В языке EXPRESS объекты определяются через их атрибуты — особенности или характеристики, имеющие важное значение для понимания и использования объектов. Представление атрибутов может иметь простой тип данных (такой как целый) или являться другим объектом. Геометрическая точка, например, определяется тремя действительными числами. Имена, которые даны атрибутам, дополняют определение объекта. Так, для геометрической точки использующиеся в ее определении три действительных числа имеют имена X, Y и Z. В языке устанавливается связь между определяемым объектом и атрибутами, которые его определяют, а также связь между атрибутом и его представлением.

Примечания.

1 При разработке языка EXPRESS были использованы несколько языков, в частности, Ada, Algol, C, C++, Euler, Modula-2, Pascal, PL/I и SQL. В языке EXPRESS добавлены некоторые возможности, которые делают язык более подходящим для задач описания информационной модели.

2 В настоящем стандарте примеры текстов на языке EXPRESS не соответствуют никакому конкретному стилю. Напротив, для того, чтобы сэкономить место и показать гибкость языка, иногда используется некорректный стиль. Приведенные примеры не отражают содержание информационных моделей, описанных в других стандартах серии ГОСТ Р ИСО 10303. Примеры создавались для того, чтобы показать конкретные особенности языка EXPRESS. Необходимо игнорировать любые сходства между примерами, приведенными в настоящем стандарте, и нормативными информационными моделями, содержащимися в других стандартах серии ГОСТ Р ИСО 10303.

3 Приложения A, B, C, D, E и F являются обязательными и составляют неотъемлемую часть настоящего стандарта. Приложения G, H и J являются справочными.

4 В тексте настоящего стандарта объекты и конструкции языка EXPRESS в ряде случаев выделены полужирным шрифтом (например, **file_description**).

ГОСУДАРСТВЕННЫЙ 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 2001—07—01

1 Область применения

В настоящем стандарте определен язык, с помощью которого могут быть описаны аспекты данных об изделии. Данный язык называется EXPRESS.

В стандарте также определено графическое представление для подмножества конструкций в языке EXPRESS. Это графическое представление называется EXPRESS-G.

В ГОСТ Р ИСО 10303-1 EXPRESS описан как язык определения данных. Данный язык состоит из элементов, которые позволяют однозначно определять данные и устанавливать ограничения на эти данные.

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

Область применения настоящего стандарта не охватывает:
- определение форматов баз данных;
- определение форматов файлов;
- определение форматов передачи;
- управление процессом;
- обработку информации;
- обработку особых ситуаций.

Язык EXPRESS не является языком программирования.

2 Нормативные ссылки

В настоящем стандарте использованы ссылки на следующие стандарты:

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—95* Информационная технология. Взаимосвязь открытых систем. Абстрактная синтаксическая нотация версии один (АСН.1). Часть 1. Требования к основной нотации

ИСО/МЭК 10646-1—93* Информационная технология. Универсальный многобайтный кодированный набор символов (UCS). Часть 1: Архитектура и основной многоязычный уровень

3 Определения

3.1 Термины, определенные в ГОСТ Р ИСО 10303-1

В настоящем стандарте использованы следующие термины:

- контекст;
- данные;
- язык определения данных;
- информация.

* Оригинал стандарта ИСО/МЭК — во ВНИИКИ Госстандарта России.

Издание официальное

3.2 Термины, определенные в ИСО/МЭК 10646-1

- графический символ.

Примечание — Данное определение охватывает только те символы из ИСО/МЭК 10646-1, которые имеют определенное визуальное представление; тем самым исключаются любые пустые или заштрихованные ячейки.

3.3 Другие определения

В настоящем стандарте использованы следующие термины с соответствующими определениями:

3.3.1 **тип данных сложного объекта** (complex entity data type): Представление объекта. Тип данных сложного объекта устанавливает область значений, определяемую общими атрибутами и ограничениями допустимой комбинации типов данных объекта внутри конкретного графа подтип/супертип.

3.3.2 **экземпляр сложного объекта (типа данных)** [complex entity (data type) instance]: Обозначенный блок данных, который представляет блок информации внутри класса, определенного объектом. Он принадлежит области значений, установленной типом данных сложного объекта.

3.3.3 **константа** (constant): Обозначенный блок данных из определенной области значений. Значение этого блока данных не может быть изменено.

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

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

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

3.3.7 **экземпляр объекта (типа данных)** [entity (data type) instance]: Обозначенный блок данных, который представляет блок информации внутри класса, определенного объектом. Он принадлежит области значений, установленной типом данных объекта.

3.3.8 **экземпляр** (instance): Обозначенное значение.

3.3.9 **частный тип данных сложного объекта** (partial complex entity data type): Потенциальное представление объекта. Частный тип данных сложного объекта является группировкой типов данных объекта внутри графа подтип/супертип, которая может образовывать тип данных сложного объекта целиком или частично.

3.3.10 **частное значение сложного объекта** (partial complex entity value): Значение частного типа данных сложного объекта. Оно не имеет собственного смысла и должно комбинироваться с другими частными значениями сложного объекта и обозначением для формирования экземпляра сложного объекта.

3.3.11 **совокупность** (population): Множество экземпляров типа данных объекта.

3.3.12 **экземпляр простого объекта (типа данных)** [simple entity (data type) instance]: Обозначенный блок данных, который представляет блок информации внутри класса, определенного объектом. Входит в область значений, установленную единственным типом данных объекта.

3.3.13 **граф подтип/супертип** (subtype/supertype graph): Объявленный набор типов данных объекта. Типы данных объекта, объявленные внутри графа подтип/супертип, связаны через выражение подтипа. Граф подтип/супертип определяет один или несколько сложных типов данных объекта.

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

3.3.15 **значение** (value): Блок данных.

4 Требования соответствия

4.1 Формальные спецификации, написанные на EXPRESS

4.1.1 Лексический язык

Формальная спецификация, написанная на EXPRESS, должна быть согласована с заданным уровнем, как указано ниже. Формальная спецификация согласована с заданным уровнем, когда все проверки, установленные для этого уровня и всех нижних уровней, удовлетворены для этой спецификации.

Уровни проверки

Уровень 1: проверка ссылок. Данный уровень состоит из проверки формальной спецификации, гарантирующей ее синтаксическую и ссылочную корректность. Формальная спецификация синтаксически корректна, если она соответствует синтаксису, полученному расширением пер-

вичных синтаксических правил (синтаксиса), установленных в приложении А. Формальная спецификация корректна в отношении ссылок, если все ссылки на каждый элемент описания языка EXPRESS соответствуют области применения и правилам видимости, установленным в разделах 10 и 11.

Уровень 2: проверка типов. Данный уровень состоит из проверки формальной спецификации, гарантирующей ее соответствие следующему:

- 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;
- переобъявления атрибутов должны удовлетворять правилам, установленным в 9.2.3.4.

Уровень 3: проверка значения. Данный уровень состоит из проверки формальной спецификации, гарантирующей ее соответствие утверждениям вида «А должно быть больше В», как установлено в разделах 7—16. Проверка справедлива для тех положений, в которых А и В могут быть выражены литералами и/или константами.

Уровень 4: полная проверка. Данный уровень состоит из проверки формальной спецификации, гарантирующей ее соответствие всем формулировкам требований, установленным в настоящем стандарте.

Пример 1 — В настоящем стандарте установлено, что функция должна иметь оператор возврата («return») для каждой из возможных ветвей, по которым может пойти процесс после вызова данной функции. Это должно быть проверено.

4.1.2 Графическая форма

Формальная спецификация, написанная на EXPRESS-G, должна быть согласована с заданным уровнем, как указано ниже. Формальная спецификация согласована с заданным уровнем, когда все проверки, установленные для этого уровня и всех нижних уровней, удовлетворены для этой спецификации.

Уровни проверки

Уровень 1: проверка символов и области применения. Данный уровень состоит из проверки формальной спецификации, гарантирующей ее соответствие как требованиям к уровню объекта, так и требованиям к уровню схемы, как определено в D.5 и D.6. Это охватывает проверку того, что в формальной спецификации символы используются в соответствии с D.2—D.4. Формальная спецификация также должна быть проверена на соответствие страничных ссылок и переобъявленных атрибутов требованиям D.4.1 и D.5.5.

Уровень 2: полная проверка. Данный уровень состоит из проверки формальной спецификации для установления тех мест, которые не соответствуют требованиям к полному уровню объекта или полному уровню схемы, определенным в приложении D, и требованиям, установленным в разделах 7—16.

4.2 Реализации EXPRESS

4.2.1 Синтаксический анализатор языка EXPRESS

Реализация синтаксического анализатора языка EXPRESS должна выполнять синтаксический разбор любой формальной спецификации, написанной на EXPRESS, в соответствии с ограничениями, установленными в приложении Е и связанными с этой реализацией. Синтаксический анализатор языка EXPRESS будет считаться соответствующим конкретному уровню проверки (как установлено в 4.1.1), если он может применять все проверки, требуемые для данного уровня (и любых нижележащих уровней), к формальной спецификации, написанной на EXPRESS.

Разработчик синтаксического анализатора языка EXPRESS должен установить любые ограничения, которые реализация накладывает на число и длину идентификаторов, на диапазон обрабатываемых чисел и на максимальную точность действительных чисел. Такие ограничения должны быть документально оформлены в виде, установленном в приложении Е, с целью проведения аттестационного тестирования.

4.2.2 Средство графического редактирования

Реализация редактора EXPRESS-G должна обеспечивать возможности создания и отображения формальных спецификаций на EXPRESS-G, в соответствии с ограничениями, установленными в приложении Е и связанными с этой реализацией. Редактор EXPRESS-G будет считаться соответствующим конкретному уровню проверки, если он может создавать и отображать формальные спецификации на EXPRESS-G, которые соответствуют указанному уровню проверки (и любому нижележащему уровню).

Разработчик редактора EXPRESS-G должен установить любые ограничения, которые реализация накладывает на число и длину идентификаторов, на число доступных символов на страницу модели и на максимальное число страниц. Такие ограничения должны быть документально оформлены в виде, установленном в приложении Е, с целью проведения аттестационного тестирования.

5 Фундаментальные принципы

Предполагается, что читатели настоящего стандарта знакомы с нижеуказанными концепциями.

Схема, написанная на языке EXPRESS, описывает набор условий, которые устанавливают область ее определения. Экземпляры могут быть оценены для определения их принадлежности к данной области значений. Если экземпляры отвечают всем условиям, тогда они объявляются принадлежащими к данной области. Если экземпляры противоречат любому из условий, тогда они нарушают условия и поэтому не принадлежат данной области. В случае, когда экземпляры не имеют значений для необязательных атрибутов, а некоторые условия содержат в себе эти необязательные атрибуты, то не всегда возможно определить, отвечают ли экземпляры всем условиям. В таком случае считается, что экземпляры входят в данную область.

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 вводит в силу ссылку на исходное представление. Так как синтаксис языка использует идентификатор для имени, то исходное представление должно быть исследовано для понимания его структуры.

Определение типа данных объекта в языке EXPRESS описывает область его значений. Предполагается, что отдельные элементы области различаются некоторыми связанными с ними уникальными идентификаторами. В EXPRESS не установлены содержание или представление этих идентификаторов.

Объявление постоянного экземпляра объекта определяет идентифицируемый элемент области, описанной типом данных объекта. Такие экземпляры объектов не будут изменены или уничтожены операциями, выполняемыми в данной области.

Процедурные описания ограничений в EXPRESS могут объявлять или делать ссылки на дополнительные экземпляры объекта как на локальные переменные, которые, как предполагается, будут временными идентифицируемыми элементами области. Данные процедурные описания могут изменять эти дополнительные экземпляры объекта, но не могут изменять постоянные элементы данной области. Эти временные элементы области доступны только в пределах действия процедурного описания, в котором они объявлены, и прекращают существование за пределами этого описания.

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

- как разрешены ссылки на имена;
- какие другие схемы известны;
- как и когда проверяются ограничения;
- что должна делать реализация, если ограничение не удовлетворено;
- имеют или нет право на существование в реализации экземпляры, которые не соответствуют EXPRESS-схеме;
- как и где в реализации создаются, изменяются и удаляются экземпляры.

6 Синтаксис определения языка

В настоящем разделе определена нотация, используемая для представления синтаксиса языка EXPRESS. Полный синтаксис для языка EXPRESS приведен в приложении А. Части этих синтаксических правил воспроизведены в различных разделах настоящего стандарта для иллюстрации синтаксиса конкретного оператора. Эти части не всегда полны. Поэтому иногда необходимо руководствоваться приложением А для недостающих в данном примере правил. Части синтаксиса в тексте настоящего стандарта представлены в рамке. Каждое правило внутри синтаксической рамки слева имеет уникальный номер для использования его в перекрестных ссылках на другие синтаксические правила.

6.1 Синтаксис спецификации

Синтаксис EXPRESS определен как производная от Синтаксической Нотации Вирта (СНВ).

Примечание — См. ссылку в приложении J [3].

Соглашение по нотации 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 `=` указывает логическое заключение. Элемент слева от знака определяется комбинацией элементов справа. Любые пробелы, появляющиеся между элементами логического заключения, не входящие в пределы литерала, не имеют значения. Логическое заключение завершается точкой `.`.

- Использование идентификатора внутри коэффициента означает нетерминальный символ, который появляется слева от другого логического заключения. Идентификатор состоит из букв, цифр и символа подчеркивания. Ключевые слова языка представляются логическими заключениями, идентификаторы которых состоят только из заглавных букв.

- Литерал используется для обозначения терминального символа, который не может быть расширен далее. Литерал является последовательностью любых символов, заключенной в апострофы. Символ, в данном случае, может быть любым символом, определенным по ИСО/МЭК 10646-1 в позициях 21-7E группы 00, плоскости 00, строки 00. Чтобы апостроф был включен в литерал, он должен быть записан дважды.

Семантика скобок определена ниже:

- фигурные скобки `{}` указывают нулевое или большее количество повторений;
- квадратные скобки `[]` указывают необязательные параметры;
- круглые скобки `()` указывают, что группа логических заключений, включенных в круглые скобки, должна быть использована как единое логическое заключение;
- вертикальная линия `|` указывает, что в выражении должен быть выбран только один элемент.

Пример 2 — Синтаксис для строкового типа:

```
Синтаксис:
293 string_type = STRING [ width_spec ] .
318 width_spec = `(` width `)` [ FIXED ] .
317 width = numeric_expression .
```

Полное определение синтаксиса (приложение А) содержит определения для STRING, **numeric_expression** и FIXED.

Пример 3 — Учитывая синтаксис, данный в примере 2, возможны следующие варианты:

a) string
b) string (22)
c) string (19) fixed

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

6.2 Специальная символьная нотация

Следующая нотация используется для представления целых символьных наборов и некоторых специальных символов, которые сложно отобразить:

- \a — представляет символы в позициях 21-7E строки 00, плоскости 00, группы 00 ИСО/МЭК 10646-1;
- \n — представляет символ «новая строка (newline)» (системно зависимый) (см. 7.1.5.2);
- \q — символ кавычки (апострофа) (') и содержится внутри \a;
- \s — символ пробела;
- \x8, \x9, \xA, \xB, \xC, \xD — представляет символы в позициях 8—13 строки 00, плоскости 00, группы 00 ИСО/МЭК 10646-1.

7 Основные элементы языка

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

Основные элементы языка компонуются в текст, разделяемый обычно на физические строки. Физическая строка является любым числом (включая нулевое) символов, заканчивающимся символом «новая строка» (см. 7.1.5.2).

Примечание — Схема более удобна для чтения, когда операторы разделены на строки, а для компоновки различных конструкций используется пробел (whitespace).

Пример 4 — Следующие записи эквивалентны:

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

7.1 Набор символов

В схемах, написанных на языке EXPRESS, должны использоваться только символы из следующего набора: символы, расположенные в 08—0D, графические символы, лежащие в диапазоне 20—7E из ИСО/МЭК 10646-1, а также специальный символ \n, означающий новую строку. Данный набор называется набором символов EXPRESS. Элемент этого набора ссылается на позицию соответствующего стандарта, в которой расположен данный символ; номера этих позиций определены в шестнадцатеричной системе. Печатаемые символы данного набора (позиции 21—7E из ИСО/МЭК 10646-1) объединяются для формирования лексем языка EXPRESS. Лексемами EXPRESS являются ключевые слова, идентификаторы, знаки или литералы. Дальнейшая классификация набора символов EXPRESS приведена ниже.

Данный набор символов определен как абстрактный набор символов; он не зависит от его представления в реализации.

Примечание — В ИСО/МЭК 6429 (см. [1] из приложения J) установлены семантики символов позиций 08—0D из ИСО/МЭК 10646-1. В настоящем стандарте не требуются семантики, установленные в ИСО/МЭК 6429.

7.1.1 Цифры

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

```
Синтаксис:
120 digit = `0` | `1` | `2` | `3` | `4` | `5` | `6` | `7` | `8` | `9` .
```

7.1.2 Буквы

В EXPRESS используются строчные и прописные буквы английского алфавита (позиции 41—5A и 61—7A набора символов EXPRESS). Регистр букв имеет значение только в явных строковых литералах.

Примечание — Текст на языке EXPRESS может быть написан с использованием верхних, нижних или смешанных регистров (заглавных, строчных или и тех и других букв — см. пример 4).

```
Синтаксис:
124 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 (выводимые на печать символы, которые не являются ни буквами, ни 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 EXPRESS.

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

7.1.4 Подчеркивание

Символ подчеркивания (`_` — ячейка 5F набора символов EXPRESS) может использоваться в идентификаторах и ключевых словах, за исключением использования его в качестве первого символа.

7.1.5 Пробел

Пробел (whitespace) определен 7.1.5.1—7.1.5.3 и 7.1.6. Пробел должен использоваться для разделения лексем в EXPRESS-схемах.

Примечание — Свободное и последовательное использование пробела 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. Нотация `\s` может быть использована для представления символа пробела в синтаксисе языка.

7.1.5.2 *Новая строка*

Символ «новая строка (newline)» отмечает физическое окончание строки внутри формальной спецификации, записанной на языке EXPRESS. Обычно новая строка трактуется как пробел, но существует разница между завершением заключительного комментария и неправильным окончанием строкового литерала. В синтаксисе языка символ новой строки представляется нотацией \n.

Конкретное представление новой строки определяется спецификой реализации.

7.1.5.3 *Другие символы*

Символы из позиций 08—0D должны рассматриваться как пробел (whitespace), кроме случаев, когда они встречаются внутри строкового литерала. Для представления этих символов в синтаксисе языка должна быть использована нотация \xn, где n является одним из символов: 8, 9, A, B, C и D.

7.1.6. Комментарии

Комментарий используется при документировании описания и должен быть интерпретирован синтаксическим анализатором языка EXPRESS как пробел (whitespace). Существует два вида комментариев: вложенный и заключительный.

7.1.6.1 *Вложенный комментарий*

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

Синтаксис:
```
142 embedded_remark = `(*` { not_lparen_star | lparen_not_star | star_not_rparen |
                             embedded_remark } `*)` .
126 not_lparen_star = not_paren_star | `)` .
127 not_paren_star = letter | digit | not_paren_star_special .
124 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` .
120 digit = `0` | `1` | `2` | `3` | `4` | `5` | `6` | `7` | `8` | `9` .
129 not_paren_star_special = not_paren_star_quote_special | `` ` `` .
128 not_paren_star_quote_special = `!` | `"` | `#` | `$` | `%` | `&` | `+` | `,` |
                                   `-` | `.` | `/` | `:` | `;` | `<` | `=` | `>` |
                                   `?` | `@` | `[` | `\` | `]` | `^` | `_` | `` ` `` |
                                   `{` | `|` | `}` | `~` .
125 lparen_not_star = `(` not_star .
132 not_star = not_paren_star | `(` | `)` .
135 star_not_rparen = `*` not_rparen .
131 not_rparen = not_paren_star | `*` | `(` .
```

Любой символ из набора символов EXPRESS может быть вставлен между началом и концом вложенного комментария, включая символ «новая строка (newline)», поэтому вложенные комментарии могут охватывать несколько физических строк.

Вложенные комментарии могут вкладываться друг в друга.

Примечание — При вложении комментариев необходимо обращать большое внимание на согласование пар символов.

Пример 5 — Пример вложенных комментариев.

(* Символ `(*` начинает комментарий, а символ `*)` его заканчивает *).

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:
144 tail_remark = `- -` { \a | \s | \x8 | \x9 | \xA | \xB | \xC | \xD } \n .

Пример 6 — -- Этот текст является комментарием, который заканчивается символом «новая строка (newline)».

7.2 Зарезервированные слова

Зарезервированными словами языка EXPRESS являются ключевые слова, а также 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 показаны в таблице 1.

Примечания

1 Ключевые слова представляются литералом, состоящим из символов заглавных букв (верхнего регистра). Это облегчает чтение синтаксических конструкций.

2 Ключевые слова CONTEXT (КОНТЕКСТ), END_CONTEXT (КОНЕЦ_КОНТЕКСТА), MODEL (МОДЕЛЬ) и END_MODEL (КОНЕЦ_МОДЕЛИ) зарезервированы для использования в последующих редакциях настоящего стандарта.

Таблица 1 — Ключевые слова языка EXPRESS

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

7.2.2 Зарезервированные слова-операторы

Операторы, определенные зарезервированными словами, приведены в таблице 2. Определения этих операторов см. в разделе 12.

Таблица 2 — Зарезервированные слова EXPRESS-операторы

AND	ANDOR	DIV	IN
LIKE	MOD	NOT	OR
XOR			

7.2.3 Встроенные константы

Имена встроенных констант приведены в таблице 3. Определения этих констант см. в разделе 14.

Таблица 3 — Зарезервированные слова EXPRESS-константы

?	SELF	CONST_E	PI
FALSE	TRUE	UNKNOWN	

7.2.4 Встроенные функции

Имена встроенных функций приведены в таблице 4. Определения этих функций см. в разделе 15.

Таблица 4 — Зарезервированные слова EXPRESS-имена функций

ABS	ACOS	ASIN	ATAN
BLENGTH	COS	EXISTS	EXP
FORMAT	HIBOUND	HIINDEX	LENGTH
LOBOUND	LOG	LOG2	LOG10
LOINDEX	NVL	ODD	ROLESOF
SIN	S1ZEOF	SQRT	TAN
TYPEOF	USEDIN	VALUE	VALUE_IN
VALUE_UNIQUE			

7.2.5 Встроенные процедуры

Имена встроенных процедур приведены в таблице 5. Определения этих процедур см. в разделе 16.

Таблица 5 — Зарезервированные слова EXPRESS—имена процедур

INSERT	REMOVE

7.3 Знаки (symbols)

Знаки являются специальными символами или группами специальных символов, которые имеют специальное значение в языке EXPRESS. Знаки используются в языке EXPRESS как разделители и операторы. Разделитель используется для начала, выделения или окончания смежных лексических или синтаксических элементов. Интерпретация этих элементов была бы невозможна без разделителей. Операторы обозначают, что функции должны выполняться на операндах, которые связаны с оператором, описания этих операторов см. в разделе 12. EXPRESS-знаки приведены в таблице 6.

Таблица 6 — EXPRES-знаки

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

7.4 Идентификаторы

Идентификаторы являются именами, заданными элементам, объявленным в схеме (см. 9.3), включая саму схему. Идентификатор не должен быть таким же, как зарезервированное слово языка EXPRESS.

```
Синтаксис:
140 simple_id = letter { letter | digit | `_` } .
124 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` .
120 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 «подчеркивание».

Разработчик синтаксического анализатора языка EXPRESS, в соответствии с приложением Е, должен определить максимальное число символов идентификатора, которое может распознаваться данной реализацией.

7.5 Литералы

Литерал имеет самоопределенное постоянное значение. Тип литерала зависит от композиции символов, формирующих лексему. Литерал может иметь один из следующих типов: двоичный, целочисленный, действительный, строковый и логический.

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

7.5.1 Двоичный литерал

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

```
Синтаксис:
136 binary_literal = `%` bit { bit } .
119 bit = `0` | `1` .
```

Разработчик синтаксического анализатора языка EXPRESS, в соответствии с приложением Е, должен определить максимальное число битов в двоичном литерале, которое может распознаваться данной реализацией.

Пример 7 — Правильный двоичный литерал.

%0101001100

7.5.2 Целочисленный литерал

Целочисленный литерал представляет собой значение целого типа данных, состоящее из одной или более цифр.

```
Синтаксис:
138 integer_literal = digits .
121 digits = digit { digit } .
120 digit = `0` | `1` | `2` | `3` | `4` | `5` | `6` | `7` | `8` | `9` .
```

Примечание — Знак целочисленного литерала не используется в синтаксисе, так как EXPRESS использует концепцию унарных операторов внутри синтаксиса выражения.

Разработчик синтаксического анализатора языка EXPRESS, в соответствии с приложением Е, должен определить максимальное значение целого для целочисленного литерала, которое может распознаваться данной реализацией.

Пример 8 — Допустимые целочисленные литералы

4016

38

7.5.3 Действительный литерал

Действительный литерал представляет собой значение действительного типа данных, состоящее из мантиссы и необязательного показателя экспоненты; мантисса должна включать десятичную точку.

Синтаксис:
```
139 real_literal = digits `.` [ digits ] [ `e` [ sign ] digits ] .
121 digits = digit { digit } .
120 digit = `0` | `1` | `2` | `3` | `4` | `5` | `6` | `7` | `8` | `9` .
286 sign = `+` | `-` .
```

Примечание — Знак действительного литерала не смоделирован в пределах синтаксиса, так как EXPRESS использует концепцию унарных операторов внутри синтаксиса выражения.

Разработчик синтаксического анализатора языка EXPRESS, в соответствии с приложением Е, должен определить максимальную точность и максимальный показатель экспоненты действительного литерала, которые могут распознаваться данной реализацией.

Примеры

9 — Допустимые действительные литералы

1.E6«E» может быть записано как строчной, так и прописной буквой

3.5e-5

359.62

10 — Недопустимые действительные литералы

.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 собой значение строкового типа данных. Существуют две формы строкового литерала: простой и кодированный. Простой строковый литерал состоит из последовательности символов из набора символов EXPRESS (см. 7.1), заключенной в апострофы (`). Апостроф внутри простого строкового литерала представляется двумя последовательными апострофами. Кодированный строковый литерал состоит из четырехоктетного кодированного представления каждого символа из последовательности символов ИСО/МЭК 10646-1, заключенного 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 = ячейка ИСО/МЭК 10646-1, в которой определен символ.

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

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

Разработчик синтаксического анализатора языка EXPRESS, в соответствии с приложением Е, должен определить максимальное число символов простого строкового литерала, которое может распознаваться данной реализацией.

Синтаксис:
```
292 string_literal = simple_string_literal | encoded_string_literal .
141 simple_string_literal = \q { ( \q \q ) | not_quote | \s | \x8 | \x9 | \xA | \xB
                            | \xC | \xD } \q .
130 not_quote = not_paren_star_quote_special | letter | digit | `(` | `)` | `*` .
128 not_paren_star_quote_special = `!` | `"` | `#` | `$` | `%` | `&` | `+` | `,` |
                                   `-` | `.` | `/` | `:` | `;` | `<` | `=` | `>` |
                                   `?` | `@` | `[` | `\` | `]` | `^` | `_` | ``` |
                                   `{` | `|` | `}` | `~` .
124 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` .
120 digit = `0` | `1` | `2` | `3` | `4` | `5` | `6` | `7` | `8` | `9` .
137 encoded_string_literal = `"` encoded_character { encoded_character } `"` .
122 encoded_character = octet octet octet octet .
133 octet = hex_digit hex_digit .
123 hex_digit = digit | `a` | `b` | `c` | `d` | `e` | `f` .
```

Разработчик синтаксического анализатора языка EXPRESS, в соответствии с приложением Е, должен также определить максимальное число октетов (которое должно быть кратно четырем) кодированного строкового литерала, которое может распознаваться данной реализацией.

Примеры

11 — Допустимые простые строковые литералы
`Ребенку нужна новая пара обуви!`
Читается как: ...Ребенку нужна новая пара обуви!
`Ed``s Computer Store`
Читается как: ...Ed`s Computer Store

12 — Недопустимые простые строковые литералы
`Ed`s Computer Store`
Всегда должно быть четное число апострофов.
`Ed``s Computer
Store`
Выходит за границы физической строки

13 — Допустимые кодированные строковые литералы
«00000041»
Читается A
«000000C5»
Читается Å
«0000795E00006238»
Это японские иероглифы в нотации Кобе

14 — Недопустимые кодированные строковые литералы
«000041»
Октеты должны быть сгруппированы по четыре
«00000041 000000C5»
Между «» разрешены только шестнадцатеричные символы.

7.5.5 Логический литерал

Логический литерал представляет собой значение логических или булевых типов данных и является одной из встроенных констант TRUE, FALSE или UNKNOWN.

Примечание — UNKNOWN не совместим с булевым типом данных.

Синтаксис:
242 logical_literal = FALS | TRUE | UNKNOWN .

8 Типы данных

В настоящем разделе представлены типы данных как часть языка. Каждый атрибут, локальная переменная или формальный параметр имеет связанный с ними тип данных.

Типы данных подразделяются на простые, сборные (агрегатные), поименованные, созданные и обобщенные. Также типы данных подразделяются в соответствии с их использованием на основные, параметрические и исходные. Взаимоотношения между двумя этими классификациями описаны в 8.6.

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

8.1 Простые типы данных

Простые типы данных определяют области элементарных единиц данных в EXPRESS. То есть они не могут быть далее декомпозированы на элементы, которые распознает EXPRESS. Простыми типами данных являются: NUMBER (ЧИСЛОВОЙ), REAL (ДЕЙСТВИТЕЛЬНЫЙ), INTEGER (ЦЕЛОЧИСЛЕННЫЙ), STRING (СТРОКОВЫЙ), BOOLEAN (БУЛЕВСКИЙ), LOGICAL (ЛОГИЧЕСКИЙ) И BINARY (ДВОИЧНЫЙ).

8.1.1 Числовой тип данных

Областью значений числового (NUMBER) типа данных являются все числовые значения в языке. Числовой тип данных должен использоваться, когда не имеет значения более определенное числовое представление.

Синтаксис:
248 number_type = NUMBER .

Пример 15 — Так как мы можем не знать контекста размера (size), мы не знаем, каким типом данных его представить, например размер толпы на футболе представляет собой целое число, тогда как площадь области подачи — действительное.

size : NUMBER ;

Примечание — В последующих редакциях настоящего стандарта может быть проведена дальнейшая специализация числового типа данных, например комплексные числа.

8.1.2 Действительный тип данных

Областью значений действительного (REAL) типа данных являются все рациональные, иррациональные и действительные научные числа. Данный тип является конкретизацией числового типа данных.

Синтаксис:
265 real_type = REAL [`(` precision_spec `)`] .
255 precision_spec = numeric_expression .

Рациональные и иррациональные числа имеют бесконечное представление и точность. Научные числа представляют физические единицы, которые известны лишь с определенной точностью. Объект **precision_spec** устанавливается в границах значащих цифр.

Литерал действительного числа представляется мантиссой и необязательным показателем экспоненты. Число цифр, составляющих мантиссу, не считая нулей, стоящих впереди первой значащей цифры до десятичной точки, является числом значащих цифр. Известная точность значения является числом первых цифр, которые необходимы для конкретного приложения.

Правила и ограничения:

a) Объект **precision_spec** задает минимальное число требуемых цифр представления действительного числа. Это выражение должно иметь целое положительное значение.

b) Когда представление не определено, точность действительного числа не ограничена.

8.1.3 Целочисленный тип данных

Областью значений целочисленного (INTEGER) типа данных являются все целые числа. Данный тип является конкретизацией действительного типа данных.

Синтаксис:
227 integer_type = INTEGER .

Пример 16 — В данном примере целочисленный тип данных используется для представления атрибута, названного **nodes**. Областью значений данного атрибута являются все целые числа без дальнейших ограничений.

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

8.1.4 Логический тип данных

Областью значений логического (LOGICAL) типа данных являются три литерала: TRUE, FALSE и UNKNOWN.

Синтаксис:
243 logical_type = LOGICAL .

Для значений логического типа данных установлен следующий порядок: FALSE < < UNKNOWN < TRUE. Логический тип данных совместим с булевским типом данных, за исключением того, что булевской переменной не может быть присвоено значение UNKNOWN.

8.1.5 Булевский тип данных

Областью значений булевского (BOOLEAN) типа данных являются два литерала: TRUE и FALSE. Булевский тип данных является конкретизацией логического типа данных.

Синтаксис:
173 boolean_type = BOOLEAN .

Для значений булевского типа данных установлен тот же порядок, что и для значений логического типа данных: FALSE < TRUE.

Пример 17 — В этом примере, атрибут, названный **planar,** представлен булевским типом данных. Значением для planar, связанным с экземпляром surface, может быть TRUE или FALSE.

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

8.1.6 Строковый тип данных

Областью значений строкового типа данных являются последовательности символов. Символами, допустимыми в строковых значениях, являются символы из ИСО/МЭК 10646-1, расположенные в позициях 08—0D, и графические символы, лежащие в диапазонах 20—7E и A0—10FFFE.

```
Синтаксис:
293 string_type = STRING [ width_spec ] .
318 width_spec = `(` width `)` [ FIXED ] .
317 width = numeric_expression .
```

Строковый тип данных может иметь как фиксированную, так и переменную ширину (число символов). Если специально не установлено, что имеется фиксированная ширина (используя зарезервированное слово FIXED в определении), то строка имеет переменную ширину.

Областью значений строкового типа данных с фиксированной шириной является набор всех последовательностей символов с шириной, точно указанной в определении типа.

Областью значений строкового типа данных с переменной шириной является набор всех последовательностей символов с шириной не большей максимальной ширины, указанной в определении.

Если ширина не указана, то областью значений будет набор всех последовательностей символов без ограничений по ширине этих последовательностей.

Подстрочные и отдельные символы могут быть адресованы, используя подстрочные индексы, как описано в 12.5. Имеет значение регистр букв (верхний или нижний) в пределах строки.

Правила и ограничения

Выражение ширины должно иметь целое положительное значение.

Примеры

18 — Следующее выражение определяет строку переменной длины, значения которой не имеют определенной максимальной длины.

```
string1 : STRING;
```

19 — Следующее выражение определяет строку, имеющую длину максимум десять символов, значения которой могут измениться по фактической длине от нуля до десяти символов.

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

20 — Следующее выражение определяет строку, имеющую длину точно десять символов, значения которой должны содержать десять символов.

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

8.1.7 Двоичный тип данных

Областью значений двоичного (BINARY) типа данных являются последовательности битов, а каждый бит представляется 0 или 1.

```
Синтаксис:
172 binary_type = BINARY [ width_spec ] .
318 width_spec = `(` width `)` [ FIXED ] .
317 width = numeric_expression .
```

Двоичный тип данных может иметь как фиксированную, так и переменную ширину (число битов). Если специально не определено, что ширина фиксирована (используя зарезервированное слово FIXED в определении), то двоичный тип данных имеет переменную ширину.

Областью значений двоичного типа данных с фиксированной шириной является набор всех последовательностей битов с шириной, точно установленной в определении типа.

Областью значений двоичного типа данных с переменной шириной является набор всех последовательностей битов с шириной, меньшей или равной максимальной ширине, указанной в определении типа. Если ширина не определена, то областью значений является набор всех последовательностей битов, без ограничения по ширине этих последовательностей.

Подстроки и отдельные биты могут быть адресованы, используя подстрочные индексы, как описано в 12.3.

Правила и ограничения

Выражение ширины должно иметь целое положительное значение.

Пример 21 — Следующее выражение может быть использовано для представления информации о шрифте символа.

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

8.2 Агрегатные (сборные) типы данных

Областями значений сборных (агрегатных) типов данных являются множества значений заданного основного типа данных (см. 8.6.1). Эти значения основного типа данных называются элементами сборного (агрегатного) множества. В EXPRESS установлено определение четырех видов сборных типов данных: ARRAY (МАССИВ), LIST (СПИСОК), BAG (МУЛЬТИМНОЖЕСТВО) и SET (НАБОР). Каждый вид сборного типа данных приписывает различные свойства для своих значений.

Тип ARRAY является упорядоченным множеством фиксированного размера. Оно проиндексировано последовательностью целых чисел.

Пример 22 — Матрица преобразований (в геометрии) может быть определена как массив массивов (чисел).

Тип LIST является последовательностью элементов, к которым можно обращаться согласно их позиции. Число элементов в списке может изменяться и быть ограничено исходя из определения типа данных.

Пример 23 — В виде списка могут быть представлены операции плана процесса. Операции упорядочены и могут быть добавлены или удалены из плана процесса.

Тип BAG является неупорядоченным множеством, в котором разрешены повторяющиеся элементы. Число элементов в мультимножестве может изменяться и быть ограничено исходя из определения типа данных.

Пример 24 — Множество крепежных деталей, используемых при сборке, может быть представлено как мультимножество. В состав его элементов может входить некоторое число одинаковых болтов, но неважно, который из них используется в конкретном отверстии.

Тип SET является неупорядоченным множеством элементов, в котором нет двух элементов, являющихся эквивалентными экземплярами. Число элементов в наборе может изменяться и быть ограничено исходя из определения типа данных.

Пример 25 — В качестве набора может быть представлено население всего мира.

Примечание — Сборки (агрегаты) в языке EXPRESS являются одномерными. Объекты, обычно рассматриваемые как многомерные (например, математические матрицы), могут быть представлены сборным типом данных, чей основной тип является в свою очередь сборным типом данных. Таким образом, сборные типы данных могут быть вложенными до произвольной глубины, позволяя представлять любые размерности.

Пример 26 — Можно представить LIST [1:3] OF ARRAY [5:10] OF INTEGER, который в действительности имеет размерность, равную двум.

8.2.1 Тип данных ARRAY

Областью значений типа данных ARRAY является индексирование фиксированного размера множества подобных элементов. Нижние и верхние границы, значения которых выражаются целыми числами, определяют диапазон значений индекса и, таким образом, размер каждого множества массива. Определение типа данных ARRAY может факультативно устанавливать, что значения массива не могут содержать повторяющихся элементов. Также может быть установлено, что значения массива не обязательно содержат элементы в каждой проиндексированной позиции.

Синтаксис:
165 array_type = ARRAY bound_spec OF [OPTIONAL] [UNIQUE] base_type .
176 bound_spec = `[` bound_1 `:` bound_2 `]` .
174 bound_1 = numeric_expression .
175 bound_2 = numeric_expression .
171 base_type = aggregation_types | simple_types | named_types .

Допустим, что *m* является нижней границей, а *n* — верхней, тогда в массиве имеется точно $n - m + 1$ элементов. Эти элементы проиндексированы от *m* до *n* включительно (см. 12.6.1).

Примечание 1 — Границы могут быть положительными, отрицательными или нулевыми, но не могут быть неопределенными (?) (см. 14.2).

Правила и ограничения

a) Оба выражения в спецификации границы, **bound_1** и **bound_2**, должны выражаться целочисленными значениями. Ни одно из них не должно иметь неопределенного (?) значения.

b) Выражение **bound_1** задает нижнюю границу массива. Оно должно иметь наименьший индекс, при котором имеется значение массива для этого типа данных.

c) Выражение **bound_2** задает верхнюю границу массива. Оно должно иметь наибольший индекс, при котором имеется значение массива для этого типа данных.

d) Выражение **bound_1** должно быть меньшим или равным **bound_2**.

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

f) Если ключевое слово OPTIONAL не указано, значение массива этого типа данных не должно содержать неопределенных (?) значений ни в одной проиндексированной позиции.

g) Если указано ключевое слово UNIQUE, то каждый элемент значения массива этого типа данных должен отличаться (то есть не быть эквивалентным экземпляром) от каждого другого элемента в том же самом значении массива.

Примечание 2 — Оба ключевых слова OPTIONAL и UNIQUE могут использоваться в одном и том же определении типа данных ARRAY. Это не исключает многократного появления неопределенных (?) значений в отдельных элементах массива. Это происходит потому, что сравнение двух неопределенных (?) значений дает результат UNKNOWN, поэтому условие уникальности не нарушается.

Пример 27 — Данный пример показывает, как объявляется многомерный ARRAY.

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

Первый массив содержит 10 элементов типа **ARRAY [11:14] OF UNIQUE something**.

Всего 40 элементов типа данных **something** в атрибуте, названном **sectors**. Внутри каждого ARRAY [11:14] не могут появляться повторяющиеся элементы, однако тот же экземпляр **something** может появляться в двух различных значениях **ARRAY [11:14]** внутри единственного значения для атрибута названного **sectors**.

8.2.2 Тип данных LIST

Областью значений типа данных LIST являются последовательности подобных элементов. Необязательные нижние и верхние границы, значения которых выражаются целыми числами, определяют минимальное и максимальное число элементов, которые могут содержаться во множестве, определенном типом данных LIST. В определении типа данных LIST может быть факультативно установлено, что значение списка не может содержать повторяющихся элементов.

Синтаксис:
237 list_type = LIST [bound_spec] OF [UNIQUE] base_type .
176 bound_spec = `[` bound_1 `:` bound_2 `]` .
174 bound_1 = numeric_expression .
175 bound_2 = numeric_expression .
171 base_type = aggregation_types | simple_types | named_types.

Правила и ограничения

a) Выражению **bound_1** должно быть присвоено целочисленное значение, большее или равное нулю. Оно задает нижнюю границу, определяющую минимальное число элементов, которое

может входить в значение списка этого типа данных. Выражение **bound_1** не должно иметь неопределенного (?) значения.

b) Выражению **bound_2** должно быть присвоено целочисленное значение, большее или равное **bound_1**, или неопределенное (?) значение. Оно задает верхнюю границу, определяющую максимальное число элементов, которое может входить в значение списка этого типа данных.

Если данное значение является неопределенным (?), то число элементов в значении списка этого типа данных не ограничено сверху.

c) Если выражение **bound_spec** опущено, то пределы — [0:?].

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

Пример 28 — Данный пример определяет список массивов. Список может содержать от нуля до десяти массивов. Каждый массив из десяти целых чисел должен отличаться от всех других массивов в конкретном списке.

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

8.2.3 Тип данных BAG

Областью значений типа данных BAG являются неупорядоченные множества подобных элементов. Необязательные нижние и верхние границы, которые выражаются целочисленными значениями, определяют минимальное и максимальное число элементов, которые могут содержаться в множестве, определяемом типом данных BAG.

Синтаксис:
170 bag_type = BAG [bound_spec] OF base_type .
176 bound_spec = `[` bound_1 `:` bound_2 `]` .
174 bound_1 = numeric_expression .
175 bound_2 = numeric_expression .
171 base_type = aggregation_types | simple_types | named_types .

Правила и ограничения

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

b) Выражению **bound_2** должно быть присвоено целочисленное значение, большее или равное **bound_1**, или неопределенное (?) значение. Оно задает верхнюю границу, определяющую максимальное число элементов, которое может входить в значение мультимножества этого типа данных.

Если данное значение является неопределенным (?), то число элементов в значении мультимножества этого типа данных не ограничено сверху.

c) Если выражение **bound_spec** пропущено, то пределы — [0:?].

Пример 29 — Данный пример определяет атрибут как мультимножество точки (где точка — поименованный тип данных, описанный в другом месте).

a_bag_of_points : BAG OF point;

Значение атрибута, названного **a_bag_of_points**, может содержать ноль или более точек. Тот же экземпляр точки может появиться более одного раза в значении **a_bag_of_points**.

Если требуется, чтобы значение содержало по крайней мере один элемент, то в спецификации может быть проставлена нижняя граница:

a_bag_of_points : BAG [1:?] OF point;

Значение атрибута, названного **a_bag_of_points**, теперь должно содержать, по крайней мере, одну точку.

8.2.4 Тип данных SET

Областью значений типа данных SET являются неупорядоченные множества подобных элементов. Тип данных SET является конкретизацией типа данных BAG. Необязательные нижние и верхние границы, выраженные целочисленными значениями, определяют минимальное и максимальное число элементов, которые могут содержаться во множестве, определенном типом данных SET. Множество, определенное типом данных SET, не должно содержать повторяющихся экземпляров.

