МЕЖДУНАРОДНЫЙ СОЮЗ ЭЛЕКТРОСВЯЗИ

МККТТ

МЕЖДУНАРОДНЫЙ
КОНСУЛЬТАТИВНЫЙ КОМИТЕТ
ПО ТЕЛЕГРАФИИ И ТЕЛЕФОНИИ

СИНЯЯ КНИГА

ТОМ X — ВЫПУСК X.1

ЯЗЫК ФУНКЦИОНАЛЬНОЙ СПЕЦИФИКАЦИИ И ОПИСАНИЯ

КРИТЕРИИ ПРИМЕНЕНИЯ МЕТОДОВ ФОРМАЛЬНЫХ ОПИСАНИЙ

РЕКОМЕНДАЦИЯ Z.100 И ПРИЛОЖЕНИЯ A, B, C И E, РЕКОМЕНДАЦИЯ Z.110

IX ПЛЕНАРНАЯ АССАМБЛЕЯ
МЕЛЬБУРН, 14–25 НОЯБРЯ 1988 ГОДА

МЕЖДУНАРОДНЫЙ СОЮЗ ЭЛЕКТРОСВЯЗИ

МККТТ

МЕЖДУНАРОДНЫЙ
КОНСУЛЬТАТИВНЫЙ КОМИТЕТ
ПО ТЕЛЕГРАФИИ И ТЕЛЕФОНИИ

СИНЯЯ КНИГА

ТОМ X — ВЫПУСК X.1

ЯЗЫК ФУНКЦИОНАЛЬНОЙ СПЕЦИФИКАЦИИ И ОПИСАНИЯ

КРИТЕРИИ ПРИМЕНЕНИЯ МЕТОДОВ ФОРМАЛЬНЫХ ОПИСАНИЙ

РЕКОМЕНДАЦИЯ Z.100 И ПРИЛОЖЕНИЯ A, B, C И E, РЕКОМЕНДАЦИЯ Z.110

IX ПЛЕНАРНАЯ АССАМБЛЕЯ
МЕЛЬБУРН, 14–25 НОЯБРЯ 1988 ГОДА

ISBN 92-61-03754-2

СОДЕРЖАНИЕ КНИГИ МККТТ, ДЕЙСТВУЮЩЕЙ ПОСЛЕ IX ПЛЕНАРНОЙ АССАМБЛЕИ (1988 г.)

СИНЯЯ КНИГА

ВЫПУСК III.6 — Передача по линии нетелефонных сигналов. Передача сигналов звукового и телевизионного вещания. Рекомендации серий H и J (Исследовательская комиссия XV).

ВЫПУСК III.7 — Цифровая сеть с интеграцией служб (ЦСИС) — Общая структура и возможности служб. Рекомендации I.110—I.257 (Исследовательская комиссия XVIII).

ВЫПУСК III.8 — Цифровая сеть с интеграцией служб (ЦСИС) — Общесетевые аспекты и функции, стыки пользователь — сеть ЦСИС. Рекомендации I.310—I.470 (Исследовательская комиссия XVIII).

ВЫПУСК III.9 — Цифровая сеть с интеграцией служб (ЦСИС) — Межсетевые стыки и принципы технической эксплуатации. Рекомендации I.500—I.605 (Исследовательская комиссия XVIII).

Том IV

ВЫПУСК IV.1 — Общие принципы технической эксплуатации: техническая эксплуатация международных систем передачи и международных телефонных каналов. Рекомендации M.10—M.782 (Исследовательская комиссия IV).

ВЫПУСК IV.2 — Техническая эксплуатация международных телеграфных, фототелеграфных и арендованных каналов. Техническая эксплуатация международной телефонной сети общего пользования. Техническая эксплуатация морских спутниковых систем и систем передачи данных. Рекомендации M.800—M.1375 (Исследовательская комиссия IV).

ВЫПУСК IV.3 — Техническая эксплуатация международных каналов звукового и телевизионного вещания. Рекомендации серии N (Исследовательская комиссия IV).

ВЫПУСК IV.4 — Требования к измерительному оборудованию. Рекомендации серии O (Исследовательская комиссия IV).

Том V — Качество телефонной передачи. Рекомендации серии P (Исследовательская комиссия XII).

Том VI

ВЫПУСК VI.1 — Общие Рекомендации по телефонной коммутации и сигнализации. Функции и информационные потоки для служб в ЦСИС. Дополнения. Рекомендации Q.1—Q.118 bis (Исследовательская комиссия XI).

ВЫПУСК VI.2 — Требования к системам сигнализации № 4 и № 5. Рекомендации Q.120—Q.180 (Исследовательская комиссия XI).

ВЫПУСК VI.3 — Требования к системе сигнализации № 6. Рекомендации Q.251—Q.300 (Исследовательская комиссия XI).

ВЫПУСК VI.4 — Требования к системам сигнализации R1 и R2. Рекомендации Q.310—Q.490 (Исследовательская комиссия XI).

ВЫПУСК VI.5 — Цифровые местные, транзитные, комбинированные и международные станции в интегральных цифровых сетях и смешанных аналого-цифровых сетях. Дополнения. Рекомендации Q.500—Q.554 (Исследовательская комиссия XI).

ВЫПУСК VI.6 — Взаимодействие системы сигнализации. Рекомендации Q.601—Q.699 (Исследовательская комиссия XI).

ВЫПУСК VI.7 — Требования к системе сигнализации № 7. Рекомендации Q.700—Q.716 (Исследовательская комиссия XI).

ВЫПУСК VI.8 — Требования к системе сигнализации № 7. Рекомендации Q.721—Q.766 (Исследовательская комиссия XI).

ВЫПУСК VI.9 — Требования к системе сигнализации № 7. Рекомендации Q.771—Q.795 (Исследовательская комиссия XI).

ВЫПУСК VI.10 — Цифровая абонентская система сигнализации № 1 (ЦАС 1), уровень звена данных. Рекомендации Q.920 и Q.921 (Исследовательская комиссия XI).

ВЫПУСК VI.11 — Цифровая абонентская система сигнализации № 1 (ЦАС 1), сетевой уровень, управление пользователь — сеть. Рекомендации Q.930—Q.940 (Исследовательская комиссия XI).

ВЫПУСК VI.12 — Сухопутная подвижная сеть общего пользования. Взаимодействие с ЦСИС и коммутируемой телефонной сетью общего пользования. Рекомендации Q.1000—Q.1032 (Исследовательская комиссия XI).

ВЫПУСК VI.13 — Сухопутная подвижная сеть общего пользования. Подсистема подвижного применения и стыки. Рекомендации Q.1051—Q.1063 (Исследовательская комиссия XI).

ВЫПУСК VI.14 — Взаимодействие со спутниковыми подвижными системами. Рекомендации Q.1100—Q.1152 (Исследовательская комиссия XI).

Том VII

ВЫПУСК VII.1 — Телеграфная передача. Рекомендации серии R. Оконечное оборудование телеграфных служб. Рекомендации серии S (Исследовательская комиссия IX).

ВЫПУСК VII.2 — Телеграфная коммутация. Рекомендации серии U (Исследовательская комиссия IX).

ВЫПУСК VII.3 — Оконечное оборудование и протоколы для телематических служб. Рекомендации T.0—T.63 (Исследовательская комиссия VIII).

ВЫПУСК VII.4 — Процедуры испытания на соответствие Рекомендациям по службе телетекс. Рекомендация T.64 (Исследовательская комиссия VIII).

ВЫПУСК VII.5 — Оконечное оборудование и протоколы для телематических служб. Рекомендации T.65—T.101, T.150—T.390 (Исследовательская комиссия VIII).

ВЫПУСК VII.6 — Оконечное оборудование и протоколы для телематических служб. Рекомендации T.400—T.418 (Исследовательская комиссия VIII).

ВЫПУСК VII.7 — Оконечное оборудование и протоколы для телематических служб. Рекомендации T.431—T.564 (Исследовательская комиссия VIII).

Том VIII

ВЫПУСК VIII.1 — Передача данных по телефонной сети. Рекомендации серии V (Исследовательская комиссия XVII).

ВЫПУСК VIII.2 — Сети передачи данных: службы и услуги, стыки. Рекомендации X.1—X.32 (Исследовательская комиссия VII).

ВЫПУСК VIII.3 — Сети передачи данных: передача, сигнализация и коммутация, сетевые аспекты, техническая эксплуатация и административные положения. Рекомендации X.40—X.181 (Исследовательская комиссия VII).

ВЫПУСК VIII.4 — Сети передачи данных: взаимосвязь открытых систем (ВОС) — Модель и система обозначений, определение службы. Рекомендации X.200—X.219 (Исследовательская комиссия VII).

ВЫПУСК VIII.5 — Сети передачи данных: взаимосвязь открытых систем (ВОС) — Требования к протоколам, аттестационные испытания. Рекомендации X.220—X.290 (Исследовательская комиссия VII).

ВЫПУСК VIII.6 — Сети передачи данных: взаимодействие между сетями, подвижные системы передачи данных, межсетевое управление. Рекомендации X.300—X.370 (Исследовательская комиссия VII).

ВЫПУСК VIII.7 — Сети передачи данных: системы обработки сообщений. Рекомендации X.400—X.420 (Исследовательская комиссия VII).

ВЫПУСК VIII.8 — Сети передачи данных: справочная служба. Рекомендации X.500—X.521 (Исследовательская комиссия VII).

Том IX — Защита от мешающих влияний. Рекомендации серии K (Исследовательская комиссия V). Конструкция, прокладка и защита кабелей и других элементов линейных сооружений. Рекомендации серии L (Исследовательская комиссия VI).

Том X

ВЫПУСК X.1 — Язык функциональной спецификации и описания (SDL). Критерии применения методов формальных описаний (FDT). Рекомендация Z.100 и приложения A,B,C и E, Рекомендация Z.110 (Исследовательская комиссия X).

ВЫПУСК X.2 — Приложение D к Рекомендации Z.100: руководство для пользователей языка SDL (Исследовательская комиссия X).

ВЫПУСК X.3 — Приложение F.1 к Рекомендации Z.100: формальное определение языка SDL. Введение (Исследовательская комиссия X).

ВЫПУСК X.4 — Приложение F.2 к Рекомендации Z.100: формальное определение языка SDL. Статическая семантика (Исследовательская комиссия X).

ВЫПУСК X.5 — Приложение F.3 к Рекомендации Z.100: формальное определение языка SDL. Динамическая семантика (Исследовательская комиссия X).

ВЫПУСК X.6 — Язык МККТТ высокого уровня (CHILL). Рекомендация Z.200 (Исследовательская комиссия X).

ВЫПУСК X.7 — Язык человек—машина (MML). Рекомендации Z.301—Z.341 (Исследовательская комиссия X).

СОДЕРЖАНИЕ ВЫПУСКА X.1 СИНЕЙ КНИГИ

Рекомендация Z.100 и приложения A, B, C и E

Рекомендация Z.110

Язык функциональной спецификации и описания (SDL)

Критерии применения методов формальных описаний (FDTs)

ПРЕДВАРИТЕЛЬНЫЕ ЗАМЕЧАНИЯ

1 Вопросы, порученные каждой Исследовательской комиссии на период 1988—1992 годов, содержатся в Документе № 1 для данной Исследовательской комиссии.

2 В настоящем выпуске для краткости термин "Администрация" используется для обозначения как Администрации связи, так и признанной частной эксплуатационной организации.

ВЫПУСК X.1

Рекомендация Z.100 и приложения A, B, C и E
Рекомендация Z.110

ЯЗЫК ФУНКЦИОНАЛЬНОЙ СПЕЦИФИКАЦИИ И ОПИСАНИЯ (SDL)

КРИТЕРИИ ПРИМЕНЕНИЯ МЕТОДОВ ФОРМАЛЬНЫХ ОПИСАНИЙ (FDTs)

PAGE INTENTIONALLY LEFT BLANK

PAGE LAISSEE EN BLANC INTENTIONNELLEMENT

СОДЕРЖАНИЕ Z.100

РЕКОМЕНДАЦИЯ Z.100

ЯЗЫК СПЕЦИФИКАЦИИ И ОПИСАНИЯ (SDL)

ПРЕДВАРИТЕЛЬНОЕ ЗАМЕЧАНИЕ

Настоящая Рекомендация заменяет Рекомендации Z.100—Z.104 и Рекомендацию X.250 КРАСНОЙ КНИГИ МККТТ.

1 Введение в SDL

1.1 *Введение*

При рекомендации SDL (Языка спецификации и описания — Specification and Description Language) преследуется цель предложить язык, обеспечивающий возможность однозначных спецификаций и описания поведения телекоммуникационных систем. Считается, что спецификации и описания, использующие SDL, являются формальными в том смысле, что допускают однозначный анализ и интерпретацию.

Термины "спецификация" и "описание" используются в нижеследующих значениях:

a) под спецификацией системы понимается описание требуемого от нее поведения;

b) под описанием системы понимается описание ее фактического поведения.

Примечание. — Поскольку нет никакой разницы между применением SDL для спецификации и его применением для описания, впредь термин "спецификация" подразумевает оба значения.

Спецификация системы в широком смысле состоит из спецификации как ее функционального поведения, так и набора общих параметров системы. Однако SDL направлен на описание только аспектов поведения системы; описание общих параметров, специфицирующих такие свойства, как емкость и вес, требует применения других методов.

1.1.1 *Цели*

При разработке SDL преследовались общие цели создания такого языка, который:

a) легко изучить, использовать и интерпретировать;

b) обеспечивает однозначную спецификацию для заявок и заказов;

c) может быть расширен для охвата новых разработок;

d) может быть с равным успехом использован в различных методах спецификации и проектирования систем.

1.1.2 *Приложения*

Основной областью применения SDL является описание различных аспектов поведения систем реального времени. Возможными применениями являются:

a) обслуживание вызова (например, обработка вызова, телефонная сигнализация, измерения) в коммутационных системах;

b) поддержание работоспособности и исправление сбоев (например, аварии, автоматическое устранение ошибок, плановое тестирование) в общих телекоммуникационных системах;

c) управление системой (например, управление перегрузками, процедуры модификации и расширения возможностей);

d) вопросы сопровождения и функционирования; управление сетями;

e) протоколы передачи данных.

Конечно, SDL может быть использован для описания поведения любого объекта, допускающего описание с помощью дискретной модели; иными словами, объект поддерживает связь со своей окружающей средой с помощью дискретных сообщений.

SDL является богатым языком, который может быть использован для неформальной (и/или формально неполной) спецификации системы на высоком уровне, для полуформальной спецификации и, наконец, для детальной спецификации. Пользователь должен избрать подходящие разделы языка, соответствующие требующемуся ему уровню общений с другими и среде, в которой язык будет использоваться. В зависимости от той среды, которая будет пользоваться осуществленной спецификацией, многие аспекты описания некоторой системы могут быть оставлены на уровне обычного взаимопонимания между разработчиком спецификации и тем, кто будет ею пользоваться.

Таким образом, SDL может использоваться для разработки:

a) требований к некоторому устройству,

b) спецификации системы,

c) Рекомендаций МККТТ,

d) спецификации проекта системы,

e) детальной спецификации,

f) проекта системы (равно на высоком уровне и детального),

g) тестирования систем.

Предприятие пользователя может само выбрать подходящий ему уровень приложения SDL.

1.1.3 *Спецификация системы*

SDL описывает поведение системы в форме "стимул/реакция", предполагая, что и стимулы, и реакции являются дискретными и несут информацию. В частности, спецификация системы рассматривается как последовательность реакций на любую заданную последовательность стимулов.

Модель спецификации системы основывается на концепции обобщенного конечного автомата.

Кроме того, SDL обеспечивает средства структурирования, облегчающие спецификацию больших и/или сложных систем. Эти конструкции позволяют расчленить спецификацию системы на обозримые части, с которыми легко обращаться и которые легко понимать независимо от других. Расчленение может быть осуществлено в несколько шагов, в результате чего возникает иерархическая структура отдельных компонентов, определяющих систему на разных уровнях.

1.2 *Грамматики SDL*

SDL предоставляет выбор между двумя различными синтаксическими формами для представления системы: Графическое Представление (SDL/GR) и представление в виде текста, состоящего из фраз (SDL/PR). Поскольку обе формы являются конкретными представлениями одной и той же SDL-семантики, они эквивалентны друг другу. В частности, они обе эквивалентны по отношению к абстрактной грамматике соответствующих представлений.

Некоторое подмножество SDL/PR входит также и в SDL/GR. Это подмножество называется общей текстовой грамматикой.

На рис. 1.1 изображены зависимости между SDL/GR, SDL/PR, конкретными грамматиками и абстрактной грамматикой.

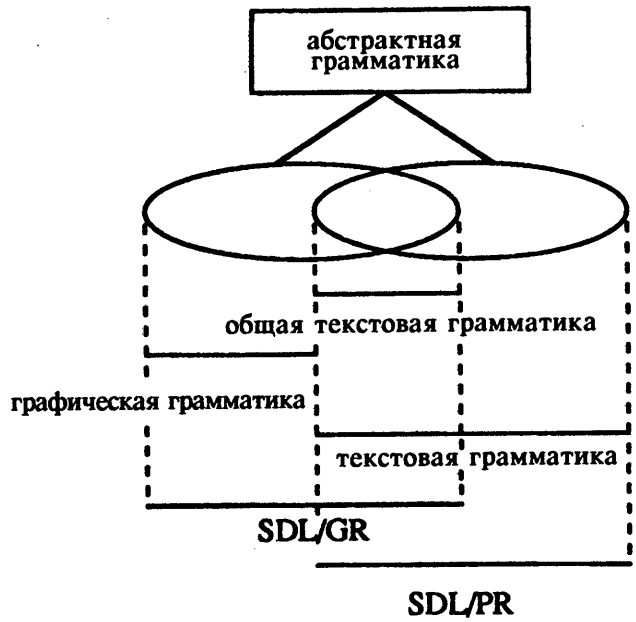


РИСУНОК 1.1

Грамматики SDL

Для каждой из конкретных грамматик существует определение ее собственного синтаксиса и ее отношения к абстрактной грамматике (то есть как преобразовать ее к абстрактному синтаксису). При таком подходе существует только одно определение семантики SDL; каждая конкретная грамматика "наследует" семантику в силу своего отношения к абстрактной грамматике. Этот подход обеспечивает в то же время эквивалентность SDL/GR и SDL/PR.

Приводится также формальное определение SDL, определяющее правила преобразования спецификации системы в абстрактный синтаксис и правила интерпретации спецификации, заданной в терминах абстрактного синтаксиса.

1.3 *Основные определения*

Несколько основных понятий и соглашений используются на всем протяжении настоящей Рекомендации. Ниже приводятся их определения.

1.3.1 *Тип, определения и экземпляр*

В настоящей Рекомендации фундаментальными являются понятия типа, его экземпляра и их взаимоотношений. Используются схема и терминология, определенные ниже и изображенные на рис. 1.2.

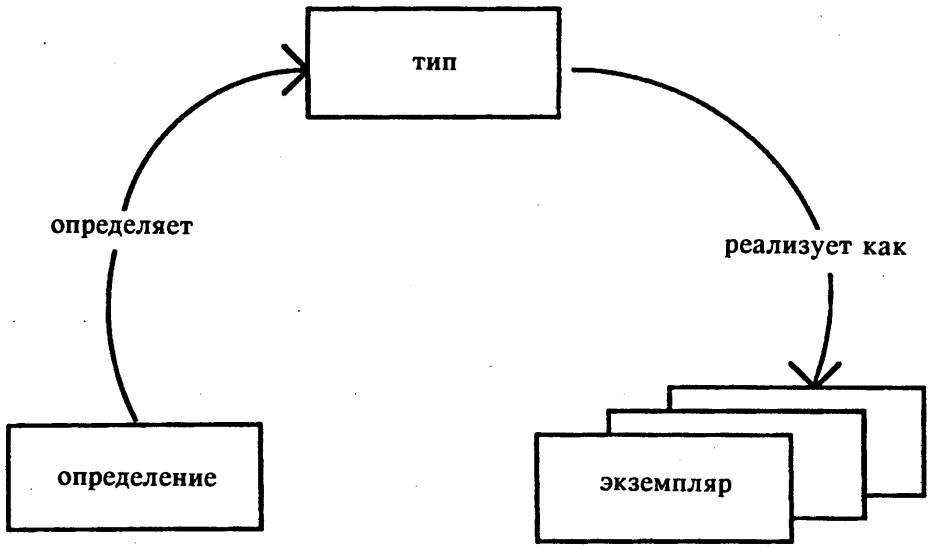


РИСУНОК 1.2

Концепция типа

Типы описываются с помощью определений. Определение типа определяет все свойства, ассоциируемые с этим типом. Тип может быть конкретизирован в любом числе экземпляров. Каждый экземпляр некоторого типа обладает всеми свойствами, присущими этому типу.

Эта схема применима к различным понятиям SDL: например, определения системы и экземпляры системы, определения процесса и экземпляры процесса.

Тип данных является специальным классом типа (см. § 2.3 и § 5).

Примечание. – Чтобы не загромождать текста, применяется соглашение, гласящее, что слово "экземпляр" может быть опущено. Например, выражение "система интерпретируется..." будет на самом деле означать "экземпляр системы интерпретируется..."

1.3.2 *Окружающая среда*

Системы, специфицированные на SDL, ведут себя в соответствии со стимулами, поступающими из внешнего мира, называемого окружающей средой специфицируемой системы.

Предполагается, что окружающая среда содержит один или более экземпляров процессов, и поэтому сигналы, поступающие в систему из окружающей среды, посланы конкретными экземплярами процессов. Значения PId этих процессов отличны от всех значений PId в системе (см. § 5.6.10).

Хотя поведение окружающей среды является недетерминированным, она должна подчиняться ограничениям, налагаемым спецификацией системы.

1.3.3 *Ошибки*

Спецификация системы считается допустимой SDL-спецификацией, только если она удовлетворяет синтаксическим правилам и статистическим условиям SDL.

Если при интерпретации допустимой SDL-спецификации обнаруживается нарушение динамических условий, то наступает ошибка. Обнаружение ошибки при интерпретации системы приводит к тому, что дальнейшее поведение системы не может быть выведено из ее спецификации.

1.4 *Форма представления*

1.4.1 *Членение текста*

В §§ 2, 3, 4, 5 настоящей Рекомендации материал членится на разделы, имеющие нижеследующую структуру. В начале раздела может находиться необязательное введение в раздел, следом за которым идут пункты, выделенные заголовками; в этих пунктах определяются:

a) *Абстрактная грамматика* — описываемая средствами абстрактного синтаксиса и статическими условиями корректности.

b) *Конкретная текстовая грамматика* — как общая текстовая грамматика, используемая и в SDL/PR и в SDL/GR, так и используемая только в SDL/PR. Эта грамматика описывается средствами текстового синтаксиса, статическими правилами и условиями корректности текстового синтаксиса. Кроме того, здесь приводится описание отношения текстового синтаксиса к абстрактному синтаксису.

c) *Конкретная графическая грамматика* — описываемая средствами графического синтаксиса, статическими правилами и условиями корректности графического синтаксиса. Кроме того, здесь приводится описание отношения графического синтаксиса к абстрактному синтаксису, приводятся также некоторые дополнительные (по отношению к описанным в § 2.2.4) правила вычерчивания.

d) *Семантика* — дает значения типов, описывает их свойства, описывает метод интерпретации отдельных экземпляров этих типов, содержит также все динамические условия, которые должны быть выполнены для экземпляров этих типов, чтобы они вели себя корректно в смысле SDL.

e) *Модель* — дает правила отображения сокращенных обозначений на конструкции, которые были предварительно определены непосредственно средствами конкретного синтаксиса.

f) *Примеры.*

1.4.2 *Пункты, выделенные заголовками*

Если раздел начинается введением, за которым следуют пункты, выделенные заголовками, то введение рассматривается как неформальная часть Рекомендации, включенная в Рекомендацию только с целью сделать ее более понятной, но не более полной.

Если некоторый пункт не содержит никакого текста, то он полностью опускается.

В остальной части этого раздела описываются другие специальные формализмы, используемые во всех озаглавленных пунктах, и используемые заголовки. Это описание может также рассматриваться как пример типографического расположения описанных выше озаглавленных пунктов первого уровня, для которых текущий текст образует введение.

Абстрактная грамматика

Нотация абстрактного синтаксиса описана в § 1.5.1.

Если пункт *Абстрактная грамматика* опущен, то это означает, что для понятия, которое изучается в данном разделе, нет дополнительного абстрактного синтаксиса и что конкретный синтаксис должен быть отображен на абстрактный синтаксис, определенный в другом разделе.

На правила, описанные в абстрактном синтаксисе, можно ссылаться из любого пункта посредством имени конкретного правила, написанного курсивом.

Правила в формальной нотации могут сопровождаться параграфами, определяющими условия, которым должны удовлетворять корректные SDL-определения и которые могут быть проверены, не прибегая к интерпретации экземпляра. На этом уровне статические условия относятся только к абстрактному синтаксису. Статические правила, относящиеся только к конкретному синтаксису, определяются после конкретного синтаксиса. Абстрактный синтаксис совместно со статическими условиями абстрактного синтаксиса образуют абстрактную грамматику языка.

Конкретная текстовая грамматика

Конкретный текстовой синтаксис определяется с помощью расширенной формы Бэкуса-Наура для описания синтаксиса, описанной в Рекомендации Z.200, § 2.1 (см. также § 1.5.2).

Текстовой синтаксис сопровождается параграфами, определяющими статические условия, которым должен удовлетворять корректный текст и которые могут быть проверены, не прибегая к интерпретации экземпляра. К текстовому синтаксису относятся также статические условия абстрактного синтаксиса, если таковые имеются.

Во многих случаях существует простое соответствие между конкретным и абстрактным синтаксисами, поскольку правило в конкретном синтаксисе изображается просто одним правилом в абстрактном синтаксисе. Если одно и то же имя используется в абстрактном и конкретном синтаксисе (что делается с целью показать, что они обозначают одно и то же понятие), тогда текст "<x> в конкретном синтаксисе обозначает X в абстрактном синтаксисе" часто опускается, но всегда подразумевается в языковых описаниях. В этом контексте существенны подчеркнутые семантические подкатегории.

Конкретный текстовой синтаксис, который не является сокращенной формой (моделируемой другими SDL-конструкциями), является непосредственно конкретным текстовым синтаксисом. Отношение конкретного текстового синтаксиса к абстрактному синтаксису приводится только для непосредственного конкретного текстового синтаксиса.

Соотношения между конкретным текстовым синтаксисом и абстрактным синтаксисом опускаются в случае, когда определяемое в разделе понятие является сокращенной формой и моделируется другими конструкциями (см. ниже *Модель*).

Конкретная графическая грамматика

Конкретный графический синтаксис определяется с помощью расширенной формы Бэкуса-Наура для описания синтаксиса; эта форма описана в § 1.5.3.

Графический синтаксис сопровождается параграфами, определяющими статические условия, которым должен удовлетворять корректный SDL/GR и которые могут быть проверены, не прибегая к интерпретации экземпляра. К графическому синтаксису относятся также статические условия абстрактного синтаксиса, если таковые имеются.

Соотношения между конкретным графическим синтаксисом и абстрактным синтаксисом опускаются в случае, когда определяемое в разделе понятие является сокращенной формой и моделируется другими конструкциями (см. ниже *Модель*).

Во многих случаях существует простое соответствие между диаграммами конкретной графической грамматики и определениями в абстрактном синтаксисе. Если в конкретной грамматике нетерминальное имя содержит слово "диаграмма" и в абстрактном синтаксисе имеется имя, которое отличается только тем, что заканчивается словом *определение*, то оба правила определяют одно и то же понятие. Например, < диаграмма системы> в конкретной грамматике и *Определение-системы* в абстрактной грамматике соответствуют друг другу.

Расширения в конкретном синтаксисе, которые возникают за счет таких средств, как удаленные определения (см. § 2.4.1), макросы (см. § 4.2), отображения литералов (см. § 5.4.1.15) и т.д., должны быть рассмотрены перед установлением соответствия между конкретным и абстрактным синтаксисами.

Семантика

В правилах корректности используются признаки: эти правила содержат или тип признака, или другие типы, ссылающиеся на этот тип.

Примером признака может быть совокупность допустимых идентификаторов входных сигналов процесса. Этот признак используется в статическом правиле, гласящем, что "Для каждой *Вершины-состояния* все входные *Идентификаторы-сигналов* (из множества допустимых входных сигналов) входят либо в *Совокупность-сохраняемых-сигналов*, либо в *Вершину-ввода*".

Все экземпляры обладают идентифицирующим признаком; если этот признак не формируется каким-то необычным способом, то он определяется согласно правилам, описанным в общих соглашениях об идентификации в ± 2. Поэтому этот признак обычно не упоминается как идентифицирующий признак. В равной мере не имеет смысла упоминать подкомпоненты определений, содержащиеся в определении, потому что владелец этих подкомпонентов очевиден из абстрактного синтаксиса. Например, совершенно очевидно, что определение блока "содержит" либо определение процесса, либо определение подблока.

Признаки являются статическими, если они могут быть определены без интерпретации спецификации SDL-системы, и динамическими, если для их определения требуется интерпретация спецификации SDL-системы.

Интерпретация описывается в функциональной форме. Всюду, где в Абстрактном Синтаксисе содержится какой-нибудь список, этот список интерпретируется в порядке входящих в него элементов. То есть в Рекомендации описывается, как из определения системы создаются ее экземпляры и как эти экземпляры интерпретируются в "абстрактной SDL-машине".

Динамическими являются условия, которые должны быть удовлетворены только в процессе интерпретации и не могут быть проверены без нее. Динамические условия могут приводить к ошибкам (см. § 1.3.3).

Модель

Некоторые конструкции рассматриваются как "производный конкретный синтаксис" (или сокращения) по отношению к другим эквивалентным конструкциям в том же конкретном синтаксисе. Например, опускание ввода некоторого сигнала является производным конкретным синтаксисом, заменяющим вершину ввода этого сигнала, за которой следует нулевой переход обратно в то же состояние.

В некоторых случаях последовательная замена (расширение) такого "производного конкретного синтаксиса" приводит к исключительно большому (возможно, даже бесконечному) представлению. Тем не менее, семантика такой специализации может быть определена.

Примеры

Пункт, озаглавленный *Примеры*, содержит примеры.

1.5 *Метаязыки*

В зависимости от конкретных потребностей для определения свойств и синтаксисов SDL используются различные метаязыки.

Ниже приводятся введения в описания использованных метаязыков; в подходящих случаях приводятся только ссылки на книги или специальные публикации МСЭ.

1.5.1 *Язык Мета IV*

Для описания абстрактного синтаксиса SDL используется нижеследующее подмножество языка Мета IV.

Определение в абстрактном синтаксисе может рассматриваться как поименованный составной объект (дерево), определяющий множество подкомпонентов.

Например, определение переменной в абстрактном синтаксисе имеет вид:

Определение-переменной :: *Имя-переменной Идентификатор-ссылки-на-сорт*

Это определение описывает домен составного объекта (дерева), называемого *Определение-переменной*. Этот объект состоит из двух подкомпонентов, которые сами могут быть деревьями.

Определение на языке Мета IV:

Идентификатор-ссылки-на-сорт = *Идентификатор*

означает, что *Идентификатор-ссылки-на-сорт* является *Идентификатором* и поэтому не может быть синтаксически отличим от прочих идентификаторов.

Объект может также принадлежать к некоторым элементарным (несоставным) доменам. В контексте SDL таковыми являются нижеследующие объекты:

a) Целые объекты

Например,

Число экземпляров :: *Intg Intg*

Число экземпляров обозначает составной домен, содержащий два целых (*Intg*) значения, обозначающих начальное и максимальное число экземпляров.

b) Выделенные объекты

Эти объекты изображаются латинскими прописными буквами или цифрами, напечатанными жирным шрифтом.

Например,

Процесс-получатель = *Идентификатор-процесса* **ENVIRONMENT**

Процессом-получателем является либо *Идентификатор-процесса*, либо среда, обозначенная выделенным словом **ENVIRONMENT**.

c) Объекты — знаки

Знаки обозначают домен элементарных знаков. Этот домен может рассматриваться как содержащий потенциально бесконечное число различных атомарных объектов, для которых не требуется никакого представления.

Например,

Имя :: *Знак*

Имя состоит из атомарного объекта, причем любое *Имя* отличимо от любого другого имени.

d) Неспецифицированные объекты

Неспецифицированный объект обозначает домены, которые могут иметь некоторое представление, которое находится вне области действия настоящей Рекомендации.

Например,

Неформальный текст :: . . .

Неформальный текст содержит объект, который не интерпретируется.

Нижеследующие операции (конструкторы) языка БНФ (см. § 1.5.2) также используются в абстрактном синтаксисе: "*" для возможно пустого списка, " + " для непустого списка, "|" для альтернативы и "[" "]" для необязательного элемента.

Скобки используются для группировки логически связанных доменов.

Наконец, в абстрактном синтаксисе используется еще одна постфиксная операция — "set" (совокупность), строящая совокупность, то есть неупорядоченное множество различных объектов. Например,

Граф-процесса :: *Стартовая-вершина-графа Вершина-состояния*-set

Граф-процесса состоит из *Стартовой-вершины-процесса* и совокупности *Вершин-состояний.*

1.5.2 *БНФ*

В форме Бэкуса-Наура терминальные символы изображаются либо тем, что они не заключены в угловые скобки (то есть знаки "меньше чем" и "больше чем", < и >), либо тем, что они являются двумя специальными терминальными объектами: <имя> и <знаковая строка>. Отметим, что два специальных терминальных объекта <имя> и <знаковая строка> могут, кроме того, нести семантическую нагрузку, выделенную так, как это описывается ниже.

Угловые скобки и заключенное в них слово (слова) являются либо нетерминальными объектами, либо одним из двух терминальных объектов: <имя> и <знаковая строка>. Синтаксическими категориями являются нетерминальные объекты, изображенные одним или несколькими словами, заключенными в угловые скобки. Для каждого нетерминального объекта задается правило порождения, указанное либо в конкретной текстовой грамматике, либо в конкретной графической грамматике. Например,

<обозревающее выражение> :: =
VIEW (<идентификатор переменной>, <выражение>)

Правило порождения для нетерминального объекта состоит из нетерминального объекта, находящегося слева от знака :: = ,

и одной или более конструкций справа от этого знака; этими конструкциями являются нетерминальные, объекты и/или терминальные объекты. Например, в вышеприведенном примере <обозревающее выражение>, <идентификатор переменной> и <выражение> являются нетерминальными объектами, а VIEW, скобки и запятая — терминальными символами.

Иногда символы содержат подчеркнутую часть. Эта подчеркнутая часть выделяет семантический аспект символа. Например, <идентификатор переменной> синтаксически идентичен <идентификатору>, но с семантической точки зрения требуется, чтобы этот идентификатор был идентификатором переменной.

Справа от знака ::= в качестве нетерминального объекта могут быть указаны несколько альтернатив, разделенных между собой вертикальной чертой (|). Например, запись

```
<область блока> ::=
                        <графическая ссылка на блок>
                  |     <диаграмма блока>
```

означает, что <область блока> является либо <графической ссылкой на блок>, либо <диаграммой блока>.

Синтаксические элементы могут быть сгруппированы друг с другом с помощью фигурных скобок ({ и }), аналогично использованию круглых скобок в языке Мета IV (см. § 1.5.1). Группа, заключенная в фигурные скобки, может содержать одну или несколько вертикальных черточек, обозначающих альтернативные синтаксические элементы. Например,

```
<область взаимодействия блоков> ::=
                        { <область блока> | <область определения канала> } +
```

Повторение группы, заключенной в фигурные скобки, изображается звездочкой (*) или знаком плюс (+). Звездочка означает, что группа не является обязательной, но может быть повторена любое число раз, знак плюс означает, что наличие группы является обязательным и что она может быть повторена любое число раз. Вышеприведенный пример означает, что <область взаимодействия блока> содержит хоть один из двух объектов: <область блока> или <область определения канала> и может содержать оба эти объекта любое число раз.

Если синтаксические элементы группированы с помощью квадратных скобок ([и]), то это означает, что наличие этой группы не обязательно. Например, запись

```
<заголовок процесса> ::=
                        PROCESS <идентификатор процесса> [ <формальные параметры>]
```

означает, что <заголовок процесса> может содержать, а может и не содержать <формальные параметры>.

1.5.3 *Метаязык для графической грамматики*

Для графической грамматики метаязык, описанный в § 1.5.2, расширен с помощью нижеследующих метасимволов:

a) **contains** (содержит)
b) **is associated with** (ассоциирован с)
c) **is followed by** (за ним следует)
d) **is connected to** (присоединен к)
e) **set** (совокупность)

Метасимвол **set** является постфиксной операцией, воздействующей на непосредственно предшествующие ей синтаксические элементы, заключенные в фигурные скобки, она обозначает (неупорядоченную) сово-

купность объектов. Каждый объект может быть произвольной группой синтаксических элементов; в этом случае к объекту сначала применяются правила порождения, а затем метасимвол **set**.
Например,

{ { <область системного текста> } * { <диаграмма макроса> } * <область взаимодействия блока> } **set**

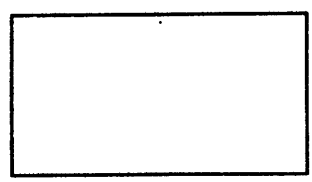
является совокупностью, элементы которой содержат ноль или более <областей системного текста>, ноль или более <диаграмм макрос> и одну <область взаимодействия блоков>.

Все остальные метасимволы являются инфиксными операциями, для которых нетерминальный графический символ является левым аргументом. Правым аргументом является либо группа синтаксических элементов, помещенная внутри фигурных скобок, либо единственный синтаксический элемент. Если в правой части правила порождения на первом месте стоит нетерминальный графический символ и имеются еще несколько этих инфиксных операций, то графический нетерминальный символ является левым аргументом каждой из этих инфиксных операций. Графическим нетерминальным символом является нетерминальный объект, в котором слово "символ" стоит непосредственно перед символом "больше чем">.

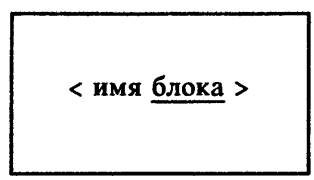
Метасимвол **contains** означает, что его правый аргумент должен быть помещен внутрь его левого аргумента и присоединенного к нему <символа расширения текста>, если таковой имеется. Например, запись

<графическая ссылка на блок> :: =
<символ блока> **contains** <имя блока>

<символ блока> :: =



означает, что должно быть изображено



Метасимвол **is associated with** означает, что его правый аргумент логически ассоциируется с левым аргументом (как если б он "содержался" в этом аргументе; однозначность "ассоциирования" обеспечивается правилами вычерчивания).

Метасимвол **is followed by** означает, что его правый аргумент следует (как логически, так и на чертеже) за его левым аргументом.

Метасимвол **is connected to** означает, что его правый аргумент присоединен (как логически, так и на чертеже) к его левому аргументу.

2 Основной SDL

2.1 *Введение*

SDL-система состоит из множества блоков. Блоки связаны друг с другом и с окружающей средой каналами. Внутри каждого блока имеется один или более процессов. Эти блоки общаются друг с другом с помощью сигналов; предполагается, что они работают параллельно.

Настоящий § 2 разбит на 8 основных разделов:

a) *Общие правила*

Основные SDL-понятия, такие как лексические правила и идентификаторы, правила видимости, неформальный текст, членение диаграмм, правила вычерчивания, комментарии, расширение текста, текстовые символы.

b) *Основные понятия данных*

Основные SDL-понятия данных, такие как значения, переменные, выражения.

c) *Структура системы*

Содержит SDL-понятия, касающиеся общих языковых принципов структурирования; таковыми являются: система, блок, процесс, процедура.

d) *Связь*

Содержит используемые в SDL механизмы связи, такие как канал, маршрут сигнала, сигнал.

e) *Поведение*

Конструкции, относящиеся к поведению процесса: общие правила соединений в графах процессов или процедур, определение переменной, старт, состояние, ввод, сохранение, метка, переход.

f) *Действие*

Активные конструкции, такие как работа, создание процесса, вызов процедуры, вывод, принятие решения.

g) *Таймеры*

Определение Таймера и примитивы Таймера.

h) *Примеры*

Примеры, на которые имеются ссылки из других разделов.

2.2 *Общие правила*

2.2.1 *Лексические правила*

Лексические правила определяют лексемы. Лексемы являются терминальными символами *Конкретного текстового синтаксиса.*

<лексемы> :: =
<имя>
| <знаковая строка>
| <специальный знак>
| <составной специальный знак>
| <замечание>
| <ключевое слово>

<имя> :: =
<слово> { <знак подчеркивания> <слово> } *

<слово> :: =
{ <алфавитно-цифровой знак> | <точка> } *
<алфавитно-цифровой знак>
{ <алфавитно-цифровой знак> | <точка> } *

<алфавитно-цифровой знак> :: =
<буква>
| <десятичная цифра>
| <национальный знак>

<буква> :: =
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
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

<десятичная цифра> :: =
0 | 1 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<национальный знак> :: =

#
| `
| ○
| @
| <левая квадратная скобка>
| \
| <правая квадратная скобка>
| <левая фигурная скобка>
| <вертикальная черта>
| <правая фигурная скобка>
| <знак надчеркивания>
| <стрелка вверх>

<левая квадратная скобка> :: =
[

<правая квадратная скобка> :: =
]

<левая фигурная скобка> :: =
{

<вертикальная линия> :: =
|

<правая фигурная скобка> :: =
}

<знак надчеркивания> :: =
~

<стрелка вверх> :: =

<точка> :: =

<знак подчеркивания> :: =
—

<знаковая строка> :: =

<апостроф> { <алфавитно-цифровой знак>
| <прочий знак>
| <специальный знак>
| <точка>
| <знак подчеркивания>
| <знак пробела>
| <апостроф> <апостроф> } * <апостроф>

<текст> :: =

{ <алфавитно-цифровой знак>
| <прочий знак>
| <специальный знак>
| <точка>
| <знак подчеркивания>
| <знак пробела>
| <апостроф> } *

<апостроф> :: =
'

<прочий знак> :: =
? | & | %

<специальный знак> :: =
+ | — | ! | / | > | * | (|) | " | , | ; | < | = | :

<составной специальный знак> :: =

```
        = =
    |   = = >
    |   / =
    |   < =
    |   > =
    |   //
    |   : =
    |   = >
    |   − >
    |   ( .
    |   . )
```

<замечание> :: =

```
        /* <текст>* /
```

<ключевое слово> :: =

```
        ACTIVE
    |   ADDING
    |   ALL
    |   ALTERNATIVE
    |   AND
    |   AXIOMS
    |   BLOCK
    |   CALL
    |   CHANNEL
    |   COMMENT
    |   CONNECT
    |   CONSTANT
    |   CONSTANTS
    |   CREATE
    |   DCL
    |   DECISION
    |   DEFAULT
    |   ELSE
    |   ENDALTERNATIVE
    |   ENDBLOCK
    |   ENDCHANNEL
    |   ENDDECISION
    |   ENDGENERATOR
    |   ENDMACRO
    |   ENDNEWTYPE
    |   ENDPROCEDURE
    |   ENDPROCESS
    |   ENDREFINEMENT
    |   ENDSELECT
    |   ENDSERVICE
    |   ENDSTATE
    |   ENDSUBSTRUCTURE
    |   ENDSYNTYPE
    |   ENDSYSTEM
    |   ENV
    |   ERROR
    |   EXPORT
    |   EXPORTED
```

| EXTERNAL
| FI
| FOR
| FPAR
| FROM
| GENERATOR
| IF
| IMPORT
| IMPORTED
| IN
| INHERITS
| INPUT
| JOIN
| LITERAL
| LITERALS
| MACRO
| MACRODEFINITION
| MACROID
| MAP
| MOD
| NAMECLASS
| NEWTYPE
| NEXTSTATE
| NOT
| NOW
| OFFSPRING
| OPERATOR
| OPERATORS
| OR
| ORDERING
| OUT
| OUTPUT
| PARENT
| PRIORITY
| PROCEDURE
| PROCESS
| PROVIDED
| REFERENCED
| REFINEMENT
| REM
| RESET
| RETURN
| REVEALED
| REVERSE
| SAVE
| SELECT
| SELF
| SENDER
| SERVICE
| SET
| SIGNAL
| SIGNALLIST
| SIGNALROUTE
| SIGNALSET
| SPELLING

| START
| STATE
| STOP
| STRUCT
| SUBSTRUCTURE
| SYNONYM
| SYNTYPE
| SYSTEM
| TASK
| THEN
| TIMER
| TO
| TYPE
| VIA
| VIEW
| VIEWED
| WITH
| XOR

<пробел> является знаком пробела в Алфавите № 5 МККТТ.

<национальные знаки>, приведенные выше, изображены так, как в International Reference Version Алфавита № 5 (Рекомендация Т.50). Ответственность за определение национальных изображений этих знаков лежит на национальных органах стандартизации.

Все <буквы> рассматриваются как прописные, за исключением случаев, когда они входят в <знаковую строку>. (Использование <национальных знаков> может быть определено национальными органами стандартизации).

<лексема> кончается, когда появляется первый знак, который не может входить в <лексему> в соответствии со специфицированным выше синтаксисом. Если за <знаком подчеркивания> следует один или более управляющих знаков (управляющие знаки определяются так, как в Рекомендации Т.50) или пробелы, то все эти знаки (включая <знак подчеркивания>) игнорируются. Например, A_B обозначает то же самое <имя>, что и AB. Такое использование <знака подчеркивания> позволяет разбить <лексему (ы)> по нескольким строкам.

Если в <имени> вслед за <знаком подчеркивания> идет <слово> и если одно из <слов>, между которыми находится <знак подчеркивания>, не является <ключевым словом>, то вместо <знака подчеркивания> можно ставить один или несколько управляющих знаков или пробелов. Например, A B обозначает то же самое <имя>, что и A_B.

Однако существует несколько случаев, при которых отсутствие в <имени> <знака подчеркивания> вносит неоднозначность. Поэтому требуется соблюдение следующих правил:

1. <знак подчеркивания> в <имени>, входящем в <квалификационный шаг>, должен быть выражен явно.

2. Если непосредственно за одним или несколькими <именами> или <идентификаторами> может следовать <сорт> (например, <определения переменных>, <определения обозревания>), то <знаки подчеркивания>, входящие в эти <имена> или <идентификаторы>, должны быть выражены явно.

3. Если <определение данных> содержит <реализацию генератора>, то <знаки подчеркивания>, входящие в <имя сорта>, следующее за ключевым словом NEWTYPE, должны быть выражены явно.

Управляющие знаки имеют тот же смысл, что и пробелы.

Управляющие знаки и пробелы могут появляться между двумя <лексемами> любое число раз. Любое число появлений управляющих знаков и пробелов между двумя <лексемами> эквивалентно одному пробелу между ними.

Знаком /, непосредственно за которым следует знак *, всегда начинается <замечание>. Появление в <замечании> знака *, непосредственно за которым следует знак /, означает окончание <замечания>. <замечание> может быть вставлено перед или после любой <лексемы>.

К <телу макроса> применимы специальные лексические правила (см. § 4.2.1).

2.2.2 *Правила видимости и идентификаторы*

Абстрактная грамматика

Идентификатор :: *Квалификатор Имя*
Квалификатор = *Квалификационный-шаг* +
Квалификационный-шаг = *Квалификатор-системы* |
Квалификатор-блока |
Квалификатор-подструктуры-блока |
Квалификатор-сигнала |
Квалификатор-процесса |
Квалификатор-процедуры |
Квалификатор-сорта

Квалификатор-системы :: *Имя-системы*

Квалификатор-блока :: *Имя-блока*
Квалификатор-подструктуры-блока :: *Имя-подструктуры-блока*
Квалификатор-процесса :: *Имя-процесса*
Квалификатор-процедуры :: *Имя-процедуры*
Квалификатор-сигнала :: *Имя-сигнала*
Квалификатор-сорта :: *Имя-сорта*
Имя :: *Элементарный знак*

Конкретная текстовая грамматика

<идентификатор> :: =
[<квалификатор>] <имя>

<квалификатор> :: =
<квалификационный шаг> { / <квалификационный шаг> } *

<квалификационный шаг> :: =
<класс сегментов> <имя>

<класс сегментов> :: =
SYSTEM
| BLOCK
| SUBSTRUCTURE
| SIGNAL
| PROCESS
| PROCEDURE

| TYPE
| SERVICE

<Классу сегментов>, обозначенному через SERVICE, в абстрактном синтаксисе ничто не соответствует. <имена> и <идентификаторы объектов>, определенные в <определении сервиса>, преобразуются соответственно в уникальные <имена> и <идентификаторы>, определенные в том <определении процесса>, которое содержит <определение сервиса>.

<квалификатор> отражает иерархичность структуры, начиная с уровня системы и вплоть до уровня определяемого контекста; при этом уровень системы является самой левой частью <квалификатора>.

Несколько самых левых <квалификационных шагов> (за исключением <удаленных определений>, см. § 2.4.1) или весь <квалификатор> могут быть опущены. Если весь <квалификатор> опущен и если <имя> обозначает объект из класса объектов, содержащего переменные, синонимы, литералы и операции (см. ниже *Семантику*), то связь <имени> с определением должна быть выявляемой из фактического контекста. В других случаях <идентификатор> связан с объектом, для которого определяющим контекстом является ближайший объемлющий сегмент, обладающий тем свойством, что в нем <квалификатор> <идентификатора> совпадает с самой правой частью полного <квалификатора>, обозначающего этот сегмент. Если <идентификатор> не содержит <квалификатора>, то требование на совпадение <квалификаторов> опускается.

Подсигнал должен квалифицироваться своим родительским сигналом, если только в этом месте нет ни одного видимого сигнала, имеющего то же <имя>.

Выявление связи <имени> с определением по контексту возможно в двух нижеследующих случаях:

a) Сегмент, в котором используется <имя>, не является <частичным определением типа>, и содержит определение, имеющее это <имя>. Тогда <имя> будет связано с этим определением.

b) Сегмент, в котором это <имя> используется, не содержит определения, имеющего это <имя>, либо этот сегмент является <частичным определением типа>, и во всем <определении системы> имеется равно одно такое видимое определение объекта, имеющего то же <имя>, с которым это <имя> может быть связано без нарушения статических условий (совместимость сортов и т.п.) той конструкции, в которой это <имя> появляется. Тогда <имя> будет связано с этим определением.

Можно использовать только видимые идентификаторы, кроме случая использования <идентификатора переменной> в <определении обозревания> и случая <идентификатора>, используемого вместо <имени> в определении, на которое делается ссылка (то есть определении, вынесенном за пределы <определения системы>).

Семантика

Сегменты определяются на основании нижеследующей схемы:

Конкретная текстовая грамматика	*Конкретная графическая грамматика*
<определение системы>	<диаграмма системы>
<определение блока>	<диаграмма блока>
<определение процесса>	<диаграмма процесса>
<определение процедуры>	<диаграмма процедуры>
<определение подструктуры блока>	<диаграмма подструктуры блока>

<определение подструктуры канала> <диаграмма подструктуры канала>
<определение сервиса> <диаграмма сервиса>
<частичное определение типа>
<уточнение сигнала>

Каждый сегмент имеет связанный с ней список определений. Каждое из определений описывает объект, принадлежащий некоторому классу объектов и имеющий связанное с ним имя. Для <частичного определения типа> связанный с ним список определений содержит <сигнатуру операций> и <сигнатуру литералов>, а также все <сигнатуры операций> и <сигнатуры литералов>, которые он наследует от своего родительского сорта, от генератора или неявно подразумеваемого за счет использования сокращенных обозначений, таких как ключевое слово ORDERING (см. § 5.4.1.8). Заметьте, что <определение обозревания> не определяет никакого объекта.

Хотя <инфиксные операции>, <операции > и <знаковая строка> имеют свою собственную синтаксическую нотацию, по существу они являются <именами>: в *Абстрактном синтаксисе* они представлены *именами*. В дальнейшем они рассматриваются (также синтаксически) как <имена>. Тем не менее, <имя состояния>, <имя коннектора>, <формальное имя генератора>, <идентификаторы значений> в равенствах, <формальные имена макросов> и <имена макросов> имеют свои специальные правила видимости и потому не могут быть квалифицированы. <имена состояний> и <имена коннекторов> являются невидимыми извне <тела процесса>, <тела процедуры> или <тела сервиса> соответственно. Прочие правила видимости изучаются в соответствующих разделах.

Говорят, что каждый объект имеет свой определяющий контекст внутри того сегмента, который его определяет. Обращение к объектам осуществляется с помощью <идентификаторов>.

<квалификатор>, входящий в <идентификатор>, специфицирует уникальный определяющий контекст соответствующего <имени>.

Существуют следующие классы объектов:

a) система
b) блоки
c) каналы, маршруты сигналов
d) сигналы, таймеры
e) процессы
f) процедуры
g) переменные (включая формальные параметры), синонимы, литералы, операции
h) сорта
i) генераторы
j) импортированные объекты
k) списки сигналов
l) сервисы
m) подструктуры блоков, подструктуры каналов.

Говорят, что <идентификатор> является видимым в некотором сегменте,

a) если определяющий контекст именной части <идентификатора> расположен внутри этого сегмента, или
b) если он является видимым в сегменте, определяющим соответствующий сегмент, или
c) если сегмент содержит <частичное определение типа>, в котором определен <идентификатор>, или
d) если сегмент содержит <определение сигнала>, в котором определен <идентификатор>.

Никакие два определения в одном и том же сегменте, принадлежащие одному и тому же классу объектов, не могут иметь одно и то же <имя>. Исключение составляют определения <сигнатуры операций> и <сигнатуры литералов>, входящие в одно и то же <частичное определение типа> (см. § 5.2.2): две или более операции

и/или два или более литерала могут иметь одно и то же <имя>, если у них различны либо <сорта аргументов>, либо сорт <результата>.
Еще одно исключение составляют импортируемые объекты. Для класса этих объектов пары (<импортированное имя>, <сорт>), входящие в <определение импорта>, должны быть различными внутри сегмента.

В конкретной текстовой грамматике необязательные имена или идентификаторы, стоящие в определениях после заключительных ключевых слов (таких как ENDSYSTEM, ENDBLOCK и т.д.), должны синтаксически совпадать с именами или идентификаторами, следующими после соответствующих начинающих ключевых слов (SYSTEM, BLOCK и т.д. соответственно).

2.2.3 *Неформальный текст*

Абстрактная грамматика

Неформальный текст :: ...

Конкретная текстовая грамматика

<неформальный текст> :: =
<знаковая строка>

Семантика

Если при спецификации SDL-системы используется неформальный текст, то это означает, что он не является формальным SDL, то есть SDL не придает ему никакого смысла. Семантика неформального текста должна быть определена какими-нибудь другими средствами.

2.2.4 *Правила вычерчивания*

Размеры графических символов могут быть выбраны самим пользователем.

Границы символов не должны накладываться друг на друга или пересекаться. Исключение из этого правила составляют линейные символы, то есть <символ канала>, <символ маршрута сигнала>, <символ линии создания>, <символ линии потока>, <символ сплошной линии ассоциации> и <символ пунктирной линии ассоциации>, которые могут пересекать друг друга. Нет никакой логической связи между пересекающимися линиями.

Метасимвол **is followed by** неявно содержит <символ линии потока>.

Символы линии могут состоять из одного и более отрезков прямых линий.

Стрелка должна быть указана при <символе линии потока>, когда он входит в другой <символ линии потока>, в <символ выходного коннектора> или в <символ следующего состояния>. В остальных случаях снабжение <символов линии потока> стрелками необязательно. Линии, образующие <символ линии потока>, должны располагаться горизонтально или вертикально.

Допустимы зеркальные отражения по отношению к вертикальной оси <символа ввода>, <символа вывода>, <символа комментария> и <символа расширения текста>.

Правый аргумент метасимвола **is associated with** должен быть ближе к левому аргументу, чем к любому другому графическому символу. Синтаксические элементы правого аргумента должны быть различимы друг от друга.

Текст внутри графических символов должен читаться слева направо, начиная с верхнего левого угла. Правая сторона символа интерпретируется как знак "новая строка"; это означает, что чтение должно быть продолжено с левого края следующей строки, если таковая имеется.

2.2.5 *Членение диаграмм*

Нижеследующее определение членения диаграмм не является частью *Конкретной графической грамматики*, но здесь используется тот же самый метаязык.

<страница> :: =
<символ кадра> **contains**
{ <область заголовка> <область номера страницы>
{ <синтаксическая единица> } * }

<область заголовка> :: =
<неявный символ текста> **contains** <заголовок>

<область номера страницы> :: =
<неявный символ текста> **contains** [<номер страницы> [(<число страниц>)]]

<номер страницы> :: =
<имя литерала>

<число страниц> :: =
<имя натурального литерала>

<страница> является начальным нетерминальным объектом и поэтому ни в одном правиле порождения нет ссылок на него. Диаграмма может быть расчленена по нескольким <страницам>: в этом случае <символ кадра>, ограничивающий диаграмму и <заголовок>, заменяется на <символ кадра> и <заголовок> для каждой <страницы>.

Пользователь SDL может заменить <символ кадра> на границу носителя, на котором изображается диаграмма.

<неявный символ текста> не изображается, но подразумевается с тем, чтобы четко отделить друг от друга <область заголовка> и <область номера страницы>. <область заголовка> помещается в левом верхнем углу <символа кадра>. <область номера страницы> помещается в правом верхнем углу <символа кадра>. <заголовок> и <синтаксическая единица> зависят от типа диаграммы.

2.2.6 *Комментарий*

Комментарий служит для изображения замечаний, касающихся символов или текста.

В *Конкретном текстовом синтаксисе* используются две формы комментария. Первая форма — <замечание> — была определена в § 2.2.1.

Примеры приведены на рис. 2.9.1 и на рис. 2.9.3.

Конкретным синтаксисом второй формы является:

<конец> :: =
[<комментарий>];

<комментарий> :: =
COMMENT <знаковая строка>

Пример приведен на рис. 2.9.2.

В *Конкретной графической грамматике* используется нижеследующий синтаксис:

<область комментария> :: =
<символ комментария> **contains** <текст>
is connected to <символ пунктирной линии ассоциации>

<символ комментария> :: =

<символ пунктирной линии ассоциации> :: =

– – – – – – – – – – –

Один конец <символа пунктирной линии ассоциации> должен быть присоединен к середине вертикального отрезка <символа комментария>.

<символ комментария> может быть соединен с любым графическим символом с помощью <символа пунктирной линии ассоциации>. <символ комментария> считается замкнутым символом за счет замыкания (в воображении) прямоугольника так, чтобы текст содержался внутри замыкания. Он содержит комментирующий текст, касающийся графического символа.

Пример приведен на рис. 2.9.4 в § 2.9.

2.2.7 *Расширение текста*

<область расширения текста> :: =
<символ расширения текста> **contains** <текст>
is connected to <символ сплошной линии ассоциации>

<символ расширения текста> :: =
<символ комментария>

<символ сплошной линии ассоциации> :: =

———————————

Один конец <символа сплошной линии ассоциации> должен быть присоединен к середине вертикального отрезка <символа расширения текста>.

<символ расширения текста> может быть соединен с любым графическим символом с помощью <символа сплошной линии ассоциации>. <символ расширения текста> считается замкнутым за счет замыкания (в воображении) прямоугольника.

Текст, содержащийся в <символе расширения текста>, является продолжением текста, расположенного внутри графического символа, и считается содержащимся внутри этого символа.

2.2.8 *Символ текста*

<символ текста> используется в любой <диаграмме>. Содержание текста зависит от диаграммы.

<символ текста> :: =

2.3 *Основные понятия данных*

Понятие данных в SDL определено в § 5; определена терминология данных SDL, средства определения новых типов данных и предопределенные свойства данных.

Данные встречаются в определениях типов данных, в выражениях, в применениях операций, в переменных, в значениях и в литералах.

2.3.1 *Определения типов данных*

Принципом SDL является рассмотрение данных в связи с типами данных. Тип данных определяет множество значений, набор операций, которые могут быть применены к этим значениям, и набор алгебраических правил (равенств), описывающих поведение этих операций, когда они применяются к значениям. Значения, операции и алгебраическне правила в совокупности определяют свойства типа данных. Эти свойства описываются с помощью определений типов данных.

SDL допускает определение любого требуемого типа данных, включая механизмы композиции (составные типы), при единственном условии, что такое определение может быть формально специфицировано. В отличие от этого в языках программирования имеются реализационные соображения, которые требуют, чтобы совокупность допустимых типов данных и, в частности, предоставляемые механизмы композиции (массивы, структуры и т.д.) были ограничены.

2.3.2 *Переменная*

Переменные являются объектами, которые могут быть ассоциированы со значениями с помощью присваивания. При обращении к переменной выдается (возвращается) ассоциированное с ней значение.

2.3.3 *Значения и литералы*

Совокупность значений, обладающих некоторыми характеристиками, называется сортом. В определении операции указывается, каких сортов могут быть его операнды и каких сортов — результаты. Например, допустимо применение операции сложения (" + ") из целого сорта в целый сорт, в то время как применение той же операции к Булевскому сорту недопустимо.

Любой сорт содержит хотя бы одно значение. Любое значение принадлежит к одному и только к одному сорту, то есть никакие два сорта не имеют общих элементов.

Для большинства сортов существуют литеральные формы, обозначающие значения, принадлежащие этому сорту (например, для целого сорта используется литерал "2", а не скажем "1 + 1"). Для обозначения одного и того же значения может существовать более одного литерала (например, одно и то же целое значение можно изобразить как с помощью литерала 12, так и с помощью литерала 012). Один и тот же литерал может использоваться для более чем одного сорта: например, литерал "A" изображает как знак, так и знаковую строку длины один. Некоторые сорта вообще не имеют литералов, например значения составного типа зачастую не имеют самостоятельного литерала. Их значения определяются операциями композиции над значениями их компонентов.

2.3.4 *Выражения*

Выражение обозначает значение. Если выражение не содержит переменной, например, если оно является литералом заданного сорта, то любое появление этого выражения будет всегда обозначать одно и то же значение. Интерпретация выражений (при интерпретации SDL-системы), содержащих переменные, может давать различ-

ные значения в зависимости от текущих значений, ассоциируемых с переменными.

2.4 *Структура системы*

2.4.1 *Удаленные определения*

Удаленным называется определение, которое было удалено из определяющего его контекста, в целях достижения большей обозримости. Оно подобно определению макроса (см. § 4.2) с той лишь разницей, что "вызвать" его можно из ровно одного места (определяющего контекста) посредством ссылки.

Конкретная грамматика

```
<удаленное определение> :: =
        <определение> | <диаграмма>

<определение системы> :: =
            { <текстовое определение системы> | <диаграмма системы> }
            { <удаленное определение> } *

<определение> :: =
            <определение блока>
        |   <определение процесса>
        |   <определение процедуры>
        |   <определение подструктуры блока>
        |   <определение подструктуры канала>
        |   <определение сервиса>
        |   <макроопределение>

<диаграмма> :: =
            <диаграмма блока>
        |   <диаграмма процесса>
        |   <диаграмма процедуры>
        |   <диаграмма подструктуры блока>
        |   <диаграмма подструктуры канала>
        |   <диаграмма сервиса>
        |   <диаграмма макроса>
```

Для любого <удаленного определения>, исключая <определение макроса> и <диаграмму макроса>, должна иметься ссылка на это определение; ссылка может находиться либо в <определении системы>, либо в <диаграмме системы>, либо в другом <удаленном определении>.

Для любой ссылки должно иметься соответствующее <удаленное определение>.

В любом <удаленном определении> непосредственно после ключевого слова должен стоять <идентификатор>. <квалификатор> этого <идентификатора> должен быть либо полон, либо опущен. Если <квалификатор> полностью опущен, то <имя> должно быть уникальным в определении системы для того класса объектов, которому принадлежит <удаленное определение>. Использование <квалификатора> в <идентификаторе>, стоящем после ключевого слова, не разрешается, если это определение не является <удаленным определением> (иными словами, обычные определения должны специфицировать некоторое <имя>).

Семантика

Прежде чем <определение конкретной системы> может быть проанализировано, любая имеющаяся в ней ссылка должна быть заменена на соответствующее <удаленное определение>. При этой замене <идентификатор> <удаленного определения> заменяется на <имя>, указанное в ссылке.

2.4.2 *Система*

Абстрактная грамматика

Определение-системы	::	*Имя-системы* *Определение-блока*-set *Определение-канала*-set *Определение-сигнала*-set *Определение-типа-данных* *Определение-синонимичного-типа*-set
Имя-системы	=	*Имя*

Определение-системы имеет имя, которое может быть использовано в квалификаторах.

Определение-системы должно содержать одно *определение-блока.*

Определения всех сигналов, каналов, типов данных, синонимичных типов, используемых в интерфейсе с окружающей средой и между блоками системы, должны содержаться в *Определении-системы.* Все предопределенные данные считаются определенными на уровне системы.

Конкретная текстовая грамматика

<текстовое определение системы> :: =
SYSTEM <имя системы> <конец>
{ <определение блока>
| <текстовая ссылка на блок>
| <определение канала>
| <определение сигнала>
| <определение списка сигналов>
| <определение отбора>
| <макроопределение>
| <определение данных> } +
ENDSYSTEM [<имя системы>] <конец>

<текстовая ссылка на блок> ::=
BLOCK <имя блока> REFERENCED <конец>

<определение отбора> приведено в § 4.3.3, <макроопределение> — в § 4.2, <определение данных> — в § 5.5.1, <определение блока> — в § 2.4.3, <определение канала> — в § 2.5.1, <определение сигнала> — в § 2.5.4, <определение списка сигналов> — в § 2.5.5.

Пример <определения системы> приведен на рис. 2.9.5 в § 2.9.

Конкретная графическая грамматика

<диаграмма системы> :: =
{ <символ кадра> contains
{ <заголовок системы>
{ { <область текста системы> } *
{ <диаграмма макроса> } *
<область взаимодействия блоков> } set }

<символ рамки> :: =

<заголовок системы> :: =
SYSTEM <имя системы>

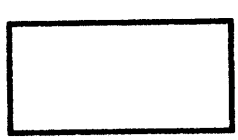
<область текста системы> :: =
<символ текста> contains
{ <определение сигнала>
| <определение списка сигналов>
| <определение данных>
| <макроопределение>
| <определение отбора> } *

<область взаимодействия блоков> :: =
{ <область блока>
| <область определения канала> } +

<область блока> :: =
| <графическая ссылка на блок>
| <диаграмма блока>

<графическая ссылка на блок> :: =
<символ блока> contains <имя блока>

<символ блока> :: =



<определение отбора> описано в § 4.3.3, <макроопределение> и <диаграммы макроса> — в § 4.2, <определение данных> — в § 5.5.1, <диаграмма блока> — в § 2.4.3, <область определения канала> — в § 2.5.1, <определение сигнала> — в § 2.5.4, <определение списка сигналов> — в § 2.5.5.

Определение-блока-set в *Абстрактной грамматике* соответствует <областям блока>, а *Определение-канала-set* соответствует <области определения канала>.

Пример <диаграммы системы> приведен на рис. 2.9.6

Семантика

Определение-системы является SDL-изображением спецификации и описания системы.

Система отделена от окружающей ее среды границей системы и содержит совокупность блоков. Связь между системой и окружающей средой или между блоками в системе может осуществляться только с помощью сигналов. Внутри системы эти сигналы передаются по каналам. Каналы соединяют блоки друг с другом или с границей системы.

Прежде чем начать интерпретацию *Определения-системы*, должно быть выделено согласованное подмножество (см. § 3.2.1). Это подмножество называется экземпляром *Определения-системы*. Экземпляр системы является реализацией (воплощением) типа системы, заданного *Определением-системы*. Интерпретация экземпляра *Определения-системы* выполняется абстрактной SDL-машиной, которая таким образом придает семантический смысл понятиям SDL. Интерпретация экземпляра *Определения-системы* состоит из:

a) инициации времени системы,
b) интерпретации блоков и соединяющих их каналов, содержащихся в выбранном согласованном подмножестве разбиения.

2.4.3 *Блок*

Абстрактная грамматика

Определение-блока	::	*Имя-блока* *Определение-процесса-***set** *Определение-сигнала-***set** *Соединение-канала-с-маршрутом-***set** *Определение-маршрута-сигнала-***set** *Определение-типа-данных* *Определение-подтипа-***set** [*Определение-подструктуры-блока*]
Имя-блока	=	*Имя*

Если *Определение-блока* не содержит *Определения-подструктуры-блока*, то блок должен содержать хотя бы одно *Определение-процесса* и одно *Определение-маршрута-сигнала.*

Допустимо осуществление разбиений блоков, специфицирующих *Определение-подструктуры-блока*; это свойство изучается в § 3.2.2.

Конкретная текстовая грамматика

<определение блока> :: =
BLOCK { <имя блока> | <идентификатор блока> } <конец>
{ <определение сигнала>
| <определение списка сигналов>
| <определение процесса>
| <текстовая ссылка на процесс>
| <определение маршрута сигнала>
| <макроопределение>
| <определение данных>
| <определение отбора>
| <соединение канала с маршрутом> } *
[<определение подструктуры блока> | <текстовая ссылка на подструктуру блока>]
ENDBLOCK [<имя блока> | <идентификатор блока>] <конец>

<текстовая ссылка на процесс> ::=
PROCESS <имя процесса> [<число экземпляров>] REFERENCED <конец>

<определение сигнала> описано в § 2.5.4, <определение списка сигналов> — в § 2.5.5, <определение процесса> — в § 2.4.4, <определение маршрута сигнала> — в § 2.5.2, <соединение канала с маршрутом> — в § 2.5.3, <определение подструктуры блока> и <текстовая ссылка на подструктуру блока> — в § 3.2.2, <макроопределение> — в § 4.2.2, <определение данных> — в § 5.5.1.

Пример <определения блока> приведен на рис. 2.9.7 в § 2.9.

Конкретная графическая грамматика

<диаграмма блока> ::=
<символ кадра>
contains { <заголовок блока>
{{ <область текста блока>} * { <диаграмма макроса> } *
[<область взаимодействия процессов>][<область подструктуры блока>] { set}
is associated with {<идентификатор канала>} *

<идентификатор канала> идентифицирует канал, присоединенный к маршруту сигнала в <диаграмме блока>. Этот канал помещается вне <символа кадра> впритык к той точке, в которой заканчивается маршрут сигнала на <символе кадра>. Если <диаграмма блока> не содержит <области взаимодействия процессов>, то она должна содержать <область подструктуры блока>.

<заголовок блока> ::=
BLOCK <имя блока> | <идентификатор блока>}

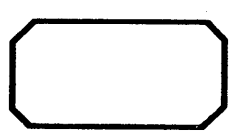
<область текста блока> ::=
<область текста системы>

<область взаимодействия процессов> ::=
{<область процесса>
| <область линии порождения>
| <область определения маршрута сигнала>} +

<область процесса> ::=
<графическая ссылка на процесс> | <диаграмма процесса>

<графическая ссылка на процесс> ::=
<символ процесса> **contains** { <имя процесса> [<число экземпляров>] }

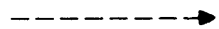
<символ процесса> ::=



<число экземпляров> определено в § 2.4.4.

<область линии порождения> ::=
<символ линии порождения>
is connected to { <область процесса> <область процесса> }

<символ линии порождения> :: =



Стрелка <символа линии порождения> указывает на ту <область процесса>, в отношении которой осуществляется акт порождения.

<диаграмма процесса> описана в § 2.4.4, <область определения маршрута сигнала> — в § 2.5.2, <область подструктуры блока> — в § 3.2.2, <диаграмма макроса> — в § 4.2.2.

Пример <диаграммы блока> приведен на рис. 2.9.8 в § 2.9.

Семантика

Определение блока является контейнером, в котором размещены один или несколько определений процессов системы и/или подструктура блока. Целью определения блока является группировка процессов, которые в совокупности реализуют некоторую функцию либо непосредственно, либо за счет подструктуры блока.

Определение блока обеспечивает статический интерфейс связи, с помощью которого процессы, содержащиеся в блоке, могут вступать в контакт с другими процессами. Кроме того, оно служит областью видимости для определений процессов.

Чтобы интерпретировать блок, необходимо создать начальные процессы этого блока.

2.4.4 *Процесс*

Абстрактная грамматика

Определение-процесса	::	*Имя-процесса* *Число-экземпляров* *Формальные-параметры-процесса** *Определение-процедуры-set* *Определение-сигнала-set* *Определение-типа-данных* *Определение-подтипа-set* *Определение-переменной-set* *Определение-обозревания-set* *Определение-таймера-set* *Граф-процесса*
Число-экземпляров	::	*Intg Intg*
Имя-процесса	=	*Имя*
Граф-процесса	::	*Стартовая-вершина-процесса* *Вершина-состояния-set*
Формальные-параметры-процесса	::	*Имя-переменной* *Идентификатор-ссылки-на-сорт*

Конкретная текстовая грамматика

<Определение процесса> :: =
PROCESS { <идентификатор процесса> | <имя процесса> }
[<число экземпляров>] <конец>
[<формальные параметры> <конец>] [<совокупность допустимых входных сигналов>]
{ <определение сигнала>
| <определение списка сигналов>
| <определение процедуры>
| <текстовая ссылка на процедуру>
| <макроопределение>
| <определение данных>
| <определение переменной>
| <определение обозревания>
| <определение отбора>
| <определение импорта>
| <определение таймера>} *
{ <тело процесса>
| <разложение на сервисы>}

ENDPROCESS [<идентификатор процесса> | <имя процесса>] <конец>

<текстовая ссылка на процедуру> :: =
PROCEDURE <имя процедуры> REFERENCED <конец>

<совокупность допустимых входных сигналов> :: =
SIGNALSET [<список сигналов>] <конец>

<тело процесса> :: =
<старт> { <состояние> } *

<формальные параметры> :: =
FPAR <имя переменной> { , <имя переменной> } * <сорт>
{ , <имя переменной> { , <имя переменной> } * <сорт> } *

<число экземпляров> :: =
([<начальное число>],[<максимальное число>])

<начальное число> :: =
<натуральное простое выражение>

<максимальное число> :: =
<натуральное простое выражение>

Начальное число экземпляров и максимальное число экземпляров, содержащиеся в *Числе-экземпляров*, получены из <числа экземпляров>. Если <начальное число> пропущено, то оно равно 1. Если <максимальное число> пропущено, то оно ничем не ограничено.

<число экземпляров>, используемых при получении указанных выше значений, является следующим:

а) Если процесс не содержит <текстовой ссылки на процесс>, то используется <число экземпляров> в <определении процесса>. Если оно не содержит <числа экземпляров>, то используется <число экземпляров>, в котором опущены и <начальное число> и <максимальное число>.

b) Если и <число экземпляров> в <определении процесса> и <число экземпляров> в <текстовой ссылке на процесс> опущены, то используется <число экземпляров>, в котором опущены и <начальное число> и <максимальное число>.

c) Если опущено либо <число экземпляров> в <определении процесса>, либо <число экземпляров> в <текстовой ссылке на процесс>, то в качестве <числа экземпляров> берется то, которое не опущено.

d) Если специфицированы и <число экземпляров> в <определении процесса> и <число экземпляров> в <текстовой ссылке на процесс>, то эти два <числа экземпляров> должны быть лексически эквивалентны; в этом случае используется это <число экземпляров>.

Аналогичные правила относятся к <числу экземпляров>, специфицированному в <диаграмме процесса> и в <графической ссылке на процесс>, как они определяются ниже.

<определение сигнала> описано в § 2.5.4, <определение списка сигналов> — в § 2.5.5, <определение обозревания> — в § 2.6.1.2, <определение переменной> — в § 2.6.1.1, <определение процедуры> — в § 2.4.5, <определение таймера> — в § 2.8, <макроопределение> — в §4.2.2, <определение импорта> — в § 4.1.3, <определение отбора> — в § 4.3.3, <простое выражение> — в § 4.3.2, <разложение на сервисы> — в § 4.10.1, <определение данных> — в § 5.5.1.

<начальное число> экземпляров должно быть не больше или равно <максимальному числу>, которое само должно быть больше нуля.

Использование <совокупности допустимых входных сигналов> описано в разделе *Модель* в § 2.5.2.

Пример <определения процесса> приведен на рис. 2.9.9 в § 2.9.

Конкретная графическая грамматика

```
<диаграмма процесса> :: =
        <символ кадра>
        contains    { <заголовок процесса>
                    {{ <область текста процесса> } *
                                { <область процедуры> } *
                                { <диаграмма макроса> } *
                                { <область графа процесса>
                                | <область взаимодействия сервисов> }} set }
        [is associated with { <идентификатора маршрута сигнала> } + ]
```

<идентификатор маршрута сигнала> идентифицирует внешний маршрут сигнала, присоединенный к маршруту сигнала внутри <диаграммы процесса>. Он помещается вне <символа кадра> впритык к той точке, в которой заканчивается маршрут сигнала на <символе кадра>.

```
<область текста процесса> :: =
                <символ текста> contains }
                        { [<совокупность допустимых входных сигналов>]
                        { <определение сигнала>
                    |   <определение списка сигналов>
                    |   <определение переменной>
                    |   <определение обозревания>
                    |   <определение импорта>
                    |   <определение данных>
                    |   <макроопределение>
                    |   <определение таймера>
                    |   <определение отбора> } * }
```

<заголовок процесса> ::=
PROCESS { <имя процесса> | <идентификатор процесса> }
[<число экземпляров> [<конец>]]
[<формальные параметры>]

<область графа процесса> ::=
<область старта> {<область состояния> | <область входного коннектора>} *

<определение сигнала> описано в § 2.5.4, <определение списка сигналов> — в § 2.5.5, <определение обозревания> — в § 2.6.1.2, <определение переменной> — в § 2.6.1.1, <область процедуры> — в § 2.4.5, <определение таймера> — в § 2.8, <макроопределение> и <диаграммы макроса> — в § 4.2.2, <определение импорта> — в § 4.1.3, <определение отбора> — в § 4.3.3, <определение данных> — в § 5.5.1, <область старта> — в § 2.6.2, <область состояния> — в § 2.6.3, <область входного коннектора> — в § 2.6.6, <область взаимодействия сервисов> — в § 4.10.1.

Пример <диаграммы процесса> приведен на рис. 2.9.10 в § 2.9

Семантика

Определение процесса вводит тип процесса, который предназначен для представления динамического поведения.

В *Числе-экземпляров* первое число означает *Число-экземпляров* процесса, которые существуют в момент создания системы, а второе число означает максимально допустимое число одновременно существующих экземпляров процесса данного типа.

Экземпляр процесса является взаимодействующим обобщенным конечным автоматом, выполняющим некоторое определенное множество действий, называемых переходами; эти действия выполняются в соответствии с получением поданного сигнала всякий раз, когда автомат находится в некотором состоянии. После завершения перехода процесс оказывается ждущим в некотором состоянии, не обязательно отличном от первоначального.

Концепция конечного автомата обобщена в том смысле, что на состояние, в котором оказывается автомат после перехода, влияет не только сигнал, породивший переход, но и решения, принятые процессом в результате опроса известных ему переменных.

В системе несколько экземпляров процесса одного и того же типа могут существовать одновременно и работать асинхронно и параллельно как друг с другом, так и с экземплярами других типов.

При создании системы начальные процессы создаются в случайном порядке. Обмен сигналами между процессами начинается только после того, как все процессы оказываются созданными. Формальные параметры этих процессов инициализируются неопределенными значениями.

Экземпляры процессов существуют либо с момента создания системы, либо могут быть созданы запросом на создание, который выдают интерпретируемые процессы; сама интерпретация начинается с момента, когда интерпретируется стартовое действие; они прекращают свое существование, выполнив действие "стоп".

Сигналы, получаемые экземплярами процессов, обозначены как входные сигналы, а сигналы, посылаемые экземплярам процессом, обозначены как выходные.

Сигнал может быть воспринят некоторым экземпляром процесса, только если он находится в некотором состоянии. Полная совокупность допустимых входных сигналов является объединением всех сигналов всех маршрутов сигналов, ведущих к процессу <совокупности допустимых входных сигналов>, неявных сигналов и сигналов от таймера.

С каждым экземпляром процесса ассоциируется один и только один входной порт. Когда к процессу поступает входной сигнал, он помещается во входной порт данного процесса.

Процесс либо ожидает, находясь в некотором состоянии, либо он активен, выполняя переход. Для каждого состояния имеется список сохраняемых сигналов (см. также § 2.6.5). Если процесс находится в состоянии ожидания, то первый же сигнал, идентификатор которого не входит в совокупность сохраняемых сигналов, извлекается из очереди и воспринимается процессом.

Входной порт может вмещать любое число входных сигналов, причем несколько входных сигналов устанавливаются в очередь к экземпляру процесса. Множество сигналов, поступивших в порт, помещаются в очередь в порядке их поступления во времени. Если два или более сигналов поступают "одновременно" по разным маршрутам, они помещаются в очередь в случайном порядке.

При создании процесса ему придается пустой порт; одновременно создаются локальные переменные с присваиваемыми им значениями.

Формальные параметры являются переменными, которые создаются либо тогда, когда создается система (но им еще не переданы никакие фактические параметры и поэтому они пока не инициализированы), либо тогда, когда экземпляр процесса создается динамически.

Любой экземпляр процесса может использовать четыре выражения, вырабатывающие значения PId (см. § 5.6.10). Этими выражениями являются SELF (сам процесс), PARENT (родитель), OFFSPRING (потомок) и SENDER (отправитель). Они выдают результат для:

a) экземпляра самого процесса (SELF),

b) экземпляра процесса, создавшего данный процесс (PARENT),

c) последнего экземпляра процесса, созданного данным процессом (OFFSPRING),

d) экземпляра процесса, от которого был получен последний воспринятый сигнал (SENDER) (см. также § 2.6.4).

Эти выражения изучаются подробнее в § 5.5.4.3.

SELF, PARENT, OFFSPRING, SENDER могут использоваться в выражениях внутри экземпляров процессов.

Для всех экземпляров процессов, существующих в момент инициализации системы, предопределенное выражение PARENT имеет значение NULL.

Для всех вновь создаваемых процессов предопределенные выражения OFFSPRING и SENDER имеют значение NULL.

2.4.5 *Процедура*

Процедуры описываются средствами определения процедуры. Процедура вызывается посредством обращения к процедуре, ссылающегося на определение процедуры. С вызовом процедуры связаны параметры: они используются как для передачи значений, так и для управления видимостью переменных при выполнении процедуры. Механизм передачи параметров управляет тем, на какие из переменных воздействует интерпретация процедуры.

Абстрактная грамматика

Определение-процедуры :: Имя-процедуры
Формальные-параметры-процедуры*
Определение-процедуры-set
Определение-типа-данных
Определение-подтипа-set
Определение-переменной-set
Граф-процедуры

Имя-процедуры = *Имя*
Формальные-параметры-процедуры = *Входной-параметр* |
Входной-выходной-параметр
Входной-параметр :: *Имя-переменной*
Идентификатор-ссылки-на-сорт
Входной-выходной-параметр :: *Имя-переменной*
Идентификатор-ссылки-на-сорт
Граф-процедуры :: *Стартовая-вершина-процедуры*
Вершина-состояния-set
Стартовая-вершина-процедуры :: *Переход*

Конкретная-текстовая-грамматика

<определение процедуры> :: =
PROCEDURE { <идентификатор процедуры> | <имя процедуры> } <конец>
[<формальные параметры процедуры> <конец>]
{ <определение данных>
| <определение переменной>
| <текстовая ссылка на процедуру>
| <определение процедуры>
| <определение отбора>
| <макроопределение> } *
<тело процедуры>

ENDPROCEDURE [<имя процедуры> <идентификатор процедуры>] <конец>

<формальные параметры процедуры> :: =
FPAR <формальный параметр-переменная>
{ , <формальный параметр-переменная> } *

<формальный параметр-переменная> :: =
[IN/OUT
| IN]
<имя переменной> { , <имя переменной> } * <сорт>

<тело процедуры> :: =
<тело процесса>

<определение переменной> описано в § 2.6.1.1, <текстовая ссылка на процедуру> — в § 2.4.4, <макроопределение> — в § 4.2, <определение отбора> — в § 4.3.3, <определение данных> — в § 5.5.1, <сорт> — в § 5.2.2.

<определение переменной>, входящее в <определение процедуры>, не должно содержать <имени переменной>, снабженной одним из ключевых слов REVEALED, EXPORTED, REVEALED EXPORTED, EXPORTED REVEALED (см. § 2.6.1).

Пример <определение процедуры> приведен на рис. 2.9.11.

Конкретная графическая грамматика

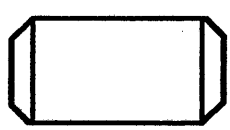
<диаграмма процедуры> :: =
<символ кадра> contains { <заголовок процедуры>
{{ <область текста процедуры>
| <область процедуры>
| <диаграмма макроса> } *
<область графа процедуры> } set }

<область процедуры> :: =
<графическая ссылка на процедуру>
| <диаграмма процедуры>

<область текста процедуры> :: =
<символ текста> contains
{ <определение переменной>
| <определение данных>
| <определение отбора>
| <макроопределение> } *

<графическая ссылка на процедуру> :: =
<символ процедуры> contains <имя процедуры>

<символ процедуры> :: =

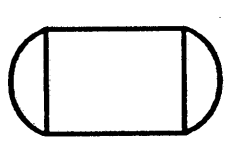


<заголовок процедуры> :: =
PROCEDURE { <имя процедуры> | <идентификатор процедуры> }
[<формальные параметры процедуры>]

<область графа процедуры> :: =
<область старта процедуры>
{ <область состояния> | <область входного коннектора> } *

<область старта процедуры> :: =
<символ старта процедуры> is followed by <область перехода>

<символ старта процедуры> :: =



<определение переменной> приведено в § 2.6.1.1, <область перехода> — в § 2.6.7.1, <область состояния> — в § 2.6.3, <область входного коннектора> — в § 2.6.6, <макроопределение> и <диаграмма макроса> — в § 4.2, <определение отбора> — в § 4.3.3, <определение данных> — в § 5.5.1.

Пример <диаграммы процедуры> приведен на рис. 2.9.12 в § 2.9.

Семантика

Процедура является средством, которое присваивает имя совокупности элементов и представляет эту совокупность одной ссылкой. Правила процедуры накладывают определенную дисциплину на метод выбора совокупности элементов и ограничивают область видимости имен переменных, определенных в процедуре.

Переменная процедуры является ее локальной переменной, которая не может быть ни раскрываемой, ни обозреваемой, ни экспортируемой, ни импортируемой. Она создается тогда, когда интерпретируется стартовая вершина процедуры, и прекращает свое существование, когда интерпретируется вершина возврата графа процедуры.

Одновременно с интерпретацией определения процедуры интерпретируется ее граф.

Определение процедуры интерпретируется только тогда, когда экземпляр процесса обращается к нему; интерпретацию осуществляет именно этот экземпляр процесса.

Интерпретация определения процедуры вызывает создание экземпляра процедуры; начинается ее интерпретация, которая выполняется следующим образом:

a) Для каждого *Входного-параметра* создается локальная переменная, *Имя* и *Сорт* которой совпадают с *Именем* и *Сортом Входного-параметра.* Этой переменной присваивается значение выражения, задаваемого соответствующим фактическим параметром; это значение может быть неопределенным.

b) Если фактический параметр пуст, то соответствующему формальному параметру присваивается неопределенное значение.

c) Формальный параметр, не имеющий явного атрибута, имеет неявный атрибут IN.

d) Для каждого *Определения-переменной* в *Определении-процедуры* создается локальная переменная, *Имя* и *Сорт* которой совпадают с *Именем* и *Сортом* в *Определении-переменной.*

e) Каждый *Входной-выходной-параметр* является синонимичным именем той переменной, которая указана в выражении для фактического параметра. Это синонимичное имя используется на протяжении всей интерпретации *Графа-процедуры* при обращениях к значению переменной или присвоениях этой переменной новых значений.

f) Интерпретируется *Переход*, содержащийся в *Стартовой-вершине-процедуре.*

2.5 *Связь*

2.5.1 *Канал*

Абстрактная грамматика

Определение-канала :: *Имя-канала*
Путь канала
[*Путь-канала*]

Путь канала :: *Исходный блок*
Приемный блок
Идентификатор-сигнала-**set**

Исходный блок = *Идентификатор-блока* |
ENVIRONMENT

Приемный блок = *Идентификатор-блока* |
ENVIRONMENT

Идентификатор-блока = *Идентификатор*

Идентификатор-сигнала = *Идентификатор*
Имя-канала = *Имя*

Идентификатор-сигнала-**set** должен содержать список всех сигналов, которые могут быть переданы по пути-канала (путям-канала), определяемых каналом.

По крайней мере одна из конечных точек канала должна быть блоком. Если обе конечные точки являются блоками, то блоки должны быть различными.

Конечные точки блока должны быть определены в пределах той же области видимости, что и сам канал.

Конкретная текстовая грамматика

<определение канала> :: =
CHANNEL <имя канала>
<путь канала>
[<путь канала>]
[<определение подструктуры канала>
| <текстовая ссылка на подструктуру канала>]
ENDCHANNEL [<имя канала>] <конец>

<путь канала> :: =
{ FROM <идентификатор блока> TO <идентификатор блока>
| FROM <идентификатор блока> TO ENV
| FROM ENV TO <идентификатор блока> }
WITH <список сигналов> <конец>

<список сигналов> описан в § 2.5.5, <определение подструктуры канала> и <текстовая ссылка на подструктуру канала> — в § 3.2.3.

В случае, если определяются два <пути канала>, один из них должен идти в направлении, обратном другому.

Конкретная графическая грамматика

<область определения канала> :: =
<символ канала>
is associated with { <имя канала> } [{ <идентификатор канала> | <идентификатор блока> }]
<область списка сигналов> [<область списка сигналов>] } **set** }
is connected to { <область блока> { <область блока> | <символ кадра> }
[<область ассоциирования подструктуры канала>] } **set**

<идентификатор канала> идентифицирует внешний канал, присоединенный к <диаграмме подструктуры блока>, ограниченной <символом кадра>. <идентификатор блока> идентифицирует внешний блок, являющийся конечной точкой для <диаграммы подструктуры канала>, ограниченной <символом кадра>.

<символя кадра> :: =
<символ канала 1>
| <символ канала 2>
| <символ канала 3>

<символ канала 1> :: =

<символ канала 2> :: =

<символ канала 3> :: =

<область списка сигналов> описана в § 2.5.5, <область блока> и <символ кадра> — в § 2.4.1, а <область ассоциирования подструктуры канала> — в § 3.2.3.

С каждой стрелкой на <символе канала> должна быть связана <область списка сигналов>. <область списка сигналов> должна быть помещена достаточно близко к стрелке, чтобы не вызывать никаких разночтений.

Семантика

Канал представляет собой средства передачи сигналов. Канал может рассматриваться как один или два независимых односторонних путей между двумя блоками или между блоком и окружающей его средой.

Сигналы, передаваемые по каналу, доставляются в конечную точку приемника.

Сигналы поставляются в конечной точке получателю в том же порядке, в котором они были поданы в начальной точке канала. Если два или более сигналов подаются на канал одновременно, то доставляются они в случайном порядке.

Канал может задерживать передаваемые по нему сигналы. Это означает, что с каждым направлением канала ассоциирована очередь, организованная по дисциплине FIFO (First-In-First-Out — Первым пришел в — Первым ушел из). Сигнал, поданный на канал, помещается в очередь задержки. После недетерминированного и непостоянного отрезка времени первый в очереди экземпляр сигнала освобождается и подается на один из каналов или маршрутов сигналов, которые подсоединены к каналу.

Между одной и той же парой конечных точек может существовать несколько каналов. Сигнал одного и того же типа может передаваться по разным каналам.

2.5.2 *Маршрут сигнала*

Абстрактная грамматика

Определение-маршрута-сигнала	::	*Имя-маршрута-сигнала* *Путь-маршрута-сигнала* [*Путь-маршрута-сигнала*]
Путь-маршрута-сигнала	::	*Исходный-процесс* *Приемный процесс* *Идентификатор-сигнала*-set
Исходный процесс	=	*Идентификатор-процесса* | ENVIRONMENT
Приемный-процесс	=	*Идентификатор-процесса* | ENVIRONMENT
Имя-маршрута-сигнала	=	*Имя*

По крайней мере, одной из конечных точек *Пути-маршрута-сигнала* должен быть процесс.

Если обе конечные точки являются процессами, то *Идентификаторы-процесса* должны быть различными.

Конечные точки процесса должны быть определены в пределах той же области видимости, в которой определен маршрут сигнала.

Конкретная текстовая грамматика

<определение маршрута сигнала> :: =
SIGNALROUTE <имя маршрута сигнала>
<путь маршрута сигнала>
[<путь маршрута сигнала>]

<путь маршрута сигнала> :: =
{ FROM <идентификатор процесса> TO <идентификатор процесса>
| FROM <идентификатор процесса> TO ENV
| FROM ENV TO <идентификатор процесса> }
WITH <список сигналов> <конец>

<список сигналов> описан в § 2.5.5.

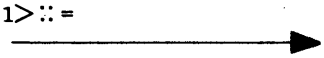
В случае, если определяются два <пути маршрута сигнала>, один из них должен идти в направлении, обратном другому.

Конкретная графическая грамматика

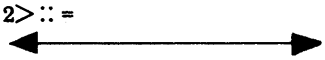
<область определения маршрута сигнала> :: =
<символ маршрута сигнала>
is associated with { <имя маршрута сигнала>
{ [<идентификатор канала>] <область списка сигналов> [<область списка сигналов>] } **set** }
is connected to
{ <область процесса> { <область процесса> | <символ кадра> }} **set**

<символ маршрута сигнала> :: =
<символ маршрута сигнала 1> | <символ маршрута сигнала 2>

<символ маршрута сигнала 1> :: =



<символ маршрута сигнала 2> :: =



Символ маршрута сигнала содержит стрелку у одного конца (односторонний маршрут) или у обоих концов (двусторонний маршрут); стрелки указывают направление движения сигналов.

С каждой стрелкой <символа маршрута сигнала> должна быть связана <область списка сигналов>. <область списка сигналов> должна быть помещена достаточно близко к стрелке, чтобы не вызвать никаких разночтений.

Если <символ маршрута сигнала> подсоединен к <символу кадра>, то <идентификатор канала> идентифицирует тот канал, к которому подсоединен маршрут сигнала.

Семантика

Маршрут сигнала изображает путь передачи сигналов. Маршрут сигнала может рассматриваться как один или два независимых односторонних пути прохождения сигналов между двумя процессами или между процессом и окружающей его средой.

Сигналы, передаваемые по маршруту сигнала, доставляются в конечную точку приемника.

Маршрут сигнала не задерживает передаваемые по нему сигналы.

Никакой маршрут сигнала не может соединять два экземпляра процесса одного и того же типа. В этом случае интерпретация *Выходного-узла* подразумевает помещение сигнала непосредственно в порт приемного процесса.

Между одной и той же парой конечных точек может существовать несколько маршрутов сигналов. Сигнал одного и того же типа может передаваться по различным маршрутам.

Модель

<совокупность допустимых входных сигналов> содержит те сигналы, которые может получать процесс. Однако <совокупность допустимых входных сигналов> не должна содержать сигналов таймеров. Если некоторое <определение блока> содержит <определение маршрута сигнала>, то <совокупность допустимых входных сигналов>, если таковая существует, не должна содержать тех сигналов, которые передаются по маршрутам сигналов, ведущих к процессу.

Если <определение блока> не содержит <определения маршрута сигнала>, то тогда все <определения процессов>, содержащиеся в этом <определении блока> должны содержать <совокупность допустимых входных сигналов>. В этом случае <определения маршрутов сигналов> и <соединение канала с маршрутами> выводятся из < совокупности допустимых входных сигналов>, <выходов> и каналов, заканчивающихся на границе блока. Сигналы, соответствующие заданному направлению по неявному маршруту сигналов между двумя процессами, образуются как пересечение сигналов, специфицированных в <совокупности допустимых входных сигналов> приемного процесса, и сигналов, указанных в выходах порождающего процесса. Если одной из конечных точек является окружающая среда, то входной совокупностью/выходной совокупностью для этой конечной точки является совокупность сигналов, подаваемых каналам в заданном направлении.

2.5.3 *Соединение*

Абстрактная грамматика

Соединение-канала-с-маршрутом :: *Идентификатор-канала*
Идентификатор-маршрута-сигнала-set

Идентификатор-маршрута-сигнала = *Идентификатор*

Другие конструкции соединения описаны в § 3.

Каждый *Идентификатор-канала*, присоединенный к охватывающему блоку, должен быть указан точно в одном *Соединении-канала-с-маршрутом*. *Идентификатор-канала*, входящий в *Соединение-канала-с-маршрутом*, должен обозначать канал, соединенный с охватывающим блоком.

Каждый *Идентификатор-маршрута-сигнала*, указанный в *Соединении-канала-с-маршрутом*, должен быть описан в том же блоке, в котором описано *Соединение-канала-с-маршрутом*, и одной из его конечных точек должна быть граница этого блока. Каждый *Идентификатор-маршрута-сигнала*, описанный в охватывающем блоке и имеющий окружающую среду в качестве одной из своих конечных точек, должен быть указан в одном и только одном *Соединении-канала-с-маршрутом*.

Для заданного направления объединение совокупности *Идентификаторов-сигнала*, указанных в маршрутах сигнала, входящих в *Соединение-канала-с-маршрутом*, должно совпадать со множеством сигналов, переданных *Идентификатором-канала* в том же самом *Соединении-канала-с-маршрутом*, соответствующем тому же направлению.

Конкретная текстовая грамматика

<соединение канала с маршрутом> :: =
CONNECT <идентификатор канала>
AND <идентификатор маршрута сигнала> { , <идентификатор маршрута сигнала> } *
<конец>

Ни один <идентификатор маршрута сигнала> не может быть указан в <соединении канала с маршрутом> дважды.

Конкретная графическая грамматика

Графически конструкция "соединение" изображается с помощью <идентификатора канала>, ассоциированного с <маршрутом сигнала> и содержащегося в <области определения маршрута сигнала> (см. § 2.5.2, раздел *Конкретная графическая грамматика*).

2.5.4 *Сигнал*

Абстрактная грамматика

Определение-сигнала	::	*Имя-сигнала*
Имя-сигнала		*Идентификатор-ссылки-на-сорт**
		[*Уточнение-сигнала*]
Имя-сигнала	=	*Имя*

Идентификатор-ссылки-на-сорт описан в § 5.2.2.

Конкретная текстовая грамматика

<определение сигнала> :: =
SIGNAL { <имя сигнала> [<список сортов>][<уточнение сигнала>] }
{ , <имя сигнала> [<список сортов>][<уточнение сигнала>]}<конец>

<список сортов> :: =
(<сорт>{ , <сорт> } *)

<уточнение сигнала> описано в § 3.3, <сорт> — в § 5.2.2.

Семантика

Экземпляр сигнала является переносом информации между процессами; он представляет из себя реализацию типа сигнала, описанного в определении сигнала. Экземпляр сигнала может быть послан либо окружающей средой, либо процессом и всегда адресован либо среде, либо процессу.

Два значения PId (см. § 5.6.10), обозначающие исходный и приемный процессы, <идентификатор сигнала>, специфицированный в соответствующем выходе, и другие значения, чьи сорта описаны в определении сигнала, ассоциируются с каждым экземпляром сигнала.

2.5.5 *Определение списка сигналов*

<Идентификатор списка сигналов> может использоваться в <определении канала>, <определении маршрута канала>, <определении списка сигналов>, <совокупности допустимых входных сигналов> и <списке сохранений> в качестве сокращенного средства перечисления идентификаторов сигнала и сигналов от таймеров.

Конкретная текстовая грамматика

<определение списка сигналов> :: =
SIGNALLIST <имя списка сигналов> = <список сигналов> <конец>

<список сигналов> :: =
<элементарный сигнал>{ , <элементарный сигнал>} *

<элементарный сигнал> :: =
<идентификатор сигнала> | <идентификатор приоритетного сигнала> |
(<идентификатор списка сигналов>)
| <идентификатор таймера>

<список сигналов>, который получен заменой всех <идентификаторов списка сигналов> в списке на те <идентификаторы сигналов>, которые они выражают, соответствует *Идентификатору-сигнала-set* в *Абстрактной грамматике.* В любом так построенном <списке сигналов> все <идентификаторы сигналов> должны быть различны.

Конкретная графическая грамматика

<область списка сигналов> ::=
<символ списка сигналов> **contains** <список сигналов>

<символ списка сигналов> :: =

2.6 *Поведение*

2.6.1 *Переменные*

2.6.1.1 *Определение переменной*

Абстрактная грамматика

Определение-переменной	::	*Имя-переменной* *Идентификатор-ссылки-на сорт* [REVEALED]
Имя-переменной	=	*Имя*

Конкретная текстовая грамматика

<определение переменной> :: =
DCL [REVEALED | EXPORTED | REVEALED EXPORTED | EXPORTED REVEALED]
<имя переменной> { , <имя переменной> } * <сорт> [: = <начальное выражение>]
{ , <имя переменной> { , <имя переменной> } * <сорт> [: = <начальное выражение>] } * <конец>

Экспортируемая переменная описывается в § 4.13.

Семантика

Семантика переменных объяснена в § 2.3.2. Значение переменной может быть модифицировано только ее владельцем. Владельцем переменной является процесс (или процедура), в котором она объявлена. Значение переменной известно только владельцу переменной, за исключением случая, когда переменная была объявлена с атрибутом REVEALED. Этот атрибут позволяет всем другим процессам того же блока обозревать переменную, при условии, что они содержат определение обозревания при объявлении этой переменной.

Модель

<начальное выражение> в <определении переменной> или значение по умолчанию в <сорте> не имеют соответствующего абстрактного синтаксиса. Это является выводимым синтаксисом для спецификации последовательности предложений присваивания в начальном переходе охватывающей области видимости. Предложения присваивания присваивают <начальное выражение> всем <именам переменных>, указанным в <определении переменной>. Если в <определении переменной> специфицированы как <начальное выражение>, так и значение

по умолчанию в <сорте>, то в <определении переменной> применяется <начальное выражение>.

2.6.1.2 *Определение обозревания*

Абстрактная грамматика

Определение обозревания :: *Идентификатор-переменной*
Идентификатор-ссылки-на-сорт

Определение-переменной, заданное с помощью *Идентификатора-переменной*, должно иметь атрибут REVEALED, и должно быть того же сорта, что и указанный *Идентификатор-ссылки-на-сорт*.

Конкретная текстовая грамматика

<определение обозревания> :: =
VIEWED
<идентификатор переменной> { , <идентификатор переменной> } * <сорт>
{ , <идентификатор переменной> { , <идентификатор переменной> } * <сорт> } *
<конец>

Квалификатор в <идентификаторе переменной>, входящем в <определение обозревания>, может быть опущен, если в блоке существует одно и только одно <определение процесса>, имеющее <определение переменной>, определяющее <имя переменной> со следующими свойствами: это имя совпадает с <именем переменной>, указанным в <определении обозревания>; это имя снабжено атрибутом REVEALED; это имя принадлежит тому же сорту, что и <сорт>, указанный в <определении обозревания>.

Семантика

Механизм обозревания позволяет экземпляру процесса непрерывно видеть значение обозреваемой переменной так, как если бы она была определена локально. Однако обозревающий экземпляр процесса не имеет права модифицировать значение этой переменной.

2.6.2 *Старт*

Абстрактная грамматика

Стартовая-вершина-процесса :: *Переход*

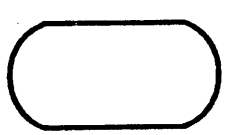
Конкретная текстовая грамматика

<старт> :: =
START <конец> <переход>

Конкретная графическая грамматика

<область старта> :: =
<символ старта> is followed by <область перехода>

<символ старта> :: =



Семантика

Интерпретируется *Переход Стартовой-вершины-процесса*

2.6.3 *Состояние*

Абстрактная грамматика

Вершина-состояния :: *Имя-состояния*
Совокупность-сохраняемых-сигналов
Входная-вершина-set

Имя-состояния = *Имя*

Вершины-состояний графа-процесса и соответственно *графа-процедуры* должны иметь различные *Имена-состояний.*

Для любой *Вершины-состояния* все *Идентификаторы-сигналов* (из совокупности допустимых входных сигналов) входят либо в *Совокупность-сохраняемых-сигналов*, либо во *Входную-вершину.*

Идентификаторы-сигналов, входящие во *Входную-вершину*-set должны быть различными.

Конкретная текстовая грамматика

<состояние> ::=
STATE <список состояний> <конец>
{ <раздел ввода>
| <приоритетный ввод>
| <раздел сохранения>
| <непрерывный сигнал>}*
[ENDSTATE[<имя состояния>] <конец>]

<список состояний> ::=
{ <имя состояния>{ , <имя состояния>}* }
| { <список состояний "звездочка"> }

<раздел ввода> описывается в § 2.6.4, <раздел сохранения> — в § 2.6.5, <непрерывный сигнал>—в § 4.1.1, <список состояний "звездочка"> — в § 4.4 и, наконец, <приоритетный ввод> — в § 4.10.2.

Когда <список состояний> содержит одно <имя состояния>, тогда это <имя состояния> изображает *Вершину-состояния.* Для каждой *Вершины-состояния Совокупность-сигналов-сохранения* представлена <разделом сохранения> и всеми неявными сохранениями сигналов. Для каждой *Вершины-состояния Входная-вершина*-set представлена <разделом ввода> и всеми неявными входными сигналами.

Необязательное <имя состояния>, завершающее <состояние>, может быть специфицировано только, если <список состояний> в <состоянии> содержит одно-единственное <имя состояния>; в этом случае оно должно совпадать со стоящим в <списке состояний>.

Конкретная графическая грамматика

<область состояния> ::=
<символ состояния> contains <список состояний> is associated with
{ <область ассоциирования ввода>
| <область ассоциирования приоритетного ввода>
| <область ассоциирования непрерывного сигнала>
| <область ассоциирования сохранения>}*

<символ состояния> ::=

<область ассоциирования ввода> ::=
<символ сплошной линии ассоциации> is connected to <область ввода>

<область ассоциирования сохранения> ::=
<символ сплошной линии ассоциации> is connected to <область сохранения>

<область ввода> описана в § 2.6.4, <область сохранения> — в § 2.6.5, <область ассоциирования непрерывного сигнала> — в § 4.11, <область ассоциирования приоритетного ввода> — в § 4.10.2.

<область состояния> изображает одну или несколько *Вершин-состояния*.

<символы сплошной линии ассоциации>, начинающиеся из <символа состояния>,могут иметь общий исходный участок пути.

Семантика

Состояние изображает специальное условие, при котором экземпляр процесса может воспринять экземпляр сигнала, в результате чего возникает переход. Если нет задержанных экземпляров сигнала, то процесс остается в этом состоянии в ожидании поступления экземпляра сигнала.

Модель

Если <список состояний> некоторого <состояния> содержит более одного <имени состояний>, то для каждого такого <имени состояния> создается копия <состояния>, после чего <состояние> заменяется этими копиями.

2.6.4 *Ввод*

Абстрактная грамматика

Входная-вершина :: *Идентификатор-сигнала*
[*Идентификатор-переменной*]*
Переход

Идентификатор-переменной = *Идентификатор*

Длина [*Идентификатора-переменной*]* должна совпадать с числом *Идентификаторов-ссылки-на-сорт*, входящих в *Определение-сигнала*, представленного *Идентификатором-сигнала*.

Сорта переменных должны соответствовать с точки зрения порядка расположения сортам значений, которые могут быть перенесены сигналом.
Не допускается специфицировать больше воспринимающих переменных, чем число значений, переносимых экземпляром сигнала.

Конкретная текстовая грамматика

<раздел ввода> ::=
INPUT <входной список> <конец>
[<разрешающее условие>] <переход>

<входной список> ::=
<входной список "звездочка">
| <стимул> { , <стимул> }*

<стимул> ::=
{ <идентификатор сигнала>
| <идентификатор таймера> }[([<идентификатор переменной>] { , [<идентификатор переменной>] }*)]

<переход> описан в § 2.6.7, <разрешающее условие> — в § 4.12 и, наконец, <входной список "звездочка"> — в § 4.6

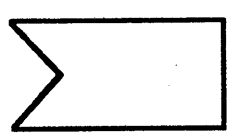
Если <входной список> содержит всего один <стимул>, то <раздел ввода> изображает <входную вершину>. В *Абстрактной грамматике* сигналы от таймера (<идентификатор таймера>) также представлены *Идентификатором-сигнала.* Так как с многих точек зрения сигналы от таймера и обычные сигналы имеют одинаковые свойства, то различают их только тогда, когда это требуется по существу вопроса. Точные свойства сигналов от таймера определены в § 2.8.

<переход> должен иметь терминатор перехода, как это описано в § 2.6.7.2.

Конкретная графическая грамматика

<область ввода> ::=
<символ ввода> **contains** <список ввода>
is followed by [<область разрешающего условия>] <область перехода>

<символ ввода> ::=



<область перехода> описана в § 2.6.7, а <область разрешающего условия> — в § 4.12.

<область ввода>, чей <список ввода> содержит только один <стимул>, соответствует одной *Входной-вершине.* Каждый <идентификатор сигнала>, содержащийся в <символе ввода>, придает имя одной из *Входных-вершин,* которые изображают этот <символ ввода>.

Семантика

Ввод обеспечивает восприятие специфицированного экземпляра входного сигнала. Восприятие экземпляра входного сигнала делает информацию, переданную с сигналом, доступной процессу. Переменным, связанным со вводом, присваиваются значения, переданные с сигналом. Если ввод не имеет связанных с ним переменных, соответствующих сорту, специфицированному в сигнале, то значение этого сорта аннулируется.

Выражение SENDER того процесса, который воспринял сигнал, получает PId-значение экземпляра процесса, пославшего сигнал; это значение передается вместе с сигналом.

Экземпляры сигналов, присланные из окружающей среды и поступившие в процесс, принадлежащий системе, всегда приносят PId-значения, отличные от любого значения в системе. Присланное значение становится доступным с помощью выражения SENDER.

Модель

Если список <стимулов> некоторого <раздела ввода> содержит более одного <стимула>, то для каждого <стимула>, входящего в список, создается отдельная копия <раздела ввода>, после чего <раздел ввода> заменяется на эти копии.

2.6.5 *Сохранение*

Сохранение специфицирует совокупность идентификаторов сигналов, которые не могут быть восприняты процессом в том его состоянии, которому сохранение придано, и которые должны быть сохранены для обработки в дальнейшем.

Абстрактная грамматика

Совокупность-сохраняемых-сигналов :: *Идентификатор-сигнала*-**set**

Для каждой *Вершины-состояния* все *Идентификаторы-сигналов*, содержащиеся в *Совокупности-сохраняемых-сигналов*, должны быть различны.

Конкретная текстовая грамматика

<раздел сохранения> ::=
SAVE <список сохранений> <конец>
<список сохранений> ::=
{<список сигналов> | <список сохранений "звездочка">}

<список сохранений> представляет *Идентификатор-сигнала*-**set**. <список сохранений "звездочка"> является сокращенным обозначением, разъясняемым в § 4.8.

Конкретная графическая грамматика

<область сохранения> ::=
<символ сохранения> **contains** <список сохранений>

<символ сохранения> ::=

Семантика

Сохраненные сигналы задерживаются в порту в том порядке, в котором они поступили.

Эффект сохранения действителен только для того состояния, которому он придан. В следующем состоянии экземпляры сигналов, которые в предыдущем были "сохранены", рассматриваются как обычные экземпляры сигналов.

2.6.6 *Метка*

Конкретная текстовая грамматика

<метка> ::=
<имя коннектора>

Все <имена коннекторов>, определенные в <теле процесса>, должны быть различными.

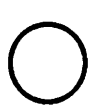
Метка обозначает входную точку "перескока" от соответствующих предложений перескока с тем же <именем коннектора> и в том же <теле процесса>.

"Перескоки" допустимы только к меткам в том же <теле процесса>.

Конкретная графическая грамматика

<область входного коннектора> ::=
<символ входного коннектора> **contains** <имя коннектора> **is followed by**
<область перехода>

<символ входного коннектора> ::=



<область перехода> описана в § 2.6.7.1.

<область входного коннектора> изображает продолжение <символа линии потока> от соответствующей <области выходного коннектора> с тем же самым <именем коннектора> в той же самой <области графа процесса> или <области графа процедуры>.

2.6.7 *Переход*

2.6.7.1 *Тело перехода*

Абстрактная грамматика

Переход	::	*Вершина-графа**
		(*Терминатор* | *Вершина-принятия-решения*)
Вершина-графа	::	*Вершина работы* |
		Выходная-вершина |
		Вершина-запроса-на-создание |
		Вершина-вызова |
		Вершина-установки |
		Вершина-сброса |
Терминатор	::	*Вершина-следующего-состояния* |
		Стоп-вершина|
		Вершина-возврата

Конкретная текстовая грамматика

```
<переход> ::=
          { <цепочка перехода> [<предложение терминатора>] }
    |     <предложение терминатора>

<цепочка перехода> ::=
                {<предложение действия>}+

<предложение действия> ::=
                  [<метка>] <действие> <конец>

<действие> ::=
          <работа>
    |     <вывод>
    |     <приоритетный вывод>
    |     <запрос на создание>
    |     <принятие решения>
    |     <необязательный переход>
    |     <установка>
    |     <сброс>
    |     <экспорт>
    |     <вызов процедуры>

<предложение терминатора> ::=
                        [<метка>] <терминатор> <конец>

<терминатор> ::=
                  <следующее состояние>
          |       <перескок>
```

| <стоп>
| <возврат>

<работа> описана в § 2.7.1, <вывод> – в § 2.7.4, <запрос на создание> – в § 2.7.2, <принятие решения> – в § 2.7.5, <установка> и <сброс> – в § 2.8, <вызов процедуры> – в § 2.7.3, <следующее состояние> – в § 2.6.7.2.1, <перескок> – в § 2.6.7.2.2, <стоп> – в § 2.6.7.2.3, <возврат> – в § 2.6.7.2.4, <приоритетный вывод> – в § 4.10.2, <необязательный переход> – в § 4.3.4, <экспорт> – в § 4.13.

Если <терминатор> <перехода> опущен, то последним действием в <переходе> должно быть либо завершающее <принятие решения> (см. § 2.7.5), либо завершающий <необязательный переход>; это не касается <переходов>, которые содержатся в <принятиях решений> и в <необязательных переходах> (<необязательные переходы> описываются в § 4.3.4).

Никакой <терминатор> или <действие> не могут следовать за <терминатором>, завершающим <необязательным переходом> или завершающим <принятием решения>.

Конкретная графическая грамматика

<область перехода> ::=
[<область цепочки перехода>] is followed by
{ <область состояния>
| <область следующего состояния>
| <область принятия решения>
| <символ стоп>
| <область слияния>
| <область выходного коннектора>
| <символ возврата>
| <область необязательного перехода>}

<область цепочки перехода> ::=
{<область работы>
| <область вывода>
| <область приоритетного вывода>
| <область установки>
| <область сброса>
| <область экспорта>
| <область запроса на создание>
| <область вызова продедуры>}
[is followed by <область цепочки перехода>]

<область работы> описана в § 2.7.1, <область вывода> – в § 2.7.4, <область запроса на создание> – в § 2.7.2, <область принятия решения> – в § 2.7.5, <область установки> и <область сброса> – в § 2.8, <область вызова процедуры> – в § 2.7.3, <область следующего состояния> – в § 2.6.7.2.1, <область слияния> – в § 2.6.7.2.2, <символ стоп> – в § 2.6.7.2.3, <символ возврата> – в § 2.6.7.2.4, <область приоритетного вывода> – в § 4.10.2, <область необязательного перехода> – в § 4.3.4, <область экспорта> – в § 4.13, <область выходного коннектора> – в § 2.6.7.2.2.

Переход состоит из последовательности действий, которые должен выполнить процесс.

<область перехода> соответствует *Переходу*, а <область цепочки перехода> соответствует *Вершине-графа.**

Семантика

Переход совершает некоторая последовательность действий. В течение перехода данные некоторого процесса могут подвергаться обработке и могут выдаваться сигналы. Переход завершается тем, что процесс входит либо в некоторое состояние, либо в "стоп", либо в "возврат".

2.6.7.2 *Терминатор перехода*

2.6.7.2.1 *Следующее состояние*

Абстрактная грамматика

Вершина-следующего-состояния :: *Имя-состояния*

Имя-состояния, специфицированное в следующем состоянии, должно быть именем состояния, содержащемся в том же *Графе-процесса* или *Графе-процедуры*.

Конкретная текстовая грамматика

<следующее состояние> ::=
NEXTSTATE <тело следующего состояния>

<тело следующего состояния> ::=
{<имя состояния> | <следующее состояние 'черточка'>}

<следующее состояние черточка> описано в § 4.9.

Конкретная графическая грамматика

<область следующего состояния> ::=
<символ состояния> **contains** <тело следующего состояния>

Семантика

<следующее состояние> представляет терминатор перехода. Оно специфицирует то состояние, в которое войдет экземпляр процесса по окончании перехода.

2.6.7.2.2 *Перескок*

Перескок изменяет продвижение по <диаграмме процесса> или по <телу процесса>, указывая, что следующим, подлежащим интерпретации <предложением действия>, является действие, содержащее то же самое <имя коннектора>.

Конкретная текстовая грамматика

<перескок> ::=
JOIN <имя коннектора>

В том же самом <теле процесса>, <теле процедуры>, соответственно <теле сервиса>, должно быть одно и только одно <имя коннектора>, соответствующее имени, указанному в <перескоке>.

Конкретная графическая грамматика

<область слияния> ::=
<символ слияния> **is connected to** <символ линии потока>

<символ слияния> ::=
<символ линии потока>

<символ линии потока> ::=

<область выходного коннектора> ::=
<символ выходного коннектора> **contains** <имя коннектора>

<символ выходного коннектора> ::=
<символ входного коннектора>

Каждой <области выходного коннектора> в <области графа процесса> или в <области графа процедуры> должна соответствовать в той же области одна и только одна <область входного коннектора> с тем же <именем коннектора>.

<область выходного коннектора> соответствует <перескоку> в *Конкретной текстовой грамматике.* Если в <область перехода> включена <область слияния>, то это эквивалентно спецификации в <области перехода> <области выходного коннектора>, содержащей уникальное <имя коннектора> и соединении <области входного коннектора> с тем же самым <именем коннектора> и <символа линии потока> в <области слияния>.

Модель

В абстрактном синтаксисе <сцепление> или <область выходного коннектора> выводимы из <цепочки перехода>, в которой к первому <предложению действия> или области подключено то же самое <имя коннектора>.

2.6.7.2.3 *Стоп*

Абстрактная грамматика

Вершина-стоп :: ()

Граф-процедуры не должен содержать *Вершину-стоп.*

Конкретная текстовая грамматика

<стоп> ::=
STOP

Конкретная графическая грамматика

<символ стоп> ::=

Семантика

"Стоп" вызывает немедленный останов процесса, выдавшего "стоп". Это означает, что все задержанные во входном порту сигналы аннулируются и прекращают свое существование переменные и таймеры, созданные для этого процесса, а также его порт и он сам.

2.6.7.2.4 *Возврат*

Абстрактная грамматика

Вершина-возврата :: ()

Граф-процесса не должен содержать *Вершины-возврата.*

Конкретная текстовая грамматика

<возврат> ::=
RETURN

Конкретная графическая грамматика

<символ возврата> ::=

Семантика

Вершина-возврата интерпретируется следующим образом:

a) Все переменные, созданные в результате интерпретации *Стартовой-вершины-процедуры,* прекращают свое существование.

b) Интерпретация *Вершины-возврата* прекращает интерпретацию *Графа-процедуры,* и экземпляр процедуры прекращает свое существование.

c) После этого возобновляется интерпретация вызывавшего процесса (или процедуры) и при том с вершины, непосредственно следующей за вершиной вызова.

2.7 *Действие*

2.7.1 *Работа*

Абстрактная грамматика

Вершина-работы :: *Предложение-присваивания* |
Неформальный текст

Конкретная текстовая грамматика

<работа> ::=
TASK <тело работы>

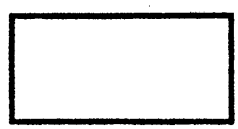
<тело работы> ::=
{<предложение присваивания> { <предложение присваивания>}*}
| {<неформальный текст> {, <неформальный текст>}*}

<предложение присваивания> описано в § 5.5.3.

Конкретная графическая грамматика

<область работы> ::=
<символ работы> **contains** <тело работы>

<символ работы> ::=



Семантика

Интерпретацией *Вершины-работы* является интерпретация *Предложения-присваивания*, которая описана в § 5.5.3, или интерпретация *Неформального-текста*, которая описана в § 2.2.3.

Модель

<работа> и <область работы> могут содержать несколько <предложений присваивания> или <неформальных текстов>. В этом случае они представлены производным синтаксисом для спецификации последовательности <работ>, по одной на каждое <предложение присваивания> или каждый <неформальный текст>; при этом должен быть сохранен первоначальный порядок, в котором они были специфицированы в <теле работы>.

Эта сокращенная запись должна быть расширена до расширения любого <импортирующего выражения> (см. § 4.13).

2.7.2 *Создание*

Абстрактная грамматика

Вершина-запроса-на-создание :: *Идентификатор-процесса*
[*Выражение*]*

Идентификатор-процесса = *Идентификатор*

Число *Выражений* в конструкции [*Выражение*]* должно совпадать с числом *Формальных-параметров-процесса* в том *Определении-процесса*, которое указано *Идентификатором-процесса*. Сорт каждого *Выражения* должен совпадать с сортом соответствующего в смысле порядка *Формального-параметра-процесса* в *Определении-процесса*, которое указано *Идентификатором-процесса*.

Конкретная текстовая грамматика

<запрос на создание> ::=
CREATE <тело создания>

<тело создания> ::=
<идентификатор процесса> [<фактические параметры>]

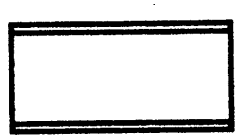
<фактические параметры> ::=
([<выражение>] {, [<выражение>]}*)

<выражение> описано в § 5.

Конкретная графическая грамматика

<область запроса на создание> ::=
<символ запроса на создание> contains <тело создания>

<символ запроса на создание> ::=



<область запроса на создание> соответствует *Вершине-запроса-на-создание*.

Семантика

При создании экземпляра процесса, ему придается пустой входной порт и создаются переменные. Затем в заданном порядке интерпретируются выражения для фактических параметров и они присваиваются (как это описано

в § 5.5.3) соответствующим формальным параметрам. Если фактический параметр пуст, то соответствующему формальному параметру присваивается неопределенное значение. После этого процесс начинается с интерпретации стартовой вершины графа процесса.

После этого созданный процесс выполняется асинхронно от остальных процессов и параллельно с ними.

Действие по созданию процесса создает экземпляр процесса только в том же самом блоке. Выражение PARENT созданного процесса имеет то же значение PId, что и выражение SELF создающего процесса. Выражение SELF созданного процесса и выражение OFFSPRING создающего процесса имеют одно и то же вновь созданное значение PId (см. § 5.6.10.1).

При попытке создать большее число экземпляров процесса, чем специфицированное в определении процесса максимальное число процессов, новый экземпляр процесса не возникает, выражение OFFSPRING создающего процесса получают значение NULL и продолжается интерпретация этого процесса.

2.7.3 *Вызов процедуры*

Абстрактная грамматика

Вершина-вызова :: *Идентификатор-процедуры*
[*Выражение*]*

Идентификатор-процедуры = *Идентификатор*

Длина конструкции [*Выражение*]* должна совпадать с числом *Формальных-параметров-процедуры* в том *Определении-процедуры*, которое указано *Идентификатором-процедуры*.

Каждое *Выражение*, соответствующее в смысле порядка *Формальному-параметру-процесса* IN, должно иметь тот же сорт, что и *Формальный-параметр-процесса*.

Каждое *Выражение*, соответствующее в смысле порядка *Формальному-параметру-процесса* IN/OUT должно быть *Идентификатором-переменной*, имеющим тот же *Идентификатор-ссылки-на-сорт*, что и *Формальный-параметр-процесса*.

Каждому *Формальному-параметру-процесса* IN/OUT должно соответствовать некоторое *Выражение*.

Конкретная текстовая грамматика

<вызов процедуры> ::=
CALL <тело вызова процедуры>

<тело вызова процедуры> ::=
<идентификатор процедуры> [<фактические параметры>]

<фактические параметры> описаны в § 2.7.2.

Пример <вызова процедуры> приведен на рис. 2.9.13 в § 2.9.