Синтаксис:
285 set_type = SET [bound_spec] OF base_type .
176 bound_spec = `[` bound_1 `:` bound_2 `]` .
174 bound_1 = numeric_expression .
175 bound_2 = numeric_expression .
171 base_type = aggregation_types | simple_types | named_types .

Правила и ограничения

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

b) Выражению **bound_2** должно быть присвоено целочисленное значение, большее или равное **bound_1,** или неопределенное (?) значение. Оно задает верхнюю границу, определяющую максимальное число элементов, которое может входить в значение набора этого типа данных.

Если данное значение является неопределенным (?), то число элементов в значении набора этого типа данных не ограничено сверху.

c) Если выражение **bound_spec** пропущено, то пределы — [0:?].

d) Каждый элемент при появлении в типе данных SET должен отличаться (то есть, не являться эквивалентным экземпляром) от любого другого элемента в том же самом значении набора.

Пример 30 — Этот пример определяет атрибут как набор точек (где точка — поименованный тип данных, описанный в другом месте).

a_set_of_points : SET OF point;

Атрибут, названный **a_set_of_points**, может содержать ноль или более точек. Требуется, чтобы каждый экземпляр точки (в значении набора) отличался от каждой другой точки в наборе.

Если требуется, чтобы значение содержало не более 15 точек, в спецификации должна быть установлена верхняя граница:

a_set_of_points : SET [0:15] OF point;

Значение атрибута **a_ set_of_points** теперь должно содержать не более 15 точек.

8.2.5 Уникальность значения в совокупностях (агрегатах)

Уникальность между элементами совокупности (агрегата) основана на сравнении экземпляра (см. 12.2.2). Совокупность (агрегат) может быть ограничена уникальными значениями своих элементов при помощи использования функции VALUE_UNIQUE (см. 15.29).

Пример 31 — Набор, ограниченный значением уникальности.

```
TYPE value_unique_set = SET OF a;
WHERE
   wr1 : value_unique(SELF);
END_TYPE;
```

Примечание — Определяемая модельщиком уникальность значения может быть установлена через пару функций, называемых, например, **my_equal** и **my_unique**, как показано в следующем псевдокоде.

```
FUNCTION my_equal (vl,v2: GENIRIC: gen): LOGICAL;
   (* Returns TRUE if v1 `equals` v2 *)
END_FUNCTION;

FUNCTION my_unique(c: AGGREGATE OF GENIRIC): LOGICAL;
   (* Returns FALSE if two elements of c have the same `value` Else returns
   UNKNOWN if any element comparison is UNKNOWN Otherwise returns 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);
      resul t:= 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 ряд свойств, позволяющих формировать классы эквивалентности. В следующем примере *S* — набор рассматриваемых объектов, а **my_equal (i, j)**, где i и j принадлежат *S*, возвращает одно из значений [FALSE, UNKNOWN, TRUE].

a) **my_equal(i, i)** имеет значение TRUE для всех i в *S* (так как неопределенность (?) не принадлежит *S*, то не требуется, чтобы **my_equal (?,?)** имело значений TRUE);

b) **my_equal(i, j)** = **my_equal(j, i)** для всех i, j в *S*;

c) (**my_equal(i, j)** = TRUE) AND (**my_equal(j, k)** = TRUE), включая (**my_equal(i, k)** = TRUE) для всех i, j, k в *S*.

8.3 Поименованные типы данных

Поименованными типами данных являются типы, которые могут быть объявлены в формальной спецификации. Существуют два вида поименованных типов данных: объекта и определенный. В настоящем разделе описаны ссылки на поименованные типы данных; объявление этих типов данных описано в разделе 9.

8.3.1 Тип данных объекта

Типы данных объекта устанавливаются в соответствии с объявлениями ENTITY (см. 9.2). Тип данных объекта устанавливается пользователем путем присвоения идентификатора объекту. На тип данных объекта ссылаются по его идентификатору.

Синтаксис:
147 entity_ref = entity_id .

Правила и ограничения

Выражение **entity_ref** должно быть ссылкой на объект, видимый в активной области (см. раздел 10).

Пример 32 — В данном примере тип данных объекта **point** использован для представления его атрибута.

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

ENTITY line;
      p0, pi : point;
END_ENTITY;
```

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

8.3.2 Определенный тип данных

Определенные типы данных назначаются объявлениями TYPE (см. 9.1). Определенный тип данных устанавливается пользователем путем присвоения идентификатора типу. На определенный тип данных ссылаются по его идентификатору.

Синтаксис:
154 type_ref = type_id .

Правила и ограничения

Выражение **type_ref** должно быть именем определенного типа данных, видимого в активной области (см. раздел 10).

Пример 33 — Следующая спецификация является определенным типом данных, используемым для указания единиц измерения, связанных с атрибутом.

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

Атрибут, названный **bulk**, представляется действительным числом, но использование определенного типа данных **volume** помогает разъяснить смысл и контекст действительного числа; например, оно определяет **volume** в отличие от другого объекта, представляемого типом REAL.

8.4 Созданные типы данных

Существуют два вида созданных типов данных в EXPRESS: ENUMERATION (перечисляемые) типы данных перечня и SELECT (выбираемые) типы данных выбора. Эти типы данных имеют подобные синтаксические структуры и используются для обеспечения исходных представлений определенных типов данных (см. 9.1).

Примечание — Созданные типы данных в EXPRESS могут быть использованы только как представления для определенных типов данных.

8.4.1 Перечисляемый тип данных

Областью значений типа данных ENUMERATION (перечисляемого) является упорядоченный набор имен. Имена представляют собой значения перечисляемого типа данных. Эти имена обозначаются как **enumeration_id**, и на них ссылаются как на перечисляемые элементы.

Синтаксис:

```
201 enumeration_type = ENUMERATION OF `(` enumeration_id { `,` enumeration_id } `)` .
```

Два различных типа данных ENUMERATION могут содержать тот же самый **enumeration_id**. В этом случае любая ссылка на **enumeration_id** (например, в выражении) должна быть переопределена в соответствии с идентификатором типа данных для обеспечения однозначности ссылки. Ссылка затем появляется как **type_id.enumeration_id**.

Примечание — Значение **type_id** всегда доступно, потому что EXPRESS допускает тип данных ENUMERATION только как исходное представление определенного типа данных.

Правила и ограничения

a) В целях сравнения, упорядочение значений перечисляемого типа должно быть определено их относительной позицией в списке **enumeration_id**: первый встречающийся элемент должен быть меньше, чем второй; второй — меньше, чем третий, и т.д.

b) Сравнение между значениями различных типов данных ENUMERATION не определено.

c) Перечисляемый тип должен быть использован только как исходный тип определенного типа данных.

Пример 34 — В данном примере типы данных ENUMERATION использованы для того, чтобы показать, как движутся 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, они задаются каждым типом, являющимся компонентом. Между двумя этими определениями идентификатора **left** не имеется никакой взаимосвязи. Ссылка на **left** или **right** сама по себе является неоднозначной. Чтобы разрешить эту неоднозначность, ссылка на любое из этих значений должна быть квалифицирована именем типа, например, **car_can_move.left**.

8.4.2 Выбираемый тип данных

Областью значений типа данных SELECT является объединение областей значений поименованных типов данных в соответствующем списке выбора. Тип данных SELECT является обобщением каждого из поименованных типов данных в списке выбора.

Синтаксис:

284 select_type = SELECT `(` named_types { `,` named_types } `)` .

Правила и ограничения:

Каждый элемент в списке выбора должен быть типом данных объекта или определенным типом данных.

Тип данных SELECT должен использоваться только как исходный тип определенного типа данных.

Примечание — Значение типа данных SELECT может быть значением более чем одного из поименованных типов данных, определенных в списке выбора для данного выбираемого типа данных. Эта ситуация возникает только тогда, когда соответствующие поименованные типы данных являются частью общего графа наследования (9.2.3).

Пример 35 — Если a и b являются подтипами c и они связаны выражением ANDOR, и существует тип, определенный SELECT (a, b); то можно получить значение типа данных SELECT, которое может быть равным a и b одновременно.

Пример 36 — В данном контексте должен быть сделан выбор среди нескольких типов предметов.

```
TYPE
  attachment_method = SELECT(permanent_attachment, temporary_attachment);
END_TYPE;

TYPE permanent_attachment = SELECT(glue, weld);
END_TYPE;

TYPE temporary_attachment = SELECT(nail, screw);
END_TYPE;

ENTITY nail;
  body_length : REAL;
  head_area   : REAL;
END_ENTITY;

ENTITY screw;
  body_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;
```

Объект **wall_mounting** включает изделие в границы, используя постоянный или временный метод присоединения, оба из которых подразделяются далее. Значение **wall_mounting** будет иметь атрибут **using**, который является значением одного **nail** (гвоздя), **screw** (винта), **glue** (клея) или **weld** (сварного шва).

8.5 Обобщенные типы данных

Синтаксис:
211 generalized_types = aggregate_type | general_aggregation_types | generic_type .

Обобщенные типы данных используются для установления обобщения некоторых других типов данных и могут быть использованы в конкретных и очень специфических контекстах. Тип CENERIC является обобщением всех типов данных. Агрегатный тип данных AGGREGATE является обобщением всех сборных (агрегатных) типов данных. Общие сборные (агрегатные) типы данных являются обобщениями сборных (агрегатных) типов, которые ослабляют некоторые ограничения, обычно налагаемые на сборные (агрегатные) типы данных. Все эти типы данных определены в 9.5.3.

8.6 Классификация использования типов данных

В языке EXPRESS типы данных используются тремя различными способами: основные типы данных используются в качестве представления атрибутов и сборных (агрегатных) элементов; параметрические типы — как представления формальных параметров для функций и процедур; а исходные типы — как представления определенных типов данных. Некоторые классы типов данных могут быть использованы любым из этих способов, в то время как другие могут быть использованы только в определенном контексте. Эти различия показаны в таблице 7.

Таблица 7 — Использование типов данных

Тип	a	b	c
Простые (simple) типы данных	●	●	●
Сборные (aggregation) типы данных	●	●	●
Поименованные (named) типы данных	●	●	●
Созданные (constructed) типы данных			●
Обобщенные (generalized) типы данных		●	

a) Основные типы данных — представление атрибута или элементов совокупности.

b) Параметрические типы данных — представление формального параметра, локальной переменной или функционального результата

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

8.6.1 Основные типы данных

Основные типы данных используются в качестве представления атрибутов или как основные типы сборных (агрегатных) типов данных.

Основными типами данных являются простые, сборные (агрегатные) и поименованные типы данных.

Синтаксис:
171 base_type = aggregation_types | simple_types | named_types .

8.6.2 Параметрические типы данных

Параметрические типы данных используются в качестве представления формальных параметров для алгоритмов (функций и процедур). Параметрические типы данных также могут также использованы для представления возвратных типов функций и локальных переменных, объявленных в алгоритмах.

Параметрическими типами данных являются простые, поименованные и обобщенные типы данных.

Синтаксис:
253 parameter_type = generalized_types | named_types | simple_types .

8.6.3 Исходные типы данных

Исходные (underlying) типы данных используются в качестве представления определенных типов данных.

Исходными типами данных являются простые, сборные (агрегатные), созданные и определенные типы данных.

Синтаксис:
309 underlying_type = constructed_types | aggregation_types | simple_types |
type_ref .

9 Объявления

В настоящем разделе определены различные объявления, доступные в языке EXPRESS. EXPRESS-объявление создает новый EXPRESS-элемент и связывает с ним соответствующий идентификатор. На EXPRESS-элемент можно ссылаться в другом месте путем описания связанного с ним имени (см. раздел 10).

Принципиальные возможности языка EXPRESS установлены в следующих объявлениях:

- типа;
- объекта;
- схемы;
- константы;
- функции;
- процедуры;
- правила.

Объявления могут быть или явными или неявными. В настоящем разделе полностью описаны явные объявления. Неявные объявления описаны в настоящем и последующих разделах, включая их элементы и условия, при которых они устанавливаются.

9.1 Объявление типа

Объявление типа создает определенный тип данных (см. 8.3.2) и объявляет идентификатор для ссылки на этот тип. Конкретно **type_id** объявляется как имя определенного типа данных. Представлением этого типа данных является **underlying_type.** Область значения определенного типа данных совпадает с областью значений **underlying_type**, далее ограниченного **where_clause** (если оно есть). Определенный тип данных является конкретизацией исходного типа и, следовательно, совместим с исходным типом.

Примечание 1 — Составные определенные типы данных могут быть связаны с тем же самым представлением. Имена могут помочь читателю в понимании назначения (или контекста) применения **underlying_type**.

Синтаксис:

```
304 type_decl = TYPE type_id `=` underlying_type `;` [ where_clause ] END_TYPE `;` .
309 underlying_type = constructed_types | aggregation_types | simple_types |
                      type_ref .
```

Пример 37 — Следующее объявление указывает определенный тип данных, названный **person_name**, с исходным представлением типа STRING. Определенный тип **person_name** далее доступен для использования в качестве представления атрибутов, локальных переменных и формальных параметров. Это придает ему большее количество значений, чем простое использование типа STRING.

```
TYPE person_name = STRING;
END_TYPE;
```

Правила области значений (оператор WHERE)

Правила области значений устанавливают ограничения, которые лимитируют область значений определенного типа данных. Область значений определенного типа данных является областью значений их исходного представления, ограниченного правилом(ами) области значений. Правила области значений следуют за ключевым словом WHERE.

Синтаксис:

```
315 where_clause = WHERE domain_rule `;` { domain_rule `;` } .
```

Каждому **domain_rule** может быть задана метка правила. Ссылки на метки правил не описаны в настоящем стандарте.

Примечание 2 — В случае их задания, метки правил могут быть использованы в реализациях, например для документации, сообщений об ошибках и обязательных спецификаций. Для этого желательно использование разметки правил.

Правила и ограничения

а) Каждому правилу области значений должно присваиваться логическое (TRUE, FALSE или UNKNOWN) или неопределенное (?) значение.

b) Ключевое слово SELF (см. 14.5) должно появляться по меньшей мере один раз в каждом правиле области значений. Правило области значений должно выражаться конкретным значением в области значений исходного типа путем замены обнаруженной в правиле лексемы SELF соответствующим значением.

c) Правило области значений будет выполнено, если выражению будет присвоено значение TRUE, и будет нарушено, если выражению будет присвоено значение FALSE, и не будет ни выполненным ни нарушенным, если выражению будет присвоено неопределенное (?) значение или значение UNKNOWN.

d) Область значений определенного типа данных состоит из всех значений области значений исходного типа, которые не нарушают ни одно из правил области.

e) Метки правила области значений должны быть уникальны внутри заданного объявления TYPE.

Пример 38 — Может быть создан определенный тип данных, который ограничивает исходный целочисленный тип только положительными целыми числами.

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

Любой атрибут, локальная переменная или формальный параметр, которые объявлены как принадлежащие к положительному типу, могут иметь только положительные целочисленные значения.

9.2 Объявление объекта

Объявление ENTITY создает тип данных объекта и объявляет идентификатор для ссылки на него.

Каждый атрибут представляет свойство объекта и может быть связан со значением каждого экземпляра объекта. Тип данных атрибута устанавливает область его возможных значений.

Каждое ограничение представляет одно из следующих свойств объекта:

- ограничения на число, вид и организацию значений атрибутов. Они определяются в объявлениях атрибута;
- требуемые отношения между значениями атрибута или ограничениями на допустимые значения атрибутов для данного экземпляра. Они представлены в настоящем разделе и на них ссылаются как на правила области значений;
- требуемые отношения между значениями атрибутов для всех экземпляров типа данных объекта. Они появляются в:
  - отдельном разделе, где на них ссылаются как на уникальные ограничения,
  - другом разделе, где на них ссылаются как на основные ограничения,
  - глобальных правилах (см. 9.6);
- требуемые отношения между экземплярами нескольких типов объекта. Они не появляются в объявлении объекта непосредственно, а проявляются как глобальные правила (см. 9.6).

Синтаксис:

```
196 entity_decl = entity_head entity_body END_ENTITY `;` .
197 entity_head = ENTITY entity_id [ subsuper ] `;` .
194 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 внутри объявления.

b) Подтип не должен объявлять атрибут, имеющий тот же самый идентификатор в качестве атрибута одного из супертипов, за исключением того, когда подтип переобъявляет атрибут, унаследованный из одного из его супертипов (см. 9.2.3.4).

9.2.1 Атрибуты

Атрибуты типа данных объекта представляют важные черты, свойства или характеристики объекта. Объявление атрибута устанавливает отношение между типом данных объекта и типом данных, вызванным атрибутом.

Имя атрибута представляет роль, играемую связанным с ним значением, в контексте объекта, в котором оно появляется.

Существуют три вида атрибутов:

- **явный** — атрибут, чье значение должно быть обеспечено реализацией для создания экземпляра объекта;
- **вычисляемый** — атрибут с вычисляемым значением;
- **инверсный** — атрибут, чье значение состоит из экземпляров объекта, которые используют объект в конкретной роли.

Каждый атрибут устанавливает отношения между экземпляром объявленного типа данных объекта и некоторым другим экземпляром или экземплярами. Атрибут, представляемый несборным (неагрегатным) типом данных устанавливает простое отношение с этим типом данных. Атрибут, представляемый сборным (агрегатным) типом данных, устанавливает как коллективные отношения для объединения значений, так и раздельные отношения с элементами этих сборных значений. Кроме того, каждый атрибут устанавливает неявное инверсионное отношение между основным и объявленным типом данных объекта.

Примечание — Дальнейшее обсуждение этих отношений см. в приложении G.

9.2.1.1 *Явный атрибут*

Явный атрибут представляет свойство, значение которого должно быть обеспечено реализацией для создания экземпляра. Каждый явный атрибут определяет отдельное свойство. Объявление явного атрибута создает один или несколько явных атрибутов, имеющих указанную область значений, присвоенную каждому из них его идентификатором.

Синтаксис:

```
203 explicit_attr = attribute_decl { `,` attribute_decl } `:` [ OPTIONAL ]
                    base_type `;` .
167 attribute_decl = attribute_id | qualified_attribute .
171 base_type = aggregation_types | simple_types | named_types .
```

Примечание 1 — Синтаксис для **qualified_attribute** предусматривает переопределение атрибута, которое описано в 9.2.3.4.

Правила и ограничения

a) Если явный атрибут не объявлен как OPTIONAL, каждый экземпляр типа данных объекта должен иметь значение для этого атрибута.

b) Ключевое слово OPTIONAL указывает на то, что в данном экземпляре объекта атрибут не обязан иметь значение. Если атрибут не имеет значения, то считается, что его значение будет неопределенным (?).

Ключевое слово OPTIONAL указывает на то, что атрибут всегда имеет смысл для экземпляров этого типа объекта, но для некоторых экземпляров может не быть такого значения, которое играло бы роль, определенную атрибутом. OPTIONAL не указывает на то, что атрибут не имеет значения для некоторых экземпляров типа данных объекта.

Примечания

2 Случай, когда атрибут не имеет значения для некоторых экземпляров, правильно моделируется путем определения подтипа (см. 9.2.3).

3 Необходимо обратить внимание на ссылки к необязательным атрибутам, особенно в правилах, так как они могут не иметь никакого значения. Встроенная функция EXISTS может быть использована для определения существования значения или встроенная функция NVL может быть использована для обеспечения вычисления значения по умолчанию. Если ни одна из них не используется, могут быть получены неожиданные результаты.

Пример 39 — 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 собой такое свойство, значение которого вычисляется путем присвоения численного значения выражению. Вычисляемые атрибуты объявляются после ключевого слова DERIVE. Объявление состоит из идентификатора атрибута, типа его представления и выражения, используемого для вычисления значения атрибута.

```
Синтаксис:
190 derived_attr = attribute_decl `:` base_type `:=` expression `;` .
167 attribute_decl = attribute_id | qualified_attribute .
171 base_type = aggregation_types | simple_types | named_types .
```

Примечание — Синтаксис для **qualified_attribute** предусматривает переопределение атрибута, которое описано в 9.2.3.4.

Выражение может ссылаться на любой атрибут, константу (включая SELF) или идентификатор функции, который находится в области применения.

Правила и ограничения

a) Выражение должно быть совместимо по присваиванию (см. 13.3) с типом данных атрибута.

b) Для конкретного экземпляра объекта значение вычисляемого атрибута определяется присвоением значения выражению путем замены каждого обнаруженного SELF на текущий экземпляр, а каждой ссылки на атрибут — на соответствующее значение атрибута.

Пример 40 — В данном примере, круг (circle) определяется центром (centre), осью (axis) и радиусом (radius). В дополнение к этим явным атрибутам необходимо учесть важные свойства, такие как площадь (area) и периметр (perimeter). Это выполняется путем определения их как вычисляемых атрибутов, задающих значения в виде выражений.

```
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 устанавливает отношение с данным объектом посредством явного атрибута, инверсный атрибут может быть использован, чтобы описать это отношение в контексте данного объекта. Этот инверсный атрибут может также быть использован для дальнейшего ограничения отношения.

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

Основные ограничения инверсного отношения устанавливаются спецификацией границы инверсного атрибута таким же образом, как для явных атрибутов.

Примечание 1 — Подробная информация об отношении между явными и инверсными атрибутами приведена в приложении G.

Инверсный атрибут представляется типом данных объекта или типами BAG и SET, являющимися основным типом типа данных объекта. На тип данных объекта ссылаются как на ссылочный объект.

Объявление инверсного атрибута также называет явный атрибут ссылочного объекта. Для конкретного экземпляра типа данных данного объекта значение инверсного атрибута состоит из экземпляра или экземпляров типов данных ссылочного объекта, которые используют данный экземпляр в определенной роли.

Каждое из трех возможных представлений типов для инверсии налагает некоторые ограничения на отношение между двумя объектами.

Тип данных BAG. Указание границ, при его наличии, определяет минимальное и максимальное число экземпляров ссылочного объекта, которое может использовать экземпляр данного объекта. Так как мультимножество может содержать отдельный экземпляр несколько раз, то несколько экземпляров могут ссылаться на данный экземпляр, а конкретный экземпляр может ссылаться на данный экземпляр более одного раза.

Примечания

2 Если инвертированный атрибут представлен неуникальным сборным (агрегатным) типом данных, то есть списком или массивом, для которых не установлено ключевое слово UNIQUE, или мультимножеством, конкретный экземпляр данного объекта может быть использован конкретным экземпляром ссылочного объекта несколько раз.

3 Если инвертированный атрибут представлен уникальным сборным (агрегатным) типом данных, например списком или набором, которые определены ключевым словом UNIQUE, или набором, конкретный экземпляр данного объекта может быть использован только один раз конкретным экземпляром ссылочного объекта.

Факультативно инверсный атрибут выражается определением нижней границы равным нулю, показывающим, что на заданный экземпляр данного объекта не обязан ссылаться ни один экземпляр ссылочного объекта.

Тип данных SET. Как и для BAG, но с дополнительным ограничением, что ссылочные экземпляры должны быть уникальными. Это ограничение также означает, что конкретный ссылочный экземпляр может использовать данный экземпляр в инвертированной роли только один раз.

Примечание 4 — Если инвертированный атрибут представлен уникальным сборным (агрегатным) типом данных, например списком или массивом, которые определены ключевым словом UNIQUE, или набором, инверсия не добавляет новых ограничений относительно уникальности.

Тип данных объекта. Инверсный атрибут точно указывает, что имеется один экземпляр типа данных ссылочного объекта, который использует данный экземпляр в конкретной роли. В этом случае мощность связи инверсного отношения 1:1.

Синтаксис:

```
234 inverse_attr = attribute_decl `:` [ ( SET | BAG ) [ bound_spec ] OF ] entity_ref
                   FOR attribute_ref `;` .
167 attribute_decl = attribute_id | qualified_attribute .
176 bound_spec = `[` bound_1 `:` bound_2 `]` .
174 bound_1 = numeric_expression .
175 bound_2 = numeric_expression .
```

Правила и ограничения

a) Объект, определяющий объявление прямого отношения с данным объектом, должен делать это так же, как и явный атрибут.

b) Тип данных явного атрибута в объекте, определяющем объявление прямого отношения, должен быть объявлен данным объектом через один из его супертипов или сборным (агрегатным) типом данных, используя данный объект или один из его супертипов как основной тип.

Пример 41 — Предполагается, что мы имеем следующее объявление для дверного блока (door assembly):

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

Затем мы можем пожелать ограничить объявление кнопок (knob) так, чтобы кнопки могли присутствовать только, если они использованы в роли **handle** в одном из экземпляров двери.

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

С другой стороны, мы можем просто пожелать определить, чтобы кнопка использовалась в одной или не в одной двери (например, она либо уже 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 утверждаются для области значений экземпляров объекта и таким образом действуют для всех экземпляров типа данных этого объекта. Существуют два вида локальных правил. Правила уникальности контролируют уникальность значений атрибутов среди всех экземпляров типа данных заданного объекта. Правила области значений описывают другие ограничения на значения атрибутов каждого экземпляра типа данных заданного объекта или между ними.

Каждому из локальных правил может быть присвоена метка правила. Ссылки на метки правил в настоящем стандарте не определены.

Примечание — Метки правил, при их наличии, могут быть использованы для обозначения правил в реализациях, например в документации, сообщениях об ошибках и технических заданиях. Присвоение меток правилам предполагается использовать именно для этой цели.

9.2.2.1 *Правило уникальности*

В правиле уникальности может быть определено ограничение уникальности для отдельных атрибутов или комбинаций атрибутов. Правила уникальности следуют за ключевым словом UNIQUE и определяют или одиночное имя атрибута, или список имен атрибута. Правило, которое определяет имя одиночного атрибута, называемое простым правилом уникальности, устанавливает, что никакие два экземпляра типа данных объекта в области значений не должны использовать тот же самый экземпляр для поименованного атрибута. Правило, которое определяет два или несколько имен атрибутов, называемое совместным правилом уникальности, определяет, что никакие два экземпляра типа данных объекта не должны иметь ту же самую комбинацию экземпляров для поименованных атрибутов.

Примечание — Проверяется эквивалентность экземпляров, а не равенство значений (см. 12.2.2).

Синтаксис:
310 unique_clause = UNIQUE unique_rule `;` { unique_rule `;` } .
311 unique_rule = [label `:`] referenced_attribute { `,` referenced_attribute } .
266 referenced_attribute = attribute_ref | qualified_attribute .

Правила и ограничения

Когда 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, правило уникальности ни нарушено, ни утверждено и, следовательно, экземпляр объекта является элементом области значений.

Примеры

42 — Если объект имел три атрибута, называемые a, b и c, мы могли бы получить:

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

Это означает, что два экземпляра объявляемого типа данных объекта не могут иметь одно и тоже значение для a, b или c.

43 — Объект **person_name** может выглядеть следующим образом:

```
ENTITY person_name;
    last       : STRING;
    first      : STRING;
    middle     : STRING;
    nickname   : STRING;
END_ENTITY;
```

и может быть использован как:

```
ENTITY employee;
```

```
    badge : NUMBER;
    name : person_name;
    ...
UNIQUE
    url : badge, name;
    ...
END_ENTITY;
```

В этом примере два экземпляра объекта **person_name** могли бы иметь тот же самый набор значений для четырех атрибутов. В случае служащего (**employee**), однако, имеется требование, чтобы признак (**badge**) и имя (**name**) вместе были уникальными. Таким образом, два экземпляра **employee** могут иметь то же самое значение или **badge,** или **name**. Однако никакие два экземпляра **employee** не могут иметь одинаковых экземпляров **badge** и **name** вместе, так как эта комбинация экземпляров должна быть уникальна (см. 9.6 для метода уникальности значения описываемого атрибута).

9.2.2.2 *Правила области значений (оператор WHERE)*

Правила области значений ограничивают значения отдельных атрибутов или комбинаций атрибутов для каждого экземпляра объекта. Все правила области значений следуют за ключевым словом WHERE.

Синтаксис:
315 where_clause = WHERE domain_rule `:` { domain_rule `;` } .

Правила и ограничения

a) Каждому выражению правила области значений должно присваиваться логическое (TRUE, FALSE или UNKNOWN) или неопределенное (?) значение.

b) Каждое выражение правила области значений должно включать ссылку на SELF или атрибуты, объявленные внутри объекта или любого из его супертипов.

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

d) Правило области значений будет верным, когда выражению присвоено значение TRUE, будет нарушенным, когда выражению присвоено значение FALSE, и будет ни верным, ни нарушенным, если выражению присвоено неопределенное (?) значение или значение UNKNOWN.

e) Для того чтобы экземпляр объекта был верным (в области значений) не должно быть нарушено ни одного правила.

Пример 44 — Для объекта **unit_vector** требуется, чтобы длина вектора была точно равна 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 значений

Правило области значений, которое содержит необязательный атрибут, должно трактоваться следующим образом.

Правила и ограничения

a) Когда атрибут имеет значение, правилу области значений должно быть присвоено значение как любому другому правилу области значений.

b) Когда атрибут не имеет значения, неопределенное (?) значение используется как значение атрибута при определении выражения правила области значений. Значения выражений, содержащих неопределенное (?) значение, принимаются в соответствии с разделом 12.

Пример 45 — Рассмотрим вариант примера 44.

```
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 значений является обеспечение того, чтобы объект unit_vector был единичным. Однако когда с имеет неопределенное (?) значение, правилу области значений всегда присваивается значение UNKNOWN, независимо от значений a и b.

Стандартная функция NVL (см. 15.18) может быть использована для обеспечения приемлемого значения, когда необязательный атрибут имеет неопределенное (?) значение. Когда необязательный атрибут имеет значение, функция NVL возвращает данное значение; в противоположном случае она возвращает заменяющее 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 наследование (например, подтип/супертип) отношения между объектами, в которых подтип наследует свойства (например, атрибуты и ограничения) соответствующего супертипа. Последовательные отношения подтип/супертип устанавливают граф наследования, в котором каждый экземпляр подтипа является экземпляром его супертипа(ов).

Граф наследования, установленный отношениями подтип/супертип, должен быть ациклическим.

Объявление объекта, которое полностью определяет все важные свойства данного объекта, объявляет простой тип данных объекта. Объявление объекта, которое устанавливает наследуемые отношения с супертипами, объявляет сложный тип данных объекта. Сложный тип данных объекта внутри графа наследования также использует характеристики соответствующего супертипа(ов). Сложный тип данных объекта может иметь дополнительные характеристики, не содержащиеся внутри супертипа(ов).

Следующие факты относятся к отношениям подтип/супертип. Эти факты ссылаются на граф подтип/супертип. Граф подтип/супертип является многомерным направленным ациклическим графом, в котором узлы представляют типы объекта, а связи представляют отношения подтипа/супертипа. Следующие за SUBTYPE OF связи ведут к супертипам, тогда как следующие за SUPERTYPE OF связи ведут к подтипам.

Синтаксис:

```
294 subsuper = [ supertype_constraint ] [ subtype_declaration ] .
297 supertype_constraint = abstract_supertype_declaration | supertype_rule .
156 abstract_supertype_declaration = ABSTRACT SUPERTYPE [subtype_constraint ] .
295 subtype_constraint = OF `(` supertype_expression `)` .
298 supertype_expression = supertype_factor { ANDOR supertype_factor } .
299 supertype_factor = supertype_term { AND supertype_term } .
301 supertype_term = entity_ref | one_of | `(` supertype_expression `)` .
250 one_of = ONEOF `(` supertype_expression { `,` supertype_expression. } `)` .
300 supertype_rule = SUPERTYPE subtype_constraint .
```

Правила и ограничения

a) Раздел супертипа (при его наличии) должен предшествовать разделу подтипа (при его наличии).

b) Подтип может иметь более одного супертипа.

c) Супертип может иметь более одного подтипа.

d) Супертип может сам быть подтипом одного или нескольких других типов объекта. То есть ветви в графе подтип/супертип могут пересекать несколько узлов.

e) Отношение подтип/супертип должно быть транзитивным. То есть, если A является подтипом B, а B является подтипом C, то A является подтипом C. Объекты, которые являются су-

пертипами конкретного типа объекта, должны быть объектами, для которых возможно пересечение связей, начиная с типа объекта и перечисленых после SUBTYPE OF.

f) Подтип не должен быть супертипом любого типа в списке всех его супертипов, то есть граф подтип/супертип является ациклическим.

9.2.3.1 *Определение подтипов*

Объект является подтипом, если он содержит объявление SUBTYPE. Объявление подтипа должно определять весь (все) непосредственный(е) супертип(ы) объекта.

Синтаксис:
296 subtype_declaration = SUBTYPE OF `(` entity_ref { `,` entity_ref } `)` .

9.2.3.2 *Определение супертипов*

Объект может проявиться как супертип посредством явного или неявного определения. Объект явно определяется как супертип, если в нем содержится объявление ABSTRACT SUPERTYPE и неявно, если он назван в объявлении подтипа по крайней мере одного другого объекта.

Синтаксис:
297 supertype_constraint = abstract_supertype_declaration | supertype_rule .
156 abstract_supertype_declaration = ABSTRACT SUPERTYPE [subtype_constraint] .
295 subtype_constraint = OF `(` supertype_expression `)` .
298 supertype_expression = supertype_factor { ANDOR supertype_factor } .
299 supertype_factor = supertype_term { AND supertype_term } .
301 supertype_term = entity_ref | one_of | `(` supertype_expression `)` .
250 one_of = ONEOF `(` supertype_expression { `,` supertype_expression } `)` .
300 supertype_rule = SUPERTYPE subtype_constraint .

Правила и ограничения

Все подтипы, упоминаемые в выражении супертипа, должны содержать объявление подтипа, которое определяет данный объект как супертип.

Пример 46 — 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 атрибута*

Идентификаторы атрибута в супертипе видимы внутри области применения подтипа (см. раздел 10). Таким образом, подтип наследует все атрибуты данного супертипа. Это позволяет подтипам определять ограничения или свои собственные атрибуты с использованием унаследованного атрибута. Если подтип имеет несколько супертипов, то подтип наследует все атрибуты из соответствующих супертипов. Это называется составным наследованием.

Правила и ограничения

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

Когда подтип наследует атрибуты из двух супертипов, являющихся непересекающимися, возможно, что в них есть различные атрибуты, имеющие одинаковый идентификатор атрибута. Неоднозначность в наименовании должна быть разрешена путем прибавления к идентификатору имени супертипа объекта, из которого унаследован каждый из атрибутов.

Пример 47 — В данном примере показано, как объект **e12** наследует два атрибута, названные **attr**, а для того, чтобы определить, какой из двух атрибутов будет ограниченным, к его имени прибавляется префикс.

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

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

ENTITY el2
SUBTYPE OF (el,e2);
    ...
WHERE
    positive : SELF\el.attr > 0.0;
            -- attr как объявлено в el
END_ENTITY;
```

Подтип может наследовать тот же самый атрибут из различных супертипов, которые в свою очередь унаследовали его из одного супертипа. Это называется повторным наследованием. В этом случае подтип наследует атрибут только один раз, то есть имеется только одно значение этого атрибута в экземпляре этого типа данных объекта.

9.2.3.4 *Переобъявление атрибута*

Атрибут, объявленный в супертипе, может быть переобъявлен в подтипе. Атрибут остается в супертипе, но разрешенная область значений для этого атрибута задается переобъявлением, заданным в подтипе. Первоначальное объявление может быть изменено тремя основными способами:

- тип данных атрибута может быть изменен для конкретизации первоначального типа данных (см. 9.2.6);

Пример 48 — Атрибут типа данных NUMBER может быть изменен на тип данных INTEGER или REAL;

- необязательный атрибут в супертипе в подтипе может быть заменен обязательным;
- явный атрибут в супертипе в подтипе может быть заменен определенным атрибутом.

Синтаксис:
262 qualified_attribute = SELF group_qualifier attribute_qualifier .
219 group_qualifier = `\` entity_ref .
169 attribute_qualifier = `.` attribute_ref .

Правила и ограничения

a) Атрибут, переобъявленный в подтипе, должен быть конкретизацией атрибута с тем же именем в супертипе.

b) Имя переобъявленного атрибута должно быть задано с использованием синтаксиса **qualified_attribute**.

c) Если атрибут супертипа переобъявлен в двух не взаимно исключающих подтипах, экземпляр, содержащий оба подтипа, будет иметь единственное значение для этого атрибута, которое является допустимым для обоих переобъявлений. Реализация синтаксического анализатора языка EXPRESS, которая претендует на соответствие уровню 4, должна быть протестирована на наличие противоречий в переобъявлениях подтипов, которые могут сосуществовать в одном экземпляре.

Примеры

49 — В некоторых геометрических системах используют координаты с плавающей точкой, в то время как в других работают в целочисленном координатном пространстве.

```
ENTITY point;
    x : NUMBER;
    y : NUMBER;
END_ENTITY;

ENTITY integer_point
```

```
    SUBTYPE OF (point);
    SELF\point.x : INTEGER;
    SELF\point.y : INTEGER;
END_ENTITY;
```

50 — Этот пример показывает изменение элементов в сборном (агрегатном) типе данных в целях указания их уникальности, уменьшения числа элементов в сборном типе данных и замены необязательного атрибута обязательным.

```
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;
```

51 — В следующем примере круг определен центром, осью и радиусом. Вариант круга определен центром и двумя точками, через которые он проходит. Эти три точки представляют данные, которыми определен этот тип круга. В дополнение к этим данным необходимо описать и другие важные черты — радиус и ось. Это выполняется переобъявлением их как вычисляемых атрибутов путем задания значений их выражениям.

```
FUNCTION distance(p1, p2 : point) : REAL;
    (* Найти кратчайшее расстояние между двумя точками *)
END_FUNCTION;

FUNCTION normal(p1, p2, p3 : point) : vector;
    (*Вычислить нормаль к плоскости, заданной тремя точками на плоскости *)
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, в то время как **radius** (радиус), **axis** (ось) и **area** (площадь) являются вычисляемыми атрибутами. Зна-

чения этих вычисляемых атрибутов вычисляются с помощью выражения следующего за оператором присвоения. Значения **radius** и **axis** получаются посредством вызова функции; значение **area** вычисляется в строке.

9.2.3.5 *Правило наследования*

Каждое локальное или глобальное правило, применимое к супертипу, применимо и к его подтипу(ам). Таким образом, подтип наследует все правила своего(их) супертипа(ов). Если подтип имеет несколько супертипов, то подтип должен наследовать все правила, ограничивающие супертипы.

Невозможно изменить или удалить ни одно из правил, связаных с подтипом, через правило наследования, но возможно добавить новые правила, которые еще более ограничивают подтип.

Правила и ограничения

Экземпляр объекта должен быть ограничен всеми ограничениями, установленными для типов данных этого объекта. Если ограничения, установленные в двух (или более) типах данных объекта, противоречат друг другу, тогда не существует допустимого экземпляра, содержащего эти типы данных объекта. Реализация синтаксического анализатора языка EXPRESS, которая претендует на соответствие уровню 4, должна быть протестирована на наличие конфликтующих ограничений в типах данных объекта, которые могут сосуществовать в одном экземпляре.

Пример 52 — В следующем примере a **graduate** (дипломант) является a **person** (личностью), которая и учит, и учится. Дипломант наследует и атрибуты, и ограничения из супертипов **[teacher** (преподаватель) и **student** (студент)] вместе с атрибутами и ограничениями из их общего супертипа **(person)**. Дипломанту **(graduate)**, в отличие от преподавателя **(teacher)**, не позволено преподавать на курсах для дипломантов.

```
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 из супертипов, то не существует соответствующего экземпляра этого подтипа, поскольку любой экземпляр нарушает одно из ограничений.

9.2.4 Ограничения подтипа/супертипа

Экземпляр типа данных объекта, являющийся подтипом, является экземпляром каждого из его супертипов. Экземпляр типа данных объекта, явно или неявно объявленный как супертип (см. 9.2.3.2), также может быть экземпляром одного или нескольких его подтипов (см. G.2).

Синтаксис:

```
294 subsuper = [ supertype_constraint ] [ subtype_declaration ] .
297 supertype_constraint = abstract_supertype_declaration | supertype_rule .
156 abstract_supertype_declaration = ABSTRACT SUPERTYPE [ subtype_constraint ] .
295 subtype_constraint = OF `(` supertype_expression `)` .
298 supertype_expression = supertype_factor { ANDOR supertype_factor } .
299 supertype_factor = supertype_term { AND supertype_term } .
301 supertype_term = entity_ref | one_of | `(` supertype_expression `)` .
250 one_of = ONEOF `(` supertype_expression { `,` supertype_expression } `)` .
300 supertype_rule = SUPERTYPE subtype_constraint .
```

Имеется возможность для установления ограничений, по которым могут быть реализованы графы подтипов/супертипов. Эти ограничения задаются в супертипе посредством ограничения SUPERTYPE

В приложении В описан формальный подход к определению возможных комбинаций подтипов/супертипов, экземпляров сущностей с учетом всех ограничений, которые описаны ниже.

9.2.4.1 *Абстрактные супертипы*

Язык EXPRESS позволяет объявлять супертипы, которые не предназначаются для непосредственного создания экземпляров. С этой целью в ограничение супертипа, определяющее тип данных объекта, должна быть включена фраза ABSTRACT SUPERTYPE. Абстрактный супертип не должен быть ограничен созданием экземпляра, связанного только с одним из подтипов.

Примечание — Это означает, что схема, содержащая абстрактный супертип, не имеющий каких-либо подтипов, является неполной и не может быть использована для создания экземпляров без объявления подтипов в ссылочной схеме.

Пример 53 — В транспортной модели транспортное средство (vehicle) может быть представлено абстрактным супертипом, поскольку все экземпляры этого типа данных объекта должны быть представлены его подтипами (например, наземное, водное транспортное средство и т.д.). Тип данных объекта для транспортного средства не должен образовывать независимых экземпляров объекта.

```
ENTITY vehicle
    ABSTRACT SUPERTYPE;
END_ENTITY;

ENTITY land_based
    SUBTYPE OF (vehicle);
    ...
END_ENTITY;

ENTITY water_based
    SUBTYPE OF (vehicle);
    ...
END_ENTITY;
```

9.2.4.2 *ONEOF*

Ограничение ONEOF устанавливает, что элементы списка ONEOF являются взаимоисключающими. Ни для одного из элементов не может быть создан экземпляр с другим элементом из

данного списка. Каждый элемент должен быть выражением супертипа, которое относится к единственному подтипу типа данных объекта.

Синтаксис:
250 one_of = ONEOF `(` supertype_expression { `,` supertype_expression } `)` .
298 supertype_expression = supertype_factor {ANDOR supertype_factor } .
299 supertype_factor = supertype_term { AND supertype_term } .
301 supertype_term = entity_ref | one_of | `(` supertype_expression `)` .

Ограничение ONEOF может комбинироваться с другими ограничениями супертипа, что позволяет описывать сложные ограничения.

Примечание — На естественном языке фраза **ONEOF(a, b, c)** читается как: «экземпляр объекта должен состоять из одного и только одного из типов данных объекта — a, b, c.

Пример 54 — Экземпляр супертипа может быть установлен посредством определения экземпляра только одного из его подтипов. Данное ограничение объявляется с использованием ограничения ONEOF. Существуют различные виды домашних животных (pet), но ни одно из конкретных домашних животных не может одновременно принадлежать к двум и 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;
```

9.2.4.3 *ANDOR*

Если подтипы не являются взаимоисключающими, это означает, что экземпляр супертипа может быть одновременно экземпляром более чем одного из его подтипов, отношение между подтипами должно быть определено ограничением ANDOR.

Примечание — На естественном языке фраза **b ANDOR c** читается как: «экземпляр должен содержать типы **b** и/или **c**».

Пример 55 — Личность (person) может быть одновременно и сотрудником (employee), и студентом (student) или только одним из них.

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

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

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

9.2.4.4 *AND*

Если экземпляры супертипа классифицируются в несколько групп взаимоисключающих подтипов (то есть несколько группировок ONEOF), указывающих, что для полной классификации супертипа используется несколько признаков, отношение между группами должно быть определено с использованием ограничения AND. Ограничение AND используется только для соответствующих группировок, определенных другими ограничениями подтип/супертип.

Примечание — На естественном языке фраза **b AND c** читается как: «экземпляр должен одновременно состоять из типов **b** и **c**».

Пример 56 — Личность (person) может быть классифицирована как лицо мужского (male) или женского (female) пола, или как гражданин данного (citizen) или иностранного (alien) государства.