Конкретная графическая грамматика

<область вызова процедуры> ::=
<символ вызова процедуры> **contains** <тело вызова процедуры>

<символ вызова процедуры> ::=

<область вызова процедуры> представляет *Вершину-вызова.*

Пример <области вызова процедуры> приведен на рис. 2.9.14 в § 2.9.

Семантика

Интерпретация вершины вызова процедуры передает интерпретацию тому определению процедуры, которое указано в вершине вызова, и затем интерпретируется граф этой процедуры. Вершины графа процедуры интерпретируются точно так же, как и эквивалентные вершины графа процесса.

Интерпретация вызывающего процесса приостанавливается впредь до завершения интерпретации вызванной процедуры.

Выражения фактических параметров интерпретируются в заданном порядке.

Интерпретация данных и параметров требует специальной семантики (см. объяснения в § 2.4.4).

2.7.4 *Вывод*

Абстрактная грамматика

Выходная-вершина	::	*Идентификатор-сигнала* [*Выражение*]* [*Приемник-сигнала*] *Направить-через*
Приемник-сигнала	=	*Выражение*
Направить-через	=	*Идентификатор-маршрута-сигнала-***set**

Длина конструкции [*Выражение*]* должна совпадать с числом *Идентификаторов-ссылки-на-сорт* в *Определении-сигнала,* которое указано *Идентификатором-сигнала.*

Сорт каждого *Выражения* должен совпадать с соответствующей (в смысле порядка) *Ссылкой-на-идентификатор-сорта* в *Определении-сигнала.*

Для любого возможного согласованного подмножества (см. § 3) должен существовать хотя бы один путь связи (либо неявный для собственного типа процесса, либо явный по маршрутам сигналов и, возможно, каналам), ведущий либо к окружающей среде, либо к процессу того типа, совокупность допустимых входных сигналов которого содержит *Идентификатор-сигнала,* и начинающийся от процесса того типа, в котором используется *Выходная-вершина.*

Для каждого *Идентификатора-маршрута-сигнала* в конструкции *Направить-через* должно выполняться требование: *Исходный-процесс* (если таковой имеется) в *Путях-маршрута-сигнала* в маршруте сигнала должен принадлежать к тому же типу процессов, что и процесс, содержащий *Выходную-вершину*; кроме того, *Путь-маршрута-сигнала* должен содержать *Идентификатор-сигнала* в своей совокупности *Идентификаторов-сигналов.*

Если конструкция *Направить-через* не содержит *Идентификатора-маршрута-сигнала,* то любой процесс, для которого существует путь связи, может получить сигнал.

Конкретная текстовая грамматика

<вывод> ::=
OUTPUT <тело вывода>

<тело вывода> ::=
<идентификатор сигнала>
[<фактические параметры>] {, <идентификатор сигнала> [<фактические параметры>]}*
[TO <выражение PId>]
[VIA {<идентификатор маршрута сигнала>}, <идентификатор маршрута сигнала>}*
| {<идентификатор канала>{, <идентификатор канала>}*}]

<фактические параметры> описаны в § 2.7.2, <выражения> — в § 5.4.2.1.

Если в блоке специфицирован хоть какой-нибудь маршрут сигнала, то в конструкции VIA нельзя специфицировать <идентификатор канала>.

Для каждого <идентификатора канала>, выходящего в <вывод>, должен существовать канал, начинающийся от охватывающего блока и способный переносить сигналы, указанные в <идентификаторах сигналов>, содержащихся в <выводе>.

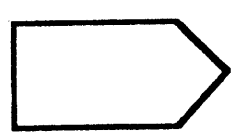
Конструкция TO <выражение PId> представляет *Приемник-сигнала.*

Конструкция VIA представляет *Направить-через.*

Конкретная графическая грамматика

<область вывода> ::=
<символ вывода> contains <тело вывода>

<символ вывода> ::=



Семантика

Выражение PId *Приемника-сигнала* интерпретируется после всех других выражений в *Выходной-вершине.*

Значения, передаваемые экземпляром сигнала, являются значениями фактических параметров в выводе. Если для сорта, указанного в определении сигнала, вывод не содержит фактического параметра, то сигнал передает неопределенное значение.

Значение исходного PId, передаваемого экземпляром сигнала, совпадает со значением PId, связанным с выраже-

нием SELF (процесса, выполняющего выходное действие). Значение PId приемника, передаваемого экземпляром сигнала, совпадает со значением выражения PId приемника сигнала, содержащимся в выводе.

После этого экземпляр сигнала подается на путь связи, способный передать его специфицированному экземпляру приемного процесса.

Если никакой *Приемник-сигнала* не специфицирован, то должен существовать один и только один получатель, до которого сигнал может дойти по маршрутам сигнала и каналам, указанным в конструкции *Направить-через*. Значение PId приемника, неявно переданное экземпляром сигнала, является значением PId этого получателя.

Окружающая среда всегда может получить любой сигнал из совокупности сигналов такого канала, который ведет к окружающей среде.

Заметьте, что спецификация одного и то же *идентификатора канала* или *идентификатора маршрута-сигнала* в конструкции *Направить-через* двух *Выходных-вершин* не означает, что сигналы будут автоматически установлены в очередь во входном порту в том же порядке, в котором интерпретировались *Выходные-вершины*. Однако порядок сохраняется, если два сигнала передаются по идентичным каналам, соединяющим *Исходный-процесс* с *Приемным-процессом* или если процессы определены в одном и том же блоке.

Если в определении сигнала специфицирован подтип, а в выводе специфицировано выражение, то в отношении выражения выполняется проверка диапазона, описанная в § 5.4.1.9.1. Если результат проверки диапазона дает значение False, то в выводе имеется ошибка и дальнейшее поведение системы неопределенно.

Вывод, посланный не существующему (или не существующему более) экземпляру процесса, приводит к ошибке в интерпретации. Проверка существования процесса делается одновременно с интерпретацией вывода. Последующее прекращение функционирования экземпляра приемного процесса приводит к аннулированию сигнала из входного порта, но это не считается ошибочной ситуацией.

Модель

Если в <теле вывода> специфицировано несколько пар (<идентификатор сигнала> <фактические параметры>), то это является производным синтаксисом для спецификации последовательности <выводов> или <областей вывода> и при том в том же порядке, в котором они специфицированы в исходном <теле вывода>; каждый из элементов этой последовательности содержит одну пару (<идентификатор сигнала> <фактические параметры>). Клаузулы TO и VIA повторяются в каждом из <выводов> или <областей вывода>. Расширение этого сокращения осуществляется до того, как осуществляется любое расширение в имеющихся выражениях.

2.7.5 *Принятие решения*

Абстрактная грамматика

Вершина-принятия-решения	::	*Вопрос-принятия-решения* *Ответ-принятия-решения*-set [*Иначе-ответ*]
Вопрос-принятия-решения	=	*Выражение* | *Неформальный текст*
Ответ-принятия-решения	::	*(Условие на-диапазон* | *Неформальный текст) Переход*
Иначе-ответ	::	*Переход*

Ответы-принятия-решения должны быть попарно взаимоисключающими.

Если *Вопрос-принятия-решения* является *Выражением*, то *условие-на-диапазон Ответа-принятия-решения* должно быть того же сорта, что и *Вопрос-принятия-решения*.

Конкретная текстовая грамматика

<принятие решения> ::=
DECISION <вопрос> <конец> <тело принятия решения> ENDDECISION

<тело принятия решения> ::=
{<раздел ответа> <раздел иначе>}
| {<раздел ответа> {<раздел ответа>}$^{+}$ [<раздел иначе>]}

<раздел ответа> ::=
(<ответ>) : [<переход>]

<ответ> ::=
<условие на диапазон> | <неформальный текст>

<раздел иначе> ::=
ELSE: [<переход>]

<вопрос> ::=
<выражение вопрос> | <неформальный текст>

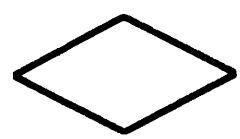
<условие на диапазон> описано в § 5.4.1.9.1, <переход> — в § 2.6.7.1, <неформальный текст> — в § 2.2.3.

<принятие решения> или <необязательный переход> (описанный в § 4.3.3) прекращается, если каждый <раздел ответа> и <раздел иначе> в <теле принятия решения> содержит <переход>, в котором либо специфицировано <предложение терминатора>, либо если <переход> содержит <цепочку перехода>, последнее <предложение действия> которой содержит завершающее принятие решения или необязательное действие.

Конкретная графическая грамматика

<область принятия решения> ::=
<символ принятия решения> **contains** <вопрос>
is followed by
{{<раздел графического ответа> <раздел графического иначе>}**set**
| {<раздел графического ответа> {<раздел графического ответа>}$^{+}$ [<раздел графического иначе>]}**set**}

<символ принятия решения> ::=



<раздел графического ответа> ::=
<символ линии потока> **is associated with** <графический ответ>
is followed by <область перехода>

<графический ответ> ::=
<ответ> | (<ответ>)

<раздел графического иначе> ::=
<символ линии потока> **is associated with ELSE**
is followed by <область перехода>

<область перехода> описана в § 2.6.7.1, <символ линии потока> — в § 2.6.7.2.2.

<графический ответ> и ELSE могут быть помещены вдоль соответствующего <символа линии потока> или в разрыве этого <символа линии потока>.

<символы линии потока>, исходящие из <символа принятия решения> могут иметь общие отрезки.

<область принятия решения> представляет *Вершину-принятия-решения.*

Семантика

Принятие решения передает интерпретацию тому из выходящих путей, чей диапазон условия содержит значение, полученное в результате интерпретации вопроса. Каждый из совокупности возможных ответов на вопрос специфицирует действия, подлежащие интерпретации на отобранном пути.

Один из ответов может быть дополнительным по отношению ко всем остальным. Это достигается спецификацией *Иначе-ответа,* который описывает последовательность действий, подлежащих выполнению в случае, когда значение выражения, с помощью которого формируется вопрос, не покрывается значениями или совокупностями значений в других ответах.

Если ответ "иначе" не специфицирован, то значение, полученное в результате вычисления выражения, стоящего в вопросе, должно подходить одному из ответов.

Между <неформальным текстом> и <знаковой строкой>, входящим в <вопрос> и <ответ>, возникает синтаксическая несогласованность. Если <вопрос> и все <ответы> являются <знаковыми строками>, то все они интерпретируются, как <неформальный текст>. Если <вопрос> или какой-либо из <ответов> является <знаковой строкой>, которая не подходит к контексту принятия решения, то <знаковая строка> обозначает <неформальный текст>. Контекст принятия решения (то есть сорт) определяется независимо от <ответов>, которые являются <знаковыми строками>.

Модель

Если <принятие решения> не является завершающим принятием решения, то это является производным синтаксисом для <принятия решения>, в котором все <разделы ответа> и <раздел иначе> включили в свои <переходы> <перескоки>, либо ведущие к первым <предложениям действия>, следующим за принятием решения, либо, если <принятие решения> является последним <предложением действия> в <цепочке перехода>, ведущие к следующему за ними <предложению терминатора>.

2.8 *Таймер*

Абстрактная грамматика

Определение-таймера :: *Имя-таймера Идентификатор-ссылки-на-сорт**

Имя-таймера	=	*Имя*
Вершина-установки	::	*Выражение-время* *Идентификатор-таймера* *Выражение**
Вершина-сброса	::	*Идентификатор-таймера* *Выражение**
Идентификатор-таймера	=	*Идентификатор*
Выражение-время	=	*Выражение*

Сорта *Выражения** в *Вершине-установки и в Вершине-сброса* должны соответствовать (в смысле порядка расположения) *Идентификатору-ссылки-на-сорт**, непосредственно следующему за *Именем-таймера,* заданным *Идентификатором-таймера.*

Выражения в *Вершине-сброса* или *Вершине-установки* должны вычисляться в заданном порядке.

Конкретная текстовая грамматика

<определение таймера> ::=

TIMER <имя таймера> [<список сортов>]
{ , <имя таймера> [<список сортов>]}* <конец>

<сброс> ::=

RESET (<предложение сброса>{ , <предложение сброса>}*)

<предложение сброса> ::=

<идентификатор таймера> [(<список выражений>)]

<установка> ::=

SET <предложение установки>{, <предложение установки>}*

<предложение установки> ::=

(<выражение-время>, <идентификатор таймера> [(<список выражений>)])

<список сортов> описан в § 2.5.4, <список выражений> — в § 5.5.2.1.

<предложение сброса> представляет *Вершину-сброса,* а <предложение установки> – *Вершину-установки.* Если <сброс> содержит несколько <предложений сброса>, то они должны интерпретироваться в заданном порядке. Если <установка> содержит несколько <предложений установки>, то они должны выполняться в заданном порядке.

Конкретная графическая грамматика

<область установки> ::=

<символ работы> **contains** <установка>

<область сброса> ::=

<символ работы> **contains** <сброс>

Семантика

Экземпляр таймера является собственным объектом процесса и может быть и активным и бездействующим. Два

вхождения одного идентификатора таймера, за которым следует список выражений, указывает на один и тот же экземпляр таймера, только если оба списка выражений имеют одинаковые значения.

При установке бездействующего таймера с ним ассоциируется значение времени. В отсутствии сбросов или других установок этого таймера до того момента, когда системное время сравняется с установленным в таймере значением, сигнал с тем же именем, что и имя таймера, подается во входной порт процесса. То же самое действие выполняется, если в таймере установлено значение времени, меньшее чем NOW. После восприятия сигнала от таймера выражение SENDER совпадает со значением выражения SELF. Если при установке таймера задается список выражений, то значения этих выражений содержатся в сигнале от таймера и при том в том же порядке. Таймер является активным от момента установки до момента восприятия сигнала от таймера.

Если сорт, специфицированный в определении таймера, является подтипом, то проверка диапазона, описанная в § 5.4.19.1, примененная к соответствующему выражению в установке или сбросе, должна дать значение True. В противном случае наступает ошибочное состояние и дальнейшее поведение системы неопределенно.

При сбросе бездействующего таймера он продолжает оставаться бездействующим.

При сбросе активного таймера связь со значением времени утрачивается. Если во входном порту имеется соответствующий задержанный сигнал от таймера, то он удаляется, а таймер становится бездействующим.

Установка активного таймера эквивалентна сбросу таймера, за которым немедленно следует установка таймера. Между этими сбросом и установкой таймер остается активным.

Перед первой установкой экземпляра таймера он является бездействующим.

2.9 *Примеры*

```
--------
INPUT S1 /*пример*/;
TASK/*пример*/T1:=0;
--------
```

РИСУНОК 2.9.1

Пример комментария (PR)

```
---------
INPUT I1 COMMENT 'пример';
TASK T1:=0;
---------
```

РИСУНОК 2.9.2

Пример комментария (PR)

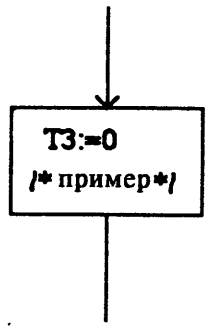


РИСУНОК 2.9.3

Пример комментария (GR)

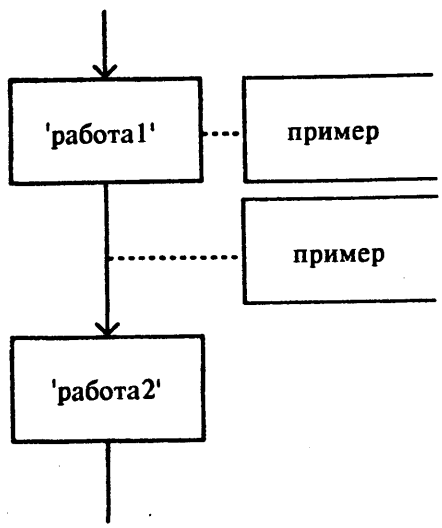


РИСУНОК 2.9.4

Пример комментария (GR)

```
SYSTEM DAEMON_GAME;
/* Эта система является игрой ..... Игрок покидает игру, выдавая сигнал Endgame */
SIGNAL Newgame, Probe, Result, Endgame, Gameid, Win, Lose, Score (Integer),
Subscr,Endsubscr, Bump;

CHANNEL C1
        FROM ENV TO Blockgame
        WITH Newgame, Probe, Result, Endgame;
        FROM Blockgame TO ENV
        WITH Gameid, Win, Lose, Score;
ENDCHANNEL C1;

CHANNEL C3 FROM Blockgame TO Blockdaemon
        WITH Subscr, Endsubscr;
ENDCHANNEL C3;

CHANNEL C4 FROM Blockdaemon TO Blockgame
        WITH Bump;
ENDCHANNEL C4;

BLOCK Blockgame REFERENCED;

BLOCK Blockdaemon REFERENCED;

ENDSYSTEM DAEMON_GAME;
```

РИСУНОК 2.9.5

Пример спецификации системы (PR)

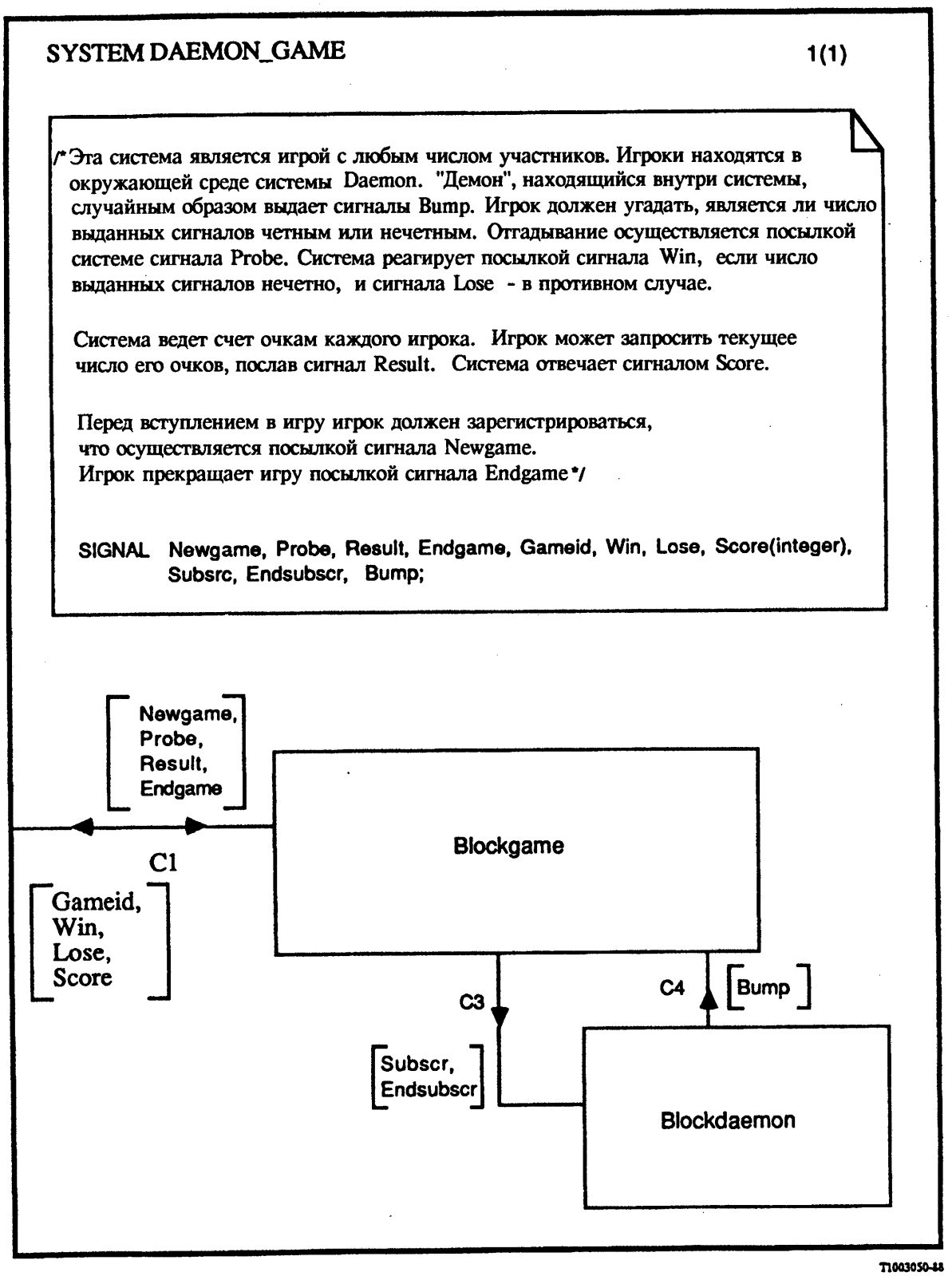


РИСУНОК 2.9.6

Пример спецификации системы (GR)

```
BLOCK Blockgame;

CONNECT C1 AND R1,R2,R3;
CONNECT C3 AND R4;
CONNECT C4 AND R5;
SIGNALROUTE R1 FROM ENV TO Monitor WITH Newgame;
SIGNALROUTE R2 FROM ENV TO Game
        WITH Probe, Result, Endgame;
SIGNALROUTE R3 FROM Game TO ENV
        WITH Gameid, Win, Lose, Score;
SIGNALROUTE R4 FROM Game TO ENV
        WITH Subscr, Endsubscr;
SIGNALROUTE R5 FROM ENV TO Game WITH Bump;

PROCESS Monitor (1,1) REFERENCED;

PROCESS Game (0,) REFERENCED;

ENDBLOCK Blockgame;
```

РИСУНОК 2.9.7

Пример спецификации блока (PR)

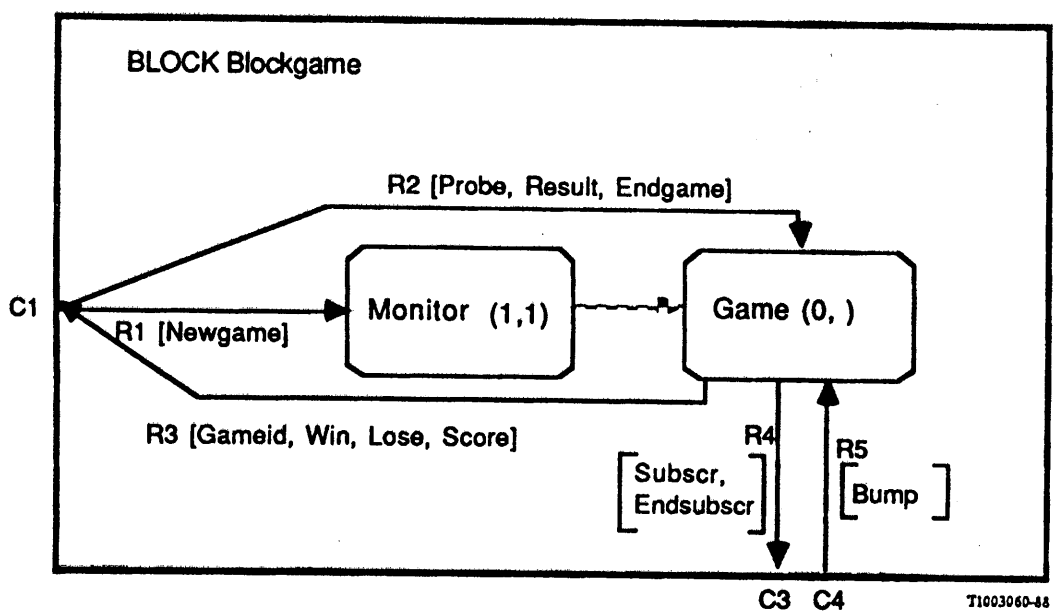


РИСУНОК 2.9.8

Пример диаграммы блока

```
PROCESS Game (0, );
FPAR Player Pid;

DCL
        Count Integer; /*счетчик для подсчета очков*/

START;

   OUTPUT Subscr;
   OUTPUT Gameid TO Player;
   TASK Count:=0;
NEXTSTATE Even;

STATE Even;

 INPUT Probe;
   OUTPUT Lose TO Player;
   TASK Count:=Count-1;
NEXTSTATE -;

 INPUT Bump;
NEXTSTATE Odd;

STATE Odd;

 INPUT Bump;
NEXTSTATE Even;

 INPUT Probe;
   OUTPUT Win TO Player;
   TASK Count:=Count+1;
NEXTSTATE -;

STATE *;

 INPUT Result;
   OUTPUT Score(Count) TO Player;
NEXTSTATE -;

 INPUT Endgame;
   OUTPUT Endsubscr;
STOP;

ENDPROCESS Game;
```

РИСУНОК 2.9.9

Пример спецификации процесса (PR)

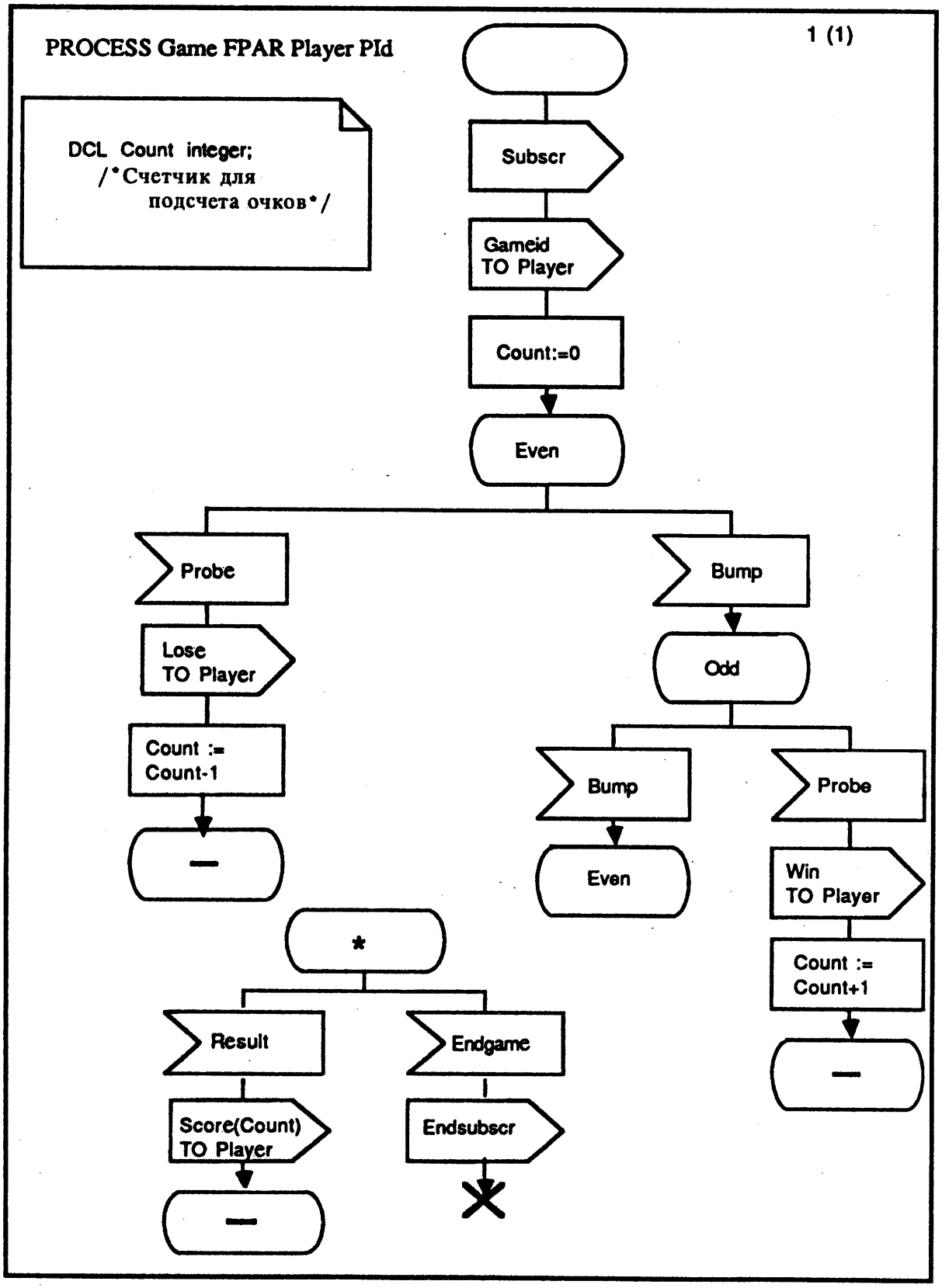


РИСУНОК 2.9.10

Пример спецификации процесса (GR)

```
PROCEDURE check;
/* Предполагается, что определены нижеследующие сигналы:
   SIGNAL sig1(Boolean), sig2, sig3(Integer,PId);   */
      FPAR IN/OUT x, y Integer;
      DCL  sum, index Integer,
           nice Boolean;
      START;
      TASK  sum := 0,
            index := 1;
      NEXTSTATE idle;
      STATE idle;
            INPUT sig1(nice);
            DECISION nice;
                  (true): TASK 'Calculate sum';
                        OUTPUT sig3(sum, SENDER);
                        RETURN;
                  (false): NEXTSTATE Jaj;
            ENDDECISION;
            INPUT sig2;
            ......
      .........
      .........

ENDPROCEDURE check;
```

РИСУНОК 2.9.11

Пример фрагмента спецификации процедуры (PR)

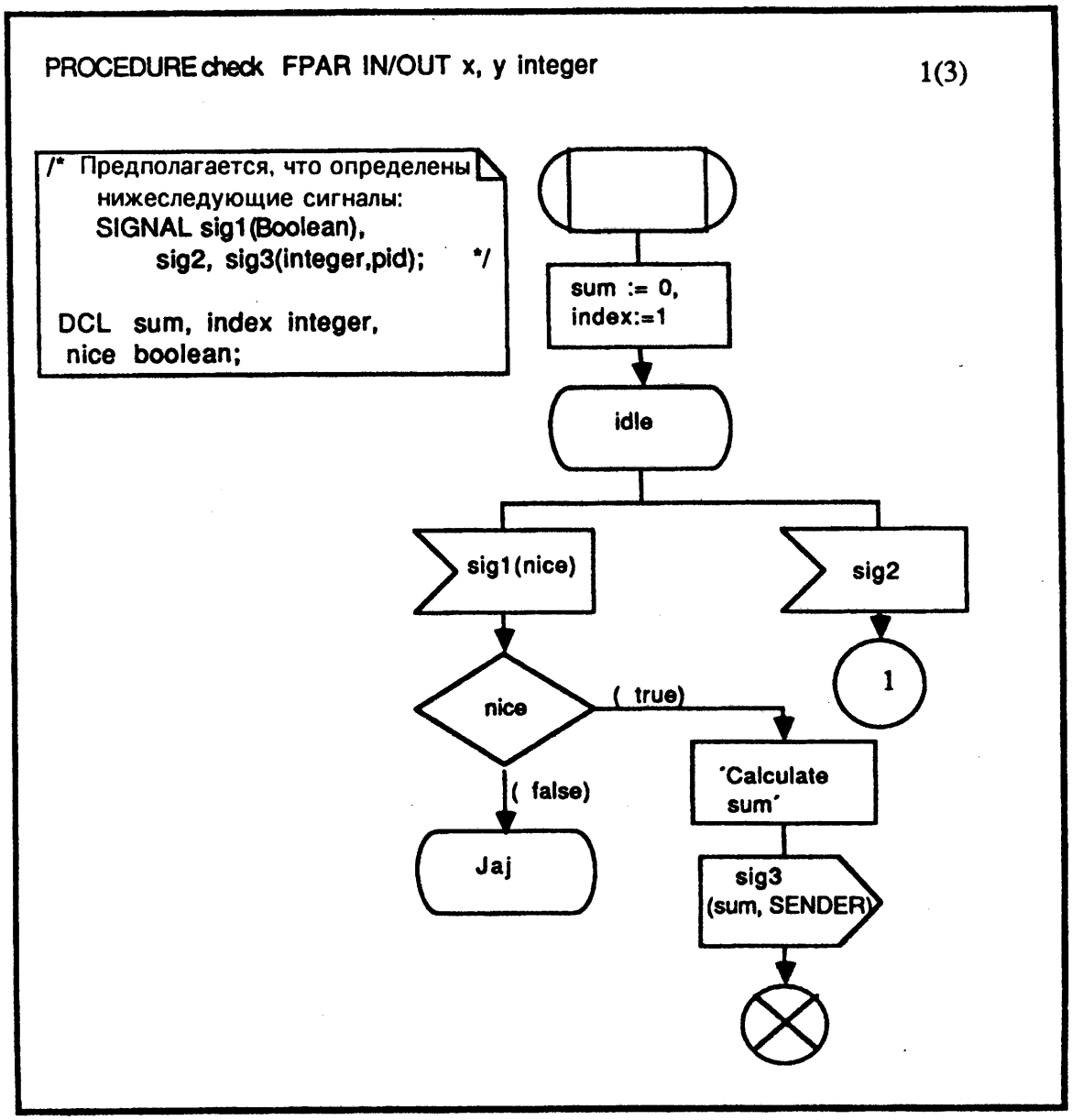


РИСУНОК 2.9.12

Пример фрагмента спецификации процедуры (GR)

```
/* Предполагается, что определены нижеследующие сигналы:
SIGNAL inquire(Integer,Integer,Integer);  */
PROCESS alfa;
DCL a,b,c Integer;
        .....
        ......
        INPUT inquire (a,b,c);
        CALL check (a,b);
        ......
ENDPROCESS;
```

РИСУНОК 2.9.13

Пример вызова процедуры из фрагмента определения процесса (PR)

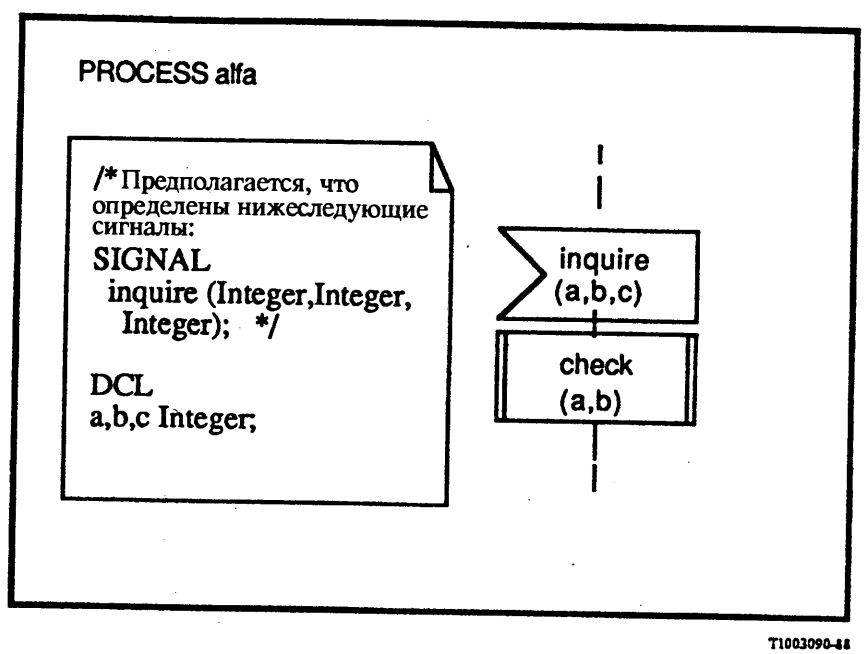


РИСУНОК 2.9.14

Пример вызова процедуры из фрагмента определения процесса (GR)

3 Понятие структурирования в SDL

3.1 *Введение*

В этом разделе описываются несколько понятий, используемых для создания с помощью SDL иерархических структур. Основы этих понятий заложены в § 2, так что понятия, определяемые в § 3, являются прямыми дополнениями к описанным в § 2.

Понятия, вводимые в этом разделе, имеют целью дать пользователю SDL средства для спецификации больших и/или сложных систем. Понятия, определенные в § 2, достаточны для спецификации сравнительно небольших систем, которые можно понять и обрабатывать, введя в рассмотрение только один уровень блоков. При спецификации больших или сложных систем значительно удобнее разбить спецификацию системы на обозримые компоненты, которые могут быть изучены и поняты независимо друг от друга. В частности, зачастую удобно бывает осуществить разбиение по нескольким уровням, в результате чего возникает иерархическая структура компонент, специфицирующих систему.

Термин "разбиение" означает разбиение компоненты на более мелкие подкомпоненты, входящие в исходную компоненту. Такое разбиение не должно влиять на статический интерфейс исходной компоненты. Одновременно с разбиением возникает потребность в добавлении новых деталей в поведении системы по мере спуска к более низким уровням иерархической структуры спецификации системы. Это обозначают термином уточнение.

3.2 *Разбиение*

3.2.1 *Общие положения*

Определение блока может быть разбито на совокупность определений подблоков, определений каналов и определений подканалов. Аналогично, определение канала может быть разбито на совокупность определений блоков, определений каналов и определений подканалов. Таким образом, любое определение блока или определение канала может иметь две версии: версию, не содержащую разбиения, и версию, содержащую разбиение в конкретном синтаксисе. Однако при отображении на абстрактный синтаксис канальные подструктуры преобразовывают. Обе версии имеют один и тот же статический интерфейс, но их поведение может в некоторой степени отличаться, поскольку порядок выдачи выходных сигналов может быть иным. Определение подблока является в то же время определением блока, а определение подканала — определением канала.

Как конкретное определение системы, так и абстрактное определение системы могут содержать две версии определения блока: без разбиения и с разбиением. В таком случае конкретное определение системы содержит несколько согласованных подмножеств разбиения, каждое из которых соответствует экземпляру системы. Согласованным подмножеством разбиения является такая совокупность *определений блоков,* входящих в *определение системы,* которая:

a) вместе с *Определением-блока* содержит определение охватывающей структурной единицы, если такая имеется;

b) должна содержать все *Определения-блоков,* введенных на уровне системы, и вместе с *Определением-подблока Определения-блока* содержит все остальные *Определения-подблоков* этого *Определения-блока;*

c) все *Определения-блока,* образующие "листья" полученной таким образом структуры, содержат *Определения-процессов.*

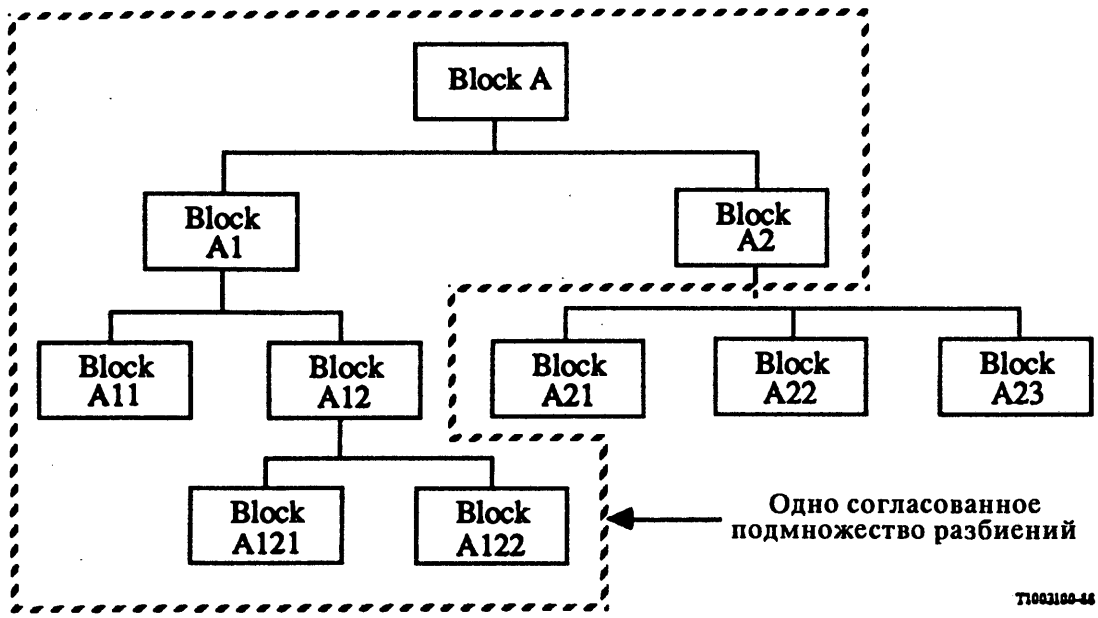


РИСУНОК § 3.2.1

Согласованное подмножество разбиения, иллюстрированное во вспомогательной диаграмме

При интерпретации системы интерпретируется согласованное подмножество разбиения.
Интерпретируются процессы, находящиеся в листьях согласованного подмножества разбиения. Если эти листья также имеют подструктуры, последние не оказывают никакого воздействия. Подструктуры, находящиеся в блоках, не являющихся листьями, оказывают воздействие на видимость, но процессы в этих блоках не интерпретируются.

3.2.2 *Разбиение блока*

Абстрактная грамматика

Определение-подструктуры-блока	::	*Имя-подструктуры-блока* *Определение-подблока-set* *Соединение-канала-set* *Определение-канала-set* *Определение-сигнала-set* *Определение-типа-данных* *Определение-подтипа-set*
Имя-подструктуры-блока	=	*Имя*
Определение-подблока	=	*Определение-блока*
Соединение-канала	::	*Идентификатор-канала* *Идентификатор-подканала-set*
Идентификатор-подканала	=	*Идентификатор-канала*
Идентификатор-канала	=	*Идентификатор*

Определение-подструктуры-блока должно содержать хотя бы одно *Определение-подблока.* В дальнейшем предполагается, если только не оговорено противное, что имеется в виду *Определение-подструктуры-блока* в абстрактном синтаксисе.

Идентификатор-блока, содержащийся в *Определении-канала,* должен обозначать *Определение-подблока. Определение-канала,* присоединяющего *Определение-подблока* к границе *Определения-подструктуры-блока,* называется определением подканала.

Для любого внешнего *Определения-канала,* присоединенного к *Определению-подструктуры-блока,* должно быть равно одно *Соединение-канала. Идентификатор-канала* в *Соединении-канала* должен описывать внешнее *Определение-канала.*

Для сигналов, направленных из *Определения-подструктуры-блока* наружу, объединение *Идентификаторов-сигналов,* ассоциируемых с *Идентификатором-подканала-set,* содержащемся в *Соединении-канала,* должно совпадать с *Идентификаторами-сигналов,* ассоциируемых с *Идентификатором-канала,* содержащемся в *Соединении-канала.* То же самое правило должно быть выполнено для сигналов, направленных внутрь *Определения-подструктуры-блока.* Однако, в случае уточнения сигналов, это правило несколько изменяется (см. § 3.3).

Каждый *Идентификатор-подканала* должен использоваться в одном и только одном *Соединении-канала.*

Поскольку *Определение-подблока* является в то же время и *Определением-блока,* оно в свою очередь может быть подвергнуто разбиению. Такое разбиение может быть повторено любое число раз, в результате чего образуется иерархическая древовидная структура *Определений-блоков* и их *Определений-подблоков. Определения-подблоков Определения-блока* считаются расположенными на следующем нижлежащем уровне дерева блоков; см. также нижеприводимый рисунок.

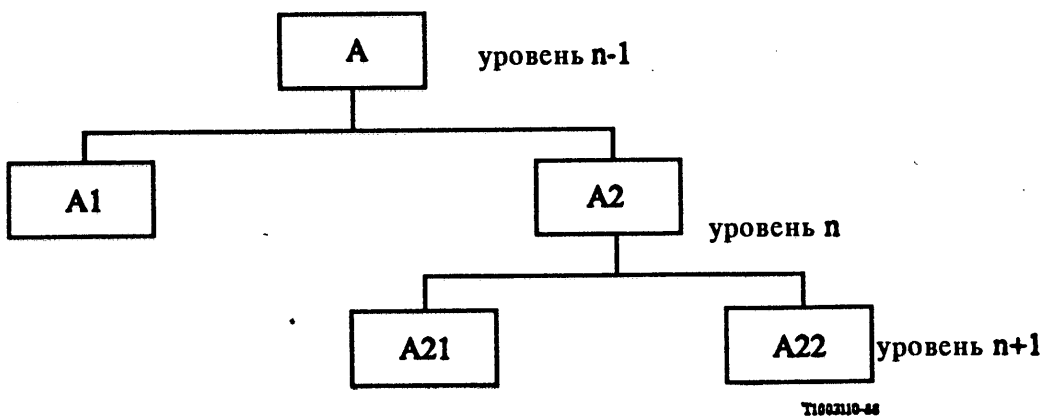


РИСУНОК 1/§ 3.2.2

Древовидная диаграмма блока

Древовидная диаграмма блока является вспомогательной диаграммой.

Конкретная текстовая грамматика

<определение подструктуры блока> ::=
SUBSTRUCTURE { [<имя подструктуры блока>]
| <идентификатор подструктуры блока>} <конец>
{<определение блока>
| <текстовая ссылка на блок>
| <определение канала>
| <соединение канала>
| <определение сигнала>
| <определение списка сигналов>
| <определение данных>
| <определение отбора>
| <макроопределение>}+
ENDSUBSTRUCTURE [{<имя подструктуры блока>
| <идентификатор подструктуры блока>}] <конец>

<имя подструктуры блока>, находящееся после ключевого слова SUBSTRUCTURE, может быть опущено только если оно совпадает с <именем блока>, охватывающего <определения блока>.

<текстовая ссылка на подструктуру блока> ::=
SUBSTRUCTURE <имя подструктуры блока> REFERENCED <конец>

<соединение канала> ::=
CONNECT <идентификатор канала> AND <идентификатор подканала>
{, <идентификатор подканала>}* <конец>

Конкретная графическая грамматика

<диаграмма подструктуры блока> ::=
<символ кадра>
contains {<заголовок подструктуры блока>
{{ <область текста подструктуры блока>}*
{<диаграмма макроса>}*
<область взаимодействия блоков>}set}
is associated with {<идентификатор канала>}*

<идентификатор канала> идентифицирует канал, присоединенный к подканалу в <диаграмме подструктуры блока>. Он помещается вне <символа кадра> впритык к точке окончания подканала на <символе кадра>.

<символ канала>, находящийся внутри <символа кадра> и присоединенный к нему, описывает подканал.

<заголовок подструктуры блока> ::=
SUBSTRUCTURE {<имя подструктуры блока> | <идентификатор подструктуры блока>}

<область текста подструктуры блока> ::=
<область текста системы>

<область подструктуры блока> ::=
<графическая ссылка на подструктуру блока>
| <диаграмма подструктуры блока>
| <диаграмма открытой подструктуры блока>

<графическая ссылка на подструктуру блока> ::=
<символ подструктуры блока> **contains** <имя подструктуры блока>

<символ подструктуры блока> ::=
<символ блока>

<диаграмма открытой подструктуры блока> ::=
{{ <область текста подструктуры блока>}*
{ диаграмма макроса }*
<область взаимодействия блоков>}**set**

Если <область подструктуры блока> является <диаграммой открытой подструктуры блока>, то охватывающая <диаграмма блока> не должна содержать ни <области текста блока>, ни <диаграммы макроса>, ни <области взаимодействия процесса>.

Семантика

См. § 3.2.1.

Модель

<диаграмма открытой подструктуры блока> преобразуется к <диаграмме подструктуры блока> таким образом, чтобы в <заголовке подструктуры блока> <имя подструктуры блока> или <идентификатор подструктуры блока> совпадал соответственно с <именем блока> или <идентификатором блока> в охватывающей <диаграмме блока>.

Пример

Ниже приводится пример <определения подструктуры блока>.

```
BLOCK A;
    SUBSTRUCTURE A ;
        SIGNAL s5(nat), s6, s8, s9(min);
        BLOCK a1 REFERENCED;
        BLOCK a2  REFERENCED;
        BLOCK a3  REFERENCED;
        CHANNEL c1 FROM a2 TO ENV WITH s1, s2; ENDCHANNEL c1;
        CHANNEL c2 FROM ENV TO a1 WITH s3;
            FROM a1 TO ENV WITH s1;  ENDCHANNEL c2;
        CHANNEL d1 FROM a2 TO ENV WITH s7; ENDCHANNEL d1;
        CHANNEL d2 FROM a3 TO ENV WITH s10; ENDCHANNEL d2;
        CHANNEL e1 FROM a1 TO a2 WITH s5, s6; ENDCHANNEL e1;
        CHANNEL e2 FROM a3 TO a1 WITH s8; ENDCHANNEL e2;
        CHANNEL e3 FROM a2 TO a3 WITH s9; ENDCHANNEL e3;
        CONNECT c AND c1, c2 ;
        CONNECT d AND d1, d2 ;
    ENDSUBSTRUCTURE A;
ENDBLOCK A ;
```

Ниже приводится <диаграмма подструктуры блока> для того же примера.

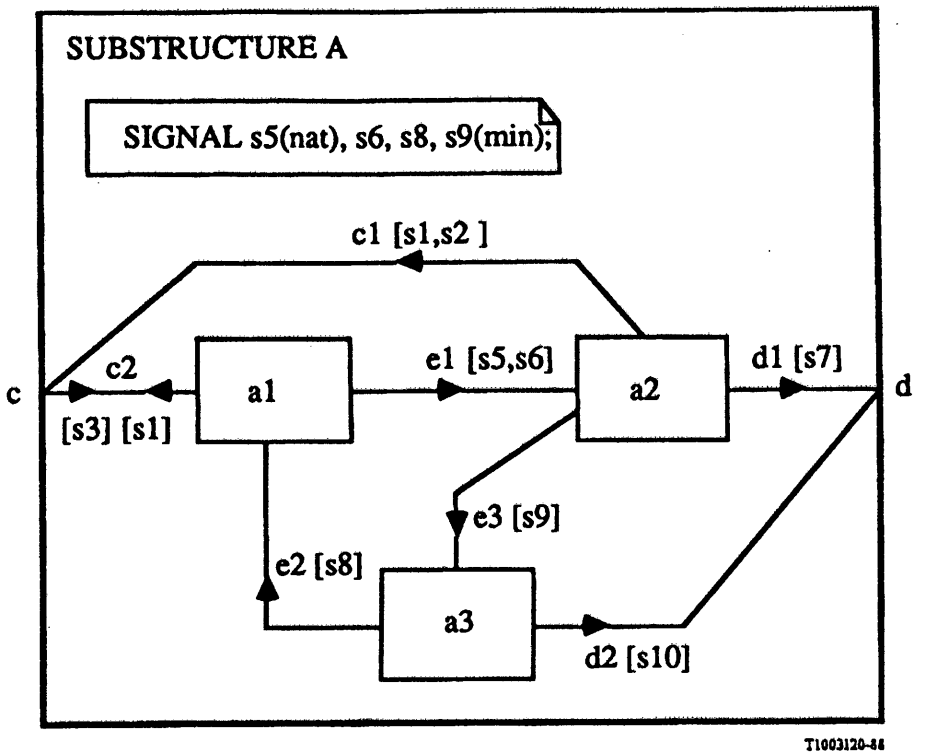


РИСУНОК 2/§ 3.2.2

Диаграмма подструктуры блока для блока A

3.2.3 *Разбиение канала*

Все статические условия описаны для случая конкретной текстовой грамматики. Аналогичные условия имеют место и для конкретной графической грамматики.

Конкретная текстовая грамматика

<определение подструктуры канала> ::=
SUBSTRUCTURE { [<имя подструктуры канала]
| <идентификатор подструктуры канала>} <конец>
{<определение блока>
| <текстовая ссылка на блок>
| <определение канала>
| <соединение конечной точки канала>
| <определение сигнала>
| <определение списка сигналов>
| <определение данных>
| <определение отбора>
| <макроопределение>}+
ENDSUBSTRUCTURE [{<имя подструктуры канала>
| <идентификатор подструктуры канала>}] <конец>

<имя подструктуры канала> после ключевого слова SUBSTRUCTURE может быть опущено, только если оно совпадает с <именем канала> в охватывающем <определении канала>.

<текстовая ссылка на подструктуру канала> ::=
SUBSTRUCTURE <имя подструктуры канала> REFERENCED <конец>

<соединение конечной точки канала> ::=
CONNECT {<идентификатор блока> | ENV} AND <идентификатор подканала>
{, <идентификатор подканала>}* <конец>

Для каждой конечной точки <определения канала>, подвергшегося разбиению, должно иметься только одно <соединение конечной точки канала>.<идентификатор блока> или ENVIRONMENT в <соединении конечной точки канала> должен идентифицировать одну из конечных точек <определения канала>, подвергшегося разбиению.

Конкретная графическая грамматика

<диаграмма подструктуры канала> ::=
<символ кадра>
contains {<заголовок подструктуры канала>
{{ <область текста подструктуры канала>}*
{<диаграмма макроса>}*
<область взаимодействия блоков>}**set**}
is associated with <идентификатор блока> | ENV}+

<идентификатор блока> или ENV идентифицирует конечную точку канала, подвергшегося разбиению. <идентификатор блока> помещается вне <символа кадра> впритык к точке окончания ассоциированного подканала на <символе кадра> . <символ канала>, находящийся внутри <символа кадра> и присоединенный к ней, описывает подканал.

<заголовок подструктуры канала> ::=
SUBSTRUCTURE <имя подструктуры канала>
| <идентификатор подструктуры канала> }

<область текста подструктуры канала> ::=
<область текста системы>

<область ассоциирования подструктуры канала> ::=
<символ пунктирной линии ассоциации>
is connected to <область подструктуры канала>

<область подструктуры канала> ::=
<графическая ссылка на подструктуру канала>
| <диаграмма подструктуры канала>

<графическая ссылка на подструктуру канала> ::=
<символ подструктуры канала> **contains** <имя подструктуры канала>

<символ подструктуры канала> ::=
<символ блока>

Модель

<определение канала>, содержащее <определение подструктуры канала>, преобразуется к <определению блока> и двум <определениям канала> так, что:

a) Каждый из двух <определений канала> присоединен к блоку и к конечной точке исходного канала. <определения канала> имеют новые различные имена и каждая из ссылок в конструкции VIA на исходный канал заменяется ссылкой на подходящий новый канал.

b) <определение блока> получает новое отличное имя и содержит только <определение подструктуры блока>, имеющего то же имя и содержащего те же определения, что и исходное <определение подструктуры канала>. Квалификаторы в новом <определении блока> изменены так, чтобы содержать имя блока. Два <соединения конечных точек канала> исходного <определения подструктуры канала> изображаются с помощью двух <соединений канала>, в которых <идентификатор блока> или ENV заменен на подходящий новый канал.

Эти преобразования должны быть выполнены сразу после выполнения преобразований для родовой системы. См. также § 4.3.

Пример

Ниже приводится пример <определения подструктуры канала>.

```
CHANNEL C FROM A TO B  WITH s1;
      FROM B TO A WITH s2;

    SUBSTRUCTURE C;

        SIGNAL s3(hel), s4(boo), s5;

        BLOCK b1 REFERENCED;
        BLOCK b2 REFERENCED;

        CHANNEL c1 FROM ENV TO b1 WITH s1;
            FROM b1 TO ENV WITH s2; ENDCHANNEL c1;

        CHANNEL c2 FROM b2 TO ENV WITH s1;
            FROM ENV TO b2 WITH s2; ENDCHANNEL c2;

        CHANNEL e1 FROM b1 TO b2 WITH s3; ENDCHANNEL e1;
        CHANNEL e2 FROM b2 TO b1 WITH s4, s5; ENDCHANNEL e2;

        CONNECT A AND c1;
        CONNECT B AND c2;

    ENDSUBSTRUCTURE C;

ENDCHANNEL C;
```

Ниже приводится <диаграмма подструктуры канала> для того же примера.

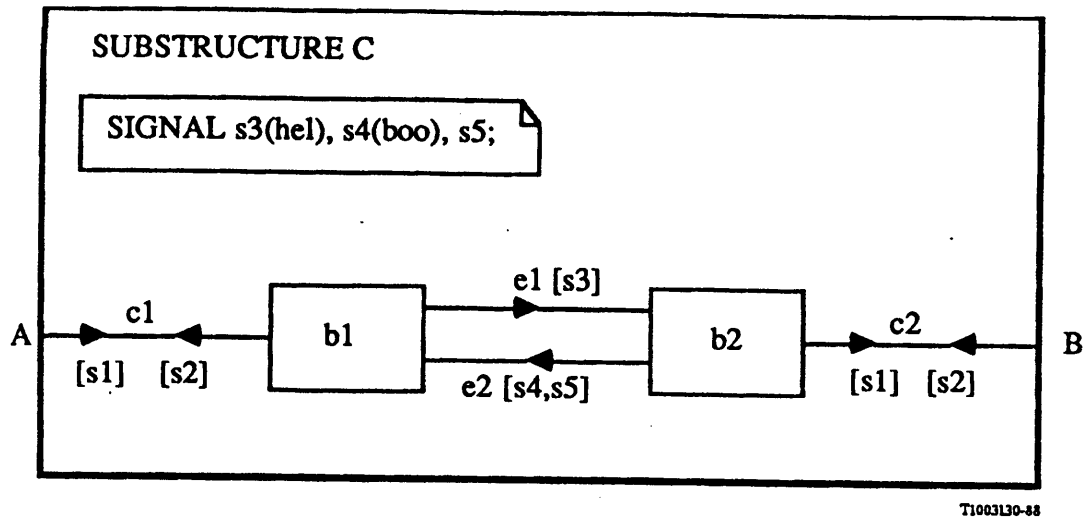


РИСУНОК § 3.2.3

Диаграмма подструктуры канала для канала C

3.3 *Уточнение*

Уточнение достигается пополнением определения сигнала за счет совокупности определений подсигналов. Определение подсигнала является в то же время определением сигнала и в свою очередь может подвергнуться уточнению. Такое уточнение можно повторять любое число раз, в результате чего получается иерархическая древовидная структура определений сигналов и определений их подсигналов. Обратите внимание на то, что определение подсигнала в некотором определении сигнала не считается компонентой этого определения сигнала.

Абстрактная грамматика

Уточнение-сигнала :: *Определение-подсигнала*-set

Определение-подсигнала :: [REVERSE] *Определение-сигнала*

Для любого *Соединения-канала* должно выполняться нижеследующее требование: каждый *Идентификатор-сигнала,* ассоциированный с *Идентификатором-канала,* либо ассоциирован хоть с одним *Идентификатором-подканала,* либо каждый из его идентификаторов подсигнала ассоциирован хоть с одним *Идентификатором-подканала.* Это правило отличается от соответствующих правил разбиения.

Никакие два сигнала в полной совокупности допустимых входных сигналов определения процесса или в *Выходных-вершинах* определения процесса не должны находиться на разных уровнях уточнения одного и того же сигнала.

Конкретная текстовая грамматика

<уточнение сигнала> ::=
REFINEMENT
<определение подсигнала> +
ENDREFINEMENT

<определение подсигнала> ::=
[REVERSE] <определение сигнала>

Семантика

Если для передачи сигнала выделен некоторый канал, то этот же канал автоматически выделяется для передачи всех подсигналов этого сигнала. Уточнение может происходить тогда, когда канал расщепляется (подвергается разбиению) на подканалы. В этом случае подканалы будут передавать подсигналы вместо сигнала, подвергшегося уточнению. Направление передачи подсигнала определяется передающим его подканалом и может быть обратным направлению сигнала, подвергшегося уточнению; это обстоятельство указывается ключевым словом REVERSE. Уточнение сигналов недопустимо в случае, когда канал подвергся расщеплению на маршруты сигнала.

Если определение системы содержит уточнение сигнала, то концепция согласованного подмножества разбиения подвергается ограничению. Про такую систему говорят, что она содержит несколько согласованных подмножеств уточнения.

Согласованное подмножество уточнения является согласованным подмножеством разбиения, на которое наложено дополнительное ограничение:

— для согласованного подмножества разбиения совокупность сигналов на маршрутах сигналов, присоединенных к конечной точке канала, не должна содержать родительских сигналов тех подсигналов, которые входят в указанную совокупность; кроме того, если другой конечной точкой оказывается ENVIRONMENT всей системы, то совокупность сигналов первой конечной точки должна совпадать с совокупностью сигналов на маршрутах сигналов, подсоединенных к другой конечной точке.

Пример

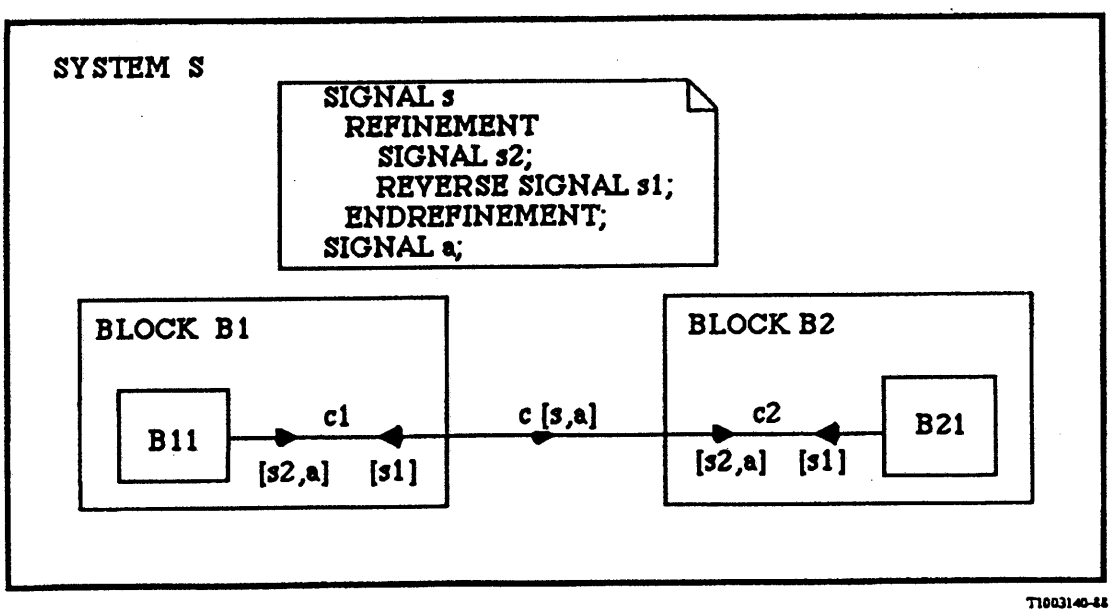


РИСУНОК § 3.3

Диаграмма системы, содержащая уточнение сигнала

В вышеприведенном примере в определениях блоков B1 и B2 уточняется сигнал s, но не сигнал a. На самом высоком уровне уточнения процессы в блоках B1 и B2 поддерживают связь друг с другом с помощью сигналов s и a. На следующем более низком уровне процессы в блоках B11 и B21 поддерживают связь с помощью сигналов s1, s2 и a.

Заметьте, что уточнение только в одном из определений блоков B1 или B2 недопустимо, так как между сигналом s и его подсигналами s1 и s2 имеется только статическое соотношение, но нет динамического преобразования.

4 Дополнительные понятия SDL

4.1 *Введение*

В настоящей главе вводятся несколько дополнительных понятий. Эти дополнительные понятия являются стандартными сокращенными обозначениями и моделируются с помощью основных понятий SDL на основе конкретного синтаксиса. Эти понятия вводятся для удобства пользователей SDL в дополнение к сокращенным обозначениям, введенным в других главах настоящей Рекомендации.

Свойства дополнительных понятий выводятся из свойств основных понятий, исходя из того, как дополнительные понятия моделируются основными (или как они преобразуются к основным). Для обеспечения легкого и однозначного использования дополнительных понятий и для исключения побочных эффектов в тех случаях, когда используется несколько дополнительных понятий, эти последние преобразуются к основным в нижеследующем фиксированном порядке:

1 Макрос § 4.2
2 Родовые системы § 4.3
3 Состояние "звездочка" § 4.4
4 Список состояний § 2.6.3
5 Кратное вхождение состояния § 4.5
6 Ввод "звездочка" § 4.6
7 Сохранение "звездочка" § 4.7
8 Список стимулов § 2.6.4
9 Список выводов § 2.7.4
10 Неявный переход § 4.8
11 Следующее состояние "черточка" § 4.9
12 Сервис § 4.10
13 Непрерывный сигнал § 4.11
14 Разрешающее условие § 4.12
15 Импортированное и экспортированное значения § 4.13

Дополнительные понятия описываются в настоящем разделе в том же порядке. Специфицированный порядок преобразования понятий означает, что при преобразовании дополнительного понятия с порядковым номером n другое дополнительное понятие с порядковым номером m может использоваться только если $m > n$.

Так как для сокращенных обозначений нет абстрактного синтаксиса, то при их определении используются термины графического синтаксиса или текстового синтаксиса. Выбор между терминами графического синтаксиса и терминами текстового синтаксиса осуществляется исходя из практических соображений и не означает, что дополнительное понятие может использоваться только в данном конкретном синтаксисе.

4.2 *Макросы*

В нижеследующем тексте термины "вызов макроса" и "макроопределение" используются в общем значении, соответствуя как SDL/GR, так и SDL/PR. Макроопределение содержит совокупность графических символов и/или лексических единиц; эта совокупность может быть включена в одно или несколько мест в <конкретное определение системы>. Каждое такое место задается вызовом макроса. Перед тем как <конкретное определение системы> может быть проанализировано, каждый вызов макроса должен быть заменен на соответствующее макроопределение.

4.2.1 *Лексические правила*

<формальное имя> ::=
[<имя> %] <макропараметр>
{% <имя> % <макропараметр> | % <макропараметр>}* [% <имя>]

4.2.2 *Макроопределение*

Конкретная текстовая грамматика

<макроопределение> ::=
MACRODEFINITION <имя макроса>
[<формальные макропараметры>] <конец>
<тело макроса>
ENDMACRO [<имя макроса>] <конец>

<формальные макропараметры> ::=
FPAR <формальный макропараметр>{ , <формальный макропараметр>}*

<формальный макропараметр> ::=
<имя>

<тело макроса> ::=
{<лексема> | <формальное имя>}*

<макропараметр> ::=
<формальный макропараметр>
| MACROID

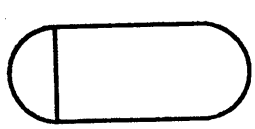
<формальные макропараметры> должны быть различными. <фактические макропараметры> вызова макроса должны быть согласованными с соответствующими <формальными макропараметрами>.

<тело макроса> не должно содержать ключевых слов ENDMACRO и MACRODEFINITION.

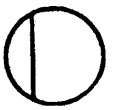
Конкретная графическая грамматика

<диаграмма макроса> ::=
<символ кадра> **contains** {<заголовок макроса> <область тела макроса>}
<заголовок макроса> ::=,
MACRODEFINITION <имя макроса> [<формальные макропараметры>]
<область тела макроса> ::=
{{<любая область>}*
<любая область> [**is connected to** <порт 1 тела макроса>]} **set**
| {<любая область> **is connected to** <порт2 тела макроса>
<любая область> **is connected to** <порт2 тела макроса>
{<любая область> [**is connected to** <порт2 тела макроса>]}*} **set**

<символ входной точки в макрос> ::=



<символ выходной точки из макроса> ::=



<порт1 тела макроса> ::=
<символ выходной точки> **is connected to** <символ кадра>
[**is associated with** <метка макроса>]
| <символ входной точки в макрос> [{**contains** | **is associated with** } <метка макроса>]
| <символ выходной точки из макроса> [{**contains** | **is associated with** } <метка макроса>]}

<порт2 тела макроса> ::=
<символ выходной точки> **is connected to** <символ кадра>
is associated with <метка макроса>
| <символ входной точки в макрос>{ **contains** | **is associated with** }<метка макроса>
| <символ выходной точки из макроса>{ **contains** | **is associated with** }<метка макроса>}

<метка макроса> ::=
<имя>

<символ выходной точки> ::=
<символ фиктивной выходной точки>
| <символ линии потока>
| <символ канала>
| <символ маршрута сигнала>
| <символ сплошной линии ассоциации>
| <символ пунктирной линии ассоциации>
| <символ линии создания>

<символ фиктивной выходной точки> ::=
<символ сплошной линии ассоциации>

<любая область> ::=
<область текста системы>
| <область взаимодействия блоков>
| <область списка сигналов>
| <область блока>
| <область текста блока>
| <область взаимодействия процессов>
| <графическая ссылка на процедуру>
| <область текста процесса>
| <область графа процесса>
| <область слияния>
| <область цепочки перехода>
| <область состояния>
| <область ввода>
| <область сохранения>
| <область установки>
| <область сброса>
| <область экспорта>
| <область расширения текста>
| <область ассоциирования подструктуры канала>
| <область подструктуры канала>

| <область подструктуры блока>
| <область приоритетного ввода>
| <область непрерывного сигнала>
| <область входного коннектора>
| <область следующего состояния>
| <область процесса>
| <область определения канала>
| <область линий создания>
| <область определения маршрута сигнала>
| <графическая ссылка на процесс>
| <диаграмма процесса>
| <область старта>
| <область вывода>
| <область приоритетного вывода>
| <область работы>
| <область запроса на создание>
| <область вызова процедуры>
| <область процедуры>
| <область принятия решения>
| <область выходного коннектора>
| <область текста процедуры>
| <область графа процедуры>
| <область старта процедуры>
| <область текста подструктуры блока>
| <область взаимодействия блоков>
| <область сервиса>
| <область определения маршрута сигнала сервиса>
| <область текста сервиса>
| <область графа сервиса>
| <область старта сервиса>
| <область комментария>
| <область вызова макроса>
| <область ассоциирования ввода>
| <область ассоциирования сохранения>
| <область отбора>
| <область текста подструктуры канала>
| <область выбираемого перехода>
| <область взаимодействия сервисов>
| <область ассоциирования приоритетного ввода>
| <область ассоциирования непрерывного сигнала>
| <область решающего условия>

<символ фиктивной выходной точки> может быть ассоциирован только с <меткой макроса>.

<символу выходной точки>, не являющемуся <символом фиктивной выходной точки>, в вызове макроса должен соответствовать только <символ ввода>, являющийся <символом фиктивной входной точки>.

<тело макроса> может появиться в любом тексте, к которому есть ссылка в <любой области>.

Семантика

<макроопределение> содержит лексемы, в то время как <диаграмма макроса> содержит синтаксические единицы. Поэтому, отображение друг на друга конструкций макросов в текстовом синтаксисе и в графическом синтаксисе вообще говоря невозможно. В силу того же самого к текстовому синтаксису и графическому синтаксису применимы различные права детализации, хотя и имеется несколько общих правил.

<имя макроса> доступно во всем определении системы, независимо от того, где появляется макроопределение. Вызов макроса может появиться до соответствующего макроопределения.

Макроопределение может содержать вызовы макросов, но макроопределение не должно содержать вызовов самого себя ни непосредственно, ни косвенно через вызовы других макроопределений.

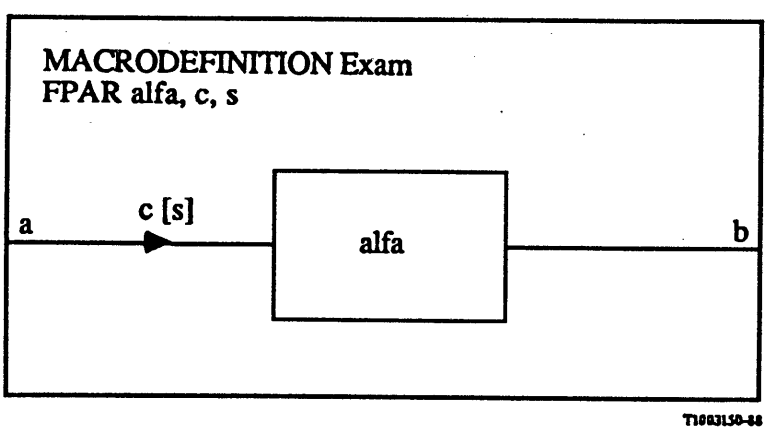
Ключевое слово MACROID может использоваться в качестве псевдо-формального макропараметра в каждом макроопределении. Такому формальному параметру не может быть передан никакой <фактический макропараметр>; в каждом макрорасширении он заменяется уникальным <именем> (в каждом конкретном расширении все вхождения ключевого слова MACROID заменяется на одно и то же <имя>).

Пример

Ниже приводится пример <макроопределения>:

```
MACRODEFINITION Exam
FPAR alfa, c, s, x;
     BLOCK alfa REFERENCED;
     CHANNEL c  FROM x TO alfa WITH s; ENDCHANNEL c;
ENDMACRO Exam;
```

Ниже приводится <диаграмма макроса> для того же примера. Однако в этом случае <формальный макропараметр> "x" не требуется.



T1083150-88

4.2.3 *Вызов макроса*

Конкретная текстовая грамматика

<вызов макроса> ::=
MACRO <имя макроса> [<тело вызова макроса>] <конец>

<тело вызова макроса> ::=
(<фактический макропараметр>{ , <фактический макропараметр>}*)
<фактический макропараметр> ::=
{<лексема>}*

В качестве <лексемы> не может быть взята запятая", "или правая скобка")". Если в <фактическом макропараметре> должен быть использован любой из этих двух знаков, то <фактический макропараметр> должен быть <знаковой строкой>. В этом случае при замене <формального макропараметра> <фактическим макропараметром> используется значение <знаковой строки>.

<вызов макроса> может стоять в любом месте, в котором допустимо использование <лексемы>.

Конкретная графическая грамматика

<область вызова макроса> ::=
<символ вызова макроса> **contains** {<имя макроса> [<тело вызова макроса>]}
[**is connected to**
{<порт1 вызова макроса> | <порт2 вызова макроса>{<порт2 вызова макроса>}+}]

<символ вызова макроса> ::=

<порт1 вызова макроса> ::=
<символ входной точки> [**is associated with** <метка макроса>]
is connected to <любая область>

<порт2 вызова макроса> ::=
<символ входной точки> **is associated with** <метка макроса>
is connected to <любая область>

<символ входной точки> ::=
<символ фиктивной входной точки>
| <символ линии потока>
| <символ канала>
| <символ маршрута сигнала>
| <символ сплошной линии ассоциации>
| <символ пунктирной линии ассоциации>
| <символ линии создания>

<символ фиктивной входной точки> ::=
<символ сплошной линии ассоциации>

<символ фиктивной входной точки> может быть ассоциирован только с <меткой макроса>.

С каждым <символом входной точки> в соответствующей <диаграмме макроса> должен сопоставляться <символ выходной точки>, ассоциированный с такой же <меткой макроса>. Если <символ входной точки> не является <символом фиктивной входной точки>, то соответствующий <символ выходной точки > должен быть <символом фиктивной выходной точки>.

За исключением случаев <символа фиктивной входной точки> и <символа фиктивной выходной точки> допустимо ассоциировать кратные (текстовые) <лексемы> с <символом входной точки> или с <символом выходной точки>. В этом случае <лексема>, ближайшая к <символу вызова макроса> или <символу кадра> <диаграммы макроса>, используется в качестве <метки макроса>, ассоциируемой с <символом входной точки> или с <символом выходной точки>.

<область вызова макроса> может появиться в любом месте, в котором может стоять какая-нибудь область. Однако требуется оставить некоторое пространство между <символом вызова макроса> и любым другим замкнутым графическим символом. Если, в соответствии с синтаксическими правилами, это пространство не должно быть пустым, то <символ вызова макроса> соединяется с замкнутым графическим символом <символом фиктивной входной точки>.

Семантика

Определение системы может содержать макроопределения и вызовы макросов. Для анализа определения системы все вызовы макросов должны быть расширены. Под расширением вызова макроса понимают замену вызова макроса на копию макроопределения, имя которого совпадает с именем, указанным в вызове.

Вызванное для замены макроопределение также подвергается расширению. Это означает, что создается копия макроопределения и каждое вхождение <формального макропараметра> в копию заменяется на соответствующий <фактический макропараметр>, имеющийся в вызове макроса. Затем подвергаются такому же расширению и все вызовы макросов, если таковые имеются, в макроопределении. При замене <формальных макропараметров> на <фактические макропараметры> все знаки процента (%), имеющиеся в <формальных именах>, удаляются.

Между <формальными макропараметрами> и <фактическими макропараметрами> должно быть взаимно однозначное соответствие.

Правила графического синтаксиса

<область вызова макроса> заменяется на копию <диаграммы макроса> следующим образом. Удаляются все имеющиеся <символы входной точки в макрос> и <символы выходной точки из макроса>. <символ фиктивной выходной точки> заменяется на <символ входной точки> с такой же <меткой макроса>. <символ фиктивной входной точки> заменяется на <символ выходной точки> с такой же <меткой макроса>. Затем <метки макросов>, присоединенные к <символам входной точки> и <символам выходной точки>, удаляются. <порт1 тела макроса> и <порт2 тела макроса>, не имеющие соответствующих <порта1 вызова макроса> и <порта2 вызова макроса>, также удаляются.

Пример

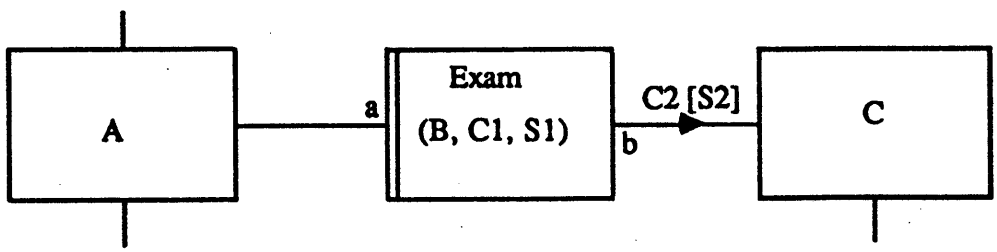
Ниже приводится пример <вызова макроса>, содержащийся во фрагменте <определения блока>.

```
.....
BLOCK A REFERENCED;
MACRO Exam (B, C1, S1, A);
BLOCK C REFERENCED;
CHANNEL C2 FROM B TO C WITH S2; ENDCHANNEL C2;
.....
```

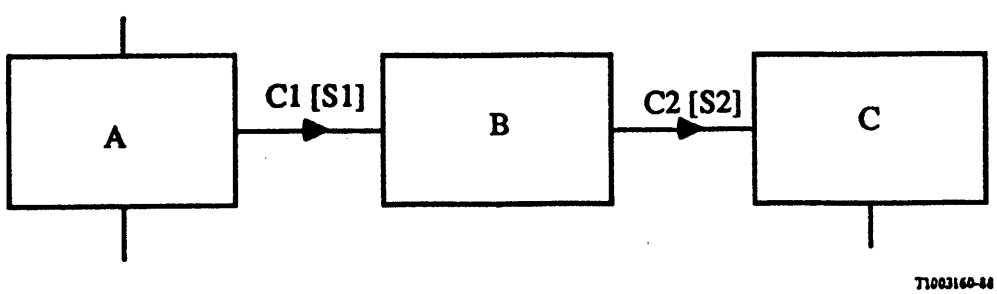
Расширение этого вызова макроса, использующее пример из § 4.2.2, дает следующий результат.

```
.....
BLOCK A REFERENCED;
BLOCK B REFERENCED;
CHANNEL C1 FROM A TO B WITH S1; ENDCHANNEL C1;
BLOCK C REFERENCED;
CHANNEL C2 FROM B TO C WITH S2; ENDCHANNEL C2;
.....
```

<область вызова макроса>, содержащаяся во фрагменте <области взаимодействия блоков>, для того же примера приводится ниже.



Расширение этого вызова макроса дает следующий результат.



4.3 *Родовые системы*

Определение системы может содержать необязательно выбираемые части и системные параметры с неопределенными значениями в целях удовлетворения различных потребностей. Такие определения системы называются родовыми; их родовые свойства описываются с помощью внешних синонимов (аналогичным формальным параметрам в определении процедуры). Спецификация получается из родовой системы отбором из нее подходящего подмножества и приданием значений каждому системному параметру. Получающаяся в результате спецификация системы не содержит внешних синонимов и называется специализированной спецификацией системы.

4.3.1 *Внешний синоним*

Конкретная текстовая грамматика

<определение внешнего синонима> ::=
SYNONYM <имя внешнего синонима> <предопределенный сорт> = EXTERNAL

<внешний синоним> ::=
<идентификатор внешнего синонима> можно помещать в любом месте

<определение внешнего синонима> можно помещать в любом месте, где допустимо ставить <определение синонима>, см. § 5.4.1.13. <внешний синоним> можно использовать в любом месте, где допустим <синоним>, см. § 5.4.2.3. Предопределенными сортами являются: Булевский (Boolean), Знаковый (Character), Знакострокового (Charstring), Целый (Integer), Натуральный (Natural), Вещественный (Real), PId (Идентификатор процесса), Длительность (Duration) и Временной (Time).

Семантика

<внешний синоним> является <синонимом>, значение которого не специфицировано в определении системы. Это указывается ключевым словом EXTERNAL, используемым вместо <простого выражения>.

Определение родовой системы является определением системы, содержащим <внешние синонимы> или <неформальный текст> в выбираемом переходе (см. § 4.3.4). Специализированное определение системы создается из определения родовой системы за счет обеспечения <внешних синонимов> значениями и заменой <неформального текста> формальными конструкциями. Вопрос о том, как это осуществляется, и отношение к абстрактной грамматике не входит в описание языка.

4.3.2 *Простое выражение*

Конкретная текстовая грамматика

<простое выражение> ::=
<начальное выражение>

<простое выражение> должно содержать только символы операций, синонимы и литералы предопределенных сортов.

Семантика

Простое выражение является *Начальным-выражением.*

4.3.3 *Выбираемое определение*

Конкретная текстовая грамматика

<определение отбора> ::=
SELECT IF (<булевское простое выражение>) <конец>
{<определение блока>
| <текстовая ссылка на блок>
| <определение канала>
| <определение сигнала>
| <определение списка сигналов>
| <определение данных>
| <определение процесса>
| <текстовая ссылка на процесс>
| <определение таймера>
| <определение маршрута сигнала сервиса>
| <соединение канала>
| <соединение конечной точки канала>
| <определение переменной>
| <определение обозревания>
| <определение импорта>
| <определение процедуры>
| <текстовая ссылка на процедуру>
| <определение сервиса>
| <текстовая ссылка на сервис>
| <определение маршрута сигнала>
| <соединение канала с маршрутом>
| <соединение маршрута сигнала>
| <определение отбора>
| <макроопределение>}$^{+}$

ENDSELECT <конец>

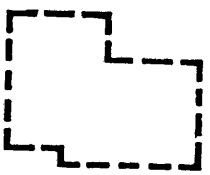
<булевское простое выражение> не должно зависеть ни от одного определения, входящего в <определение отбора>. <определение отбора> должно содержать только такие определения, которые являются синтаксически допустимыми в этом месте.

Конкретная графическая грамматика

<область отбора> ::=
<символ отбора> contains
{SELECT IF (<булевское простое выражение>)
{ <область блока>
| <область определения канала>
| <область текста системы>
| <область текста блока>
| <область текста процесса>
| <область текста процедуры>
| <область текста подструктуры блока>
| <область текста подструктуры канала>
| <область текста сервиса>
| <диаграмма макроса>

| <область процесса>
| <область определения маршрута сигнала>
| <область линии создания>
| <область процедуры>
| <область отбора>
| <область сервиса>
| <область определения маршрута сигнала сервиса>}+}

<символом отбора> является многоугольник со сплошными углами, сторонами которого являются отрезки пунктирной линии. Например,



Считается, что <символ отбора> логически полностью содержит весь одномерный графический символ, пересекаемый его границей, (то есть когда одна конечная точка этого одномерного символа оказывается вне <символа отбора>).

<булевское простое выражение> не должно зависеть ни от одной диаграммы или области, находящейся внутри <области отбора>.

<область отбора> может стоять всюду, кроме <области графа процесса>, <области графа процедуры> и <области графа сервиса>. <область отбора> должна содержать только те области и диаграммы, которые являются синтаксически допустимыми в этом месте.

Семантика

Если значение <булевского простого выражения> ложно, то конструкции, содержащиеся в <определении отбора> и <символе отбора> не отбираются. В противном случае конструкции отбираются.

Модель

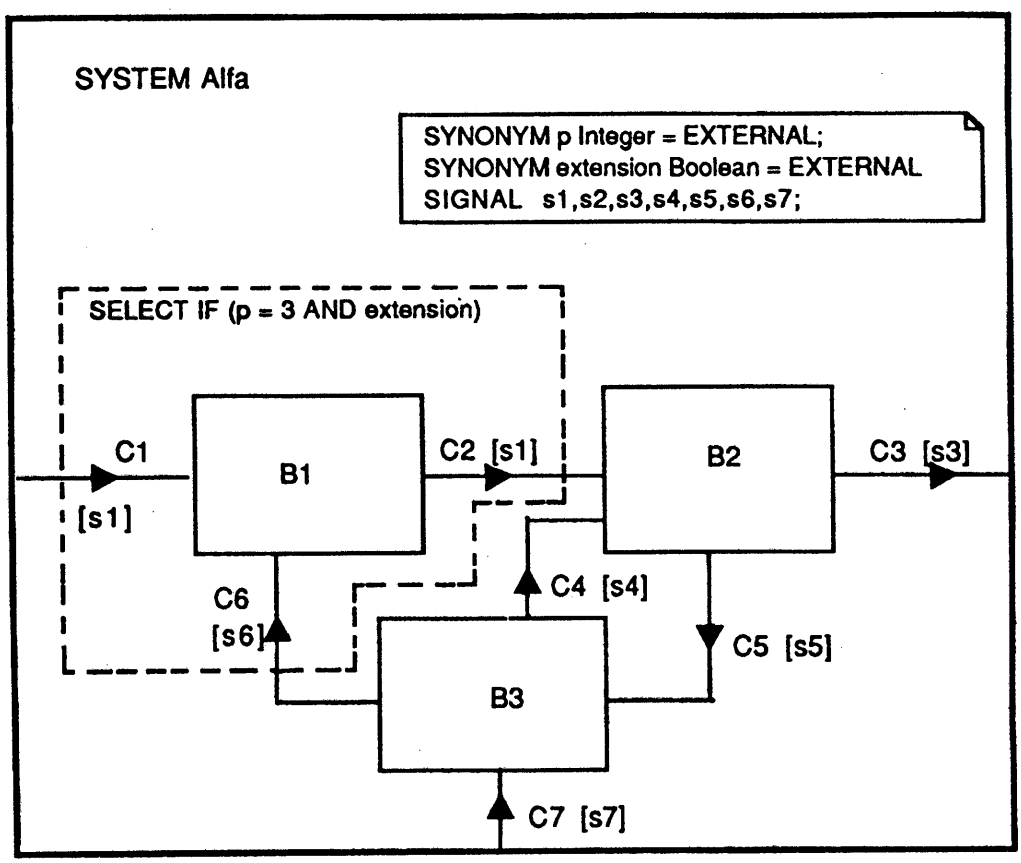
На этапе преобразования <определение отбора> и <область отбора> исключаются и заменяются на содержащиеся в них отобранные конструкции, если таковые оказываются в наличии.

Пример

В системе Alfa имеются три блока: B1, B2, B3. Блок B1 и каналы, присоединенные к нему, являются выбираемыми в зависимости от значения внешнего синонима p и расширения. В SDL/PR этот пример имеет следующий вид.

```
SYSTEM Alfa;
      SYNONYM p Integer = EXTERNAL;
      SYNONYM extension Boolean = EXTERNAL;
      SIGNAL  s1,s2,s3,s4,s5,s6,s7;
      SELECT  IF (p = 3 AND extension);
        BLOCK B1 REFERENCED;
        CHANNEL C1 FROM ENV TO B1 WITH s1 ; ENDCHANNEL C1;
        CHANNEL C2 FROM B1 TO B2 WITH s2 ; ENDCHANNEL C2;
        CHANNEL C6 FROM B3 TO B1 WITH s6; ENDCHANNEL C6;
      ENDSELECT;
      CHANNEL C3 FROM B2 TO ENV WITH s3 ; ENDCHANNEL C3;
      CHANNEL C4 FROM B3 TO B2 WITH s4 ; ENDCHANNEL C4;
      CHANNEL C5 FROM B2 TO B3 WITH s5; ENDCHANNEL C5;
      CHANNEL C7 FROM ENV TO B3 WITH s7 ; ENDCHANNEL C7;
      BLOCK B2 REFERENCED;
      BLOCK B3 REFERENCED;
ENDSYSTEM Alfa;
```

Тот же пример в SDL/GR изображается нижеследующим образом.



4.3.4 *Выбираемая цепочка перехода*

Конкретная текстовая грамматика

<выбираемый переход> ::=
ALTERNATIVE <вопрос альтернативы> <конец>
{<раздел ответа> <раздел иначе>
| <раздел ответа> {<раздел ответа>}+ [<раздел иначе>]}
ENDALTERNATIVE

<вопрос альтернативы> ::=
<простое выражение>
| <неформальный текст>

Каждое <начальное выражение> в <ответе> должно быть <простым выражением>. <ответы> в <выбираемом переходе> должны быть попарно взаимно исключающими. Если <вопрос альтернативы> является <выражением>, то *Условие-на-диапазон* <ответов> должно быть того же сорта, что и <вопрос альтернативы>.

Конкретная графическая грамматика

<область выбираемого перехода> ::=
<символ выбираемого перехода> **contains** {<вопрос альтернативы>}
is followed by {<выбираемая выходная точка1> {<выбираемая выходная точка1> | <выбираемая выходная точка2>}
{<выбираемая выходная точка1>}*} **set**

<символ выбираемого перехода> ::=

<выбираемая выходная точка1> ::=
<символ линии потока> **is associated with** <графический ответ>
is followed by <область перехода>

<выбираемая выходная точка2> ::=
<символ линии потока> **is associated with** ELSE
is followed by <область перехода>

<символ линии потока> в <выбираемой выходной точке1> и <выбираемой выходной точке2> подсоединены к основанию <символа выбираемого перехода>. <символы линии потока>, исходящие из <символа выбираемого перехода>, могут иметь общий отрезок. <графические ответы> и ELSE могут быть помещены либо вдоль соответствующих <символов линии потока>, либо в их разрывах.

<графические ответы> в <области выбираемого перехода> должны быть попарно взаимно исключающими.

Семантика

Конструкции в <выбираемой выходной точке1> бывают отобранными, если <ответ> содержит значение <вопроса альтернативы>. Если ни один из <ответов> не содержит значения <вопрос альтернативы>, то тогда бывают отобраны конструкции из <выбираемой выходной точки2>.

Если <выбираемая выходная точка2> отсутствует, и ни один из исходящих путей не оказывается отобранным, то отбор недопустим.

Модель

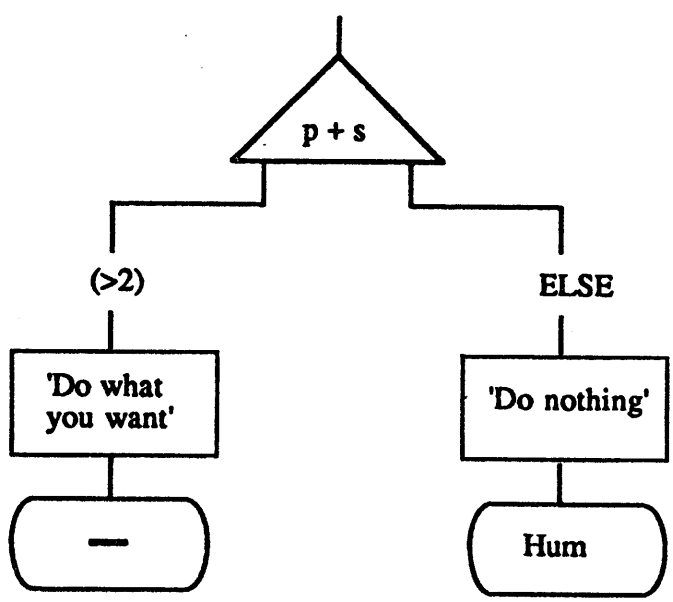
На этапе преобразования <выбираемый переход> и <область выбираемого перехода> исключаются и заменяются на отобранные конструкции, содержащиеся в них.

Пример

Ниже приведен фрагмент <определения процесса>, содержащий <выбираемый переход>. В нем *p* и *s* являются синонимами.

```
.....
ALTERNATIVE p + s;
        (>2) : TASK 'Do what you want';
               NEXTSTATE -;
        ELSE: TASK 'Do nothing';
               NEXTSTATE Hum;
ENDALTERNATIVE;
.....
```

Тот же пример в конкретном графическом синтаксисе имеет нижеследующий вид.



4.4 *Состояние "звездочка"*

Конкретная текстовая грамматика

<список состояний "звездочка"> ::=
<звездочка> [(<имя состояния>} , <имя состояния>}*)]

<звездочка> ::=
*

В <теле процесса>, <теле процедуры> или <теле сервиса> по крайней мере один <список состояний> должен быть отличным от <списка состояний "звездочка">. Все <имена состояний> в <списке состояний "звездочка"> должны быть различными и должны содержаться в других <списках состояний> охватывающего <тела процесса>, <тела процедуры> или <тела сервиса>.

<имена состояний>, входящие в <список состояний "звездочка"> не должен содержать все <имена состояний> охватывающего <тела процесса>, <тела процедуры> или <тела сервиса>.

Конкретная графическая грамматика

<область состояния>, содержащая <список состояний "звездочка">, не должна совпадать с <областью следующего состояния>.

Модель

<список состояний "звездочка"> преобразуется в <список состояний>, содержащий все <имена состояний> рассматриваемого <тела процесса>, <тела процедуры> или <тела сервиса>, за исключением <имен состояний>, входящих в <список состояний "звездочка">.

4.5 *Кратное вхождение состояния*

Конкретная текстовая грамматика

<имя состояния> может использоваться в более чем одном <состоянии> <тела процесса>, <тела сервиса> или <тела процедуры>.

Модель

Если несколько <состояний> имеют одно и то же <имя состояния>, то эти <состояния> сливаются в одно <состояние> с тем же самым <именем состояния>.

4.6 *Ввод "звездочка"*

Конкретная текстовая грамматика

<входной список "звездочка"> ::=
<звездочка>

<состояние> может содержать не более одного <входного списка "звездочка">. <состояние> не должно содержать одновременно и <входного списка "звездочка">, и <списка сохранения "звездочка">.

Модель

<входной список "звездочка"> преобразуется в список <стимулов>, содержащий полную совокупность допустимых входных сигналов охватывающего <определения процесса> или <определения сервиса>, за исключением <идентификаторов сигналов> неявных сигналов и <идентификаторов сигналов>, входящих в другие <входные списки> и <списки сохранения> данного <состояния> и во все <приоритетные вводы> <определения сервиса> § 4.10.

4.7 *Сохранение "звездочка"*

Конкретная текстовая грамматика

<список сохранений "звездочка"> ::=
<звездочка>

<состояние> может содержать не более одного <списка состояний "звездочка">. <состояние> не должно содержать одновременно и <входной список "звездочка"> и <список сохранения "звездочка">.

Модель

<список сохранения "звездочка"> преобразуется в список <стимулов>, содержащий полный список допустимых входных сигналов охватывающего <определения процесса> или <определения сервиса>, за исключением <идентификаторов сигналов> неявных сигналов и <идентификаторов сигналов>, входящих в другие <списки ввода> и <списки сохранения> данного <состояния> и во все <приоритетные вводы> <определения сервиса> § 4.10.

4.8 *Неявный переход*

Конкретная текстовая грамматика

<идентификатор сигнала>, содержащийся в совокупности всех допустимых входных сигналов <определения процесса> или <определения сервиса>, может быть опущен из совокупности <идентификаторов сигнала>, входящих во <входной список>, <приоритетный входной список> или <список сохранений> <состояния>.

Модель

Для каждого <состояния> имеется неявный <раздел ввода>, содержащий <переход>, который состоит из одного лишь <следующего состояния>, ведущего обратно в то же самое <состояние>.

4.9 *Следующее состояние "черточка"*

Конкретная текстовая грамматика

<следующее состояние "черточка"> ::=
<дефис>
-

<дефис> ::=

<переход>, содержащийся в <старте>, не должен приводить, непосредственно или косвенно, к <следующему состоянию "черточка">.

В каждом <следующем состоянии> некоторого <состояния> <следующее состояние "черточка"> заменяется на <имя состояния> данного <состояния>.

4.10 *Сервис*

Поведение процесса описывается в базисном SDL с помощью графа процесса. Понятие сервиса представляет альтернативу графу процесса в виде совокупности определений сервиса. Во многих случаях определение сервиса может понизить общую сложность системы и повысить обозримость определения процесса. Кроме того, каждое из определений сервиса может охватывать некоторую часть поведения процесса, что бывает полезным в некоторых приложениях.

4.10.1 *Разложение на сервисы*

Конкретная текстовая грамматика

<разложение на сервисы> ::=
{ <определение маршрута сигнала сервиса>
| <соединение маршрута сигнала>
| <определение сервиса>
| <определение отбора>
| <текстовая ссылка на сервис>} +

<определение маршрута сигнала сервиса> ::=
SIGNALROUTE <имя маршрута сигнала сервиса>
<путь маршрута сигнала сервиса>
[<путь маршрута сигнала сервиса>]

<путь маршрута сигнала сигнала сервиса> ::=
{ FROM <идентификатор сервиса> TO <идентификатор сервиса>
| FROM <идентификатор сервиса> TO ENV
| FROM ENV TO <идентификатор сервиса>}
WITH <список сигналов> <конец>

<соединение маршрута сигнала> ::=
CONNECT <идентификатор маршрута сигнала>
AND <идентификатор маршрута сигнала сервиса> { , <идентификатор маршрута сигнала сервиса>}*
<конец>

<текстовая ссылка на сервис> ::=
SERVICE <имя сервиса> REFERENCED <конец>

<определение процесса>, содержащее <разложение на сервисы>, не должно содержать <определение таймера> вне <разложения на сервисы>.

<разложение на сервисы> должно содержать хоть одно <определение сервиса>.

К <маршруту сигнала сервиса> приложимы правила корректности, аналогичные таковым для <маршрута сигнала>.

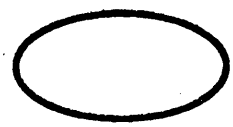
Конкретная графическая грамматика

<область взаимодействия сервисов> ::=
{<область сервиса> | <область определения маршрута сигнала сервиса>}+

<область сервиса> ::=
<графическая ссылка на сервис>
| <диаграмма сервиса>

<графическая ссылка на сервис> ::=
<символ сервиса> **contains** <имя сервиса>

<символ сервиса> ::=



<область определения маршрута сигнала сервиса> ::=
<символ маршрута сигнала>
is associated with {<имя маршрута сигнала сервиса>
[<идентификатор маршрута сигнала>]
<область списка сигналов>
[<область списка сигналов>]}**set**
is connected to <область сервиса>
{<область сервиса> | <символ кадра>} **set**

Если <символ маршрута сигнала> присоединен к <символу кадра>, то это значит, что <идентификатор маршрута сигнала> идентифицирует внешний маршрут сигнала, к которому подсоединен данный маршрут сигнала.

Семантика

<разложение на сервисы> является альтернативой <телу процесса> и выражает то же самое поведение.

Модель

Понятие сервиса моделируется преобразованием <разложения на сервисы> к первичным понятиям. В результате преобразования <определения маршрута сигнала сервиса> и <соединения маршрута сигнала> ничего не возникает.

4.10.2 *Определение сервиса*

Конкретная текстовая грамматика

<определение сервиса> ::=
SERVICE <имя сервиса> | <идентификатор сервиса>} <конец>
[<совокупность допустимых входных сигналов>]
{<определение переменной>
| <определение данных>
| <определение таймера>
| <определение обозревания>
| <определение импорта>
| <определение отбора>
| <макроопределение>
| <определение процедуры>
| <текстовая ссылка на процедуру>}*
<тело сервиса>
ENDSERVICE [{<имя сервиса> | <идентификатор сервиса>}] <конец>

<тело сервиса> ::=
<тело процесса>

<приоритетный ввод> ::=
PRIORITY INPUT <приоритетный входной список> <конец> <переход>

<приоритетный входной список> ::=
<приоритетный стимул>{, <приоритетный стимул>}*

<приоритетный стимул> ::=
<идентификатор приоритетного сигнала> [([<идентификатор переменной>] { , [<идентификатор переменной>]}*)]

<приоритетный вывод> ::=
PRIORITY OUTPUT <тело приоритетного вывода>

<тело приоритетного вывода> ::=
<идентификатор приоритетного сигнала> [<фактические параметры>]
{, <идентификатор приоритетного сигнала> [<фактические параметры>]}*

Сигнал является высокоприоритетным сигналом процесса в том и только в том случае, если он указан в <приоритетном вводе> <определения сервиса> этого процесса.

<определение переменной> в <определении сервиса> не должно содержать ключевых слов EXPORTED или REVEALED.

<идентификатор приоритетного сигнала>, входящий в <приоритетный вывод>, не должен входить в <раздел ввода> или в <раздел сохранения>. <идентификатор приоритетного сигнала>, входящий в <приоритетный ввод>, не должен входить в <вывод>.

Для совокупности допустимых входных сигналов и маршрутов сигнала сервиса имеют силу правила, аналогичные описанным в § 2.5.2 для процессов.

<определения маршрутов сигнала сервиса> может быть специфицировано в <разложении на сервисы> только в том случае, если охватывающее <определение блока> содержит <определения маршрутов сигналов>.

Только одно из <определений сервиса>, входящих в <разложение на сервисы>, может иметь <старт>, содержащий <цепочку перехода>. Все остальные <старты> должны содержать только <следующее состояние>.

Полные совокупности допустимых входных сигналов (каждая такая совокупность является объединением <совокупности допустимых входных сигналов> и совокупности сигналов, передаваемых по входящим <маршрутам сигналов сервиса> <определения сервиса>) <определений сервисов>, входящих в <определение процесса>, не должны содержать общие сигналы.

<определение процедуры> не должно иметь <состояний>, если охватывающее <определение процесса> содержит <определение сервиса>. <определение процедуры>, видимое более чем одним сервисом, не должно содержать конструкции VIA.

Совокупности приоритетов, ассоциированных с <непрерывными сигналами> в различных <определениях сервиса> одного и того же <разложения на сервисы>, не должны пересекаться.

К <соединению маршрута сигнала> приложимы те же правила корректности, что и к <соединению канала с маршрутом>.

Если охватывающее <разложение на сервисы> содержит какое-либо <определение маршрута сигнала сервиса>, то для каждого <идентификатора маршрута сигнала>, входящего в <вывод>, должен существовать маршрут сигнала сервиса, начинающийся от охватывающего сервиса и присоединенного к маршруту сигнала; этот маршрут сигнала должен быть в состоянии передавать сигналы, обозначенные <идентификаторами сигналов>, содержащимися в <выводе>.

Если <вывод> содержит конструкции VIA, то должен существовать по меньшей мере один путь связи (либо неявный по отношению к сервису, владеющему этим <выводом>, либо через (возможно неявные) маршруты сигналов сервиса и, возможно, маршруты сигналов и каналы), начинающиеся от сервиса, который в состоянии передавать сигналы, обозначенные <идентификаторами сигналов>, содержащимися в <выводе>.

Для каждого <приоритетного вывода> должен существовать по меньшей мере один путь связи (либо неявный по отношению к сервису, владеющему этим <приоритетным выводом>, либо через (возможно неявные) маршруты сигналов сервиса), начинающиеся от сервиса, который способен передавать сигналы, обозначенные <идентификатором приоритетных сигналов>, содержащимися в <приоритетном выводе>.

<приоритетный ввод> допустим только в <теле сервиса>. <приоритетный вывод> допустим только в <теле сервиса> и в <теле процедуры>.

Конкретная графическая грамматика

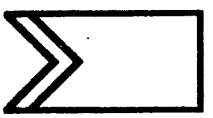
```
<диаграмма сервиса> ::=
        <символ кадра> contains
        {<заголовок сервиса>
            {{<область текста сервиса>}*
             {<графическая ссылка на процедуру>}*
             {<диаграмма процедуры>}*
             {<диаграмма макроса>}*
             <область графа сервиса>}set}

<заголовок сервиса> ::=
        SERVICE  <имя сервиса> | <идентификатор сервиса>

<область текста сервиса> ::=
        <символ текста> contains
        {<определение переменной>
            | <определение данных>
            | <определение таймера>
            | <определение обозревания>
            | <определение импорта>
            | <определение отбора>
            | <макроопределение>}*

<область графа сервиса> ::=
```

<область графа процесса>

<область ассоциирования приоритетного ввода> ::=
<символ сплошной линии ассоциации> **is connected to** <область приоритетного ввода>

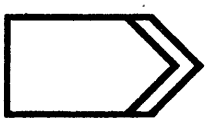
<область приоритетного ввода> ::=
<символ приоритетного ввода> **contains** <приоритетный входной список>
is followed by <область перехода>

<символ приоритетного ввода> ::=



<область приоритетного вывода> ::=
<символ приоритетного вывода> **contains** <тело приоритетного вывода>

<символ приоритетного вывода> ::=



Семантика

Свойства сервиса могут быть выведены из требования, согласно которому <разложение на сервисы>, заменяющее <тело процесса>, должно представлять то же самое поведение, что и <тело процесса>.

В каждом экземпляре процесса имеется по экземпляру сервиса для каждого <определения сервиса>, входящего в <определение процесса>. Экземпляры сервиса являются компонентами экземпляра процесса и поэтому с ними нельзя ничего делать (создавать, адресовать их или прекращать) как с независимыми объектами. Они используют входной порт и выражения SELF, PARENT, OFFSPRING и SENDER, принадлежащие экземпляру процесса.

Экземпляр сервиса является конечным автоматом, но он не может функционировать параллельно с другими экземплярами сервисов экземпляра процесса. Иными словами в каждый данный момент времени только один сервис, из принадлежащих некоторому экземпляру процесса, может совершать переход.

В <теле приоритетного вывода> неявно существует конструкция TO SELF. Приоритетные сигналы образуют специальный класс сигналов, имеющих более высокий приоритет, чем обычные сигналы. Эти сигналы могут пересылаться только между экземплярами сервисов, принадлежащих одному и тому же экземпляру процесса.

Входной сигнал, находящийся во входном порту, передается экземпляру того сервиса, который способен получить его.

Модель

а) *Преобразование определений*

Локальные определения, входящие в <определение сервиса> переносятся на уровень процесса заменой каждого вхождения в сервис некоторого имени на одно и то же новое имя, отличное от всех остальных. Все ссылки на сервисы, имеющиеся в квалификаторах, удаляются.
Определения обозревания или определения импорта, содержащие одни и те же обозреваемые или импортируемые переменные, сливаются в одно определение обозревания или импорта.

b) *Преобразование <тел сервисов>*

Совокупность <тел сервисов> преобразуется в <тело процесса>. Это можно осуществить несколькими альтернативными путями. Ниже описывается простейший вариант, поскольку основной целью является описание понятия сервиса непосредственно в конкретном синтаксисе. Из практических соображений <тело сервиса> и <тело процесса> рассматриваются как графы, составленные из состояний, цепочек переходов между состояниями, цепочек переходов "стоп" и одной стартовой цепочки перехода. Цепочка перехода однозначно определяется своим стартовым состоянием, вводом и конечным состоянием.

1) *Состояние*

Каждое состояние в результирующем граф процессе идентифицируется именем-n-кой. Размерность n-ки равна числу графов сервиса. Каждая из компонент n-ки уникально сопоставлена одному из исходных графов сервиса, при этом в качестве значения этой компоненты n-ки берется одно из сопоставленного ей графа сервиса. Таким образом, именами состояний графа процесса будут все n-ки, которые можно построить на основании сформулированного правила.

Пример:

Пусть заданы два графа сервиса и их состояния

f1: <a> <b>
f2: <A> <B> <C>,

тогда граф результирующего процесса содержит нижеследующие состояния

<a.A> <a.B> <a.C> <b.A> <b.B> <b.C>.

Этот "взрыв" числа состояний обычно может быть существенно сокращен, но здесь это не рассматривается.

2) *Цепочки переходов*

Каждая цепочка перехода графа сервиса копируется в граф процесса в одно или несколько мест. Она копируется так, чтобы соединить каждую пару состояний-n-ок, удовлетворяющую следующим требованиям:

— одна компонента начального состояния-n-ки ссылается на начальное состояние цепочки перехода;
— одна компонента заключительного состояния-n-ки ссылается на заключительное состояние цепочки перехода;
— остальные значения компонент пары должны совпадать для обоих состояний-n-ок.

Пример:

В вышеприведенном примере в графе f2 существовала цепочка перехода из <B> в <C>. В графе результирующего процесса эта цепочка будет связывать <a.B> с <a.C> и <b.B> с <b.C>. Это может быть выражено более сжато (используя сокращенные обозначения конкретного синтаксиса) в виде:

<*.B> преобразуется в <-.C>.

3) *Стартовые цепочки перехода*

Если один из графов сервиса содержит стартовую цепочку перехода, то эта цепочка перехода преобразуется в стартовую цепочку перехода графа процесса. Стартовая цепочка перехода графа процесса ведет к тому состоянию-n-ки, компоненты которой являются всеми начальными именами графов сервиса.

4) *Цепочки перехода "стоп"*

Любой переход, ведущий к <стопу>, копируется в графе процесса, в котором он присоединяется к каждому состоянию-n-ке, одна из компонент которой ссылается на стартовое состояние этого перехода.

5) *Приоритетные сигналы*

Приоритетные сигналы преобразуются следующим образом.

Каждое состояние результирующего графа процесса разлагается на два состояния. Приоритетные сигналы исходного состояния подсоединяются к первому состоянию, а все остальные сигналы — ко второму; в первом состоянии они включатся в число сохраняемых. Цепочка перехода, которая вела в исходное состояние, теперь ведет в первое состояние. К этой цепочке перехода добавляется нижеследующая цепочка действий:

— генерируется уникальное псевдозначение, которое присваивается неявной переменной SAME—TOKEN,
— выражению SELF посылается неявный сигнал X—CONT, несущий указанное псевдозначение.

К первому состоянию добавлен вход для сигнала X—CONT; за этим входом следует нижеследующая цепочка перехода:

принятие решения сравнивает полученное псевдозначение со значением SAME_TOKEN. Если значения совпадают, то выбирается путь, ведущий ко второму состоянию, а в противном случае — путь, ведущий снова в первое состояние.

Пример

Ниже приводится пример <определения процесса>, содержащего <разложение на сервисы>, равно как и соответствующие <определения сервисов>. Поведение этого процесса совпадает с поведением процесса, приведенного на рис. 2.9.9 в § 2.9

```
PROCESS Game;
        FPAR Player pid;
        SIGNAL Proberers (integer);
        DCL A integer;

        SIGNALROUTE IR1 FROM Game_handler TO ENV WITH Score,Gameid;
        SIGNALROUTE IR2 FROM Game_handler TO ENV WITH Subscr,Endsubscr;
        SIGNALROUTE IR3 FROM ENV TO Game_handler WITH Result,Endgame;
        SIGNALROUTE IR4 FROM ENV TO Bump_handler WITH Probe;
        SIGNALROUTE IR5 FROM ENV TO Bump_handler WITH Bump;
        SIGNALROUTE IR6 FROM Bump_handler TO ENV WITH Lose,Win;
        SIGNALROUTE IR7 FROM Bump_handler TO Game_handler WITH Proberers;

        CONNECT R5 AND IR5;
        CONNECT R2 AND IR3,IR4;
        CONNECT R3 AND IR1,IR6;
        CONNECT R4 AND IR2;

        SERVICE Game_handler REFERENCED;
        SERVICE Bump_handler REFERENCED;

ENDPROCESS Game;
```

```
SERVICE Game_handler;

    /*Этот сервис управляет игрой, выполняя действия: запуск игры,
    прекращение игры, подсчет очков и сообщения о счете*/

    DCL Count integer;
    /*Счетчик для подсчета очков*/

    START;
        OUTPUT Subscr;
        OUTPUT Gameid TO Player;
        TASK Count:=0;
        NEXTSTATE STARTED;
    STATE STARTED;
        PRIORITY INPUT Proberers(A);
            TASK Count:=Count+A;
            NEXTSTATE _;
        INPUT Result;
            OUTPUT Score(Count) TO Player;
            NEXTSTATE _;
        INPUT Endgame;
            OUTPUT Endsubscr;
            STOP;
    ENDSTATE STARTED;
ENDSERVICE Game_handler;


SERVICE Bump_handler;

    /*Этот сервис выполняет действия по регистрации сигнала
    bump и обработке сигнала probe, поступающих от игрока.
    Результат сигнала probe посылается игроку и сервису Game_handler*/

    START;
        NEXTSTATE EVEN;
    STATE EVEN;
        INPUT Probe;
            OUTPUT Lose TO Player;
            PRIORITY OUTPUT Proberers(-1);
            NEXTSTATE _;
        INPUT Bump;
            NEXTSTATE ODD;
    ENDSTATE EVEN;
    STATE ODD;
        INPUT Bump;
            NEXTSTATE EVEN;
        INPUT Probe;
            OUTPUT Win TO Player;
            PRIORITY OUTPUT Proberers(+1);
            NEXTSTATE _;
    ENDSTATE ODD;
ENDSERVICE Bump_handler;
```

Тот же пример на SDL/GR приведен в нижеследующих диаграммах:

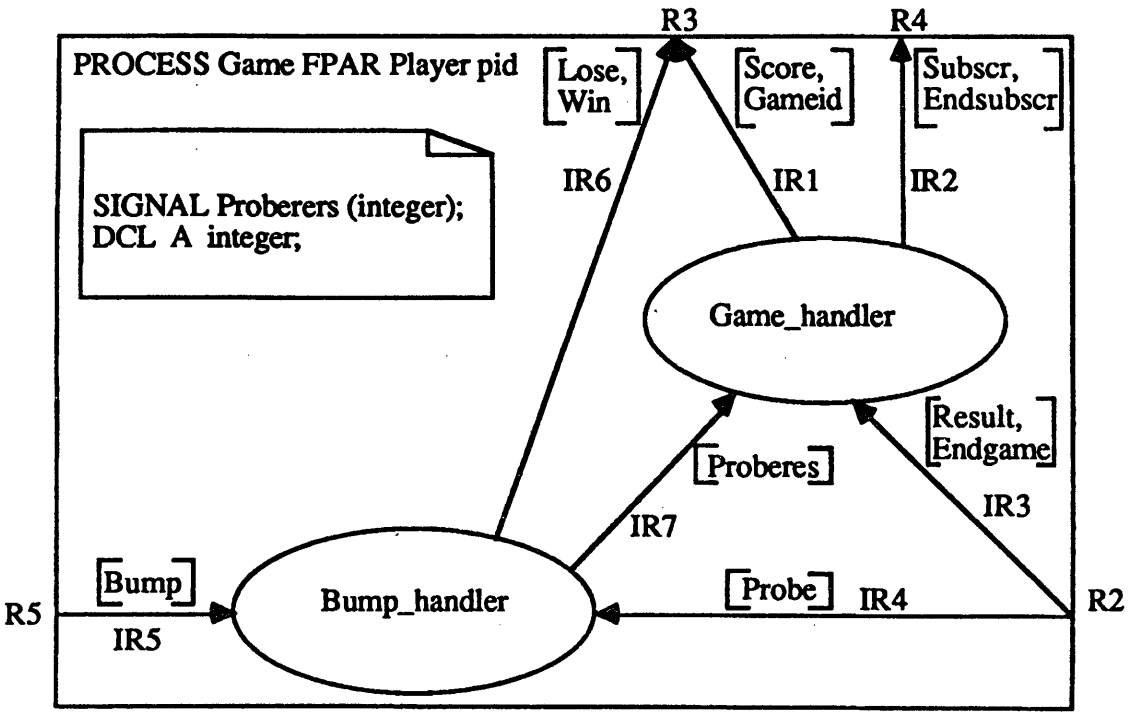


РИСУНОК 4.10.1

Пример диаграммы процесса, содержащей разложение на сервисы

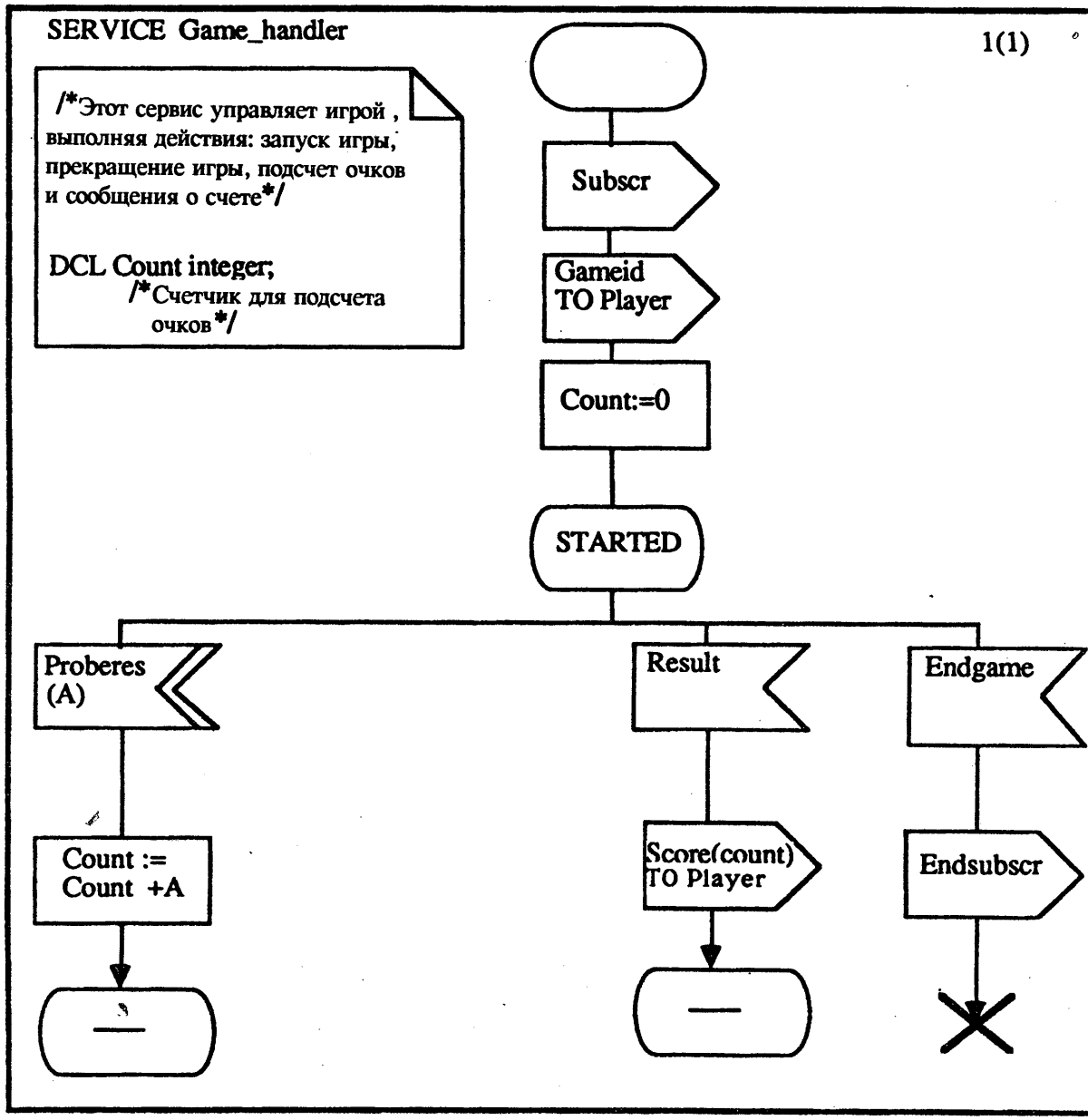


РИСУНОК 4.10.2

Пример диаграммы сервиса

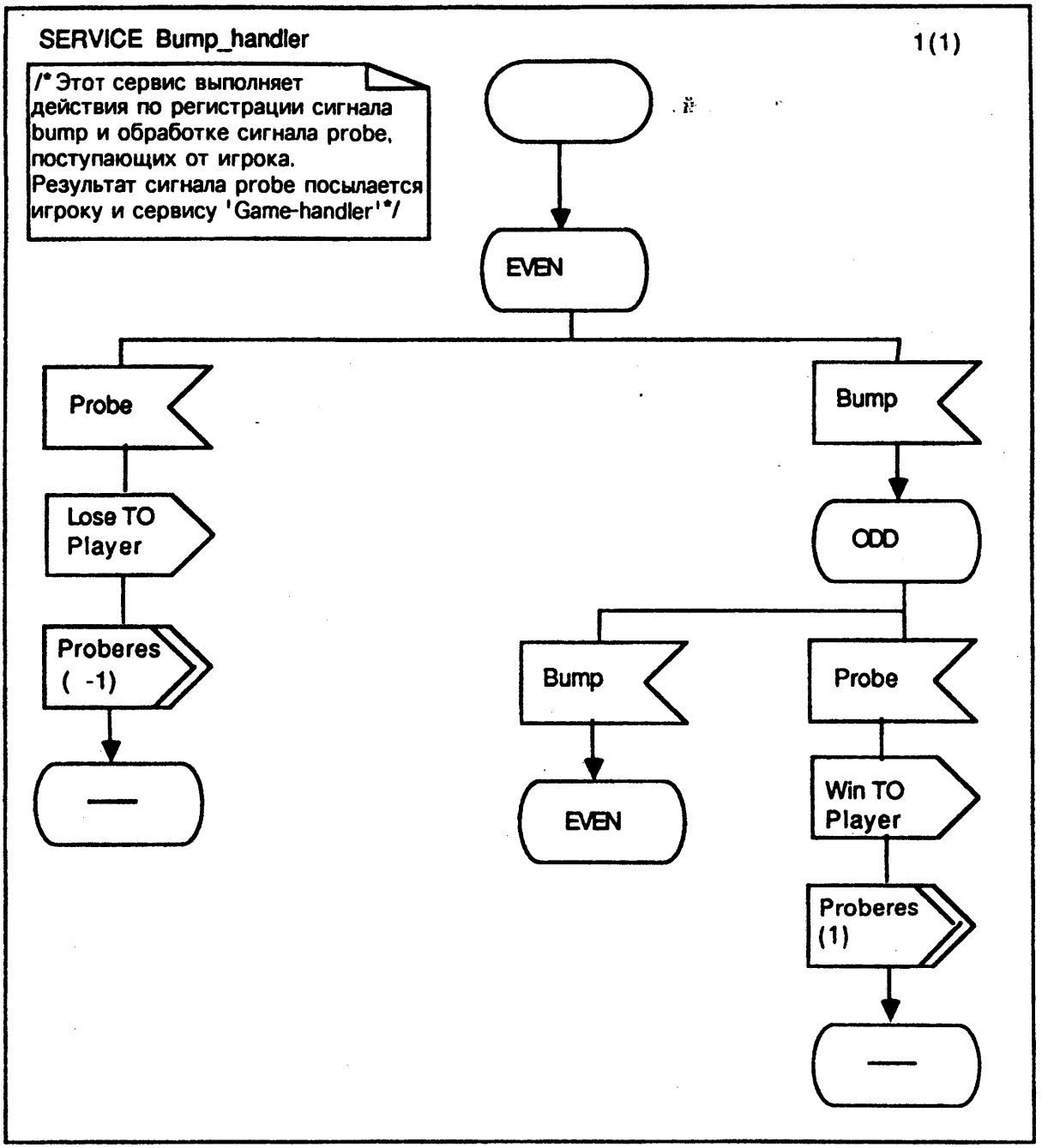


РИСУНОК 4.10.3

Пример диаграммы сервиса

Применение правил 1—4 преобразования сервиса приводит к графу процесса, изображенному на рис. 4.10.4. В нем приоритетные сигналы еще не преобразованы. Упрощая очевидным образом те переходы, которые содержат приоритетные сигналы, используя состояние "звездочка", можно получить тот же самый процесс, что и изображенный на рис. 2.9.10 в § 2.9. (Обратите внимание на то, что состояния EVEN и ODD соответствуют состояниям STARTED.EVEN и STARTED.ODD.)