```
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;
```

9.2.4.5 *Приоритет операторов супертипов*

Присвоение значений выражениям супертипов производится слева направо, высший приоритет имеют операторы, выполняемые первыми. В таблице 8 представлены правила приоритетности для операторов выражений супертипов. Операторы в одной строке имеют равный приоритет, а строки упорядочиваются по мере понижения приоритета.

Таблица 8 — Приоритет оператора выражения супертипа

Приоритет	Операторы
1	( ) ONEOF
2	AND
3	ANDOR

Пример 57 — Следующие два выражения неэквивалентны:

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

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

9.2.4.6 *Ограничения между подтипами по умолчанию*

Если в объявлении объекта не упомянуто никакого ограничения супертипа, подтипы (при их наличии) должны использоваться одновременно, то есть как если бы все подтипы были бы упомянуты в конструкция ANDOR.

В случае ограничения супертипа, которое установлено для подмножества подтипов данного объекта, ограничение должно быть установлено для этих подтипов по умолчанию, а для других подтипов — через ANDOR.

Пример 58 — Модель из примера 55 эквивалентна данной модели, в которой используются конструкции по умолчанию.

```
ENTITY person
    ...
END_ENTITY;

ENTITY employee
    SUBTYPE OF (person);
    ...
END_ENTITY;

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

9.2.5 Неявные объявления

Когда объявляется объект, одновременно с этим неявным образом объявляется конструктор (constructor). Идентификатор конструктора такой же как и идентификатор объекта и видимость объявления конструктора такая же, как и объявления объекта.

Конструктор после своего вызова должен возвращать в точку вызова частное значение сложного объекта для данного типа данных объекта. Каждый атрибут этого частного значения сложного объекта задается фактическим параметром, передаваемым в вызов конструктора, если фактический параметр является экземпляром объекта, а данный экземпляр объекта играет роль, описываемую атрибутом в частном значении сложного объекта. Конструктору должны передаваться только те атрибуты, которые явно определены в объявлении конкретного объекта.

Синтаксис:
195 entity_constructor = entity_ref `(` [expression { `,` expression }] `)` .

Когда создается экземпляр сложного объекта (экземпляр объекта, встречающийся в графе подтип/супертип), конструкторы для каждого компонента объектов должны комбинироваться с помощью оператора || (см. 12.10).

Правила и ограничения

a) Конструктор должен иметь один формальный параметр для каждого явного атрибута, объявленного в соответствующем типе данных объекта. Этим не охватываются атрибуты, наследуемые от супертипов и переобъявляемые в этом типе данных объекта.

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

c) Параметрический тип данных для каждого формального параметра должен быть идентичен типу данных соответствующего атрибута.

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

Примечание — Это отличается от явно объявленных функций.

e) Необязательным (OPTIONAL) атрибутам при неявном вызове конструктора могут присваиваться неопределенные (?) значения. Это означает, что явное значение таким атрибутам не присвоено.

f) Если в экземпляре сложного объекта имеется подтип, содержащий вычисляемые атрибуты, переобъявленные из явных атрибутов в супертипе, конструктору супертипа должны быть заданы значения для этих переопределенных атрибутов. Эти значения игнорируются в вычисляемом значении.

Пример 59 — Допустим следующее объявление объекта:

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

неявно объявленный конструктор этого объекта может быть таким:

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

конструктор может использоваться для присвоения значений экземпляру этого типа данных объекта:

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

9.2.6 Конкретизация (специализация)

Конкретизация (специализация) является наиболее ограниченной формой исходного объявления. Следующие случаи являются определенными конкретизациями:

- объект-подтип является конкретизацией любого из его супертипов;
- типы INTEGER и REAL являются двумя конкретизациями типа NUMBER;
- тип INTEGER является конкретизацией типа REAL;
- тип BOOLEAN является конкретизацией типа 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**;
- допустим, что сокращение **AGG** обозначает один из типов: ARRAY, BAG, LIST или SET, тогда выражение **AGG OF item** является конкретизацией выражения **AGG OF original** при условии, что **item** является конкретизацией **original**;
- допустим, что сокращение **AGG** обозначает один из типов: BAG, LIST или SET, тогда выражение **AGG [b : t]** является конкретизацией выражения **AGG [l : u]** при условии, что $b \le t$ и $l \le b \le u$ и $l \le t \le u$;
- допустим, что сокращение **BSR** обозначает один из типов данных: BINARY, STRING или REAL, тогда выражение **BSR (length)** является конкретизацией **BSR**;
- выражение **BSR (short)** является конкретизацией **BSR (long)** при условии, что **short** меньше, чем **long**;
- тип BINARY, использующий в определении ключевое слово FIXED, является конкретизацией типа BINARY переменной длины;
- тип STRING, использующий в определении ключевое слово FIXED, является конкретизацией типа STRING переменной длины;
- определенный тип данных является конкретизацией исходного типа данных, используемого для объявления определенного типа данных.

9.3 Схема

Объявление SCHEMA определяет общую область действия для множества взаимосвязанных объявлений объектов и других типов данных.

Пример 60 — Геометрия (Geometry) может быть именем схемы, содержащей объявления точек, кривых, поверхностей и других, связанных с ними, типов данных.

Объявления могут появляться в объявлении схемы в произвольном порядке.

Объявления, сделанные в одной схеме, могут стать видимыми в области действия другой схемы посредством определения интерфейса, описанного в разделе 11.

Синтаксис:
281 schema_decl = SCHEMA schema_id `;` schema_body END_SCHEMA `;` .
280 schema_body = { interface_specification } [constant_decl]
{ declaration | rule_decl } .
228 interface_specification = reference_clause | use_clause .
189 declaration = entity_decl | function_decl | procedure_decl | type_decl .

9.4 Константа

Объявление константы используется для объявления поименованных констант. Областью действия идентификатора константы должна быть функция, процедура, правило или схема, в которых встречается объявление константы. Поименованная константа, появляющаяся в объявлении CONSTANT, должна иметь явную начальную загрузку (инициализацию), значение которой появляется в результате вычисления выражения. Поименованная константа может появляться в объявлении другой поименованной константы.

Синтаксис:
185 constant_decl = CONSTANT constant_body { constant_body } END_CONSTANT `;` .
184 constant_body = constant_id `:` base_type `:=` expression `;` .
171 base_type = aggregation_types | simple_types | named_types .

Правила и ограничения

a) Значение константы не должно изменяться после начальной загрузки.

b) Любое появление поименованной константы вне пределов объявления константы равнозначно появлению начального значения константы.

c) Выражение должно возвращать значение установленного основного типа.

Пример 61 — Следующие объявления констант являются правильными:

```
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 требуемому конечному состоянию. Существуют два вида алгоритмов, которые могут быть определены как функции и процедуры.

Формальные параметры определяют исходные данные для алгоритма. В момент вызова алгоритма фактические параметры обеспечивают фактические значения или экземпляры. Фактические параметры должны быть согласованы с формальными параметрами по типу, порядку и количеству (числу).

При необходимости локальных объявлений в алгоритме, они задаются непосредственно за заголовком алгоритма. Такими объявлениями могут быть типы, локальные переменные, другие алгоритмы и т.д.

За локальными объявлениями следует тело алгоритма.

9.5.1 Функция

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

Функция должна прерываться (завершаться) при выполнении оператора RETURN. Значение выражения, связанное с оператором RETURN, определяется результатом, полученным при вызове функции.

Синтаксис:
208 function_decl = function_head [algorithm_head] stmt { stmt } END_FUNCTION `;` .
209 function_head = FUNCTION function_id [`(` formal_parameter
{ `;` formal_parameter } `)`] `:` parameter_type `;` .
206 formal_parameter = parameter_id { `,` parameter_id } `:` parameter_type .
253 parameter_type = generalized_types | named_types | simple_types .
163 algorithm_head = { declaration} [constant_decl] [local_decl] .
189 declaration = entity_decl | function_decl | procedure_decl | type_decl .

Правила и ограничения

a) В теле функции должен быть определен оператор RETURN для каждой из возможных ветвей процесса обработки, выбираемых при обращении к данной функции.

b) Каждый оператор RETURN в пределах функции должен определять выражение, по которому вычисляется значение, возвращаемое в точку вызова.

c) Выражение, определяемое в каждом из операторов RETURN, должно быть совместимо по присваиваемому значению с объявленным возвращаемым типом функции.

d) Функции не имеют побочных эффектов. Поскольку формальные параметры функции не могут быть определены как VAR, изменения этих параметров внутри функции не отражаются в точке вызова функции.

e) Функция может изменять локальные переменные или параметры, которые объявлены во внешней области действия, то есть если данная функция объявлена в заголовке (**algorithm_head**) какой-либо функции (FUNCTION), процедуры (PROCEDURE) или правила (RULE).

9.5.2 Процедура

Процедура является алгоритмом, который принимает параметры из точки вызова и обрабатывает их некоторым образом для получения требуемого конечного состояния. Изменения параметров внутри процедуры отражаются в точке вызова только в том случае, если перед определением формального параметра стоит ключевое слово VAR (изменяемый).

Синтаксис:
258 procedure_decl = procedure_head [algorithm_head] { stmt } END_PROCEDURE `;` .
259 procedure_head = PROCEDURE procedure_id [`(` [VAR] formal_parameter
{ `;` [VAR] formal_parameter } `)`] `;` .
206 formal_parameter = parameter_id { `,` parameter_id } `:` parameter_type .
253 parameter_type = generalized_types | named_types | simple_types .
163 algorithm_head = { declaration } [constant_decl] [local_decl] .
189 declaration = entity_decl | function_decl | procedure_decl | type_decl .

Правила и ограничения

Процедуры могут изменять локальные переменные или параметры, которые объявлены во внешней области действия, то есть если данная процедура объявлена в заголовке (**algorithm_head**) какой-либо функции (FUNCTION), процедуры (PROCEDURE) или правила (RULE).

9.5.3 Параметры

Функция или процедура могут иметь формальные параметры. Каждый формальный параметр устанавливает имя и тип параметра. Имя является идентификатором, который должен быть уникальным в области действия функции или процедуры. Формальный параметр процедуры также может быть объявлен как VAR (изменяемый), чем устанавливается, что если такой параметр изменяется внутри процедуры, то изменение распространится и на точку вызова процедуры. Параметры, не объявленные как VAR, также могут быть изменены, но такое изменение не будет видимым после передачи управления в точку вызова процедуры.

Синтаксис:
206 formal_parameter = parameter_id { `,` parameter_id } `:` parameter_type .
253 parameter_type = generalized_types | named_types | simple_types .

Пример 62 — Следующие объявления показывают, как могут быть объявлены формальные параметры.

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

Для обобщения типов данных, используемых для представления формальных параметров функций и процедур, применяются обобщенные типы данных (AGGREGATE и GENERIC). Обобщенные сборные (агрегатные) типы данных также могут быть использованы для обобщения исходных типов данных, допустимых для конкретных сборных (агрегатных) типов данных.

9.5.3.1 *Агрегатный (сборный) тип данных*

Тип данных AGGREGATE является обобщением всех сборных (агрегатных) типов данных

Когда вызывается процедура или функция, формальный параметр которой определен как сборный (агрегатный) тип данных, переданный процедуре или функции фактический параметр должен быть одного из типов: ARRAY, BAG, LIST или SET. Тогда выполняемые действия должны зависеть от типа данных фактического параметра.

Для обеспечения того, чтобы два или более переданных параметра имели одинаковый тип данных или чтобы возвращаемый тип данных был таким же, как у одного из переданных параметров, независимо от переданных фактических типов данных (см. 9.5.3.3), могут быть использованы метки типов.

Синтаксис:
161 aggregate_type = AGGREGATE [`:` type_label] OF parameter_type .
306 type_label = type_label_id | type_label_ref .
253 parameter_type = generalized_types | named_types | simple_types .

Правила и ограничения

a) Тип данных AGGREGATE должен использоваться только как тип формального параметра функции или процедуры, или в соответствии с правилом (b).

b) Тип данных AGGREGATE может также использоваться в качестве типа результата функции или типа локальной переменной внутри функции или процедуры. Для такого применения требуются ссылки на метки типов и необходимо ссылаться на метки типов, объявленные формальными параметрами (см. 9.5.3.3).

Пример 63 — Данная функция написана для приема агрегата чисел. Функция должна возвращать тот же тип, что и у переданного агрегата, содержащий масштабированные числа.

```
FUNCTION scale(input:AGGREGATE : intype OF NUMBER;
               scalar : NUMBER) : AGGREGATE : intype OF NUMBER;
  LOCAL
    result : AGGREGATE : intype OF NUMBER;
  END_LOCAL;

  REPEAT i := LOINDEX(input) TO HIINDEX(input);
    result[i] := scalar * input [i];
  END_REPEAT;

  RETURN(result);
END_FUNCTION;
```

9.5.3.2 *Обобщенный тип данных*

Тип данных GENERIC является обобщением всех других типов данных.

Когда вызывается процедура или функция с обобщенным параметром, переданный фактический параметр может не иметь тип данных GENERIC. Выполняемые операции зависят от типа данных фактического параметра.

Для обеспечения того, чтобы два или более переданных параметра имели одинаковый тип данных или чтобы возвращаемый тип данных был таким же, как у одного из переданных параметров, независимо от переданных фактических типов данных (см. 9.5.3.3), могут быть использованы метки типов.

Синтаксис:
218 generic_type = GENERIC [`:` type_label] .
306 type_label = type_label_id | type_label_ref .

Правила и ограничения

a) Тип данных GENERIC должен использоваться только как тип формального параметра функции или процедуры, или в соответствии с правилом (b).

b) Тип данных GENERIC может также использоваться в качестве типа результата функции или типа локальной переменной внутри функции или процедуры. Для такого применения требуются ссылки на метки типов и необходимо ссылаться на метки типов, объявленные формальными параметрами (см. 9.5.3.3).

Пример 64 — В этом примере показана обобщенная функция, которая складывает числа или вектора

```
FUNCTION add(a,b:GENERIC:intype):GENERIC:intype;
   LOCAL
      nr : NUMBER; -- целое или действительное
      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 TYPEQF(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 мы получаем неправильные исходные данные, тогда возвращается неопределенное значение
END_FUNCTION;
```

9.5.3.3 *Метки типов*

Метки типов должны использоваться для установления отношения между типом данных фактического параметра в точке вызова с типами данных других фактических параметров, локальных переменных или возвращаемого типа функции. Метки типов объявляются для типов данных AGGREGATE и GENERIC внутри объявления формального параметра функции или процедуры и на них могут быть даны ссылки типами данных AGGREGATE или GENERIC в объявлении формального параметра, локальной переменной или возвращаемого типа функции.

Синтаксис:
306 type_label = type_label_id | type_label_ref .

Правила и ограничения

a) Первое появление метки типа в объявлении формального параметра объявляет метку данного типа, все последующие появления метки данного типа являются ссылками на первую метку.

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

c) Типы данных локальных переменных и возвращаемых типов функций, которые ссылаются через метку типа на тип данных параметра, должны быть идентичными типу данных параметра, в котором объявлена метка типа.

Пример 65 — В этом примере показано, как метки типов могут использоваться при определении функции и проверке совместимости результирующего типа вызываемой 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, описанный в 12.11
v_c : c := a(...)| |b(...)| |c(...);
v_x : b;
END_LOCAL;

v_x := test(v_b, v_a); -- неверный v_a не совместим с типом b.
v_x := test(v_a, v_b); -- неверное присваивание, функция вернет тип a.
```

В разделе 15 приведены другие примеры использования меток типов.

9.5.3.4 *Общие агрегатные (сборные) типы данных*

Общие сборные (агрегатные) типы данных образуют часть класса типов, называемых обобщенными типами данных. Они представляют собой конкретный сборный (агрегатный) тип данных (ARRAY, BAG, LIST и SET) с ослабленными ограничениями, которые обычно налагаются при определении этот сборного (агрегатного) типа данных (то есть **general_list_type** является обобщением **list_type**).

Синтаксис:

```
212 general_aggregation_types = general_array_type | general_bag_type |
                                general_list_type | general_set_type .
213 general_array_type = ARRAY [ bound_spec ] OF [ OPTIONAL ] [ UNIQUE ]
                         parameter_type .
176 bound_spec = `[` bound_1 `:` bound_2 `]` .
174 bound_1 = numeric_expression .
175 bound_2 = numeric_expression .
253 parameter_type = generalized_types | named_types | simple_types .
214 general_bag_type = BAG [ bound_spec ] OF parameter_type .
215 general_list_type = LIST [ bound_spec ] OF [ UNIQUE ] parameter_type .
217 general_set_type = SET [ bound_spec ] OF parameter_type .
```

Когда общие сборные (агрегатные) типы данных используются как тип данных формального параметра, допускаются следующие фактические параметры функций и процедур:

- массивы, безотносительно к диапазону значений индексов. Это означает, что в определении формального параметра для массива не определено **bound_spec**.

Примечание — Для того чтобы определить действительные границы массива, в алгоритме должны использоваться функции HIINDEX и LOINDEX;

- агрегации, у которых исходными типами могут быть GENERIC, AGGREGATE или общий сборный (агрегатный) тип данных.

Пример 66 — В этом примере показано, как SET может быть записан в объявлении формального параметра; это не может быть описано в объявлении атрибута, поскольку исходный тип для SET не может включать 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 действия алгоритма, в котором она объявлена. Локальным переменным могут быть присвоены значения, и эти переменные могут присутствовать в выражениях.

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

Инициализация (начальная загрузка) локальных переменных

В момент вызова алгоритма все локальные переменные имеют неопределенное (?) значение до тех пор, пока явно не задан инициализатор. Если инициализатор задан, то локальной переменной при вызове алгоритма присваивается начальное значение.

Пример 67 — Переменной **r_result** присвоено начальное значение 0.0

```
LOCAL
    r_result : REAL := 0.0;
    i_result : INTEGER;
END_LOCAL;

...
EXISTS(r_result) -- TRUE
EXISTS(i_result) -- FALSE подразумевается, что присвоенных значений нет
```

9.6 Правило

Правила позволяют определить ограничения, накладываемые внутри области действия схемы на один или более типов данных объекта. Локальные правила (то есть правила уникальности ограничений и области значений в объявлении объекта) объявляют ограничения, налагаемые отдельно на каждый экземпляр типа данных объекта. Объявление RULE позволяет определить ограничения, налагаемые совокупно на всю область значений типа данных объекта или на экземпляры нескольких типов данных объекта. Одним из применений RULE является согласованное ограничение значений атрибутов, присутствующих в различных объектах.

При объявлении правила ему присваивается имя и определяется, к каким объектам оно относится.

Синтаксис:
277 rule_decl = rule_head [algorithm_head] { stmt } where_clause END_RULE `;` .
278 rule_head = RULE rule_id FOR `(` entity_ref { `,` entity_ref } `)` `;` .
163 algorithm_head = { declaration } [constant_decl] [local_decl] .
189 declaration = entity_decl | function_decl | procedure_decl | type_decl .

Тело правила состоит из локальных объявлений, исполняемых операторов и правил областей значений. Окончательная формулировка правила показывает, должно ли быть удовлетворено или нет некоторое глобальное ограничение. Правило оценивается посредством исполнения операторов с последующей оценкой каждого из правил областей значений. Если правило нарушено для набора экземпляров типов данных объекта, переданных в качестве параметров, то экземпляры не соответствуют EXPRESS-схеме.

Правила и ограничения

a) Каждое правило области значений должно определяться логическим (LOGICAL) или неопределенным (?) значением.

b) Выражение верно, когда ему присвоено значение TRUE; выражение неверно, когда ему присвоено значение FALSE; и оно ни верно, ни неверно, когда ему присвоено неопределенное (?) значение или значение UNKNOWN.

c) Правило области значений не должно быть нарушено для верного набора экземпляров объектов типов данных объекта, определенных в заголовке правила.

Пример 68 — Следующее правило требует, чтобы в первом и седьмом октантах находилось равное количество точек.

```
RULE point_match FOR (point);
LOCAL
    first_oct ,
    seventh_oct : SET OF POINT := []; -- пустой набор точек (see 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;
```

Пример 69 — Правило может быть использовано для установления общего значения, уникального для атрибутов объекта.

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

Общее ограничение уникальности в **b** применяется к экземплярам **c** и **d**. Следующее правило еще более ограничивает основное значение общей уникальности.

```
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
      wr1 : VALUE_UNIQUE(s);
END_RULE;
```

Неявное объявление

Внутри RULE каждая совокупность неявно объявляется как локальная переменная, содержащая набор всех экземпляров поименованного объекта в области значений, то есть набор экземпляров объекта, подчиняющихся данному правилу.

Синтаксис:
254 population = entity_ref .

Правила и ограничения

Ссылки на конкретную совокупность (**population**) могут быть даны только в глобальном правиле, которое содержит ссылку на соответствующий тип данных объекта в заголовке правила.

Пример 70 — Рассмотрим следующее объявление:

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

неявное объявление выглядело бы:

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

10 Область действия и видимость

EXPRESS-объявление образует идентификатор, который может быть использован для ссылки на объявленный элемент из других частей данной схемы (или из других схем). Некоторые конструкции EXPRESS неявно объявляют элементы языка EXPRESS, присваивая им идентификаторы. В тех областях, из которых можно ссылаться на идентификатор объявленного элемента, считают объявленный элемент видимым в этой области. Ссылки на элемент могут появляться только там, где видимым является его идентификатор. Правила видимости приведены в 10.2. Подробная информация о ссылках на элементы с использованием их идентификаторов приведена в 12.7.

Некоторые элементы EXPRESS образуют фрагмент (блок) текста, называемый областью действия элемента. Эта область действия ограничивает видимость идентификаторов, объявленных внутри нее. Области действия могут быть вложенными, то есть элемент языка, образующий область действия, может входить в область действия другого элемента. Существуют ограничения, при которых элементы могут появляться в области действия конкретного элемента языка EXPRESS. Эти ограничения, как правило, определяются синтаксисом EXPRESS (см. приложение А).

Для каждого из элементов, указанных в таблице 9, в последующих подразделах установлены границы образованной области действия, при ее наличии, и видимость объявленного идентификатора как в общих терминах, так и с конкретными деталями.

Таблица 9 — Области действия и идентификаторы определенных элементов

Элемент	Область действия	Идентификатор
alias statement	•	•[1]
attribute		•
constant		•
enumeration		•
entity	•	•
function	•	•
parameter		•
procedure	•	•
query expression	•	•[1]
repeat statement	•	•[1], [2]
rule	•	•[3]
rule label		•
schema	•	•
type	•	•
type label		•
variable		•

[1] Идентификатором является неявно объявленная переменная в области действия объявления.
[2] Переменная объявляется неявно только тогда, когда установлен инкрементный контроль.
[3] Неявное объявление переменной проводится для всех объектов, которые ограничены правилом.

10.1 Правила областей действия

Ниже приведены общие правила, применяемые ко всем формам областей действия, существующим в языке EXPRESS; список элементов языка, образующих области действия, приведен в таблице 9.

Правила и ограничения

a) Все объявления должны присутствовать в пределах области действия.

b) В пределах единственной области действия идентификатор может быть объявлен или явно импортирован (см. раздел 11) только один раз. Идентификатор объекта или типа, явно импортированный в данную схему более одного раза разными способами, с использованием одного и того же исходного объявления, учитывается только один раз.

c) Области действия должны быть вложены корректно, то есть перекрытие областей действия не допускается. (Это определено синтаксисом языка).

В настоящем стандарте максимально допустимая глубина вложения областей действия не установлена, но разработчики синтаксических анализаторов языка EXPRESS могут установить максимальную глубину вложения областей действия.

10.2 Правила видимости

Ниже приведены правила видимости идентификаторов. Список элементов языка EXPRESS, объявляющих идентификаторы, приведен в таблице 9. Правила видимости идентификаторов поименованных типов данных несколько отличаются от правил для других идентификаторов; эти отличия описаны в 10.2.2.

10.2.1 Общие правила видимости

Следующие общие правила применимы ко всем идентификаторам, кроме идентификаторов поименованных типов данных, на которые правило (d) не распространяется.

Правила и ограничения

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

b) Если идентификатор видим в данной области действия, он также видим во всех областях действия, определенных внутри данной области в соответствии с правилом (d).

c) Идентификатор не видим в любой области действия вне его локальной области действия в соответствии с правилом (f).

d) Когда идентификатор *i*, видимый в области действия *P*, переобъявлен в некоторой внутренней области действия *Q*, вложенной в *P*, в области действия *Q* и в любых областях действия, объявленных в *Q*, будет виден только идентификатор *i*, объявленный в области действия *Q*. Идентификатор *i*, объявленный в области действия *P*, является видимым в области действия *P* и в любых внутренних областях действия, которые не переобъявляют *i*.

e) Считается, что встроенные константы, функции, процедуры и типы языка EXPRESS объявлены в виртуальной универсальной области действия. Все схемы являются вложенными в эту область действия. Идентификаторы, относящиеся к встроенным константам, функциям, процедурам, типам языка EXPRESS и схемам, видимы во всех областях действия, определенных в языке EXPRESS.

f) Идентификаторы перечисляемых элементов, объявленные в области действия определенного типа данных, видимы в следующей внешней области действия, за исключением случая, когда следующая внешняя область действия содержит объявление того же идентификатора для некоторого другого элемента.

Примечание — Если следующая внешняя область действия содержит объявление того же идентификатора, перечисляемые элементы остаются доступными, но к ним добавляется идентификатор определенного типа данных (см. 12.7.2).

g) Объявления из одной схемы могут быть видимыми для элементов другой схемы через определение (спецификацию) интерфейса (см. раздел 11).

Пример 71 — В следующей схеме показаны примеры идентификаторов и ссылок на них в соответствии с выше приведенными 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 ; -- выполняется правило (a) : a видим в локальной области действия
        wr2: a <> b ; -- выполняется правило (b) : b видим из внешней области действия
    END_ENTITY;
```

```
ENTITY entity2;
    c : REAL; -- выполняется правило (c) : здесь константа c невидима
END_ENTITY;

ENTITY d;
    attr1 : INTEGER;
    attr2 : enum;
WHERE
    d : ODD(attr1); -- выполняется правило (d) : функция ODD видима везде
    wr : attr2 <> e; -- выполняется правило (e) : e видим вне области действия,
                     -- определенной типом enum
END_ENTITY;

END SCHEMA;
```

10.2.2 Правила видимости идентификаторов поименованных типов данных

Идентификаторы поименованных типов данных подчиняются тем же правилам видимости, что и прочие идентификаторы, за одним исключением. Этим исключением является правило видимости (d). Идентификатор *i* объекта или определенного типа данных, объявленный в области действия *P*, остается видимым во внутренней области действия *Q*, даже если он переопределяется в области действия *Q*, тем самым обеспечивается, что:

a) область действия *Q* определяется объявлением объекта, а идентификатор *i* объявляется как атрибут в данной области, или

b) область действия *Q* определяется объявлением функции, процедуры или правила, а идентификатор *i* объявляется как формальный параметр или переменная в данной области.

Пример 72 — В объекте **entity1 d** относится как к типу данных объекта, так и к атрибуту.

```
SCHEMA example;

ENTITY d;
    attr1 : REAL;
END_ENTITY;

ENTITY entity1;
    d : d; -- d в данной области действия является и объектом и атрибутом
END_ENTITY;

END SCHEMA;
```

10.3 Явные правила для элементов

В этом разделе дается более подробное описание того, как общие правила областей действия и видимости применяются к различным элементам языка EXPRESS.

10.3.1 Оператор переименования (alias statement)

Определение оператора ALIAS (переименования) см. в 13.2.

Видимость. Идентификатор, неявно объявленный в операторе переименования, является видимым в области действия, определенной оператором переименования.

Область действия. Оператор переименования определяет новую область действия. Эта область действия простирается от ключевого слова ALIAS до ключевого слова END_ALIAS, которое завершает данный оператор переименования.

Объявления. Следующие элементы языка EXPRESS могут объявляться идентификаторами в области действия оператора переименования:

- оператор переименования;
- выражение запроса;
- оператор цикла.

10.3.2 Атрибут (attribute)

Видимость. Идентификатор атрибута является видимым в области действия объекта, в которой он объявлен, и в областях действия всех подтипов данного объекта.

10.3.3 Константа (constant)

Видимость. Идентификатор константы является видимым в области действия функции, процедуры, правила или схемы, в которой он объявлен.

10.3.4 Элемент перечисления (enumeration item)

Видимость. Идентификатор элемента перечисления является видимым в области действия функции, процедуры, правила или схемы, в которой объявлен его тип. Это является исключением из правила видимости 10.2.1f. Идентификатор не должен объявляться с любым другим предназначением в данной области действия, за исключением объявления другого перечисляемого типа данных в той же области действия. Если один и тот же идентификатор объявлен двумя перечисляемыми типами данных как элемент перечисления, то при ссылке на элемент перечисления следует добавлять идентификатор типа данных для устранения неоднозначности ссылки (см. 8.4.1).

10.3.5 Объект (entity)

Видимость. Идентификатор объекта является видимым в области действия функции, процедуры, правила или схемы, в которой объявлен его тип. Идентификатор объекта остается видимым при условиях, определенных 10.2.2, во внутренних областях действия, в которых переобъявлен данный идентификатор.

Область действия. Объявление объекта определяет новую область действия. Данная область действия простирается от ключевого слова ENTITY до ключевого слова END_ENTITY, которое завершает данное объявление объекта. Атрибуты, объявленные в супертипе объекта, являются видимыми в объекте подтипа за счет наследования.

Примечание — Область действия объекта подтипа не считается вложенной в область действия супертипа.

Объявления. Следующие элементы языка EXPRESS могут объявляться идентификаторами в области действия объявления объекта:

- атрибуты (явные, вычисляемые и инверсные);
- метка правила (правил уникальности и областей значений);
- выражение запроса (внутри вычисляемых атрибутов и правил областей значений).

Пример 73 — Идентификаторы атрибута **batt** в двух объектах не конфликтуют, поскольку они объявлены в разных областях действия.

```
ENTITY entity1;
    aatt : INTEGER;
    batt : INTEGER;
END_ENTITY;

ENTITY entity2;
    a    : entity1;
    batt : INTEGER;
END_ENTITY;
```

Пример 74 — Следующая спецификация недопустима, поскольку идентификатор атрибута **aatt** в области действия объекта **illegal** одновременно и наследуется, и объявляется (см. 9.2.3.3). Метки правила **lab** в двух объектах не конфликтуют, поскольку они объявлены в разных областях действия; верный экземпляр объекта **illegal**, игнорируя ошибку с атрибутом **aatt**, подчиняется обоим правилам областей значений.

```
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 (function)

Видимость. Идентификатор функции является видимым в области действия функции, процедуры, правила или схемы, в которой он объявлен.

Область действия. Объявление функции определяет новую область действия. Эта область действия простирается от ключевого слова FUNCTION до ключевого слова END_FUNCTION, завершающего объявление данной функции.

Объявления. Следующие элементы языка EXPRESS могут объявляться идентификаторами в области действия объявления функции:

- оператор переименования;
- константа;
- объект;
- перечисление;
- функция;
- параметр;
- процедура;
- выражение запроса;
- оператор возврата;
- тип;
- метка типа;
- переменная.

Пример 75 — Следующие определения неверны, 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 (parameter)

Видимость. Идентификатор формального параметра является видимым в области действия функции или процедуры, в которой он объявлен.

10.3.8 Процедура (procedure)

Видимость. Идентификатор процедуры является видимым в области действия функции, процедуры, правила или схемы, в которой он объявлен.

Область действия. Объявление процедуры определяет новую область действия. Эта область действия простирается от ключевого слова PROCEDURE до ключевого слова END_PROCEDURE, завершающего объявление процедуры.

Объявления. Следующие элементы языка EXPRESS могут объявляться идентификаторами в области действия объявления процедуры:

- оператор переименования;
- константа;
- объект;
- перечисление;
- функция;
- параметр;
- процедура;
- выражение запроса;
- оператор возврата;
- тип;
- метка типа;
- переменная.

10.3.9 Выражение запроса (query expression)

Определение выражения QUERY (запрос) см. в 12.6.7.

Видимость. Идентификатор, неявно объявленный в выражении запроса, является видимым в области действия, определенной выражением запроса.

Область действия. Выражение запроса определяет новую область действия. Эта область действия простирается от открывающей круглой скобки `(` после ключевого слова QUERY до закрывающей круглой скобки `)`, завершающей данное выражение запроса.

Объявления. Следующие элементы языка EXPRESS могут объявляться идентификаторами в области действия выражения запроса:

- выражение запроса.

10.3.10 Оператор цикла (repeat statement)

Определение оператора REPEAT см. в 13.9.

Видимость. Идентификатор, неявно объявленный в операторе цикла, управляющем приращением, является видимым в области действия данного оператора цикла.

Область действия. Оператор цикла определяет новую область действия. Эта область действия простирается от ключевого слова REPEAT до ключевого слова END_REPEAT, завершающего оператор цикла.

Объявления. Следующие элементы языка EXPRESS могут объявляться идентификаторами в области действия оператора цикла:

- оператор переименования;
- выражение запроса;
- оператор цикла.

10.3.11 Правило (rule)

Видимость. Идентификатор правила является видимым в области действия схемы, в которой он правила объявил.

Примечание — Идентификатор правила может быть использован только для реализации. В языке EXPRESS отсутствуют механизмы ссылки на идентификатор правила.

Область действия. Объявление правила определяет новую область действия. Эта область действия простирается от ключевого слова RULE до ключевого слова END_RULE, завершающего объявление правила.

Объявления. Следующие элементы языка EXPRESS могут быть объявлены идентификаторами в области действия объявления правила:

- оператор переименования;
- константа;
- объект;
- перечисление;
- функция;
- параметр;
- процедура;
- выражение запроса;
- оператор возврата;
- метка правила;
- тип;
- метка типа;
- переменная.

Пример 76 — Следующее определение неверно, так как идентификатор **point** относится к объекту, на который распространяется правило, неявно объявлен как переменная внутри правила и в то же время явно объявлен как локальная переменная.

```
RULE illegal FOR (point);
LOCAL
     point : STRING;
END_LOCAL;
     ...
END_RULE;
```

10.3.12 Метка правила (rule label)

Видимость. Метка правила является видимой в области действия объекта, правила или типа, в которой она объявлена.

Примечание — Метка правила используется только в реализации. В языке EXPRESS отсутствуют механизмы ссылки на метку правила.

10.3.13 Схема (schema)

Видимость. Идентификатор схемы является видимым для всех других схем.

Примечание — В соответствующей реализации может применяться механизм обобщения областей действия, который позволяет трактовать группу схем как область действия.

Область действия. Объявление схемы определяет новую область действия. Эта область действия простирается от ключевого слова SCHEMA до ключевого слова END_SCHEMA, завершающего объявление данной схемы.

Объявления. Следующие элементы языка EXPRESS могут объявляться идентификаторами в области действия объявления схемы:

- константа;
- объект;
- перечисление;
- функция;
- процедура;
- правило;
- тип.

Пример 77 — Следующая схема неверна по двум причинам. Во-первых, идентификатор **adef** импортирован в схему с помощью оператора USE, но одновременно был объявлен как имя типа. Во-вторых, имя **fdef** используется как идентификатор в двух объявлениях (одно для объекта, а другое для функции).

```
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 Тип (type)

Видимость. Идентификатор типа является видимым в области действия функции, процедуры, правила или схемы, в которой он объявлен. Идентификатор типа остается видимым при условиях, определенных в 10.2.2, во внутренней области действия, которая переобъявляет данный идентификатор.

Область действия. Объявление типа определяет новую область действия. Эта область действия простирается от ключевого слова TYPE до ключевого слова END_TYPE, завершающего объявление типа.

Объявления. Следующие элементы языка EXPRESS могут объявляться идентификаторами в области действия объявления типа:

- перечисление;
- метка правила (правило области значений);
- выражение запроса (внутри правила области значений).

10.3.15 Метка типа (type label)

Видимость. Метка типа является видимой в области действия функции или процедуры, в которой она объявлена. Метка типа неявно объявляется при первом ее появлении в области действия, которая должна присутствовать в определении (спецификации) формального параметра. На объявленную таким образом метку типа можно ссылаться либо в определении формального параметра, либо в локальных объявлениях функции или процедуры. Если метка типа объявлена в функции, на метку типа могут делаться ссылки в спецификации возвращаемого типа функции.

10.3.16 Переменная (variable)

Видимость. Идентификатор переменной является видимым в области действия функции, процедуры или правила, в которой она объявлена.

11 Спецификация интерфейса

В этом разделе описаны конструкции, позволяющие элементам, объявленным в одной схеме, стать видимыми в другой схеме. Существуют два вида спецификации интерфейса: USE (применить) и REFERENCE (сослаться), которые обеспечивают видимость элементов. Спецификация USE позволяет элементам, объявленным в одной схеме, иметь независимые экземпляры в схеме, определяющей конструкцию USE.

Экземпляр объекта считается независимым, если он не играет роль, предписываемую атрибутом другого экземпляра объекта, то есть функция ROLESOF (см. 15.20) для независимого экземпляра объекта будет возвращать пустой набор. Тип данных объекта, который был объявлен локально в схеме или с использованием USE в данной схеме, может быть применен для независимого создания экземпляров или играть роль, предписываемую атрибутом объекта в данной схеме.

Объекты, которые явно импортированы с помощью спецификации REFERENCE или импортированы в схему неявно, должны создаваться только экземпляры, играющие роль, описанную атрибутом реализации объекта в схеме.

```
Синтаксис:
228 interface_specification = reference_clause | use_clause .
```

Внешним объявлением является любое объявление (например, объекта), которое появляется во внешней схеме (любой схеме, отличной от данной схемы).

Другое различие между двумя видами интерфейса состоит в том, что спецификация USE применяется только к поименованным типам данных (типам данных объекта и определенным типам данных), в то время как спецификация REFERENCE применяется ко всем объявлениям, за исключением правил и схем.

Внешний элемент языка EXPRESS может быть в данной схеме задан новым именем. На элемент языка EXPRESS в данной схеме следует ссылаться по новому имени, заданному после ключевого слова AS.

11.1 Спецификация интерфейса USE

Тип данных объекта или определенный тип данных, объявленный во внешней схеме, становится видимым посредством использования спецификации USE. Спецификация USE задает имя внешней схемы и факультативно объявленные в ней имена типов данных объекта или определенных типов данных. Если в спецификации USE не определен **named_types**, все поименованные типы данных, объявленные или используемые внешней схемой, трактуются как объявленные локально в данной схеме.

```
Синтаксис:
313 use_clause = USE FROM schema_ref [ `(` named_type_or_rename
                 { `,` named_type_or_rename } `)` ] `;` .
246 named_type_or_rename = named_types [ AS ( entity_id | type_id ) ] .
```

11.2 Спецификация интерфейса REFERENCE

Спецификация REFERENCE дает возможность сделать видимыми в данной схеме следующие элементы языка EXPRESS, объявленные во внешней схеме:

- константа;
- объект;
- функция;
- процедура;
- тип.

Спецификация REFERENCE задает имя внешней схемы и, факультативно, имена элементов языка EXPRESS, объявленные в ней. Если имена не установлены, то все элементы языка EXPRESS, объявленные или используемые внешней схемой, являются видимыми в текущей схеме.

```
Синтаксис:
267 reference_clause = REFERENCE FROM schema_ref [ `(` resource_or_rename
                       { `,` resource_or_rename } `)` ] `;` .
274 resource_or_rename = resource_ref [ AS rename_id ] .
275 resource_ref = constant_ref | entity_ref | function_ref | procedure_ref |
                  type_ref .
270 rename_id = constant_id | entity_id | function_id | procedure_id | type_id .
```

Внешние объявления в спецификации REFERENCE не рассматриваются как локальные объявления, а поэтому для них не могут быть созданы независимые экземпляры, но могут быть созданы экземпляры, играющие роль, описанную атрибутом объекта в данной (текущей) схеме.

11.3 Взаимодействие USE и REFERENCE

Если тип данных объекта или определенный тип данных одновременно импортированы в данную схему с использованием USE и REFERENCE, спецификация USE имеет приоритет.

Пример 78 — Операторы

```
USE FROM s1 (a1);
REFERENCE FROM s1 (a1);
```

a1 трактуется как локальное объявление.

Когда поименованный тип данных импортирован USE, этот поименованный тип данных может быть импортирован из данной схемы другой схемой с USE или REFERENCE (то есть возможен последовательный импорт из схемы в схему с помощью спецификации USE).

Пример 79 — Даны следующие два объявления схем:

```
SCHEMA s1;
    ENTITY e1;
    END_ENTITY;
END_SCHEMA;

SCHEMA s2;
USE FROM s1 (e1 AS e2);
END_SCHEMA;
```

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 REFERENCE, не рассматриваются как элементы, имеющие локальное объявление, их дальнейший импорт невозможен.

11.4 Неявные интерфейсы

Внешние объявления могут ссылаться на идентификаторы, которые не являются видимыми в данной схеме. Эти элементы языка EXPRESS, на которые ссылаются неявно, необходимы для понимания данной (текущей) схемы, но они не видимы для элементов языка EXPRESS, объявленных в данной схеме. Каждый импортированный неявно элемент также может ссылаться на другие элементы языка EXPRESS, которые не являются видимыми в данной схеме; эти элементы языка EXPRESS также необходимы для полного понимания данной (текущей) схемы.

Пример 80 — Неявно импортированные элементы и последующие неявные интерфейсы.

```
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 **e2** используется **e1**, объект **e1** неявно импортируется схемой **s2**. Однако поскольку **e1** не был явно импортирован в схему **s2**, **e1** не может упоминаться в схеме **s2**. Также, в определении **e1** используется **t1**, следовательно, **t1** также неявно импортирован в схему **s2**.

В последующих пунктах слово «импортирован» будет использоваться в смысле неявного импортирования или импортирования посредством USE или REFERENCE.

11.4.1 Интерфейсы констант

При импорте констант неявно импортируются:

- любые определенные типы данных, используемые в объявлении импортируемой константы;
- любые типы данных объекта, используемые в объявлении импортируемой константы;
- любые константы, используемые в объявлении импортируемой константы;
- любые функции, используемые в объявлении импортируемой константы.

11.4.2 Интерфейсы определенных типов данных

При импорте определенного типа данных неявно импортируются:

- любые определенные типы данных, используемые в объявлении импортируемого типа, за исключением случая, когда импортируемым типом является тип SELECT; в результате импорта этого типа ни один из выбираемых элементов не будет импортирован неявно;
- любые константы или функции, используемые в объявлении импортируемого определенного типа данных;
- любые константы или функции, используемые в правилах областей значений импортируемого определенного типа данных;
- любые определенные типы данных, представленные типом данных SELECT, чей список выбора содержит импортируемый определенный тип данных.

Пример 81 — 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 тип **sel1** представлен типом SELECT, содержащим **t1**; **sel1** является неявной ссылкой.

11.4.3 Интерфейсы типов данных объектов

При импорте типа данных объекта неявно импортируются:

- все типы данных объекта, которые являются супертипами импортируемого типа данных объекта;

Примечание — Подтипы импортируемого типа данных объекта неявно импортированы не будут, даже если они входят в выражение SUPERTYPE OF;

- все правила, ссылающиеся на импортируемый тип данных объекта и не на один или несколько других типов данных объекта, все из которых явно или неявно импортиртированы в данную схему;

- любые константы, определенные типы данных, типы данных объекта или функции, используемые в объявлении атрибутов импортируемого типа данных объекта;
- любые константы, определенные типы данных, типы данных объекта или функции, используемые в правилах области значений импортируемого типа данных объекта;
- любые определенные типы данных, представленные типом данных SELECT, имеющие в списке выбора импортируемый тип данных объекта.