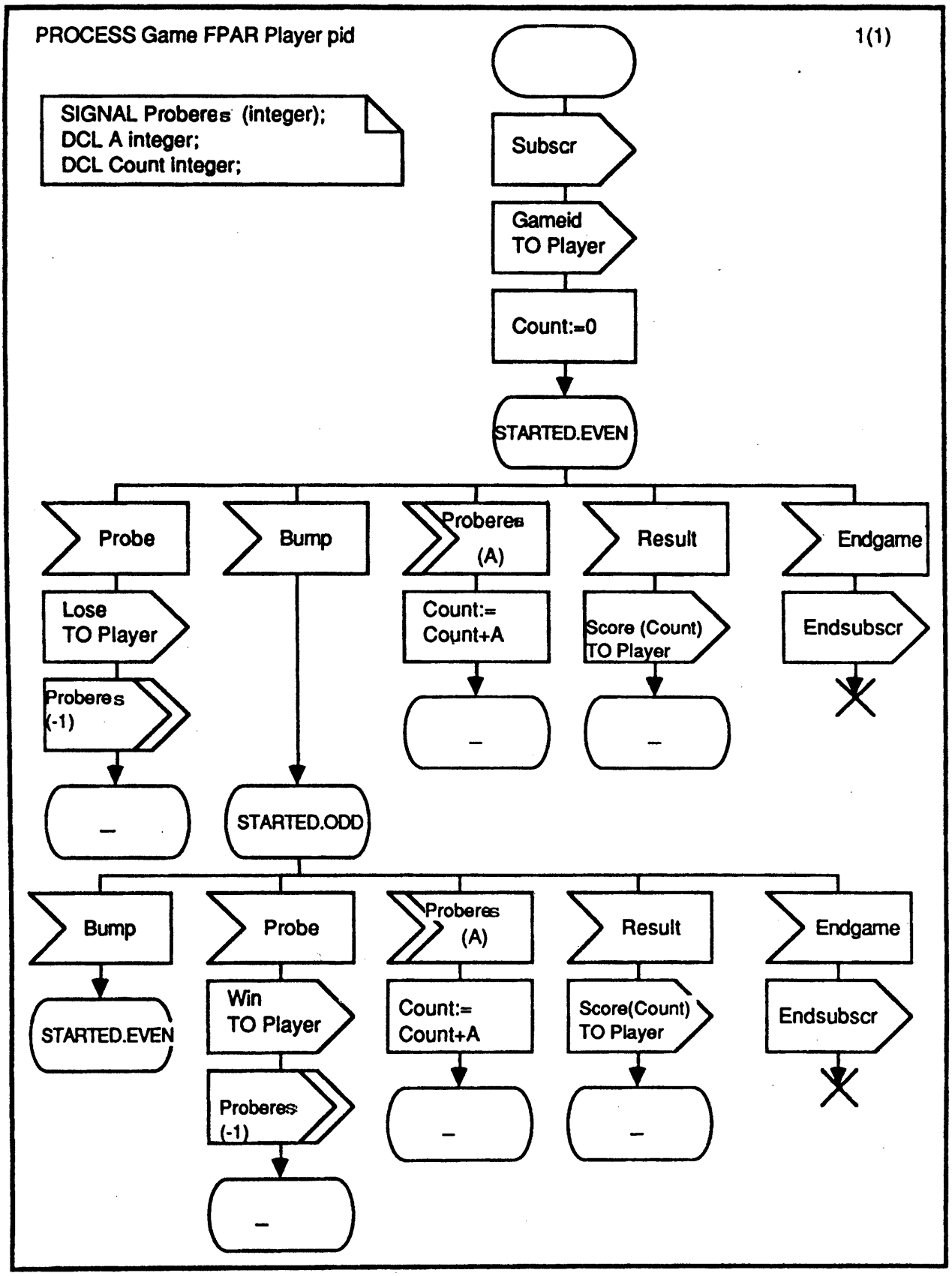


РИСУНОК 4.10.4 **Пример частичного преобразования**

4.11 *Непрерывный сигнал*

При описании системы с помощью SDL может возникнуть ситуация, при которой пользователь захочет показать, что переход возник как непосредственный результат того, что некоторое Булевское выражение имеет значение "истина". Для осуществления этого можно использовать модель, в которой значение выражения вычисляется в некотором состоянии и инициируется переход, если значение оказалось истинным. Соответствующая этой модели сокращенная форма называется Непрерывным сигналом, который позволяет инициировать переход непосредственно в результате выполнения некоторого условия.

Конкретная текстовая грамматика

<непрерывный сигнал> ::=
PROVIDED <булевское выражение> <конец>
[PRIORITY <имя целого литерала> <конец>] <переход>

Значения <имен целых литералов> в <непрерывных сигналах> <состояния> должны быть различными. Конструкция PRIORITY может быть опущена, если только <состояние> содержит равно один <непрерывный сигнал>.

Конкретная графическая грамматика

<область ассоциирования непрерывного сигнала> ::=
<символ сплошной линии ассоциации> **is connected to** <область непрерывного сигнала>

<область непрерывного сигнала> ::=
<символ разрешающего условия>
contains <булевское выражение> [[<конец>] PRIORITY <имя целого литерала>]}
is followed by <область перехода>

Семантика

<булевское выражение> в <непрерывном сигнале> вычисляется при входе в то состояние, с которым оно ассоциировано, при условии, что процесс в этот момент пребывает в ожидании, и при этом во входном порту нет ни одного стимула, входящего в ассоциированный <входной список>. Если значением <булевского выражения> является True, то инициируется переход. Если <булевское выражение> имеет значение True в более чем одном <непрерывном сигнале>, то инициируется переход, связанный с <непрерывным сигналом>, имеющим наивысший приоритет, то есть наименьшее значение <имени целого литерала>.

Модель

Если обозначить через state_name (имя_состояния) имя состояния, содержащего <непрерывные сигналы>, то это состояние преобразуется следующим образом. Вводятся две новые неявные переменные: n и newn (новое n), причем переменная n инициализируется к значению 0. Далее вводится неявный сигнал emptyQ (пустое Q), переносящий целые значения.

1) Все <следующие состояния>, которые указывают state_name заменяются на JOIN 1;

2) Вводится нижеследующий переход:

```
1: TASK n:=n+1;
   OUTPUT emptyQ(n) TO SELF,
   NEXTSTATE state_name;
```

3) Нижеследующий <раздел ввода> добавляется к <состоянию> state_name:

```
   INPUT emptyQ (newn);
(добавляется также <принятие решения>, содержащее <вопрос>:
                        (newn=n)
```

4a) Ветвь false <раздела ответа> содержит

NEXTSTATE state_name;

4b) Ветвь true <раздела ответа> содержит последовательность <принятий решений>, соответствующих <непрерывным сигналам> в приоритетном порядке (более высокий приоритет обозначен меньшим значением <имени целого литерала>).
Ветвь Faise <раздела ответа> содержит следующее <принятие решения> для всех <принятий решений>, кроме последнего, для которого эта ветвь содержит **JOIN** 1;
Каждая ветвь true <раздела ответа> этих <принятий решений> ведет к <переходу> соответствующего <непрерывного сигнала>.

Пример

См. § 4.12.

4.12 *Разрешающее условие*

В SDL поступление сигнала в момент, когда процесс находится в одном из своих состояний, вызывает немедленный переход. Понятие Разрешающего условия делает возможным наложение дополнительного условия на инициацию перехода.

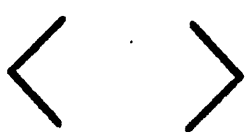
Конкретная текстовая грамматика

<разрешающее условие> ::=
PROVIDED <булевское выражение> <конец>

Конкретная графическая грамматика

<область разрешающего условия> ::=
<символ разрешающего условия> **contains** <булевское выражение>

<символ разрешающего условия> ::=



Семантика

<булевское выражение> в <разрешающем условии> вычисляется до входа в рассматриваемое <состояние> и это совершается каждый раз, когда процесс вновь входит в это состояние под влиянием <стимула>. Если разрешающих условий несколько, то они вычисляются в случайном порядке до входа в состояние. Модель преобразования обеспечивает эти повторные вычисления выражения посылкой через входной порт дополнительных <стимулов>. Сигнал, обозначенный во <входном списке> и активизирующий работу <разрешающего условия>, может запустить переход, только если значение<булевского выражения> равно True. Если же это значение равно False, то сигнал поступает на сохранение.

Модель

Если обозначить через state_name (имя_состояния) состояние, содержащее <разрешающее условие>, то это состояние преобразуется следующим образом. Вводятся две новые неявные переменные; n и newn (новое n), причем переменная n инициализируется к значению 0. Далее вводится неявный сигнал emptyQ (пустое Q), переносящий целые значения. Выполняются следующие преобразования.

1) Все <следующие состояния>, которые указывают state_name заменяются на JOIN 1.

2) Вводится нижеследующий переход:

```
1: TASK n: = n + 1;
   OUTPUT emptyQ (n) TO SELF;
```

К состоянию иерархически и в случайном порядке добавляется несколько принятий решений, каждое из которых содержит только одно <булевское выражение>, соответствующее некоторому <разрешающему условию>, сопоставленному данному состоянию; это добавление осуществляется так, чтобы возможно было вычислить все комбинации истиностных значений всех разрешающих условий, сопоставленных состоянию.
Каждая такая комбинация ведет к новому, отличному от всех прочих, состоянию.

3) Каждое из этих новых состояний имеет набор <разделов ввода>, состоящих из копий этих <разделов ввода> исходного состояния, не связанные с разрешающими условиями, плюс <разделы ввода>, для которых <булевские выражения> <разрешающих условий> получают значение True в этом состоянии. <стимулы> для оставшейся <части ввода> состоят из <списка сохранений> нового <раздела сохранения>, относящегося к этому состоянию. <раздел сохранения> исходного состояния также согласуется с этим новым состоянием.

4) К каждому новому состоянию добавляется:

```
INPUT emptyQ(newn);
```

и <принятие решения>, содержащее <вопрос> (newn = n);
Ветвь false <раздела ответа> содержит в качестве <следующего состояния> то же самое новое состояние.

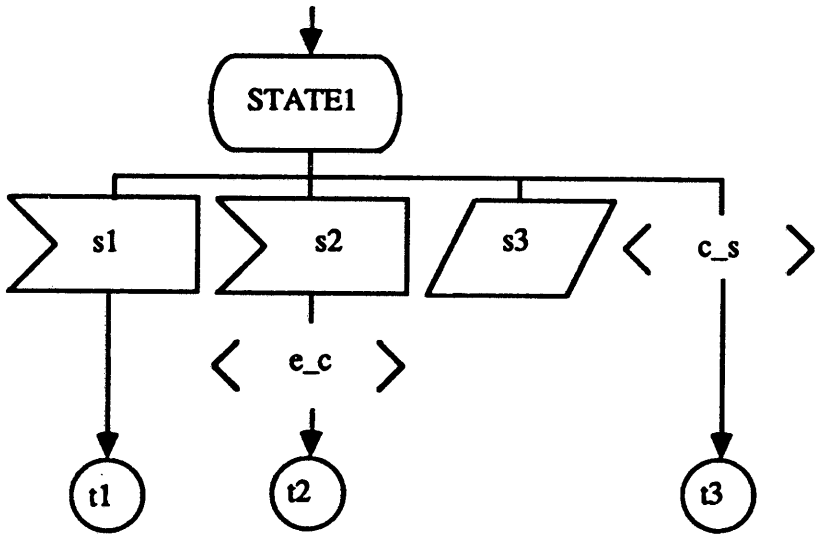
5) Ветвь true <раздела ответа> содержит JOIN 1.

6) Если в одном и том же <состоянии> используются как <непрерывные сигналы>, так и <разрешающие условия>, то вычисления <булевских выражений> из <непрерывных сигналов> осуществляются заменой шага 5 в модели для <разрешающего условия> на шаг 4b из модели для <непрерывного сигнала>.

Пример

Ниже приводится пример, иллюстрирующий преобразования разрешающего условия и непрерывного сигнала, содержащихся в некотором состоянии.

Следует отметить, что коннектор "ec" был введен для удобства. Это не является составной частью модели преобразования.



преобразуется в

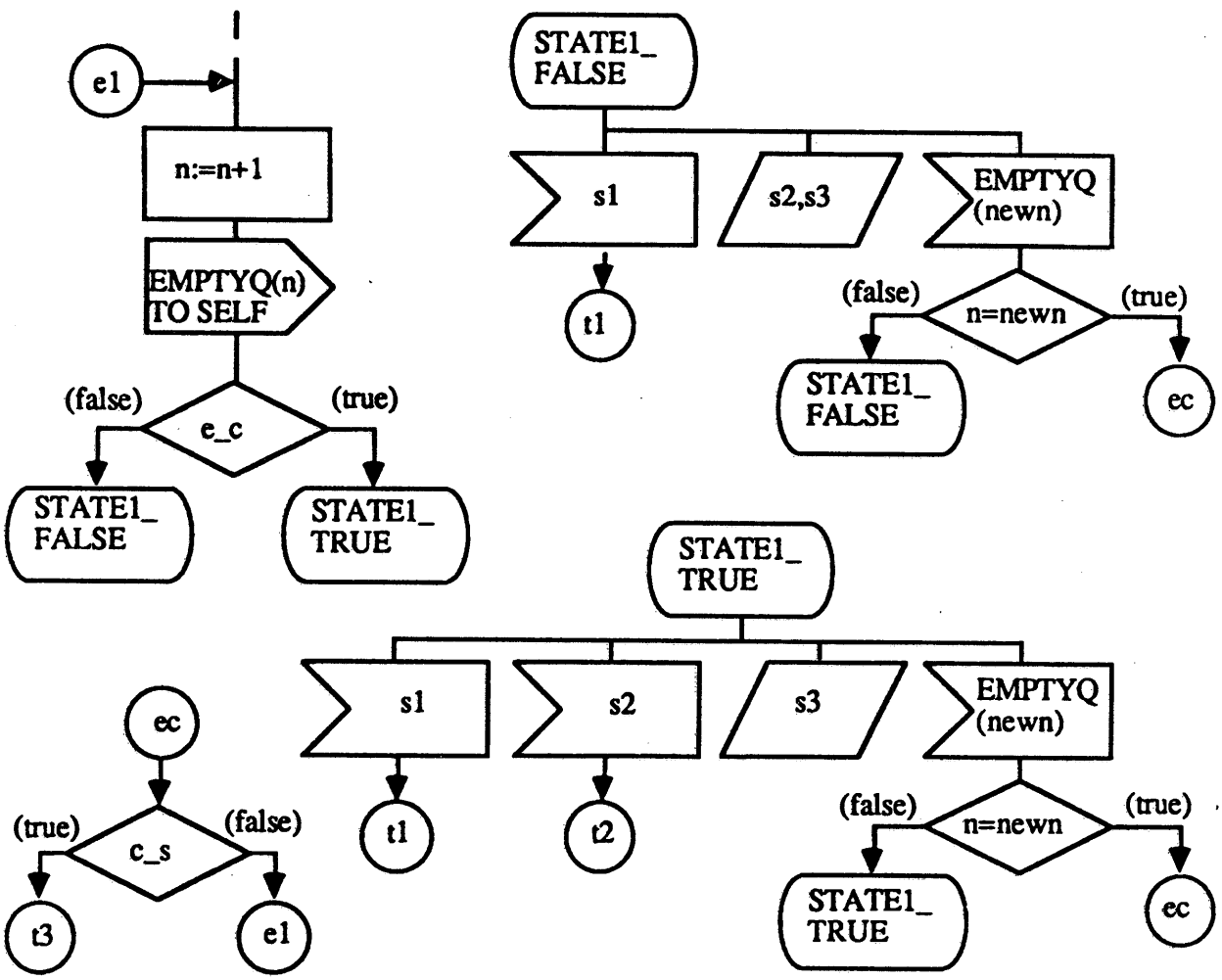


РИСУНОК 4.12.1

Преобразование непрерывного сигнала и разрешающего условия, содержащихся в одном и том же состоянии

4.13 *Импортируемое и экспортируемое значение*

В SDL любая переменная принадлежит некоторому экземпляру процесса: говорят, что она локальна в этом экземпляре процесса. Обычно переменная доступна только в том экземпляре процесса, которому она принадлежит. Однако она может быть объявлена совместно используемой (см. § 2), что позволяет экземплярам других процессов в том же блоке иметь доступ к значению этой переменной. Если значение переменной нужно экземпляру процесса в каком-нибудь другом блоке, то необходим обмен сигналами с экземпляром процесса, являющимся владельцем переменной.

Это можно осуществить с помощью нижеследующего сокращенного обозначения, называемого импортом и экспортом значений. Этот же сокращенный способ можно использовать для экспорта значений другим экземплярам процессов, находящимся в том же блоке. В таком случае этот метод является альтернативой совместному использованию значений.

Конкретная текстовая грамматика

<определение импорта> ::=
IMPORTED <имя импорта> { , <имя импорта> } * <сорт>
, <имя импорта> { , <имя импорта> } * <сорт> } * <конец>

<импортирующее выражение> ::=
IMPORT(<идентификатор импорта> [, <выражение PId>])

<экспорт> ::=
EXPORT (<идентификатор переменной> { , <идентификатор переменной> } *)

Конкретная графическая грамматика

<область экспорта> ::=
<символ работы> **contains** <экспорт>

Семантика

Экземпляр процесса, экспортирующий значения принадлежащей ему переменной другим экземплярам процессов, называется экспортером переменной. Другие экземпляры процесса, использующие эти значения, называются импортерами переменной. Сама переменная называется экспортируемой.

Некоторый экземпляр процесса может быть одновременно и экспортером и импортером, но нельзя импортировать из окружающей среды или импортировать ей.

a) *Экспортная операция*

Экспортируемые переменные в своем <определении переменной> содержат ключевые слова **EXPORTED**; они имеют неявные копии, используемые в импортных операциях.

Экспортной операцией является выполнение <экспорта>, с помощью которого экспортер сообщает ("раскрывает") другим экземплярам процессов текущее значение экспортируемой переменной. Экспортируемая операция вызывает помещение текущего значения экспортируемой переменной в ее неявную копию.

b) *Импортная операция*

Каждому <определению импорта>, содержащемуся в импортере, сопоставлен набор неявных переменных, имена и сорта которых совпадают с именем и сортом указанным в <определении импорта>. Эти неявные переменные служат для помещения в них импортируемых значений.

Импортная операция является выполнением <импортирующего выражения>, с помощью которой импорт получает доступ к значению экспортируемой переменной. Это значение помещается в неявную переменную, заданную <идентификатором импорта> в <импортирующем выражении>. Экспортер, содержащий экспортируемую переменную, задан <выражением PId> в <импортирующем выражении>. Если никакое <выражение PId> не задано, то должен существовать всего один экземпляр, экспортирующий переменную. Соответствие между экспортируемой переменной в экспортере и неявной переменной у импортера специфицируется совпадением <идентификаторов> в <экспорте> и <импортирующем выражении>. Кроме того, импортируемая переменная и неявная переменная должны быть одного и того же сорта.

Модель

Импортная операция моделируется обменом сигналами. Эти сигналы являются неявными и переносятся по неявным каналам и маршрутам сигналов. Импортер посылает сигнал экспортеру и ждет ответа. В ответ на этот сигнал экспортер посылает обратно сигнал импортеру; с этим сигналом посылается значение, содержащееся в неявной копии экспортируемой переменной.

Если с экспортируемой переменной связано присвоение по умолчанию или если экспортируемая переменная инициируется при ее определении, то тогда и неявная копия инициируется и с тем же значением, что и экспортируемая переменная.

Существует по два неявных <определения сигнала> для каждой комбинации <имени импорта> и <сорта>, содержащихся во всех <определениях импорта>, входящих в определение системы. <имена сигналов> в этих <определениях сигнала> обозначены через *xtQUERY* и соответственно через *xtREPLY*, где *x* обозначает <имя импорта>, а *t* — <сорт>. Неявная копия экспортируемой переменной обозначена через *imcx*.

a) *Импортер*

<импортирующее выражение> IMPORT (x, pidexp) преобразуется к следующему:

```
OUTPUT [xtQUERY TO pidexp;
Ожидание в состоянии xtWAIT с сохранением всех других сигналов;
INPUT xtREPLY (x);
Замена <импортирующего выражения> на x, (то есть на <имя> неявной переменной).
```

Если <импортирующее выражение> встречается в <выражении> более одного раза, то для каждого вхождения используется отдельная неявная переменная с тем же самым <именем>.

b) *Экспортер*

К каждому <состоянию> экспортера, включая его неявные состояния, добавляется следующая <часть ввода>:

```
INPUT [xtQUERY;
OUTPUT xtREPLY (imcx) TO SENDER;
/* следующее состояние является тем же самым*/
```

<Экспорт> EXPORT(x) заменяется на:

```
TASK imcx: =x;
```

5 Данные в SDL

5.1 *Введение*

Во введении описывается в общих чертах формальная модель, используемая для определения типов данных, и структура остальной части § 5.

Существенной особенностью языка спецификаций является возможность формального описания типов данных в терминах их поведения, в отличие от принятого в некоторых языках программирования способа построения этих типов из заранее заданных примитивов. Второй подход всегда предполагает конкретную реализацию типа данных и тем самым ограничивает свободу выбора подходящего представления типа данных. Подход, связанный с абстрактными типами данных, допускает любую реализацию при условии, что она осуществима и корректна относительно спецификации.

5.1.1 *Абстракция в типах данных*

Все данные, используемые в SDL, основываются на абстрактных типах данных, которые определяются скорее в терминах их абстрактных свойств, нежели в терминах какой-либо конкретной реализации. Примеры определения абстрактных типов данных даются в § 5.6, в котором определяется аппарат предварительно определенных данных языка.

Несмотря на то, что все типы данных в SDL абстрактные и что пользователь может пренебречь аппаратом предопределенных данных, в SDL сделана попытка обеспечить набор предопределенных данных, общепринятых по поведению и синтаксису. Следующие типы данных предопределены:

a) Булевский

b) Знаковый

c) Строка

d) Знакостроковый

e) Целый

f) Натуральный

g) Вещественный

h) Массив

i) Множественный тип

j) PId

k) Длительность

l) Временно́й

Концепция структурированного сорта (STRUCT) может использоваться для образования составных объектов.

5.1.2 *Общее описание формализмов, используемых при моделировании данных*

Данные моделируются посредством инициальной алгебры. В алгебре имеются заданные сорта и множество операций, отображающих одни сорта в другие. Каждый сорт является совокупностью всех возможных значений, которые могут быть порождены соответствующим множеством операций. Каждое значение может быть обозначено по крайней мере одним выражением, содержащим только литералы и операции (за исключением

особого случая PId-значений). Литералы являются частным случаем операций без аргументов.

Сорта и операции вместе с поведением типа данных (специфицируемых алгебраическими правилами), образуют свойства типа данных. Тип данных вводится рядом частичных определений типа, каждое из которых определяет сорт, ассоциированные с этим сортом операции и алгебраические правила.

Ключевым словом NEWTYPE вводится частичное определение типа, определяющее отдельный новый сорт. Сорт может быть создан со свойствами, наследуемыми у другого сорта, но с другими идентификаторами сорта и операций.

Введение подтипа состоит в выделении подмножества значений уже существующего сорта.

Генератор — это неполное описание NEWTYPE: для того, чтобы приобрести статус сорта, генератор должен быть конкретизирован посредством добавления недостающей информации.

Одни операции отображают тот же сорт, создавая таким образом (возможно, новые) значения сорта. Другие операции придают смысл сорту, отображая его в другие, уже определенные сорта. Многие операции отображают различные сорта в Булевский сорт, но эти операции не должны ни в коем случае расширять Булевский сорт.

Под пассивной операцией в SDL понимается обычное функциональное соответствие, которое не может воздействовать на значения переменных, являющихся ее параметрами. В SDL также определяется присваивание, которое может изменять значения, ассоциированные с переменными.

5.1.3 *Терминология*

Терминология, используемая в § 5 или в модели данных, выбрана так, чтобы гармонировать с опубликованной работой по инициальным алгебрам. В частности, "тип данных" используется для обозначения совокупности сортов, совокупности операций, ассоциированных с этими сортами, и определения свойств этих сортов и операций алгебраическими равенствами. "Сорт" — это множество значений, обладающих общими характеристиками. "Операция" — это отношение между сортами. "Равенство" — это определение эквивалентности термов данного сорта. "Значение" — это совокупность эквивалентных термов. "Аксиома" — это равенство, определяющее Булевское выражение, которое должно быть эквивалентно Истине (True). Однако "аксиомы" используются как термы в "аксиомах" и "равенствах", а "равенство" может быть "аксиомой".

5.1.4 *Строение текста о данных*

Модель инициальной алгебры, используемая для данных в SDL, описана так, что большинство понятий, относящихся к данным, может быть определено в терминах ядра языка данных абстрактных данных SDL.

Текст § 5 разбит на настоящее введение (§ 5.1), ядро языка данных (§ 5.2), модель инициальной алгебры (§ 5.3), пассивное использование данных (§ 5.4), активное использование данных (§ 5.5) и предопределенные данные (§ 5.6).

Ядро языка данных определяет часть данных в SDL, непосредственно соответствующую формализму инициальной алгебры.

Текст по инициальной алгебре является более детальным введением в математические основы данного формализма. Более строгая математическая формулировка содержится в приложении I.

Пассивное использование SDL включает возможности неявного и сокращенного описания данных SDL, что позволяет использовать его для определения абстрактных типов данных. Оно включает в себя также интерпретацию выражений, не содержащих значений, присвоенных переменным. Такие "пассивные" выражения отвечают в языке использованию функций.

Активное использование данных состоит в расширении языка за счет использования присваивания. В него входят использование присваивания и инициализация переменных. Говорят, что SDL используется активно, если в нем осуществляется присваивание значений переменным или доступ к значениям переменных. Разница между активным и пассивным выражениями состоит в том, что значение пассивного выражения не зависит от того, когда это выражение было проинтерпретировано, в то время как значение, полученное при интерпретации активного выражения, зависит от текущих значений переменных или текущего состояния системы.

Последней из тем являются предопределенные данные.

5.2 *Ядро языка данных*

Ядро языка данных может использоваться для определения абстрактных типов данных.

Более удобные конструкции для определения типов данных могут быть определены в терминах конструкций, определенных в ядре данных, за исключением случаев, когда требуется присваивание переменной. (Понятие ошибки и подтипа могут быть определены в терминах ядра, однако в § 5.4.1.7 и § 5.4.1.9 используются альтернативные, более краткие определения этих понятий.)

5.2.1 *Определения типов данных*

В каждой точке SDL-спецификации применимо некоторое определение типа данных. Определение типа данных определяет законность выражений и соотношения между ними. Это определение вводит операции и множества значений (сорта).

Между конкретным и абстрактным синтаксисами определений типов данных нет простого соответствия, поскольку в конкретном синтаксисе определение типа данных вводится постепенно, и делается акцент на понятии сорта (см. также § 5.3).

Определения в конкретном синтаксисе часто взаимозависимы и не могут быть разнесены в разные структурные единицы. Например,

```
NEWTYPE even LITERALS 0;
    OPERATORS   plusee      : even, even      ->even,
                plusoo      : odd, odd        ->even;
    AXIOMS      plusee (a,0)      == a;
                plusee (a, b)     == plusee (b, a);
                plusoo (a, b)     == plusoo (b, a);
    ENDNEWTYPE even COMMENT 'четные "числа" с плюсом—зависят от нечетных;
NEWTYPE odd LITERALS 1;
    OPERATORS   plusoe      : odd, even       ->odd;
                pluseo      : even, odd       ->odd;
    AXIOMS      plusoe (a, 0)     == a;
                pluseo (a, b)     == plusoe (b, a);
    ENDNEWTYPE odd; /* нечетные "числа" с плюсом — зависят от четных*/
```

Каждое определение типа данных полно; недопустимы ссылки на сорта и операции, не входящие в определение типа данных, применимое в данной точке. Определение типа данных не должно также нарушать семантику определения типа данных в непосредственно объемлющей структурной единице. Тип данных в объемлемой структурной единице может лишь пополнить множество операций сортов, определенных в объемлющей структурной единице. Значение сорта, определенного в данной структурной единице, может быть свободно использовано, либо получено из иерархически более низкой структурной единицы, либо передано из одной такой единицы в другую. Поскольку предопределенные данные определены на уровне системы, предопределенные сорта (например, Булевский или Целый) могут использоваться в любом месте системы.

Абстрактная грамматика

Определение-типа-данных	::	*Имя-типа* *Объединение-типов* *Сорта* *Сигнатура-set* *Равенства*
Объединение-типов	=	*Идентификатор-типа-set*
Идентификатор-типа	=	*Идентификатор*
Сорта	=	*Имя-сорта-set*
Имя-типа	=	*Имя*
Имя-сорта	=	*Имя*
Равенства	=	*Равенство-set*

В определении типа данных каждому *Сорту* должна соответствовать по крайней мере одна *Сигнатура* с *Результатом* (см. § 5.2.2), который совпадает с данным *Сортом.*

Определение типа данных не должно добавлять новые значения ни одному из *сортов* типа данных, идентификатор которого содержится в *объединении типов.*

Если один *терм* (см. § 5.2.3) не эквивалентен другому *терму* в соответствии с типом данных, идентификатор которого содержится в *объединении типов* из *определения типа данных,* то эти *термы* не должны определяться как эквивалентные в *определении типа данных.*

Кроме того, два Булевских *терма* Истина и Ложь не должны прямо или косвенно определяться как эквивалентные (см. § 5.4.3.1). Также запрещается сокращать число значений предопределенного сорта PId.

Примечание. — В абстрактном синтаксисе в *объединении типов* допускается более одного идентификатора типа для согласованности с используемым в лежащей в основе модели более общим классом алгебр, а SDL ссылается только на один тип, поскольку в конкретном синтаксисе доступный тип данных неявно определяется объемлющим <классом структурной единицы>; поэтому <объединение типов> упоминается только в абстрактном синтаксисе и является либо <*идентификатором типа*> объемлющей структурной единицы, либо пустым множеством в случае <определения системы>.

Конкретная текстовая грамматика

<частичное определение типа> :: =
NEWTYPE <имя сорта> [<расширенные свойства>] <выражение свойств>
ENDNEWTYPE [<имя сорта>]

<выражение свойств> :: =
<операции> [AXIOMS <аксиомы>] [<отображение литералов>] [<присваивание по умолчанию>.

Необязательные <расширенные свойства>, <отображение литералов> и <присваивание по умолчанию> не являются частью ядра данных и определяются в § 5.4.1, § 5.4.1.15 и § 5.5.3.3, соответственно.

Определение типа данных представляется объединением всех <частичных определений типа> в текущем <классе структурной единицы> вместе с *определением типа данных,* идентифицируемым *объединением типов* объемлющего <класса структурной единицы>. Имя типа из <определения типа данных> неявное и не имеет конкретного

синтаксического представления. *Идентификатор типа* из *объединения типов* неявно полагается совпадающим с *идентификатором определения типа данных* объемлющей структурной единицы.

Каждый из последующих <классов структурной единицы> (см. § 2.2.2) представляет элемент абстрактного синтаксиса, содержащий *определение типа данных:* <определение системы>, <определение блока>, <определение процесса>, <определение процедуры>, <определение подструктуры канала> или <определение подструктуры блока>, либо соответствующие диаграммы в графическом синтаксисе. <частичное определение типа> в <определении сервиса> представляет часть *определения типа данных* <определения процесса>, объемлющего данное <определение сервиса> (см. § 4.10).

Сорта <класса структурной единицы> представлены множеством <имен сортов>, введенных множеством <частичных определений типа> данного <класса структурной единицы>.

Множество *сигнатур* и *равенства* <класса структурной единицы> представлены <выражениями свойств> из <частичных определений типа> данного <класса структурной единицы>.

<операции> в <выражении свойств> представляют часть множества *сигнатур* в абстрактном синтаксисе. Полное множество *сигнатур* является объединением множеств *сигнатур,* определенных <частичными определениями типа> в <классе структурной единицы>.

<аксиомы> в <выражении свойств> представляют часть множества *равенств* в абстрактном синтаксисе. *Равенства* — это объединение множеств *равенств,* определенных <частичными определениями типа> в <классе структурной единицы>.

Предопределенные сорта данных имеют свои неявные <частичные определения типа> на уровне системы.

Если <имя сорта> приведено после ключевого слова **ENDNEWTYPE**, то оно должно совпадать с <именем сорта>, приведенным после ключевого слова **NEWTYPE**.

Семантика

Определение типа данных определяет тип данных. Тип данных имеет множество свойств типа, а именно: множество сортов, множество операций и множество равенств.

Свойства типов данных определяются в конкретном синтаксисе посредством частичных определений типа. Частичное определение типа не задает всех свойств типа данных, а лишь частично определяет некоторые из свойств, относящихся к сорту, вводимому этим частичным определением типа. Полный набор свойств типа данных может быть получен при рассмотрении всей совокупности частичных определений типа, применяемых в структурной единице, содержащей соответствующее определение типа данных.

Сорт — это совокупность значений данных. Два различных сорта не имеют общих значений.

Определение типа данных образовано определением типа данных структурной единицы, в которой определяется текущая структурная единица, вместе со всеми сортами, операциями и равенствами, определенными в текущей структурной единице. Определение системы содержит определение предопределенных сортов данных.

Определение типа данных, применяемое в данной точке, является типом данных, определенным для структурной единицы, непосредственно объемлющей данную точку, если только эта точка не лежит в <частичном определении типа>, <детализации сигнала> или <определении сервиса>. Определение типа данных, применяемое в <частичном определении типа> или <детализации сигнала>, является определением типа данных для структурной единицы, объемлющей соответственно данное <частичное определение типа> или <детализация сигнала>. В <определении сервиса> применяется *определение типа данных* <определения процесса>, объемлющего данное <определение сервиса> (см. § 4.10).

Множество сортов типа данных состоит из множества сортов, введенных в текущей структурной единице, вместе с множеством сортов типа данных, идентифицированного объединением типов. Множество операций типа дан-

ных состоит из множества операций, введенных в текущей структурной единице, вместе с множеством операций типа данных, идентифицированного объединением типов. Множество равенств типа данных состоит из множества равенств, введенных в текущей структурной единице, вместе с множеством равенств типа данных, идентифицированного объединением типов.

Каждый сорт, вводимый в определении типа данных, имеет идентификатор, являющийся именем, введенным частичным определением типа в структурной единице, которое квалифицировано идентификатором этой структурной единицы.

Тип данных имеет идентификатор, являющийся уникальным именем типа в абстрактном синтаксисе, которое квалифицировано идентификатором структурной единицы. В конкретном синтаксисе тип данных не имеет имени.

Пример

NEWTYPE telephone
/*операции и конструкции значений, определенных в других местах*/
ENDNEWTYPE telephone;

5.2.2 *Литералы и операции с параметрами*

Абстрактная грамматика

Сигнатура	=	*Сигнатура-литерала* | *Сигнатура-операции*
Сигнатура-литерала	::	*Имя-операции-литерала* *Результат*
Сигнатура-операции	::	*Имя-операции* *Список-аргументов* *Результат*
Список-аргументов	=	*Идентификатор-ссылки-на-сорт*$^+$
Результат	=	*Идентификатор-ссылки-на-сорт*
Идентификатор-ссылки-на-сорт	=	*Идентификатор-сорта* | *Идентификатор-подтипа*
Имя-операции-литерала	=	*Имя*
Имя-операции	=	*Имя*
Идентификатор-сорта	=	*Идентификатор*

Подтипы и *идентификаторы подтипов* не входят в ядро (см. § 5.4.1.9).

Конкретная текстовая грамматика

<операции>::=
[<список литералов>] [<список операций>]

<список литералов>::=
LITERALS <сигнатура литерала> { , <сигнатура литерала> *}[<конец>]

<сигнатура литерала> ::=
<имя операции литерала>
| <расширенное имя литерала>

<список операций> ::=
OPERATORS
<сигнатура операции> { <конец> <сигнатура операции> }* [<конец>]

<сигнатура операции> ::=
<имя операции>: <список аргументов> –> <результат>
| <упорядочение>

<имя операции> ::=
<имя операции>
| <расширенное имя операции>

<список аргументов> ::=
<сорт аргумента> { , <сорт аргумента> }*

<сорт аргумента> ::=
<расширенный сорт>

<результат> ::=
<расширенный сорт>

<расширенный сорт> ::=
<сорт>
| <генераторный сорт>

<сорт> ::=
<идентификатор сорта>
| <подтип>

Альтернативы <расширенное имя операции>, <расширенное имя литерала>, <упорядочение>, <генераторный сорт> и <подтип> не входят в ядро данных и определяются соответственно в § 5.4.1, § 5.4.1, § 5.4.1, § 5.4.1.8, § 5.4.1.12.1, § 5.4.1.12.1 и § 5.4.1.9.

Литералы вводятся <сигнатурами литералов>, приводимыми списком после ключевого слова LITERALS. *Результатом сигнатуры литерала* является сорт, введенный <частичным определением типа>, определяющим литерал.

Каждая <сигнатура операции> в списке <сигнатур операций>, приводимом после ключевого слова OPERATORS, представляет *сигнатуру операции* с *именем операции*, *списком аргументов* и *результатом*.

<имя операции> отвечает *имени операции* в абстрактном синтаксисе, которое уникально в определяющей его структурной единице, хотя имя может и не быть уникальным в конкретном синтаксисе.

Уникальное *Имя-операции* или *Имя-операции литерала* в абстрактном синтаксисе выводится из:

a) <имени операции> (или <имени операции-литерала>), плюс

b) списка идентификаторов сортов аргументов, плюс

c) идентификатора сорта результата, плюс

d) идентификатора сорта частичного определения типа, в котором определено <имя операции> (или <имя операции-литерала>).

В случае, когда специфирован <идентификатор операции>, уникальное *имя операции* в *идентификаторе операции* может быть выведено аналогичным способом вместе со списком сортов аргументов и сортов результата, выведенными из контекста. Если у двух операций одно и то же <имя>, но сорта хотя бы одного из аргументов либо результата различны, то у этих операций разные *имена*.

Каждый <сорт аргумента> в <списке аргументов> представляет *идентификатор ссылки на сорт* в *списке аргументов*. <результат> представляет *идентификатор ссылки* на *сорт результата*.

В случае, когда <квалификатор> <идентификатора операции> (или <идентификатора операции-литерала>) содержит <квалификационный шаг> с ключевым словом **TYPE**, <имя сорта> после этого ключевого слова не является частью *Квалификатора Идентификатора-операции* (или *Идентификатора-операции-литерала*), а используется при выводе уникального *Имени* в *Идентификаторе*. В этом случае *Квалификатор* формируется из списка <квалификационных шагов>, предшествующих ключевому слову **TYPE**.

Семантика

Операция "всюду определена", то есть применение операции к произвольному списку значений, имеющих сорта аргументов, приводит к значению, имеющему сорт результата.

Сигнатура операции определяет, как эта операция может использоваться в выражениях. Сигнатура операции состоит из идентификатора операции, списка сортов аргументов и сорта результата. По сигнатуре операции определяется, является ли выражение доступным выражением в языке в соответствии с правилами подбора сортов аргументных выражений.

Операция, не имеющая аргументов, называется литералом.

Операция-литерал представляет фиксированное значение, имеющее сорт результата этой операции.

У операции имеется результирующий сорт, который идентифицируется результатом.

Примечание. — Рекомендуется, чтобы в <сигнатуре операции> упоминался сорт, вводимый объемлющим <частичным определением типа>, как сорт <аргумента> или <результата>.

Пример 1

```
LITERALS free, busy;
```

Пример 2

```
OPERATORS
        findstate :  Telephone —> Availability;
```

Пример 3

```
LITERALS      empty_list
OPERATORS     add_to_list     : list_of_telephones, telephone —> list_of_telephones;
              sub_list        : list_of_telephones, telephone —> list_of_telephones
```

5.2.3 *Аксиомы*

Аксиомы определяют, какие термы представляют одно и то же значение. Из аксиом в определении типа данных определяются соотношения между значениями аргументов и результатов операций и, тем самым, опера-

циям придается смысл. Аксиомы задаются либо в виде Булевских аксиом, либо в виде равенств, устанавливающих алгебраическую эквивалентность.

Абстрактная грамматика

Равенство	=	*Неквантифицированное-равенство* | *Квантифицированные-равенства* | *Условное-равенство* | *Неформальный-текст*
Неквантифицированное-равенство	::	*Терм* *Терм*
Квантифицированные-равенства	::	*Имя-значения-set* *Идентификатор-сорта* *Равенства*
Имя-значения	=	*Имя*
Терм	=	*Пассивный-терм* | *Составной-терм* | *Терм-ошибка*
Составной-терм	::	*Идентификатор-значения* | *Идентификатор-операции Терм*$^{+}$ | *Условный-составной-терм*
Идентификатор-значения	=	*Идентификатор*
Идентификатор-операции	=	*Идентификатор*
Пассивный-терм	::	*Идентификатор-операции-литерала* | *Идентификатор-операции Пассивный-терм*$^{+}$ | *Условный-пассивный-терм*
Идентификатор-операции-литерала	=	*Идентификатор*

Альтернативы *Условный-составной-терм* и *Условный-пассивный-терм* в правилах *Составной-терм и Пассивный-терм* соответственно не входят в ядро данных, хотя равенства, содержащие эти термы, можно заменить на семантически эквивалентные равенства, написанные в терминах ядра языка (см. § 5.4.1.6). Альтернатива *терм-ошибка* в правиле *терм* не входит в ядро данных и определяется в § 5.4.1.7.

Определения *неформального текста* и *условных равенств* даны соответственно в § 2.2.3 и § 5.2.4.

Каждый *терм* (или *пассивный терм*) из списка термов, приведенного после *идентификатора операции*, должен иметь сорт, совпадающий с соответствующим ему (по месту в списке) сортом из *списка аргументов сигнатуры операции*.

Оба *терма* в *неквантифицированном равенстве* должны быть одного сорта.

Конкретная текстовая грамматика

<аксиомы> :: =
<равенство> { <конец> <равенство> } * [<конец>]

<равенство> :: =
<неквантифицированное равенство>
| <квантифицированные равенства>
| <условное равенство>
| <неформальный текст>

<квантифицированные равенства> :: =
<квантификация> (<аксиомы>)

<квантификация> :: =
FOR ALL <имя значения>{ , <имя значения>} * IN <расширенный сорт>

<неквантифицированное равенство> :: =
<терм> == <терм>
| <Булевская аксиома>

<терм> :: =
<пассивный терм>
| <составной терм>
| <терм-ошибка>
| <считывающий терм>

<составной терм> :: =
<идентификатор значения>
| <идентификатор операции> (<список составных термов>)
| (<составной терм>)
| <расширенный составной терм>

<список составных термов> :: =
<составной терм>{ , <терм>} *
| <терм>, <список составных термов>

<пассивный терм> :: =
<идентификатор литерала>
| <идентификатор операции> (<пассивный терм>{ , <пассивный терм>} *)
| (<пассивный терм>)
| <расширенный пассивный терм>

<идентификатор литерала> :: =
<идентификатор операции литерала>
| <расширенный идентификатор литерала>

Альтернативы <Булевская аксиома> в правиле <неквантифицированное равенство>, <терм-ошибка> и <считывающий терм> в правиле <терм>, <расширенный составной терм> в правиле <составной терм>, <расширенный пассивный терм> в правиле <пассивный терм> и <расширенный идентификатор литерала> в правиле <идентификатор литерала> не входят в ядро данных и определены соответственно в § 5.4.1.5, § 5.4.1.7, § 5.4.1.15, § 5.4.1, § 5.4.1 и § 5.4.1.

<сорт> в <квантификации> представляет *идентификатор сорта* в *квантифицированных равенствах.* <имена значений> в <квантификации> представляют множество *имен значений* в *квантифицированных равенствах.*

<список составных термов> представляет список *термов. Идентификатор операции,* за которым следует список *термов,* является только одним *составным термом,* если в списке *термов* содержится хотя бы один *идентификатор значения.*

<идентификатор>, являющийся неквалифицированным именем, встречающимся в <терме>, представляет

a) *идентификатор операции*, если он предшествует левой круглой скобке (либо он будет <именем операции>, являющимся <расширенным именем операции> — см. § 5.4.1), либо

b) *идентификатор значения*, если существует определение этого имени в <квантификации> <квантифицированных равенств>, объемлющих <терм> подходящего по контексту сорта, либо

c) *идентификатор операции-литерала*, если имеется доступный литерал с таким именем и подходящим по контексту сортом, либо

d) *идентификатор значения*, имеющий неявное *квантифицированное равенство* в абстрактном синтаксисе для <неквантифицированного равенства>.

Два и более вхождений в <равенство> одного и того же несвязанного <идентификатора значения> неявно подразумевают одну *квантификацию.*

Идентификатор операции выводится из контекста, так что если <имя операции> перегружено (то есть одно и то же <имя> используется для нескольких операций), то доступная операция с тем же именем и сортами аргументов и результата, согласованными с применением операции, будет идентифицирована *именем операции.* Если <имя операции> перегружено, то для того чтобы определить *имя операции,* возможно придется выводить сорта аргументов из аргументов, а сорт результата из контекста.

В пределах одного <неквантифицированного равенства> для каждого неявно квантифицированного идентификатора значения должен существовать ровно один сорт, согласованный со всеми использованиями этого идентификатора.

Требуется, чтобы каждый неквалифицированный <идентификатор операции> или <идентификатор операции-литерала> можно было связать ровно с одним определенным *идентификатором операции* или *идентификатором операции-литерала,* удовлетворяющим всем условиям конструкции, в которой используется этот <идентификатор>. То есть такое связывание должно быть единственным.

Примечание. — Рекомендуется, чтобы аксиома всегда имела отношение к сорту объемлющего определения типа в том смысле, что в ней должны упоминаться операция или литерал с результатом данного сорта, либо операция с аргументом данного сорта; аксиома должна определяться ровно один раз.

Семантика

Каждое равенство является утверждением об алгебраической эквивалентности термов. Термы в левой и правой частях равенства полагаются эквивалентными, так что всюду, где появляется один из термов, может быть подставлен другой. Если в равенстве встречается идентификатор значения, то можно подставить один и тот же терм одновременно во все вхождения этого идентификатора значения. При такой подстановке термом может быть любой пассивный терм, имеющий тот же сорт, что и идентификатор значения.

Идентификаторы значений вводятся в квантифицированных равенствах именами значений. Идентификатор значения используется для представления произвольного значения данных, принадлежащего сорту квантификации. Если вместо каждого вхождения идентификатора значения одновременно подставляется одно и то же значение, равенство должно выполняться независимо от того, какое значение выбрано.

Пассивный терм — это терм, не содержащий идентификаторов значений. Пассивный терм представляет конкретное, известное значение. Каждое значение сорта представлено по крайней мере одним пассивным термом, представляющим это значение.

Если в некоторых аксиомах содержится неформальный текст, то интерпретация выражений в SDL формально не определяется, однако может быть получена из неформального текста интерпретатором. Предполагается, что если специфицирован неформальный текст, то множество равенств неполно и тем самым полная формальная спецификация на SDL отсутствует.

Имя значения в абстрактном синтаксисе всегда вводится квантифицированными равенствами, и соответствующее значение имеет идентификатор значения, являющийся именем значения, квалифицированным идентификатором сорта объемлющих квантифицированных равенств. Например,

FOR ALL z, z IN X (FOR ALL z IN X ...))

вводит ровно один идентификатор значения с именем z сорта X.

В конкретном синтаксисе запрещается указывать квалификаторы для идентификаторов значений.

Каждый идентификатор значения, введенный квантифицированными равенствами, имеет сорт, который идентифицируется в квантифицированных равенствах *идентификатором ссылки на сорт*. Сорт в неявных квантификациях определяется по контексту вхождений несвязанного идентификатора. Если контекст, в котором встречается идентификатор значения, имеющий неявную квантификацию, допускает выбор различных сортов, то идентификатор связывают с сортом, согласованным со всеми использованиями этого идентификатора в равенстве.

Терм имеет сорт, являющийся сортом идентификатора значения или сортом результата операции (-литерала).

Разные литералы обозначают различные значения, если только из равенств нельзя вывести противного.

Пример 1

```
FOR ALL b IN logical ( eq(b, b)==T)
```

Пример 2

```
neq(T,F) == T;   neq(T, T) == F;
neq(F, T) == T;   neq(F, F) == F;
```

Пример 3

```
eq(b,   b)                  == T;
eq(F,   eq(T, F))           == T;
eq(eq(b,a),eq(a,b))         == T;
```

5.2.4 *Условные равенства*

Условные равенства позволяют специфицировать равенства, выполняющиеся только тогда, когда выполняются некоторые ограничения. Ограничения записываются в форме простых равенств.

Абстрактная грамматика

Условное-равенство	:	*Ограничение-set* *Ограничиваемое-равенство*
Ограничение	=	*Неквантифицированное-равенство*
Ограничиваемое-равенство	=	*Неквантифицированное-равенство*

Конкретная текстовая грамматика

```
<условное равенство> :: =
        <ограничение>{ , <ограничение>} * ==> <ограничиваемое равенство>

<ограничиваемое равенство> :: =
        <неквантифицированное равенство>
```

<ограничение> :: =
<неквантифицированное равенство>

Семантика

Ограничиваемое равенство определяет, что его термы обозначают одинаковые значения только при условии, что из других равенств может быть показано, что значения, обозначаемые идентификаторами значений в ограничиваемом равенстве, удовлетворяют ограничениям. Значение удовлетворяет ограничению только в том случае, если для этого значения ограничение может быть выведено из других равенств.

Семантика множества равенств типа данных, включающего условные равенства, выводится следующим образом:

a) От квантификации избавляются генерированием всех возможных равенств с пассивными термами, которые выводятся из квантифицированных равенств. После того как это проделано как с явной, так и неявной квантификацией, остается лишь множество неквантифицированных равенств с пассивными термами.

b) Назовем доказуемым условным равенством условное равенство, все ограничения которого (содержащие только пассивные термы) могут быть выведены из неквантифицированных равенств, не являющихся ограничиваемыми равенствами. Если доказуемые условные равенства существуют, их заменяют ограничиваемыми равенствами доказуемых условных равенств.

c) Если во множестве равенств остаются только такие условные равенства, ни одно из которых не является доказуемым условным равенством, то эти условные равенства нужно отбросить, а в противном случае вернуться к шагу (b).

d) Оставшееся множество неквантифицированных равенств определяет семантику типа данных.

Пример

z/0 == Истина ==> (x/z)*z == x

5.3 *Модель инициальной алгебры (неформальное описание)*

Определение данных в SDL основано на ядре данных, определенном в § 5.2. В дополнение к ранее приведенному определению, операциям и значениям следует придать дополнительное содержание, с тем чтобы можно было интерпретировать выражения. Например, выражения, используемые в непрерывных сигналах, разрешающих условиях, вызовах процедур, выходных действиях, запросах на создание, предложениях присваивания, предложениях установки и сброса, предложениях экспорта, предложениях импорта, принятиях решения и обозревании.

Необходимое дополнительное содержание придается выражениям с помощью формализма инициальной алгебры, излагаемого ниже в §§ 5.3.1 — 5.3.6[1].

В каждой точке SDL спецификации применим не только последний определенный по иерархии тип данных, но и видимо некоторое множество сортов. Это множество сортов будет объединением всех сортов, относящихся к иерархически более высоким уровням, как объясняется в § 5.2.

---

[1] По соглашению между МККТТ и ISO текст §§ 5.3.1 – 5.3.6 был принят в качестве общего неформального описания модели инициальной алгебры для абстрактных типов данных. Этот же текст (с соответствующими типографскими изменениями и изменениями в нумерации) приведен в приложении к ISO IS8807.

(В настоящем разделе символ = используется как символ эквивалентности в равенствах. В SDL в качестве символа эквивалентности в равенствах используется символ ==, с тем чтобы использовать символ = для операции равенства. Символ = использован в этом разделе как общепринятый символ из опубликованной работы по инициальным алгебрам.)

5.3.1 *Введение*

Смысл и интерпретация данных, основанные на инициальной алгебре, излагаются в три этапа:

a) Сигнатуры

b) Термы

c) Значения

5.3.1.1 *Представления*

Идея, что разные обозначения могут представлять одно и то же понятие, банальна. Например, общепринято, что арабскими цифрами (1, 2, 3, 4, ...) и римскими цифрами (I, II, III, IV, ...) представляется одно и то же множество чисел с одними и теми же свойствами. Другим примером являются префиксная нотация (plus(1, 1)), инфиксная нотация (1 + 1) и обратная польская нотация (1 1 +), используемые для представления одной и той же операции. Кроме того, разные пользователи могут использовать различные названия для одних и тех же понятий, (может быть потому, что они пользуются разными языками), поэтому, например, пары { true, false }, {T, F}, {0,1}, { vrai, faux } могут быть различными представлениями Булевского сорта.

Существенно же не конкретное представление, а абстрактные отношения между элементами. Так, для чисел нас интересуют соотношения между 1 и 2, и они будут теми же, что и соотношения между I и II. Аналогично содержательной информацией об операции являются соотношения между ее идентификатором и идентификаторами других операций, а также список аргументов. Конкретные конструкции, такие как скобки, позволяющие различать (a + b)*c и a + (b*c), интересны лишь постольку, поскольку позволяют определить лежащее в основе абстрактное понятие.

Эти абстрактные понятия входят в состав абстрактного синтаксиса понятия, который может быть реализован посредством различных конкретных синтаксисов. Так, следующие два конкретных примера описывают свойства одного и того же типа данных, но в различных конкретных синтаксисах.

```
NEWTYPE     bool LITERALS true, false;
OPERATORS   "not" :bool ->bool;
AXIOMS
        not(true)       == false;
        not(not(a))     == a;
ENDNEWTYPE bool;

NEWTYPE int LITERALS zero, one;
OPERATORS   plus        :int,int ->int;
            minus       :int,int ->int;
AXIOMS
        plus(zero,a)            == a;
        plus(a,b)               == plus(b,a);
        plus(a,plus(b,c))       == plus(plus(a,b),c);
        minus(a,a)              == zero;
        minus(a,zero)           == a;
        minus(a,minus(b,c))     == minus(plus(a,c),b);
        minus(minus(a,b),c)     == minus(a,plus(b,c));
        plus(minus(a,b),c)      == minus(plus(a,c),b);
ENDNEWTYPE int;

NEWTYPE tree LITERALS nil;
OPERATORS
        tip     : int       ->tree;
        isnil   : tree      ->bool;
        istip   : tree      ->bool;
        node    : tree,tree ->tree;
        sum     : tree      ->int;
AXIOMS
        istip(nil)              == false;
        istip(tip(i))           == true;
        istip(node(t1,t2))      == false;
        isnil(nil)              == true;
        isnil(tip(i))           == false;
        isnil(node(t1,t2))      == false;
        sum(node(t1,t2))        == plus(sum(t1),sum(t2));
        sum(tip(i))             == i;
        sum(nil)                == zero;
ENDNEWTYPE tree;
```

ПРИМЕР 1

```
TYPE      bool      IS
SORTS     bool
OPNS      true  :      -> bool
          false :      -> bool
          not   :bool-> bool
EQNS OFSORT      bool FOR ALL a:bool
      not(true)      = false;
      not(not(a))    = a
ENDTYPE

TYPE      int   IS   bool WITH
SORTS     int
OPNS      zero :               -> int
          one  :               -> int
          plus :    int,int    -> int
          minus:    int,int    -> int
EQNS OFSORT      int   FOR ALL a,b,c:int
      plus(zero,a)             = a ;
      plus(a,b)                = plus(b,a);
      plus(a,plus(b,c))        = plus(plus(a,b),c) ;
      minus(a,a)               = zero;
      minus(a,zero)            = a ;
      minus(a,minus(b,c))      = minus(plus(a,c),b);
      minus(minus(a,b),c)      = minus(a,plus(b,c));
      plus(minus(a,b),c)       = minus(plus(a,c),b)
ENDTYPE


TYPE      tree  IS   int   WITH
SORTS     tree
OPNS      nil   :            ->tree
          tip   : int        ->tree
          isnil : tree       ->bool
          istip : tree       ->bool
          node : tree,tree   ->tree
          sum  : tree        ->int
EQNS OFSORT      bool FOR ALL i:int, t1,t2:tree
          istip(nil)              = false;
          istip(tip(i))           = true ;
          istip(node(t1,t2))      = false;
          isnil(nil)              = true ;
          isnil(tip(i))           = false;
          isnil(node(t1,t2))      = false
      OFSORT      int   FOR ALL i:int, t1,t2:tree
          sum(node(t1,t2))        = plus(sum(t1),sum(t2));
          sum(tip(i))             = i ;
          sum(nil)                = zero
ENDTYPE
```

ПРИМЕР 2

Этот пример будет использован для иллюстрации. Первоначально будет рассмотрено определение сортов и литералов.

Следует отметить, что литералы рассматриваются как частный случай операций, а именно как операции, не имеющие параметров.

Мы можем ввести некоторые сорта и литералы первым способом:

```
NEWTYPE int LITERALS zero, one; ...
NEWTYPE bool LITERALS true, false; ...
NEWTYPE tree LITERALS nil; ...
```

или вторым способом:

```
...
SORTS bool
OPNS  true  :      -> bool
      false :      -> bool

...
SORTS int
OPNS  zero  :      -> int
      one   :      -> int

...
SORTS tree
OPNS  nil   :      -> tree

...
```

В дальнейшем будет использоваться только второй способ, как наиболее близкий к формулировке, используемой во многих публикациях по инициальной алгебре. Следует отметить, что в обоих случаях вид термов один и тот же, а наиболее значительной является разница в способах введения литералов. Нужно помнить, что конкретную нотацию необходимо принять для того, чтобы передавать понятия, но содержательный смысл алгебр от нотации не зависит, так что систематическое переименование (сохраняющее единственность) и переход от инфиксной нотации к польской не изменят содержания, определенного определениями типа.

5.3.2 *Сигнатуры*

С каждым сортом будут ассоциированы одна или несколько операций. У каждой операции есть функциональная запись, то есть она относит одному или нескольким входным сортам сорт результата.

Например, к вышеопределенным сортам могут быть добавлены следующие операции:

```
...
SORTS bool
OPNS  true  :          -> bool
      false :          -> bool
      not   : bool     -> bool

...
SORTS int
OPNS  zero  :                    -> int
      one   :                    -> int
      plus  :      int, int      -> int
      minus:       int, int      -> int

...
SORTS tree
OPNS  nil   :                    -> tree
      tip   :      int           -> tree
      isnil :      tree          -> bool
      istip :      tree          -> bool
      node  :      tree, tree    -> tree
      sum   :      tree          -> int

...
```

Применяемая сигнатура типа состоит из множества сортов и множества операций (как литералов, так и операций с параметрами), являющихся доступными.

Сигнатура типа называется полной (замкнутой), если для каждой операции, принадлежащей сигнатуре, сорта, входящие в функциональную запись операции, содержатся во множестве сортов данного типа.

5.3.3 *Термы и выражения*

Нас интересует язык, в котором допускаются выражения, являющиеся переменными, литералами или операциями, примененными к выражениям. Переменная — это объект данных, ассоциированный с некоторым выражением. Интерпретацию переменной можно заменить интерпретацией выражения, ассоциированного с переменной. Переменные могут быть таким образом устранены, так что интерпретация выражений сведется к применению различных операций к литералам.

Таким образом, при интерпретации открытое выражение (выражение, содержащее переменные) становится замкнутым выражением (выражением, не содержащим переменных), что происходит за счет подстановки в открытое выражение фактических аргументов (являющихся замкнутыми выражениями).

Замкнутое выражение соответствует пассивному терму.

Множество всех возможных пассивных термов данного сорта называется множеством пассивных термов сорта. Например, для определенного выше bool множество пассивных термов будет содержать

{ true, false, not(true), not(false), not(not(true)), ...}

Отсюда видно, что даже для такого простого сорта множество пассивных термов бесконечно.

5.3.3.1 *Генерация термов*

Если задана сигнатура типа, то для этого типа можно сгенерировать множество пассивных термов.

Множество литералов типа рассматривается как базисное множество пассивных термов. Каждый литерал имеет сорт, следовательно, каждый пассивный терм имеет сорт. Для типа, который был определен выше, таким базисным множеством пассивных термов будет

{ zero, one, true, false, nil}

Для каждой операции из множества операций типа пассивные термы генерируются подстановкой в аргументы операции всех ранее сгенерированных пассивных термов, сорта которых соответствуют сортам аргументов. Сорт результата каждой операции является сортом пассивного терма, сгенерированного этой операцией. Полученное множество пассивных термов добавляется к уже существующему множеству пассивных термов, с тем чтобы образовать новое множество пассивных термов. Для вышеприведенного типа таким образом будет получено

{zero,	one,	true,	false,	nil,
plus(zero,zero),	plus(one,one),	plus(zero,one),	plus(one,zero),	
minus(zero,zero),	minus(one,one),	minus(zero,one),	minus(one,zero),	
not(true),	not(false),	tip(zero),	tip(one),	
isnil(nil),	istip(nil),	node(nil,nil),	sum(nil) }	

Это новое множество пассивных термов берется затем вместо предыдущего множества пассивных термов и к нему применяется описанный алгоритм,с тем чтобы образовать очередное множество пассивных термов. Это множество пассивных термов будет включать

{ zero,	one,	true,	false,	nil,
plus(zero,zero),	plus(one,one),	plus(zero,one),	plus(one,zero),	...
plus(zero,plus(zero,zero)),		plus(zero,plus(one,one)),		...
plus(zero,sum(nil)),		...		
isnil(node(nil,nil)),		istip(node(nil,nil)),	node(nil,node(nil,nil)),	
...,			sum(node(nil,nil)) }	

Для того чтобы сгенерировать все возможные для данного типа пассивные термы, то есть множество пассивных термов типа, данный алгоритм применяется снова и снова. Множеством пассивных термов сорта является множество пассивных термов типа, имеющих данный сорт.

Обычно генерацию можно продолжать до бесконечности и получить бесконечное множество термов.

5.3.4 *Значения и алгебры*

Каждый терм данного сорта представляет значение этого сорта. Из вышеприведенного видно, что даже такой простой сорт как bool имеет бесконечное множество термов и, следовательно, бесконечное множество значений, если только не определено, какие термы считать эквивалентными (то есть представляющими одно и то же значение). Эквивалентность задается равенствами, определенными для термов. При отсутствии операций istip и isnil число значений сорта bool может быть ограничено двумя значениями с помощью равенств

```
not(true) = false;
not(false) = true
```

Такие равенства определяют эквивалентность термов, и при этом возможно получить два класса эквивалентных термов

```
{true,   not(false),   not(not(true)),   not(not(not(false))), ...}
{false,  not(true),    not(not(false)),  not(not(not(true))), ... }
```

При этом каждый класс эквивалентности представляет одно значение и элементы класса являются различными представлениями одного и того же значения.

Отметим, что если термы не определены как эквивалентные посредством равенства, то они не эквивалентны (то есть представляют различные значения).

В алгебре определяется множество термов, удовлетворяющих сигнатуре алгебры. Равенства в алгебре соотносят одни термы с другими.

В общем случае каждое значение сорта в алгебре будет представляться несколькими способами.

Алгебра является инициальной алгеброй для заданной сигнатуры в том и только в том случае, если любая другая алгебра, задающая те же свойства сигнатуры, может быть систематически преобразована в рассматриваемую инициальную алгебру. (Формально такое преобразование является гомоморфизмом.)

При условии, что операции not, istip и isnil всегда доставляют значения из классов эквивалентности true и false, инициальной алгеброй для bool будет пара литералов

```
{ true, false}
```

и она не будет содержать равенств.

5.3.4.1 *Равенства и квантификация*

Для сорта, подобного bool, число значений которого ограничено, все равенства могут быть записаны лишь с помощью пассивных термов, то есть термов, содержащих только литералы и операции.

Если же число значений сорта велико, непрактично выписывать все равенства, используя только пассивные термы, а для сортов с бесконечным числом значений (таких как целые числа), подобное явное перечисление становится невозможным. Техника написания квантифицированных равенств используется для представления возможно бесконечного числа равенств одним квантифицированным равенством.

В термах квантифицированного равенства содержатся идентификаторы значений. Такие термы называются составными термами. Множество равенств, содержащих только пассивные термы, может быть выведено из квантифицированного равенства систематической генерацией таких равенств, осуществляемой подстановкой вместо каждого идентификатора значения пассивных термов того же сорта, что и идентификатор значения. Например,

```
FOR ALL b : bool  not(not(b))=b
```

представляет

```
not(not(true))      = true;
not(not(false))     = false
```

В качестве альтернативного множества равенств для bool может быть взято

```
FOR ALL b : bool
    not(not(b))     = b;
    not(true)       = false
```

Если сорт квантифицированного идентификатора значения очевиден из контекста, то клаузу, определяющую идентификатор значения, принято опускать, так что пример приобретает вид

```
not(not(b))     = b;
not(true)       = false
```

5.3.5 *Алгебраическая спецификация и семантика (смысловое содержание)*

Алгебраическая спецификация состоит из сигнатуры и наборов равенств для каждого сорта этой сигнатуры. Эти наборы равенств индуцируют отношения эквивалентности, определяющие смысловое содержание спецификации.

Символ = обозначает отношение эквивалентности, обладающее свойствами рефлективности, симметричности и транзитивности, а также свойством подстановки.

Равенства, задаваемые с типом, позволяют разбить термы на классы эквивалентности. Любые два терма из одного класса эквивалентности интерпретируются как имеющие одно и то же значение. Этот механизм может быть использован для идентификации синтаксически различных термов, имеющих одно и то же предполагаемое значение.

Два терма одного сорта, TERM1 и TERM2, принадлежат одному классу эквивалентности, если

a) имеет место равенство
   TERM1=TERM2

или

b) одним из равенств, выведенных из заданного множества квантифицированных равенств, является
   TERM1=TERM2,

или

c) i) TERM1 принадлежит классу эквивалентности, содержащему TERMA, и

ii) TERM2 принадлежит классу эквивалентности, содержащему TERMB, и

iii) равенство

TERMA=TERMB

либо существует, либо выводится из данного множества квантифицированных равенств, или

d) подставляя вместо некоторого подтерма TERM1 терм из того же класса, что и подтерм, и получая таким образом TERM1A, можно показать, что TERM1A лежит в том же классе эквивалентности, что и TERM2.

Используя все имеющиеся равенства, множество всех термов каждого сорта можно разбить на одни или несколько классов эквивалентности. Существует столько же значений сорта, сколько и классов эквивалентности. Каждый класс эквивалентности представляет одно значение и каждый представитель класса представляет то же самое значение.

5.3.6 *Представление значений*

Таким образом, при интерпретации выражения сперва выводится пассивный терм, для чего определяются фактические значения переменных, используемых в выражении в точке интерпретации, а затем устанавливается класс эквивалентности этого пассивного терма. Класс эквивалентности этого терма определяет значение выражения.

Операциям, используемым в выражениях, придается смысловое содержание посредством определения результирующего значения по заданному множеству аргументов.

Обычно для представления значения класса эквивалентности в этом классе выбирается литерал. Например, bool представляется литералами true и false, а натуральные числа — литералами 0, 1, 2, 3 и т.д. Если литерал отсутствует, обычно используется терм наименьшей возможной сложности (с наименьшим числом операций). Например, обычная нотация для целых отрицательных чисел такова: —1, —2, —3 и т.д.

5.4 *Пассивное использование данных в SDL*

В § 5.4.1 определяются расширения конструкций определений данных из § 5.2. В § 5.4.2 определяется, как следует интерпретировать использование абстрактных типов данных в выражениях для случая, когда выражение "пассивно" (то есть не зависит от переменных или от состояния системы). В § 5.5 определяется, как следует интерпретировать выражения, не являющиеся пассивными (то есть "активные" выражения).

5.4.1 *Конструкции расширенных определений данных*

Конструкции, определенные в § 5.2, являются основой для более кратких форм, излагаемых ниже.

Абстрактная грамматика

Для большей части этих конструкций нет дополнительного абстрактного синтаксиса. Абстрактный синтаксис, относящийся к § 5.4.1 и всем его подразделам, содержится в основном в § 5.2

Конкретная текстовая грамматика

```
<расширенные свойства> :: =
        <правило наследования>
    |   <конкретизация генератора>
    |   <определение структуры>

<расширенный составной терм> :: =
        <расширенный идентификатор операции> (<список составных термов>)
    |   <составной терм> <инфиксная операция> <терм>
    |   <терм> <инфиксная операция> <составной терм>
    |   <одноместная операция> <составной терм>
    |   <условный составной терм>

<расширенный пассивный терм> :: =
        <расширенный идентификатор операции>
                (<пассивный терм> { , <пассивный терм> } *)
    |   <пассивный терм> <инфиксная операция> <пассивный терм>
    |   <одноместная операция> <пассивный терм>
    |   <условный пассивный терм>

<расширенный идентификатор операции>
        <идентификатор операции> <восклицание>
    |   <генераторное формальное имя>
    |   [ <квалификатор> ] <"операция">

<расширенное имя операции> :: =
        <имя операции> <восклицание>
    |   <генераторное формальное имя>
    |   <"операция">

<восклицание> :: =
        !

<расширенное имя литерала> :: =
        <литерал знаковой строки>
    |   <генераторное формальное имя>
    |   <литерал класса имен>
```

<расширенный идентификатор литерала> ::=
<идентификатор литерала знаковой строки>
| <генераторное формальное имя>

Правила <расширенные свойства>, <расширенный состав терм>, <расширенный пассивный терм>, <расширенное имя операции>, <расширенное имя литерала> и <расширенный идентификатор литерала> обобщают соответствующие правила для <частичного определения типа> (§ 5.2.1), <составного терма> (§ 5.2.3), <пассивного терма> (§ 5.2.3), <имени операции> (§ 5.2.2), <литерала> (§ 5.2.2) и <идентификатора литерала> (§ 5.2.3) из ядра данных. Вышеприведенные правила обобщаются далее с помощью правил <правило наследования> (§ 5.4.1.11), <конкретизации генератора> (§ 5.4.1.12.2), <генераторное формальное имя> (§ 5.4.1.12.1), <условный составной терм> (§ 5.4.1.6), <условный пассивный терм> (§ 5.4.1.6), <литерал знаковой строки> и <идентификатор литерала знаковой строки> (§ 5.4.1.2) и <литерал класса имен> (§ 5.4.1.14), Правила <инфиксная операция>, <одноместная операция>, <инфиксная "операция"> и <одноместная "операция"> определены в § 5.4.1.1.

Альтернативы <генераторное формальное имя> действительны лишь в <выражениях свойств> в <тексте генератора> (см. § 5.4.1.12), где это имя определяется как формальный параметр.

Альтернативы <расширенного составного терма> и <расширенного пассивного терма>, в которых <генераторное формальное имя> предшествует "(", действительны лишь если <генераторное формальное имя> определяется как принадлежащее классу OPERATOR (см. § 5.4.1.12).

Альтернатива <расширенного имени литерала> с <генераторным формальным именем> действительна, если <генераторное формальное имя> определяется как принадлежащее классу LITERAL (см. § 5.4.1.12).

Альтернатива <расширенного идентификатора литерала> с <генераторным формальным именем> действительна только в том случае, если <генераторное формальное имя> определяется как принадлежащее классам LITERAL или CONSTANT (см. § 5.4.1.12).

Если имя операции определяется с <восклицанием>, то семантически <восклицание> будет частью *имени.*

Формы <имя операции> <восклицание> или <идентификатор операции> <восклицание> представляют соответственно *имя операции* (§ 5.2.2) и *идентификатор операции* (§ 5.2.3).

Семантика

Имя операции, определенное с <восклицанием>, имеет обычную семантику операции, но имя операции — при этом доступно лишь в аксиомах.

5.4.1.1 *Специальные операции*

Таковыми являются имена операций, имеющие специальные синтаксические формы. Специальный синтаксис вводится с тем, чтобы арифметические и Булевские операции имели свою обычную синтаксическую форму. То есть пользователь сможет написать "(1 + 1) = 2" вместо вынужденного использования, например, равно (сложить (1,1),2). Какие сорта допустимы для каждой из операций, будет зависеть от определения типа данных.

Конкретная текстовая грамматика

<"операция"> ::=
<кавычки><инфиксная операция> <кавычки>
| <кавычки><одноместная операция> <кавычки>
<кавычки> ::=

<инфиксная операция> :: =

=>
| OR
| XOR
| AND
| IN
| /=
| =
| >
| <
| <=
| >=
| +
| /
| *
| //
| MOD
| REM
| -

<одноместная операция> :: =

-
| NOT

Семантика

Инфиксная операция в терме имеет обычную семантику операции, но синтаксически записывается либо в инфиксной форме, либо в префиксной форме с кавычками, как это описано выше.

Одноместная операция в терме имеет обычную семантику операции, но синтаксически записывается либо в префиксной форме, либо в префиксной форме с кавычками, как это описано выше.

Инфиксные и одноместные операции, записанные с кавычками, являются доступными именами операций.

Инфиксные операции имеют порядок приоритетности, который определяет очередность применения операций. Очередность применения такая же, как и в специфицированных в § 5.4.2.1 <выражениях>.

Если очередность применения не определяется однозначно, как например в

a OR b XOR c,

то операции применяются в порядке следования слева направо, так что предыдущий терм эквивалентен следующему:

(a OR b) XOR c.

Отметим, что у <"операций"> MOD и REM нет предопределенной семантики, поскольку они не определяются в предопределенных сортах данных.

Модель

Терм вида

<терм1> <инфиксная операция> <терм2>

имеет производную синтаксическую форму от

"<инфиксная операция>" (<терм1>, <терм2>),

где "<инфиксная операция>" используется как правильное имя и "<инфиксная операция>" представляет *имя операции.*

Аналогично

<одноместная операция> <терм>

имеет производную синтаксическую форму от

"<одноместная операция>" (<терм>),

где "<одноместная операция>" используется как правильное имя и представляет *имя операции.*

(Отметим, что операцию равенства в SDL (=) не следует путать с символом эквивалентности термов в SDL (==).)

5.4.1.2 *Литералы знаковых строк*

Конкретная текстовая грамматика

<идентификатор литерала знаковой строки> ::=
K [<квалификатор>] <литерал знаковой строки>

<литерал знаковой строки> ::=
<знаковая строка>

<знаковая строка> — это лексема, определенная в § 2.2.1.

<идентификатор литерала знаковой строки> представляет в абстрактном синтаксисе *Идентификатор-операции-литерала.*

<литерал знаковой строки> представляет в абстрактном синтаксисе, производном от синтаксиса <знаковой строки>, уникальное *Имя-операции-литерала* (§ 5.2.2).

Семантика

Идентификаторы литералов знаковых строк являются идентификаторами, образованными из литералов знаковых строк в термах и выражениях.

Литералы знаковых строк используются в предопределенных сортах данных Знакостроковый и Знаковый (см. § 5.6). Они также связаны определенным образом с литералами классов имен (см. § 5.4.1.14) и отображениями литералов (см. § 5.4.1.15). Эти литералы могут быть определены и для других целей.

Длина <литерала знаковой строки> равна сумме числа <алфавитно-цифровых знаков>, <прочих знаков>, <специальных знаков>, <точек>, <знаков подчеркивания>, <пробелов> и пар <апостроф><апостроф>, содержащихся в <знаковой строке> (см. § 2.2.1).

Для <литерала знаковой строки>

a) длина которого больше единицы, и

b) имеющего подстроку, полученную из <знаковой строки> отбрасыванием последнего знака (<алфавитно-цифрового>, или <прочего знака>, или <специального знака>, или <точки>, или <знака подчеркивания>, или <пробела>, или пары <апостроф><апостроф>), причем

c) эта подстрока определена как литерал такой, что подстрока//отброшенный знак в кавычках является допустимым термом того же сорта, что и <литерал знаковой строки>,

в конкретном синтаксисе имеется неявное равенство, состоящее в том, что <литерал знаковой строки> эквивалентен подстроке, за которой следуют инфиксная операция "//" и отброшенный знак, взятый в апострофы с тем, чтобы образовать <знаковую строку>.

Например, для литералов 'ABC', 'AB''' и 'AB' в

```
NEWTYPE s
LITERALS 'ABC', 'AB''', 'AB', 'A', 'B', '''';
OPERATORS "//": s, s -> s;
```

имеются неявные равенства

```
'ABC'   ==  'AB' // 'C';
'AB'''  ==  'AB' // '''';
'AB'    ==  'A' // 'B';
```

5.4.1.3 *Предопределенные данные*

Предопределенные данные, включая Булевский сорт, определяющий свойства для двух литералов, Истина и Ложь, определяются в § 5.6. Семантика Операций равенства (§ 5.4.1.4), Булевских аксиом (§ 5.4.1.5), Условных термов (§ 5.4.1.6), Упорядочения (§ 5.4.1.8) и Подтипов (§ 5.4.1.9) основывается на определении Булевского сорта (§ 5.6.1). Семантика Литералов Классов Имен (если используются <регулярные интервалы>, § 5.4.1.14) и Отображений Литералов (§ 5.4.1.15) также основывается соответственно на определениях сортов Знаковый (§ 5.6.2) и Знакостроковый (§ 5.6.4).

Предопределенные данные считаются определенными на уровне системы.

5.4.1.4 *Операции равенства*

Конкретная текстовая грамматика

Каждому имени сорта, вводимому <частичным определением типа>, соответствуют неявные *сигнатуры операций* для = и / =, и неявные множества *равенств* для этих операций.

В <частичном определении типа>, которым вводится сорт с именем S, имеется пара неявных *сигнатур операций*, эквивалентных сигнатурам

```
"=" : S, S -> Булевский
"/=": S, S ->Булевский,
```

где Булевский — это предопределенный Булевский сорт.

В <частичном определении типа>, которым вводится сорт с именем S, имеется неявное множество *равенств*

```
FOR ALL a, b, c IN S (
    a=a                              == Истина;
    a=b                              == b=a;
    ((a=b) AND (b=c)) => a=c         == Истина;
    a/=b                             == NOT (a=b);
    a=b == Истина  ==>     a         == b    )
```

Последнее равенство выражает свойство подстановки операции равенства.

Если из равенств (явных, неявных и производных) можно вывести, что

Истина == Ложь,

то это противоречит принятым свойствам Булевского типа данных, так что определение будет недопустимым. Вывод

Истина == Ложь

должен быть невозможным.

Всякое Булевское пассивное выражение, используемое вне определений типов данных, должно интерпретироваться либо как Истина, либо как Ложь. Если подобное выражение нельзя свести к Истине или ко Лжи, то значит, спецификация не полна и допускает различные интерпретации типа данных.

Семантика

Для каждого сорта, вводимого частичным определением типа данных, имеется неявное определение операций равенства и равенств для них.

В конкретном синтаксисе символы = и / = представляют имена операций, называемых операциями равенства и неравенства.

5.4.1.5 *Булевские аксиомы*

Конкретная текстовая грамматика

<Булевская аксиома>:: =
<Булевский терм>

Семантика

Булевская аксиома является утверждением об истинности, выполняющемся при любых условиях для определяемого типа данных, что позволяет использовать ее для специфицирования поведения типа данных.

Модель

Синтаксис аксиомы, записанной в виде
<Булевский терм>,
является производным от конкретного синтаксиса равенства
<Булевский терм> == Истина,
обычным образом связанного с абстрактным синтаксисом равенства.

5.4.1.6 *Условные термы*

В дальнейшем равенство, содержащее условный терм, будет называться равенством с условным термом.

Абстрактная грамматика

Условный-составной-терм	=	*Условный-терм*
Условный-пассивный терм	=	*Условный-терм*
Условный-терм	::	*Условие* *Следствие* *Альтернатива*
Условие	=	*Терм*
Следствие	=	*Терм*
Альтернатива	=	*Терм*

Сортом *Условия* должен быть предопределенный Булевский сорт, и *Условие* не должно быть *Термом-ошибкой*. *Следствие* и *альтернатива* должны быть одного сорта.

Условный терм является *условным составным термом* в том и только в том случае, когда один или несколько *термов условия, следствия* или *альтернативы* являются *составными термами*.

Условный терм является *условным пассивным термом* в том и только в том случае, когда все *термы условия, следствия* и *альтернативы* являются *пассивными термами.*

Конкретная текстовая грамматика

<условный составной терм> :: =
<условный терм>

<условный пассивный терм> :: =
<условный терм>

<условный терм> :: =
IF <условие> THEN <следствие> ELSE <альтернатива> FI

<условие> :: =
<Булевский терм>

<следствие> :: =
<терм>

<альтернатива> :: =
<терм>

Семантика

Условный терм, используемый в равенстве, семантически эквивалентен двум множествам равенств, в которых элиминированы все квантифицированные идентификаторы значений в Булевском терме.

Множество равенств может быть получено одновременными подстановками в равенство с условным термом *пассивных термов* подходящего роста вместо *идентификаторов значений* в *условии.* В этом множестве равенств *условие* всегда заменяется на Булевский *пассивный терм.* В дальнейшем это множество равенств будет называться расширенным пассивным множеством.

Равенство с условным термом эквивалентно множеству *равенств,* содержащему

a) для всякого *равенства* из расширенного пассивного множества, *условие* в котором эквивалентно Истине, само это *равенство,* в котором *условный терм* заменен (пассивным) *следствием,* и

b) для всякого *равенства* из расширенного пассивного множества, *условие* в котором эквивалентно Лжи, само это *равенство,* в котором *условный терм* заменен (пассивной) *альтернативой.*

Отметим, что в частном случае равенство вида
ex1 == IF a THEN b ELSE c FI
эквивалентно паре условных равенств
a == Истина ==> ex1 == b,
a == Ложь ==> ex1 == c.

Пример

IF i =; * j THEN posroot(i) ELSE abs(j) FI == IF positive(j) THEN j ELSE –j FI;

Примечание. — Есть лучшие способы специфицировать такие свойства; это всего лишь пример.

5.4.1.7 *Ошибки*

Ошибки используются для того, чтобы можно было приводить полное определение свойств типа данных даже в тех случаях, когда результату операции не может быть придано никакого конкретного содержания.

Абстрактная грамматика

Терм-ошибка :: ()

Терм-ошибка не должен использоваться как аргументный *терм* для *идентификатора операции* в *составном терме.*

Терм-ошибка не должен использоваться как часть *ограничения.*

Из *Равенств* должно быть невозможным вывести, что *идентификатор операции-литерала* равен *терму-ошибке.*

Конкретная текстовая грамматика

<терм-ошибка> :: =
ERROR <восклицание>

Семантика

Термом может быть ошибка; это позволяет специфицировать обстоятельства, при которых результатом операции будет ошибка. Если эти обстоятельства возникают во время интерпретации, то дальнейшее поведение системы не определено.

5.4.1.8 *Упорядочение*

Конкретная текстовая грамматика

<упорядочение> :: =
ORDERING
(на <упорядочение> имеется ссылка в § 5.2.2)

Семантика

Ключевое слово <упорядочение> является сокращением, используемым вместо явного специфицирования операций упорядочения и множества соответствующих равенств в частичном определении типа.

Модель

В <частичном определении типа>, которым вводится сорт с именем S с ключевым словом ORDERING, имеется неявное множество *сигнатур операций,* эквивалентное следующим явным определениям:

"<" : S, S —> Булевский;
">" : S, S —> Булевский;
"<=" : S, S —> Булевский;
">=" : S, S —> Булевский;

где Булевский — это предопределенный Булевский сорт, а также неявное множество Булевских *аксиом*:

```
FOR ALL a, b IN S
(
        "<"(a, a)   ==  Ложь;
        "<"(a,b)    ==  ">"(b,a);
        "<="(a,b)   ==  "OR"("<"(a,b), "="(a,b));
        ">="(a,b)   ==  "OR"(">"(a,b), "="(a,b));
        "<"(a,b)    => NOT("<"(b,a));
        "<"(a,b)    AND "<"(b,c) => "<"(a,c);
);
```

Если в <частичном определении типа> имеется одновременно <список литералов> и ключевое слово ORDERING, то <сигнатуры литералов> перечисляются по возрастанию, так что

```
LITERALS A, B, C;
OPERATORS ORDERING
```

предполагает, что A<B, B<C.

5.4.1.9 *Подтипы*

Подтип специфицирует некоторое множество значений сорта. Подтип используется как сорт, имеющий ту же семантику, что и сорт, на который ссылается подтип, но с проверкой принадлежности значений заданному множеству значений исходного сорта.

Абстрактная грамматика

Идентификатор-подтипа	=	*Идентификатор*
Определение-подтипа	::	*Имя подтипа* *Идентификатор-родительского-сорта* *Условие-на-диапазон*
Имя-подтипа	=	*Имя*
Идентификатор-родительского-сорта	=	*Идентификатор-сорта*

Конкретная текстовая грамматика

```
<подтип> :: =
        <идентификатор подтипа>

<определение подтипа> :: =
        SYNTYPE
            <имя подтипа> = <идентификатор родительского сорта>
            [<присваивание по умолчанию>] [CONSTANTS <условие на диапазон>
            ENDSYNTYPE <имя подтипа>
    |   NEWTYPE <имя подтипа> [<расширенные свойства>]
            <выражение свойств> CONSTANTS <условие на диапазон>
            ENDNEWTYPE [<имя подтипа>]

<идентификатор родительского сорта> :: =
            <сорт>
```

<подтип является альтернативой для <сорта> (см. § 5.2.2).

<определение подтипа> с ключевым словом SYNTYPE и "=<идентификатор подтипа>" имеет производный синтаксис, определяемый ниже.

<определение подтипа> с ключевым словом **SYNTYPE** в конкретном синтаксисе соответствует *Определению-подтипа* в абстрактном синтаксисе.

<определение подтипа> с ключевым словом **NEWTYPE** можно отличить от <частичного определения типа> по тому, что в нем содержится **CONSTANTS** <условие на диапазон>. Такое <определение подтипа> является сокращением, соответствующим введению <частичного определения типа> с анонимным именем, и следующего за ним <определения подтипа> с ключевым словом **SYNTYPE**, основывающегося на сорте с этим анонимным именем. То есть

```
NEWTYPE X /*детали*/
    CONSTANTS /*список констант*/
ENDNEWTYPE X;
```

эквивалентно

```
NEWTYPE anon /*детали*/
ENDNEWTYPE anon;
```

со следующим за ним

```
SYNTYPE X=anon
    CONSTANTS /*список констант*/
ENDSYNTYPE X;
```

Когда <идентификатор подтипа> используется как <аргумент> в <списке аргументов>, определяющем операцию, сортом этого аргумента в *списке аргументов* является *идентификатор родительского сорта* подтипа.

Когда <идентификатор подтипа> используется как результат операции, сортом *результата* является *идентификатор родительского сорта* подтипа.

Когда <идентификатор подтипа> используется как квалификатор имени, то *квалификатором* будет *идентификатор родительского сорта* подтипа.

Необязательное <имя подтипа>, приводимое в конце <определения подтипа> после ключевого слова **ENDSYNTYPE** или **ENDNEWTYPE**, должно совпадать с <именем подтипа>, специфицированным соответственно после **SYNTYPE** или **NEWTYPE**.

Если используется ключевое слово **SYNTYPE**, но <условие на диапазон> опущено, то считается, что все значения сорта удовлетворяют условию на диапазон, и тем самым <идентификатор подтипа> имеет ту же семантику, что и идентификатор сорта, а условие на диапазон всегда выполнено.

Семантика

Определение подтипа определяет подтип, который ссылается на идентификатор сорта и на условие на диапазон. Идентификатор подтипа специфицируется так же, как и идентификатор родительского сорта подтипа, за исключением случаев

a) присваивания значения переменной, объявленной с подтипом (см. § 5.5.3),

b) вывода сигнала, если один из сортов, специфицированных для сигнала, оказывается подтипом (см. § 2.7.4),

c) вызова процедуры, если подтипом оказывается один из сортов, специфицированных для переменных, являющихся **IN**-параметрами процедуры (см. § 2.4.5).

d) создания процесса, когда подтипом оказывается один из сортов, специфицированных для параметров процесса (см. § 2.7.2 и § 2.4.4),

e) ввода сигнала, если одна из переменных, ассоциированных с вводом, имеет сорт, являющийся подтипом (см. § 2.6.4),

f) использования в выражении операции, у которой сорт аргумента, либо результата — подтип (см. § 5.4.2.2 и § 5.5.2.4),

g) предложения установки или сброса таймера, когда один из сортов в определении таймера — подтип (см. § 2.8),

h) определения импорта (см. § 4.13).

Например <определение подтипа>, в которое вместе с ключевым словом SYNTYPE входит "=<идентификатор подтипа>", эквивалентно <определению подтипа>, в которое вместо <идентификатора родительского типа> подставлен <идентификатор родительского типа> из <определения подтипа>, указанного <идентификатором подтипа>. То есть

```
SYNTYPE s2=nl CONSTANTS a1:a3; ENDSYNTYPE s2;
SYNTYPE s3=s2 CONSTANTS a1:a2; ENDSYNTYPE s3;
```

эквивалентно

```
SYNTYPE s2=n1 CONSTANTS a1:a3; ENDSYNTYPE s2;
SYNTYPE s3=n1 CONSTANTS a1:a2; ENDSYNTYPE s3;
```

Когда подтип определяется в терминах <идентификатора подтипа>, эти два подтипа не должны быть взаимозависимы.

Подтип, определяемый определением подтипа, имеет идентификатор, состоящий из имени, введенного именем подтипа и квалифицированного идентификатором объемлющей структурной единицы.

Подтип имеет сорт, идентифицируемый идентификатором родительского сорта, из определения подтипа.

Подтип имеет диапазон, который является набором значений, специфицированных константами из определения подтипа.

5.4.1.9.1 *Условие на диапазон*

Абстрактная грамматика

Условие-на-диапазон	::	*Идентификатор-операции-или* *Элемент-условия*-**set**
Элемент-условия	=	*Открытый-диапазон* | *Замкнутый диапазон*
Открытый-диапазон	::	*Идентификатор-операции* *Пассивное-выражение*
Замкнутый-диапазон	::	*Идентификатор-операции-и* *Открытый-диапазон* *Открытый-диапазон*
Идентификатор-операции-или	=	*Идентификатор*
Идентификатор-операции-и	=	*Идентификатор*

Конкретная текстовая грамматика

<условие на диапазон> :: =
{ <замкнутый диапазон> | <открытый диапазон> }{, { <замкнутый диапазон> | <открытый диапазон> }} *

<замкнутый диапазон> :: =
<константа> : <константа>

<открытый диапазон> ::=
<константа>
|{ = | / = | < | > | < = | > = } <константа>

<константа> ::=
<пассивное выражение>

Символ "<" (соответственно, "<=", ">", ">=") может использоваться в конкретном синтаксисе <условия на диапазон> только если этот символ был определен с <сигнатурой операции>
P, P — > Булевский,
где P — сорт подтипа. Эти символы представляют *идентификатор операции.*

<замкнутый диапазон> может использоваться только в случае, когда определен символ "<=" с <сигнатурой операции>
P, P — > Булевский,
где P — сорт подтипа.

<константа> в <условии на диапазон> должна иметь тот же сорт, что и подтип.

Семантика

Условие на диапазон определяет проверку на диапазон. Проверка на диапазон используется, когда у подтипа имеется дополнительная семантика по сравнению с родительским сортом этого подтипа (см. § 5.4.1.9 и случаи, когда подтипы имеют различную семантику — см. § 5.5.3, § 2.6.4, § 2.7.2, § 2.5.4, § 5.4.2.2 и § 5.5.4). Проверка на диапазон используется также для определения интерпретации принятия решения (см. § 2.7.5).

Проверка на диапазон состоит в применении операции из условия на диапазон. Результат применения этой операции должен быть эквивалентен истине, в противном случае дальнейшее поведение системы будет неопределенным. Проверка на диапазон производится следующим образом:

a) для всякого элемента в <условии на диапазон> (<открытого диапазона> или <замкнутого диапазона>) в *элементе условия* имеется соответствующий *открытый диапазон* или *замкнутый диапазон;*

b) <открытый диапазон> вида <константа> эквивалентен <открытому диапазону> вида =<константа>;

c) для данного терма A, таким образом,
   i) в проверке на диапазон, отвечающей <открытому диапазону> вида =<константа>, /=<константа>, <<константа>, <= <константа>, > <константа> и >= <константа>, имеются соответственно подтермы вида A= <константа>, A /= <константа>, A <<константа>, A < =<константа>, A ><константа> и A >= <константа>;

   ii) в проверке на диапазон, отвечающей <замкнутому диапазону> вида <первая константа> : <вторая константа>, имеется подтерм вида <первая константа> <= A AND A <= <вторая константа>, где AND отвечает Булевской операции AND и соответствует *Идентификатору операции* AND и в абстрактном синтаксисе;

d) *идентификатор операции или* отвечает операции, применяемой одновременно ко всем элементам *эле-*

мент-условия-set, который является Булевским объединением (OR) всех этих элементов. Проверка на диапазон является термом, образованным Булевским объединением (OR) всех подтермов, произведенных из <условия на диапазон>.

Если подтип специфицирован без <условия на диапазон>, значением проверки на диапазон будет Истина.

5.4.1.10 *Структурные сорта*

Конкретная текстовая грамматика

<определение структуры> ::=
STRUCT <список полей> [<конец>] [ADDING]

<список полей> ::=
<поля> { <конец> <поля> } *

<поля> ::=
<имя поля> { , <имя поля> } * <сорт поля>

<сорт поля> ::=
<сорт>

Каждое <имя поля> структурного сорта должно отличаться от любого другого <имени поля> того же <определения структуры>.

Семантика

Определение структуры определяет структурный сорт, значения которого формируются из списка значений полей различных сортов.

Длина списка значений определяется определением структуры, а сорт значения определяется его местом в списке значений.

Модель

Определение структуры имеет производный синтаксис от синтаксиса определения

a) операции Сделать! для создания значений структуры и

b) операций, как модифицирующих значения структуры, так и извлекающих значения полей из значений структуры.

Именем неявной операции, модифицирующей поле, является имя поля, сцепленное с "Модифицировать!".

Именем неявной операции, извлекающей поле, является имя поля, сцепленное с "Извлечь!".

<списком аргументов> операции Сделать! является список <сортов полей>, встречающихся в списке полей и взятых в том порядке, в котором они встречаются.

<результатом> операции Сделать! является идентификатор сорта структуры.

<списком аргументов> операции, модифицирующей поле, является идентификатор сорта структуры со следующим за ним <сортом поля> этого поля. <результатом> операции, модифицирующего поле, является идентификатор сорта структуры.

<списком аргументов> операции, извлекающей поле, является идентификатор сорта структуры. <результатом> операции, извлекающего поле, является <сорт поля> этого поля.

Для каждого поля существует неявное равенство, определяющее, что модификация поля структуры с помощью значения есть то же самое, что и создание значения структуры, в котором данное поле имеет указанное значение.

Для каждого поля существует неявное равенство, определяющее, что при извлечении поля из значения структуры будет возвращено значение, которое было ассоциировано с этим полем, когда оно создавалось.

Например,

```
NEWTYPE s STRUCT
    b    Булевский;
    i    Целый;
    c    Знаковый;
ENDNEWTYPE s;
```

предполагает

```
NEWTYPE s
OPERATORS
    Сделать!          : Булевский, Целый, Знаковый   -> s;
    bМодифицировать!  : s, Булевский                 -> s;
    iМодифицировать!  : s, Целый                     -> s;
    cМодифицировать!  : s, Знаковый                  -> s;
    bИзвлечь!         : s                            -> Булевский;
    iИзвлечь!         : s                            -> Целый;
    cИзвлечь!         : s                            -> Знаковый;
AXIOMS
    bМодифицировать! (Сделать! (x,y,z),b)            == Сделать!(b,y,z);
    iМодифицировать! (Сделать!(x,y,z),i)             == Сделать!(x,i,z);
    cМодифицировать! (Сделать!(x,y,z),c)             == Сделать!(x,y,c);
    bИзвлечь!        (Сделать!(x,y,z))               == x;
    iИзвлечь!        (Сделать!(x,y,z))               == y;
    cИзвлечь!        (Сделать!(x,y,z))               == z;
ENDNEWTYPE s;
```

5.4.1.11 *Наследование*

Конкретная текстовая грамматика

```
<правило наследования> ::=
        INHERITS <родительский сорт> [<переименование литералов>]
            [[OPERATORS] { ALL | (<список наследования>) } [<конец>]] [ADDING]

<родительский сорт> ::=
        <сорт>

<список наследования> ::=
        <наследуемая операция> { , <наследуемая операция> } *

<наследуемая операция> ::=
        [<имя операции> =] <имя наследуемой операции>

<имя наследуемой операции> ::=
        <имя операции родительского сорта>
```

<переименование литералов> ::=
LITERALS <список переименования литералов> <конец>

<список переименования литералов> ::=
<пара переименования литерала> { , <пара переименования литерала> } *

<пара переименования литерала> ::=
<сигнатура переименования литерала> = <сигнатура переименования родительского литерала>

<сигнатура переименования литерала> ::=
<имя операции-литерала>
| <литерал знаковой строки>

Сорт не должен циклически основываться на себе посредством наследования.

Все <сигнатуры переименования литерала> в <списке переименования литералов> должны быть различными. Все <сигнатуры переименования родительского литерала> в <списке переименования литералов> должны быть различными.

Все <имена наследуемых операций> в <списке наследования> должны быть различными. Все <имена операций> в <списке наследования> должны быть различными.

<имя наследуемой операции>, специфицированное в <списке наследования>, должно быть доступной операцией <родительского сорта>, определенным в <частичном определении типа>, определяющем <родительский сорт>. Имя операции недоступно в данной точке, если оно определено с <восклицанием>.

Если несколько операций <родительского сорта> имеют одно и то же имя, которое приводится как <имя наследуемой операции>, то наследуются все эти операции.

Семантика

Один сорт можно определять как потомок другого, используя NEWTYPE в комбинации с правилом наследования. Сорт-потомок не пересекается с родительским сортом.

Имена литералов, определенных в родительском сорте, наследуются потомком, если только эти литералы не были переименованы. Литерал переименовывается, если имя родительского литерала встречается в паре переименования литерала в качестве второго имени. В этом случае литералу присваивается первое имя указанной пары.

Для каждой операции родительского сорта, кроме "=" и "/=", появляется наследуемая операция потомка. Операцией родительского сорта считается всякая операция, которая одновременно

a) определена в частичном определении типа или определении подтипа (исключая то, которое определяется), определяющем сорт, видимый в точке наследования, а также

b) имеет аргумент или результат родительского сорта.

Имена операций потомка указываются посредством ALL или списком наследования. Имя операции потомка определяется следующим образом:

a) оно совпадает с именем операции родительского сорта, если указано ALL и это имя явно или неявно определяется как имя операции в том частичном определении типа или определении подтипа, которое определяет родительский сорт, либо

b) если идентификатор родительской операции приведен в списке наследования и для операции потомка дается имя операции, за которым следует "=", то операции будет присвоено данное имя, либо

c) если идентификатор родительской операции приведен в списке наследования, но для операции потомка не дается имени операции со следующим за ним "=", то будет унаследовано имя операции родительского сорта, либо

d) если ALL не указано и идентификатор родительской операции не упомянут в списке наследования, то операции будет присвоено недоступное, но уникальное имя. Такие имена не могут явно использоваться ни в аксиомах, ни в выражениях.

Сорта аргументов и результата операции сорта-потомка те же самые, что и у соответствующей операции родительского сорта, за исключением случаев, когда сортом аргумента или результата является родительский сорт. В этом случае родительский сорт заменяется потомком. Таким образом каждое вхождение родительского сорта в операции потомка будет заменяться на этот сорт-потомок.

Из каждого равенства родительского сорта при наследовании производится новое равенство-потомок. Равенствами родительского сорта считаются:

a) все равенства, содержащие операцию (или литерал) родительского сорта, а также

b) любое равенство, определяемое частичным определением типа или определением подтипа (исключая то, которое определяется), которым определяется рост, видимый в точке наследования.

Равенство-потомок будет совпадать с равенством родительского сорта, за исключением того, что

a) каждое вхождение родительского сорта будет заменяться на новый сорт, и

b) операции (или литералы) родительского сорта, для которых имеются переименованные операции потомка (или литералы), подлежат тому же переименованию в равенстве-потомке.

Вследствие изменения сортов согласно (a) идентификаторы литералов и операций потомка будут квалифицироваться идентификатором нового сорта.

Модель

Конкретный синтаксис <правила наследования> связан с конкретным синтаксисом <выражения свойств> в <частичном определении типа> или <определении подтипа>, содержащем <правило наследования>.

Множеству <литералов> нового сорта в абстрактном синтаксисе соответствует множество <сигнатур литералов> в <выражении свойств> вместе с множеством литералов-потомков.

Множеству <операций> нового сорта в абстрактном синтаксисе соответствует множество <сигнатур операций> в <выражении свойств> вместе со множеством операций-потомков.

Множеству <равенств> нового сорта в абстрактном синтаксисе соответствуют <аксиомы> в <выражении свойств> вместе со множеством равенств-потомков.

Пример

```
NEWTYPE bit
        INHERITS Булевский
            LITERALS    1 = Истина,  0= Ложь;
            OPERATORS ("NOT", "AND", "OR")
        ADDING
        OPERATORS
            EXOR:  bit,bit -> bit;
        AXIOMS /* примечание — здесь использованы 2 разных способа записи NOT */
            EXOR(a,b) == (a AND "NOT"(b)) OR (NOT a AND b);
ENDNEWTYPE bit;
```

5.4.1.12 *Генераторы*

Генератор позволяет определить зависящий от параметров текстовый шаблон, который заполняется при конкретизации, прежде чем будет рассмотрена семантика типов данных.

5.4.1.12.1 *Определение генератора*

Конкретная текстовая грамматика

<определение генератора> ::=
GENERATOR <имя генератора> (<список параметров генератора>) <текст генератора>
ENDGENERATOR [<имя генератора>]

<текст генератора> ::=
[<конкретизация генераторов>] <выражение свойств>

<список параметров генератора> ::=
<параметр генератора> { , <параметр генератора> } *

<параметр генератора> ::=
{ TYPE | LITERAL | OPERATOR | CONSTANT }
<генераторное формальное имя> { , <генераторное формальное имя>} *

<генераторное формальное имя> ::=
<генераторное формальное имя>

<генераторный сорт> ::=
<генераторное формальное имя>
| <имя генератора>

<имя генератора> или <генераторное формальное имя> должны использоваться в <выражении свойств>, если только <выражение свойств> содержится в <тексте генератора>.

В <определении генератора> все <генераторные формальные имена> одного класса (TYPE, LITERAL, OPERATOR или CONSTANT) должны быть различными. В одном <определении генератора> имена класса LITERAL должны отличаться от имен класса CONSTANT.

<имя генератора>, которое дается после ключевого слова GENERATOR, должно отличаться от всех имен сортов в <определении генератора>, а также от всех <параметров генератора> класса TYPE данного <определения генератора>.

<генераторный сорт> допустим, если только он встречается в <тексте генератора> как <расширенный сорт> (см. § 5.2.2), а имя является либо <именем генератора> этого <определения генератора>, либо <генераторным формальным именем>, определенным в этом определении.

Если <генераторным сортом> является <генераторное формальное имя>, то оно должно быть именем класса TYPE.

Необязательное <имя генератора> после ключевого слова ENDGENERATOR должно совпадать с <именем генератора>, которое дается после ключевого слова GENERATOR.

<генераторное формальное имя> не должно использоваться в <квалификаторе>. <имя генератора> или <генераторное формальное имя> не должны

a) квалифицироваться, или
b) использоваться с <восклицанием>, или
c) использоваться в <присваивании по умолчанию>.

Семантика

Генератор именует фрагмент текста, который может использоваться в конкретизациях генератора.

Предполагается, что тексты конкретизаций генераторов в тексте генератора вычисляются в точке определения текста генератора.

Каждый параметр генератора имеет класс (TYPE, LITERAL, OPERATOR или CONSTANT), специфицируемый соответственно ключевыми словами TYPE, LITERAL, OPERATOR или CONSTANT.

Модель

Текст, задаваемый определением генератора, связывается с абстрактным синтаксисом только после конкретизации генератора. В точке определения определение генератора не имеет соответствующего абстрактного синтаксиса.

Пример

```
GENERATOR bag(TYPE item)
LITERALS empty;
OPERATORS
        put    : item, bag —> bag;
        count  : item, bag —> Целый;
        take   : item, bag —> bag;
AXIOMS
        take(i,put(i,b))     == b;
        take(i,empty)        == ERROR!
        count(i,empty)       == 0
        count(i,put(j,b))    == count(i,b)+IF i=j THEN 1 ELSE 0 FI;
        put(i,put(j,b))      == put(j,put(i,b));
ENDGENERATOR bag;
```

Примечание. — В формальном определении (Приложение F.2) запрещается использовать <генераторное формальное имя> в квалификаторах. Эта тема в рекомендации подверглась исправлению уже после того как Приложение F.2 было опубликовано. Таким образом, Приложение F.2 не применимо к данной теме.

5.4.1.12.2 *Конкретизация генератора*

Конкретная текстовая грамматика

<конкретизация генератора> ::=
        {<конкретизация генератора> [<конец>] [ADDING]} +

<конкретизация генератора> ::=
        <идентификатор генератора> (<список фактических параметров генератора>)

<список фактических параметров генератора> ::=
        <фактический параметр генератора> { , <фактический параметр генератора> }*

<фактический параметр генератора> ::=
        <расширенный сорт>
    | <сигнатура литерала>
    | <имя операции>
    | <пассивный терм>

Если <параметр генератора> принадлежит классу TYPE, то соответствующий <фактический параметр генератора> должен быть <расширенный сорт>.

Если <параметр генератора> принадлежит классу LITERAL, соответствующим <фактическим параметром генератора> должна быть <сигнатура литерала>.

<сигнатура литерала>, являющаяся <литералом класса имен>, может быть использована в качестве <фактического параметра генератора> в том и только в том случае, если соответствующее <генераторное формальное имя> не встречается в <аксиомах> или в <отображении литералов> <выражения свойств> в <тексте генератора>.

Если <парамептр генератора> принадлежит классу OPERATOR, соответствующим <фактическим параметром генератора> должно быть <имя операции>.

Если <параметр генератора> принадлежит классу CONSTANT, соответствующим <фактическим параметром генератора> должен быть <пассивный терм>.

Если <фактическим параметром генератора> является <генераторное формальное имя>, класс <генераторного формального имени> должен совпадать с классом <фактического параметра генератора>.

Семантика

Использование конкретизации генератора в расширенных свойствах или в тексте генератора обозначает конкретизацию текста, идентифицированного идентификатором генератора. Чтобы образовать из текста генератора вычисляемый текст для литералов, операций и аксиом, следует

a) подставить фактические параметры генератора вместо параметров генератора, а также

b) вместо имени генератора подставить

   i) идентификатор сорта, определяемого частичным определением типа или определением подтипа, в случае, если конкретизация генератора содержится в этом частичном определении типа или определении подтипа, либо

   ii) имя генератора, если конкретизация генератора содержится в этом генераторе.

Конкретизацией текста для литералов является конкретизация текста из литералов в выражении свойств текста генератора с опущенным ключевым словом LITERALS.

Конкретизацией текста для операций является конкретизация текста из списка операций в выражении свойств текста генератора, с опущенным ключевым словом OPERATORS.

Конкретизацией текста для аксиом является конкретизация текста из аксиом в выражении свойств текста генератора, с опущенным ключевым словом AXIOMS.

Если в списке конкретизаций генераторов имеется несколько конкретизаций генератора, конкретизация текста для литералов (операций и аксиом) образуется сцеплением конкретизаций текстов для литералов (операций и аксиом) всех генераторов, взятых в том порядке, в котором они встречаются в списке.

Конкретизацией текста для литералов является список литералов выражения свойств объемлющего частичного выражения типа, определения подтипа или определения генератора, предшествующий любому списку литералов, явно упомянутому в выражении свойств. То есть в случае, когда специфицировано упорядочение, литералы, определенные конкретизациями генератора, упорядочиваются согласно порядку, в котором они конкретизуются, и предшествуют всем прочим литералам.

Конкретизации текста для операций и аксиом добавляются соответственно к спискам операций и аксиом объемлющего частичного определения типа, определения подтипа или определения генератора.

Если конкретизация текста добавляется к выражению свойств, предполагается, что ключевые слова LITERALS, OPERATORS и AXIOMS будут добавлены при необходимости, с тем чтобы получить корректный конкретный синтаксис.

Модель

Абстрактный синтаксис, соответствующий конкретизации генератора, определяется только после конкретизации. Он определяется по конкретизации текста в точке конкретизации.

Пример

```
NEWTYPE boolbag bag (Булевский)
        ADDING
        OPERATORS
            yesvote : boolbag —> Булевский;
        AXIOMS
            yesvote(b) == count (Истина,b) > count (Ложь,b);
ENDNEWTYPE boolbag;
```

5.4.1.13 *Синонимы*

Синонимы именуют пассивные выражения, представляющие одно из значений сорта.

Конкретная текстовая грамматика

```
<определение синонима> ::=
        SYNONYM <имя синонима> [<сорт>] = <пассивное выражение>
    |   <определение внешнего синонима>
```

Альтернатива <определение внешнего синонима> описана в § 4.3.1.

Если сорт <пассивного выражения> не определяется однозначно, то этот сорт должен быть специфицирован в <определении синонима>.

Сортом, идентифицируемым <сортом>, должен быть один из тех сортов, к которым может быть отнесено <пассивное выражение>.

<пассивное выражение> не должно ссылаться на синоним, определенный <определением синонима>, ни прямо, ни косвенно (через другой синоним).

Семантика

Значение, представляемое синонимом, определяется из контекста, в котором появляется определение синонима.

Если сорт пассивного выражения не может быть определен однозначно по контексту синонима, то он должен задаваться <сортом>.

Значением синонима является значение пассивного терма из определения синонима.

Сортом синонима является сорт пассивного терма из определения синонима.

Модель

<пассивное выражение> в конкретном синтаксисе обозначает *пассивный терм* в абстрактном синтаксисе, как определено в § 5.4.2.2.

Если <сорт> специфицирован, результат <пассивного выражения> связан с этим *сортом*. <пассивное выражение> представляет в абстрактном синтаксисе *пассивный терм*, имеющий *идентификатор операции* с тем же именем и с теми же сортами аргументов, что и в конкретном синтаксисе, и с сортом результата, совпадающим с сортом, специфицированным в конкретном синтаксисе.

5.4.1.14 *Литералы классов имен*

Литерал класса имен является сокращенной записью множества имен литералов (возможно, бесконечного), определенных регулярным выражением.

Конкретная текстовая грамматика

<литерал класса имен> ::=
NAMECLASS <регулярное выражение>

<регулярное выражение> ::=
<частичное регулярное выражение>
{ [OR] <частичное регулярное выражение> } *

<частичное регулярное выражение> ::=
<регулярный элемент> [<имя натурального литерала> | + | *]

<регулярный элемент> ::=
(<регулярное выражение>)
| <литерал знаковой строки>
| <регулярный интервал>

<регулярный интервал> ::=
<литерал знаковой строки> : <литерал знаковой строки>

Имена, образованные <литералом класса имен>, должны удовлетворять обычным статическим условиям для литералов (см. § 5.2.2) и либо лексическим правилам для имен (см. § 2.2.1), либо конкретному синтаксису <литерала знаковой строки> (см. § 5.4.1.2).

Оба <литерала знаковой строки> в <регулярном интервале> должны быть единичной длины и оба должны быть литералами, определенными Знаковым сортом (см. § 5.6.2).

Семантика

Литерал класса имен — это альтернативный способ специфицировать сигнатуры литералов.

Модель

Множество имен, которому эквивалентен литерал класса имен, определяется как совокупность имен, удовлетворяющих синтаксису, специфицированному <регулярным выражением>. В абстрактном синтаксисе литерал класса имен эквивалентен этому множеству имен.

<регулярное выражение>, являющееся списком <частичных регулярных выражений>, не содержащим OR, специфицирует имена, образуемые из знаков, определенных первым <частичным регулярным выражением>, за которыми следуют знаки, определенные вторым <частичным регулярным выражением>.

Если между двумя <частичными регулярными выражениями> специфицировано OR, то имена образуются либо из первого, либо из второго из этих <частичных регулярных выражений>. Отметим, что OR имеет боль-

ший приоритет, чем обычная конкатенация, так что

NAMECLASS 'A' '0' OR '1' '2';

эквивалентно

NAMECLASS 'A' ('0' OR '1') '2';

и определяет литералы A02, A12.

Если за <регулярным элементом> следует <имя натурального литерала>, то <частичное регулярное выражение> эквивалентно <регулярному элементу>, повторенному специфицированное <именем натурального литерала> число раз.

Например,

NAMECLASS 'A' ('A' OR 'B') 2

определяет имена AAA, AAB и ABB.

Если за <регулярным элементом> следует '*', то <частичное регулярное выражение> эквивалентно <регулярному элементу>, повторенному несколько раз, или опущенному.

Например,

NAMECLASS 'A' ('A' OR 'B')*

определяет имена A, AA, AB, AAA, AAB, ABA, ABB, AAAA, ... и т.д.

Если за <регулярным элементом> следует '+', то <частичное регулярное выражение> эквивалентно <регулярному элементу>, повторенному не менее одного раза.

Например,

NAMECLASS 'A' ('A' OR 'B')+

определяет имена AA, AB, AAA, AAB, ABA, ABB, AAAA, ... и т.д.

<регулярный элемент>, взятый в скобки <регулярным выражением>, определяет последовательности знаков, определяемые <регулярным выражением>.

<регулярный элемент>, являющийся <литералом знаковой строки>, определяет последовательность знаков, задаваемую литералом знаковой строки (опуская кавычки).

<регулярный элемент>, являющийся <регулярным интервалом>, определяет все знаки, специфицируемые <регулярным интервалом>, в качестве альтернативных последовательностей знаков. Знаками, которые определяются <регулярным интервалом>, являются все знаки, большие или равные первому знаку, и меньшие или равные второму знаку, в соответствии с определением знакового сорта (см. § 5.6.2)

Например,

'a' : 'f'

определяет альтернативы 'a' или 'b' или 'c' или 'd' или 'e' или 'f'.

Если существенна последовательность, в которой определяются имена (например, если специфицировано ORDERING), то предполагается, что имена определяются в алфавитном порядке, согласно упорядочиванию знакострокового сорта. Если два имени начинаются одинаково, но имеют разную длину, первым определяется более короткое из них.

5.4.1.15 *Отображение литералов*

Отображения литералов являются сокращениями, используемыми для определения отображения литералов в значения.

Конкретная текстовая грамматика

<отображение литералов> ::=

MAP <литеральное равенство> { <конец> <литеральное равенство> } *[<конец>]

<литеральное равенство> ::=
<литеральная квантификация>
<литеральные аксиомы>{<конец> <литеральные аксиомы>} * [<конец>])

<литеральные аксиомы> ::=
<равенство>
| <литеральное равенство>

<литеральная квантификация> ::=
FOR ALL <имя значения>{ , <имя значения>} * IN <расширенный сорт> LITERALS

<считывающий терм> ::=
SPELLING (<идентификатор значения>)

Правила <отображение литералов> и <считывающий терм> не являются частью ядра данных, но встречаются в правилах <выражение свойств> и <пассивный терм> соответственно в § 5.2.1 и § 5.2.3.

Семантика

Отображение литералов является сокращением, используемым для определения большого (возможно, бесконечного) числа аксиом, охватывающих все литералы сорта. Отображение литералов позволяет отображать литералы сорта в значения этого сорта. Когда сорт содержит большое (или бесконечное) число значений, отображение литералов является единственным средством, позволяющим практически определить значения, отвечающие каждому литералу.

Считывающий терм используется в отображениях литералов для отсылки к знаковой строке, образованной из данного литерала. Этот механизм позволяет использовать Знакостроковые операции для определения отображений литералов.

Модель

<отображение литералов> является сокращенным обозначением для множества <аксиом>. Это множество <аксиом> выводится из <литеральных равенств> в <отображении литералов>. <равенствами>, используемыми для этого вывода, являются все <равенства>, содержащиеся в <аксиомах> правил <литеральные аксиомы>. В каждом из этих <равенств> подвергаются замене <идентификаторы значений>, определяемые <именем значения> в <литеральной квантификации>. В каждом из выведенных <равенств> всякое вхождение одного <идентификатора значения> заменяется одним и тем же <идентификатором операции-литерала>, <сорт> которого определяется <литеральной квантификацией>. Производное множество <аксиом> содержит все возможные <равенства>, которые могут быть выведены таким образом.
Производное множество <аксиом> для <литеральных равенств> добавляется к <аксиомам>, определяемым после ключевого слова AXIOMS и до ключевого слова MAP в том же <частичном определении типа> (если таковые имеются).

Например,

```
NEWTYPE abc LITERALS 'A', b, 'c';
    OPERATORS
        "<": abc, abc -> Булевский;
        "+" : abc, abc -> Булевский;
    MAP FOR ALL x, y IN abc LITERALS
        (x < y => y + x);
ENDNEWTYPE abc;
```

представляет собой сокращение для

```
NEWTYPE abc LITERALS 'A', b, 'c';
    OPERATORS
        "<" : abc, abc -> Булевский;
        "+" : abc, abc -> Булевский;
```

```
AXIOMS
    'A'  < 'A'    => 'A'   + 'A';
    'A'  < b      => b     + 'A';
    'A'  < 'c'    => 'c'   + 'A';
    b    < 'A'    => 'A'   + b;
    b    < b      => b     + b;
    b    < 'c'    => 'c'   + b;
    'c'  < 'A'    => 'A'   + 'c';
    'c'  < b      => b     + 'c';
    'c'  < 'c'    => 'c'   + 'c';
ENDNEWTYPE abc;
```

Если в <литеральной квантификации> содержится один или несколько <считывающих термов>, то вместо <считывающих термов> подставляются Знакостроковые литералы (см. § 5.6.3).

Если <сигнатурой литерала> <идентификатора операции литерала> в <считывающем терме> является <имя операции литерала> (см. § 5.2.2), то <считывающий терм> будет сокращенным обозначением строки знаков верхнего регистра, произведенной из <идентификатора операции-литерала>. Эта строка знаков состоит из знаков верхнего регистра, соответствующих знакам, образующим <имя операции-литерала> — <идентификатора операции-литерала>.

Если <сигнатурой литерала> <идентификатора операции-литерала> в <считывающем терме> является <литерал знаковой строки> (см. § 5.2.2 и § 5.4.1.2), то <считывающий терм> является сокращенным обозначением строки знаков, произведенной из <литерала знаковой строки>. Эта строка знаков состоит из знаков, образующих <литерал знаковой строки>.

Знаковая строка подставляется вместо <иденцификатора значения> после того как <литеральное равенство>, содержащее <считывающий терм>, распространяется вышеописанным образом.

Например,

```
NEWTYPE abc LITERALS 'A',Bb,'c';
    OPERATORS
        "<" : abc,abc -> Булевский;
    MAP FOR ALL x,y IN abc LITERALS
        SPELLING(x) < SPELLING(y) => x < y;
    ENDNEWTYPE abc;
```

представляет собой сокращение для

```
NEWTYPE abc LITERALS 'A',Bb,'c';
    OPERATORS
        "<" : abc,abc -> Булевский;
    AXIOMS
        /* отметим, что 'A', Bb, 'c' отнесены к локальному сорту abc */
        /* '"A"', 'BB'  и  '"c"' должны квалифицироваться идентификатором
        знакостроки, если эти литералы неоднозначны; ниже это для краткости опускается*/
        '"A"'   <  '"A"'  => 'A'   < 'A';
        '"A"'   <  'BB'   => 'A'   < Bb;
        '"A"'   <  '"c"'  => 'A'   < 'c';
        'BB'    <  '"A"'  => Bb    < 'A';
        'BB'    <  'BB'   => Bb    < Bb;
        'BB'    <  '"c"'  => Bb    < 'c';
        '"c"'   <  '"A"'  => 'c'   < 'A';
        '"c"'   <  'BB'   => 'c'   < Bb;
        '"c"'   <  '"c"'  => 'c'   < 'c';
    ENDNEWTYPE abc;
```

Всякое <неквантифицированное равенство> в <литеральных аксиомах> должно содержать <считывающий терм> либо <идентификатор операции-литерала>.

<считывающий терм> должен быть в <отображении литералов>.

<идентификатором значения> в <считывающем терме> должен быть <идентификатор значения>, определенный <литеральной квантификацией>.

5.4.2 *Использование данных*

Далее будет определено, как интерпретируются в выражениях типы данных, сорта, литералы, операции и синонимы.

5.4.2.1 *Выражения*

Выражениями являются литералы, операции, ссылки на переменные, условные выражения и императивные операции.

Абстрактная грамматика

Выражение = *Пассивное-выражение* | *Активное-выражение*

Выражение является *активным выражением,* если оно содержит *активное первичное выражение* (см. § 5.5).

Выражение, не содержащее *активного первичного выражения,* является *пассивным выражением.*

Конкретная текстовая грамматика

Для простоты описания не делается различия между конкретными синтаксисами *пассивного выражения* и *активного выражения.* Конкретный синтаксис <выражения> приводится ниже, в § 5.4.2.2.

Семантика

Выражение интерпретируется как значение пассивного выражения или активного выражения. Если значением является ошибка, то дальнейшее поведение системы не определено.

Выражение имеет сорт пассивного выражения или активного выражения.

5.4.2.2 *Пассивные выражения*

Абстрактная грамматика

Пассивное-выражение :: *Пассивный-терм*

Статические условия для *пассивного терма* применимы также к *пассивному выражению.*

Конкретная текстовая грамматика

<пассивное выражение> ::=
<пассивное выражение>

<выражение> ::=
        <операнд0>
    | <подвыражение> => <операнд0>

<подвыражение> ::=
        <выражение>

<операнд0> ::=
        <операнд1>
    | <подоперанд0> { OR| XOR } <операнд1>

<подоперанд0> ::=
        <операнд0>

<операнд1> ::=
        <операнд2>
    | <подоперанд1> AND <операнд2>

<подоперанд1> ::=
        <операнд1>

<операнд2> ::=
        <операнд3>
    | <подоперанд2>{ = | /= | > | > = | < | < = | IN}<операнд3>

<подоперанд2> ::=
        <операнд2>

<операнд3> ::=
        <операнд4>
    | <подоперанд3> { + | − | // } <операнд4>

<подоперанд3> ::=
        <операнд3>

<операнд 4> ::=
        <операнд5>
    | <подоперанд4>{ * | / | MOD | REM } <операнд5>

<подоперанд4> ::=
        <операнд4>

<операнд5> ::=
        [− | NOT / <первичное выражение>

<первичное выражение> ::=
        <пассивное первичное выражение>
    | <активное первичное выражение>
    | <обобщенное первичное выражение>

<пассивное первичное выражение> ::=
        <идентнфикатор литерала>
    | <идентификатор операции> (<список пассивных выражений>)
    | (<пассивное выражение>)
    | <условное пассивное выражение>

<обобщенное первичное выражение> ::=
<синоним>
| <индексированное первичное выражение>
| <полевое первичное выражение>
| <структурное первичное выражение>

<список пассивных выражений> ::=
<пассивное выражение> { , <пассивное выражение> }*

<идентификатор операции> ::=
<идентификатор операции>
| [<квалификатор>] <"операция">

<выражение>, не содержащее <активных первичных выражений>, представляет *пассивное выражение* в абстрактном синтаксисе. <пассивное выражение> не должно содержать <активные первичные выражения>.

Если <выражение> является <пассивным первичным выражением> с <идентификатором операции>, и <сорт аргумента> <сигнатуры операции> является <подтипом>, то к соответствующему значению аргумента применяется проверка на диапазон для данного подтипа, определенная в § 5.4.1.9.1. Значением проверки на диапазон должна быть Истина.

Если <выражение> является <пассивным первичным выражением> с <идентификатором операции>, и <сорт результата> <сигнатуры операции> является <подтипом>, то к соответствующему значению результата применяется проверка на диапазон для данного подтипа, определенная в § 5.4.1.9.1. Значением проверки на диапазон должна быть Истина.

Если в <выражении> содержится <обобщенное первичное выражение> <то есть <синоним>, <индексированное первичное выражение>, <полевое первичное выражение> или <структурное первичное выражение>), то оно заменяется на уровне конкретного синтаксиса как определено соответственно в § 5.4.2.3, § 5.4.2.4, § 5.4.2.5 и § 5.4.2.6, прежде чем будет рассмотрено соответствие с абстрактным синтаксисом.

Необязательный <квалификатор> перед <"операцией"> соотносится с абстрактным синтаксисом так же, как и <квалификатор> <идентификатора операции> (см. § 5.2.2).

Семантика

Пассивное выражение интерпретируется как значение, обозначаемое пассивным термом, синтаксически эквивалентным этому пассивному выражению.

Вообще говоря, нет необходимости делать различие между пассивным термом и значением пассивного терма. Например, пассивный терм, значением которого является целая единица, может быть записан как "1". Обычно существуют различные пассивные термы, обозначающие одно и то же значение, как например, целые пассивные термы "0+1", "3—2" и "(7+5)/12", и, как правило, для обозначения выражения принято использовать пассивный терм наиболее простого вида (в данном случае "1").

Пассивное выражение имеет сорт эквивалентного ему пассивного терма.

Пассивное выражение имеет значение, являющееся значением эквивалентного ему пассивного терма.

5.4.2.3 *Синоним*

Конкретная текстовая грамматика

<синоним> ::=
<идентификатор синонима>
| <внешний синоним>

Альтернатива <внешний синоним> описана в § 4.3.1.

Семантика

Синоним является сокращением, используемым для обозначения определенного выражения в другом месте.

Модель

<синоним> представляет <пассивное выражение>, определенное <определением синонима>, которое идентифицировано <идентификатором синонима>. Используемый в <пассивном выражении> <идентификатор> представляет *идентификатор* в абстрактном синтаксисе, соответствующем контексту <определения синонима>.

5.4.2.4 *Индексированное первичное выражение*

Индексированное первичное выражение является сокращенным синтаксическим обозначением, которое может быть использовано для обозначения "индексирования" значения "массива". Однако, помимо особой синтаксической формы, индексированное первичное выражение не имеет никаких специальных свойств и обозначает просто операцию, параметром которой является первичное выражение.