Графы подтип/супертип могут быть усечены в результате следования только связям SUBTYPE OF при комплектовании неявных интерфейсов импортируемого типа данных объекта. Алгоритм, используемый для вычисления допустимых реализаций усеченного графа подтип/супертип, приведен в приложении С.

11.4.4 Интерфейсы функций

При импорте функций неявно импортируются:

- любые определенные типы данных или типы данных объекта, используемые в объявлении параметров для импортируемой функции;
- любые определенные типы данных или типы данных объекта, используемые в определении возвращаемого значения для импортируемой функции;
- любые определенные типы данных или типы данных объекта, используемые в объявлении локальных переменных внутри импортируемой функции;
- любые константы, функции или процедуры, используемые внутри импортируемой функции.

11.4.5 Интерфейсы процедур

При импорте процедуры неявно импортируются:

- любые определенные типы данных или типы данных объекта, используемые в объявлении параметров для импортируемой процедуры;
- любые определенные типы данных или типы данных объекта, используемые в объявлении локальных переменных внутри импортируемой процедуры;
- все константы, функции или процедуры, используемые внутри импортируемой процедуры.

11.4.6 Интерфейсы правил

При импорте правила неявно импортируются:

- любые определенные типы данных или типы данных объекта, используемые в объявлении локальных переменных внутри импортируемого правила;
- все константы, функции или процедуры, используемые внутри импортируемого правила.

12 Выражение

Выражение является комбинацией операторов, операндов и вызовов функций, которые вычисляются для получения значения.

Синтаксис:

```
204 expression = simple_expression [ rel_op_extended simple_expression ] .
269 rel_op_extended = rel_op | IN | LIKE .
268 rel_op = `<` | `>` | `<=` | `>=` | `<>` | `=` | `:<>:` | `:=:` .
287 simple_expression = term { add_like_op term } .
303 term = factor { multiplication_like_op factor } .
206 factor = simple_factor [ `**` simple_factor ] .
288 simple_factor = aggregate_initializer | entity_constructor |
                  enumeration_reference | interval | query_expression |
                  ( [ unary_op ] ( `(` expression `)` | primary ) ) .
308 unary_op = `+` | `-` | NOT .
256 primary = literal | ( qualifiable_factor { qualifier } ) .
244 multiplication_like_op = `*` | `/` | DIV | MOD | AND | `||` .
158 add_like_op = `+` | `-` | OR | XOR .
```

Некоторые операторы требуют одного операнда, другие операторы требуют двух операндов. Операторы, требующие только одного операнда, должны располагаться перед операндом. Операторы, требующие двух операндов, должны располагаться между этими операндами. В настоящем разделе определены операторы и установлены типы данных операндов, которые могут использоваться каждым из операторов.

Существует семь классов операторов:

a) Арифметические операторы принимают числовые операнды и выдают числовые результаты. Тип данных результирующего значения арифметического оператора зависит от оператора и типов данных операндов (см. 12.1).

b) Операторы отношений получают различные типы данных и выдают результаты типа LOGICAL (имеющие значения TRUE, FALSE или UNKNOWN).

c) Двоичные (BINARY) операторы получают двоичные (BINARY) операнды и выдают результаты типа BINARY.

d) Логические (LOGICAL) операторы получают логические (LOGICAL) операнды и выдают результаты типа LOGICAL.

e) Строковые (STRING) операторы получают строковые (STRING) операнды и выдают результаты типа STRING.

f) Сборные (агрегатные) операторы комбинируют различными способами агрегатные значения с другими агрегатными значениями или отдельными элементами и выдают результаты сборного (агрегатного) типа.

g) Операторы ссылок на компоненты и операторы индексирования извлекают компоненты из экземпляров объектов и из агрегатных значений.

Вычисление выражений осуществляется в соответствии с приоритетом операторов, входящих в выражение.

Выражение, заключенное в круглые скобки, вычисляется первым и рассматривается как единый операнд.

Вычисление осуществляется слева направо, при этом операторы с более высоким приоритетом вычисляются первыми. В таблице 10 установлены приоритеты для всех операторов языка EXPRESS. Операторы в одной строке имеют равный приоритет, а строки упорядочены по уменьшению приоритета.

Таблица 10 — Приоритет операторов

Приоритет	Описание	Операторы
1	Ссылки на компонент	[ ] . \
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.		

Если операнд расположен между операторами, имеющими разный приоритет, то операнд относится к тому оператору, который имеет более высокий приоритет. Если операнд расположен между операторами, имеющими одинаковый приоритет, то операнд относится к тому оператору, который расположен слева от операнда.

Пример 82 — Выражение -10**2 вычисляется как (-10)**2, и результат будет равен 100. Выражение 10/20*30 вычисляется как (10/20)*30, и результат будет равен 15.0.

12.1 Арифметические операторы

Арифметические операторы, требующие одного операнда, являются тождествами (+) и отрицаниями (-). Операнд должен иметь числовой тип (NUMBER, INTEGER или REAL). Для оператора (+) результат равен операнду, для оператора (—) результат противоположен операнду. Когда операнд имеет неопределенное (?) значение, результат также будет иметь неопределенное (?) значение, независимо от того, какой оператор использован.

Арифметическими операторами, требующими двух операндов, являются сложение (+), вычитание (-), умножение (*), деление (/), возведение в степень (**), целочисленное деление (DIV) и остаток от целочисленного деления (MOD). Операнды должны иметь числовой тип (NUMBER, INTEGER или REAL).

Операторы сложения, вычитания, умножения, деления и возведения в степень выполняют математические операции с теми же именами. За исключением деления, они выдают целочис-

ленный 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). Деление (/) действительного типа дает результат действительного типа (если при этом ни один из операндов не имеет неопределеного (?) значения).

Остаток от целочисленного деления (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 оператора его значение преобразуется в значение типа INTEGER, то есть дробная часть теряется. Для любых целочисленных значений **a** и **b** всегда выполняется равенство **(a DIV b)*b + (a MOD b) = a**. Абсолютное значение **a MOD b** всегда должно быть меньше, чем абсолютное значение **b**, а знак **a MOD b** должен быть таким же, как знак **b**.

Если хотя бы один 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*, используя следующий алгоритм (точность *p* задается явно для типа REAL или принимается точность, установленная для реализации в соответствии с приложением E):

a) преобразовать число в экспоненциальную форму без предшествующих нулей;

b) текущий указатель цифры *k* устанавливается на *p* позиций правее десятичной точки;

c) если действительное значение положительно, выполняются следующие действия:

- если цифра в позиции *k* лежит в диапазоне 5..9, добавить 1 к цифре в позиции *k—1*, цифра в позиции *k* и все последующие цифры игнорируются. Перейти к шагу e;
- если цифра в позиции *k* лежит в диапазоне 0..4, цифра в позиции *k* и все последующие цифры игнорируются. Перейти к шагу h;

d) если действительное значение отрицательно, выполняются следующие действия:

- если цифра в позиции *k* лежит в диапазоне 6..9, добавить 1 к цифре в позиции *k—1*, цифра в позиции *k* и все последующие цифры игнорируются. Перейти к шагу e;
- если цифра в позиции *k* лежит в диапазоне 0..5, цифра в позиции *k* и все последующие цифры игнорируются. Перейти к шагу h;

e) установить значение указателя *k* в *k—1*;

f) если цифра в позиции *k* лежит в диапазоне 0..9, перейти к шагу *h*;

g) если цифра в позиции *k* имеет значение 10, добавить 1 к цифре в позиции *k—1* и установить цифру в позиции *k* в 0. Перейти к шагу e;

h) число теперь округлено.

Примечание — В результате действия такого механизма округления 0,5 округляется до 1, а −0,5 — до 0.

Пример 83 — Данный пример показывает влияние установки количества значащих цифр в дробной части действительного числа, то есть его точность.

```
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-xl)**2 + (y2-yl)**2 + (z2-zl)**2);
```

Вычисленное значение **distance** равно **1.9104973... e+1,** но, поскольку для этой переменной задана точность только шесть значащих цифр, значение **distance** будет равно **1.91050e+1.**

12.2 Операторы отношений

К операторам отношений относятся следующие операторы: сравнения значений, сравнения экземпляров, принадлежности (IN) и сравнения строк (LIKE). Результатом выполнения операто-

ра 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 (<=).

Эти операторы могут применяться к числовым, логическим, строковым, двоичным операндам и к операндам, являющимся элементами перечисления. Помимо этого операторы = и <> могут применяться к сборным (агрегатным) типам данных и к типам данных объекта. Два сравниваемых операнда оператора сравнения значений должны иметь совместимые типы данных (см. 12.11).

Для двух заданных величин **a** и **b**, независимо от их типа, выражение **a <> b** эквивалентно **NOT (a = b)** для всех типов данных. Также для величин **a** и **b**, не являющихся ни сборными (агрегатными) типами данных, ни типами данных объекта, дополнительно:

a) одно из следующих утверждений имеет значение TRUE: **a < b**, **a = b** или **a > b**;

b) **a <= b** эквивалентно **(a < b) OR (a = b);**

c) **a >= b** эквивалентно **(a > b) OR (a = b).**

12.2.1.1 *Сравнение числовых величин*

Операторы сравнения значений, при их применении к числовым операндам, должны соответствовать математическому упорядочению действительных чисел.

Примечание — При сравнении двух действительных чисел их точность не учитывается.

Пример 84 — Дано:

```
a : REAL(3) := 1.23
b : REAL(5) := 1.2300;
```

выражение **a = b** имеет значение TRUE (истина).

12.2.1.2 *Сравнение двоичных величин*

Для сравнения двух двоичных величин, сравнивают биты в соответствующих позициях каждой величины, начиная с первой пары соответствующих битов (в самой левой позиции), затем, в следующей позиции и т.д. до тех пор, пока не будет встречена пара битов, значения которых в сравниваемых величинах различаются. Когда встречена пара несовпадающих битов, меньшей считается та величина, у которой в соответствующей позиции находится бит с нулевым значением. После этого никакого дополнительного сравнения не требуется. Если несовпадающей пары битов не встречено, меньшей считается более короткая величина (длина определяется с помощью функции BLENGTH). Если у сравниваемых величин равны длины и не встречено ни одной пары несовпадающих битов, сравниваемые величины равны.

12.2.1.3 *Сравнение логических величин*

При сравнении двух значений типа LOGICAL (или BOOLEAN) необходимо соблюдать следующий порядок:

FALSE < UNKNOWN < TRUE.

12.2.1.4 *Сравнение строковых значений*

Для сравнения двух строковых значений сравнивают символы в соответствующих позициях каждого значения, начиная с первой пары соответствующих символов (в самой левой позиции), затем в следующей позиции и т.д. до тех пор, пока не будет встречена пара символов, значения которых в сравниваемых величинах различаются или пока не будут проверены все пары символов. Когда встречена пара несовпадающих символов, меньшей считается та символьная строка, у которой в соответствующей позиции находится символ с меньшим значением кода (как определено значениями октетов для символов по ИСО/МЭК 10646-1). После этого никакого дополнительного сравнения не требуется. Если несовпадающей пары символов не встречено, меньшей считается более короткая символьная строка (длина определяется с помощью функции

LENGTH). Если у сравниваемых символьных строк длины равны и не встречено ни одной пары несовпадающих символов, то два сравниваемых строковых значения считаются равными.

12.2.1.5 *Сравнение элементов перечисления*

Сравнение значений элементов перечисления основывано на том, в какой последовательности элементы перечислены в объявлении перечисляемого типа данных. См. правило (a) в 8.4.1.

12.2.1.6 *Сравнение агрегатных значений*

Операторами сравнения значений, установленными для агрегатных значений, являются: равно (=) и не равно (<>). Два агрегатных значения могут быть сравнимы только в том случае, если их типы данных совместимы (см. 12.11).

Все агрегатные сравнения должны проверять число элементов в каждом операнде: если **SIZEOF (a) <> SIZEOF (b),** то агрегаты не равны. При агрегатных сравнениях сравнивают элементы агрегатного значения, используя сравнения значений. Если любое сравнение элементов принимает значение FALSE, то агрегатное сравнение принимает значение FALSE. Если одно или несколько сравнений элементов для конкретного агрегатного сравнения принимает значение UNKNOWN, а все остальные сравнения принимают значения TRUE, то агрегатное сравнение принимает значение UNKNOWN. Во всех других случаях агрегатное сравнение принимает значение TRUE.

Определение равенства сравниваемых агрегатов зависит от их агрегатных типов данных:

- два массива **a** и **b** равны тогда и только тогда, когда каждый элемент массива **a** равен по значению элементу массива **b**, расположенному 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 каждый элемент списка **a** равен по значению элементу списка **b**, расположенному в той же позиции;
- два мультимножества (BAG) или набора (SET) равны тогда и только тогда, когда каждый элемент **VALUE_IN a** появляется в **VALUE_IN b** равное число раз, а каждый элемент **VALUE_IN b** появляется в **VALUE_IN a** равное число раз.

Примечание — Мультимножество может быть равно по значению набору, даже если мультимножество содержит повторяющиеся элементы. Это возможно потому, что проверка основывается на определении количества равных по значению элементов.

12.2.1.7 *Сравнение значений объектов*

Два экземпляра объектов равны при сравнении значений, если равны значения их соответствующих атрибутов. Поскольку экземпляры объектов могут иметь атрибуты, представленные типами данных объекта, возможно наличие экземпляров со ссылками на самих себя, в этом случае экземпляры объектов равны при сравнении значений, если все атрибуты, представленные простыми типами данных, имеют одинаковые значения и те же атрибуты в обоих экземплярах объектов являются ссылающимися на сам экземпляр.

Более точно, предположим, что сравниваются два экземпляра **l** и **r**. Если **l :=: r**, то **l = r**. Если это не так, то введем следующие описания:

- определим упорядочение на совокупности рассматриваемых экземпляров. На практике эта совокупность конечна, поэтому такое упорядочивание выполнимо;
- для целей данного рассуждения определим оператор индексирования агрегатов, который учитывает данное упорядочение таким образом, что для любого агрегата **agg** и для любых индексов **i** и **j** условие **i < j** эквивалентно условию **agg[i] < agg[j]**;
- определим последовательность ссылок как одну или более ссылок атрибутов или индексов. Применим последовательность ссылок **s** к экземпляру **i,** записав **s(i).** Тогда **s(i)** может быть вычислена, если ни одна из ссылок, за исключением последней, не принимает неопределенного (?) значения.

Тогда значение выражения l = r определяется исходя из следующих условий:

a) Если **TYPEOF(l) <> TYPEOF(r)**, то **l = r** имеет значение FALSE.

b) Если существует такая последовательность ссылок **s**, что вычислим строго один **s(l)** и **s(r),** то **l = r** имеет значение FALSE.

c) Если существует такая последовательность ссылок **s**, что и **s(l)** и **s(r)** выдают значения простых типов, и если **s(l) <> s(r)**, то **l = r** имеет значение FALSE.

d) Если существует такая последовательность ссылок **s**, что **NOT EXISTS(s(l))** или **NOT EXISTS(s(r)),** то **l = r** имеет значение UNKNOWN.

e) Иначе **l = r** имеет значение TRUE.

Пример 85 — Следующий алгоритм является одной из реализаций вышеописанного теста сравнения значений. Этот алгоритм приводится только в качестве иллюстрации и не является обязательной частью какой-либо реализации.

Допустим, что **l** и **r** являются переменными типа GENERIC в данном алгоритме.

a) Инициализируем **l** как левый экземпляр объекта, а **r** как правый экземпляр объекта.

b) Если **l** и **r** — один и тот же экземпляр, то есть **l :=: r**, выражение имеет значение TRUE.

c) Инициализируем пустой список **plist**, который будет содержать упорядоченные пары идентификаторов экземпляров объектов.

Примечание 1 — Представление идентификаторов экземпляра зависит от особенностей реализации.

d) Сравним **l** и **r**, используя глубокий алгоритм равенства, описанный ниже.

e) Выражение будет иметь значение, возвращаемое глубоким алгоритмом равенства.

Глубокий алгоритм равенства

a) Если **l, r** или обе переменные 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 объектов, то алгоритм возвращает значение **l = r**, используя соответствующий тест на равенство.

d) Если **l** и **r** являются одним и тем же экземпляром объекта, то есть **l :=: r**, то алгоритм возвращает значение TRUE.

e) Если пара экземпляров **(l, r)** присутствует в **plist**, алгоритм возвращает значение TRUE.

f) Если пара экземпляров **(l, r)** не присутствует в **plist**, выполняют следующие действия:

1) Добавляют к **plist** пару **(l, r).**

2) Для каждого из атрибутов **a**, определенных для **l** и **r**, сравнивают **l.a** и **r.a**, используя глубокий алгоритм равенства, предположив, что **l = l.a** и **r = r.a.**

Примечание 2 — Это рекурсивный вызов.

3) Если на шаге (f2) глубокий алгоритм равенства для какого-либо из атрибутов возвращает значение FALSE, общий результат будет иметь значение FALSE. Иначе, если алгоритм возвращает для какого-либо из атрибутов значение UNKNOWN, общий результат будет иметь значение UNKNOWN. Иначе, общий результат будет иметь значение TRUE.

Примечание 3 — Если хотя бы один из результатов сравнения имеет значение FALSE, общий результат имеет значение FALSE. Если все результаты имеют значение TRUE, общий результат имеет значение TRUE. Когда какое-либо сравнение имеет результат UNKNOWN, а все другие результаты имеют значение TRUE, общий результат имеет значение UNKNOWN.

Пример 86 — Локальные переменные **i1** и **i2** имеют тип **loop_of_integer** и при присвоении им значений в этом примере переменные 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 -- присваивается значение «ложь»
```

Сравнение значений объектов может быть применено для экземпляров объектов и для сгруппированных экземпляров объектов (см. 12.7.4). При сравнении экземпляров объектов сравниваются все атрибуты супертипов и подтипов сравниваемых экземпляров. При сравнении сгруппированных экземпляров объектов сравниваются только те атрибуты, которые объявлены атрибутами в объявлении объекта для типа данных объекта, поименованного в квалификаторе

группы (при этом не включаются наследуемые атрибуты, которые переопределены в конкретном типе данных объекта).

12.2.2 Операторы сравнения экземпляров

Операторами сравнения экземпляров являются:

- равенство экземпляров (**:=:**);
- неравенство экземпляров (**:<>:**).

Эти операторы применимы к операндам числового, логического, двоичного, строкового, перечисляемого, агрегатного типов данных и к операндам типа данных объекта. Оба операнда в операторе сравнения экземпляра должны быть совместимы по типу данных (см. 12.11).

Для двух операндов **a** и **b** выражение **(a :<>: b)** эквивалентно выражению **NOT (a :=: b)** для всех типов данных.

Операторы сравнения экземпляров, применяемые к числовым, логическим, строковым, двоичным и перечисляемым типам данных, эквивалентны соответствующим операторам сравнения значений. Это означает, что выражение **(a :=: b)** эквивалентно **(a = b)**, а выражение **(a :<>: b)** эквивалентно **(a <> b)** для этих типов данных.

12.2.2.1 *Сравнение экземпляров агрегатов*

Операторами сравнения экземпляров, определенными для агрегатных значений, являются операторы равенства **(:=:)** и неравенства **(:<>:)**. Два агрегатных значения могут сравниваться только в том случае, если совместимы их типы данных (см. 12.11).

Все агрегатные сравнения должны проверять число элементов в каждом из операндов; если **SIZEOF(a)** <> **SIZEOF(b)**, агрегаты не равны. При агрегатном сравнении проводится сравнение элементов агрегатного значения с использованием сравнений экземпляров. Если при сравнении любого элемента получено значение FALSE, то результат агрегатного сравнения также будет иметь значение FALSE. Если одно или несколько сравнений элементов при конкретном агрегатном сравнении дают результат UNKNOWN, а все прочие сравнения дают результат TRUE, то результат агрегатного сравнения будет иметь значение UNKNOWN. Во всех других случаях агрегатное сравнение принимает значение TRUE.

Определение равенства экземпляров агрегатов зависит от типа сравниваемых агрегатных типов данных:

- два массива **a** и **b** равны тогда и только тогда, когда каждый элемент массива **a** является тем же экземпляром, что и элемент массива **b**, находящийся в той же позиции, то есть **a[i] :=: b[i]** (12.6.1);
- два списка **a** и **b** равны тогда и только тогда, когда каждый элемент списка **a** является тем же экземпляром, что и соответствующий элемент списка **b**;
- экземпляры двух мультимножеств (BAG) **a** и **b** равны тогда и только тогда, когда каждый элемент **IN a** появляется в **IN b** ровно столько же раз, а каждый элемент **IN b** появляется в **IN a** ровно столько же раз;
- два набора (SET) **a** и **b** равны тогда и только тогда, когда каждый элемент **a** присутствует в **b** и каждый элемент **b** присутствует в **a**;
- экземпляр набора (SET) равен экземпляру мультимножества (BAG) тогда и только тогда, когда каждый элемент набора присутствует в мультимножестве один раз, а в мультимножестве нет ни одного элемента, который не входил бы в набор.

Пример 87 — Сравнение экземпляров двух массивов

```
LOCAL
    al, a2 : ARRAY [1:10] OF b;
END_LOCAL;
...
IF (al :=: a2) THEN ...
```

12.2.2.2 *Сравнение экземпляров объектов*

Операторы равенства **(:=:)** и неравенства **(:<>:)** экземпляра объекта получают два совместимых экземпляра объектов и вычисляют значение типа LOGICAL.

Выражение **a :=: b** имеет значение TRUE, если **a** является тем же экземпляром объекта, что и **b**, то есть имеют один и тот же идентификатор, зависящий от реализации. Выражение имеет значение FALSE, если экземпляр **a** отличается от экземпляра **b**. Выражение имеет значение UNKNOWN, если хотя бы один из операндов имеет неопределенное (?) значение.

Помимо прочего, сравнение экземпляров объектов должно проводиться при сравнении двух экземпляров объектов, таких как сравнения агрегатов, и при проверке правил уникальности UNIQUE.

Пример 88 — Все дети имеют матерей, но некоторые дети имеют также 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
          -- удостоверимся, что данный экземпляр объекта
          -- не является одним из братьев или сестер
not_identical : SIZEOF ( QUERY ( i <* siblings | i :=: SELF ) ) = 0;
          -- удостоверимся, что каждый из братьев или сестер
          -- имеет общего отца или мать с данным экземпляром объекта
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 некоторому агрегату и вычисляет значение типа LOGICAL. Правый операнд должен быть значением агрегатного типа данных, а левый операнд должен быть совместим с основным типом данного агрегатного значения. Выражение **e IN agg** вычисляется следующим образом:

a) если хотя бы один из операндов имеет неопределенное (?) значение, выражение имеет значение UNKNOWN;

b) если существует элемент **agg[i]** такой, что **e :=: agg[i],** выражение имеет значение TRUE;

c) если существует хотя бы один элемент **agg[i],** имеющий неопределенное (?) значение, выражение имеет значение UNKNOWN;

d) иначе выражение будет иметь значение FALSE.

Примечание — Для того чтобы проверить, существует ли элемент агрегата, имеющий заданное значение, может быть использована функция **VALUE_IN** (см. 15.28).

Принадлежность к определенной моделирующей системе может быть проверена с помощью пары функций, называемых, например, **my_equal** (см. 8.2.5) и **my_in**, как это показано в нижеприведенном псевдокоде:

```
FUNCTION my_in(c:AGGREGATE OF GENERIC:gen; v:GENERIC:gen) : LOGICAL;
    (* Если v или c имеет неопределенное (?) значение, возвращается значение UNKNOWN.
       Иначе, если какой-либо из элементов агрегата c имеет значение v, возвращается значение
       TRUE. Иначе, если хотя бы одно из сравнений значения v со значениями элементов агре-
       гата дает результат UNKNOWN, возвращается значение UNKNOWN. Иначе возвращается
       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;
```

Например, это может быть использовано следующим образом:

```
LOCAL
      v : a;
      c : SET OF a;
END_LOCAL;

...
IF my_in(c, v) THEN ...
```

12.2.4 Выражения интервалов

Выражение интервала проверяет, лежит ли значение внутри заданного интервала. Выражение содержит три операнда, которые должны быть совместимыми (см. 12.11). Типы операндов должны быть упорядоченными, то есть простые типы (см. 8.1) и определенные типы данных, исходными типами которых являются или простые, или перечисляемые типы.

Синтаксис:

```
229 interval = `{` interval_low interval_op interval_item interval_op
                  interval_high `}` .
232 interval_low = simple_expression .
233 interval_op = `<` | `<=` .
231 interval_item = simple_expression .
230 interval_high = simple_expression .
```

Примечание — Выражение интервала:
{ interval_low interval_op interval_item interval_op interval_high }
семантически эквивалентно следующему выражению:
(interval_low interval_op interval_item) AND
(interval_item interval_op interval_high)

Считаем, что во втором выражении **interval_item** вычисляется только один раз.

Выражение интервала вычисляется для типа LOGICAL и принимает значение TRUE, если обе операции сравнения дают результат TRUE. Выражение принимает значение FALSE, если хотя бы одна из операций сравнения дает результат FALSE, и значение UNKNOWN, если хотя бы один из операндов имеет неопределенное (?) значение.

Пример 89 — Здесь проверяется, имеет ли **b** значение, большее 5.0 и меньшее или равное 100.0

```
LOCAL
      b : REAL := 20.0;
END_LOCAL;
...
IF { 5.0 < b <= 100.0 } THEN -- имеет значение TRUE
...
```

12.2.5 Оператор подобия

Оператор подобия (LIKE) сравнивает два строковых значения, используя описанный ниже алгоритм сравнения с образцом, формирующий значение типа LOGICAL. Левый операнд является сравниваемой строкой. Правый операнд является эталонной строкой.

Алгоритм сравнения с образцом определяется следующим образом. Каждый символ эталонной строки сравнивается с соответствующим символом(и) сравниваемой строки. Если любая пара сравниваемых символов не совпадает, то сравнение является ошибочным и выражению присваивается значение FALSE.

Некоторые специальные символы в эталонной строке могут совпадать с несколькими символами в сравниваемой строке. Эти символы приведены в таблице 11. Все сравниваемые символы должны быть идентичными или совпадать с приведенными в таблице 11, чтобы выражению было присвоено значение TRUE. Если любой из операндов имеет неопределенное (?) значение, то выражению присваивается значение UNKNOWN.

Когда символ сравниваемой строки должен совпадать с каким-либо специальным символом эталонной строки, то образец должен содержать эталонную управляющую последовательность. Эталонная управляющая последовательность должна содержать управляющий символ (\), за которым следует сопоставляемый специальный символ.

Пример 90 — Для сопоставления с символом @ используется управляющая последовательность \@.

Следующие примеры показывают, как сравниваются строки.

Примеры

91 — Если **a := '\AAAA';** следующие сравнения дают результаты:

```
a LIKE '\\AAAA'      --> TRUE
a LIKE '\AAAA'       --> FALSE
a LIKE '\\A?AA'      --> TRUE
a LIKE '\\!\\AAA'    --> TRUE
a LIKE '\\&'         --> TRUE
a LIKE '\$'          --> FALSE
```

92 — Если **a := 'The quick red fox ';** следующее сравнение дает результат:

```
a LIKE '$$$$'        --> TRUE
```

93 — Если **a := 'Page 407';** следующее сравнение дает результат:

```
a LIKE '$*'          --> TRUE
```

Таблица 11 — Эталонные сравниваемые символы

Символ	Значение
@	Сравнение любой буквы
^	Сравнение любой заглавной буквы (верхнего регистра)
?	Сравнение любого символа
&	Сравнение остатка строки
#	Сравнение любой цифры
$	Сравнение любой подстроки, завершаемой символом пробела или конца строки
*	Сравнение любого числа символов
\	Начало эталонной управляющей последовательности
!	Отрицание символа (используется с другими символами)

12.3 Двоичные операторы

Кроме операторов отношения, определенных в 12.2.1.2, для типа данных BINARY установлены: оператор индексирования ([ ]) и оператор конкатенации (+).

12.3.1 Двоичное индексирование

Оператор двоичного индексирования требует двух операндов: индексируемого двоичного значения и спецификации индекса, а выдает двоичное значение длины (**index_2 − index_1 + 1**). Полученное двоичное значение равно последовательности битов от позиции **index_1** до **index_2** включительно. Если требуется двоичное значение единичной длины, необходимо указать только **index_1**. Значение индекса, равное единице, соответствует самому левому биту.

Синтаксис:
226 index_qualifier = `[` index_1 [`:` index_2] `]` .
224 index_1 = index .
223 index = nunieric_expression .
225 index_2 = index .

Правила и ограничения

a) Значение **index_1** должно быть положительным целым числом.

b) Должно выполняться условие **1 < index_1 < BLENGTH** (двоичное значение), иначе будет возвращено неопределенное (?) значение.

c) Значение **index_2,** при его указании, должно быть положительным целым числом.

d) Должно выполняться условие **index1 ≤ index_2 ≤ BLENGTH** (двоичное значение), иначе будет возвращено неопределенное (?) значение.

Пример 94 — Должен быть проверен четвертый бит двоичного числа **image**.

```
image := % 01010101

IF image[4]=%1 THEN ... -- имеет значение TRUE
IF image[4:4]=%1 THEN ... -- эквивалентное выражение
```

Пример 95 — Должны быть проверены биты с четвертого по десятый двоичного числа **image**.

```
IF image[4:10]= %1011110 THEN ...
```

12.3.2 Оператор двоичной конкатенации

Оператор двоичной конкатенации (+) является двоичным оператором, который объединяет вместе два двоичных значения. Оба операнда должны иметь двоичные значения, а выражению должно быть присвоено двоичное значение, содержащее конкатенацию двух операндов, при этом содержимое первого операнда находится слева.

Пример 96 — Двоичные значения могут быть объеденены следующим образом:

```
image := %101000101 + %101001;
(* переменная image теперь содержит двоичное значение %101000101101001 *)
```

12.4 Логические операторы

Логическими операторами являются: NOT (отрицание), AND (и), OR (или) и XOR (исключающее или). Каждый из операторов выдает результат типа LOGICAL. Операторам AND, OR и XOR требуются два логических операнда, а оператору NOT — один логический операнд.

12.4.1 Оператор NOT

Оператор NOT требует одного логического операнда (операнд располагается справа от оператора NOT) и формирует результат в виде логического значения, как показано в таблице 12.

Таблица 12 — Оператор NOT

Значение операнда	Значение результата
TRUE	FALSE
UNKNOWN	UNKNOWN
FALSE	TRUE

12.4.2 Оператор AND

Оператор AND требует двух логических операндов и формирует результат в виде логического значения, как показано в таблице 13. Оператор AND является коммутативным.

Таблица 13 — Оператор AND

Значение операнда 1	Значение операнда 2	Значение результата
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

Оператор OR требует двух логических операндов и формирует результат в виде логического значения, как показано в таблице 14. Оператор OR является коммутативным.

Таблица 14 — Оператор OR

Значение операнда 1	Значение операнда 2	Значение результата
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

Оператор XOR требует двух логических операндов и формирует результат в виде логического значения, как показано в таблице 15. Оператор XOR является коммутативным.

Таблица 15 — Оператор XOR

Значение операнда 1	Значение операнда 2	Значение результата
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

Кроме операторов отношения, определенных в 12.2.1.4 и 12.2.5, для строкового типа STRING установлены: оператор индексирования ([ ]) и оператор конкатенации (+).

12.5.1 Индексирование строк

Оператор индексирования строк требует двух операндов: индексируемого строкового значения и спецификация индекса, а выдает строковое значение длины (**index_2 − index_1 + 1**). Полученное строковое значение равно последовательности символов от позиции **index_1** до **index_2** включительно. Если требуется строковое значение единичной длины, необходимо указать только **index_1**. Значение индекса, равное единице, соответствует самому левому символу.

Синтаксис:
226 index_qualifier = `[` index_1 [`:` index_2] `]` .
224 index_1 = index .
223 index = numeric_expression .
225 index_2 = index .

Правила и ограничения

a) Значение **index_1** должно быть положительным целым числом.

b) Должно выполняться условие **1 ≤ index_1 ≤ LENGTH** (строковое значение), иначе будет возвращено неопределенное (?) значение.

c) Значение **index_2**, при его наличии, должно быть положительным целым числом.

d) Должно выполняться условие **index1 ≤ index_2 ≤ LENGTH** (строковое значение), иначе будет возвращено неопределенное (?) значение.

Пример 97 — Проверка символа, находящегося в седьмой позиции строки **name**

IF name[7]="00125FEl" THEN... -- представление в соответствии с ИСО/МЭК 10646-1

IF name[7:7]="00125FEl" THEN ... -- эквивалентное выражение

Пример 98 — Проверка символов, находящихся в позициях с седьмой по десятую строки **name**

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 строковые значения, а выражению должно быть присвоено строковое значение, содержащее конкатенацию двух операндов, при этом содержимое первого операнда находится слева.

Пример 99 — Две строки могут быть объединены следующим образом:

name := `ABC` + ` ` + `DEF` ;

(* **name** теперь содержит строку 'ABC DEF' *)

12.6 Агрегатные операторы

Агрегатными операторами являются: оператор индексирования (**[ ]**), оператор пересечения (*), оператор объединения (+), оператор разности (-), оператор подмножества (<=), оператор надмножества (>=) и оператор запроса (QUERY). Эти операторы определены в следующих пунктах. Ко всем агрегатным значениям применимы также определенные в 12.2 оператор равенства (=), оператор неравенства (<>), оператор равенства экземпляров (:=:), оператор неравенства экземпляров (:<>:) и оператор принадлежности IN.

Примечание — Некоторые операции, производимые над агрегатными значениями, требуют неявного сравнения элементов агрегатных значений, во всех таких случаях используется сравнение экземпляров.

12.6.1 Индексирование агрегатов

Оператор индексирования агрегатов требует двух операндов: индексируемого агрегатного значения и спецификации индекса, а выдает единственный элемент из агрегатного значения. В качестве типа данных этого элемента выбирается основной тип данных индексируемого агрегатного значения.

Синтаксис:
226 index_qualifier = `[` index_1 [`:` index_2] `]` .
224 index_1 = index .
223 index = numeric_expression .
225 index_2 = index .

Правила и ограничения

a) Значение **index_2** не должено присутствовать, поскольку из агрегатного значения может быть индексирован только один элемент.

b) Значение **index_1** должно быть целым числом

c) Должно выполняться условие: **LOINDEX** (агрегатное значение) ≤ **index_1** ≤ **HIINDEX** (агрегатное значение), иначе будет возвращено неопределенное (?) значение.

d) Если типом агрегатного значения является ARRAY или LIST, выражение выдает элемент агрегатного значения, находящийся в позиции, указанной **index_1**.

e) Если типом агрегатного значения является BAG или SET, то для каждого **index_1** в диапазоне от **LOINDEX** (агрегатное значение) до **HIINDEX** (агрегатное значение) выражение должно выдавать разные элементы агрегатного значения.

f) При повторном индексировании того же агрегатного значения с тем же самым значением **index_1** должен возвращаться тот же элемент только в том случае, если между обращениями агрегатное значение не было изменено. Если агрегатное значение было изменено для агрегатных типов данных BAG или SET, то результат повторного агрегатного индексирования измененного агрегатного значения непредсказуем.

Пример 100 — Индексирование для мультимножеств (bags) и наборов (sets) может быть использовано для повторного извлечения всех значений из данного агрегатного значения.

```
a_set : SET [1:20] OF INTEGER;
result : INTEGER;

result := 1;
REPEAT FOR index := LOINDEX(a_set) TO HIINDEX(a_set);
     result := result * a_set[index];
END_REPEAT;
```

После выхода из оператора REPEAT переменная **result** будет содержать произведение всех целочисленных элементов агрегата **a_set.**

12.6.2 Оператор пересечения

Оператор пересечения (*) принимает два операнда агрегатного значения и выдает агрегатное значение. Допустимые типы операндов и соответствующие им типы результата приведены в таблице 16. Результирующее агрегатное значение неявно объявляется как агрегат типа, определенного в таблице 16 с границами [0..?]. Основные типы данных операндов должны быть совместимыми (см. 12.11). Если пересечение двух операндов не содержит элементов, размер результирующего агрегатного значения должен быть нулевым.

Таблица 16 — Оператор пересечения — типы операндов и результата

Первый операнд	Второй операнд	Результат
Bag (мультимножество)	**Bag** (мультимножество)	**Bag** (мультимножество)
Bag (мультимножество)	**Set** (набор)	**Set** (набор)
Set (набор)	**Set** (набор)	**Set** (набор)
Set (набор)	**Bag** (мультимножество)	**Set** (набор)

Если одним из операндов является набор (set), то результатом должен быть набор, содержащий каждый элемент, появляющийся одновременно в обоих операндах.

Если оба операнда являются мультимножествами (bags), и конкретный элемент **e** появляется в одном мультимножестве **m** раз, а в другом мультимножестве — **n** раз (где **m** меньше или равно **n**), то результат должен содержать элемент **e m** раз.

12.6.3 Оператор объединения

Оператор объединения (+) принимает два операнда, один из которых должен иметь агрегатное значение, и выдает агрегатное значение. Допустимые типы операндов и соответствующий им тип результата приведены в таблице 17.

Таблица 17 — Оператор объединения — типы операндов и результата

Первый операнд	Второй операнд	Результат
Bag (мультимножество)	**Bag** (мультимножество)	**Bag** (мультимножество)
Bag (мультимножество)	**Element** (элемент)	**Bag** (мультимножество)
Element (элемент)	**Bag** (мультимножество)	**Bag** (мультимножество)
Bag (мультимножество)	**Set** (набор)	**Bag** (мультимножество)
Bag (мультимножество)	**List** (список)	**Bag** (мультимножество)
Set (набор)	**Set** (набор)	**Set** (набор)
Set (набор)	**Element** (элемент)	**Set** (набор)
Element (элемент)	**Set** (набор)	**Set** (набор)
Set (набор)	**Bag** (мультимножество)	**Set** (набор)
Set (набор)	**List** (список)	**Set** (набор)
List (список)	**List** (список)	**List** (список)[1]
Element (элемент)	**List** (список)	**List** (список)[2]
List (список)	**Element** (элемент)	**List** (список)[3]

Примечания

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.

Операция объединения определяется типом операндов и их порядком следующим образом:

Если тип одного из операндов (**E**) совместим с основным типом другого операнда (**A**), операнд **E** добавляется к **A** следующим образом:

- если **A** является значением набора (set), результирующим набором является **A**, к которому добавлен **E**, если только **E** уже не присутствовал в **A**;
- если **A** является значением списка (list), результирующим списком является **A** с вставленным в позицию 1 **E**, если **E** был левым операндом, и в позицию SIZEOF(A+1), если **E** был правым операндом.

Примечание 1 — Результирующий список может содержать повторяющиеся элементы, даже если операнд списка был объявлен как LIST OF UNIQUE;

- если **A** имеет значение мультимножества (bag), то результирующим мультимножеством является **A** с добавленным к нему **E**.

Если оба операнда являются совместимыми списками (list), то результирующим списком является левый операнд с добавленным в его конец правым операндом.

Примечание 2 — Результирующий список может содержать повторяющиеся элементы, даже если оба операнда были объявлены как LIST OF UNIQUE.

Если левый операнд имеет значение набора (set), то результирующий набор создается из левого операнда с добавленными к нему теми элементами правого операнда, которые в левом операнде отсутствуют.

Если левый операнд имеет значение мультимножества (bag), то результатом является левый операнд с добавлением к нему всех элементов правого операнда.

12.6.4 Оператор разности

Оператор разности (-) принимает два операнда, левый из которых должен быть агрегатным значением, и выдает агрегатное значение. Допустимые типы операндов и соответствующий им тип результата приведены в таблице 18. Результирующее агрегатное значение содержит элементы первого операнда за исключением соответствующих им элементов второго операнда, то есть каждый элемент второго операнда, который входит в первый операнд, исключается из первого операнда. Результирующее агрегатное значение неявно объявляется как агрегат, тип которого определяется в соответствии с таблицей 18 с границами [0..?]. Основные типы операндов должны быть совместимы (см. 12.11). Тип данных возвращаемого агрегатного значения должен быть та-

ким же, как тип первого операнда. Если оба операнда являются мультимножествами (bags), и конкретный элемент **e** присутствует **m** раз в первом операнде и **n** раз во втором операнде, то результат должен содержать **e m—n** раз, если **m** больше **n,** и не содержать **e**, если **m** меньше или равно **n**. Если второй операнд содержит элементы, не присутствующие в первом операнде, такие элементы будут игнорированы и не войдут в результирующее агрегатное значение.

Таблица 18 — Оператор разности — типы операндов и результата

Первый операнд	Второй операнд	Результат
Bag (мультимножество)	**Bag** (мультимножество)	**Bag** (мультимножество)
Bag (мультимножество)	**Set** (набор)	**Bag** (мультимножество)
Bag (мультимножество)	**Element** (элемент)	**Bag** (мультимножество)
Set (набор)	**Set** (набор)	**Set** (набор)
Set (набор)	**Bag** (мультимножество)	**Set** (набор)
Set (набор)	**Element** (элемент)	**Set** (набор)

Пример 101 — Если **A** является мультимножеством целых чисел **[1, 2, 1, 3,]**, то **A — 1** имеет значение **[1,2,3],** которое эквивалентно **[2,1,3].**

12.6.5 Оператор подмножества

Оператор подмножества (subset) (<=) принимает два операнда, определенных в таблице 19, и выдает результат типа LOGICAL. Выражение принимает значение TRUE тогда и только тогда, когда любой элемент **e**, присутствующий **n** раз в первом операнде, присутствует не менее **n** раз во втором операнде. Выражение принимает значение UNKNOWN, когда хотя бы один из операндов имеет неопределенное (?) значение, а иначе принимает значение FALSE.

Типы операндов должны быть совместимы (см. 12.11).

Таблица 19 — Операторы подмножества и надмножества — типы операндов

Первый операнд	Второй операнд
Bag (мультимножество)	**Bag** (мультимножество)
Bag (мультимножество)	**Set** (набор)
Set (набор)	**Bag** (мультимножество)
Set (набор)	**Set** (набор)

12.6.6 Оператор надмножества

Оператор надмножества (superset) (>=) принимает два операнда, определенных в таблице 19, и выдает результат типа LOGICAL. Выражение принимает значение TRUE тогда и только тогда, когда любой элемент **e**, присутствующий **n** раз во втором операнде, присутствует не менее **n** раз в первом операнде. Выражение принимает значение UNKNOWN, когда хотя бы один из операндов имеет неопределенное (?) значение, а иначе принимает значение FALSE.

Типы операндов должны быть совместимы (см. 12.11).

Выражение **b >= a** должно быть точно эквивалентным **a <= b.**

12.6.7 Выражение запроса

Выражение запроса QUERY применяет **logical_expression** отдельно к каждому элементу агрегатного значения и выдает агрегатное значение, содержащее элементы, для которых **logical_expression** получило значение TRUE. Результатом этого является подмножество исходного агрегатного значения, в котором все элементы данного подмножества удовлетворяют условию, представленному логическим выражением.

Синтаксис:

```
264 query_expression = QUERY `(` variable_id `<*` aggregate_source `|`
                              logical_expression `)` .
160 aggregate_source = simple_expression .
241 logical_expression = expression .
```

Правила и ограничения

a) Выражение **variable_id** неявным образом объявляется как переменная внутри области действия выражения запроса.

Примечание — Эта переменная нигде не объявляется и не существует вне выражения запроса.

b) Выражение **aggregate_source** должно быть агрегатным значением (ARRAY, BAG, LIST или SET).

c) Третий операнд (**logical_expression**) должен быть выражением, выдающим результат типа LOGICAL.

Элементы извлекаются поочередно из исходного агрегата и подставляются в логическое выражение **logical_expression** вместо переменной **variable_id**. Затем вычисляется **logical_expression**. Если **logical_expression** принимает значение TRUE, данный элемент добавляется к результату; в других случаях он не добавляется. Эти действия повторяются для каждого элементы из исходного агрегата. Результирующее агрегатное значение пополняется в соответствии с конкретным видом агрегатного типа данных:

Массив (ARRAY). Результирующий массив имеет тот же основной тип и границы, что и исходный массив, но элементы массива объявляются как OPTIONAL. Изначально каждый элемент является неопределенным (?). Любой элемент исходного массива, для которого **logical_expression** выдает значение TRUE, затем помещается на соответствующую индексную позицию в результирующем массиве.

Мультимножество (BAG). Результирующее мультимножество имеет тот же основной тип и верхнюю границу, что и исходное мультимножество. Нижняя граница является нулевой. Изначально результирующее мультимножество является пустым. Любой элемент из исходного мультимножества, для которого **logical_expression** выдает значение TRUE, затем добавляется к результирующему мультимножеству.

Список (LIST). Результирующий список имеет тот же основной тип и верхнюю границу, что и исходный список. Нижняя граница является нулевой. Изначально результирующий список является пустым. Любой элемент из исходного списка, для которого **logical_expression** выдает значение TRUE, затем добавляется в конец результирующего списка. Порядок следования элементов исходного списка сохраняется.

Набор (SET). Результирующий набор имеет тот же основной тип и верхнюю границу, что и исходный набор. Нижняя граница является нулевой. Изначально результирующий набор является пустым. Любой элемент из исходного набора, для которого **logical_expression** выдает значение TRUE, затем добавляется к результирующему набору.

Пример 102 — Предположим, что **colour** является определенным типом, имеющим в качестве исходного типа тип ENUMERATION, который охватывает элементы **pink** и **scarlet**. В примере показано, как извлечь из массива **colour** те цвета, которые имеют значение **pink** или **scarlet**.