Конкретная текстовая грамматика

<индексированное первичное выражение> ::=
<первичное выражение> (<список выражений>)

Семантика

Индексированное выражение представляет применение операции Извлечь!

Модель

<первичное выражение>, за которым следует взятый в скобки <список выражений>, служит производным конкретным синтаксисом для конкретного синтаксиса
Извлечь! (<первичное выражение>, <список выражений>)

При этом оно рассматривается как правильное выражение, хотя Извлечь! недопустимо использовать в качестве имени операции в конкретном синтаксисе для выражений. По этому конкретному выражению абстрактный синтаксис определяется согласно § 5.4.2.2.

5.4.2.5 *Полевое первичное выражение*

Полевое первичное выражение является сокращенным синтаксическим обозначением, которое может быть использовано для обозначения "отбора поля" в "структурах". Однако помимо особой синтаксической формы, полевое первичное выражение не имеет никаких специальных свойств и обозначает просто операцию, параметром которой является первичное выражение.

Конкретная текстовая грамматика

<полевое первичное выражение> ::=
<первичное выражение> <отбор поля>

<отбор поля> ::=
        !<имя поля>
    | (<имя поля>{, <имя поля>} *),
Это имя поля должно быть именем поля, определенным для сорта первичного выражения.

Семантика

Полевое первичное выражение представляет применение одной из операций извлечения поля структурного сорта.

Модель

Форма
        <первичное выражение> (<имя поля>)
служит производным синтаксисом для
        <первичное выражение> ! <имя поля>

Форма
        <первичное выражение> (<имя первого поля>{ , <имя поля>} *)
служит производным синтаксисом для
        <первичное выражение> ! <имя первого поля>{ ! <имя поля>} *),
где сохранен порядок имен полей.

Форма
        <первичное выражение> ! <имя поля>
служит производным синтаксисом для
        <имя операции извлечения поля> (<первичное выражение>),

где имя операции извлечения поля образовано сцеплением имени поля и Извлечь! в указанном порядке. Например,
        s ! fl
служит производным синтаксисом для
        fl Извлечь! (s).

При этом данное выражение считается правильным, хотя в конкретном синтаксисе для выражений fl Извлечь! недопустимо в качестве имени операции. По этому конкретному выражению абстрактный синтаксис определяется согласно § 5.4.2.2.

Если для сорта определена такая операция, что
        Извлечь! (s, имя)

является допустимым термом, когда "имя" совпадает с допустимым именем поля для сорта, которому принадлежит s, то первичное выражение
        s (имя)

служит производным конкретным синтаксисом для
        Извлечь! (s, имя)
и отбор поля должен записываться в форме
        s ! имя

5.4.2.6 *Структурное первичное выражение*

Конкретная текстовая грамматика

<структурное первичное выражение> ::=
        [<квалификатор>] (. <список выражений>.)

Семантика

Структурное первичное выражение представляет значение структурного сорта, построенное из выражений для каждого поля структуры.

Форма

(. <список выражений> .)

служит производным конкретным синтаксисом для

Сделать! (<список выражений>)

рассматриваемого как правильное пассивное выражение, хотя в конкретном синтаксисе для пассивных выражений Сделать! не допускается в качестве имени операции. По этому конкретному пассивному выражению абстрактный синтаксис определяется согласно § 5.4.2.2.

5.4.2.7 *Условное пассивное выражение*

Конкретная текстовая грамматика

<условное пассивное выражение> ::=
IF <Булевское пассивное выражение>
THEN <пассивное выражение-следствие>
ELSE <пассивное выражение-альтернатива>
FI

<пассивное выражение-следствие> ::=
<пассивное выражение>

<пассивное выражение-альтернатива> ::=
<пассивное выражение>

<условное пассивное выражение> представляет *пассивное выражение* в абстрактном синтаксисе. Если <Булевское пассивное выражение> представляет Истину, то *пассивное выражение* представляется <пассивным выражением-следствием>, в противном же случае оно представляется <пассивным выражением-альтернативой>.

Сорт <пассивного выражения-следствия> должен совпадать с сортом <пассивного выражения-альтернативы>.

Семантика

Условное пассивное выражение является пассивным первичным выражением, интерпретируемым либо как пассивное выражение-следствие, либо как пассивное выражение-альтернатива.

Если значением <Булевского пассивного выражения> является Истина, то <пассивное выражение-альтернатива> не интерпретируется. Если значением <Булевского пассивного выражения> является Ложь, то <пассивное выражение-следствие> не интерпретируется. Если значением интерпретируемого <пассивного выражения> является ошибка, то дальнейшее поведение системы не определено.

Сортом условного пассивного выражения является сорт пассивного выражения-следствия (а также пассивного выражения-альтернативы).

5.5 *Использование данных с переменными*

В этом разделе определяется, как используются данные и переменные, объявленные в процессах и процедурах, а также императивные операции, получающие значения из лежащей в основе системы.

Переменная имеет сорт и ассоциированное значение этого сорта. Значение, ассоциированное с переменной, может быть изменено присвоением переменной нового значения. Значение, ассоциированное с переменной, может быть использовано в выражении осуществлением доступа к переменной.

Любое выражение, содержащее переменную, считается "активным", поскольку значение, получаемое при интерпретации выражения, может изменяться в зависимости от значения, присвоенного переменной последним.

5.5.1 *Переменная и определения данных*

Конкретная текстовая грамматика

```
<определение данных> ::=
            {   <частичное определение данных>
              | <определение подтипа>
              | <определение генератора>
              | <определение синонима>} <конец>
```

Определение данных составляет часть *определения типа данных,* если оно является <частичным определением типа> или <определением подтипа>, как определено соответственно в § 5.2.1 и § 5.4.1.9. Правила <определение генератора> и <определение синонима> определены соответственно в § 5.4.1.12 и § 5.4.1.13.

Синтаксис для введения переменных процесса и для переменных-параметров процедуры дается соответственно в § 2.5.1.1 и § 2.3.4. Переменная, определенная в процедуре, не должна быть открытой.

Семантика

Определение данных используется либо для определения части типа данных, либо для определения синонима для выражения, как определяется далее в § 5.2.1, § 5.4.1.9 и в § 5.4.1.13.

Когда переменная создана, она содержит специальное значение, называемое неопределенным, которое отличается от всех других значений сорта этой переменной.

5.5.2 *Доступ к переменным*

Ниже определяется, как интерпретируется выражение, содержащее переменные.

5.5.2.1 *Активные выражения*

Абстрактная грамматика

Активное-выражение = *Доступ-к-переменной* |
Условное-выражение |
Применение-операции |
Императивная-операция

Конкретная текстовая грамматика

```
<активное выражение> ::=
        <активное выражение>
```

<активное первичное выражение> ::=
<доступ к переменной>
| <применение операции>
| <условное выражение>
| <императивная операция>
| (<активное выражение>)
| <активное обобщенное первичное выражение>

<активное обобщенное первичное выражение> ::=
<активное обобщенное первичное выражение>

<список выражений> ::=
<выражение> { , <выражение> } *

Для краткости конкретный синтаксис <активного выражения> дается <выражением> в § 5.4.2.2. <выражение> является <активным выражением>, если оно содержит <активное первичное выражение>.

Также для краткости конкретный синтаксис <активного обобщенного первичного выражения> дается <обобщенным первичным выражением> в § 5.4.2.2. <обобщенное первичное выражение> является <активным обобщенным первичным выражением>, если оно содержит <активное первичное выражение>. Замена на уровне конкретного синтаксиса для <обобщенного первичного выражения> осуществляется как определено в § 5.4.2.3, § 5.4.2.4, § 5.4.2.5 и § 5.4.2.6, прежде чем будет рассмотрено соответствие с абстрактным синтаксисом.

Семантика

Активным выражением является выражение, значение которого зависит от текущего состояния системы.

Сортом активного выражения является сорт эквивалентного пассивного терма.

Значением активного выражения является пассивный терм, эквивалентный активному выражению в момент интерпретации.

Модель

Всякий раз, когда интерпретируется активное выражение, значение активного выражения определяется нахождением пассивного терма, эквивалентного активному выражению. Этот пассивный терм определяется из пассивного выражения, образуемого подстановкой вместо каждого активного первичного выражения в активном выражении пассивного терма, эквивалентного значению данного активного первичного выражения. Значением активного выражения будет значение полученного пассивного выражения.

В активном выражении операции интерпретируются в порядке, определяемом конкретным синтаксисом, приведенным в § 5.4.2.2, либо, если порядок определяется неоднозначно, слева направо. В списке активных выражений или в списке выражений элементы списка интерпретируются слева направо.

5.5.2.2 *Доступ к переменной*

Абстрактная грамматика

Доступ-к-переменной = *Идентификатор-переменной*

Конкретная текстовая грамматика

<доступ к переменной> ::=
<идентификатор переменной>

Семантика

Доступ к переменной интерпретируется как выдача значения, ассоциированного с идентифицированной переменной.

Сортом доступа к переменной является сорт переменной, идентифицированной доступом к переменной.

Значением доступа к переменной является последнее из значений, ассоциированных с переменной, либо, если этим значением было специальное значение "неопределенное", то значением доступа к переменной будет ошибка. Если значением доступа к переменной является ошибка, то дальнейшее поведение системы не определено.

5.5.2.3 *Условное выражение*

Условным выражением является выражение, интерпретируемое либо как следствие, либо как альтернатива.

Абстрактная грамматика

Условное-выражение	::	*Булевское-выражение* *Выражение-следствие* *Выражение-альтернатива*
Булевское-выражение	=	*Выражение*
Выражение-следствие	=	*Выражение*
Выражение-альтернатива	=	*Выражение*

Сорт *выражения-следствия* должен совпадать с сортом *выражения-альтернативы*.

Конкретная текстовая грамматика

<условное выражение> ::=
IF <Булевское активное выражение>
THEN <выражение-следствие>
ELSE <выражение-альтернатива>
FI
| IF <Булевское выражение>
THEN <активное выражение-следствие>
ELSE <выражение-альтернатива>
FI
| IF <Булевское выражение>
THEN <выражение-следствие>
ELSE <активное выражение-альтернатива>
FI

<выражение-следствие> ::=
<выражение>

<выражение-альтернатива>:: =
<выражение>

<условное выражение> отличается от <условного пассивного выражения> наличием в <условном выражении> <активного выражения>.

Семантика

Интерпретация условного выражения состоит в интерпретации условия, за которой следует либо интерпретация выражения-следствия, либо интерпретация выражения-альтернативы. Следствие интерпретируется только если значением условия будет Истина, поэтому если значением условия будет Ложь, дальнейшее поведение системы окажется не определенным лишь в случае, когда выражением-альтернативой является ошибка. Аналогично, альтернатива интерпретируется только если значением условия будет Ложь, поэтому если значением условия будет Истина, дальнейшее поведение системы окажется неопределенным лишь в случае, когда выражением-следствием является ошибка.

Сортом условного выражения является сорт следствия и альтернативы.
Значением условного выражения является значение следствия, если условие есть Истина, и значение альтернативы, если условие есть Ложь.

5.5.2.4 *Применение операции*

Применение операции состоит в применении операции, одним или несколькими фактическими аргументами которого являются активные выражения.

Абстрактная грамматика

Применение-операции :: *Идентификатор-операции*
Выражение$^{+}$

Если сортом аргумента *сигнатуры операции* является подтип, и соответствующее *выражение* в списке *выражений* является *пассивным выражением,* то значением определенной в § 5.4.1.9.1 проверки на диапазон, примененной к значению *выражения,* должна быть Истина.

Конкретная текстовая грамматика

<применение операции> ::=
<идентификатор операции> (<список активных выражений>)

<список активных выражений> ::=
<активное выражение>], <список выражений>]
| <пассивное выражение>, <список активных выражений>

<применение операции> отличается от синтаксически подобного ему <пассивного выражения> тем, что одно из <выражений> в приведенном в скобках списке <выражений> является <активным выражением>. Если все <выражения> в скобках являются <пассивными выражениями>, то конструкция представляет определенное в § 5.4.2.2 *пассивное выражение.*

Семантика

Применение операции является активным выражением, значением которого является значение пассивного терма, эквивалентного применению операции. Эквивалентный пассивный терм определяется согласно § 5.5.2.1.

Список выражений в применении операции интерпретируется в заданном порядке прежде чем будет проинтерпретирована операция.

Если сортом аргумента сигнатуры операции является подтип, и соответствующее выражение в списке активных выражений есть активное выражение, то к значению этого выражения применяется проверка на диапазон, определенная в § 5.4.1.9.1. Если в момент интерпретации значением проверки на диапазон является Ложь, значит система находится в состоянии ошибки и дальнейшее ее поведение не определено.

Если сорт результата сигнатуры операции является подтипом, то к значению применения операции применяется проверка на диапазон, определенная в § 5.4.1.9.1. Если в момент интерпретации значением проверки на диапазон является Ложь, значит система находится в состоянии ошибки и дальнейшее ее поведение не определено.

5.5.3 *Предложение присваивания*

Абстрактная грамматика

Предложение присваивания	::	*Идентификатор-переменной* *Выражение*

Сорт *идентификатора переменной* и сорт *выражения* должны совпадать.

Если *переменная* объявлена с *подтипом* и *выражение* является *пассивным выражением,* значением определенной в § 5.4.1.9.1 проверки на диапазон, примененной к *выражению,* должна быть Истина.

Конкретная текстовая грамматика

<предложение присваивания> ::=
<переменная> := <выражение>

<переменная> ::=
<идентификатор переменной>
| <индексированная переменная>
| <полевая переменная>

Если <переменной> является <идентификатор переменной>, то <выражение> в конкретном синтаксисе представляет <выражение> в абстрактном синтаксисе. Другие формы <переменной> — <индексированная переменная> и <полевая переменная> имеют производный синтаксис и <выражение> в абстрактном синтаксисе определяется из эквивалентного конкретного синтаксиса, определенного ниже, в § 5.5.3.1 и § 5.5.3.2.

Семантика

Предложение присваивания интерпретируется как ассоциирование значения выражения в предложении присваивания с переменной, идентифицированной в предложении присваивания. Предыдущее значение, ассоциированное с переменной, теряется.

Если переменная объявлена с подтипом и выражение является активным выражением, то к выражению применяется проверка на диапазон, определенная в § 5.4.1.9.1. Если проверка на диапазон эквивалентна Лжи, то присваивание ошибочно и дальнейшее поведение системы не определено.

5.5.3.1 *Индексированная переменная*

Индексированная переменная является сокращенным синтаксическим обозначением, которое может быть использовано для обозначения "индексирования" "массивов". Однако, кроме специальной синтаксической формы, индексированное активное первичное выражение не имеет специальных свойств и обозначает просто операцию с активным первичным выражением в качестве параметра.

Конкретная текстовая грамматика

<индексированная переменная> ::=
<переменная> (<список выражений>)

Должно иметься подходящее определение операции Модифицировать!.

Семантика

Индексированная переменная представляет присваивание значения, образованного применением операции Модифицировать! к доступу к переменной и к выражению, заданному в индексированной переменной.

Модель

Конкретная синтаксическая форма

<переменная> (<список выражений>) := <выражение>

служит производным конкретным синтаксисом для

<переменная> := Модифицировать! (<переменная>, <список выражений>, <выражение>),

где повторно взята та же <переменная> и текст считается правильным присваиванием, хотя в конкретном синтаксисе для выражений недопустимо использовать Модифицировать! в качестве имени операции. Абстрактный синтаксис для этого <оператора присваивания> определяется согласно вышеприведенному § 5.5.3.

Модель для индексированных переменных должна применяться до модели для импорта (см § 4.13).

5.5.3.2 *Полевая переменная*

Полевая переменная является сокращенным обозначением для присваивания значения переменной, при котором изменяется значение только в одном из полей этой переменной.

Конкретная текстовая грамматика

<полевая переменная> ::=

<переменная> <отбор поля>

Должно иметься подходящее определение операции Модифицировать!. Обычно это определение неявно содержится в определении структурного сорта.

Семантика

Полевая переменная представляет присваивание значения, образованного применением операции модифицирования поля.

Модель

Приведенный в скобках отбор поля служит производным синтаксисом для определенного в § 5.4.2.5 отбора поля !<имя поля>.

Конкретная синтаксическая форма

<переменная> ! <имя поля> := <выражение>

служит производным конкретным синтаксисом для

<переменная> := <имя операции модифицирования поля> (<переменная>, <выражение>),

где

a) повторно берется та же <переменная>

b) <имя операции модифицирования поля> образовано сцеплением имени поля с "Модифицировать!", и тогда

c) текст считается правильным присваиванием, хотя в конкретном синтаксисе для выражений <имя операции модифицирования поля> недопустимо в качестве имени операции.

Если в отборе поля имеется несколько <имен полей>, то они моделируются вышеуказанным образом с заменой каждого! <имени поля> по очереди слева направо. При этом остающаяся часть <полевой переменной> рассматривается как <переменная>. Например,

var ! fielda ! fieldb := expression;

моделируется вначале посредством

var ! fielda := fieldb Модифицировать! (far ! fielda, expression);

а затем посредством

var := fielda Модифицировать! (var, fieldb Модифицировать! (var ! fielda, expression));

Абстрактный синтаксис для <предложения присваивания>, образованного при моделировании, определяется согласно вышеприведенному § 5.5.3.

5.5.3.3 *Присваивание по умолчанию*

Присваивание по умолчанию является сокращенным обозначением для присваивания одного и того же значения всем переменным специфицированного сорта непосредственно после того как они созданы.

Конкретная текстовая грамматика

<присваивание по умолчанию> ::=

DEFAULT <пассивное выражение> [<конец>]

В <частичном определении типа> или <определении подтипа> должно содержаться не более одного <присваивания по умолчанию>. (Этим предупреждаются многократные присваивания, которые могут возникнуть при конкретизациях генераторов.)

Семантика

Присваивание по умолчанию — необязательная дополнительная возможность выражения свойств сорта. Присваиванием по умолчанию специфицируется, что любой переменной, объявленной с сортом, введенным частичным определением типа или определением подтипа, изначально присваивается значение пассивного выражения.

Если присваивание по умолчанию отсутствует, то переменная при объявлении будет ассоциирована с неопределенным значением.

Переменной может быть присвоено и другое значение, если она объявляется с явной инициализацией.

Присваивание по умолчанию не наследуется.

Модель

Конкретная синтаксическая форма

DEFAULT <пассивное выражение>,

используемая в выражении свойств там, где вводится сорт s, означает неявное присваивание переменной <пассивного выражения>. Это присваивание интерпретируется непосредственно после объявления переменной и до интерпретации любого явно специфицированного действия в том же процессе или процедуре. Например, если для сорта s задано

DEFAULT 2*dnumber

и в конкретном синтаксисе имеется объявление

DCL v s;

то тогда имеется неявное присваивание

v := 2*dnumber;

Если в объявлении имеется также <начальное значение>, то <начальное значение> присваивается переменной после того как ей присвоено <пассивное выражение> из <присваивания по умолчанию>.

Неявное присваивание связано с абстрактным синтаксисом как обычное <предложение присваивания> (см. § 5.5.3).

Если для <определения данных> специфицировано <присваивание по умолчанию>, то для <сорта> (представляющего подтип или сорт) указано значение, присваиваемое по умолчанию, являющееся значением <пассивного выражения> из <присваивания по умолчанию>. Если в <определении подтипа> не содержится <присваивания по умолчанию>, то подтип имеет значение, присваиваемое по умолчанию, если для *идентификатора родительского сорта* (идентифицирующего подтип или сорт), заданного в *определении подтипа*, указано значение, присваиваемое по умолчанию.

Для <определения подтипа> присваивания интерпретируются в том и только в том случае, когда значением определенной в § 5.4.1.9.1 проверки на диапазон, примененной к значению, присваиваемому по умолчанию, является Истина. То есть для каждой переменной этого подтипа имеется неявное принятие решения вида.

```
DECISION <проверка на диапазон>;
    (Истина): <присваивание по умолчанию>
    ELSE: ENDDECISION.
```

5.5.4 *Императивные операции*

Императивные операции получают значения из подчеркнутых состояний системы.

Абстрактная грамматика

Императивная-операция	*Выражение-"сейчас"* |
	Pid-выражение |
	Обозначающее-выражение |
	Выражение-активности-таймера

Конкретная текстовая грамматика

```
<императивная операция> :: =
        <выражение-"сейчас">
    |   <импортирующее выражение>
    |   <PId-выражение>
    |   <обозревающее выражение>
    |   <выражение активности таймера>
```

Альтернатива <импортирующее выражение> определена в § 4.13.

Императивными операциями являются выражения, используемые для проверки того, активны ли таймеры, и для осуществления доступа к системным часам, к ассоциированным с процессором PId-значениям или к импортированным переменным.

5.5.4.1 *NOW*

Абстрактная грамматика

Выражение -"сейчас"	::	()

Конкретная текстовая грамматика

```
<выражение-"сейчас">  ::=
        NOW
```

Семантика

Выражение-"сейчас" является выражением, осуществляющим доступ к переменной системных часов для определения абсолютного системного времени.

Выражение-"сейчас" представляет выражение, запрашивающее текущее значение системных часов, задающих время. Начало отсчета и единица отсчета времени зависят от системы. От системы зависит, будут ли два различных NOW в одном переходе выдавать одно и то же значение.

Выражение-"сейчас" имеет временный сорт.

5.5.4.2 *Выражение IMPORT*

Конкретная текстовая грамматика

Конкретный синтаксис импортирующего выражения определен в § 4.3.

Семантика

В дополнение к семантике, определенной в § 4.13, импортирующее выражение интерпретируется как доступ (см. § 5.5.2.2) к неявной переменной импортирующего выражения.

Модель

Импортирующее выражение имеет неявный синтаксис для импортирования значения, как определено в § 4.13, а также имеет неявный *доступ к переменной* для неявной переменной импорта в контексте, в котором появляется <импортирующее выражение>.

5.5.4.3 *PId-выражение*

Абстрактная грамматика

Pid-выражение	=	*Выражение-"сам"* |
		Выражение-"родитель" |
		Выражение-"потомок" |
		Выражение-"отправитель"
Выражение-"сам"	::	()
Выражение-"родитель"	::	()
Выражение-"потомок"	::	()
Выражение-"отправитель"	::	()

Конкретная текстовая грамматика

```
PId-выражение> ::=
        SELF
    |   PARENT
    |   OFFSPRING
    |   SENDER
```

Семантика

PId-выражение осуществляет доступ к одной из неявных переменных процесса, определенных в § 2.4.4. Выражение-переменная процесса интерпретируется как последнее из значений, ассоциированных с соответствующей неявной переменной.

Сортом PId-выражение является PId.

Значением PId-выражения является последнее значение из значений, ассоциированных с соответствующей переменной, как определено в § 2.4.4.

5.5.4.4 *Обозревающее выражение*

Обозревающее выражение позволяет процессу получить значение переменной другого процесса из того же блока, как если бы эта переменная была определена локально. Обозревающий процесс не может модифицировать значение, ассоциированное с переменной.

Абстрактная грамматика

Обозревающее-выражение :: *Идентификатор-переменной*
Выражение

Выражение должно быть PId-выражением.

Идентификатор переменной должен быть одним из идентификаторов одной из переменных процесса, идентифицированного *выражением.*

Конкретная текстовая грамматика

<обозревающее выражение> ::=
VIEW (<идентификатор переменной>, <PId-выражение>)

<идентификатор переменной> должен быть определен как обозреваемый в <определении обозревания> в процессе, содержащем <обозревающее выражение>. <квалификатор> в <идентификаторе переменной> может быть опущен лишь в случае если в <определении обозревания> объемлющего <определения процесса> не содержатся другие переменные с тем же <именем>.

Семантика

Обозревающее выражение интерпретируется таким же образом, как и доступ к переменной (см. § 5.5.2.2). Переменной, к которой осуществлен доступ, является переменная в процессе, идентифицированном PId-выражением, соответствующим PId-выражению (см. § 5.5.4.3).

Сортом и значением обозревающего выражения являются сорт и значение доступа к переменной.

PId-выражение должно идентифицировать существующий процесс в том же блоке, что и процесс, в котором интерпретируется обозревающее выражение. Иначе обозревающее выражение будет ошибкой и дальнейшее поведение системы будет не определено. PId-выражение должно идентифицировать тот же тип процесса, что и *Идентификатор-процесса* в соответствующем определении обозревания.

5.5.4.5 *Выражение активности таймера*

Абстрактная грамматика

Выражение-активности-таймера :: *Идентификатор-таймера*
*Выражение**

Сорта *Выражения** в *Выражении-активности-таймера* должны соответствовать по позиции сортам в ***Идентификатор-ссылки-на-сорт****, следующим непосредственно за *Именем-таймера* (§ 2.8), идентифицированным ***Идентификатором-таймера***.

Конкретная текстовая грамматика

<выражение активности таймера> ::=
ACTIVE (<идентификатор таймера> [(<список выражений<)])

<список выражений> определен в § 5.5.2.1.

Семантика

Выражение активности таймера является выражением Булевского сорта, имеющим значение Истина, если таймер, идентифицированный идентификатором таймера и установленный с теми же значениями, что приведены в списке выражений (если такой имеется), активен (см. § 2.8.2). В противном случае выражение активности таймера имеет значение Ложь. Выражения интерпретируются в порядке очередности.

5.6 *Предопределенные данные*

В настоящем разделе определяются сорта данных и генераторы данных, неявно определенные на уровне системы. Отметим, что синтаксис и иерархия специальных операций (инфиксных и одноместных) определены в разделе 5.4.1.1, но семантика этих операций (кроме REM и MOD) определяется в настоящм разделе в определениях данных.

5.6.1 *Булевский сорт*

5.6.1.1 *Определение*

```
NEWTYPE Булевский
        LITERALS Истина, Ложь;
        OPERATORS
                "NOT"   :  Булевский                    -> Булевский;
                                                                /*
                "="     :  Булевский, Булевский         -> Булевский;      Операции "=" и "/=" неявные.
                "/="    :  Булевский, Булевский         -> Булевский;      См. § 5.4.1.4.
                                                                */
                "AND"   :  Булевский, Булевский         -> Булевский;
                "OR"    :  Булевский, Булевский         -> Булевский;
                "XOR"   :  Булевский, Булевский         -> Булевский;
                "=>"    :  Булевский, Булевский         -> Булевский;
        AXIOMS
                "NOT" (Истина)              == Ложь;
                "NOT" (Ложь)                == Истина;

                "="   (Истина, Истина)      == Истина;
                "="   (Истина, Ложь)        == Ложь;
                "="   (Ложь, Истина)        == Ложь;
                "="   (Ложь, Ложь)          == Истина;

                "/="  (Истина, Истина)      == Ложь;
                "/="  (Истина, Ложь)        == Истина;
                "/="  (Ложь, Истина)        == Истина;
                "/="  (Ложь, Ложь)          == Ложь;

                "AND" (Истина, Истина)      == Истина;
                "AND" (Истина, Ложь)        == Ложь;
                "AND" (Ложь, Истина)        == Ложь;
                "AND" (Ложь, Ложь)          == Ложь;

                "OR" (Истина, Истина)       == Истина;
                "OR" (Истина, Ложь)         == Истина;
                "OR" (Ложь, Истина)         == Истина;
                "OR" (Ложь, Ложь)           == Ложь;

                "XOR" (Истина, Истина)      == Ложь;
                "XOR" (Истина, Ложь)        == Истина;
                "XOR" (Ложь, Истина)        == Истина;
                "XOR" (Ложь, Ложь)          == Ложь;

                "=>" (Истина, Истина)       == Истина;
                "=>" (Истина, Ложь)         == Ложь;
```

```
        "=>" (Ложь, Истина)        == Истина;
        "=>" (Ложь, Ложь)          == Истина;
ENDNEWTYPE Булевский;
```

5.6.1.2 *Использование*

Булевский сорт используется для представления значений истина и ложь. Он часто используется как результат сравнения.

Булевский сорт используется во многих сокращенных формах данных в SDL, таких как аксиомы без символа "==" и неявные операции равенства "=" и "/=".

5.6.2 *Знаковый сорт*

5.6.2.1 *Определение*

```
NEWTYPE Знаковый

LITERALS
    NUL, SOH, STX, ETX, EOT, ENQ, ACK, BEL,
    BS,  HT,  LF,  VT,  FF,  CR,  SO,  SI,
    DLE, DC1, DC2, DC3, DC4, NAK, SYN, ETB,
    CAN, EM,  SUB, ESC, IS4, IS3, IS2, IS1,
    ' ', '!', '"', '#', '¤', '%', '&', '''',
    '(', ')', '*', '+', ',', '-', '.', '/',
    '0', '1', '2', '3', '4', '5', '6', '7',
    '8', '9', ':', ';', '<', '=', '>', '?',
    '@', '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', '[', '\', ']', '^', '_',
    '`', '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', '{', '|', '}', '~', DEL;

    /* это апостроф, ' ' это пробел, '~' это надстрочный символ или тильда */
OPERATORS

                                                        /*
    "="   : Знаковый, Знаковый   -> Булевский;          Сигнатуры операций "=" и "/=" не-
    "/="  : Знаковый, Знаковый   -> Булевский;          явные — см. § 5.4.1.4
                                                        */
    "<"   : Знаковый, Знаковый   -> Булевский;
    "<="  : Знаковый, Знаковый   -> Булевский;
    ">"   : Знаковый, Знаковый   -> Булевский;
    ">="  : Знаковый, Знаковый   -> Булевский;
AXIOMS
/* ниже специфицируется отношение "меньше" между соседними знаковыми литералами */

    NUL < SOH == Истина;     SOH < STX == Истина;
    STX < ETX == Истина;     ETX < EOT == Истина;
    EOT < ENQ == Истина;     ENQ < ACK == Истина;
    ACK < BEL == Истина;     BEL < BS  == Истина;
    BS  < HT  == Истина;     HT  < LF  == Истина;
    LF  < VT  == Истина;     VT  < FF  == Истина;
    FF  < CR  == Истина;     CR  < SO  == Истина;
    SO  < SI  == Истина;     SI  < DLE == Истина;
```

```
DLE  < DC1  == Истина;      DC1  < DC2  == Истина;
DC2  < DC3  == Истина;      DC3  < DC4  == Истина;
DC4  < NAK  == Истина;      NAK  < SYN  == Истина;
SYN  < ETB  == Истина;      ETB  < CAN  == Истина;
CAN  < EM   == Истина;      EM   < SUB  == Истина;
SUB  < ESC  == Истина;      ESC  < IS4  == Истина;
IS4  < IS3  == Истина;      IS3  < IS2  == Истина;
IS2  < IS1  == Истина;      IS1  < ' '  == Истина;
' '  < '!'  == Истина;      '!'  < '"'  == Истина;
'"'  < '#'  == Истина;      '#'  < '$'  == Истина;
'$'  < '%'  == Истина;      '%'  < '&'  == Истина;
'&'  < '''  == Истина;      '''  < '('  == Истина;
'('  < ')'  == Истина;      ')'  < '*'  == Истина;
'*'  < '+'  == Истина;      '+'  < ','  == Истина;
','  < '-'  == Истина;      '-'  < '.'  == Истина;
'.'  < '/'  == Истина;      '/'  < '0'  == Истина;
'0'  < '1'  == Истина;      '1'  < '2'  == Истина;
'2'  < '3'  == Истина;      '3'  < '4'  == Истина;
'4'  < '5'  == Истина;      '5'  < '6'  == Истина;
'6'  < '7'  == Истина;      '7'  < '8'  == Истина;
'8'  < '9'  == Истина;      '9'  < ':'  == Истина;
':'  < ';'  == Истина;      ';'  < '<'  == Истина;
'<'  < '='  == Истина;      '='  < '>'  == Истина;
'>'  < '?'  == Истина;      '?'  < '@'  == Истина;
'@'  < 'A'  == Истина;      'A'  < 'B'  == Истина;
'B'  < 'C'  == Истина;      'C'  < 'D'  == Истина;
'D'  < 'E'  == Истина;      'E'  < 'F'  == Истина;
'F'  < 'G'  == Истина;      'G'  < 'H'  == Истина;
'H'  < 'I'  == Истина;      'I'  < 'J'  == Истина;
'J'  < 'K'  == Истина;      'K'  < 'L'  == Истина;
'L'  < 'M'  == Истина;      'M'  < 'N'  == Истина;
'N'  < 'O'  == Истина;      'O'  < 'P'  == Истина;
'P'  < 'G'  == Истина;      'Q'  < 'R'  == Истина;
'R'  < 'S'  == Истина;      'S'  < 'T'  == Истина;
'T'  < 'U'  == Истина;      'U'  < 'U'  == Истина;
'U'  < 'W'  == Истина;      'W'  < 'X'  == Истина;
'X'  < 'Y'  == Истина;      'Y'  < 'Z'  == Истина;
'Z'  < '['  == Истина;      '['  < '\'  == Истина;
'\'  < ']'  == Истина;      ']'  < '^'  == Истина;
' '  < '_'  == Истина;      '_'  < '`'  == Истина;
'v'  < 'a'  == Истина;      'a'  < 'b'  == Истина;
'b'  < 'c'  == Истина;      'c'  < 'd'  == Истина;
'd'  < 'e'  == Истина;      'e'  < 'f'  == Истина;
'f'  < 'g'  == Истина;      'g'  < 'h'  == Истина;
'h'  < 'i'  == Истина;      'i'  < 'j'  == Истина;
'j'  < 'k'  == Истина;      'k'  < 'l'  == Истина;
'l'  < 'm'  == Истина;      'm'  < 'n'  == Истина;
'n'  < 'o'  == Истина;      'o'  < 'p'  == Истина;
'p'  < 'q'  == Истина;      'q'  < 'r'  == Истина;
'r'  < 's'  == Истина;      's'  < 't'  == Истина;
't'  < 'u'  == Истина;      'u'  < 'v'  == Истина;
'v'  < 'w'  == Истина;      'w'  < 'x'  == Истина;
'x'  < 'y'  == Истина;      'y'  < 'z'  == Истина;
'z'  < '{'  == Истина;      '{'  < '|'  == Истина;
'|'  < '}'  == Истина;      '}'  < '~'  == Истина;
```

```
    ' ' < DEL == Истина;
    FOR ALL a,b,c IN Знаковый (
        a < a                          == Ложь;
        a < b AND b < c => a < c  == Истина;
        a < b                          == b > a;
        a < b OR a  > b                == a /= b;
        a < b => NOT (b < a);
        NOT (a /= b)                   == a = b;
        a < b OR a = b                 == a <= b;
        a > b OR a = b                 == a >= b;)

ENDNEWTYPE Знаковый;
```

5.6.2.2 *Использование*

Знаковый сорт определяет строки знаков длины 1. Знаки принадлежат Международному Алфавиту № 5. Они определяются либо как строки, либо как сокращения, согласно Международной Справочной Версии алфавита. Печатное представление может меняться в зависимости от национального использования алфавита.

Для Знакового сорта определены 128 различных литералов и значений. Определяется упорядочивание значений и операций равенства и неравенства.

5.6.3 *Генератор Строка*

5.6.3.1 *Определение*

```
GENERATOR Строка (TYPE Сортэлем, LITERAL Пустстр)  /* Строки "индексируются" с единицы*/
    LITERALS Пустстр;
    OPERATORS
        СдСтроку   : Сортэлем    ->  Строка;                    /* сделать строку из элемента*/
        Длина      : Строка      ->  Целый;                     /* длина строки*/
        Первый     : Строка      ->  Сортэлем;                  /* первый элемент строки*/
        Последний  : Строка      ->  Сортэлем;                  /* последний элемент строки*/
        "//"       : Строка, Строка  -> Строка;                 /* сцепление*/
        Извлечь!   : Строка, Целый   -> Сортэлем;               /* получить элемент строки*/
        Модифицировать !: Строка, Целый, Сортэлем -> Строка;    /* модифицировать значение строки*/
        Подстрока: Строка, Целый, Сортэлем -> Строка;           /* получить подстроку строки*/
        /* подстрока (s, i, j) выдает строку длины j, начинающуюся с i-го элемента */
    AXIOMS
    FOR ALL item, itemi, itemj, item1, item2 IN Сортэлем (
    FOR ALL s, s1, s2, s3 IN Строка (
    FOR ALL i, j IN Целый (
        type Строка Длина (Пустстр)                  == 0;
        type Строка Длина (СдСтроку (item))          == 1;
        type Строка Извлечь! (СдСтроку(item), 1)     == item;
        Первый (s)                                   == Извлечь! (s, 1);
        Последний (s)                                == Извлечь! (s, Длина (s));
        Длина (s1//s2)                               == Длина (s1) + Длина (s2);
        Длина (Модифицировать! (s,i,item))           == Длина (s);
        (s1//s2)//s3                                 == s1//(s2//s3);
        Пустстр//s                                   == s;
        s//Пустстр                                   == s;
        Пустстр = (СдСтроку (item)//s2)              == Ложь;
        (СдСтроку (item1//s1)=(СдСтроку (item2)//s2) == (item1 = item2) AND (s1 = s2);
```

```
        i > 0 AND i <= Длина (s) == Истина ==>
                        Извлечь! (Модифицировать! (s,i,item),i)== item;
        i/= j AND i > 0 AND i <= Длина (s) AND j > 0 AND j <= Длина (s)== Истина ==>
                        Извлечь! (Модифицировать! (s,i,item),j)== Извлечь! (s,j),
        i <= 0 OR i > Длина (s) == Истина ==> Извлечь! (s, i) == EPPOR!
        i/=j == Истина ==>
            Модифицировать! (Модифицировать! (s,i.itemi),j,itemj) == Модифицировать!
                (Модифицировать! (s,j,itemj),i,itemi);
        Модифицировать! (Модифицировать! (s,i,item1),i,item2) == Модифицировать! (s,i,item2);
        i <= 0  OR  i > Длина (s) == Истина ==> Модифицировать! (s,i,item) == EPPOR!
        i <= Длина (s1) == Истина ==> Извлечь! (s1//s2,i) == Извлечь! (s1, i);
        i > Длина (s1) == Истина ==> Извлечь! (s1//s2,i) == Извлечь! (s2,i — Длина (s1));
        i > 0 AND i <= Длина (s) == Истина ==> Подстрока (s,i,0) == Пустстр;
        i > 0 AND i <= Длина (s) == Истина ==>
            ПодСтрока (s,i,1) == СдСтроку (Извлечь! (s,i));
        i > 0 AND i <= Длина (s) AND i—1+j <= Длина (s) AND j > 1 == Истина ==>
            ПодСтрока (s,i,j) == ПодСтрока (s,i,1)//ПодСтрока (s,i + 1,j—1);
        i < 0 OR i > Длина (s) OR j >= 0 OR i + j > Длина (s) == Истина ==>
            ПодСтрока (s,i,j) == ERROR!
        i > 0) AND i <= Длина (s) == Истина ==>
                        Модифицировать! (s,i,item) ==
                            ПодСтрока (s,1,i—1)//СдСтроку (item)//ПодСтрока (s,i + 1, Длина (s)—i);)));
ENDGENERATOR Строка;
```

5.6.3.2 *Использование*

Генератор Строка может использоваться для определения сорта, значениями которого будут строки элементов произвольного заданного сорта. Общепринятым является Знакостроковый сорт, определяемый ниже.

Как правило, операции Извлечь! и Модифицировать! используются вместе с определенными в § 5.4.2.4 и § 5.5.3.1 сокращениями для доступа к значениям строк и присваивания значений строкам.

5.6.4 *Знакостроковый сорт*

5.6.4.1 *Определение*

```
NEWTYPE Знакостроковый Строка (Знаковый,'')
        ADDING LITERALS NAMECLASS''''((' ':'&') OR '''''' OR ('(':'~')) + '''';
        /* строка знаков любой длины, состоящая из любых знаков от пробела' ' до тильды '~' */
        /* равенства вида
            'ABC' == 'AB' // 'C';
            неявные — см. § 5.4.1.2 */
        MAP     FOR ALL c IN Знакостроковый LITERALS (
                        FOR ALL charstr IN Знакостроковый LITERALS (
                            Spelling(sharstr) == Spelling(c) ==> charstr == Сдстроку (c);
                    )       ); /* строка 'A' образована из знака 'A' и т.д.*/
ENDNEWTYPE Знакостроковый;
```

5.6.4.2 *Использование*

Знакостроковый сорт определяет строки знаков. Знакостроковый литерал может содержать печатаемые знаки и пробелы. Не печатаемые знаки могут использоваться как строки, если применить Сдстроку, например, Сдстроку (DEL).

```
/* Пример */ SYNONYM newline_prompt Знакостроковый = Сдстроку (CR)// Сдстроку (LF)//'$ > ';
```

5.6.5 *Целый сорт*

5.6.5.1 *Определение*

```
NEWTYPE Целый
        LITERALS NAMECLASS ('0': '9')* ('0' : '9');
            /* необязательная последовательность цифр,
               предшествующая одной из цифр от 0 до 9*/
        OPERATORS
            "—"     : Целый            —> Целый;
            "+"     : Целый, Целый     —> Целый;
            "—"     : Целый, Целый     —> Целый;
            "*"     : Целый, Целый     —> Целый;
            "/"     : Целый, Целый     —> Целый;
                                                    /*
            "="     : Целый, Целый     —> Булевский;      Сигнатуры операций "=" и "/=" неявные —
                                                          см. § 5.4.1.4
            "/="    : Целый, Целый     —> Булевский;
                                                    */
            "<"     : Целый, Целый     —> Булевский;
            ">"     : Целый, Целый     —> Булевский;
            "<="    : Целый, Целый     —> Булевский;
            ">="    : Целый, Целый     —> Булевский;
            Float   : Целый            —> Вещественный; /* аксиомы в определении NEWTYPE Вещественный*/
            Fix     : Вещественный     —> Целый; /* аксиомы в определении NEWTYPE Вещественный*/
        AXIOMS
        FOR ALL a,b,c IN Целый (
            /* отрицание */
        0 — a                          == — a;
        /* сложение */
        0 + a                          == a;
        a + b                          == b + a;
        a + (b + c)                    == (a + b) + c;
        /* вычитание*/
        a — a                          == 0;
        (a — b) — c                    == a — (b + c);
        (a — b) + c                    == (a + c) — b;
        a — (b — c)                    == (a + c) — b;
        /*умножение*/
        a * 0                          == 0;
        a * 1                          == a;
        a * b                          == b * a;
        a * (b * c)                    == (a * b) * c;
        a * (b + c)                    == a * b + a * c;
        a * (b — c)                    == a * b — a * c;
        /*упорядочение*/
        a + 1 > a                      == Истина;
        a — 1 < a                      == Истина;
        /*операция равенства*/
```

```
        (a > b) OR (b > a) == NOT (a = b);
        /* обычные аксиомы порядка*/
        "<" (a,a)                          == Ложь;
        "<" (a,b)                          == ">" (b,a);
        "<=" (a,b)                         == "OR"("<"(a,b),"="(a,b));
        ">="(a,b)                          == "OR"(">"(a,b),"="(a,b));
        "<"(a,b) == Истина                          ==> NOT("<"(b,a)) == Истина;
        "<" (a,b) AND "<" (b,c) == Истина  ==> "<"(a,c) ==Истина;
        /*деление*/
        a/0                                == ERROR!
        a >= 0 AND b > a                   == Истина       ==> a/b == 0;
        a >= 0 AND b <= a AND b > 0 == Истина   ==> a/b == 1 + (a—b)/b;
        a >= 0 AND b < 0 == Истина                  ==> a/b == —(a/(—b));
        a < 0 AND b < 0    == Истина                ==> a/b == (—a)/(—b);
        a < 0 AND b > 0    == Истина                ==> a/b == —((—a)/b);
        /*Литералы с 2 по 9*/
        TYPE Целый 2 == 1 + 1; TYPE Целый 3 == 2 + 1;
        TYPE Целый 4 == 3 + 1; TYPE Целый 5 == 4 + 1;
        TYPE Целый 6 == 5 + 1; TYPE Целый 7 == 6 + 1;
        TYPE Целый 8 == 7 + 1; TYPE Целый 9 == 8 + 1;
    MAP /* Литералы, отличные от литералов с 0 по 9 */
        FOR ALL a,b,c IN Целый LITERALS
        (Spelling(a) == Spelling(b)//Spelling(c),
        Длина (Spelling(c)) == 1 ==>
                        a == b * (9 + 1) + c;
        );
ENBNEWTYPE Целый;
```

5.6.5.2 *Использование*

Целый сорт отвечает целым числам в десятичной записи.

5.6.6 *Натуральный подтип*

5.6.6.1 *Определение*

```
SYNTYPE Натуральный = Целый CONSTANTS > = 0  ENDSYNTYPE Натуральный;
```

5.6.6.2 *Использование*

Натуральный подтип используется, если требуются только положительные целые числа. Все операции являются операциями целого сорта, но когда значение используется как параметр или присваивается, оно повторяется. Отрицательное значение будет ошибкой.

5.6.7 *Вещественный сорт*

5.6.7.1 *Определение*

```
NEWTYPE Вещественный
        LITERALS NAMECLASS (('0':'9')* ('0':'9')) OR (('0':'9')* '-' ('0':'9')+);
        OPERATORS
            "—"  : Вещественный                    —> Вещественный;
            "+"  : Вещественный, Вещественный      —> Вещественный;
            "—"  : Вещественный, Вещественный      —> Вещественный;
            "*"  : Вещественный, Вещественный      —> Вещественный;
            "/"  : Вещественный, Вещественный      —> Вещественный;
```

```
                                        /*
    "="    : Вещественный, Вещественный   —>  Булевский;      Сигнатуры операций "=" и "/="
    "/="   : Вещественный, Вещественный   —>  Булевский;      неявные — см. § 5.4.1.4
                                         */
    "<"    : Вещественный, Вещественный   —>  Булевский;
    ">"    : Вещественный, Вещественный   —>  Булевский;
    "<="   : Вещественный, Вещественный   —>  Булевский;
    ">="   : Вещественный, Вещественный   —>  Булевский;
AXIOMS
FOR ALL a,b,c IN Вещественный (
    /*отрицание*/
    0 — a                                 ==  — a;
    /*сложение*/
    0 + a                                 ==  a;
    a + b                                 ==  b + a;
    a + (b + c)                           ==  (a + b) + c;
    /*вычитание*/
    a — a                                 ==  0;
    (a — b) — c                           ==  a — (b + c);
    (a — b) + c                           ==  (a + c) — b;
    a — (b — c)                           ==  (a + c) — b;
    /*умножение*/
    a * 0                                 ==  0;
    a * 1                                 ==  a;
    a * b                                 ==  b * a;
    a * (b * c)                           ==  (a * b) * c;
    a * (b + c)                           ==  a * b + a * c;
    a * (b — c                            ==  a * b — a *  c;);
    /*упорядочение*/
    FOR ALL, i, j IN Целый (
        Float(i) > Float(j) == TYPE Целый ">"(i,j);
        Float(j) = 0 == Ложь ==> Float(i)/Float(i) > 0 == Float(i) > 0
                                          AND Float(j) > 0 OR Float(i) < 0 AND Float(j) < 0;
        Float(i) > 0 AND Float(j) > AND Float(i) > Float(j)
                                          ==> Float(i)/Float(j) > 1 == Истина; ) ;
        FOR ALL a,r,b IN Вещественный (a + r < b + r == a < b;
            r > 0 ==> a * r < b * r == a < b;
            r < 0 ==> a * r < b * r == b < a;);
        /*обычные аксиомы порядка*/
        FOR ALL a,b,c,d IN Вещественный (
            /*операции равенства и сравнения*/
            (a > b) OR (b > a) == NOT (a = b)
            "<"(a,a)                              ==  Ложь;
            "<"(a,b)                              ==  ">"(b,a);
            "<=(a,b)                              ==  "OR"("<"(a,b),"="(a,b));
            ">="(a,b)                             ==  "OR"("<"(a,b),"="(a,b));
            "<"(a,b) == Истина                    ==> NOT("<"(b,a)) == Истина;
            "<"(a,b) AND "<"(b,c) == Истина       ==> "<"(a,c) == Истина;
        /*деление*/
            a/0                                   ==  ERROR!;
            a = 0 == Ложь ==> a/a                 ==  1;
            a = 0 == Ложь ==> 0/a                 ==  0;
            b = 0 == Ложь ==> (a/b)*b             ==  a;
```

```
            b = 0 OR c = 0 == Ложь ==> (a * b) / (c * b) == a/c;
            b = 0 OR d = 0 == Ложь ==> a/b + c/d == (a * d + b * c) / (b * d);
            b = 0 OR d = 0 == Ложь ==> a/b — c/d == (a * d — b * c) / (b * d);
            b = 0 OR d = 0 == Ложь ==> (a/b) * (c/d) == (a * c) / (b * d);
            b = 0 OR d = 0 == Ложь ==> (a/d) / (c/d) == (a * d) / (b * c););
        /* преобразования целых в вещественные и обратно*/
        FOR ALL a,i,j IN Целый (
        FOR ALL r IN Вещественный (
            Fix(Float(a))                    == a;
            r — 1.0 < Float(Fix(r))          == Истина;
                        /* Замечание: Fix(1.5) == 1, Fix(—0.5) == —1 */
            Float(Fix(r)) <= r               == Истина;
            Float(TYPE Целый "+"(i,j)) == Float(i) + Float(j);));
    MAP
        FOR ALL r,s IN Вещественный LITERALS (
            FOR ALL i,j IN Целый LITERALS (
            Spelling(r) == Spelling(i) ==> r == Float(i);
            Spelling(r) == Spelling(i) ==> i == Fix(r);
            Spelling(r) == Spelling(i)//Spelling(s), Spelling(s) == '.'//Spelling(j)
                                        ==> r == Float(i) + s;
            Spelling(r) == '.'//Spelling(i), Длина (Spelling(i)) == 1 ==> r == Float(i)/10;
            Spelling(r) == '.'//Spelling(i)//Spelling(j), Длина (Spelling(i)) == 1
                Spelling(s) == '.'//Spelling(j)
                                        ==> r == (Float(i) + s)/10;
        )   );
ENDNEWTYPE Вещественный;
```

5.6.7.2 *Использование*

Вещественный сорт используется для представления вещественных чисел. В Вещественном сорте могут быть представлены все числа, представимые как отношение двух целых чисел. Числа, которые не могут быть представлены подобным образом (иррациональные числа, например, $\sqrt{2}$) не являются частью вещественного сорта. Однако для практических инженерных целей обычно можно использовать достаточно точные приближения. Не используя дополнительного аппарата, невозможно определить множество чисел, включающее все иррациональные.

5.6.8 *Генератор Массив*

5.6.8.1 *Определение*

```
GENERATOR Массив (TYPE Индекс, TYPE Сортэлем)
    OPERATORS
        Сделать!          : Сортэлем                    —> Массив;
        Модифицировать    : Массив, Индекс, Сортэлем    —> Массив;
        Извлечь!          : Массив, Индекс              —> Сортэлем;
    AXIOMS
    FOR ALL item, item1, item2, inemi, itemj IN Сортэлем (
    FOR ALL i, j, ipos IN Индекс (
    FOR ALL a, s IN Массив (
        type Массив Извлечь! (Сделать (item,i)) == item;
        Модифицировать! (Модифицировать! (s,i,item1),i,item2) == Модифицировать! (s,i,item2);
        Извлечь! (Модифицировать! (a,ipos,item),ipos) == item;
        i = j == Ложь ==> Извлечь! (Модифицировать! (a,j,item),i) == Извлечь!(a,i);
        i = j == Ложь ==>
            Модифицировать! (Модифицировать! (s,i,itemi),j,itemj) == Модифицировать! (Модифицировать!
                (s,j,itemj),i,itemi);)));
        /* операции равенства*/
```

```
        type Массив Сделать! (item1) = Сделать! (item2) == item1 = item2;
        Модифицировать! (a,i,item) =s  == (Извлечь! (s,i) = item) AND (a=s);
ENDGENERATOR Массив;
```

5.6.8.2 *Использование*

Генератор Массив может использоваться для определения сорта, индексируемого другим сортом. Например,

```
NEWTYPE indexbychar Массив (Знаковый, Целый)
ENDNEWTYPE indexbychar;
```

определяет массив целых чисел, идексируемых знаками.

Массивы обычно используются в сочетании с определенными в § 5.5.3.1 и § 5.4.2.4 сокращенными формами Модифицировать! и Извлечь! для индексации. Например,

```
DCL charvalue indezbychar;
. . . . . .
TASK charvalue('A') := charvalue('B')-1;
```

5.6.9 *Генератор Множественный тип*

5.6.9.1 *Определение*

```
GENERATOR Множественный_ тип (TYPE Сортэлем)
    LITERALS Пустое;
    OPERATORS
        "IN"    : Сортэлем, Множественный_тип -> Булевский
                        /*операция есть элемент*/
        Вкл     : Сортэлем, Множественный_тип -> Множественный_тип;
                        /*включить элемент во множество*/
        Искл    : Сортэлем, Множественный_тип -> Множественный_тип;
                        /*исключить элемент из множества*/
        "<"     : Множественный_тип, Множественный_тип -> Булевский;
                        /*есть собственное подмножество*/
        ">"     : Множественный_тип, Множественный_тип -> Булевский;
                        /*содержит как собственное подмножество*/
        "<="    : Множественный_тип, Множественный_тип -> Булевский;
                        /*есть подмножество*/
        ">="    : Множественный_тип, Множественный_тип -> Булевский;
                        /*содержит*/
        "AND"   : Множественный_тип, Множественный_тип -> Множественный_тип;
                        /*пересечение множеств*/
        "OR"    : Множественный_тип, Множественный_тип -> Множественный_тип;
                        /*объединение множеств*/
    AXIOMS
    FOR ALL i,j IN Сортэлем (
    FOR ALL p,ps,a,a,b,c IN Множественный_тип (
        i IN type Множественный_тип Пустое          == Ложь;
        i IN Вкл (i,ps)                             == Истина;
        i IN ps                                     == i IN Вкл (j,ps);
        type Множественный_тип Искл (i, Пустое)     == Пустое;
        NOT (i IN ps)                               == Искл (i,ps) = ps;
        Искл (i,,Вкл (i,ps))                        == ps;
        i = j == Ложь ==>  Искл (i,Вкл(j,ps))       == Вкл (j,Искл (i,ps));
        Вкл (i,Вкл (j,p))                           == Вкл (j,Вкл (i,p));
        Вкл (i,Вкл (i,p))                           == Вкл (i,p);
        a < b => (i IN a => i IN b)                 == Истина;
        i IN (a AND b)          == TYPE Булевский "AND" (i IN a,i IN b);
        i IN (a OR b)           == TYPE Булевский "OR"(i IN a,i IN b);
        /*операции равенства*/
        Пустое = Вкл (i,ps)     == Ложь;
        Вкл (i,a) = b           == (i IN b) AND (a=Искл (i,b));
        /*обычные аксиомы порядка*/
```

```
        "<"(a,a)                == Ложь;
        "<"(a,b)                == ">"(b,a);
        "<="(a,b)               == "OR"("<"(a,b),"="(a,b));
        ">="(a,b)               == "OR"(">"(a,b),"="(a,b));
        "<"(a,b) == Истина      ==> NOT("<"(b,a)) == Истина;
        TYPE Булевский "AND"("<"(a,b),"<"(b,c)) == Истина ==> "<"(a,c) == Истина; ) )
ENDGENERATOR Множественный_тип;
```

5.6.9.2 *Использование*

Множественный тип используется для представления множеств. Например,

```
NEWTYPE Boolset Множественный_тип (Булевский) ENDNEWTYPE Bolset;
```

может использоваться для переменной, принимающей значения (Истина), (Ложь), (Истина, Ложь) или пустое множество.

5.6.10 *Сорт PId*

5.6.10.1 *Определение*

```
NEWTYPE PId
        LITERALS Нуль;
        OPERATORS       уникальный! : PId -> PId;
                        /*
                        "="     : Pid,Pid -> Булевский; Сигнатуры операций "= и "/=" неявные — см. § 5.4.1.4
                        */
        AXIOMS
        FOR ALL p,p1,p2 IN PId (
            уникальный! (p) = Нуль                  == Ложь
            уникальный! (p1) = уникальный! (p2)     == p1 = p2);
DEFAULT Нуль;
ENDNEWTYPE PId;
```

5.6.10.2 *Использование*

Сорт PId используется для задания идентификаторов процессов. Отметим, что кроме значения Нуль других литералов нет. Когда создается процесс, система использует операцию уникальный! для генерации нового уникального значения.

5.6.11 *Сорт Длительность*

5.6.11.1 *Определение*

```
NEWTYPE Длительность INHERITS Вещественный ("+", "-", ">")
ADDING
        OPERATORS
            "*" : Длительность, Вещественный -> Длительность;
            "/" : Длительность, Вещественный -> Длительность;
        AXIOMS /* далее каждое d должно быть по контексту значением длительности*/
        FOR ALL d, z IN Длительность (
        FOR ALL r IN Вещественный (
            /*операции равенства*/
            (d > z) OR (z > d) == NOT (d = z);
            /* Длительность, умноженная на Вещественный */
            d * 0     == 0;
            0 * r     == 0;
            d * TYPE Вещественный "+"(1,r) == d + (d * r);
```

```
        d * TYPE Вещественный "–" (1,r) == d – (d * r);
        d * TYPE Вещественный "–" (r,1) == (d * r) – d;
        d * TYPE Вещественный "–" (r)   == 0 – (d * r);
        /*Длительность, деленная на Вещественный*/
        d/0  == ERROR!
        r = 0 == Ложь == > d/r == d * TYPE Вещественный "/" (1,r);
        /* то есть деление это то же самое, что и умножение на (вещественное) обратное*/
        r = 0 == Ложь ==> z * r = d == (d/r) = z;));
    MAP
        FOR ALL d IN Длительность LITERALS (
        FOR ALL r IN Вещественный LITERALS ( Spelling(d) == Spelling(r) ==> d == 1 * r))
ENDNEWTYPE Длительность;
```

5.6.11.2 *Использование*

Сорт длительность используется для значений, которые прибавляются к текущему значению времени для установки таймеров. Литералы сорта Длительность такие же, как у Вещественного сорта. Выбор единицы длительности зависит от определяемой системы.

Длительность можно умножать и делить на вещественные числа.

5.6.12 *Сорт Временно́й*

5.6.12.1 *Определение*

```
NEWTYPE Временной INTERITS Вещественный OPERATORS ("<", "<=", ">", ">=") ADDING
    OPERATORS
        "+"  : Временно́й, Длительность      –> Временно́й;
        "–"  : Временно́й, Длительность      –> Временно́й;
        "–"  : Временно́й, Временно́й        –> Длительность;
    AXIOMS
    FOR ALL t, t1, t2 IN Временно́й (
    FOR ALL d, d1, d2 IN Длительность (
        (t1 > t2) OR (t2 > t1)    == NOT (t1 = t2);
        t + 0                     == t;
        t – d                     == t + TYPE Длительность "–" (0,d);
        (t + d1) + d2             == t + TYPE Длительность "+"(d1,d2);
        (t + d1) – (t + d2)       == TYPE Длительность "–" (d1,d2);));
    MAP
        FOR ALL d IN Длительность LITERALS (
        FOR ALL t IN Временно́й LITERALS ( Spelling(d) == Spelling(t) ==> t == 0 = d));
ENDNEWTYPE Временно́й;
```

5.6.12.2 *Использование*

Выражение NOW возвращает значение временно́го сорта. Для получения новых временных значений ко временно́му значению можно прибавлять длительность, и вычитать из него длительность. Разность двух временных значений есть длительность. Временные значения используются для установки сроков активности таймеров.

Начальный момент времени зависит от системы. Единицей времени является время, представляемое прибавлением единицы длительности к значению времени.

Добавление I
(к Рекомендации Z.100)

Формальная модель непараметризованных типов данных[1]

I.1 Многосортные алгебры

Многосортная алгебра A это пара <D,O>, в которой

a) D это множество множеств, и элементы D называются носителями данных (алгебры A); элементы носителя данных dc называются значениями-данных; и

b) O это множество всюду определенных функций, областями определения которых являются декартовы произведения носителей данных A и областью значений — один из носителей данных.

I.2 Семантика определений типов данных

I.2.1 Общие понятия

I.2.1.1 Сигнатура

Сигнатура SIG это пара <S,OP>, в которой

a) S это множество идентификаторов-сортов (называемых также сортами); и

b) OP это множество операций.

Операция состоит из идентификаторов-операции op, списка сортов (аргументов) w с элементами из S и сорта (значения) $s \in S$. Это обычно записывается как op:w→ s. Если w является пустым списком, op:w → s называется нуль-арной операцией или символом константы сорта s.

I.2.1.2 Морфизм сигнатур

Пусть $SIG_1 = <S_1, OR_1>$ и $SIG_2 = <S_2, OP_2>$ — сигнатуры. Морфизм сигнатур $g:SIG_1 \rightarrow SIG_2$ это пара отображений

$$g = <gs:S_1 \rightarrow S_2, gop:OP_1 \rightarrow OP_2>$$

таких что для любых $e\text{-}opid_1 = <opidf_1 . <gs(e\text{—}sidf_1), \ldots,$

$gs(e=sidf_k)>, gs(e\text{-}res), pos> \in OP_1$

$gop(e\text{-}opid_1) = <opidf_2, <(e\text{-}sidf_1), \ldots, (e\text{-}sidf_k)>, (e\text{-}res), pos>$

для некоторого идентификатора-операции $opidf_2$.

---

1) По соглашению между МККТТ и ISO текст настоящего добавления был принят в качестве общего формального описания модели инициальной алгебры для абстрактных типов данных. Этот же текст (с соответствующими терминологическими и типографическими изменениями, а также изменениями в нумерации) приведен в ISO IS8807. §§ I.1, I.2.1.1, I.2.1.2, I.2.1.3, I.2.1.4, I.2.1.5, I.2.1.6, I.3, I.4.1, I.4.2, I.4.3, I.4.4, I.4.5 и I.4.6 настоящего добавления приведены соответственно в §§ 5.2, 7.2.2.1, 7.3.2.8, 7.2.2.2, 7.2.2.3, 7.2.2.4, 7.2.2.5, 4.7, 7.4.2.1, 7.4.2.2, 7.4.3 и 7.4.4 IS 8807. Термины идентификатор-сорта, операция идентификатор-переменной, переменная, алгебраическая спецификация SPEC и операции из настоящего добавления заменены в IS8807 соответственно на переменную-сорта, переменную-операции, переменную-значения, переменную-значения, представление данных pres и функции.