```
LOCAL
      colours   : ARRAY OF colour;
      reds      : ARRAY OF OPTIONAL colour;
END_LOCAL;
...
reds := QUERY ( element <* colours | ( element = pink ) OR
                                     ( element = scarlet ) );
...
```

Пример 103 — Это правило использует выражение запроса для проверки всех экземпляров объекта типа **point** (точка). Результирующий набор содержит все экземпляры точек, 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 этом примере показано использование трех неявных объявлений. Первым является переменная **point**, которая неявно объявлена в заголовке правила как набор всех экземпляров **point**. Вторым является переменная **temp**, в которую при вычислении выражения запроса подставляются

элементы агрегатного значения **point**. Третьим является конструктор объекта **point**, который появляется в результате объявления объекта.

12.7 Ссылки

Когда локально используется элемент, видимый в локальной области действия, ссылка на элемент осуществляется по идентификатору, объявленному для этого элемента.

12.7.1 Простые ссылки

Простая ссылка является просто именем (идентификатором), заданным элементу в данной области действия. Таким способом можно делать ссылки на следующие элементы:

- атрибуты внутри объявления объекта*;
- константы *;
- элементы из перечисляемого типа*;
- объекты**;
- функции*;
- локальные переменные внутри тела алгоритма*;
- параметры внутри тела алгоритма*;
- процедуры;
- правила;
- схемы внутри спецификации интерфейса;
- типы.

На элементы, помеченные (*), таким же способом можно ссылаться внутри выражения. На объекты, помеченные (**), можно ссылаться как на конструктор (см. 9.2.5) или как на локальную переменную в глобальном правиле (см. 9.6.).

Пример 104 — Правильные простые ссылки:

```
line        (* тип объекта *)
Circle      (* тип объекта *)
RED         (* элемент перечисления *)
z_depth     (* атрибут *).
```

12.7.2 Ссылки с префиксами

В случае, когда одинаковое имя элемента перечисления объявлено в нескольких определенных типах данных, видимых в одной и той же области действия, (см. раздел 10), имя элемента перечисления должно задаваться с префиксом, являющимся идентификатором определенного типа данных для этого элемента, для того чтобы однозначно идентифицировать этот элемент. Ссылка с префиксом состоит из имени определенного типа данных, за которым следует точка (.), а за ней следует имя элемента перечисления.

Пример 105 — В этом примере показано, как элемент перчисления **red** может быть уникально идентифицирован для использования в переменной **stop_signal**.

```
TYPE traffic_light = ENUMERATION OF (red, amber, green);
EMD_TYPE;

TYPE rainbow = ENUMERATION OF
                    (red, orange, yellow, green, blue, indigo, violet);
END_TYPE;

stop_signal : traffic_light := traific_light.red;
ink_colour : rainbow := blue;
```

12.7.3 Ссылки на атрибуты

Ссылка на атрибут (.) обеспечивает ссылку на единственный атрибут внутри экземпляра объекта. Выражение, находящееся слева от ссылки на атрибут, должно быть экземпляром объекта или значением компонента сложного объекта. Идентификатору атрибута, на который дается ссылка, должна предшествовать точка (.).

Синтаксис:
169 attribute_qualifier = `.` attribute_ref .

Ссылка на атрибут, используемая в выражении, возвращает значение конкретного атрибута в экземпляре объекта или в значении компонента сложного объекта. Если конкретный атрибут

не представлен в экземпляре объекта или в значении компонента сложного объекта, то возвращается неопределенное (?) значение.

Если ссылка на атрибут встречается в левой части оператора присваивания или в качестве модифицируемого (VAR) параметра процедуры, это приводит к ссылке в экземпляр объекта, в котором находится соответствующее значение. Если конкретный атрибут не представлен в экземпляре объекта при выполнении оператора присваивания, то результат непредсказуем.

Пример 106 — Этот 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) || coloured_point (red);
      second : coloured_point := point(1.0, 2.0, 3.0) || coloured_point(red);
      x_coord : REAL;
END_LOCAL;

...
x_coord := first.x; -- значение 1.0
IF first.colour = red THEN     -- colour является неверной ссылкой, так как он не
                               -- является атрибутом объявленного типа first
IF second.colour = red THEN-- TRUE, так как colour является верной ссылкой.
```

12.7.4 Групповые ссылки

Групповая ссылка (\) обеспечивает ссылку на значение компонента сложного объекта внутри экземпляра сложного объекта. Выражение, находящееся слева от групповой ссылки, должно быть экземпляром сложного объекта. Тип данных объекта значения компонента сложного объекта, на который дается ссылка, должен следовать за обратной косой чертой (\).

Синтаксис:
219 group_qualifier = `\` entity_ref .

Групповая ссылка, используемая в выражении, возвращает значение компонента сложного объекта, соответствующее поименованному типу данных объекта внутри экземпляра сложного объекта, на который дается ссылка. Если конкретный тип данных объекта не представлен в экземпляре сложного объекта, на который дается ссылка, то возвращается неопределенное (?) значение. Групповая ссылка может быть уточнена за счет последующей ссылки на атрибут. В этом случае групповая ссылка устанавливает область действия ссылки на атрибут.

Примечание — Это используется тогда, когда тип экземпляра сложного объекта имеет множество атрибутов с одинаковым именем или когда выбираемый тип данных содержит множество объектов с атрибутами, имеющими одинаковое имя.

Правила и ограничения

Групповая ссылка, которая не уточняется ссылкой на атрибут, должна появляться в качестве операнда или в операторе сравнения значения объекта (=) или в конструкторе экземпляра сложного объекта (||).

Примеры

107 — В этом примере показано, как групповая ссылка может использоваться для сравнения значений.

```
ENTITY E1
ABSTRACT SUPERTYPE;
      attrib1 : REAL;
```

```
        attrib2 : REAL;
        attrib3 : REAL;
END_ENTITY;

ENTITY E2
SUBTYPE OF (El);
        attribA : INTEGER;
        attribB : INTEGER;
        attribC : INTEGER;
END_ENTITY;

LOCAL
        a : El;
        b : E2;
END.LOCAL;
        -- построим сложные экземпляры 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
        -- через их атрибуты, объявленные в E1
a\El = b\El -- TRUE
(*
   это эквивалентно
   (a.attrib1 = b.attrib1) AND
   (a.attrib2 = b.attrib2) AND
   (a.attrib3 = b.attrib3)
   *)
```

108 — Этот пример показывает, как групповая ссылка может использоваться для указания типа данных конкретного объекта, рассматриваемого для имени атрибута.

```
ENTITY foo1;
        attr : REAL;
END_ENTITY;

ENTITY foo2
        SUBTYPE OF (foo1);
        attr2 : BOOLEAN;
END_ENTITY;

TYPE t = BINARY (80) FIXED;
END_TYPE;

TYPE crazy = SELECT(foo2, t);
END_TYPE;

...
LOCAL
        v : crazy;
END_LOCAL;
...
IF `THIS.F002` IN TYPEOF(v) THEN -- этим обеспечивается отсутствие
                                 -- непредсказуемых результатов («часовой»)
        v\foo1.attr := 1.5;      -- присваивает 1,5 attr атрибута v,
        -- так как attr определен в foo1, для использования foo1 имеется групповая ссылка
END_IF;
```

12.8 Обращение к функции

Обращение к функции вызывает функцию. Обращение к функции состоит из идентификатора и возможного списка фактических параметров, следующих за идентификатором. Число фак-

тических параметров, их тип и порядок следования, должны соответствовать реквизитам формальных параметров, установленных для данной функции. Выражение обращения к функции получает возвращаемое значение при подстановке фактических параметров вместо формальных параметров в объявлении функции.

Вызов функции расширяет пространство экземпляров. Любые экземпляры, созданные в процессе выполнения функции, должны быть уникально обозначены в общей совокупности известных экземпляров. Как правило, созданный экземпляр недоступен вне создающей функции и, в частности, не является частью совокупности экземпляров, доступных для рассмотрения. Исключением является случай, когда экземпляр есть результат исполнения функции или доступен при обращении к ней. В этом случае экземпляр остается доступным в точке обращения. Если экземпляр, появившийся таким образом (то есть как значение вычисляемого атрибута или константы), возвращается на уровень схемы, он становится частью всей исследуемой совокупности.

Синтаксис:
207 function_call = (built_in_function | function_ref) [actual_parameter_list] .
157 actual_parameter_list = `(` parameter { `,` parameter } `)` .
251 parameter = expression .

Правила и ограничения

Передаваемые фактические параметры должны быть совместимы по присвоению с формальными параметрами.

Пример 109 — Пример использования обращения к функции.

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

FUNCTION midpoint_of_line (1: line):point;
...
END_FUNCTION;

IF midpoint_of_line(L506).x = 9.0 THEN ...
                    -- оператор ссылки на атрибут применяется
                    -- непосредственно к результату функции
END_IF;
```

12.9 Инициализатор агрегатов

Инициализатор агрегатов используется для установления значения массива, мультимножества, списка или набора. В квадратных скобках заключены ноль или несколько выражений, присваивающих значения типу данных, совместимому с основным типом данных агрегата. Когда выражение имеет два или более значения, эти значения разделяются запятыми. При инициализации разреженного массива для обозначения пропущенных элементов используется неопределенное (?) значение. Выражение инициализатора агрегата присваивает агрегату значение, содержащее значения конкретных элементов. Число инициализированных элементов должно соответствовать любым границам, заданным для агрегатного типа данных.

Когда используется инициализатор агрегатов, не содержащий ни одного элемента, он создает пустое мультимножество, список или набор (эта конструкция не может быть использована для инициализации пустых массивов).

Синтаксис:
159 aggregate_initializer = `[` [element { `,` element }] `]` .
193 element = expression [`:` repetition] .
273 repetition = numeric_expression .

Пример 110 — Задано объявление:

```
a : SET OF INTEGER;
```

значения могут быть присвоены следующим образом:

```
a := [ 1, 3, 6, 9*8, -12 ] ; -- 9*8 является выражением = 72.
```

Когда ряд последовательных значений является совпадающим, может быть использовано повторение. Оно представляется двумя выражениями, разделенными символом двоеточия (:). Выражение слева от двоеточия является повторяемым значением. Выражение справа от двоеточия, повторитель, указывает, сколько раз должно быть повторено повторяемое значение. Выражение справа от двоеточия вычисляется один раз, перед инициализацией, и должно иметь неотрицательное целочисленное значение.

Пример 111 — Задано следующее объявление:

```
a : BAG OF BOOLEAN ;
```

Следующие два оператора эквивалентны:

```
a : = [ TRUE : 5 ];
a : = [ TRUE, TRUE, TRUE, TRUE, TRUE ];
```

12.10 Оператор конструирования экземпляра сложного объекта

Оператор конструирования экземпляра сложного объекта (||) создает экземпляр сложного объекта посредством комбинирования значений компонентов сложного объекта. Значения компонентов сложного объекта могут комбинироваться в произвольном порядке. Значением выражения оператора конструирования экземпляра сложного объекта является значение компонента сложного объекта или экземпляр сложного объекта. Тип данных компонента сложного объекта может появляться только один раз на данном уровне выражения оператора конструирования экземпляра сложного объекта. Значение компонента сложного объекта может появляться на разных уровнях данного выражения, если они являются вложенными, то есть если значение компонента сложного объекта используется для создания экземпляра сложного объекта, который играет роль атрибута в значении компонента сложного объекта, используемом для создания экземпляра сложного объекта. Более полную информацию об экземплярах сложных объектов смотри в приложении В.

Пример 112 — 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;
```

Далее могут быть созданы 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;                     -- это неверно, так как экземпляр имел бы тип a&b&a&c
```

12.11 Совместимость типов

Операнды в операторе должны быть совместимы с типом(ами) данных, требуемым для оператора. Типы данных обоих операндов некоторого оператора также должны быть совместимы между собой; такие случаи были описаны в данном разделе выше. Типы данных могут быть совместимыми, не будучи идентичными. Типы данных совместимы, когда выполняется одно из следующих условий:

- типы данных являются одинаковыми;

- один тип данных является подтипом или конкретизацией другого (включая определенные типы данных, использующие определенный тип данных как исходный тип данных);
- оба типа данных являются типами данных массивов с совместимыми основными типами данных и одинаковыми границами;
- оба типа данных являются типами данных списков с совместимыми основными типами данных;
- оба типа данных являются типами данных мультимножеств или наборов с совместимыми основными типами данных.

Пример 113 — 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;
```

Совместимы следующие типы данных:

Задан	Совместим с
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, set_of_up_to_five_positive

13 Исполняемые операторы

Исполняемые операторы определяют действия функций, процедур и правил. Эти операторы могут производить действия только над локальными переменными для FUNCTION, PROCEDURE и RULE. Эти операторы используются для того, чтобы определить логические отношения и действия, необходимые для поддержки определения ограничений, то есть разделов WERE и RULE. Данные операторы не оказывают влияния на экземпляры объектов в области значений, как это определено в разделе 5. Исполняемыми операторами являются: пустой (null), ALIAS (псевдоимени), присваивания (assignment), CASE (альтернативы), составной (compound), ESCAPE (выхода), IF (условный), вызов процедуры (procedure call), REPEAT (цикла), RETURN (возврата) и SKIP (пропуска).

Синтаксис:

```
291 stmt = alias_stmt | assignment_stmt | case_stmt | compound_stmt | escape_stmt |
           if_stmt | null_stmt | procedure_call_stmt | repeat_stmt | return_stmt |
           skip_stmt .
```

Исполняемые операторы могут появляться только внутри FUNCTION, PROCEDURE или RULE.

13.1 Пустой оператор

Исполняемый оператор, который состоит только из точки с запятой (;), называется пустым (null) оператором. Никаких действий пустой оператор не выполняет.

```
Синтаксис:
247 null_stmt = `;` .
```

Пример 114 — Показано возможное использование пустого оператора.

```
IF a = 13 THEN
    ; -- это пустой оператор.
ELSE
    b := 5;
END_IF;
```

13.2 Оператор псевдоимени

Оператор ALIAS (псевдоимени) обеспечивает возможность локального переименования определенных переменных и параметров.

```
Синтаксис:
164 alias_stmt = ALIAS variable_id FOR general_ref { qualifier } `;` stmt { stmt }
                 END_ALIAS `;` .
216 general_ref = parameter_ref | variable_ref .
```

В области действия оператора ALIAS выражение **variable_id** неявно объявляется соответствующей типизированной переменной, имеющие значения определенного идентификатора, следующего за ключевым словом FOR.

Примечание — Правила видимости для **variable_id** описаны в 10.3.1.

Пример 115 — Предположим, что существует тип объекта **point** (точка) с атрибутами **x,y,z**, оператор ALIAS может быть использован в функции **calculate_length** (вычислить длину) 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 Оператор присваивания

Оператор присваивания (assignment) используется для назначения экземпляра локальной переменной или параметра. Тип данных значения, присваиваемого переменной, должен быть совместим по присваиванию с переменной или параметром.

Примечание — Оператор присваивания может быть использован для определения того, что две локальные переменные являются одним и тем же экземпляром объекта.

```
Синтаксис:
166 assignment_stmt = general_ref { qualifier } `:=` expression `;` .
216 general_ref = parameter_ref | variable_ref .
```

Пример 116 — Следующие выражения являются правильными присваиваниями.

```
LOCAL
    a, b  : REAL;
    p     : point;
```

```
END LOCAL;
...
 a := 1.1;
b := 2.5 * a;
p.x := b;
```

Совместимость по присваиванию

Считается, что тип данных, присваиваемый переменной или параметру, совместим по присваиванию с типом данных результирующего выражения, если выполняется одно из следующих условий:

- типы данных являются одинаковыми;
- выражению присваивается тип, являющийся подтипом или конкретизацией типа, который объявлен для присваиваемой переменной;
- объявленный тип присваиваемой переменной является определенным типом, чьим фундаментальным типом является выбираемый тип, а выражению присваивается значение типа, совместимого с одним или более типами, установленными в списке выбора выбираемого типа.

Фундаментальным типом определенного типа является фундаментальный тип исходного типа, а фундаментальным типом типа, отличного от определенного типа, является сам этот тип;

- переменная представлена определенным типом, фундаментальным типом которого является простой тип, а выражению присвоено значение этого простого типа;
- переменная представлена агрегатным типом данных, а выражением является инициализатор агрегата, элементы которого совместимы по присваиванию с основным типом агрегатного типа данных.

Экземпляры компонента сложного объекта, которые не являются допустимыми экземплярами сложного объекта (см. приложение В), не могут быть присвоены параметрам или переменным и не могут быть переданы функциям и процедурам в качестве фактических параметров. Это не ограничивает присвоения допустимых экземпляров сложных объектов.

13.4 Оператор CASE

Оператор CASE (альтернативы) является механизмом, обеспечивающим выборочное исполнение выполняемых операторов, в зависимости от значения выражения. Оператор выполняется в зависимости от значения выражения **selector**. Оператор альтернативы состоит из выражения, являющегося переключателем (селектором) альтернативы, и списка альтернативных действий, каждому из которых предшествует одно или несколько выражений, являющихся метками альтернативы. Результирующий тип метки альтернативы должен быть совместим с типом переключателя альтернативы. При исполнении оператора альтернативы будет исполнен первый оператор, у которого значение метки альтернативы совпадает со значением переключателя альтернативы. Исполняется не более одного из альтернативных операторов. В случае, если ни у одной из меток альтернативы значение не совпадает со значением переключателя альтернативы, то:

- если имеется предложение OTHERWISE, исполняется оператор, связанный с OTHERWISE;
- если предложение OTHERWISE отсутствует, ни одного оператора исполнено не будет.

Синтаксис:

```
182 case_stmt = CASE selector OF { case_action } [ OTHERWISE `:` stmt ]
                END_CASE `;` .
283 selector = expression .
180 case_action = case_label { `,` case_label } `:` stmt .
181 case_label = expression .
```

Правила и ограничения

Тип значения, присвоенного альтернативным меткам, должен быть совместим с типом значения, присвоенного селектору альтернативы.

Пример 117 — Простейший пример оператора альтернативы, использующего целочисленные альтернативные метки.

```
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) ; -- это оператор исполнено!
      4, 5  : x := LOG(x) ;
OTHERWISE : x := 0.0 ;
END_CASE ;
```

13.5 Составной оператор

Составным оператором является последовательность операторов, заключенная между ключевыми словами BEGIN и END. Составной оператор действует как единичный оператор.

Примечание — Составной оператор не образует новой области действия.

Синтаксис:
183 compound_stmt = BEGIN stmt { stmt } END `;` .

Пример 118 — Простейший составной оператор:

```
BEGIN
      a = a+1 ;
      IF a > 100 THEN
      a := 0 ;
      END_IF;
END ;
```

13.6 Оператор выхода

Оператор ESCAPE вызывает немедленный переход к оператору, следующему непосредственно за тем оператором REPEAT, в котором появился данный оператор ESCAPE.

Примечание — Применение оператора ESCAPE является единственным способом выхода из бесконечного цикла при исполнения оператора REPEAT.

Синтаксис:
202 escape_stmt = ESCAPE `;` .

Правила и ограничения

Оператор ESCAPE должен появляться только внутри области действия оператора REPEAT.

Пример 119 — Оператор ESCAPE передает управление оператору, следующему за END_REPEAT, если **a < 0,**

```
REPEAT UNTIL (a=l);
      IF (a < 0) THEN
            ESCAPE;
      END_IF;
        ...
END_REPEAT;
-- управление передается здесь
```

13.7 Оператор If ... Then ... Else

Условный оператор IF ... THEN ... ELSE допускает условное выполнение операторов на основе выражения типа LOGICAL. Когда **logical_expression** принимает значение TRUE, исполняется оператор, следующий за ключевым словом THEN. Когда **logical_expression** принимает значение FALSE или UNKNOWN, исполняется оператор, следующий за ключевым словом ELSE, если это ключевое слово имеется в выражении. Если **logical_expression** принимает значение FALSE или UNKNOWN, а ключевое слово ELSE отсутствует, управление передается следующему оператору.

Синтаксис:
220 if_stmt = IF logical_expression THEN stmt { stmt } [ELSE stmt { stmt }]
END_IF `;` .
241 logical_expression = expression .

Пример 120 — Простой оператор IF:

```
IF a < 10 THEN
      c := c + 1;
ELSE
      c := c — 1;
END_IF;
```

13.8 Оператор вызова процедуры

Оператор вызова процедуры вызывает процедуру. Фактические параметры при вызове процедуры должны быть согласованы с формальными параметрами, установленными для этой процедуры, по числу, порядку и типу.

Синтаксис:

```
257 procedure_call_stmt = ( built_in_procedure | procedure_ref )
                                [ actual_parameter_list ] `;` .
157 actual_parameter_list = `(` parameter { `,` parameter } `)` .
251 parameter = expression .
```

Правила и ограничения

Передаваемые фактические параметры должны быть совместимы по присваиванию с формальными параметрами.

Пример 121 — Вызов встроенной процедуры INSERT.

```
INSERT (point_list, this_point, here);
```

13.9 Оператор цикла

Оператор цикла (REPEAT) используется для повторения в зависимости от условий выполнения последовательности операторов. Начало или продолжение повторения определяется путем оценки управляющего условия(ий). Управляющими условиями являются:

- конечная итерация;
- пока условие имеет значение TRUE;
- до тех пор, пока условие не получит значение TRUE.

Синтаксис:

```
272 repeat_stmt = REPEAT repeat_control `;` stmt { stmt } END_REPEAT `;` .
271 repeat_control = [ increment_control ] [ while_control ] [ until_control ] .
222 increment_control = variable_id `:=` bound_1 TO bound_2 [ BY increment ] .
174 bound_1 = numeric_expression .
175 bound_2 = numeric_expression .
221 increment = nunmeric_expression .
```

Для задания условий прерывания повторения может использоваться комбинация управляющих элементов. Для управления итерациями данные условия вычисляются следующим образом:

a) Если задано управление приращением, то до начала выполнения оператора REPEAT вычисляется оператор управления приращением, как описано в 13.9.1.

b) Вычисляется управляющее выражение WHILE, при его наличии. Если результат принимает значение TRUE (или условие WHILE отсутствует), тело оператора REPEAT исполняется. Если результат принимает значение FALSE или UNKNOWN, исполнение оператора REPEAT прерывается.

c) Когда исполнение тела оператора REPEAT завершено, вычисляется какое-либо управляющие выражение UNTIL. Если результирующим значением является TRUE, дальнейшее исполнение итераций прекращается и выполнение оператора REPEAT завершается. Если результат имеет значение FALSE или UNKNOWN, управление оператором REPEAT возвращается к управлению приращением. Если управляющее условие UNTIL отсутствует, управление оператором REPEAT возвращается к управлению приращением.

d) Если имеется управление приращением, значение переменной цикла изменяется на величину приращения на каждой итерации. Если значение переменной цикла находится в пределах от **bound_1** до **bound_2**, включая сами границы, управление проводится по вышеописанному пункту b), в противном случае выполнение оператора REPEAT прерывается.

Пример 122 — В данном примере показано, как в операторе REPEAT могут быть использованы несколько управляющих условий. Исполнения операторов будут повторяться до тех пор, пока не будет достигнута требуемая точность или не будет выполнено сто циклов, независимо от первого; то есть повторение прекращается, даже если решение не сошлось.

```
REPEAT i := 1 TO 100 UNTIL epsilon < 1.E-6;
  ...
  epsilon := ...;
END_REPEAT;
```

13.9.1 Управление приращением

Управление приращением приводит к выполнению тела оператора цикла для ряда значений из соответствующей последовательности. При входе в оператор цикла неявно объявляется переменная числового типа **variable_id**, принимающая значение **bound_1**. После каждой итерации переменной **variable_id** присваивается значение **variable_id + increment**. Если приращение (**increment**) не определено, для него по умолчанию принимается значение равное единице (1). Если значение **variable_id** лежит между **bound_1** и **bound_2** (включая случай, когда **variable_id = bound_2**), выполнение оператора цикла продолжается.

Синтаксис:
222 increment_control = variable_id `:=` bound_1 TO bound_2 [BY increment] .
174 bound_1 = numeric_expression .
175 bound_2 = numeric_expression .
221 increment = numeric_expression .

Правила и ограничения

a) Выражениям **numeric_expression**, представляющим **bound_1**, **bound_2** и **increment**, должны быть присвоены числовые значения.

b) Выражения **numeric_expression**, представляющие границы и приращение, вычисляются один раз при входе в оператор REPEAT.

c) Если какое-либо из выражений **numeric_expression**, представляющих границы или приращение, имеет неопределенное (?) значение, оператор REPEAT не выполняется.

d) Перед первым вычислением оператора управления приращением проверяются следующие условия:

- если **increment** (приращение) положительно, а **bound_1 > bound_2**, оператор REPEAT не выполняется;
- если **increment** отрицательно, а **bound_1 < bound_2**, оператор REPEAT не выполняется;
- если **increment** нулевое (0), оператор REPEAT не выполняется;
- иначе, оператор REPEAT выполняется до тех пор, пока значение переменной **variable_id** не выйдет за установленные границы или пока один из других операторов управления циклом не прервет его выполнение.

e) Переменной цикла **variable_id** присваивается значение **bound_1** в начале первого итерационного цикла, а в начале каждого последующего цикла она увеличивается на величину **increment** (приращения).

f) Тело оператора REPEAT не должно изменять значение переменной цикла.

g) Оператор REPEAT образует локальную область действия, в которой переменная цикла **variable_id** объявляется неявно как числовая переменная. Вследствие этого, любое объявление **variable_id** во внешней области действия не видимо в операторе REPEAT, а значение переменной цикла недоступно за пределами оператора REPEAT.

13.9.2 Управление WHILE

Управление WHILE обеспечивает инициализацию и продолжение выполнения тела оператора REPEAT, пока управляющее выражение имеет значение TRUE. Управляющее выражение вычисляется перед каждой итерацией.

Если присутствует управление WHILE, а управляющее выражение имеет значение FALSE или UNKNOWN, тело оператора REPEAT не выполняется.

Синтаксис:
316 while_control = WHILE logical_expression .
241 logical_expression = expression .

Правила и ограничения:

a) Логическому выражению **logical_expression** должно присваиваться значение типа LOGICAL.

b) Логическое выражение **logical_expression** вычисляется заново в начале каждой итерации.

13.9.3 Управление UNTIL

Управление UNTIL обеспечивает продолжение выполнения тела оператора REPEAT до тех пор, пока управляющее выражение не примет значение TRUE. Выражение должно вычислятся после каждой итерации.

Если управление UNTIL является единственным видом управления, всегда должна быть выполнена по крайней мере одна итерация.

Синтаксис:
312 until_control = UNTIL logical_expression .
241 logical_expression = expression .

Правила и ограничения

a) Логическому выражению **logical_expression** должно присваиваться значение типа LOGICAL.

b) Логическое выражение **logical_expression** вычисляется заново в конце каждой итерации.

13.10 Оператор возврата

Оператор возврата RETURN прерывает исполнение функции FUNCTION или процедуры PROCEDURE. В функции оператор RETURN должен определять выражение. Значение, полученное при вычислении данного выражения, является результатом функции и возвращается в точку ее вызова. Выражение должно быть совместимо по присваиванию с объявленным возвращаемым типом функции. В процедуре оператор RETURN не должен содержать никаких выражений.

Синтаксис:
276 return_stmt = RETURN [`(` expression `)`] `;` .

Правила и ограничения

Оператор RETURN может появляться только в функции FUNCTION или процедуре PROCEDURE.

Пример 123 — Виды оператора RETURN.

```
RETURN (50) ;              (* из функции *)
RETURN(work_point) ;       (* из функции *)
RETURN ;                   (* из процедуры *)
```

13.11 Оператор пропуска

Оператор пропуска SKIP вызывает немедленный переход в конец тела оператора REPEAT, в котором он появился. Затем вычисляются и оцениваются управляющие условия, как это описано в разделе 13.9.

Синтаксис:
290 skip_stmt = SKIP `;` .

Правила и ограничения

Оператор SKIP должен появляться только в области действия оператора REPEAT.

Пример 124 — Оператор SKIP передает управление оператору END_REPEAT, который вызывает оценку управления UNTIL.

```
REPEAT UNTIL (a=l);

    ...
     IF (a < 0) THEN
          SKIP;
     END_IF;
    ... -- этот оператор будет пропущен, если a < 0
END_REPEAT;
```

14 Встроенные константы

В языке EXPRESS существует несколько встроенных констант, которые описаны ниже.

Примечание — Считается, что встроенные константы имеют точное значение, даже если это значение непредставимо в компьютере.

14.1 Константа *e*

CONST_E является константой типа REAL, представляющей математическое значение *e* — основание функции натурального логарифма (ln). Ее значение задается следующей математической формулой:

$$e = \sum_{i=0}^{\infty} i!^{-1}$$

14.2 Неопределенная

Символ неопределенности (?) символизирует неоднозначное значение. Он совместим со всеми типами данных.

Примечание — Наиболее часто неопределенность (?) используется для спецификации верхней границы мультимножества, списка или набора. Этим представляется понятие о неограниченности размера агрегатного значения, определенного агрегатным типом данных.

14.3 Ложь

Константа FALSE (ложь) является константой типа LOGICAL, представляющей логическую запись понятия "ложь". Она совместима с типами данных BOOLEAN и LOGICAL.

14.4 Пи

Константа PI (пи) является константой типа REAL, представляющей математическое значение π, отношение длины окружности к ее диаметру.

14.5 Self

Псевдоконстанта SELF ссылается на данный экземпляр объекта или значение типа. SELF может появляться внутри объявления объекта, объявления типа или конструктора объекта.

Примечание — SELF не является константой, но ведет себя как константа в каждом контексте, где она может появиться.

14.6 Истина

Константа TRUE (истина) является константой типа LOGICAL, представляющей логическую запись понятия «истина». Она совместима с типами данных BOOLEAN и LOGICAL.

14.7 Неизвестная

Константа UNKNOWN (неизвестное) является константой типа LOGICAL, представляющей тот факт, что для оценки логического условия недостаточно данных. Она совместима с типом данных LOGICAL, но несовместима с типом данных BOOLEAN.

15 Встроенные функции

Предполагается, что все функции (и математические операции вообще) вычисляются с точным результатом.

Для каждой встроенной функции задан ее прототип, показывающий типы ее формальных параметров и результат (возвращаемое значение).

15.1 Abs — арифметическая функция

FUNCTION ABS (V:NUMBER) : NUMBER;

Функция ABS возвращает абсолютное значение числа

Параметры. V является числом.

Результат. Абсолютное значение V. Возвращаемый тип данных идентичен типу параметра V.

Пример 125 — **ABS (-10)** --> 10

15.2 ACos — арифметическая функция

FUNCTION ACOS (V:NUMBER) : REAL;

Функция ACOS возвращает величину угла по заданному значению косинуса.

Параметры. V — число, являющееся косинусом угла.

Результат. Угол в радианах ($0 \le$ результат $\le \pi$), косинус которого равен V.

Условия. $-1.0 \le V \le 1.0$

Пример 126 — **ACOS (0.3)** --> 1.266103 **...**

15.3 ASin — арифметическая функция

FUNCTION ASIN (V:NUMBER) : REAL;

Функция ASIN возвращает величину угла по заданному значению синуса.

Параметры. V — число, являющееся синусом угла.

Результат. Угол в радианах ($-\pi/2 \le$ результат $\le \pi/2$), синус которого равен V.

Условия. $-1.0 \le V \le 1.0$

Пример 127 — **ASIN (0.3)** --> 3.04692...e^{-1}

15.4 ATan — арифметическая функция

FUNCTION ATAN (VI:NUMBER; V2: NUMBER) : REAL;

Функция ATAN возвращает значение угла по заданному значению тангенса V, где V задано выражением V = V1/V2.

Параметры

a)V1 — число.

b)V2 — число.

Результат. Угол в радианах ($-\pi/2 \le$ результат $\le \pi/2$), тангенс которого равен V. Если V2 равно нулю, результатом является $\pi/2$ или $-\pi/2$ в зависимости от знака V1.

Условия. V1 и V2 не должны одновременно иметь нулевое значение.

Пример 128 — **ATAN (-5.5, 3.0)** --> -1.071449...

15.5 Blength — двоичная функция

FUNCTION BLENGTH (V:BINARY) : INTEGER;

Функция BLENGTH возвращает число битов в двоичном значении.

Параметры. V является двоичным значением.

Результат. Возвращаемым значением является число битов в переданном двоичном значении.

Пример 129

```
LOCAL
n : NUMBER;
x : BINARY := %01010010 ;
END_LOCAL;
 ...
n := BLENGTH ( x ); -- n присваивается значение 8
```

15.6 Cos — арифметическая функция

FUNCTION COS (V:NUMBER) : REAL;

Функция COS возвращает значение косинуса угла.

Параметры. V — число, являющееся значением угла в радианах.

Результат. Косинус V ($-1.0 \le$ результат ≤ 1.0).

Пример 130 — **COS (0.5)** --> 8.77582...E-1

15.7 Exists — общая функция

FUNCTION EXISTS (V:GENERIC) : BOOLEAN;

Функция EXISTS возвращает значение TRUE, если существует значение для входного параметра, или FALSE, если входной параметр не имеет значения. Функция EXISTS полезна для проверки, имеют ли значения OPTIONAL атрибуты или инициализированы ли переменные.

Параметры. V является выражением, результаты которого могут быть любого типа.

Результат. TRUE или FALSE, в зависимости от того, имеет ли V фактическое или неопределенное (?) значение.

Пример 131 — **IF EXISTS (a) THEN ...**

15.8 Exp — арифметическая функция

FUNCTION EXP (V:NUMBER) : REAL;

Функция EXP возвращает *e* (основание натуральной логарифмической системы), возведенное в степень V.

Параметры. V — число

Результат. Значение e^V.

Пример 132 — **EXP (10)** --> 2.202646...E+4

15.9 Format — общая функция

FUNCTION FORMAT (N:NUMBER; F:STRING) : STRING;

Функция FORMAT возвращает форматированное строковое представление числа.

Параметры

a) N — число (целое или действительное);

b) F — строка, содержащая команды форматирования.

Результат. Строка, представляющая число **N**, отформатированное в соответствии с **F**. При необходимости строковое представление округляется.

Строка форматирования содержит специальные символы, показывающие вид результата. Строка форматирования может быть записана тремя способами:

a) Строка форматирования может задавать символьное описание выходного представления.

b) Строка форматирования может задавать описание шаблона выходного представления.

c) Когда строка форматирования пустая, создается стандартное выходное представление.

15.9.1 Символьное представление

Общая форма символьного представления имеет вид: **[sign]width[.decimals]type**.

Параметр **sign** (знак) показывает, как представляется знак числа. Если **sign** не определен или определен как минус (-), первым возвращаемым символом для отрицательных чисел является минус, а для положительных чисел (включая ноль) — пробел. Если **sign** определен как плюс (+), то первым возвращаемым символом для отрицательных чисел является минус, для положительных чисел — плюс и для нуля — пробел.

Параметр **width** (ширина) задает общее число символов в возвращаемой строке. Он должен быть целым числом, большим двух. Если **width** задается с предшествующим нулем, возвращаемая строка будет содержать предшествующие нули, иначе предшествующие нули запрещены. Если форматируемое число содержит больше символов, чем задано значением **width**, возвращается строка с числом символов необходимого формата.

Параметр **decimals** (десятичные знаки) задает число цифр, которые возвращаются в строке справа от десятичной точки. Число десятичных знаков, если оно задается, должно быть положительным целым числом. Если параметр **decimals** не задан, возвращаемая строка не будет содержать десятичной точки следующих за ней цифр.

Параметр **type** является буквой, указывающей вид числа, представляемого в строке:

- если **type** имеет значение **I,** число должно быть представлено как целое, при этом:
  - параметр **decimals** не устанавливается,
  - значение параметра **width** должно быть не менее двух;
- если **type** имеет значение **F**, число должно быть представлено как действительное число с фиксированной десятичной точкой, при этом:
  - значение параметра **decimals,** если он задан, должно быть не менее единицы,
  - если параметр **decimals** не задан, его значение по умолчанию принимается равным двум,
  - значение параметра **width** должно быть не менее четырех;
- если **type** имеет значение **E**, число должно быть представлено как действительное число в экспоненциальной форме, при этом:
  - для **type E** всегда должен всегда устанавливаться параметр **decimals,**
  - значение параметра **decimals** должно быть не менее единицы,
  - значение параметра **width** должно быть не менее семи (7),
  - если в параметре **width** задан предшествующий ноль, первыми двумя символами мантиссы должны быть **0.**,
  - экспоненциальная часть должна содержать по меньшей мере два символа с обязательным знаком,
  - отображаемый знак экспоненты `E` должен быть прописной буквой.

Примечание — В таблице 20 показано, как форматирование влияет на представление различных значений.

Таблица 20 — Пример влияния символьного форматирования

Число	Формат	Вид	Комментарии
10	+71	` +10`	Нули удалены
10	+071	`+000010`	Нули не удалены
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`	Предшествующий ноль обязателен
9.876E123	8.2E	`9.88E+123`	Экспоненциальная часть содержит три символа, а ширина игнорирована
32.777	61	` 33`	Округлено

15.9.2 Представление в виде шаблона

В шаблонном формате каждый символ шаблона соответствует символу в возвращаемой строке. Используемые символы заданы в таблице 21.

Таблица 21 — Символы шаблонного форматирования

Символ	Значение
# (знак номера)	Представляет цифру
, (запятая)	Разделитель
. (точка)	Разделитель
+ — (плюс и минус)	Представляет знак
( ) (круглые скобки)	Представляет отрицание

Разделители точка `.` и запятая `,` используются следующим образом:

- если запятая `,` появляется в строке формата раньше точки `.`, то запятая `,` является символом группирования, а точка `.` — десятичным символом;
- если точка `.` появляется в строке формата раньше запятой `,`, то точка `.` является символом группирования, а запятая `,` — десятичным символом;
- если в строке формата есть только один разделитель, этот разделитель является десятичным символом.

Любой другой символ изображается без изменения.

Примечание — В таблице 22 показано, как форматирование влияет на представление различных значений.

Таблица 22 — Пример влияния шаблонного форматирования

Число	Формат	Вид	Комментарии
10	# # #	` 10`	
10	(# # #)	` 10`	Круглые скобки игнорированы
-10	(# # #)	`( 10)`	
7123.456	# # # , # # # . # #	` 7,123.46`	Нотация США
7123.456	# # # . # # # , # #	` 7.123,46`	Европейская нотация

15.9.3 Стандартное представление

Стандартным представлением для целого числа является `7I`. Стандартным представлением для действительного числа является `10E`. Описание символьных представлений представлено в 15.9.1.

15.10 HiBound — арифметическая функция

FUNCTION HIBOUND (V:AGGREGATE OF GENERIC) : INTEGER;

Функция HIBOUND возвращает объявленный верхний индекс массива (ARRAY) или объявленную верхнюю границу мультимножества (BAG), списка (LIST) или набора SET.

Параметры. V — агрегатное значение.

Результат

a) Когда V является массивом (ARRAY), возвращаемым значением является объявленный верхний индекс.

b) Когда V является мультимножеством (BAG), списком (LIST) или набором (SET), возвращаемым значением является объявленная верхняя граница; если границы не объявлены или верхняя граница объявлена неопределенной (?), то возвращается неопределенное (?) значение.

Пример 133 — Использование функции HIBOUND для вложенных агрегатных значений.

```
LOCAL
    a : ARRAY[-3:19] OF SET[2:4] OF LIST [0:?] OF INTEGER;
    hi, h2, h3 : INTEGER;
END_LOCAL;
 ...
a[-3][l][l] := 2;                -- помещает значение в список
hi := HIBOUND(a);                -- =19 (верхний индекс массива)
h2 := HIBOUND(a[-3]);            -- = 4 (верхняя граница набора)
h3 := HIBOUND(a[-3][1]);         -- = ? (верхняя граница списка (неограниченного))
```

15.11 HiIndex — арифметическая функция

FUNCTION HIINDEX (V:AGGREGATE OF GENERIC) : INTEGER;

Функция HIINDEX возвращает верхний индекс массива (ARRAY) или число элементов в мультимножестве (BAG), списке (LIST) или наборе (SET).

Параметры. V — агрегатное значение

Результат

a) Когда V — массив (ARRAY), возвращаемым значением является объявленный верхний индекс.

b) Когда V — мультимножество (BAG), список (LIST) или набор (SET), возвращаемым значением является фактическое число элементов в агрегатном значении.

Пример 134 — Использование функции HIINDEX для вложенных агрегатных значений

```
LOCAL
    a : ARRAY[-3:19] OF SET[2:4] OF LIST[0:?] OF INTEGER;
    hi, 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 по отношению к
                                                        -- границам набора
    h3 := HIINDEX(a[-3][1]); -- = 1 (размер списка)
```

15.12 Length — строковая функция

FUNCTION LENGTH (V:STRING) : INTEGER;

Функция LENGTH возвращает число символов в строке.

Параметры. V — строковое значение.

Результат. Возвращаемым значением является число символов в строке и оно должно быть большим или равным нулю.

Пример 135 — Использование функции LENGTH.

```
LOCAL
    n : NUMBER;
    xl : STRING := `abc`;
    x2 : STRING := «000025FF000101B5»;
END_LOCAL;
...
```

```
n := LENGTH ( x1 ); -- n присваивается значение 3
n := LENGTH ( x2 ); -- n присваивается значение 2
```

15.13 LoBound — арифметическая функция

FUNCTION LOBOUND (V:AGGREGATE OF GENERIC) : INTEGER;

Функция LOBOUND возвращает объявленный нижний индекс массива (ARRAY) или объявленную нижнюю границу мультимножества (BAG), списка (LIST) или набора (SET).

Параметры. V — агрегатное значение.

Результат

a) Когда V — массив (ARRAY), возвращаемым значением является объявленный нижний индекс.

b) Когда V — мультимножество (BAG), список (LIST) или набор (SET), возвращаемым значением является объявленная нижняя граница; если нижняя граница не объявлена, то возвращается нуль (0).

Пример 136 — Использование функции LOBOUND для вложенных агрегатных значений.

```
LOCAL
  a : ARRAY[-3: 19] OF SET [2:4] OF LIST[0:?] OF INTEGER;
  hi, 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 Log — арифметическая функция

FUNCTION LOG (V:NUMBER) : REAL;

Функция LOG возвращает натуральный логарифм числа.

Параметры. V — число

Результат. Действительное число, которое является натуральным логарифмом V.

Условия. V > 0.0

Пример 137 — **LOG (4.5)** --> 1.504077...E0

15.15 Log2 — арифметическая функция

FUNCTION LOG2 (V:NUMBER) : REAL;

Функция LOG2 возвращает логарифм числа по основанию два (2).

Параметры. V — число.

Результат. Действительное число, которое является логарифмом V по основанию два (2).

Условия. V > 0.0

Пример 138 — **LOG2 (8)** --> 3.00...E0

15.16 Log10 — арифметическая функция

FUNCTION LOG10 (V:NUMBER) : REAL;

Функция LOG10 возвращает логарифм числа по основанию десять (10).

Параметры. V — число.

Результат. Действительное число, которое является логарифмом V по основанию десять (10).

Условия. V > 0.0

Пример 139 — **LOG10 (10)** --> 1.00...E0

15.17 LoIndex — арифметическая функция

FUNCTION LOINDEX (V:AGGREGATE OF GENERIC) : INTEGER;

Функция LOINDEX возвращает нижний индекс агрегатного значения.

Параметры. V — агрегатное значение.

Результат

a) Когда V — массив (ARRAY), возвращаемым значением является объявленный нижний индекс.

b) Когда V — мультимножество (BAG), список (LIST) или набор (SET), возвращаемым значением является единица (1).

Пример 140 — Использование функции LOINDEX для вложенных агрегатных значений.

```
LOCAL
    a : ARRAY[-3:19] OF SET[2:4] OF LIST[0:?] OF INTEGER;
    hi, 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 NVL — функция нулевого значения

FUNCTION NVL(V:GENERIC:GEN1; SUBSTITUTE:GENERIC:GEN1):GENERIC:GEN1;

Функция NVL возвращает или входное значение, или альтернативное значение, если входной параметр имеет неопределенное (?) значение.

Параметры

a) V — выражение любого типа.

b) SUBSTITUTE — выражение, которое не должно быть неопределенным (?).

Результат. Когда V не является неопределенным (?), возвращается значение V. Иначе возвращается значение SUBSTITUTE.

Пример 141 —

```
ENTITY unit_vector;
    x, y : REAL;
    z : OPTIONAL REAL;
WHERE
    x**2 + y**2 + NVL(z, 0.0)**2 = 1.0;
END_ENTITY;
```

Функция NVL используется для подстановки нуля (0.0) в качестве значения **Z**, когда **Z** является неопределенным (?).

15.19 Odd — арифметическая функция

FUNCTION ODD (V:INTEGER) : LOGICAL;

Функция ODD возвращает значение TRUE или FALSE, в зависимости от того, является ли число нечетным или четным.

Параметры. V — целое число.

Результат. Когда V MOD 2 = 1, возвращается TRUE, иначе возвращается FALSE.

Условия. Ноль не является нечетным числом.

Пример 142 — **ODD (121)** --> TRUE

15.20 RolesOf — общая функция

FUNCTION ROLESOF (V:GENERIC) : SET OF STRING;

Функция ROLESOF возвращает набор строк, содержащих полные квалифицированные имена ролей, играемых заданным экземпляром объекта. Полное квалифицированное имя определяется как имя атрибута, квалифицированное именем схемы и объекта, в которых объявлен атрибут (то есть `SCHEMA.ENTITY.ATTRIBUTE`).

Параметры. V — любой экземпляр типа данных объекта.

Результат. Набор строковых значений (в верхнем регистре), содержащий полные квалифицированные имена атрибутов экземпляров объектов, которые используют экземпляр V.

Если поименованный тип данных импортирован с помощью спецификаций USE или REFERENCE, также возвращается имя типа в исходной схеме и имя исходной схемы, если тип был переименован при импорте. Поскольку операторы USE могут быть связанными в цепочку, возвращаются все имена связанных схем и имя поименованного типа в каждой схеме.

Пример 143 — Этот пример показывает, что точка может быть использована как центр окружности. Функция ROLESOF возвращает строку, показывающую, какую роль фактически играет данный экземпляр объекта.

```
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;
    centre   : point;
    axis     : vector;
    radius   : REAL;
END ENTITY,
...
 LOCAL

    p : point := point(1.0, 0.0, 0.0);
    c : circle := circle(p, vector(l, 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 Sin — арифметическая функция

FUNCTION SIN (V:NUMBER) : REAL;

Функция SIN возвращает значение синуса угла.

Параметры. V — число, представляющее угол в радианах

Результат. Синус V (-1.0 ≤ результат ≤1.0)

Пример 144 — **SIN (PI)** --> 0.0

15.22 SizeOf — агрегатная функция

FUNCTION SIZEOF (V:AGGREGATE OF GENERIC) : INTEGER;

Функция SIZEOF возвращает число элементов в агрегатном значении.

Параметры. V — агрегатное значение.

Результат

a) Когда V — массив (ARRAY), возвращаемым значением является объявленное для него число элементов агрегатного типа данных.

b) Когда V — мультимножество (BAG), список (LIST) или набор (SET), возвращаемым значением является фактическое число элементов в агрегатном значении.

Пример 145 —
```
LOCAL
    n : NUMBER;
    y : ARRAY[2:5] OF b;
```

```
        END_LOCAL;
          ...
          n := SIZEOF (y); -- n присваивается значение 4
```

15.23 Sqrt — арифметическая функция

FUNCTION SQRT (V:NUMBER) : REAL;

Функция SQRT возвращает неотрицательный квадратный корень числа.

Параметры. V — любое неотрицательное число.

Результат. Неотрицательный квадратный корень V.

Условия. $V \geq 0.0$

Пример 146 — **SQRT (121)** --> 11.0

15.24 Tan — арифметическая функция

FUNCTION TAN (V:NUMBER) : REAL;

Функция TAN возвращает значение тангенса угла.

Параметры. V — число, выражающее угол в радианах.

Результат. Тангенс угла. Если угол имеет значение $n\pi/2$, где n нечетное целое число, возвращается неопределенное (?) значение.

Пример 147 — **TAN (0.0)** --> 0.0

15.25 TypeOf — общая функция

FUNCTION TYPEOF (V:GENERIC) : SET OF STRING;

Функция TYPEOF возвращает набор строк, содержащий имена всех типов данных, параметр которых является членом. За исключением простых типов данных (BINARY, BOOLEAN, INTEGER, LOGICAL, NUMBER, REAL и STRING) и агрегатных типов данных (ARRAY, BAG, LIST, SET), имена которых квалифицируются именем схемы, содержащей определение типа.

Примечание 1 — Основное назначение данной функции состоит в проверке, может ли данное значение (переменная, значение атрибута) использоваться для заданной цели, например, чтобы убедиться в совместимости двух значений по присваиванию. Также данная функция может использоваться, если различные подтипы или конкретизации заданного типа по-разному трактуются в некотором контексте.

Параметры. V — значение любого типа.

Результат. Содержимым возвращаемого набора строковых значений являются имена (в верхнем регистре) всех типов, где V является членом. Если это не простой и не агрегатный тип данных, такие имена квалифицируются именем схемы, содержащей определение типа (`SHEMA.TYPE`). Возвращаемое значение может быть получено с помощью нижеприведенного алгоритма (алгоритм приводится в целях разъяснения, а не в качестве необходимого компонента реализации).

a) Когда именем типа является поименованный тип данных, результирующий набор инициализируется включением в набор имени того типа, к которому принадлежит V. Если V является формальным параметром, он заменяется соответствующим фактическим параметром. Если V является агрегатным значением, именем типа является точное имя соответствующего агрегатного типа данных (ARRAY, BAG, LIST, SET).

b) Повторять до тех пор, пока наращивание списка не прекратится:

1) Повторить для всех имен в результирующем наборе:

- если текущее имя является именем простого типа данных, пропустить;
- если текущее имя является именем агрегатного типа данных (ARRAY, BAG, LIST, SET), пропустить;
- если текущее имя является именем перечисляемого типа данных, пропустить;
- если текущее имя является именем выбираемого типа данных, к результирующему набору добавляются имена всех типов (с именем схемы) из списка выбора, экземпляром которых действительно является V. (Этих имен может быть несколько, так как список выбора может содержать имена типов, являющихся совместимыми подтипами общего супертипа или конкретизацией одного общего обобщенного типа);
- если текущее имя является именем любого другого вида определенного типа данных, то имя типа, на который ссылается текущий тип, добавляется к результи-

рующему набору с именем схемы. Если ссылка делается на агрегатный тип данных, то добавляется только имя агрегатного типа;
- если текущее имя является именем объекта, то к результирующему набору добавляются имена всех его подтипов (включая, при необходимости, имя схемы) экземпляром которых является V.

2) Повторить для всех имен в результирующем наборе:
- если текущее имя является именем подтипа, к результирующему набору добавляются имена всех его супертипов;
- если текущее имя является именем специализации (конкретизации), к результирующему набору добавляются имена всех ее обобщений.

3) Повторить для всех имен в результирующем наборе:
- если текущее имя появляется в списке типов по крайней мере одного выбираемого типа данных, к результирующему набору добавляются имена всех выбираемых типов данных, определения которых базируются на типе с текущим именем.

4) Повторить для всех имен в результирующем наборе:
- если текущее имя импортировано в текущую схему с помощью спецификации USE или REFERENCE, в результирующий набор добавляется имя из схемы, откуда был осуществлен импорт, квалифицированное именем этой схемы. Поскольку операторы USE могут быть последовательными, в результирующий набор добавляется имя типа из всех связанных схем, квалифицированное именами соответствующих схем.

c) Возврат результирующего набора.

Если V имеет неопределенное (?) значение , функция TYPEOF возвращает пустой набор.

Примечание 2 — Функция TYPEOF завершает дальнейший выбор при встрече агрегатного типа данных. Функция не представляет никаких сведений об основном типе агрегатного значения. При необходимости данные сведения можно получить путем применения функции TYPEOF к корректным элементам агрегатного значения.

Пример 148 — 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 истинны:

```
TYPEOF (lst)      = [`THIS_SCHEMA.MYLIST`, `LIST`]
TYPEOF (lst [17]) = [`REAL`, `NUMBER`]
```

Пример 149 — Показано влияние использования импорта со спецификацией USE или REFERENCE на основе предыдущего примера.

```
SCHEMA another_schema;
    REFERENCE FROM this_schema (mylist AS hislist);
    ...
        lst : hislist;
    ...
END_SCHEMA;
```

Теперь можно сказать:

```
TYPEOF (lst) = [`ANOTHER_SCHEMA.HISLIST`, `THIS_SCHEMA.MYLIST`, `LIST`]
```

15.26 UsedIn — общая функция

```
FUNCTION USEDIN ( T:GENERIC; R:STRING ) : BAG OF GENERIC;
```

Функция USEDIN возвращает каждый экземпляр объекта, который использует данный экземпляр объекта в указанной роли.

Параметры

a) T — любой экземпляр любого типа данных объекта.

b) R — строка, содержащая полностью квалифицированное имя атрибута (роли), как это определено в 15.20.

Результат. Каждый экземпляр объекта, использующий данный экземпляр в указанной роли, возвращается в мультимножество.

Если экземпляр **T** не играет никаких ролей или роль **R** не найдена, возвращается пустое мультимножество.

Когда **R** является пустой строкой, то документируется каждое использование **T**. Проверяются все связи, направленные в сторону **T**. Когда связь начинается с атрибута, имеющего имя **R**, экземпляр объекта, содержащий **T**, относится данный атрибут, добавляется в результирующее мультимножество. Заметим, что, если **T** нигде не используется, то возвращается пустое мультимножество.

Пример 150 — Этот пример показывает, как может быть написано правило, проверяющее, что должна быть точка в начале координат, используемая как центр окружности. Заметим, что в этом примере выражение запроса (см. 12.6.7) используется как параметр при обращении к функции SIZEOF.

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

ENTITY circle;
    centre   : point;
    axis     : vector;
    radius   : REAL;
END_ENTITY; ...
(* Это правило находит каждую точку, используемую как центр окружности, и затем проверяет,
что по меньшей мере одна из этих точек лежит в начале координат *)

 ...
RULE exampl FOR (point);
LOCAL
    centre_points : SET OF circle := [ ]; -- пустой набор окружностей
END_LOCAL;
REPEAT i := LOINDEX(point) TO HIINDEX(point);
    centre_points := centre_points +
                     USEDIN(point[i], `THIS_SCHEMA.CIRCLE.CENTRE` );
END_REPEAT;
WHERE Rl : SIZEOF(
               QUERY(                                  -- начало запроса
                 at_zero <* centre_points |            -- получить все точки
                 (at_zero.centre = point(0.0, 0.0, 0.0)) -- на 0,0,0
                 )
           ) >= 1;                                     -- по меньшей мере одна
END_RULE;
```

15.27 Value — арифметическая функция

FUNCTION VALUE (V:STRING) : NUMBER;

Функция VALUE возвращает численное представление строки.

Параметры. V — строка, содержащая действительный или целочисленный литерал (см. 7.5).

Результат. Число, соответствующее представлению строки. Если строка не может быть интерпретирована ни как действительный или целочисленный литерал, возвращается неопределенное (?) значение.

Пример 151

```
VALUE ( `1.234` )   --> 1.234 (действительное)
VALUE ( `20` )      --> 20 (целое)
VALUE ( `abc` )     --> ? (неопределенное)
```

15.28 Value_in — функция принадлежности

FUNCTION VALUE_IN (C:AGGREGATE OF GENERIC:GEN; V:GENERIC:GEN) : LOGICAL;

Функция VALUE_IN возвращает логическое значение, в зависимости от того, является ли конкретное значение членом агрегата.

Параметры

a) C — агрегат любого типа.

b) V — выражение, которое совместимо по присваиванию с основным типом C.

Результат

a) Если **V** или **C** имеет неопределенное (?) значение, возвращается значение UNKNOWN.

b) Если какой-либо элемент **C** имеет значение, равное **V**, возвращается значение TRUE.

c) Если какой-либо из элементов **C** имеет неопределенное (?) значение, возвращается значение UNKNOWN.

d) Иначе возвращается значение FALSE.

Пример 152 — Следующая проверка гарантирует, что существует хотя бы одна точка, 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 Value_unique — функция уникальности

FUNCTION VALUE_UNIQUE (V:AGGREGATE OF GENERIC) : LOGICAL;

Функция VALUE_UNIQUE возвращает логическое (LOGICAL) значение, в зависимости от того, являются ли элементы агрегата уникальными по значению.

Параметры. V — агрегат любого типа.

Результат

a) Если **V** имеет неопределенное (?) значение, возвращается значение UNKNOWN.

b) Если значения любых двух элементов **V** равны, возвращается значение FALSE.

c) Если любой из элементов **V** имеет неопределенное (?) значение, возвращается значение UNKNOWN.

d) Иначе возвращается значение TRUE.

Пример 153 — Этот тест служит для проверки, что каждая точка в наборе имеет позицию, отличную от других точек (по определению, эти точки должны быть представлены разными экземплярами объектов).

```
IF VALUE_UNIQUE(points) THEN ...
```

16 Встроенные процедуры

В языке EXPRESS имеется две встроенные процедуры, каждая из которых используется для управления списками. В данном разделе приведено описание этих процедур.

Для того чтобы показать типы данных формальных параметров, задан заголовок каждой из процедур.

16.1 Insert

PROCEDURE INSERT (VAR L:LIST OF GENERIC:GEN; E:GENERIC:GEN; P:INTEGER);

Процедура INSERT вставляет элемент в заданную позицию в списке.

Параметры

a) **L** — это значение списка, в который должен быть вставлен элемент.

b) **E** — это экземпляр, вставляемый в **L**. **E** должен быть совместим с основным типом **L**, что указывается в заголовке процедуры с помощью меток типа.

c) **P** — целое число, задающее позицию в **L**, на которую должен быть вставлен элемент **E**.

Результат. L изменяется за счет вставки **E** в **L** на указанную позицию. Элемент вставляется сразу за существующим элементом в позиции **P** так, что при **P** = 0 элемент **E** становится первым элементом.

Условия. 0 ≤ P ≤ SIZEOF(L).

16.2 Remove

PROCEDURE REMOVE (VAR L:LIST OF GENERIC; P:INTEGER);

Процедура REMOVE удаляет элемент из заданной позиции в списке.

Параметры

a) **L** — это список, из которого должен быть удален элемент.

b) **P** — целое число, задающее позицию удаляемого элемента в **L**.

Результат. Список **L** изменяется за счет удаления элемента, находившегося в позиции **P**.

Условия. 1 ≤ P ≤ SIZEOF(L).

ПРИЛОЖЕНИЕ А
(обязательное)

Синтаксис языка EXPRESS

В настоящем приложении определены лексические элементы языка и грамматические правила, которым они должны подчиняться.

Примечание — При непосредственном использовании этих синтаксических определений синтаксический анализ будет неоднозначным. Определения написаны для того, чтобы передать информацию, относящуюся к использованию идентификаторов. Интерпретируемые идентификаторы определяют лексемы, которые ссылаются на объявленные идентификаторы и, следовательно, не могут распознать простые идентификаторы (**simple_id**). Ввиду этого разработчик синтаксического анализатора должен создать просмотровую таблицу или подобную конструкцию для обеспечения возможности разрешения ссылок на идентификаторы с возвратом ссылок средству проверки грамматических правил. Такой подход разработчиков синтаксических анализаторов обеспечит отсутствие неоднозначностей при использовании идентификаторов.

A.1 Лексемы

Следующие правила определяют лексемы, используемые в языке EXPRESS. За исключением случаев, когда это явно задано в синтаксических правилах, внутри текста, удовлетворяющего простому синтаксическому правилу, определенному в пунктах А.1.1 — А.1.4, не должно появляться игнорируемых фрагментов или комментариев.

А.1.1 Ключевые слова

В этом пункте заданы правила представления ключевых слов EXPRESS.

Примечание — Представление ключевых слов в этом подразделе соответствует типографскому соглашению, установленному в 6.1, по которому каждое ключевое слово представлено как синтаксическое правило, левая часть которого является самим ключевым словом в верхнем регистре (прописными буквами). Поскольку строковые литералы в синтаксических правилах не учитывают регистр, ключевые слова могут записываться в верхнем, нижнем или смешанном регистрах.

```
 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 BEGIN = 'begin' .
13 BINARY = 'binary' .
14 BLENGTH = 'blength' .
15 BOOLEAN = 'boolean' .
16 BY = 'by' .
17 CASE = 'case' .
18 CONSTANT = 'constant' .
19 CONST_E = 'const_e' .

20 CONTEXT = 'context' .
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_CONTEXT = 'end_context' .

30 END_ENTITY = 'end_entity' .
31 END_FUNCTION = 'end_function' .
```

```
32 END_IF = 'end_if' .
33 END_LOCAL = 'end_local' .
34 END_MODEL = 'end_model' .
35 END_PROCEDURE = 'end_procedure' .
36 END_REPEAT = 'end_repeat' .
37 END_RULE = 'end_rule' .
38 END_SCHEMA = 'end_schema' .
39 END_TYPE = 'end_type' .

40 ENTITY = 'entity' .
41 ENUMERATION = 'enumeration' .
42 ESCAPE = 'escape' .
43 EXISTS = 'exists' .
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 HIBOUND = 'hibound' .
53 HIINDEX = 'hiindex' .
54 IF = 'if' .
55 IN = 'in' .
56 INSERT = 'insert' .
57 INTEGER = 'integer' .
58 INVERSE = 'inverse' .
59 LENGTH = 'length' .

60 LIKE = 'like' .
61 LIST = 'list' .
62 LOBOUND = 'lobound' .
63 LOCAL = 'local' .
64 LOG = 'log' .
65 LOG10 = 'log10' .
66 LOG2 = 'log2' .
67 LOGICAL = 'logical' .
68 LOINDEX = 'loindex' .
69 MOD = 'mod' .

70 MODEL = 'model' .
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 REPEAT = 'repeat' .
87 RETURN = 'return' .
88 ROLESOF = 'rolesof' .
89 RULE = 'rule' .
```

```
 90 SCHEMA = 'schema' .
 91 SELECT = 'select' .
 92 SELF = 'self' .
 93 SET = 'set' .
 94 SIN = 'sin' .
 95 SIZEOF = 'sizeof' .
 96 SKIP = 'skip' .
 97 SQRT = 'sqrt' .
 98 STRING = 'string' .
 99 SUBTYPE = 'subtype' .

100 SUPERTYPE = 'supertype' .
101 TAN = 'tan' .
102 THEN = 'then' .
103 TO = 'to' .
104 TRUE = 'true' .
105 TYPE = 'type' .
106 TYPEOF = 'typeof' .
107 UNIQUE = 'unique' .
108 UNKNOWN = 'unknown' .
109 UNTIL = 'until' .

110 USE = 'use' .
111 USEDIN = 'usedin' .
112 VALUE = 'value' .
113 VALUE_IN = 'value_in' .
114 VALUE_UNIQUE = 'value_unique' .
115 VAR = 'var' .
116 WHERE = 'where' .
117 WHILE = 'while' .
118 XOR = 'xor' .
```

А.1.2 Классы символов

Следующие правила определяют различные классы символов, которые используются в написании лексем в А.1.3.

```
119 bit = '0' | '1' .
120 digit = '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9' .
121 digits = digit { digit } .
122 encoded_character = octet octet octet octet .
123 hex_digit = digit | 'a' | 'b' | 'c' | 'd' | 'e' | 'f' .
124 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' .
125 lparen_not_star = '(' not_star .
126 not_lparen_star = not_paren_star | ')' .
127 not_paren_star = letter | digit | not_paren_star_special .
128 not_paren_star_quote_special = '!' | ' " ' | '#' | '$' | '%' | '&' | '+' | ',' |
                                  '-' | '.' | '/' | ':' | ';' | '<' | '=' | '>' |
                                  '?' | '@' | '[' | '\' | ']' | '^' | '_' | '`' |
                                  '{' | '|' | '}' | '~' .
129 not_paren_star_special = not_paren_star_quote_special | '"' .

130 not_quote = not_paren_star_quote_special | letter | digit | '(' | ')' | '*' .
131 not_rparen = not_paren_star | '*' | '(' .
132 not_star = not_paren_star | '(' | ')' .
133 octet = hex_digit hex_digit .
134 special = not_paren_star_quote_special | '(' | ')' | '*' | '"' .
135 star_not_rparen = '*' not_rparen .
```

А.1.3 Лексические элементы

Следующие правила определяют, как определенные комбинации символов интерпретируются в качестве лексических элементов языка.

```
136 binary_literal = '%' bit { bit } .
137 encoded_string_literal = ' " ' encoded_character { encoded_character } ' " ' .
138 integer_literal = digits .
139 real_literal = digits '.' [ digits ] [ 'e' [ sign ] digits ] .
140 simple_id = letter { letter | digit | '_' } .
141 simple_string_literal = \q { ( \q \q ) | not_quote | \s | \x8 | \x9 | \xA
                          | \xB | \xC | \xD } \q .
```

А.1.4 Комментарии

Следующие правила определяют синтаксис комментариев в языке EXPRESS.

```
142 embedded_remark = '(*' { not_lparen_star | lparen_not_star | star_not_rparen |
                      embedded_remark } '*)' .
143 remark = embedded_remark | tail_remark .
144 tail_remark = '--' { \a | \s | \x8 | \x9 | \xA | \xB | \xC | \xD } \n .
```

А.1.5 Интерпретируемые идентификаторы

Следующие правила представляют идентификаторы, конкретный смысл которых известен (то есть они объявляются как типы, функции и т.д.).

Примечание — Предполагается, что идентификаторы, соответствующие этим правилам, распознаются реализацией. Способ, которым реализация получает эту информацию, не связан с описанием языка. Одним из способов получения данной информации является многопроходный синтаксический анализ, когда на первом проходе выбираются идентификаторы из соответствующих объявлений, так что на следующем проходе появляется возможность отличить, к примеру, ссылку на переменную (**variable_ref**) от ссылки на функцию (**function_ref**).

```
145 attribute_ref = attribute_id .
146 constant_ref = constant_id .
147 entity_ref = entity_id .
148 enumeration_ref = enumeration_id .
149 function_ref = function_id .

150 parameter_ref = parameter_id .
151 procedure_ref = procedure_id .
152 schema_ref = schema_id .
153 type_label_ref = type_label_id .
154 type_ref = type_id .
155 variable_ref = variable_id .
```

А.2 Грамматические правила

Следующие правила задают способы, которыми рассмотренные выше лексические элементы могут быть объединены в конструкции EXPRESS. Игнорируемые фрагменты текста и/или комментарии в этих правилах могут появляться между двумя любыми лексемами. Основным синтаксическим правилом для EXPRESS является синтаксис (**syntax**).

```
156 abstract_supertype_declaration = ABSTRACT SUPERTYPE [ subtype_constraint ] .
157 actual_parameter_list = '(' parameter { ',' parameter } ')' .
158 add_like_op = '+' | '-' | OR | XOR .
159 aggregate_initializer = '[' [ element { ',' element } ] ']' .

160 aggregate_source = simple_expression .
161 aggregate_type = AGGREGATE [ ':' type_label ] OF parameter_type.
162 aggregation_types = array_type | bag_type | list_type | set_type .
163 algorithm_head = { declaration } [ constant_decl ] [ local_decl ] .
164 alias_stmt = ALIAS variable_id FOR general_ref { qualifier } ';' stmt { stmt }
                 END_ALIAS ';' .
165 array_type = ARRAY bound_spec OF [ OPTIONAL ] [ UNIQUE ] base_type .
166 assignment_stmt = general_ref { qualifier } ':=' expression ';' .
167 attribute_decl = attribute_id | qualified_attribute .
168 attribute_id = simple_id .
169 attribute_qualifier = '.' attribute_ref .

170 bag_type = BAG [ bound_spec ] OF base_type .
171 base_type = aggregation_types | simple_types | named_types .
172 binary_type = BINARY [ width_spec ] .
```

```
173 boolean_type = BOOLEAN .
174 bound_1 = numeric_expression .
175 bound_2 = numeric_expression .
176 bound_spec = '[' bound_1 ':' bound_2 ']' .
177 built_in_constant = CONST_E | PI | SELF | '?' .
178 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 .
179 built_in_procedure = INSERT | REMOVE .

180 case_action = case_label { ',' case_label } ':' stmt .
181 case_label = expression .
182 case_stmt = CASE selector OF { case_action } [ OTHERWISE ':' stmt ]
                END_CASE ';' .
183 compound_stmt = BEGIN stmt { stmt } END ';' .
184 constant_body = constant_id ':' base_type ':=' expression ';' .
185 constant_decl = CONSTANT constant_body { constant_body }
                    END_CONSTANT ';' .
186 constant_factor = built_in_constant | constant_ref .
187 constant_id = simple_id .
188 constructed_types = enumeration_type | select_type .
189 declaration = entity_decl | function_decl | procedure_decl | type_decl .

190 derived_attr = attribute_decl ':' base_type ':=' expression ';'.
191 derive_clause = DERIVE derived_attr { derived_attr } .
192 domain_rule = [ label ':' ] logical_expression .
193 element = expression [ ':' repetition ] .
194 entity_body = { explicit_attr } [ derive_clause ] [ inverse_clause ]
                  [ unique_clause ] [ where_clause ] .
195 entity_constructor = entity_ref '(' [ expression { ',' expression } ] ')' .
196 entity_decl = entity_head entity_body END_ENTITY ';' .
197 entity_head = ENTITY entity_id [ subsuper ] ';' .
198 entity_id = simple_id .
199 enumeration_id = simple_id .

200 enumeration_reference = [ type_ref '.' ] enumeration_ref .
201 enumeration_type = ENUMERATION OF '(' enumeration_id { ',' enumeration_id } ')' .
202 escape_stmt = ESCAPE ';' .
203 explicit_attr = attribute_decl { ',' attribute_decl } ':' [ OPTIONAL ]
                    base_type ';' .
204 expression = simple_expression [ rel_op_extended simple_expression ] .
205 factor = simple_factor [ '**' simple_factor ] .
206 formal_parameter = parameter_id { ',' parameter_id } ':' parameter_type .
207 function_call = ( built_in_function | function_ref ) [ actual_parameter_list ] .
208 function_decl = function_head [ algorithm_head ] stmt { stmt }
                    END_FUNCTION ';'
209 function_head = FUNCTION function_id [ '(' formal_parameter
                    { ';' formal_parameter } ')' ] ':' parameter_type ';' .

210 function_id = simple_id .
211 generalized_types = aggregate_type | general_aggregation_types| generic_type .
212 general_aggregation_types = general_array_type | general_bag_type |
                                general_list_type | general_set_type .
213 general_array_type = ARRAY [ bound_spec ] OF [ OPTIONAL ] [ UNIQUE ]
                         parameter_type .
214 general_bag_type = BAG [ bound_spec ] OF parameter_type .
215 general_list_type = LIST [ bound_spec ] OF [ UNIQUE ] parameter_type .
216 general_ref = paraineter_ref | variable_ref .
217 general_set_type = SET [ bound_spec ] OF parameter_type .
218 generic_type = GENERIC [ ':' type_label ] .
```

```
219 group_qualifier = '\' entity_ref .

220 if_stmt = IF logical_expression THEN stmt { stmt } [ ELSE stmt { stmt } ]
              END_IF ';' .
221 increment = numeric_expression .
222 increment_control = variable_id ':=' bound_1 TO bound_2 [ BY increment ] .
223 index = numeric_expression .
224 index_1 = index .
225 index_2 = index .
226 index_qualifier = '[' index_1 [ ':' index_2 ] ']' .
227 integer_type = INTEGER .
228 interface_specification = reference_clause | use_clause .
229 interval = '{' interval_low interval_op interval_item interval_op
               interval_high '}' .

230 interval_high = simple_expression .
231 interval_item = simple_expression .
232 interval_low = simple_expression .
233 interval_op = '<' | '<=' .
234 inverse_attr = attribute_decl ':' [ ( SET | BAG ) [ bound_spec] OF ] entity_ref
                   FOR attribute_ref ';' .
235 inverse_clause = INVERSE inverse_attr { inverse_attr } .
236 label = simple_id .
237 list_type = LIST [ bound_spec ] OF [ UNIQUE ] base_type .
238 literal = binary_literal | integer_literal | logical_literal | real_literal |
            string_literal .
239 local_decl = LOCAL local_variable { local_variable } END_LOCAL ';' .

240 local_variable = variable_id { ',' variable_id } ':' parameter_type
                    [ ':=' expression ] ';' .
241 logical_expression = expression .
242 logical_literal = FALSE | TRUE | UNKNOWN .
243 logical_type = LOGICAL .
244 multiplication_like_op = '*' | '/' | DIV | MOD | AND | '||' .
245 named_types = entity_ref | type_ref .
246 named_type_or_rename = named_types [ AS ( entity_id | type_id ) ] .
247 null_stmt = ';' .
248 number_type = NUMBER .
249 numeric_expression = simple_expression .

250 one_of = ONEOF '(' supertype_expression { ',' supertype_expression } ')' .
251 parameter = expression .
252 parameter_id = simple_id .
253 parameter_type = generalized_types | named_types | simple_types .
254 population = entity_ref .
255 precision_spec = numeric_expression .
256 primary = literal | ( qualifiable_factor { qualifier } ) .
257 procedure_call_stmt = ( built_in_procedure | procedure_ref )
                          [ actual_parameter_list ] ';' .
258 procedure_decl = procedure_head [ algorithm_head ] { stmt } END_PROCEDURE ';' .
259 procedure_head = PROCEDURE procedure_id [ '(' [ VAR ] formal_parameter
                   { ';' [ VAR ] formal_parameter } ')' ] ';' .

260 procedure_id = simple_id .
261 qualifiable_factor = attribute_ref | constant_factor | function_call |
                         general_ref | population .
262 qualified_attribute = SELF group_qualifier attribute_qualifier .
263 qualifier = attribute_qualifier | group_qualifier | index_qualilier .
264 query_expression = QUERY '(' variable_id '<*' aggregate_source '|'
                         logical_expression ')'.
265 real_type = REAL [ '(' precision_spec ')' ] .
266 referenced_attribute = attribute_ref | qualified_attribute .
```

```
267 reference_clause = REFERENCE FROM schema_ref [ '(' resource_or_rename
                       { ',' resource_or_rename } ')' ] ';' .
268 rel_op = '<' | '>' | '<=' | '>=' | '<>' | '=' | ':<>:' | ':=:' .
269 rel_op_extended = rel_op | IN | LIKE .

270 rename_id = constant_id | entity_id | function_id | procedure_id | type_id .
271 repeat_control = [ increment_control ] [ while_control ] [ until_control ] .
272 repeat_stmt = REPEAT repeat_control ';' stmt { stmt } END_REPEAT ';' .
273 repetition = numeric_expression .
274 resource_or_rename = resource_ref [ AS rename_id ] .
275 resource_ref = constant_ref | entity_ref | function_ref | procedure_ref | type_ref .
276 return_stmt = RETURN [ '(' expression ')' ] ';' .
277 rule_decl = rule_head [ algorithm_head ] { stmt } where_clause END_RULE ';' .
278 rule_head = RULE rule_id FOR '(' entity_ref { ',' entity_ref } ')' ';' .
279 rule_id = simple_id .

280 schema_body = { interface_specification } [ constant_decl ]
                  { declaration | rule_decl } .
281 schema_decl = SCHEMA schema_id ';' schema_body END_SCHEMA ';' .
282 schema_id = simple_id .
283 selector = expression .
284 select_type = SELECT '(' named_types { ',' named_types } ')' .
285 set_type = SET [ bound_spec ] OF base_type .
286 sign = '+' | '-' .
287 simple_expression = term { add_like_op term } .
288 simple_factor = aggregate_initializer | entity_constructor|
                    enumeration_reference | interval | query_expression |
                    ( [ unary_op ] ( '(' expression ')' | primary ) ) .
289 simple_types = binary_type | boolean_type | integer_type | logical_type |
                   number_type | real_type | string_type .

290 skip_stmt = SKIP ';' .
291 stmt = alias_stmt | assignment_stmt | case_stmt | compound_stmt | escape_stmt |
           if_stmt | null_stmt | procedure_call_stmt | repeat_stmt | return_stmt |
           skip_stmt .
292 string_literal = simple_string_literal | encoded_string_literal .
293 string_type = STRING [ width_spec ] .
294 subsuper = [ supertype_constraint ] [ subtype_declaration ] .
295 subtype_constraint = OF '(' supertype_expression ')' .
296 subtype_declaration = SUBTYPE OF ' (' entity_ref { ',' entity_ref } ')' .
297 supertype_constraint = abstract_supertype_declaration | supertype_rule .
298 supertype_expression = supertype_factor { ANDOR supertype_factor } .
299 supertype_factor = supertype_term { AND supertype_term } .

300 supertype_rule = SUPERTYPE subtype_constraint .
301 supertype_term = entity_ref | one_of | '(' supertype_expression ')' .
302 syntax = schema_decl { schema_decl } .
303 term = factor { multiplication_like_op factor } .
304 type_decl = TYPE type_id '=' underlying_type ';' [ where_clause ] END_TYPE ';' .
305 type_id = simple_id .
306 type_label = type_label_id | type_label_ref .
307 type_label_id = simple_id .
308 unary_op = '+' | '-' | NOT .
309 underlying_type = constructed_types | aggregation_types | simple_types |
                      type_ref .

310 unique_clause = UNIQUE unique_rule ';' { unique_rule ';' } .
311 unique_rule = [ label ':' ] referenced_attribute { ',' referenced_attribute } .
312 until_control = UNTIL logical_expression .
313 use_clause = USE FROM schema_ref [ '(' named_type_or_rename
                 { ',' named_type_or_rename } ')' ] ';' .
314 variable_id = simple_id .
```

315 where_clause = WHERE domain_rule ';' { domain_rule ';' } .
316 while_control = WHILE logical_expression .
317 width = numeric_expression .
318 width_spec = '(' width ')' [FIXED] .

А.3 Список перекрестных ссылок

Конструктив, указанный слева, используется в конструктивах, указанных справа.

ПРИЛОЖЕНИЕ В
(обязательное)

Определение разрешенных экземпляров объектов

В конкретном графе подтип/супертип может содержаться большое число типов данных сложных объектов и типов данных простых объектов, для которых могут создаваться экземпляры. Настоящее приложение показывает, как эти экземпляры идентифицируются при объявлении общего графа подтип/супертип.

Примечание — В качестве примера рассмотрим множество натуральных чисел [1, 2, 3 , ...]. Это множество может быть структурировано разными способами.

Четные и нечетные числа: [2, 4, 6, ...] и [1, 3, 5, 7, ...].

Простые числа: [2, 3, 5, 7, ...].

Числа, делимые на 3: [3, 6, 9, 12, ...].

Числа, делимые на 4: [4, 8, 12, 16, ...].

В терминах EXPRESS натуральные числа могут быть представлены как супертип, а другие множества чисел — как его подтипы. Очевидно, что некоторые из этих подтипов будут неперекрывающимися множествами (например, четные и нечетные числа), а другие будут перекрываться частично (например, подтип "числа делимые на 3" и подтип "числа делимые на 4").

B.1 Формальный подход

Любая группа типов данных объектов, связанная отношениями подтип/супертип, может быть рассмотрена как граф подтип/супертип, для которого могут создаваться экземпляры. В этом формальном подходе такие группы типов данных объектов называются частными типами данных сложных объектов. Частный тип данных сложного объекта может включать тип данных простого объекта, поскольку для типа данных простого объекта могут создаваться экземпляры. Частный тип данных сложного объекта обозначается через имена образующих его типов данных объекта, разделенных символом **&**. Частные типы данных сложных объектов должны комбинироваться для образования других типов данных сложных объектов. Частный тип данных сложного объекта может быть типом данных сложного объекта, но для того, чтобы в этом удостовериться, необходима полная оценка. Экземпляр сложного объекта является экземпляром типа данных сложного объекта и предметом описания в настоящем приложении.

Для любого частного типа данных сложного объекта справедливы следующие тождества:

- $A\&A \bullet\equiv A$

то есть конкретный тип данных объекта может появиться в заданном частном типе данных сложного объекта только один раз.

- $A\&B \equiv B\&A$

то есть группирование частных типов данных сложных объектов является коммутативным.

- $A\&(B\&C) \equiv (A\&B)\&C \bullet\equiv A\&B\&C$

то есть группирование частных типов данных сложных объектов является ассоциативным; в данном случае круглые скобки, задающие приоритет в вычислении, не влияют на результат.

Результирующее множество определяется как математическое множество частных типов данных сложных объектов, описанных именами частных типов данных сложных объектов, разделяемыми запятой ',' и заключенными в квадратные скобки. Пустое результирующее множество записывается как [].

Для комбинирования частного типа данных сложного объекта и результирующего множества могут быть использованы два оператора:

- $A+[B1,B2] \equiv\bullet\ [B1,B2]+A \bullet\equiv [A,B1,B2]$

Оператор + добавляет частный тип данных сложного объекта к результирующему множеству в качестве нового элемента этого множества. Один и тот же частный тип данных сложного объекта не должен присутствовать в одном результирующем множестве более одного раза.

- $A\&[B1, B2] \equiv\bullet\ [B1,B2]\&A \equiv\bullet\ [A\&B1,A\&B2]$

Оператор **&** добавляет частный тип данных сложного объекта ко всем типам данных сложных объектов в результирующем множестве. Следовательно, эта операция дистрибутивна над результирующими множествами.

Результирующие множества могут комбинироваться с использованием тех же двух механизмов:

- $[A1,A2] + [B1,B2] \equiv\bullet\ [A1,A2,B1,B2]$

Может быть сформировано результирующее множество, содержащее все элементы двух объединяемых множеств. Это операция является объединением двух множеств.

- $[A1,A2]\&[B1,B2] \equiv\bullet\ [A1\&B1,A1\&B2,A2\&B1,A2\&B2]$

Результирующее множество может быть сформировано путем повторного применения правила дистрибутивности для оператора & к каждому элементу первого результирующего множества над вторым результирующим множеством.

Результирующие множества могут быть отфильтрованы с помощью оператора/для 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, в которые входит заданный частный тип данных сложного объекта.

- [*A,A&B,A&C,A&B&D,B&G,D*]/[*B,D*] ≡• [*A&B,A&B&D,B&C,D*]

Новое результирующее множество может быть сформировано путем повторной фильтрации первого результирующего множества каждым частным типом данных сложного объекта из второго результирующего множества и последующей комбинации результатов с помощью оператора +.

Результирующие множества могут вычитаться с помощью оператора − для создания нового результирующего множества:

- [*A1,A2,B1,B2*]−[*A2,B1*] ≡• [*A1,B2*]

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 неупорядочены.

- [*A,A,B*] ≡• [*A,B*]

Конкретный тип данных сложного объекта может появиться в любом заданном результирующем множестве только один раз.

- [*A,*[*B,C*]] ≡• [*A,B,C*]

Результирующие множества могут быть вложенными.

В.2 Операторы супертипов

Используя вышеописанный формализм, можно переписать выражение супертипов EXPRESS в терминах результирующих множеств. При этом превращения, описанные в В.2.1—В.2.3, применяются рекурсивно до тех пор, пока не останется ни одного выражения супертипа (ONEOF, AND или ANDOR).

Эти превращения не могут полностью описать содержимое операторов супертипов, в частности, оператора ONEOF. Для этого требуется полный алгоритм, описанный в В.3.

В.2.1 ONEOF

Список ONEOF преобразуется в результирующее множество, содержащее выборки ONEOF, то есть:

ONEOF(*A, B, ...*) → [*A, B, ...*]

В.2.2 AND

Оператор AND эквивалентен оператору & и, действуя над частными типами данных сложных объектов или над результирующими множествами, создает частный тип данных сложного объекта или результирующее множество.

***A* AND *B* → [*A&B*]**

***A* AND ONEOF(*B1,B2*) → *A*&[*B1,B2*] = [*A&B1,A&B2*]**

ONEOF(*A1,A2*) AND ONEOF(*B1,B2*) → [*Al,A2*]&[*Bl,B2*] = [*A1&B1,A1&B2,A2&B1,A2&B2*]

В.2.3 ANDOR

Оператор ANDOR создает результирующее множество, содержащее каждый из операндов по отдельности, а операнды комбинируются с использованием оператора &. Оператор ANDOR действует над частными типами данных сложных объектов или над результирующими множествами.

***A* ANDOR *B* → [*A,B,A&B*]**

***A* ANDOR ONEOF(*B1,B2*) → [*A,*[*Bl,B2*],*A*&[*B1,B2*]] = [*A,Bl,B2,A&Bl,A&B2*]**

ONEOF(*A1,A2*) ANDOR ONEOF(*B1,B2*] → [[*A1,A2*],[*B1,B2*],[*A1,A2*]&[*Bl,B2*]] =
= [*A1,A2,B1,B2,A1&B1,A1&B2,A2&B1,A2&B2*]

В.2.4 Приоритет операторов

Вычисление результирующих множеств производится слева направо, операторы, имеющие высший приоритет, выполняются первыми в соответствии с 9.2.4.5.

Пример 154 — Нижеприведенное выражение вычисляется следующим образом:

A ANDOR *B* AND *C* → [*A,*[*B*&*C*],*A*&[*B*&*C*]] = [*A,B*&*C,A*&*B*&*C*]

В.3 Интерпретация возможных типов данных сложных объектов

Интерпретация выражения супертипов с добавлением информации, доступной из объявленной структуры, позволяет разработчику EXPRESS-схемы определить типы данных сложных объектов, экземпляры которых могут быть созданы, исходя из заданных объявлений. Для построения такого определения может быть создано результирующее множество типов данных сложных объектов для графа подтип/супертип. Для этого введем следующие определения:

Подтип с множественным наследованием (multiply inheriting subtype)**:** Подтипом с множественным наследованием является подтип, который имеет два или более супертипа в объявлении данного подтипа.

Корневой супертип (root supertype)**:** Корневым супертипом является супертип, не являющийся подтипом.

Результирующее множество ***R*** типов данных сложных объектов вычисляется в результате следующего процесса:

a) Определяются все объявления объектов, формирующие граф подтип/супертип.

Примечание 1 — В случае сложных графов подтип/супертип для этого может потребоваться несколько итераций.

b) Для каждого супертипа в графе подтип/супертип создается полное выражение супертипа, включая неявные ограничения ANDOR с подтипами, которые не включены в выражение супертипа.

c) Для каждого супертипа ***i*** в графе подтип/супертип создается результирующее множество, представляющее ограничения между подтипами данного супертипа, путем применения преобразований из В.2 и тождеств из В.1, чтобы получить полное выражение супертипа, как это задано на шаге (b). Для комбинирования ***i*** с результатом используется оператор &. Если ***i*** не является абстрактным супертипом, ***i*** добавляется к результату с помощью оператора +. Назовем это множество E_i.

d) Для каждого корневого супертипа ***r*** в графе подтип/супертип расширяется множество E_r следующим образом:

1) Для каждого подтипа ***s*** из ***r*** заменяется каждое его вхождение в E_r (включая вхождение в состав частных типов данных сложных объектов) на E_s, если это возможно, и применяются преобразования из В.2.

2) Рекурсивно повторяется шаг (d1) для каждого ***s***, расширяя подтипы ***s*** до тех пор, пока не будут достигнуты листьевые объекты (для которых не существует E_s).

Примечание 2 — Эта рекурсия должна завершиться, поскольку в графе подтип/супертип отсутствуют циклы.

e) Комбинируются корневые множества. Создается $R = \Sigma\bullet_r E_r \equiv\bullet\ E_{r1} + E_{r2} + \ldots$, то есть ***R*** является объединением множеств, полученных на шаге (d)

f) Для каждого подтипа с множественным наследованием ***m*** выполняется следующее:

1) Для каждого из его непосредственных супертипов ***s*** создается множество ***R/m/s***, содержащее строго те сложные типы данных из ***R***, которые включают и ***m***, и ***s***.

2) Создается результирующее множество комбинаций супертипов, разрешенных для ***m***, P_m = ***R/m/s1&R/m/s2& ...***, то есть комбинируются результирующие множества, полученные на шаге (f1) с использованием оператора &.

3) Создается результирующее множество комбинаций супертипов, которое может не включать все супертипы ***m***, $X_m = \Sigma\bullet_s R/m/s$, то есть объединяются результирующие, полученные на шаге (f1).

4) Присваивается $R = (R - X_m) + P_m$.

g) Для каждого выражения супертипа ***k***, имеющего вид **ONEOF(*S*$_1$, *S*$_2$,)**, выполняется следующее:

1) Для каждой пары подвыражений S_i, управляемых ***k*** ($i < j$), вычисляется множество комбинаций, запрещенных оператором **ONEOF(*S*$_i$,*S*$_j$)**: $D_k^{i,j} = [S_i \& S_j]$. Выполняется преобразование $D_k^{i,j}$ в соответствии с преобразованиями из В.2 и тождествами из В.1.

2) Задается множество $D_k = \Sigma\bullet_{i,j} D_k^{i,j}$, то есть D_k является объединением множеств, вычисленных на шаге (g1).

3) Присваивается $R = R - (R/D_k)$.

h) Для каждого выражения супертипа ***k*** вида S_1 **AND** S_2 выполняется следующее:

1) Вычисляется множество требуемых комбинаций, задаваемых k, $Q_k = [S_1 \& S_2]$. Преобразуется Q_k в соответствии с преобразованиями из В.2 и тождествами из В.1.

2) Для каждого типа данных объекта ***i***, названного в ***k***, вычисляется множество недопустимых комбинаций объектов, содержащих ***i***, которые запрещены ***k***, $D_k^i = R/i - R/(Q_k/i)$.

3) Создается множество $D_k = \Sigma\bullet_i D_k^i$, то есть D_k является объединением множеств, вычисленных на шаге (h2).

4) Присваивается $R = R - D_k$.

i) Конечное результирующее множество ***R*** является результирующим множеством для исходного графа подтип/супертип.

Пример 155 — В данном примере заданы только объявления супертипов и подтипов объектов, так как это вся информация, которая необходима для интерпретации возможных типов данных сложных объектов.

```
SCHEMA example;
ENTITY p SUPERTYPE OF (ONEOF(m, f) AND ONEOF(c, a));
ENTITY m SUBTYPE OF (p);
ENTITY f SUBTYPE OF (p);
ENTITY c SUBTYPE OF (p);
ENTITY a ABSTRACT SUPERTYPE OF (ONEOF (l, i)) SUBTYPE OF (p);
ENTITY l SUBTYPE OF (a);
ENTITY i SUBTYPE OF (a);
END_SCHEMA;
```

Это представлено EXPRESS-G-диаграммой на рисунке. В.1.

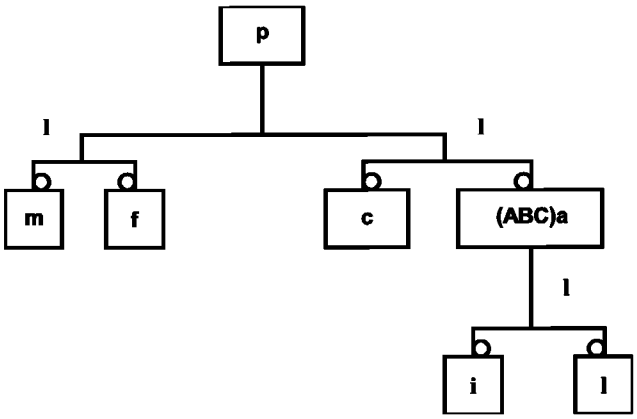


Рисунок В.1 — EXPRESS-G-диаграмма для схемы из примера 155

Возможные типы данных сложных объектов могут быть определены следующим образом.

Вышеприведенный текст EXPRESS уже дает нам все объявления объектов и полные выражения супертипов, как это требуется на шагах a) и b).

Применение шага c) дает:

$$E_p \rightarrow [p\&m\&c,\ p\&m\&a,\ p\&f_2\&c,\ p\&f_2\&a,\ p]$$
$$E_a \rightarrow [a\&l,\ a\&i]$$

Применение шага d) расширяет объявления корневых объектов **p**. Результирующим множеством является:

$$E_p = [p\&m\&c,\ p\&m\&a\&l,\ p\&m\&a\&i,\ p\&f_2\&c,\ p\&f_2\&a\&l,\ p\&f_2\&a\&i,\ p]$$

Комбинация корневых множеств на шаге e) дает:

$$R = [p\&m\&c,\ p\&m\&a\&l,\ p\&m\&a\&i,\ p\&f_2\&c,\ p\&f_2\&a\&l,\ p\&f_2\&a\&i,\ p]$$

Подтипы с множественным наследованием отсутствуют, поэтому шага f) не требуется.

Применение шага g) к каждому ограничению ONEOF дает:

- ONEOF(m,f):

$$D_1^{1,2} = [m\&f]$$
$$D_1 = [m\&f]$$

Удаляя D_1 из R в соответствии с шагом g3), оставляем R неизменным. Следовательно, мы остаемся с:

$$R = [p\&m\&c,\ p\&m\&a\&l,\ p\&m\&a\&i,\ p\&f_2\&c,\ p\&f_2\&a\&l,\ p\&f_2\&a\&i,\ p]$$

- ONEOF(c,a):

$$D_1^{1,2} = [c\&a]$$
$$D_2 = [c\&a]$$

Удаляя D_2 из R в соответствии с шагом g3), оставляем R неизменным. Следовательно, мы остаемся с:

$$R = [p\&m\&c,\ p\&m\&a\&l,\ p\&m\&a\&i,\ p\&f_2\&c,\ p\&f_2\&a\&l,\ p\&f_2\&a\&i,\ p]$$

- ONEOF(1, i):

$$D_3^{1,2} = [l\&i]$$
$$D_3 = [l\&i]$$

Удаляя D_3 из R в соответствии с шагом g3), оставляем R неизменным. Следовательно, мы остаемся с:

$$R = [p\&m\&c,\ p\&m\&a\&l,\ p\&m\&a\&i,\ p\&f_2\&c,\ p\&f_2\&a\&l,\ p\&f_2\&a\&i,\ p]$$

Применение шага h) к каждому ограничению AND дает:

- ONEOF(m,f) AND ONEOF(c,a):

$$Q_1 = [m\&c,\ m\&a,\ f_2\&c,\ f_2\&a]$$
$$D_1^m = []$$
$$D_1^f = []$$
$$D_1^c = []$$
$$D_1^a = []$$
$$D_1 = []$$

Удаляя D_1 из R в соответствии с шагом h4), оставляем R неизменным. Следовательно, мы остаемся с:

$$R = [p\&m\&c,\ p\&m\&a\&l,\ p\&m\&a\&i,\ p\&f_2\&c,\ p\&f_2\&a\&l,\ p\&f_2\&a\&i,\ p]$$

В соответствии с шагом i) результатом является:

$$R = [p\&m\&c,\ p\&m\&a\&l,\ p\&m\&a\&i,\ p\&f_2\&c,\ p\&f_2\&a\&l,\ p\&f_2\&a\&i,\ p]$$

Этот пример, будучи произвольным, может стать более реалистичным, если дать объектам более описательные имена. Например, если вместо **p, m, f, c, a, l** и **i** назвать объекты соответственно **person** (личность), **male** (мужчина), **female** (женщина), **citizen** (гражданин), **alien** (иностранец), **legal_alien** (легальный иностранец), и **illegal_alien** (нелегальный иностранец).

При использовании такой интерпретации прочтение некоторых элементов из окончательного результирующего множества дает:

- личность **(person);**
- 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 **...**

Пример 156 — Этот пример показывает, что ONEOF является глобальным ограничением, которое не может быть переопределено множественным наследованием.

```
SCHEMA diamond;
ENTITY a SUPERTYPE OF (ONEOF(b, c));
ENTITY b SUPERTYPE OF (d) SUBTYPE OF (a);
ENTITY c SUPERTYPE OF (d) SUBTYPE OF (a);
ENTITY d SUBTYPE OF (b, c);
END_SCHEMA;
```

Это представлено EXPRESS-G-диаграммой на рисунке В.2.

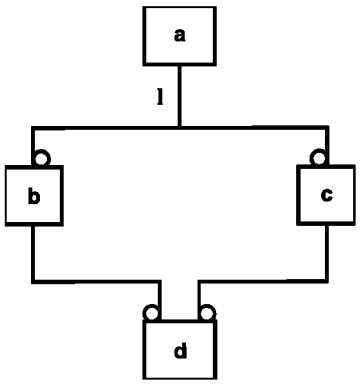


Рисунок В.2 — EXPRESS-G-диаграмма для схемы из примера 156

Возможные типы данных сложных объектов могут быть определены следующим образом:

Вышеприведенный текст EXPRESS уже дает нам все объявления объектов и полные выражения супертипов, как это требуется на шагах a) и b).

Применение шага c) дает:

$$E_a \rightarrow [a\&b,\ a\&c,\ a]$$
$$E_b \rightarrow [b\&d,\ b]$$
$$E_c \rightarrow [c\&d,\ c]$$
$$E_d \rightarrow [d]$$

Применение шага d) расширяет объявления корневых объектов **a**. Результирующим множеством является:

$$E_a = [a\&b\&d,\ a\&b,\ a\&c\&d,\ a\&c,\ a]$$

Комбинация корневых множеств на шаге e) дает:

$$R = [a\&b\&d,\ a\&b,\ a\&c\&d,\ a\&c,\ a]$$

Применение шага f) к каждому подтипу с множественным наследованием дает следующие результаты:

- Для объекта *d*:

$$C_d{}^b = [a\&b\&d]$$
$$C_d{}^c = [a\&c\&d]$$
$$P_d = [a\&b\&d\&c]$$
$$X_d = [a\&b\&d,\ a\&c\&d]$$

Новым множеством $R = (R - X_d) + P_d$ тогда является: $[a\&b,\ a\&c,\ a,\ a\&b\&d\&c]$.

Применение шага g) к каждому ограничению ONEOF дает:

- ONEOF(b,c):

$$D_1{}^{1,2} = [b\&c]$$
$$D_1 = [b\&c]$$

Удаление D_1 из R в соответствии с шагом g3) удаляет из R следующие элементы: $[a\&b\&d\&c]$.

Таким образом, мы остаемся с:

$$R = [a\&b,\ a\&c,\ a]$$

Выражения супертипов, использующие оператор AND, отсутствуют, поэтому шаг h) не требуется.
В соответствии с шагом i) результатом является:

$$R = [a\&b,\ a\&c,\ a]$$

Пример 157 — Этот пример показывает влияние применения ограничений к сложной структуре, содержащей по меньшей мере один из возможных типов ограничений. Этот пример не является моделью какой-либо полезной концепции и используется исключительно для демонстрации алгоритма.

```
SCHEMA complex;
ENTITY a SUPERTYPE OF (ONEOF(b, c) AND d ANDOR f);
ENTITY b SUBTYPE OF (a);
ENTITY c SUBTYPE OF (a);
ENTITY d ABSTRACT SUPERTYPE OF (ONEOF(k, l)) SUBTYPE OF (a);
ENTITY f SUBTYPE OF (a, z);
ENTITY k SUBTYPE OF (d);
ENTITY l SUBTYPE OF (d, y);
ENTITY x SUBTYPE OF (z);
ENTITY y SUPERTYPE OF (l) SUBTYPE OF (z);
ENTITY z SUPERTYPE OF (f ANDOR x ANDOR y);
END_SCHEMA;
```

Это представлено с помощью EXPRESS-G-диаграммы на рисунке В.3.

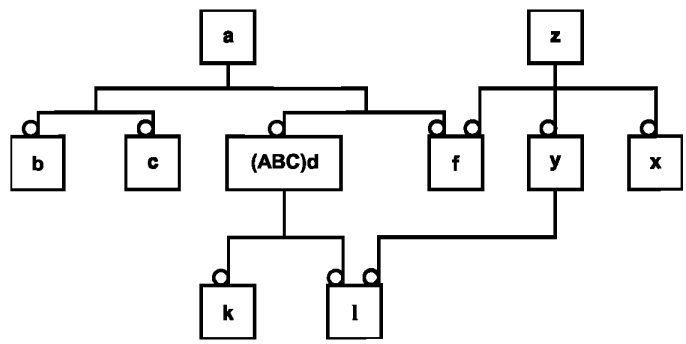


Рисунок В.3 – EXPRESS-G-диаграмма для схемы из примера 157

Возможные типы данных сложных объектов могут быть определены следующим образом:
Вышеприведенный текст EXPRESS уже дает все объявления объектов и полные выражения супертипов, как это требуется на шагах a) и b).
Применение шага c) дает:

$$E_a \rightarrow [a,\ a\&b\&d,\ a\&b\&d\&f,\ a\&c\&d,\ a\&c\&d\&f,\ a\&f]$$
$$E_d \rightarrow [d\&k,\ d\&l]$$
$$E_y \rightarrow [l\&y,\ y]$$
$$E_z \rightarrow [f\&x\&y\&z,\ f\&x\&z,\ f\&y\&z,\ f\&z,\ x\&y\&z,\ x\&z,\ y\&z,\ z]$$

Применение шага d) расширяет объявления корневых объектов **a, z**. Результирующими множествами являются:

$$E_a = [a,\ a\&b\&d\&k,\ a\&b\&d\&l,\ a\&b\&d\&f\&k,\ a\&b\&d\&f\&l,\ a\&c\&d\&k,\ a\&c\&d\&l,$$
$$a\&c\&d\&f\&k,\ a\&c\&d\&f\&l,\ a\&f]$$
$$E_z = [f\&l\&x\&y\&z,\ f\&l\&y\&z,\ f\&x\&y\&z,\ f\&x\&z,\ f\&y\&z,\ f\&z,\ l\&x\&y\&z,\ l\&y\&z,$$
$$x\&y\&z,\ x\&z,\ y\&z,\ z]$$

Комбинация корневых множеств на шаге e) дает:

$$R = [a,\ a\&b\&d\&k,\ a\&b\&d\&l,\ a\&b\&d\&f\&k,\ a\&b\&d\&f\&l,\ a\&c\&d\&k,\ a\&c\&d\&l,$$
$$a\&c\&d\&f\&k,\ a\&c\&d\&f\&l,\ a\&f,\ f\&l\&x\&y\&z,\ f\&l\&y\&z,\ f\&x\&y\&z,\ f\&x\&z,\ f\&y\&z,$$
$$f\&z,\ l\&x\&y\&z,\ l\&y\&z,\ x\&y\&z,\ x\&z,\ y\&z,\ z]$$

Применение шага f) к каждому подтипу с множественным наследованием *f* дает следующие результаты:
для объекта f:

$$C_f^a = [a\&b\&d\&k\&f,\ a\&b\&d\&l\&f,\ a\&c\&d\&k\&f,\ a\&c\&d\&l\&f,\ a\&f]$$
$$C_f^z = [f\&l\&x\&y\&z,\ f\&l\&y\&z,\ f\&x\&y\&z,\ f\&x\&z,\ f\&y\&z,\ f\&z]$$

$$P_f = [a\&b\&d\&f\&k\&z,\ a\&b\&d\&f\&k\&x\&z,\ a\&b\&d\&f\&k\&l\&y\&z,\ a\&b\&d\&f\&k\&y\&z,$$
$$a\&b\&d\&f\&k\&l\&x\&y\&z,\ a\&b\&d\&f\&k\&x\&y\&z,\ a\&b\&d\&f\&l\&z,\ a\&b\&d\&f\&l\&x\&z,$$

a&b&d&f&l&y&z, a&b&d&f&l&x&y&z, a&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&z]

Новым множеством $R = (R\text{-}X) + P$ тогда является: [*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&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]

Новым множеством $R = (R\text{-}X_l) + P_l$ тогда является: [*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]

Применение шага g) к каждому ограничению ONEOF дает:
- ONEOF(b,c):

$D_1^{1,2}$ = [*b&c*]
D_1 = [*b&c*]

Удаляя D_1 из R в соответствии с шагом (g3), удаляем из R 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]

Таким образом, мы остаемся с:

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):

$$D_2^{1,2} = [k\&l]$$
$$D_2 = [k\&l]$$

Удаляя D_2 из R в соответствии с шагом g3), удаляем из R 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 остаемся с:
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&kc&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*]

Применение шага h) к каждому ограничению AND дает:
- ONEOF(b,c) ANDd:

$$Q_1 = [b\&d,\ c\&d]$$
$$D_1^b = []$$
$$D_1^c = []$$
$$D_1^d = []$$
$$D_1 = []$$

Удаляя D_1 из R в соответствии с шагом h4), оставляем R неизменным. Следовательно, мы остаемся с:
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&c&d&f&k&l&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&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*]

В соответствии с шагом i) результат является:
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&c&d&f&k&l&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&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

Когда импортируются сложные графы подтип/супертип, допустимые типы данных сложных объектов вычисляются с использованием расширений правил, описанных в разделе 11 и в приложении В. За счет того, что определяются только те объекты, которые необходимы в данной схеме, граф подтип/супертип, определенный в одной или нескольких других схемах, может быть усечен для использования в данной схеме.

В настоящем приложении заданы правила, необходимые для разложения графов подтип/супертип в том случае, если один или несколько типов данных объектов, присутствующих в исходных графах, не импортируются. Такие пропущенные типы данных объектов оставляют пустоты в выражениях супертипов. В

настоящем приложении такие пустоты обозначаются как <>. Для удаления пустот из выражения супертипа используются следующие преобразования:

- ONEOF(*A*,<>, ...) → ONEOF(*A*, ...)
- ONEOF(<>)→ <>
- ONEOF(*A*) → *A*
- *A* AND <> → ONEOF(*A*, *A*)
- *A* ANDOR <> → *A*

Обработка оператором AND гарантирует, что те типы данных объектов, которые в исходной схеме обязательно должны комбинироваться, не будут присутствовать в данной схеме (обеспечивается оператором ONEOF(A, A)), если сочетаемые с ними типы данных объектов не импортированы.

Результирующее множество допустимых типов данных сложных объектов для схемы, связанной с другими схемами, вычисляется по следующему алгоритму:

a) Создается полный пул объектов для данной схемы. Полный пул объектов состоит из:

1) всех объектов, определенных в данной схеме;

2) всех объектов, импортированных в данную схему с помощью спецификаций USE и REFERENCE;

3) всех объектов, неявно импортированных в данную схему.

Примечание 1 — Полный пул объектов может содержать несколько объектов с одинаковым именем (в случае их неявного импортирования из разных схем) или может включать один и тот же объект под разными именами (когда этот объект был переименован при импорте посредством выражения USE FROM ... AS). В первом случае пул будет содержать каждый объект, имеющий идентичное имя. Во втором же случае пул будет содержать только один объект, несмотря на одновременное наличие у него нескольких имен.

b) Для каждого супертипа из пула объектов сокращают выражение супертипа, удаляя все ссылки на объекты, отсутствующие в пуле объектов. Повторно применяют описанные выше преобразования для удаления образовавшихся пустот и получают правильное выражение супертипа, ссылающееся только на объекты из пула объектов.

c) Для каждого супертипа из пула объектов выписывают полное выражение супертипа, включающее неявные ограничения ANDOR с подтипами, не указанными в выражении супертипа.

Примечание 2 — Неявные ограничения ANDOR включают как подтипы, добавленные в схемах, отличных от объявленной схемы, так и подтипы из объявленной схемы, которые не вошли в выражение супертипа.

d) Вычисляют результирующее множество в соответствии с алгоритмом, описанным в приложении B, начиная с шага c).

Для типа данных сложного объекта из результирующего множества, который содержит по крайней мере один локально объявленный или импортированный с помощью спецификации USE объект, могут быть созданы независимые экземпляры. Для типа данных сложного объекта, не содержащего таких объектов, независимые экземпляры в данной схеме не могут быть созданы.

Примечание 3 — Если имеется явно импортированный объект, который не присутствует ни в каком типе данных сложного объекта из полученного результирующего множества, для такого объекта экземпляры не могут быть созданы вообще. Это означает ошибку при импорте данного объекта.

Примеры:

158 — Схема **example** (см. пример 155 из приложения B) используется для демонстрации алгоритма.

```
SCHEMA test;
USE FROM example (l);
REFERENCE FROM example (m);
END_SCHEMA;
```

Возможные типы данных сложных объектов определяются следующим образом:

Пул объектов включает объекты *l, m, a, p*: *l* и *m* импортированы явно, *a* и *p* импортированы неявно, поскольку они входят в цепочку супертипов объекта *l*.

Усечение выражения супертипа для *p* и его преобразование (шаг b) дает:

ONEOF(*m*, *f*) AND ONEOF(*c*, *a*)
ONEOF(*m*, <>) AND ONEOF(<>, *a*)
ONEOF(*m*) AND ONEOF(*a*)
m AND *a*

Подобным же образом находим для *a*:

ONEOF(*l*, *i*)
ONEOF(*l*, <>)
ONEOF(*l*)
l

В этом случае выражения супертипа уже полные, как это требуется для шага c).

Применение алгоритма результирующего множества на шаге d) дает результирующее множество *R* = [*p*, *a&l&m&p*]

Тип данных сложного объекта *a&l&m&p* содержит объект *l*, явно импортированный с помощью спецификации USE, и следовательно для него могут быть созданы независимые экземпляры. С другой стороны, для объекта *p* в данной схеме независимые экземпляры созданы быть не могут.

159 — Предположим доступность следующих схем:

```
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 этих схем являются:
s1 [*e*1, *e*1&*e*11, *e*1&*e*12, *e*1&*e*11&*e*12]
s2 [*e*1&*f*, *e*1&*f*&*e*211, *e*1&*f*&*e*212, *e*1&*f*&*e*211&*e*212]
s3 [*e*1&*g*, *e*1&*g*&*e*321, *e*1&*g*&*e*322, *e*1&*g*&*e*321&*e*322]

Если схема **test** описывается, как приведено ниже:

```
SCHEMA test;
USE FROM s2 (e211);
USE FROM s3 (e322);
END_SCHEMA;
```

Возможные типы данных сложных объектов для нее определяются следующим образом:

Пул объектов состоит из *e*211, *e*322, *f*, *g*, *e*1: *e*211 и *e*322 импортированы явно; *f*, *g* и *e*1 импортированы неявно, поскольку они входят в цепочку супертипов *e*211 и *e*322. Объекты *f* и *g* являются переименованными *e*11 и *e*12 соответственно, так что *e*11 и *e*12 являются действительными членами пула объектов.

Усечение выражения супертипа для *e*1 и его преобразование (шаг b) дают:

*e*11 ANDOR *e*12
f ANDOR *g*

для *f* находим:

*e*211 ANDOR *e*212
*e*211 ANDOR <>
*e*211

для *g* находим:

*e*321 ANDOR *e*322
<> ANDOR *e*322
*e*322

В этом случае выражения супертипов уже являются полными, как требуется для шага c).

Применение алгоритма результирующего множества на шаге d) дает результирующее множество *R* = = [*e*l, *e*l&*f*, *e*l&*g*, *e*l&*f*&*g*, *e*l&*f*&*e*211, *e*l&*f*&*g*&*e*211, *e*l&*g*&*e*322, *e*l&*f*&*g*&*e*322, *e*l&*f*&*g*&*e*211&*e*322]. Типы данных сложных объектов *e*l&*f*&*e*211, *e*l&*f*&*g*&*e*211, *e*l&*g*&*e*322, *e*1&*f*&*g*&*e*322, *e*1&*f*&*g*&*e*211&*e*322 содержат один из объектов — *e*211 или *e*322, импортированных явно с помощью спецификации USE, и поэтому для них могут быть созданы независимые экземпляры. С другой стороны, в данной схеме не могут быть созданы независимые экземпляры для *e*l, *e*1&*f*, *e*l&*g*, *e*l&*f*&*g*.

ПРИЛОЖЕНИЕ D
(обязательное)

EXPRESS-G. Графическое подмножество EXPRESS

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 следующее:

- различные уровни абстракции данных;
- диаграммы, размещаемые более чем на одной странице;
- диаграммы, использующие минимум компьютерных графических средств, включая использование только неграфических символов.

Нотация EXPRESS-G представлена графическими символами, образующими диаграмму. В нотации используется три типа символов:

определения — символы, обозначающие простые типы данных, поименованные типы данных, сконструированные типы данных и объявления схем;

отношения — символы, описывающие отношения, существующие между определениями;

компоновки — символы, позволяющие размещать диаграммы более чем на одной странице.

EXPRESS-G поддерживает простые типы данных, поименованные типы данных, отношения и количество элементов (множества). Также EXPRESS-G поддерживает нотации одной или нескольких схем. Поддержка механизма ограничений, задаваемых средствами языка EXPRESS, в EXPRESS-G отсутствует.

Примечание — EXPRESS-G может использоваться как самостоятельный язык определения данных, то есть не требует наличия соответствующей спецификации на языке EXPRESS.

Пример 160 — На рисунках D.1 и D.2 показаны EXPRESS-G-диаграммы для простой схемы на языке EXPRESS из примера 171 (см. приложение H). Диаграмма разбита на несколько страниц, чтобы показать, как создаются многостраничные диаграммы.

Личность (**person**) имеет некоторые определяющие ее характеристики, включая фамилию, имя, необязательное прозвище имя, дату рождения и описание волос. Личность является мужчиной (**male**) или женщиной (**female**). Мужчина может иметь жену — женщину, в этом случае женщина имеет мужа — мужчину. Личность может иметь детей, которые также являются личностями.

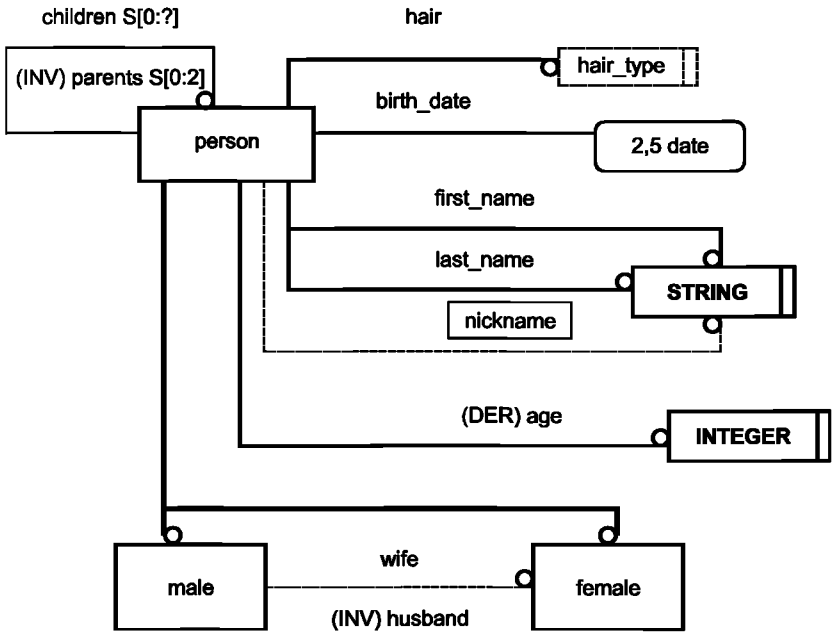


Рисунок D.1 — Полная диаграмма уровня объекта для примера 171 (страница 1 из 2)

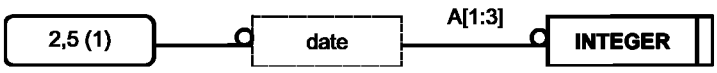


Рисунок D.2 — Полная диаграмма уровня объекта для примера 171 (страница 2 из 2)

D.2 Символы определения

Определения типов данных и схем в диаграмме обозначаются прямоугольниками, содержащими внутри имя определяемого элемента. Отношения между элементами обозначаются линиями, соединяющими прямоугольники. Для различных видов определений и отношений используются различные стили линий.

D.2.1 Символы для простых типов данных

Символ для обозначения простого типа данных EXPRESS представляет собой сплошной прямоугольник с вертикальной двойной чертой справа. Имя типа данных заключено в прямоугольник, как это показано на рисунке D.3.

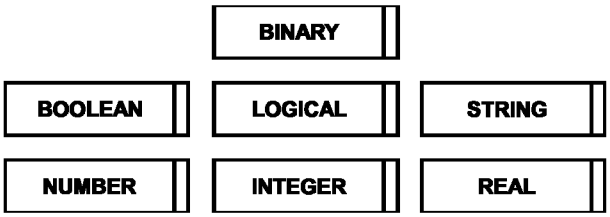


Рисунок D.3 — Символы для простых типов данных EXPRESS

D.2.2 Символы для сконструированных типов данных

Символы для сконструированных типов данных EXPRESS, выбираемого типа SELECT и перечисляемого типа ENUMERATION представляют собой пунктирные прямоугольники, в которые заключено имя типа данных, как это показано на рисунке D.4.

Рисунок D.4 — Символы для сконструированных типов данных EXPRESS

Символ для типа данных SELECT типа состоит из пунктирного прямоугольника с двойной вертикальной линией слева.

Символ для типа данных ENUMERATION типа состоит из пунктирного прямоугольника с двойной вертикальной линией справа. EXPRESS-G не обеспечивает представления списка элементов перечисления.

Примечание — Поскольку простые типы данных и тип данных ENUMERATION являются в EXPRESS-G атомарными типами данных, символ, обозначающий тип данных ENUMERATION, сходен с символом, обозначающим простой тип данных, который также имеет вторую вертикальную черту справа.

EXPRESS допускает использование типов данных SELECT и ENUMERATION только для представления определенного типа данных. EXPRESS-G предоставляет сокращенную нотацию, когда имя определенного типа данных пишется внутри пунктирного прямоугольника, представляющего типы SELECT или ENUMERATION, вместо имени типа данных, а символ определенного типа данных не дается, как это показано на рисунке D.5 (см. D.5.4).

Рисунок D.5 — Сокращенные символы для сконструированных типов данных EXPRESS, используемые для представления определенных типов

Пример 161 — Две диаграммы на рисунке D.5 эквивалентны.

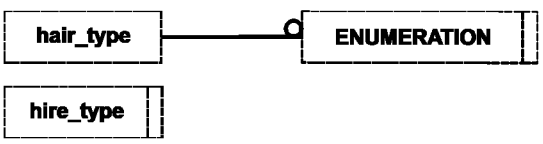


Рисунок D.6 — Пример альтернативных методов представления типа ENUMERATION

Реализация инструментальных средств редактирования EXPRESS-G может использовать полную форму представления сконструированных типов данных, сокращенную форму или обе формы. Разработчик инструментальных средств редактирования EXPRESS-G должен указать, какая из этих форм применяется, используя приложение Е.

D.2.3 Символы для определенных типов данных

Символ для определенного типа данных представляет собой пунктирный прямоугольник, в который заключено имя типа, как это показано на рисунке D.7.

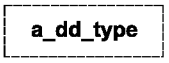


Рисунок D.7 — Символ для определенного типа данных EXPRESS

D.2.4 Символы для типов данных объектов

Символ для типа данных объекта представляет собой сплошной прямоугольник, в который заключено имя объекта, как это показано на рисунке D.8.

Рисунок D.8 — Символ для типа данных объекта в EXPRESS

D.2.5 Символы для функций и процедур

EXPRESS-G не поддерживает никакой нотации ни для обозначения функций (FUNCTION), ни для обозначения процедур (PROCEDURE).

D.2.6 Символы для правил

EXPRESS-G не поддерживает нотацию для определения правил (RULE). Имена объектов, являющихся параметрами правил, могут быть помечены звездочкой (см. D.5.3).

D.2.7 Символы для схем

Символ для схемы (рисунок D.9) представляет собой сплошной прямоугольник с именем схемы в верхней половине прямоугольника, отделенный от нижней половины горизонтальной чертой. Нижняя половина прямоугольника остается пустой.

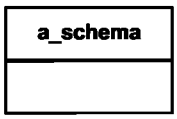


Рисунок D.9 — Символ для схемы

D.3 Символы отношений

Символы определений соединяются линиями различных стилей, как это показано на рисунке D.10.

Рисунок D.10 — Стили линий, обозначающих отношения

Отношение для необязательного атрибута типа данных объекта представляется пунктирной линией.

Ссылка между схемами представляется пунктирной линией. Отношение наследования (то есть отношение между подтипом и супертипом) представляется толстой линией. Все прочие отношения представляются сплошными линиями нормальной толщины.

Отношения являются двунаправленными, но одно из направлений выделяется. Если объект А имеет явный атрибут, который является объектом В, то выделенным является направление от А к В. В EXPRESS-G отношения отмечаются пустым кружком в выделенном направлении, в данном случае в конце линии со стороны В. Для отношения наследования выделенным направлением является направление к подтипу, то есть кружок располагается на конце линии со стороны подтипа.

Пример 162 — Направления отношений показаны на рисунке D.11, который является неполным отражением кода EXPRESS из примера 172 приложения Н. Диаграмма содержит шесть типов данных объектов, три определенных типа данных и несколько простых типов данных. Объект **super** имеет два подтипа, названные **sub_1** и **sub_2**. Объект **sub_2** имеет атрибут выбираемого типа данных, названный **choice**, выбираемый ме-

жду типом данных объекта, названным **an_ent,** и определенным типом данных **name**. Тип данных объекта **an_ent** имеет в качестве атрибута целочисленный тип данных, а **name** является строковым типом данных.

Тип данных объекта **sub_1** имеет в качестве атрибута тип данных объекта **from_ent**. Тип данных объекта **from_ent** в качестве необязательного атрибута имеет тип данных объекта **to_ent**, а в качестве обязательного атрибута — действительный тип данных. В свою очередь, тип данных объекта **to_ent** в качестве обязательного атрибута имеет определенный тип данных, названный **strings**, а **strings** является списком (не показанным на диаграмме) строкового типа данных.

Примечания

1 Хотя в приведенной диаграмме показаны только прямые линии отношений, линии могут иметь любую конфигурацию (например, быть скругленными).

2 Не всегда удается построить диаграмму, избежав при этом взаимного пересечения линий отношений. Задача распознавания и разводки линий связей в точках их пересечения возлагается на разработчика диаграммы.

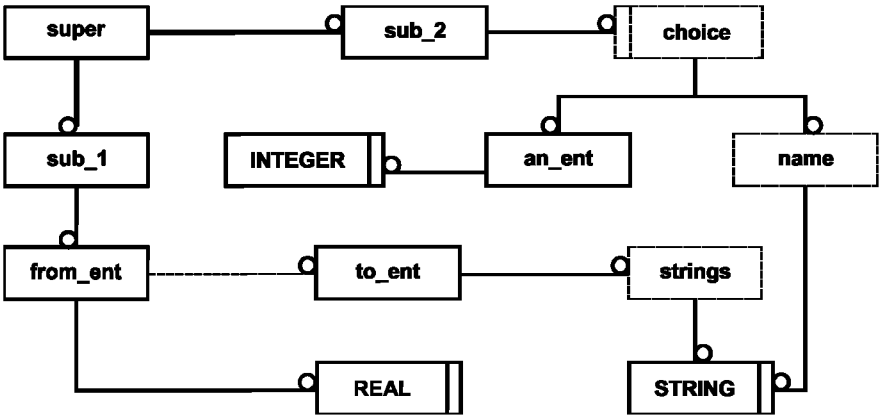


Рисунок D.11 — Частная диаграмма уровня объекта, иллюстрирующая направления отношений из примера 172 (страница 1 из 1)

D.4 Символы компоновки

Графические представления могут располагаться на нескольких страницах. Каждая страница должна быть пронумерована. Символы, обеспечивающие ссылки между страницами, приведены на рисунке D.12.

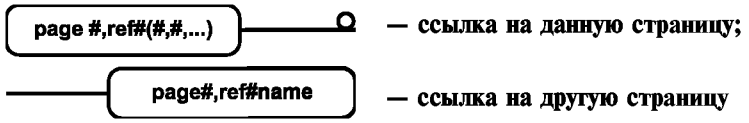


Рисунок D.12 — Символы компоновки: страничные ссылки

Схема может содержать ссылки на определения из других схем. Символы, обеспечивающие ссылки между схемами, приведены на рисунке D.13.

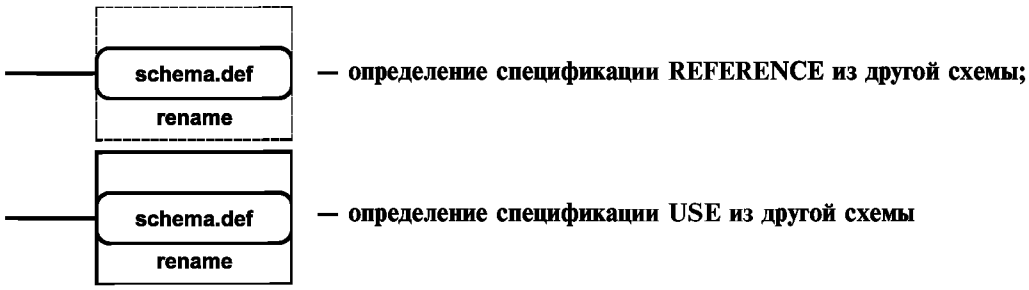


Рисунок D.13 — Символы компоновки: ссылки между схемами

D.4.1 Страничные ссылки

Когда существует отношение между определениями на разных страницах, линия отношения на каждой странице оканчивается скругленным прямоугольником. Скругленный прямоугольник должен содержать

номер страницы и номер ссылки на странице, как показано на рисунке D.12. Номером страницы является номер страницы, содержащей определение, на которое дается ссылка. Номер ссылки используется для различения множества ссылок на одну и ту же страницу. На той странице, откуда исходит ссылка, символ компоновки должен содержать имя определения, на которое дана ссылка. На той странице, на которую дана ссылка, символ компоновки может содержать заключенный в круглые скобки список страниц, с которых исходят ссылки.

Примечание — Использование страничных ссылок показано на рисунках D.1 и D.2. Скругленный прямоугольник, помеченный 2,5, исходящий от определения **person**, означает, что определение, на которое дается ссылка, находится на странице 2 диаграммы как ссылка номер 5. На странице 2 диаграммы, как показано на рисунке. D.2, текст в скругленном прямоугольнике указывает, что на данное определение ссылка сделана из определения, расположенного на другой странице диаграммы. Число один (1) в круглых скобках показывает, что определение, из которого исходит ссылка, расположено на первой странице диаграммы.

D.4.2 Ссылки между схемами

Ссылки между схемами обозначаются скругленным прямоугольником, содержащим имя определения, квалифицированное именем схемы, как показано на рисунке D.13.

Определения, доступные из другой схемы через спецификацию REFERENCE, заключают в пунктирный прямоугольник. Если определение переименовано при импорте, новое имя может быть написано в прямоугольнике под внутренним скругленным прямоугольником.

Определения, доступные из другой схемы через спецификацию USE, заключают в сплошной прямоугольник. Если определение переименовано при импорте, новое имя может быть написано в прямоугольнике под внутренним скругленным прямоугольником.

Примечание — Использование ссылок между схемами показано на рисунке D.17.

D.5 Диаграммы уровня объекта

EXPRESS-G может быть использован для представления определений и отношений между ними в пределах одной схемы. Компоненты такой диаграммы включают простые типы данных, определенные типы данных, типы данных объектов и символы отношений, вместе с соответствующей информацией о ролях и количестве элементов, представляя содержимое одной схемы.

D.5.1 Имена ролей

В EXPRESS атрибут типа данных объекта именуется для указания роли типа данных, на которую ссылаются, когда экземпляр участвует в отношении, установленном атрибутом. Строка текста с именем роли может быть размещена над линией отношения, соединяющей символ типа данных объекта с символом его атрибута. Подобные имена ролей должны быть согласованы с правилами области действия и видимости, приведенными в разделе 10.

D.5.2 Количество элементов

Атрибуты объектов и определенные типы данных могут быть представлены агрегатными типами данных (то есть ARRAY, BAG, LIST и SET). В EXPRESS-G тип агрегации указывают около линии отношения для атрибута сразу за именем атрибута. Используется только первая буква агрегатного типа данных (A, B, L или S), а ключевое слово OF опускается. Если тип агрегации не указан, то количество элементов равняется единице для обязательного отношения и нулю или единице — для необязательного атрибута.

Примечание — На рисунке. D.14 приведена полная EXPRESS-G-диаграмма для текста на языке EXPRESS из примера 172. Для компонентов выбираемого типа имена ролей не даны.

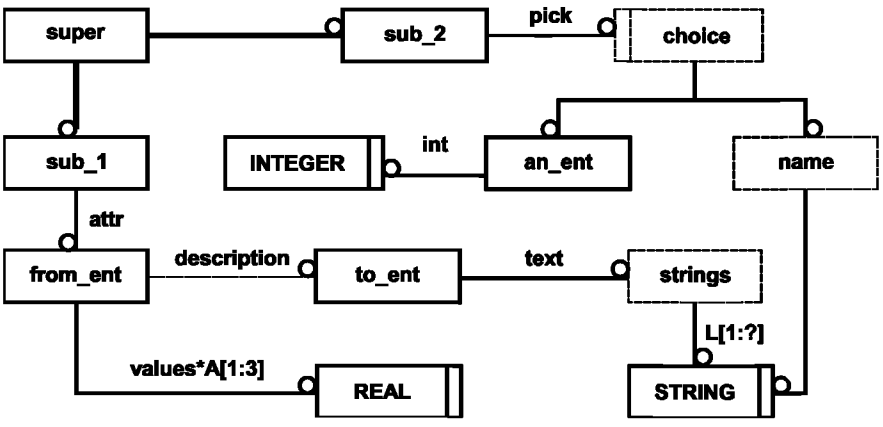


Рисунок D.14 — Полная диаграмма уровня объекта для примера 172 (страница 1 из 1)

D.5.3 Ограничения

Кроме механизма задания ограничений на количество элементов, EXPRESS-G не имеет никаких других механизмов для задания ограничений. Тот факт, что на элемент наложены некоторые ограничения, может быть обозначен применением звездочки (*), предшествующей имени элемента. Применяют следующие правила:

- если объект является параметром в глобальном правиле RULE, перед именем объекта может ставиться звездочка;
- если атрибут объекта ограничен предложением UNIQUE или предложением WHERE внутри объекта, перед именем атрибута может ставиться звездочка;
- если определенный тип данных ограничен предложением WHERE, перед именем определенного типа данных может ставиться звездочка;
- если агрегатный тип данных ограничен ключевым словом UNIQUE, перед первым символом, обозначающим агрегатный тип, может ставиться звездочка.

D.5.4 Сконструированные и определенные типы данных

Тип данных SELECT представляется символом выбираемого типа данных (см. рисунок D.4) плюс отношение и определение типа данных для каждого выбираемого элемента. Никаких количеств элементов или имен ролей для отношений не указывается.

Тип данных ENUMERATION полностью представляется своим символом (см. рисунок D.4).

Примечание 1 — EXPRESS-G не обеспечивает механизм для записи элементов перечисления.

Определенный тип данных представляется символом определения типа (см. рисунок D.7), внутри которого записывается имя определения, определением представляемого типа данных и линией отношения, направленной от определения определенного типа данных к определению представляемого типа данных. Над линией отношения может быть указано количество элементов представления.

Примечание 2 — Представление определенного типа данных можно увидеть на типе **strings** на рисунке D.14.

D.5.5 Типы данных объекта

Для определений объектов в EXPRESS-G используется символ сплошного прямоугольника (см. рисунок D.8). Имя типа данных объекта записывается внутри прямоугольника.

В EXPRESS-G объект может:

- быть частью ациклического графа наследования;
- иметь явные атрибуты;
- иметь вычисляемые атрибуты;
- иметь инверсные атрибуты;

Каждый явный или вычисляемый атрибут в EXPRESS-объекте задает отношение в соответствующей EXPRESS-G-диаграмме. Имя роли атрибута вместе с следующим за именем количеством элементов могут быть надписаны над линией отношения. Вычисляемый атрибут отличается от явного атрибута тем, что перед именем вычисляемого атрибута в круглых скобках пишутся символы DER, то есть (DER).

В случае, когда для атрибута определен инверсный атрибут, имя инверсного атрибута и количество его элементов пишутся с противоположной стороны линии связи по отношению к имени атрибута, для которого он является инверсным. Перед именем инверсного атрибута в круглых скобках пишутся символы INV, то есть (INV).

Примечания

1 Типовые диаграммы уровня объекта показаны на рисунках D.1 и D.14.

2 Обозначения правил областей значений, применяемых для атрибутов, можно увидеть на рисунке D.1 на примере ролей **husband** и **maiden_name**.

3 Пример объектов, ограниченных правилами, показан на рисунке D.1 для объектов **male** и **female**.

Подтип/супертип

Объекты, формирующие граф наследования, соединяются толстыми сплошными линиями. Кружок ставится на конце линии отношения со стороны подтипа. Когда супертип является абстрактным супертипом, перед именем объекта внутри прямоугольного символа объекта в круглых скобках пишутся символы ABS, то есть (ABS).

EXPRESS-G обеспечивает ограниченную нотацию для обозначения логической структуры графа наследования. Отношение ONEOF может быть показано путем разветвления линии отношения, идущей от супертипа к каждому из подтипов, относящихся друг к другу как ONEOF. Около точки разветвления ставится цифра 1.

Примечания

4 На рисунке D.15, представляющем EXPRESS-G-диаграмму для примера 173, показано, что объект **sub2** является абстрактным супертипом.

5 На рисунке D.15 показано, что объекты **sub1**, **sub2** и **sub5** являются подтипами супертипа **super**. Логические отношения между этими подтипами (то есть AND или ANDOR) никак не обозначены. Экземпляр **super** может не иметь подтипов, поскольку **super** не является абстрактным супертипом. Объекты **sub3** и **sub4** являются подтипами супертипа **sub2**. Объекты **sub3** и **sub4** друг с другом находятся в отношении ONEOF.

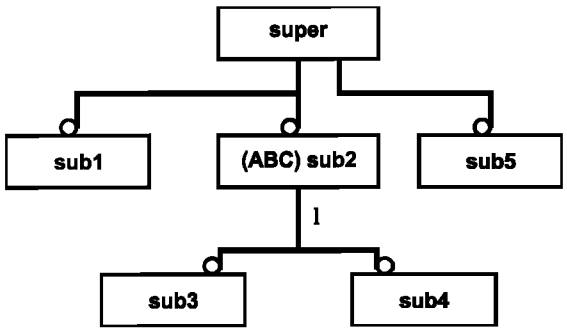


Рисунок D.15 — Полная диаграмма уровня объекта для графа наследования из примера 173 (страница 1 из 1)

EXPRESS дает возможность переобъявлять в подтипе атрибутов супертипа, представляя переобъявленный атрибут в конкретизации типа атрибута супертипа. В EXPRESS-G переобъявленный атрибут представляется таким же способом, как и атрибут супертипа, но с добавлением перед именем атрибута символов RT (redeclared type — переобъявленный тип), заключенных в круглые скобки, то есть (RT).

Примечание 6 — На рисунке D.16 показана одна из форм переобъявления атрибута, заданная в тексте на языке EXPRESS из примера 174. Объект **sub_a** переобъявляет атрибут **attr** из его супертипа в подтип атрибута его супертипа. Объект **sup_b** имел необязательный атрибут типа NUMBER. В подтипе данный атрибут переопределен как обязательный атрибут типа REAL.

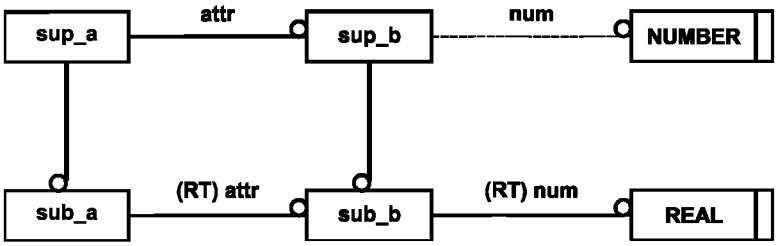


Рисунок D.16 — Полная диаграмма уровня объекта для примера 174, показывающая переобъявления атрибутов в подтипах (страница 1 из 1)

D.5.6 Ссылки между схемами

Когда определение из данной схемы ссылается на определение из другой схемы, используется символ ссылки между схемами, содержащий имя определения, на которое сделана ссылка, квалифицированное именем схемы.

Примечание — На рисунке D.17 показана диаграмма уровня объекта единичной схемы. Исходный текст EXPRESS для этой диаграммы приведен в примере 175. Полная диаграмма состоит из двух схем — **top** и **geom** (см. рисунок D.18), а некоторые объекты схемы **top** имеют атрибуты, использующие определения из схемы **geom**. Поскольку диаграмма уровня объекта состоит только из тех элементов, которые определены в одной схеме, в представлении схемы **top** в этом примере необходимо использование ссылок между схемами.

D.6 Диаграммы уровня схемы

Диаграмма уровня схемы состоит из представления нескольких схем и интерфейсов между ними.

Содержимое EXPRESS-G-диаграммы уровня схемы ограничено схемами, входящими в диаграмму, и интерфейсами между схемами. В диаграмму может быть включено следующее:

- схемы, которые ссылаются на другие схемы с помощью спецификации USE;
- схемы, которые ссылаются на другие схемы с помощью спецификации REFERENCE;
- имена элементов, на которые даны ссылки или которые используются в схемах.

Импорт с помощью спецификации USE представляется сплошной линией отношения нормальной толщины, направленной от схемы, использующей данный элемент, к схеме, содержащей данный элемент, оканчивающейся пустым кружком со стороны символа, обозначающего схему, содержащую данный элемент. Импорт с помощью спецификации REFERENCE представляется пунктирной линией отношения нормальной толщины, направленной от схемы, ссылающейся на данный элемент, к схеме, содержащей данный элемент, оканчивающейся пустым кружком со стороны схемы, содержащей данный элемент.

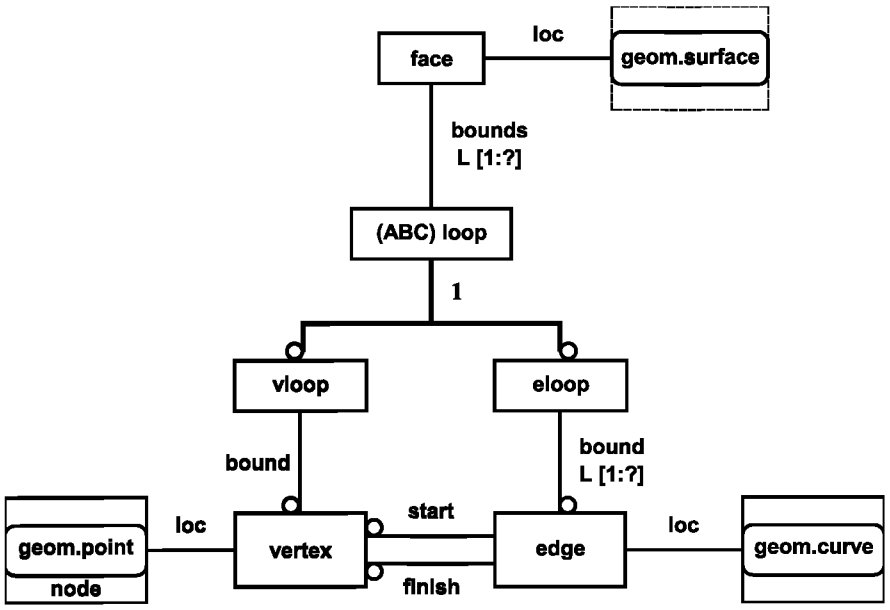


Рисунок D.17 — Полная диаграмма уровня объекта для схемы top из примера 175, показывающая ссылки между схемами (страница 1 из 1)

Определения, которые использованы в схеме (импортированы в нее) или на которые даны ссылки из данной схемы, могут быть показаны в виде списка имен, смежного с соответствующей линией отношения и соединенного с линией отношения линией, заканчивающейся стрелкой у линии отношения. Переименованное определение указывается после исходного имени определения вслед за знаком «больше» (>).

Примечание 1 — Диаграмма с двумя схемами показана на рисунке D.18. Схема **top** имеет интерфейс со схемой **geom**. В частности, схема **top** ссылается на определение **surface** (поверхность) и импортирует (использует) определения **curve** (кривая) и **point** (точка) из схемы **geom**. Определение **point** в схеме **top** переименовано в **node** (узел).

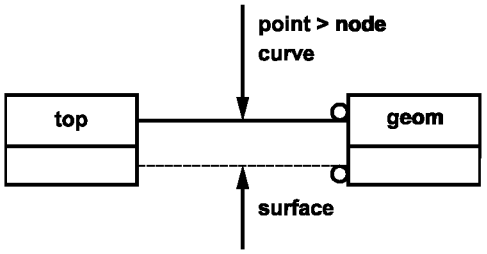


Рисунок D.18 — Полная диаграмма уровня схемы для примера 175 (страница 1 из 1)

Если диаграмма уровня схемы размещается на нескольких страницах, а схемные интерфейсы пересекают границы страниц, тогда используются символы страничных ссылок.

Примечание 2 — В примере 176 задан исходный текст на языке EXPRESS для сокращенной версии диаграммы уровня схемы. Диаграма EXPRESS-G схемы для этого примера показана на рисунке D.19.

D.7 Полные диаграммы EXPRESS-G

В полной диаграмме EXPRESS-G должны быть точно представлены в пределах ограничений нотации EXPRESS-G все определения, отношения и ограничения, используемые в диаграмме уровня объекта или уровня схемы.

D.7.1 Полная диаграмма уровня объекта

Диаграммы, представляющие полную диаграмму одной схемы, должны иметь содержание, определяемое следующими правилами:

a) Каждая страница должна иметь заголовок, начинающийся со слов: “Complete entity level diagram of ...” (Полная диаграмма уровня объекта для ...).

b) Каждая страница должна быть пронумерована в виде «Page X of N» (Страница X из N), где N — общее число страниц, образующих диаграмму, а X — номер данной страницы.

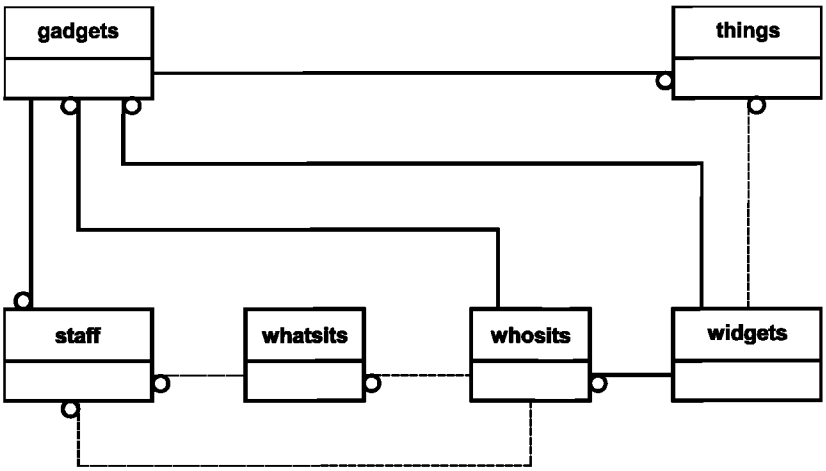


Рисунок D.19 — Полная диаграмма уровня схемы для примера 176 (страница 1 из 1)

c) Должны быть показаны все типы данных объектов, определенные типы данных и простые символы, используемые в данной схеме.
d) Символы схем не должны применяться.
e) Должны быть показаны все отношения, имена атрибутов и количества элементов.
f) Должны быть показаны все атрибуты, включая явные, вычисляемые и инверсные атрибуты.
g) Должны быть показаны все отношения наследования (то есть отношения подтип/супертип).
h) Должны быть помечены все абстрактные (ABSTRACT) супертипы.
i) Должны быть помечены все подтипы, находящиеся в отношениях ONEOF.
j) Все определения из других схем, используемые в данной схеме или на которые даны ссылки из данной схемы, должны быть представлены символами скругленных прямоугольников, заключенных в прямоугольники соответствующего стиля (сплошные — для используемых определений, а пунктирные — для определений, на которые даны ссылки из данной схемы).
k) Любое переименование должно быть представлено в соответствующем символе ссылки между схемами.
l) Все объекты, ограниченнные правилом (RULE), должны быть помечены звездочкой (*).
m) Все атрибуты, на которые наложены ограничения, должны быть помечены звездочкой (*).
n) Все определенные типы, на которые наложены ограничения, должны быть помечены звездочкой (*).
o) Все агрегатные типы, на которые наложены ограничения, должны быть помечены звездочкой (*).

Все отношения объект — объект, не помеченные инверсными атрибутами, должны интерпретироваться как отношения с количеством элементов, равным нулю или более. Нет логического структурирования, которое может быть выведено из отношения немаркированного подтипа, исключая случай, когда данное отношение не является отношением ONEOF.

D.7.2 Полная диаграмма уровня схемы

Диаграммы, представляющие полную диаграмму уровня схемы, должны иметь содержание, определяемое следующими правилами:

a) Каждая страница должна иметь заголовок, начинающийся со слов: “Complete schema level diagram of ... ” (Полная диаграмма уровня схем для ...).
b) Каждая страница должна быть пронумерована в виде «Page X of N» (Страница X из N), где N — общее число страниц, образующих диаграмму, а X — номер данной страницы.
c) Должны быть показаны все использованные схемы.
d) Не должны быть показаны простые символы и символы объектов и типов.
e) Должны быть показаны все отношения схема — схема, представленные спецификациями USE и REFERENCE.
f) Имена всех определений, которые используются или на которые даны ссылки, вместе с любыми их перенаименованиями должны быть привязаны к соответствующей линии связи. Если к линии связи не привязано никаких имен, это интерпретируется как схема объекта, используемая целиком, или на нее дается ссылка в целом.

Примечание — При разработке моделей или изображаемых диаграмм полезно иметь возможность изображать диаграммы на разных уровнях абстракции. Например, на диаграмме могут быть заданы не все атрибуты или показаны не все имена ролей. Такой подход вне области действия EXPRESS-G, но рекомендуется, чтобы уровень абстракции был согласован и документирован до начала разработки. Помимо того, рекомендуется, чтобы в заголовке диаграммы были отражены используемые уровни абстракции.

ПРИЛОЖЕНИЕ Е
(обязательное)

Заявка о соответствии реализации протоколу (ЗСРП)

Является ли реализация синтаксическим анализатором/верификатором языка EXPRESS? Если да, то должны быть даны ответы на вопросы, приведенные в Е.1.

Является ли реализация инструментальным средством редактирования EXPRESS-G? Если да, то должны быть даны ответы на вопросы, приведенные в Е.2.

Е.1 Синтаксический анализатор языка EXPRESS

Для какого из уровней заявляется поддержка:

☐ уровень 1 — проверка ссылок;

☐ уровень 2 — проверка типов;

☐ уровень 3 — проверка значений;

☐ уровень 4 — полная проверка.

Примечание — Для того чтобы заявить о поддержке конкретного уровня, должна быть обеспечена поддержка всех нижних уровней.

Каково максимальное целочисленное значение [integer_literal]?: .....:

Какова максимальная точность действительных значений [real_literal]?: .....:

Каков максимальный показатель степени действительных значений [real_literal]?: .....:

Какова максимальная длина строки (в символах) [simple_string_literal]?: .....:

Какова максимальная длина строки (в восьмибитовых байтах) [encoded_string_literal]?: .....:

Какова максимальная длина двоичных значений (в битах) [binary_literal]?: .....:

Существует ли ограничение на общее количество объявленных уникальных идентификаторов? Если да, то чему оно равно?: .....:

Существует ли ограничение на количество символов в идентификаторе? Если да, то чему оно равно?: .....:

Существует ли ограничение на глубину вложения областей действия? Если да, то чему оно равно?: .....:

Реализуется ли концепция многоименных областей действия, в которых могут появляться имена схем? Если да, то как называются эти области действия?: .....:

Как представлена стандартная константа '?' [built_in_constant]?: .....:

Е.2 Инструментальное средство редактирования EXPRESS-G

Для какого из уровней заявляется поддержка:

☐ уровень 1 — проверка символов;

☐ уровень 2 — полная проверка.

Примечание — Для того чтобы заявить о поддержке конкретного уровня, должна быть обеспечена поддержка всех нижних уровней.

Существует ли ограничение на общее количество объявленных уникальных идентификаторов? Если да, то чему оно равно?: .....:

Существует ли ограничение на количество символов в идентификаторе? Если да, то чему оно равно?: .....:

Существует ли ограничение на количество символов на страницу модели? Если да, то чему оно равно?: .....:

Существует ли ограничение на количество страниц в модели? Если да, то чему оно равно?: .....:

Реализуется ли концепция многоименных областей действия, в которых могут появляться имена схем? Если да, то как называются эти области действия?: .....:

Реализуется ли полная форма представления сконструированных типов данных, сокращенная форма или обе формы?: .....:

ПРИЛОЖЕНИЕ F
(обязательное)

Регистрация информационного объекта

Для того чтобы обеспечить однозначную идентификацию информационного объекта в открытой системе, настоящему стандарту присвоен идентификатор объекта:

{ iso standard 10303 part(11) version(2) }

Смысл этого значения определен в соответствии с ИСО/МЭК 8824-1 и уточнен в ГОСТ Р ИСО 10303-1.

ПРИЛОЖЕНИЕ G
(справочное)

Отношения

G.1 Отношения через атрибуты

В языке EXPRESS объявление в типе данных объекта атрибута, областью значения которого является другой тип данных, явно устанавливает отношение между этими двумя типами данных. На это отношение можно ссылаться как на простую связь, устанавливающую связь между экземпляром объявленного объекта и одним из экземпляров представленного типа данных.

Для того чтобы охарактеризовать отношения, устанавливаемые атрибутами, имеющими агрегатные значения, определяют фундаментальный основной тип данных типа данных как неагрегатный тип данных, заданный:

- фундаментальным основным типом данных неагрегатного типа данных, являющегося самим типом данных;
- фундаментальным основным типом данных агрегатного типа данных, являющегося фундаментальным основным типом данных этого основного типа.

Когда фундаментальным основным типом атрибута **A** является **T**, будем говорить, что **A** основан на **T**.

Следовательно, объявление в типе данных объекта атрибута, областью значений которого является агрегатный тип данных, основанный на фундаментальном основном типе данных, устанавливает два вида отношений:

- коллективное отношение между объявляемым объектом и агрегатным типом данных. Тем самым связывается экземпляр объявленного объекта с множеством экземпляров фундаментального основного типа;
- распределенное отношение между объявляемым объектом и фундаментальным основным типом. Тем самым индивидуально связывается экземпляр объявляемого объекта с одним или несколькими экземплярами фундаментального основного типа.

Примечание — Данный подход отличается от подхода, принятого в некоторых других языках моделирования. Например, в модели объект—отношение (Entity-Relationship — ER) объекты и отношения моделируются различными конструктивами.

Как простое, так и распределенное отношения направлены от объявляемого объекта к некоторому другому типу данных. Такой подход полезен при рассмотрении количества элементов в данных отношениях (с точки зрения объявляемого объекта). Количество элементов будет следующим. Если данным количеством элементов является **m** : **n** (где $0 \le$ **m** $\le$ **n**), каждый экземпляр объявленного объекта связывает не менее **m** и не более **n** экземпляров заданного типа данных. Если **n** имеет неопределенное (?) значение, то максимальное число экземпляров заданного типа данных, с которыми может быть связан экземпляр объявляемого объекта, не ограничено.

Данный подход полезен при рассмотрении инверсного отношения, которое имеет направление, обратное простому или распределенному отношению. Такое отношение неявно существует всегда, и по умолчанию число элементов в нем равно 0 : ?. Оно может быть явно поименованным и необязательно ограниченным инверсным атрибутом (INVERSE), объявленным в представляемом типе данных, если представляемый тип данных является типом данных объекта.

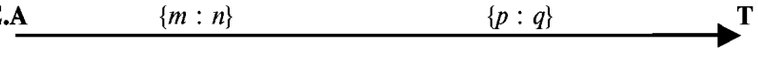
Пример 163 — В этом примере существует простое отношение между типами данных объектов **first** и **second**, в котором **second** играет роль **ref**. Количество элементов в этом отношении с точки зрения **first** всегда будет **1 : 1** (то есть каждый экземпляр объекта **first** связан строго с одним экземпляром объекта **second**).

Относительно объекта **second** количество элементов в отношении будет **0 : ?**, или неограниченным (то есть один экземпляр объекта **second** может быть связан с нулем или более экземплярами объекта **first**), это является количеством элементов в инверсном отношении, принимаемым по умолчанию.

```
ENTITY first;
       ref   : second;
       fattr : STRING;
END_ENTITY;

ENTITY second;
       sattr : STRING;
END_ENTITY;
```

Если тип данных объекта **E** имеет отношение с типом данных **T**, установленное через атрибут **A**, это отношение может быть изображено как:



при $0 \le m \le n$ и $0 \le p \le q$. Здесь $m : n$ является количеством элементов в прямом отношении от **E** к **T**, в то время как $p : q$ является количеством элементов в обратном отношении от **T** к **E**.

Ниже формально описаны три вида отношений и соответствующие им количества элементов.

G.1.1 Простое отношение

Простое отношение — отношением, устанавливаемое атрибутом, представление которого является другим типом данных объекта. Эти отношения устанавливаются между двумя типами данных объектов.

Простое отношение всегда существует между экземпляром объявляемого объекта и не более чем одним экземпляром представляющего объекта. На диаграмме это может быть показано следующим образом:

E.A $\{m : 1\}$ ——————— $\{p : q\}$ **T**

при $0 \le m \le 1$ и $0 \le p \le q$.

Это означает, что для каждого экземпляра **E** роль **A** не играется ни одним экземпляром **T** или же играется строго одним экземпляром **T**. Для каждого экземпляра **T** существует от p до q экземпляров **E**, в которых **T** играет роль **A**.

Следующие случаи значений p и q являются значащими классами ограничений, накладываемых на простые отношения между **E** и **T**:

- если $q = 1$, то существует ограничение, заключающееся в том, что экземпляр **T** не может играть роль **A** более чем в одном экземпляре **E**;
- если $1 \le p$, то существует ограничение, наложенное на **T**. То есть для каждого экземпляра **T** должно существовать по меньшей мере p (но не больше чем q) экземпляров **E**, использующих этот экземпляр **T** в роли **A**.

Для ограничения числа элементов простого отношения и соответствующего ему инверсного отношения используют несколько различных конструкций EXPRESS:

- случай $m = 0$ обеспечивается объявлением необязательного (OPTIONAL) атрибута **A**. Если **A** не объявлен как OPTIONAL, $m = 1$;
- случай $q = 1$ обеспечивается объявлением простого инверсного атрибута или применением к **E.A** правила уникальности, которое требует, чтобы каждая роль **A** в совокупности **E** использовалась для различных экземпляров, то есть конкретный экземпляр T может быть использован не более чем одним **E.A**;
- другие ограничения на количество элементов в инверсном отношении выражаются путем объявления в **T** инверсного (INVERSE) атрибута в виде **INVERSE I : SET [p : q] OF E FOR A**. Случай, когда $p = q = 1$, может быть сокращенно записан как **INVERSE I : E FOR A**.

Ниже даны примеры простых отношений и связанных с ними ограничений на количество элементов.

Примеры:

164 — **CIRCLE.CENTRE** {1 : 1} ——————— {0 : ?} **POINT**

Каждая окружность **CIRCLE** имеет строго одну точку **POINT**, играющую роль ее центра **CENTRE**. Каждая точка **POINT** может играть роль центра в произвольном числе окружностей (в том числе ни в одной). Это может быть объявлено так:

```
ENTITY point;
    ...
END_ENTITY;

ENTITY circle;
    centre : point;
    ...
END_ENTITY;
```

165 — **PRODUCT_VERSION.BASE_PRODUCT** {1 : 1} ——————— {1 : ?} **PRODUCT**

Каждая версия изделия **PRODUCT_VERSION** имеет строго одно изделие **PRODUCT**, играющее роль базового изделия **BASE_PRODUCT**. **PRODUCT** может играть роль **BASE_PRODUCT** в любом количестве **PRODUCT_VERSION**, но не менее чем в одном (существующая зависимость). Это может быть объявлено так:

```
ENTITY product_version;
    base_product : product;
    ...
END_ENTITY;

ENTITY product;
    ...
INVERSE
    versions : SET [1:?] OF product_version FOR base_product;
END_ENTITY;
```

166 — **PERSON.LUNCH** {0 : 1} ——————— {0 : ?} **MEAL**

Каждая личность **PERSON** может употреблять пищу **MEAL** в роли завтрака **LUNCH**. **MEAL** может играть роль **LUNCH** для любого числа личностей (предположим, что пищи достаточно много). Это может быть объявлено так:

```
ENTITY person;
    lunch : OPTIONAL meal;
    ...
END_ENTITY;

ENTITY meal;
    calories : energy_measure;
    amount : weight_measure;
    ...
END_ENTITY;
```

G.1.2 Коллективное отношение

Атрибут, имеющий агрегатное значение типа данных объекта, устанавливает коллективное отношение между типом данных объекта и агрегатным типом данных, используемым для представления атрибута.

Примечание — Коллективное отношение не распространяется на экземпляры объектов, из которых, в конечном счете, составлено агрегатное значение атрибута. Такие экземпляры участвуют в распределенном отношении (см. G.1.3).

Коллективное отношение сходно с простым отношением в случае неагрегатного значения атрибута. Коллективное отношение всегда существует между экземпляром объявляемого объекта и не более чем одним экземпляром представляющего агрегатного типа данных. Так же, как и в случае простого отношения, это может быть показано следующим образом:

E.A $\{m : 1\}$ $\{r : s\}$ **T**

при $0 \le m \le 1$ и $0 \le r \le s$.

Следующие случаи для значений r и s являются значащими классами ограничений, накладываемых на коллективные отношения между **E** и **T**:

- если $s = 1$, существует ограничение уникальности на коллективное значение атрибута **A**;
- если $1 \le r$, существует ограничение на существование **T**.

Также, как и в случае простого отношения, m управляется объявлением атрибута **A** необязательным ($m = 0$). Ограничение уникальности, когда $s = 1$, также как и в случае простого отношения, может быть охвачено написанием в объявлении **E** правила уникальности **A**. Иными способами r и s не могут быть ограничены.

Ниже даны примеры коллективных отношений и связанных с ними ограничений на количество элементов:

Примеры

167 — **POLY_CURVE.COEF** {1 : 1} {0 : ?} **LIST [1 : ?] OF REAL**

Каждая поликривая **POLY_CURVE** имеет список значений действительного типа, играющих роль коэффициентов **COEF**. Каждый список действительных значений **LIST [1 : ?] OF REAL**, может играть роль **COEF** в произвольном количестве **POLY_CURVE** (включая их отсутствие). Это может быть объявлено так:

```
ENTITY poly_curve;
    coef : LIST [1: ?] OF REAL;
    ...
END_ENTITY
```

168 — **LOOP.EDGES** {0 : 1} {0 : 1} **LIST [1 : ?] OF EDGES**

Каждый контур **LOOP** может иметь список ребер **EDGE**, играющий роль ребер **EDGES**. Каждый список ребер **LIST [1 : ?] OF EDGE** может играть роль **EDGES** не более чем в одном экземпляре **LOOP**. Это может быть объявлено так:

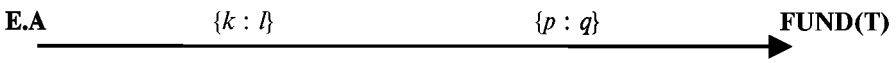
```
ENTITY loop;
    edges : OPTIONAL LIST [1 : ?] OF edge;
    ...
UNIQUE
    un1 : edges;
END_ENTITY;

ENTITY edge;
    ...
END_ENTITY;
```

G.1.3 Распределенное отношение

В дополнение к коллективному отношению, рассмотренному выше, атрибут, имеющий агрегатное значение, устанавливает распределенное отношение между типом данных объекта и фундаментальным основным типом агрегатного типа данных, используемым для представления атрибута.

Распределенное отношение индивидуально связывает экземпляр объявленного объекта с произвольным числом экземпляров представляющего фундаментального основного типа. Количество элементов данного отношения ограничено количеством элементов агрегатного типа (ов) данных, используемого для представления атрибута. Обозначив фундаментальный основной тип данных атрибута как **FUND(T)**, распределенное отношение можно показать как:



при $0 \le k \le l$ и $0 \le p \le q$.

Это означает, что для каждого экземпляра из **E** атрибут **A** состоит из экземпляров **FUND(T)**, заключенных между k и l. Экземпляр **FUND(T)** может появиться в роли **A** между экземплярами **E** от p до q.

Следующие случаи для значений p и q являются значащими классами ограничений, накладываемых на распределенное отношение между **E** и **FUND(T)**:

- если $q = 1$, существует ограничение, что один экземпляр **FUND(T)** не может появиться в роли **A** более чем в одном экземпляре **E**;
- если $1 \le p$, существует ограничение на существование **FUND(T)**. То есть для каждого экземпляра **FUND(T)** должно существовать по меньшей мере p (но не более q) экземпляров **E**, которые содержат экземпляр **FUND(T)** в роли **A**.

Для задания ограничений на количество элементов в распределенном отношении и в соответствующем ему инверсном отношении используются следующие конструкции EXPRESS:

- значения k и l задаются указанием границ агрегатного типа данных, используемого для представления **A**. В простейшем случае, тип данных атрибута будет просто **SET [k : l] OF FUND(T)** (или подобным образом могут быть записаны типы BAG или LIST).

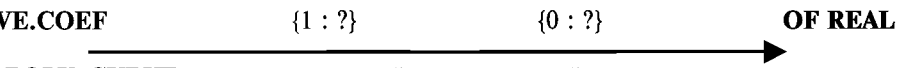
Примечание 1 — При таком подходе к отношениям не существует различия между одномерными и многомерными агрегатными значениями;

- случай $q = 1$ для распределенного отношения не может быть задан применением правила уникальности к атрибуту **E.A**. Вместо этого должен быть объявлен инверсный атрибут и наложено ограничение на **FUND(T)** в виде **INVERSE I : E FOR A**;
- другие ограничения на количество элементов в инверсном отношении выражаются путем объявления в **FUND(T)** инверсного атрибута в виде **INVERSE I : SET [p : q] OF E FOR A**. Случай, когда $p = q = 1$, может быть сокращенно записан как **INVERSE I : E FOR A**.

Ниже даны примеры распределенных отношений и связанных с ними ограничений на количество элементов.

Примеры

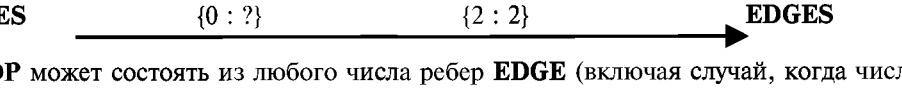
169 — В противоположность примеру 167.



Поликривая **POLY_CURVE** имеет по меньшей мере одно действительное число, играющее роль коэффициента **COEF**. Конкретное действительное число может быть использовано в атрибуте **COEF** в неограниченном количестве **POLY_CURVE** (включая случай, когда число не используется ни в одной поликривой). Это может быть объявлено так:

```
ENTITY poly_curve;
    coef : LIST [1:?] OF REAL;
    ...
END_ENTITY;
```

170 — Сравните этот пример с примером 168.



Контур **LOOP** может состоять из любого числа ребер **EDGE** (включая случай, когда число ребер равно нулю). **EDGE** должно использоваться строго в двух различных **LOOP**. Это может быть объявлено так:

```
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;
```

G.1.4 Инверсный атрибут

Каждому отношению, установленному атрибутом, соответствует неявное инверсное отношение. По умолчанию инверсное отношение игнорируется, то есть на него не могут делаться ссылки, и на его количество элементов ограничения не накладываются. Правило уникальности, наложенные на атрибут, объявляющий простое отношение, ограничивает количество элементов в инверсном отношении. EXPRESS предоставляет конструкции, позволяющие поименовать и ограничить инверсные отношения. Данные конструкции частично были описаны в пунктах, посвященных другим классам отношений, а в данном пункте дается их сводное описание.

Идентификатор инверсному отношению присваивается путем объявления инверсного (INVERSE) атрибута. Тип инверсного атрибута может ограничить количество элементов данного инверсного отношения.

Рассмотрим конкретные конструкции EXPRESS и их влияние на количество элементов в инверсном отношении. Допустим, что для объекта **E** объявлен атрибут **A** типа данных **T**. Если **T** является агрегатным типом данных, его фундаментальным основным типом будет **FUND(T)**. Представляющий объект (**FUND(T)**, или **T**, соответственно) обозначим **R**. Допустим, что инверсное отношение объявляется инверсным атрибутом **I** в **R**.

Если **A** является неагрегатным атрибутом, с которым связано правило уникальности, простое отношение ограничено таким образом, что среди совокупности **E** каждый **A** уникален. Следовательно, экземпляр **T** может играть роль **A** не более чем в одном экземпляре **E**, то есть. $q = 1$. Это эквивалентно записи **INVERSE I : SET [0 : 1] OF E FOR A**.

Если **A** является агрегатным атрибутом, с которым связано правило уникальности, на коллективное отношение накладывается ограничение $s = 1$. То есть экземпляр **T** (который является агрегатным значением) может играть роль **A** не более чем в одном экземпляре **E**. На распределенные отношения ограничений нет: экземпляр **R** может играть **A** в любом числе экземпляров **E**.

Если **I** объявлен как **INVERSE I : BAG [p:q] OF E FOR A**, то количество элементов в отношении, инверсном простому или распределенному отношению, ограничено соответствующими значениями p и q. То есть экземпляр **R** может играть роль **A** в экземплярах **E** от p до q. Поскольку мультимножество BAG допускает наличие экземпляров с эквивалентными элементами, конкретный экземпляр из **R** может несколько раз играть роль **A** в конкретном экземпляре из **E**. Это имеет смысл только в том случае, когда **T** является агрегатным типом данных, допускающим наличие повторяющихся элементов.

Если **I** объявлен как **INVERSE I : SET [p:q] OF E FOR A**, то количество элементов в отношении, инверсном простому или распределенному отношению, ограничено соответствующими значениями p и q. То есть экземпляр из **R** может играть роль **A** в экземплярах **E** от p до q. Поскольку набор SET не допускает наличия экземпляров с эквивалентными элементами, конкретный экземпляр из **R** может не играть данную роль более одного раза в конкретном экземпляре из **E**.

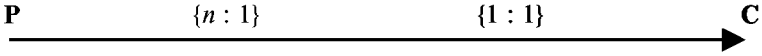
Если **I** объявлен как **INVERSE I : E FOR A**, это равносильно объявлению как **SET [1:1] OF E**. То есть каждый экземпляр из **R** должен играть роль **A** строго в одном экземпляре из **E**.

Любое объявление **I** как BAG или SET с $p \geq 1$ или отличным от BAG или SET, устанавливает ограничение на существование **R**: требуется, чтобы любой экземпляр из **R** играл роль **A** в, по меньшей мере, одном экземпляре из **E**.

G.2 Отношения подтип/супертип

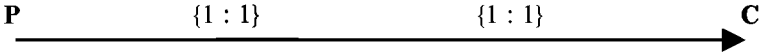
Объявление подтипа внутри объекта определяет отношение между объектом подтипа и объктами указанных супертипов.

Для заданного объекта супертипа **P**, имеющего подтип **C**, отношение может быть изображено следующим образом:



при $0 \leq n \leq 1$. Это означает, что для каждого экземпляра **P** существует ноль или один экземпляр **C**. Для каждого экземпляра **C** существует один экземпляр **P**.

В случае, когда P является абстрактным супертипом, отношение будет следующим:



Это означает, что для каждого экземпляра **P**, существует один экземпляр **C**. Для каждого экземпляра **C** существует один экземпляр **P**.

ПРИЛОЖЕНИЕ Н
(справочное)

Модели EXPRESS для иллюстрации примеров EXPRESS-G

В данном приложении приведены тексты на языке EXPRESS для нескольких примеров, использованных для иллюстрации моделирования с помощью EXPRESS-G.

Эти примеры не являются реальными или "правильными". В частности, модели из этих примеров не имеют никакого отношения к моделям из других стандартов серии ГОСТ Р ИСО 10303.

Н.1 Пример модели единичной схемы

Модель из примера 171 в основном показывает, что личность может быть мужчиной или женщиной. Каждая личность имеет определенные характеристики, такие как имя и фамилия, дата рождения, цвет волос, а также может иметь ноль или более детей (которые, конечно, также являются людьми). Мужчина может быть женат на женщине, в этом случае, женщина имеет инверсное отношение с мужчиной.

Возраст (**age**) личности является вычисляемым атрибутом, который вычисляется с помощью функции **years**, определяющей количество лет между датой рождения, введенной в качестве параметра, и текущей датой.

Личность (**person**) имеет инверсный атрибут, который связывает детей со своими родителями. Нижняя граница этого инверсного атрибута равна 0 на тот случай, когда невозможно обеспечить полное семейное дерево.

Примечание — Если бы существовал явный атрибут **parents** (родители), а атрибут **children** (ребенок) был бы инверсным атрибутом, семейное дерево необходимо было бы продлевать назад во времени до бесконечности.

Пример 171 — Модель единичной EXPRESS-схемы.

```
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
    ABSTRACT SUPERTYPE OF (ONEOF (female, male));
    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;

ENTITY female
    SUBTYPE OF (person);
INVERSE
    husband : SET [0 : 1] OF male FOR wife; -- муж не обязателен!
END_ENTITY;

ENTITY male
    SUBTYPE OF (person);
    wife : OPTIONAL female;
END_ENTITY;

FUNCTION years(past : date) : INTEGER:
    (* Данная функция подсчитывает число лет между предыдущей и текущей датами *)
END_FUNCTION;

END_SCHEMA;
```

Н.2 Шаблон отношения

В примере 172 приведена простая модель для того, чтобы показать некоторые из определений и отношений из EXPRESS. Модель содержит объекты супертипов, объекты подтипов и объекты, не являющиеся ни тем, ни другим. Также показаны два определенных типа данных, выбираемый тип и несколько простых типов.

Пример 172 — Простой объект из EXPRESS и модель отношений типа.

```
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;
```

Н.3 Простое дерево подтип/супертип

EXPRESS позволяет определять очень сложные деревья (и сети) подтипов/супертипов. Дерево, показанное в примере 173, относительно простое.

Пример 173 — Дерево подтип/супертип в языке EXPRESS.

```
SCHEMA simple_trees;

ENTITY super;
END_ENTITY;

ENTITY sub1
      SUBTYPE OF (super);
END_ENTITY;

ENTITY sub2
      ABSTRACT SUPER OF (ONEOF(sub3,
                               sub4))
      SUBTYPE OF (super);
END_ENTITY;

ENTITY sub3
      SUBTYPE OF (sub2);
END_ENTITY;
```

```
ENTITY sub4
    SUBTYPE OF (sub2);
END_ENTITY;

ENTITY sub5
    SUBTYPE OF (super);
END_ENTITY;

END_SCHEMA;
```

Н.4 Переобъявление атрибута

В EXPRESS допускается переобъявление наследуемых атрибутов, обеспечивающее совместимость новых типов атрибутов. В примере 174 показаны некоторые допустимые формы переобъявления:

- тип переобъявленного атрибута является подтипом наследуемого типа;
- тип переобъявленного атрибута является совместимым простым типом;
- значение переобъявленного атрибута является обязательным, в то время как наследуемое значение было необязательным.

Пример 174 — Переобъявление атрибута в 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;
```

Н.5 Многосхемные модели

Модели в EXPRESS состоят по меньшей мере из одной схемы. В примере 175 показана модель, состоящая из двух схем.

Пример 175 — Двухсхемная модель в EXPRESS.

```
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
        ABSTRACT SUPERTYPE OF;
                (ONEOF(eloop, vloop));
    END_ENTITY;

    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 задан в примере 176. Следует иметь ввиду, что внутри каждой из объявленных схем существуют объекты, типы и другие определения, которые здесь не показаны для экономии места.

Пример 176 — Многосхемная модель в 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 whostis;
    REFERENCE FROM stuff;
    REFERENCE FROM whatsits;
END_SCHEMA;
```

ПРИЛОЖЕНИЕ J
(справочное)

Библиография

[1] ИСО/МЭК 6429—92* Информационная технология. Представление числовых значений в строках символов для обмена информацией
[2] ИСО ТО/9007—87* Системы обработки информации. Концепции и терминология для концептуальной схемы и информационной базы
[3] 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
[4] 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

* Оригинал стандарта ИСО — во ВНИИКИ Госстандарта России.

Предметный указатель

УДК 656.072:681.3:006.354 ОКС 25.040.40 П87 ОКСТУ 4002

Ключевые слова: автоматизация, средства автоматизации, прикладные автоматизированные системы, промышленные изделия, данные, представление данных, обмен данными, искусственные языки, язык EXPRESS, описание

Редактор *В.П. Огурцов*
Технический редактор *О.Н. Власова*
Корректоры *В.С. Черная, В.И. Кануркина*
Компьютерная верстка *А.С. Юфина*

Изд. лиц. № 02354 от 14.07.2000. Сдано в набор 23.11.2000. Подписано в печать 05.02.2001. Усл.печ.л. 17,21. Уч.-изд.л. 17,00.
Тираж 300 экз. С 331. Зак. 253.

ИПК Издательство стандартов, 107076, Москва, Колодезный пер., 14.
Набрано в Издательстве на ПЭВМ.
Калужская типография стандартов, 248021, Калуга, ул. Московская, 256
ПЛР № 040138