I.2.1.3 Термы

Пусть v — любое множество переменных и пусть <S,OP> — сигнатура. Множества TERM(OP, V, s) **термов** сорта $s \in S$ с операциями из OP и переменными из V определяются индуктивно в три шага:

a) каждая переменная $x:s \in V$ принадлежит TERM(OP, V, s);

b) каждая нуль-арная операция $op \in OP$ с res(op) = s принадлежит TERM(OP, V, s);

c) если термы t_i сортов s_i принадлежат TERM(OP, V, s_i) для i = 1, . . .,n, то для всякого $op \in OP$ с $arg(op) = <s_1, ...,s_n>$ и res(op) = s, $op(t_1, . . .,t_n)$ принадлежит TERM(OP, V, s).

Если терм t является элементом TERM(OP, V, s), то s называется сортом t и обозначается sort(t). Множество TERM(OP, s) **пассивных термов** сорта $s \in S$ определяется как множество TERM(OP, V, s).

I.2.1.4 Равенства

Равенство сорта s, связанное с сигнатурой <S,OP> — это тройка <V,L,R>, в которой

a) V это множество идентификаторов-переменных;

b) $L,R \in T(OP, V, s)$; и

c) $s \in S$.

Равенство e' = < { } ,L',R'> является пассивным экземпляром равенства e = <V,L,R>, если L' и R' могут быть получены из L и R следующим образом: для каждой переменной v:s из V все ее вхождения в L и R заменяются одним и тем же пассивным термом сорта s.

Нотация L=R используется для пассивного экземпляра <{ },L,R> равенства.

Примечания. — Равенство <V,L,R> также может быть записано в форме L=R, если это не приводит к семантическим затруднениям.

I.2.1.5 Условные равенства

Условное равенство сорта s, связанное с сигнатурой <S,OP> это тройка <V,Eq,e>, в которой

a) V это множество идентификаторов-переменных;

b) Eq это множество равенств, связанных с <S,OP>, с переменными из V; и

c) e это равенство сорта s, связанное с <S,OP>, с переменными из V.

I.2.1.6 Алгебраические спецификации

Алгебраическая спецификация SPEC это тройка <S,OP,E>, в которой

a) <S,OP> это сигнатура; и

b) E это множество условных равенств, связанных с <S,OP>.

I.3 Аксиоматические системы

Аксиоматическая система — это тройка $D = <A, Ax, I>$, в которой

a) A это множество, элементы которого называются утверждениями;

b) $A \supseteq Ax$ — множество аксиом; и

c) I это множество правил вывода.

Каждое правило вывода $R \in I$ имеет следующий формат:

$$R: \frac{P_1, \ldots, P_n}{Q},$$

где $P_1, \ldots, P_n, Q \in A$.

Вывод утверждения P в аксиоматической системе D — это конечная последовательность s утверждений, удовлетворяющая следующим условиям:

a) последним элементом s является P;

b) если Q является элементом s, то либо $Q \in Ax$, либо существует правило $R \in I$

$$R: \frac{P_1, \ldots, P_n}{Q},$$

в котором $P_1, \ldots, P_n$ — элементы s, предшествующие Q.

Если в (аксиоматической системе D существует вывод P, это записывается следующим образом: $D \vdash P$. Если D однозначно определяется по контексту, можно использовать сокращенную запись $\vdash P$.

I.4 Семантика алгебраических спецификаций

Всякий раз, когда в § I.4 встречается множество сортов S, множество операций OP или множество равенств E, имеется в виду заданная алгебраическая спецификация $SPEC = <S,OP,E>$, определенная в § I.2.1.6.

Для того чтобы определить семантику алгебраической спецификации SPEC, используется аксиоматическая система, ассоциированная со SPEC. Эта система определяется в §§ I.4.1.—I.4.3. С помощью этой аксиоматической системы в § I.4.4 и § I.4.5 определяются соотношения на множестве пассивных термов, связанных с $<S,OP,E>$, и классы конгруэнтности. Эти соотношения используются в § I.4.6 для определения алгебры (см. § I.1), представляющей тип данных, специфицированный посредством $<S,OP,E>$.

I.4.1 Аксиомы, генерируемого равенствами

Пусть ceq — условное равенство. Множество аксиом, генерируемое ceq, обозначается Ax(ceq) и определяется следующим образом:

a) если $ceq = <V,Eq,e>$ и $Eq \neq \{\}$, то $Ax(ceq) = \{\}$; и

b) если $ceq = <V,\{\},e>$, то Ax(ceq) является множеством всех пассивных экземпляров e (см. § I.2.1.3).

I.4.2 Правила вывода, генерируемые равенствами

Пусть ceq — условное равенство. Множество правил вывода, генерируемого ceq, обозначается Inf(ceq) и определяется следующим образом:

a) если ceq = $<V,\{\},e>$, то Inf(ceq) = $\{\}$; и

b) если ceq = $<V,\{e_1,\ldots,e_n\},e>$, где $n > 0$, то Inf(ceq) содержит все правила вида

$$\frac{e_1',\ldots,e_n'}{e'},$$

где $e_1',\ldots,e_n',e'$ являются пассивными экземплярами $e_1,\ldots,e_n$, e соответственно, получаемыми следующим образом: для каждой переменной x из V все ее вхождения в $e_1,\ldots,e_n$, e заменяются одним и тем же пассивным термом сорта sort(x).

I.4.3 Генерируемые аксиоматические системы

Аксиоматическая система D = $<A,Ax,I>$ (см. § I.3), генерируемая алгебраической спецификацией SPEC = $<S,OP,E>$, определяется следующим образом:

a) A это множество всех пассивных экземпляров равенств, связанных с $<S,OP>$;

b) $Ax = \cup\ Ax(ceq) \mid ceq \in E \cup ID$,
где $ID = \{\ t = t \mid t$ пассивный терм$\}$; и

c) $I = \cup\{\ Inf(ceq) \mid ceq \in E\} \cup SI$,
где SI задается следующими схемами

i) $\dfrac{t_1 = t_2}{t_2 = t_1}$ для всех пассивных термов t_1, t_2; и

ii) $\dfrac{t_1 = t_2, t_2 = t_3}{t_1 = t_3}$ для всех пассивных термов t_1, t_2, t_3; и

iii) $$\frac{t_1 = t_1',\ldots,t_n = t_n'}{op(t_1,\ldots,t_n) = op(t_1',\ldots,t_n')}$$
для всех операций $op:s_1,\ldots,s_n \rightarrow s \in OP$ с $n > 0$, и всех пассивных термов t_i, t_i' сортов s_i для $i = 1,\ldots,n$.

I.4.4 Отношение конгруэнтности, генерируемое алгебраической спецификацией

Пусть D — аксиоматическая система, генерируемая алгебраической спецификацией SPEC = $<S,OP,E>$. Два пассивных терма t_1 и t_2 называются **конгруэнтными** относительно SPEC, что обозначается $t_1 \equiv_{SPEC} t_2$, в том и только в том случае, когда

$$D \vdash t_1 = t_2.$$

I.4.5 Классы конгруэнтности

Класс SPEC-конгруэнтности [t] пассивного терма t — это множество всех термов, конгруэнтных t относительно SPEC, то есть

$$[t] = \{t' \mid t \equiv_{SPEC} t'\} .$$

I.4.6 Алгебра фактор-термов

Семантической интерпретацией алгебраической спецификации SPEC-<S,OP,E> является следующая многосортная алгебра Q = <Dq,Oq>, называемая **алгеброй фактор-термов**, где

a) Dq это множество $\{ Q(s) \mid s \in S \}$, где
$Q(s) = \{ [t] \mid t$ пассивный терм сорта $s \}$ для каждого $s \in S$; и

b) Oq это множество операций $\{ op' \mid op \in OP \}$, где op' определяется следующим образом:
$op'([t_1], \ldots, [t_n]) = [op(t_1, \ldots, t_n)]$.

Приложения A, B, C и E

к Рекомендации Z.100

ЯЗЫК ФУНКЦИОНАЛЬНОЙ СПЕЦИФИКАЦИИ И ОПИСАНИЯ (SDL)

PAGE INTENTIONALLY LEFT BLANK

PAGE LAISSEE EN BLANC INTENTIONNELLEMENT

ПРИЛОЖЕНИЕ A

(к Рекомендации Z.100)

Глоссарий SDL

Рекомендация Z.100 содержит формальные определения терминологии SDL. Глоссарий SDL составлен с целью помочь новым пользователям SDL при чтении Рекомендации и приложений к ней; в глоссарии приводится краткое определение термина и указывается раздел Рекомендации, в котором этот термин определен. Определения, приводимые в глоссарии, могут обобщать или видоизменять формальные определения и потому могут быть неполными.

Термины, входящие в определение, могут быть найдены в глоссарии. Если словосочетание набрано курсивом, например *идентификатор процедуры,* но его нет в глоссарии, это означает, что указанное словосочетание является конкатенацией двух терминов, например, в приведенном случае это термин *идентификатор,* за которым следует термин *процедура.* Если слово набрано курсивом, но не может быть найдено в глоссарии, то оно может быть дериватом термина, имеющегося в глоссарии. Например, *экспортированный* — это дериват термина *экспорт.*

За исключением тех случаев, когда термин является синонимом другого термина, после определения состоит основная ссылка на то место, где в Рекомендации Z.100 используется этот термин. Эти ссылки приведены после определения и заключены в квадратные скобки [ ]. Например, [3.2] означает, что основная ссылка делается на § 3.2.

абстрактный тип данных

англ. : abstract data type
исп. : tipo abstracto de datos
фр. : type abstrait de données

Абстрактный тип данных является синонимом термина *тип данных.* Все *типы данных SDL* являются *абстрактными типами данных.*

абстрактная грамматика

англ. : abstract grammar
исп. : gramática abstracta
фр. : grammaire abstraite

Абстрактная грамматика определяет семантику SDL. Абстрактная грамматика описывается средствами *абстрактного синтаксиса* и *правилами корректности* [1.2, 1.4.1].

абстрактный синтаксис

англ. : abstract syntax
исп. : sintaxis abstracta
фр. : syntaxe abstraite

Абстрактный синтаксис является средством описания концептуальной структуры *спецификаций SDL* в сопоставлении с *конкретными синтаксисами,* существующими для любого *конкретного синтаксиса SDL,* а именно SDL/GR и SDL/PR [1.2].

доступ

англ. : access
исп. : acceder
фр. : accès

Доступ является операцией, применяемой к *переменной;* результатом операции является то *значение,* которое было присвоено *переменной* последним. *Доступ* к *переменной,* имеющей *неопределенное значение,* приводит к *ошибке.*

действие

англ. : action
исп. : acción
фр. : action

Действие является операцией, выполняемой в *цепочке перехода,* например *работа, вывод, принятие решения, запрос на создание* или *вызов процедуры* [2.7].

активный таймер

англ.: active timer
исп. : temporizador activo
фр. : temporisateur actif

Активным таймером является такой *таймер, сигнал таймера* от которого находится во *входном порту процесса,* являющегося обладателем *таймера,* либо это такой таймер, для которого установлено, что он должен выработать *сигнал таймера* в некоторый будущий момент времени [2.8.2, 5.5.4.5].

фактический параметр

англ. : actual parameter
исп. : parámetro efectivo
фр. : paramétre rèel

Фактический параметр является *выражением*, которое передается *процессу* или *процедуре* на место соответствующего *формального параметра* при *создании процесса* или *процедуры* (или при их *вызове*). Следует отметить, что в некоторых случаях вызова процедуры *фактический параметр* должен быть *переменной* (то есть частным случаем *выражения*; см. IN/OUT) [2.7.2, 2.7.3, 4.2.2].

список фактических параметров

англ. : actual parameter list
исп. : lista de parámetros efectivos
фр. : liste de paramétres rèels

Список фактических параметров является списком *фактических параметров*. *Фактические параметры* сопоставляются по порядку расположения с элементами соответствующего *списка формальных параметров*.

область

англ. : area
исп. : área; zona
фр. : zone

Область является двумерным регионом в *конкретном графическом синтаксисе*. *Области* часто соответствуют *вершинам* в *абстрактном синтаксисе* и обычно содержат *общий текстовой синтаксис*. В *диаграммах взаимодействия* области могут быть соединены *каналами* или *маршрутами сигналов*. В *диаграммах потока управления области* могут быть соединены *линиями потока*.

массив

англ. : array
исп. : matriz
фр. : tableau (array)

Массив является предопределенным *генератором*, используемым для введения понятия массив и облегчающим определение конкретных массивов.

присваивание

англ. : assign
исп. : asignar
фр. : affectation

Присваивание является операцией, применяемой к *переменным;* она связывает *переменную* со *значением*, заменяя *значение*, которое было связано с *переменной* до этого [5.5.3].

предложение присваивания

англ. : assignment statement
исп. : sentencia de asignación
фр. : instruction d'affectation

Предложение присваивания является предложением, присваивающим *переменной значение* [5.5.3].

область ассоциирования

англ. : association area
исп. : área de asociación
фр. : zone d'association

Область ассоциирования является средством связывания *областей* в *диаграммах взаимодействия* с помощью *символа* ассоциирования. Существует пять областей ассоциирования: *область ассоциирования подструктуры канала, область ассоциирования ввода, область ассоциирования приоритетного ввода, область ассоциирования непрерывного сигнала* и *область ассоциирования сохранения* [2.6.3, 3.2.3, 4.10.2, 4.11].

аксиома

англ. : axiom
исп. : axioma
фр. : axiome

Аксиома является специальным видом *равенства*, в котором подразумевается эквивалентность *Булевскому литералу* True. Термин *аксиомы* используется в качестве синонима выражения *аксиомы и равенства* [5.1.3].

базисный SDL

англ. : basic SDL
исп. : LED básico
фр. : LDS le base

Базисный SDL является подмножеством *SDL*, определенным в § 2 Рекомендации Z.100.

поведение

англ. : behaviour
исп. : comportamiento
фр. : comportement

Поведением или *функциональным поведением системы* является совокупность последовательностей ответов на последовательности стимулов [1.1.3].

блок

англ. : block
исп. : bloque
фр. : bloc

Блок является частью *системы* или родительского *блока*. Термин *блок*, используемый самостоятельно, является синонимом термина *экземпляр блока*. *Блок* является *единицей подразделения* и обеспечивает статический интерфейс [2.4.3].

область блока

англ. : block area
исп. : área de bloque
фр. : zone de bloc

Область блока является или определением *блока*, или ссылкой на *блок* в *диаграмме взаимодействия* [2.4.2].

определение блока

англ. : block definition
исп. : definición de bloque
фр. : définition de bloc

Определение блока является определением *блока* в *SDL/PR* [2.4.2].

диаграмма блока

англ. : block diagram
исп. : diagrama de bloque
фр. : diagramme de bloc

Диаграмма блока является определением *блока* в *SDL/GR* [2.4.3].

подструктура блока

англ. : block substructure
исп. : subestructura de bloque
фр. : sous-structure de bloc

Подструктурой блока является *разбиение блока* на *подблоки* и новые *каналы* на более низком *уровне абстракции* [3.2.2].

определение подструктуры блока

англ. : block substructure definition
исп. : definición de subestructura de bloque
фр. : définition de sous-structure de bloc

Определение подструктуры блока является изображением в *SDL/PR подструктуры блока* некоторого *блока*, подвергшегося *разбиению* [3.2.2].

диаграмма подструктуры блока

англ. : block substructure diagram
исп. : diagrama de subestructura de bloque
фр. : diagramme de sous-structure de bloc

Диаграмма подструктуры блока является изображением в *SDL/GR подструктуры* блока некоторого *блока*, подвергшегося *разбиению* [3.2.2].

диаграмма дерева блоков

англ. : block tree diagram
исп. : diagrama de árbol de bloques
фр. : diagramme d'arborescence de bloc

Диаграмма дерева блоков является вспомогательным документом *SDL/GR*, представляющим *разбиение системы* на *блоки* на более низких *уровнях абстракции* с помощью перевернутой диаграммы дерева (то есть диаграммы, в которой родительский *блок* расположен над потомком) [3.2.2].

БНФ (Форма Бэкуса-Наура)

англ. : BNF (Backus-Naur Form)
исп. : FBN (forma Backus-Naur)
фр. : forme BNF (Backus-Naur Form)

БНФ (Форма Бэкуса-Наура) является формальной нотацией, используемой для выражения *конкретного текстового синтаксиса* языка. Расширенная форма *БНФ* используется для изображения *конкретного графического синтаксиса* [1.5.2, 1.5.3].

Булевский

англ. : boolean
исп. : booleano
фр. : booléen

Булевский является *сортом*, определенным в предопределенном *частичном определении типа;* этому сорту принадлежат значения True (Истина) и False (Ложь). Для *Булевского сорта* предопределенными операциями являются NOT, AND, OR, XOR и импликация [5.6.1].

канал

англ. : channel
исп. : canal
фр. : canal

Канал является связью, передающей *сигналы* между двумя *блоками.* Кроме того, с помощью *каналов сигналы* передаются между *блоками* и *окружающей средой. Каналы* могут быть односторонними или двусторонними [2.5.1].

определение канала

англ. : channel definition
исп. : definición de canal
фр. : définition de canal

Определение канала является определением *канала* в *SDL/PR* [2.5.1].

область определения канала

англ. : channel definition area
исп. : área de definición de canal
фр. : zone de définition de canal

Область определения канала является определением *канала* в *SDL/GR* [2.5.1].

подструктура канала

англ. : channel substructure
исп. : subestructura de canal
фр. : sous-structure de canal

Подструктура канала является *разбиением канала* на совокупность *каналов* и *блоков* на более низком *уровне абстракции* [3.2.3].

определение подструктуры канала

англ. : channel substructure definition
исп. : definición de subestructura de canal
фр. : définition de sous-structure de canal

Определение подструктуры канала является определением *подструктуры канала* в *SDL/PR* [3.2.3].

диаграмма подструктуры канала

англ. : channel substructure diagram
исп. : diagrama de subestructura de canal
фр. : diagramme de sous-structure de canal

Диаграмма подструктуры канала является определением *подструктуры канала* в *SDL/GR* [3.2.3].

знаковый

англ. : character
исп. : carácter; character
фр. : caractère (character)

Знаковый является *сортом*, определенным в предопределенном *частичном определении типа; значениями* этого сорта являются элементы алфавита № 5 МККТТ (например, 1, A, B, C и т.д.). Для *знакового сорта* предопределенными являются *операции упорядочения* [5.6.2].

знакостроковый

англ. : chartstring
исп. : cadena-de-caracteres; chartstring
фр. : chaine de caractares (character string)

Знакостроковый является *сортом*, определенным в предопределенном *частичном определении типа; значениями* этого сорта являются *строки*, составленные из *знаков*. *Операциями* этого *сорта* являются *операции* предопределенного *генератора* для *строки*, конкретизированные сортом *знаковый* [5.6.4].

комментарий

англ. : comment
исп. : comentario
фр. : commentaire

Комментарий является информацией, дополняющей или уточняющей *спецификации SDL*. В SDL/GR *комментарии* могут быть подсоединены к некоторому *символу* с помощью пунктирной линии. В *SDL/PR комментарии* вводятся с помощью ключевого слова COMMENT. Смысл *комментариев* в *SDL* не определяется. См. также *примечание* [2.2.6].

общая текстовая грамматика

англ. : common textual grammar
исп. : gramática textual común
фр. : grammaire textuelle commune

Общая текстовая грамматика является подмножеством *конкретной текстовой грамматики*, используемой и в *SDL/GR* и в *SDL/PR* [1.2].

путь связи

англ. : communication path
исп. : trayecto de comunicación
фр. : trajet de communication

Путь связи – это транспортное средство, переносящее *экземпляры сигналов* от одного *экземпляра процесса* или от *окружающей среды* к другому *экземпляру процесса* или к *окружающей среде*. *Путь связи* состоит или из пути (путей) *канала* или из пути (путей) *маршрута сигнала*, или из их комбинации [2.7.4].

полная совокупность допустимых входных сигналов

англ. : complete valid input signal set
исп. : conjunto completo de señales de entrada válidas
фр. : ensemble complet de signaux d'entrée valides

Полной совокупностью допустимых входных сигналов процесса является объединение *совокупности допустимых входных сигналов*, локальных *сигналов*, *сигналов* от *таймера* и *неявных сигналов процесса* [2.4.4].

конкретная грамматика

англ. : concrete grammar
исп. : gramática concreta
фр. : grammaire concrète

Конкретной грамматикой является объединение *конкретного синтаксиса с правилами корректности* для этого *конкретного синтаксиса. SDL/GR и SDL/PR* являются *конкретными грамматиками SDL.* Для определения их *семантики конкретные грамматики* отображаются на *абстрактную грамматику* [1.2].

конкретная графическая грамматика

англ. : concrete graphical grammar
исп. : gramática gráfica concreta
фр. : grammaire graphic concrète

Конкретная графическая грамматика является *конкретной грамматикой* графической части *SDL/GR* [1.2].

конкретный графический синтаксис

англ. : concrete graphical syntax
исп. : sintaxis gráfica concreta
фр. : syntaxe graphique concrète

Конкретный графический синтаксис является *конкретным синтаксисом* графической части *SDL/GR.* В Рекомендации Z.100 *конкретный графический синтаксис* выражен с помощью расширенной формы *БНФ* [1.2, 1.5.3].

конкретный синтаксис

англ. : concrete syntax
исп. : sintaxis concreta
фр. : syntaxe concrète

Конкретным синтаксисом различных представлений *SDL* является набор фактических *символов,* используемых для представления *SDL,* а также взаимосвязь между этими *символами,* определяемые синтаксическими правилами *SDL.* В Рекомендации Z.100 используются два *конкретных синтаксиса: конкретный текстовой синтаксис* и *конкретный графический синтаксис* [1.2].

конкретный текстовой синтаксис

англ. : concrete textual syntax
исп. : sintaxis textual concreta
фр. : syntaxe textuelle concrète

Конкретный текстовой синтаксис является *конкретным синтаксисом* для *SDL/PR* и текстовых частей *SDL/GR.* В Z.100 *конкретный текстовой синтаксис* выражен с помощью *БНФ* [1.2, 1.5.2].

условное выражение

англ. : conditional expression
исп. : expresión condicional
фр. : expression conditionnelle

Условным выражением является *выражение,* содержащее *Булевское выражение* и управляющее тем, которое из *выражений* должно интерпретироваться: непосредственно следующее или альтернативное *выражение* [5.5.2.3].

соединять

англ. : connect
исп. : conectar
фр. : connect

Соединять — это указывать присоединение *канала* к одному или нескольким *маршрутам сигналов* [2.5.3].

коннектор

англ. : connector
исп. : conector
фр. : connecteur

Коннектор является *символом SDL/GR*; используются *входные коннекторы* и *выходные коннекторы.* Подразумевается, что от *входного коннектора* к соответствующему *выходному коннектору,* идет неявная *линия потока.* Под соответствующим коннектором понимается *входной коннектор,* находящийся в том же *процессе* или той же *процедуре* и имеющий то же *имя,* что и исходный *коннектор* [2.6.6].

согласованное подмножество разбиения

англ. : consistent partitioning subset
исп. : subconjunto de partición consistente
фр. : sous-ensemble de subdivision cohérent

Согласованное подмножество разбиения является совокупностью *блоков* и *подблоков спецификации системы*, обеспечивающее полное представление о *системе*, включая части, связанные с соответствующим *уровнем абстракции*. Следовательно, если *блок* или *подблок* входит в *согласованное подмножество разбиения*, то вместе с ним в это подмножество входят его предки и все его братья и сестры [3.2.1].

согласованное подмножество уточнения

англ. : consistent refinement subset
исп. : subconjunto de refinamiento consistente
фр. : sous-ensemble de raffinement cohérent

Согласованное подмножество уточнения является *согласованным подмножеством разбиения*, содержащим все *блоки* и *подблоки*, использующие *сигналы*, которые используются любым из *блоков* или *подблоков* [3.3].

непрерывный сигнал

англ. : continuous signal
исп. . señal continua
фр. : signal continu

Непрерывный сигнал является средством, определяющим тот факт, что если ассоциированное с некоторым *состоянием Булевское выражение* приняло значение True, то должен интерпретироваться *переход*, следующий за *непрерывным сигналом* [4.11].

диаграмма потока управления

англ. : control flow diagram
исп. : diagrama de flujo de control
фр. : diagramme de liaison de contrôle

Диаграмма потока управления является либо *диаграммой процесса*, либо *диаграммой процедуры*, либо *диаграммой сервиса*.

создание

англ. : create
исп. : crear
фр. : criér

Создание является синонимом *запроса на создание*.

запрос на создание

англ. : create request
исп. : petición de crear
фр. : demande de création

Запрос на создание является *действием*, вызывающим создание и запуск нового *экземпляра процесса*; для этого в качестве шаблона используется *тип процесса*. *Фактические параметры запроса на создание* заменяют *формальные параметры процесса* [2.7.2].

область линии создания

англ. : create line area
исп. : área de línea de crear
фр. : zone de ligne de création

В диаграмме блока область линии создания связывает *область процесса* того *процесса*, который *создает (PARENT)* новый *процесс*, *с областью процесса* вновь *созданного процесса (OFFSPRING)* [2.4.3].

тип данных

англ. : data type
исп. : tipo de datos
фр. : type de données

Тип данных является определением совокупности *значений (сортов)*, совокупности *операций*, применяемых к этим *значениям*, и совокупности алгебраических правил *(равенств)*, определяющих *поведение операций* при их применении к *значениям* [2.3.1].

определение типа данных

англ. : data type definition
исп. : definición de tipo de datos
фр. : définition de type de données

Определение типа данных определяет допустимость *выражений* и отношения между *выражениями* в любом месте *спецификации SDL* [5.2.1].

принятие решения

англ. : decision
исп. : decisión
фр. : décision

Принятие решения является *действием* в *переходе*, задающим вопрос, ответ на который может быть получен в данный момент; в соответствии с ответом выбирается один из выходящих из *принятия решения переходов*, продолжая тем самым интерпретацию [2.7.5].

область принятия решения

англ. : decision area
исп. : área de decisión
фр. : zone de décision

Область принятия решения является изображением *принятия решения* в *SDL/GR* [2.7.5].

по умолчанию

англ. : default
исп. : por defecto
фр. : défault

Присваивание по умолчанию является обозначением *значения*, которое изначально ассоциируется с каждой *переменной* данного *сорта* в клаузуле *умолчания*. Клаузула *умолчания* может входить в *определения типов данных* [5.5.3.3].

описание

англ. : description
исп. : descripción
фр. : description

Описание системы является описанием ее фактического *поведения* [1.1].

диаграмма

англ. : diagram
исп. : diagrama
фр. : diagramme

Диаграмма является изображением в *SDL/GR* части *спецификации* [2.4.2].

длительность

англ. : duration
исп. : duración; duration
фр. : durée (duration)

Длительность является *сортом*, определенным в предопределенном *частичном определении типа; значения* этого *сорта* обозначают с помощью *вещественного сорта;* эти *значения* изображают интервал между двумя моментами времени [5.6.11].

разрешающее условие

англ. : enabling condition
исп. : condición habilitante (o habilitadora)
фр. : condition de validation

Разрешающее условие является средством, обеспечивающим условное восприятие *сигнала* в качестве *входа* [4.12].

область разрешающего условия

англ. : enabling condition area
исп. : área de condición habilitante (o habilitadora)
фр. : zone de condition de validation

Область разрешающего условия является изображением *разрешающего условия* в *SDL/GR* [4.12].

класс объектов

англ. : entity class
исп. : clase de entidad
фр. : classe d'entité

Класс объектов является объединением в одну категорию *типов SDL*; объединение основывается на сходстве использования *типов* [2.2.2].

окружающая среда

англ. : environment
исп. : entorno
фр. : environnement

Термин *окружающая среда* является синонимом термина *окружающая среда системы*. Кроме того, если это допускается контекстом, данный термин может быть синонимом терминов *окружающая среда блока, процесса, процедуры* или *сервиса* [1.3.2].

окружающая среда системы

англ. : environment of a system
исп. : entorno de un sistema
фр. : environnement d'un système

Окружающей средой системы является весь внешний мир специфицируемой *системы*. Окружающая среда взаимодействует с *системой*, посылая ей и получая от нее *экземпляры сигналов* [1.3.2].

равенство

англ. : equation
исп. : ecuación
фр. : équation

Равенство является таким отношением между *термами* одного и того же *сорта*, которое выполняется для всевозможных *значений*, подставляемых вместо всех *идентификаторов значений*, входящих в *равенство*. *Равенство* может быть *аксиомой* [5.1.3, 5.2.3].

ошибка

англ. : error
исп. : error
фр. : erreur

Ошибка возникает в процессе интерпретации допустимой *спецификации системы*, если при этом нарушается одно из динамических условий *SDL*. После обнаружения *ошибки SDL* не определяет дальнейшее *поведение системы* [1.3.3].

экспорт

англ. : export
исп. : exportación
фр. : export

Термин *экспорт* является синонимом термина *экспортная операция*.

экспортируемая переменная

англ. : exported variable
исп. : variable exportada
фр. : variable exportée

Экспортируемая переменная является *переменной*, которая может быть использована в *экспортной операции* [4.13].

экспортер

англ. : exporter
исп. : exportador
фр. : exportateur

Экспортером переменной является *процесс*, который владеет *переменной* и *экспортирует* ее *значение* [4.13].

экспортная операция

англ. : export operation
исп. : operación de exportación
фр. : opération d'exportation

Экспортная операция является операцией, с помощью которой *экспортер* раскрывает *значение переменной*. См. *импортная операция* [4.13].

выражение

англ. : expression
исп. : expresión
фр. : expression

Выражение является либо *литералом*, либо применением *операции*, либо *синонимом*, либо *доступом к переменной*, либо *условным выражением*, либо *императивной операцией*, примененной к одному или нескольким *выражениям*. В результате интерпретации выражения выдается значение (либо в *системе* обнаружена *ошибка*) [2.3.4 и 5.4.2.1].

внешний синоним

англ. : external synonym
исп. : sinónimo externo
фр. : synonyme externe

Внешний синоним предопределенного *сорта*, *значение* которого не специфицировано в *спецификации системы* [4.3.1].

извлечь!

англ. : extract!
исп. : extraer!; extract!
фр. : extract!

Извлечь является *операцией*, которая подразумевается в *выражении* в случаях, когда непосредственно за *переменной* следует (следуют) *выражение (выражения)*, взятое (взятые) в скобки [5.4.2.4 и 5.6.8].

линия потока

англ. : flow line
исп. : linea de flujo
фр. : ligne de liaison

Линия потока является *символом*, используемым для соединения *областей* в *диаграммах потока управления* [2.2.4 и 2.6.7.2.2].

формальный параметр

англ. : formal parameter
исп. : parámetro formal
фр. : paramètre formel

Формальный параметр является *именем переменной*, которому присваиваются *фактическое значение*, либо которое заменяется на *фактические переменные* [2.4.4, 2.4.5, 4.2 и 4.10].

список формальных параметров

англ. : formal parameter list
исп. : lista de parámetros formales
фр. : liste de paramètres formels

Список формальных параметров является списком *формальных параметров*.

функциональное поведение

англ. : functional behaviour
исп. : comportamiento funcional
фр. : comportement fonctionnel

Термин функциональное поведение является синонимом термина *поведение.*

общая область выбираемого определения

англ. : general option area
исп. : área de opción general
фр. : zone d'option générale

Общая область выбираемого определения является изображением *выбираемого определения* в *SDL/GR* [4.3.3].

общие параметры

англ. : general parameters
исп. : parámetros generales
фр. : paramètres généraux

Общие параметры как в *спецификации системы,* так и в ее *описании* имеют отношение к таким характеристикам, как температурные пределы, конструкция, пропускная способность, класс обслуживания и т.д. Эти параметры средствами *SDL* не описываются [1.1].

генератор

англ. : generator
исп. : generador
фр. : générateur

Генератор является неполным описанием *нового типа.* Прежде чем *генератор* получает статус *нового типа,* должна быть осуществлена его конкретизация; это достигается поставкой недостающей информации [5.4.1.1.2].

граф

англ. : graph
исп. : gráfico
фр. : graphe

В *абстрактном синтаксисе граф* является частью *спецификации средствами SDL как, например, граф процесса* или *граф процедуры.*

пассивное выражение

англ. : ground expression
исп. : expresión fundamental
фр. : expression close

Пассивным выражением является *выражение,* содержащее только *операции, синонимы* и *литералы* [5.4.2.2].

иерархическая структура

англ. : hierarchical structure
исп. : estructure jerárquica
фр. : structure hiérarchique

Иерархической структурой является такая структура *спецификации системы,* в которой *разбиение* и *уточнение* обеспечивают различные обзоры *системы* на различных *уровнях абстракции. Иерархическая структура* делает доступным *спецификацию* сложных систем. См. также *диаграмму дерева блоков* [3.1].

идентификатор

англ. : identifier
исп. : identificador
фр. : identificateur

Идентификатор является средством уникальной идентификации объекта и состоит из *квалификационной части* и *имени* [2.2.2].

императивная операция

англ. : imperative operator
исп. : operador imperativo
фр. : opérateur impératif

Императивной операцией является *выражение* "сейчас", *выражение обозревания, выражение активного таймера, импортирующее выражение* или одно из *PId-выражений: SELF, PARENT, OFFSPRING* или *SENDER* [5.5.4].

неявный переход

англ. : implicit transition
исп. : transición implícita
фр. : transition implicite

В *конкретном синтаксисе неявный переход* инициируется *сигналом*, входящим в *полную совокупность допустимых входных сигналов*, но не специфицированным во *входах* или *сохранении* некоторого *состояния*. *Неявный переход* не содержит никакого *действия* и непосредственно приводит обратно в то же *состояние* [4.6].

импорт

англ. : import
исп. : importación
фр. : import

Термин *импорт* является синонимом термина *импортная операция*.

импортируемая переменная

англ. : imported variable
исп. : variable importada
фр. : variable importée

Импортируемая переменная является *переменной*, используемой в *импортной операции* [4.13].

импортер

англ. : importer
исп. : importador
фр. : importeur

Импортером импортируемой переменной является *экземпляр процесса, импортирующий значение* [4.13].

импортная операция

англ. : import operation
исп. : operación de importación
фр. : opération d'importation

Импортной операцией является операция, результатом которой является выдача *значения экспортируемой переменной* [4.13].

IN-переменная

англ. : IN variable
исп. : variable IN
фр. : variable "IN"

IN-переменная является атрибутом *формального параметра*, определяющим тот случай, когда *значение* передается *процедуре* с помощью *фактического параметра* [2.4.5].

IN/OUT-переменная

англ. : IN/OUT variable
исп. : variable IN/OUT
фр. : variable "IN/OUT"

IN/OUT-переменная является атрибутом *формального параметра*, определяющим тот случай, когда *имя формального параметра* используется в качестве синонима *переменной* (то есть *фактический параметр* должен быть *переменной*) [2.4.5].

входной коннектор

англ. : in-connector
исп. : conector de entrada
фр. : connecteur d'entrée

Входной коннектор является *коннектором.*

инфиксная операция

англ. : infix operator
исп. : operador infijo
фр. : opérateur infixe

Инфиксной операцией является одна из предопределенных *двуместных операций SDL* (=>, OR, XOR, AND, IN, /=, =, >, <, <=, >=, +, –, //, *, /, MOD, REM), помещаемых между двумя его аргументами [5.4.1.1].

неформальный текст

англ. : informal text
исп. : texto informal
фр. : texte informel

Неформальным текстом является такой текст, включаемый в *спецификации SDL,* для которого *семантика* определяется не средствами *SDL,* а с помощью какой-то другой модели. *Неформальный текст* заключается в апострофы [2.2.3].

инициальная алгебра

англ. : initial algebra
исп. : álgebra inicial
фр. : algèbre initiale

Инициальная алгебра является формализмом, используемым для определения *абстрактного типа данных* [5.3].

входная точка

англ. : inlet
исп. : acceso de entrada
фр. : accès entrant

Входная точка представляет линию, такую как *канал* или *линия потока,* входящую в *вызов макроса* в *SDL/GR* [4.2.3].

ввод

англ. : input
исп. : entrada
фр. : entrée

Ввод является восприятием *сигнала* из *входного порта,* запускающим *переход.* В процессе восприятия *сигнала значения,* ассоциированные с *сигналом,* становятся доступными *экземпляру процесса* [2.6.4 и 4.10.2].

область ввода

англ. : input area
исп. : área de entrada
фр. : zone d'entrée

Область ввода является *SDL/GR*-изображением *ввода* [2.6.4].

входной порт

англ. : input port
исп. : puerto de entrada
фр. : port d'entrée

Входной порт процесса является очередью, которая получает и задерживает *сигналы* в порядке их поступления до тех пор, пока *сигналы* не будут восприняты *вводом. Входной порт* может содержать любое число *сигналов* [2.4.4].

экземпляр

англ. : instance
исп. : instancia
фр. : instance

Экземпляром типа является объект, обладающий свойствами *типа* (данными в определении) [1.3.1].

конкретизация

англ. : instantiation
исп. : instanciación
фр. : instantiation

Конкретизацией является создание *экземпляра типа* [1.3.1].

целый

англ. : integer
исп. : entero; integer
фр. : entier (integer)

Целый является *сортом*, определенным в предопределенном *частичном определении типа, значениями* которого являются математические целые числа (...—2, —1,0, +1, +2, ...). Для *сорта целый* предопределенными являются операции: +, —, *, / и *операции упорядочения* [5.6.5].

диаграмма взаимодействия

англ. : interaction diagram
исп. : diagrama de interacción
фр. : diagramme d'interaction

Диаграммой взаимодействия является *диаграмма блока, диаграмма системы, диаграмма подструктуры канала* или *диаграмма подструктуры блока.*

ключевое слово

англ. : keyword
исп. : palabra clave
фр. : mot clé

Ключевое слово является резервированной *лексической единицей конкретного текстового синтаксиса* [2.2.1].

метка

англ. : label
исп. : etiqueta
фр. : étiquette

Метка является *именем*, за которым следует двоеточие; она используется в *конкретном текстовом синтаксисе* в целях связывания [2.6.6].

уровень

англ. : level
исп. : nivel
фр. : niveau

Термин *уровень* является синонимом термина *уровень абстракции.*

уровень абстракции

англ. : level of abstraction
исп. : nivel de abstracción
фр. : niveal d'abstraction

Уровень абстракции является одним из уровней *диаграммы дерева блоков.* Описанием *системы* является один *блок* на верхнем *уровне абстракции* и изображается в качестве единственного *блока* в верхней точке *диаграммы дерева блоков* [3.2.1].

лексические правила

англ. : lexical rules
исп. : reglas léxicas
фр. : règles lexicales

Лексические правила являются правилами, описывающими методы построения *лексических единиц* из знаков [2.2.1, 4.2.1].

лексема

англ. : lexical unit
исп. : unidad léxica
фр. : unités lexicales

Лексемы являются терминальными *символами конкретного текстового синтаксиса* [2.2.1].

литерал

англ. : literal
исп. : literal
фр. : littéral

Литерал обозначает *значение* [2.3.3, 5.1.2, 5.4.1.14].

макрос

англ. : macro
исп. : macro
фр. : macro

Макрос является поименованной совокупностью синтаксических или текстовых элементов, заменяющей *вызов макроса,* прежде чем рассматривается смысл некоторого *SDL*-изображения (иными словами, *макрос* имеет смысл только будучи помещенным в конкретный контекст) [4.2].

вызов макроса

англ. : macro call
исп. : llamada a (de) macro
фр. : appel de macro

Вызов макроса указывает то место, где должна быть выполнена подстановка *макроопределения* с тем же именем [4.2.3].

макроопределение

англ. : macro definition
исп. : definición de macro
фр. : definition de macro

Макроопределение является определением *макроса* в *SDL/PR* [4.2.2].

диаграмма макроса

англ. : macro diagram
исп. : diagrama de macro
фр. : diagramme de macro

Диаграмма макроса является макроопределением в *SDL/GR* [4.2.2].

сделать!

англ. : make!
исп. : hacer!; make!
фр. : make!

Сделать! является операцией, используемой только в определениях *типа данных* для образования *значения* составного типа (например, *структурного сорта*) [5.4.1.10, 5.6.8].

область слияния

англ. : merge area
исп. : área de fusión
фр. : zone de fusion

Область слияния является тем местом, где одна *линия потока* соединяется с другой [2.6.7.2.2].

Мета IV

англ. : Meta IV
исп. : Meta IV
фр. : Meta IV

Мета IV является формальной нотацией, выражающей *абстрактный синтаксис* языка [1.5.1].

модель

англ. : model
исп. : modelo
фр. : modèle

Модель обеспечивает отображение *сокращенных обозначений,* выраженное в терминах ранее определенного *конкретного синтаксиса* [1.4.1, 1.4.2].

модифицировать!

англ. : modify!
исп. : modificar!; modify!
фр. : modify!

Модифицировать! является *операцией,* которая подразумевается в *выражениях* в тех случаях, когда непосредственно за *переменной* следует *выражение,* взятое в скобки, а за ним — знак :=. В аксиомах *операция модифицировать!* используется явно (см. *извлечь!*) [5.4.1.10, 5.6.8].

имя

англ. : name
исп. : nombre
фр. : nom

Имя является *лексической единицей,* используемой в *SDL* для именования объектов [2.2.1, 2.2.2].

натуральный

англ. : natural
исп. : natural
фр. : naturel

Натуральный является *подтипом,* определенным в предопределенном *частичном определении типа; значениями* этого *типа* являются неотрицательные целые числа (то есть 0, 1, 2, ...). *Операциями* являются *операции целого сорта* [5.6.6].

новый тип

англ. : newtype
исп. : nuevo tipo
фр. : nouveau type (newtype)

Новый тип вводит *сорт,* совокупность *операций* и совокупность *равенств.* Следует отметить, что термин *новый тип* может вводить в заблуждение, так как фактически вводится *новый сорт,* однако термин *новый тип* сохранен по историческим соображениям [5.2.1].

вершина

англ. : node
исп. : nodo
фр. : noeud

В *абстрактном синтаксисе вершиной* является обозначение любого базисного понятия *SDL.*

замечание

англ. : note
исп. : nota
фр. : note

Замечанием является текст, заключенный между знаком /* и знаком */; этот текст не имеет в *SDL* определенной семантики. См. *комментарий* [2.2.1].

null

англ. : null
исп. : null; nulo
фр. : null

Null является *литералом сорта PId* [5.6.10].

OFFSPRING (потомок)

англ. : OFFSPRING
исп. : OFFSPRING; VASTAGO
фр. : DESCENDANT (OFFSPRING)

OFFSPRING является *выражением сорта PId*. Когда в процессе вычисляется выражение *OFFSPRING*, оно дает *значение PId* того *процесса*, который был последним *создан* данным *процессом*. Если *процесс* не *создавал* никаких других *процессов*, то результатом вычисления *выражения OFFSPRING* является *null* [2.4.4, 5.5.4.3].

операция

англ. : operator
исп. : operador
фр. : opérateur

Операция является обозначением операции. *Операции* определяются в *частичном определении типа*. Например, +, —, *, / являются *именами операций* для *целого сорта* [5.1.2, 5.1.3].

сигнатура операции

англ. : operator signature
исп. : signatura de operador
фр. : signature d'opérateur

Сигнатура операции определяет *сорт (сорта) значений*, к которым могут быть применены *операция* и *сорт* результирующего *значения* [5.2.2].

выбираемое определение

англ. : option
исп. : opción
фр. : option

Выбираемое определение является конструкцией *конкретного синтаксиса* в порождающей *спецификации SDL-системы;* эта конструкция позволяет выбирать *системы* различной структуры, прежде чем начинать интерпретацию *системы* [4.3.3, 4.3.4].

операции упорядочения

англ. : ordering operators
исп. : operadores de ordenación
фр. : opérateurs de relation d'ordre

Операциями упорядочения являются <, <=, > или >= [5.4.1.8].

выходной коннектор

англ. : out connector
исп. : conector de salida
фр. : connecteur de sortie

Выходной коннектор является *коннектором.*

выходная точка

англ. : outlet
исп. : acceso de salida
фр. : accès sortant

Выходная точка представляет линию, такую как *канал* или *линия потока,* выходящую из *диаграммы макроса* [4.2.2].

вывод

англ. : output
исп. : salida
фр. : sortie

Вывод является *действием,* содержащимся в *переходе;* он порождает *экземпляр сигнала.*

область вывода

англ. : output area
исп. : área de salida
фр. : zone de sortie

В *диаграмме потока управления область вывода* является *SDL/GR*-изображением концепции *вывода* [2.7.4].

страница

англ. : page
исп. : página
фр. : page

Страница является одной из компонент физического разбиения *диаграммы* [2.2.5].

PARENT (родитель)

англ. : PARENT
исп. : PARENT; PROGENITOR
фр. : PARENT

PARENT является *PId-выражением.* Когда в процессе вычисляется это *выражение,* результатом является *значение PId* родительского *процесса.* Если *процесс* был *создан* на этапе инициализации *системы,* результатом является *null* [2.4.4, 5.5.4.3].

частичное определение типа

англ. : partial type definition
исп. : definición parcial de tipo
фр. : défintion partielle de type

По отношению к *сорту частичное определение типа* определяет некоторые из свойств *сорта. Частичное определение типа* является частью определения *типа данных* [5.2.1].

разбиение

англ. : partitioning
исп. : partición
фр. : subdivision

Разбиение является подразделением некоторого элемента на более мелкие компоненты; взятые в своей совокупности эти компоненты ведут себя так же, как и исходный элемент. *Разбиение* не влияет на статический интерфейс элемента [3.1, 3.2].

PId

англ. : PId
исп. : PId
фр. : PId

PId является *сортом,* определенным в предопределенном *частичном определении типа. Null* является единственным *литералом* этого *сорта. PId* является аббревиатурой process instance identifier (идентификатор экземпляра процесса); *значения* этого *сорта* используются для идентификации *экземпляров процессов* [5.5.4.3, 5.6.10],

множественный

англ. : powerset
исп. : conjunista
фр. : mode ensembliste

Множественный является предопределенным *генератором,* используемым для введения математических множеств. *Операциями* для него являются IN, Incl, Del, union (объединение), intersection (пересечение) и *операции упорядочения* [5.6.9].

предопределенные данные

англ. : predefined data
исп. : datos predefinidos
фр. : données prédéfinies

Для простоты описания термин *предопределенные данные* применяется как к предопределенным *именам сортов*, вводимых *частичным определением типа*, так и предопределенным *именам генераторов типов данных. Boolean (Булевский), character (знаковый), charstring (знакостроковый), duration (длительность), integer (целый), natural (натуральный), PId* (идентификатор экземпляра процесса), *real (вещественный), и time (временно́й)* являются предопределенными *именами сортов. Array (массив), powerset (множественный)* и *string (строка)* являются *именами* предопределенных *генераторов типов данных. Предопределенные данные* считаются неявно определенными на *уровне системы* для всех *SDL-систем* [5.6].

процедура

англ. : procedure
исп. : procedimiento
фр. : procédure

Процедура является инкапсуляцией некоторой части *поведения процесса. Процедура* определяется в одном каком-нибудь месте, но допускается многократное обращение к ней из одного и того же *процесса.* См. *формальные параметры* и *фактические параметры* [2.4.5].

вызов процедуры

англ. : procedure call
исп. : llamada a (de) procedimiento
фр. : appel de procédure

Вызов процедуры является возбуждением интерпретации поименованной *процедуры* и передачей ей *фактических параметров* [2.7.3].

область вызова процедуры

англ. : procedure call area
исп. : área de llamada a (de) procedimiento
фр. : zone d'appel de procédure

Область вызова процедуры является *SDL/GR-изображением вызова процедуры* [2.7.3].

определение процедуры

англ. : procedure definition
исп. : definición de procedimiento
фр. : définition de procédure

Определение процедуры является определением *процедуры в SDL/PR* [2.4.5].

диаграмма процедуры

англ. : procedure diagram
исп. : diagrama de procedimiento
фр. : diagramme de procédure

Диаграмма процедуры является изображением *процедуры в SDL/GR* [2.4.5].

граф процедуры

англ. : procedure graph
исп. : gráfico de procedimiento
фр. : graphe de procédure

Граф процедуры является нетерминальным символом *абстрактного синтаксиса,* изображающим *процедуру* [2.4.5].

возврат из процедуры

англ. : procedure return
исп. : retorno de procedimiento
фр. : retour de procédure

Термин *возврат из процедуры* является синонимом термина *возврат.*

процесс

англ. : process
исп. : proceso
фр. : processus

Процесс является взаимодействующим обобщенным конечным автоматом. Взаимодействие осуществляется либо с помощью *сигналов,* либо за счет совместно используемых *переменных. Поведение процесса* зависит от порядка, в котором *сигналы* прибывают в его *входной порт* [2.4.4].

область процесса

англ. : process area
исп. : área de proceso
фр. : zone de processus

В *SDL/GR область процесса* является изображением в *диаграмме взаимодействия* либо *процесса,* либо ссылки на *процесс* [2.4.3].

определение процесса

англ. : process definition
исп. : definición de proceso
фр. : définition de processus

Определение процесса является *SDL/PR*-изображением *процесса* [2.4.4].

диаграмма процесса

англ. : process diagram
исп. : diagrama de proceso
фр. : diagramme de processus

Диаграмма процесса является *SDL/GR*-изображением определения *процесса* [2.4.4].

граф процесса

англ. : process graph
исп. : gráfico de proceso
фр. : graphe de processus

Граф процесса является нетерминальным символом, изображающим *процесс* в *абстрактном синтаксисе* [2.4.4].

экземпляр процесса

англ. : process instance
исп. : instancia de proceso
фр. : instance de processus

Экземпляр процесса является динамически созданным *экземпляром* некоторого *процесса.* См. *SELF, SENDER, PARENT* и *OFFSPRING* [2.4.4].

квалификатор

англ. : qualifier
исп. : calificador
фр. : partie qualificative (qualificatif)

Квалификатор является частью *идентификатора,* сообщающей дополнительно информацию *именной* части *идентификатора;* делается это с целью обеспечения уникальности. *Квалификаторы* всегда присутствуют в *абстрактном синтаксисе,* но в *конкретном синтаксисе квалификаторы* могут использоваться только в меру необходимости для обеспечения уникальности, если *квалификатор идентификатора* не может быть выявлен из контекста использования *именной* части [2.2.2].

вещественный

англ. : real
исп. : real
фр. : réel

Вещественный является предопределенным *сортом,* определенным в *частичном определении типа, значениями* которого являются числа, допускающие представленные в виде одного *целого,* поделенного на другое. Предопределенные *операции* этого *зорта* имеют те же *имена,* что и *операции сорта целый* [5.6.7].

уточнение

англ. : refinement
исп. : refinamiento
фр. : reaffinement

Уточнение является добавлением новых деталей функционирования на некотором *уровне абстракции. Уточнение системы* обогащает ее *поведение* или ее возможности для обработки дополнительных типов *сигналов* и информации, включая *сигналы,* поступающие из *окружающей среды* или передаваемые ей. Сравните с *разбиением* [3.3].

удаленное определение

англ. : remote definition
исп. : definición remota
фр. : définition distante

Удаленное определение является синтаксическим средством рассредоточения *определения системы* по нескольким частям и связывания этих частей друг с другом [2.4.1].

сброс

англ. : reset
исп. : reincializar; reponer
фр. : reset (réinitialisation)

Сброс является операцией, определенной на *таймерах;* эта операция позволяет приводить *таймер* в неактивные состояния. См. *активный таймер* [2.8].

задержанный сигнал

англ. : retained signal
исп. : señal retenida
фр. : signal retenu

Задержанный сигнал является *сигналом,* находящимся во *входном порту процесса,* то есть это *сигнал,* который был получен, но пока не воспринят *процессом* [2.4.4].

возврат

англ. : return
исп. : retorno
фр. : retour

Возврат из *процедуры* является передачей управления вызвавшей *процедуре* или *процессу* [2.6.7.2.4].

атрибут раскрытия

англ. : reveal attribute
исп. : atributo revelado
фр. : attribut d'exposition

Переменная, которой владеет некоторый *процесс,* может иметь *атрибут раскрытия;* в этом случае другой *процесс* в том же *блоке* получает разрешение обозревать *значение,* ассоциируемое с *переменной.* См. *определение обозревания* [2.6.1.1].

сохранение

англ. : save
исп. : conservación
фр. : mise en réserve

Сохранение является объявлением тех *сигналов,* которые не должны быть восприняты в данном *состоянии* [2.6.5].

область сохранения

англ. : save area
исп. : área de conservación
фр. : zone de mise en réserve

Область сохранения является *SDL/GR*-изображением *сохранения* [2.6.5].

совокупность сохраняемых сигналов

англ. : save signal set
исп. : conjunto de señales de conservación
фр. : ensemble de signaux de mise en réserve

Совокупность сохраняемых сигналов некоторого *состояния* является совокупностью сохраняемых в этом *состоянии сигналов* [2.6.5].

SDL (язык спецификации и описания МККТТ)

англ. : SDL (CCITT Specification and Description Language)
исп. : LED (lenguaje de especificación y descripción del CCITT)
фр. : LDS (langage de description et de spécification du CCITT)

SDL (язык спецификации и описания) МККТТ является формальным языком, обеспечивающим совокупность конструкций для *спецификации* функционирования системы.

SDL/GR

англ. : SDL/GR
исп. : LED/GR
фр. : LDS/GR

SDL/GR является графическим изображением *SDL*. *Грамматика SDL/GR* определяется *конкретной графической грамматикой* и *общей текстовой грамматикой* [1.2].

SDL/PE

англ. : SDL/PE
исп. : LED/EP
фр. : LDS/PE

SDL/PE является совокупностью картинок, которые могут использоваться в сочетании с *символом состояния*, входящим в *SDL/GR* [Приложение E].

SDL/PR

англ. : SDL/PR
исп. : LED/PR
фр. : LDS/PR

SDL/PR является текстовым изображением *SDL*, состоящим из отдельных фраз. *Грамматика SDL/PR* определяется *конкретной текстовой грамматикой* [1.2].

единица подразделения

англ. : scope unit
исп. : unidad de àmbito
фр. : unité de portée

В *конкретной грамматике единица подразделения* определяет область *видимости идентификаторов*. Примерами *единиц подразделения* являются: *система, блок, процесс, процедура, частичное определение типа* и *определение сервиса* [2.2.2].

отбор

англ. : selection
исп. : selección
фр. : sélection

Отбором является обеспечение *внешними синонимами*, необходимыми для получения конкретной *спецификации системы* из родовой *спецификации системы* [4.3.3].

SELF (сам)

англ. : SELF
исп. : SELF; MISMO
фр. : SELF

SELF является *PId-выражением*. Когда *процесс* вычисляет это *выражение*, результатом является *значение PId* данного *процесса*. *SELF* никогда не дает в качестве результата *значение Null*. См. также *PARENT, OFFSPRING, PId* [2.4.4, 5.5.4.3].

семантика

англ. : semantics
исп. : semántica
фр. : semantique

Семантика придает значение некоторому объекту, а именно: свойства, которыми он обладает, метод интерпретации его *поведения* и любые динамические условия, которые должны быть выполнены, для того чтобы *поведение* объекта удовлетворяло требованиям *SDL* [1.4.1, 1.4.2].

SENDER (отправитель)

англ. : SENDER
исп. : SENDER; EMISOR
фр. : SENDER (émetteur)

SENDER является *PId-выражением.* При вычислении это *выражение* выдает *значение PId процесса*-отправителя того *сигнала,* который вызвал текущий *переход* [2.4.4, 2.6.4, 5.5.4.3].

сервис

англ. : service
исп. : servicio
фр. : service

Сервис является альтернативным средством спецификации *процесса.* Каждый *сервис* может определять часть *поведения* некоторого *процесса* [4.10].

область сервиса

англ. : service area
исп. : área de servicio
фр. : zone de service

Область сервиса является либо *диаграммой сервиса,* либо ссылкой на *сервис* [4.10.1].

определение сервиса

англ. : service definition
исп. : definición de servicio
фр. : définition de service

Определение сервиса является определением *сервиса* в *SDL/PR* [4.10.1].

диаграмма сервиса

англ. : service diagram
исп. : diagrama de servicio
фр. : diagramme de service

Диаграмма сервиса является определением *сервиса* в *SDL/GR* [4.10].

установка

англ. : set
исп. : inicializar; poner
фр. : set (initialisation)

Установка является операцией, определенной на *таймерах,* которая позволяет привести *таймеры* в *активное состояние* [2.8].

сокращенное обозначение

англ. : shorthand notation
исп. : notación taquigrafica (o abreviada)
фр. : notation abrégée

Сокращенное обозначение является обозначением, используемым в *конкретном синтаксисе* для более компактного представления; такие обозначения неявно ссылаются на понятия *Базисного SDL* [1.4.2].

сигнал

англ. : signal
исп. : señal
фр. : signal

Сигнал является экземпляром некоторого *типа* сигнала; *сигнал* передает информацию некоторому *экземпляру процесса* [2.5.4].

определение сигнала

англ. : signal definition
исп. : definición de señal
фр. : définition de signal

Определение сигнала определяет *именованный тип сигнала* и ассоциирует *с именем сигнала* ноль или большее число *идентификаторов сорта.* Это позволяет *сигналам* переносить *значения* [2.5.4].

список сигналов

англ. : signal list
исп. : lista de señales
фр. : liste de signaux

Список сигналов является списком *идентификаторов сигналов,* используемым в *определениях каналов* и *маршрутов сигналов;* с помощью списка описываются все *сигналы,* которые могут быть переданы данным *каналом* или *маршрутом сигнала* в одном направлении [2.5.5].

область списка сигналов

англ. : signal list area
исп. : área de lista de señales
фр. : zone de liste de signaux

Область списка сигналов изображает в *диаграмме взаимодействия список сигналов,* ассоциируемых с *каналом* или *маршрутом сигнала* [2.5.5].

маршрут сигнала

англ. : signal route
исп. : ruta de señales
фр. : acheminement de signaux

Маршрут сигнала описывает направление движения *сигнала* между *типом процесса* и либо другим *типом процесса* в том же *блоке,* либо *каналом,* присоединенным к *блоку* [2.5.2].

простое выражение

англ. : simple expression
исп. : expresión simple
фр. : expression simple

Простое выражение является *выражением,* содержащим только *операции, синонимы* и *литералы* предопределенных *сортов* [4.3.2].

сорт

англ. : sort
исп. : género
фр. : sorte

Сорт является совокупностью *значений* с общими характеристиками. *Сорта* всегда не пусты и не имеют общих элементов [2.3.3, 5.1.3].

спецификация

англ. : specification
исп. : especificación
фр. : spécification

Спецификацией является определение требований, предъявляемых к *системе. Спецификация* состоит из *общих параметров*, требуемых от *системы* и *функциональной спецификации*, требуемого от нее *поведения*. Термин *спецификация* может, кроме того, использоваться в качестве сокращения термина "*спецификация* и/или *описание*", как, например, *спецификация SDL* или *спецификация системы* [1.1].

старт

англ. : start
исп. : arranque
фр. : départ

Старт процесса интерпретируется до любого *состояния* или *действия. Старт* инициализирует *процесс*, заменяя в нем *формальные параметры* на *фактические параметры* в соответствии со спецификацией в *запросе на создание* [2.6.2].

состояние

англ. : state
исп. : estado
фр. : état

Состояние является условием, при котором *экземпляр процесса* может воспринять *сигнал* [2.6.3].

область состояния

англ. : state area
исп. : área de estado
фр. : zone d'état

Область состояния является *SDL/GR*-изображением одного или более *состояний* [2.6.3].

картина состояния

англ. : state picture
исп. : pictograma de estado
фр. : représentation graphique d'état

Картина состояния является *символом состояния*, содержащим картинные моменты, используемые для расширения *SDL/GR* к *SDL/PE* [Приложение E].

стоп

англ. : stop
исп. : parada
фр. : arrêt

Стоп является действием, прекращающим *экземпляр процесса.* При интерпретации *стопа* все переменные, принадлежащие *экземпляру процесса*, разрушаются, а все *задержанные* во *входном порту сигналы* перестают быть доступными [2.6.7.2.3].

строка

англ. : string
исп. : cadena; string
фр. : chaîne (string)

Строка является предопределенным *генератором*, используемым для введения списков. Предопределенными *операциями* являются Length (длина), First (первый), Last (последний), Substring (подстрока) и concatenation (конкатенация) [5.6.3].

структурный сорт

англ. : structured sort
исп. : género estructurado
фр. : sorte structurée

Структурный сорт является *сортом* с неявными *операциями* и *равенствами* и специальным *конкретным синтаксисом* для этих неявных *операций. Структурный сорт* используется для образования *значений* сопоставляемых так называемых полей. Обеспечиваются независимый *доступ* к значениям полей и их *модификация* [5.4.1.10].

подблок

англ. : subblock
исп. : subbloque
фр. : sous-bloc

Подблок является *блоком*, содержащимся в другом *блоке*. *Подблоки* образуются при *разбиении блока* [3.2.1, 3.2.2].

подканал

англ. : subchannel
исп. : subcanal
фр. : sous-canal

Подканал является *каналом*, образующимся при *разбиении блока*. *Подканал* соединяет *подблок* с границей *блока*, подвергшегося *разбиению*, или *блок* с границей *канала*, подвергшегося *разбиению* [3.2.2, 3.2.3].

подсигнал

англ. : subsignal
исп. : subseñal
фр. : sous-signal

Подсигнал является *уточнением сигнала;* он может быть в свою очередь подвергнут *уточнению* [3.3].

символ

англ. : symbol
исп. : simbolo
фр. : symbole

Символ является оконечным объектом *конкретного графического синтаксиса*. *Символ* может быть одним из совокупности контуров *конкретного графического синтаксиса*.

синоним

англ. : synonym
исп. : sinónimo
фр. : synonyme

Синоним является *именем*, представляющим *значение* [5.4.1.13].

синтаксическая диаграмма

англ. : syntax diagram
исп. : diagrama de sintaxis
фр. : diagramme de syntaxe

Синтаксические диаграммы являются иллюстрациями определений *конкретного текстового синтаксиса* [Приложение С2].

подтип

англ. : syntype
исп. : sintipo
фр. : syntype

Подтип определяет множество *значений*, соответствующее подмножеству *значений* родительского *типа*. *Операции подтипа* совпадают с таковыми родительского *типа* [5.4.1.9].

система

англ. : system
исп. : sistema
фр. : systeme

Системой является совокупность *блоков*, соединенных между собой и с *окружающей средой* с помощью *каналов*.

определение системы

англ. : system definition
исп. : definición de sistema
фр. : définition de système

Определение системы является *SDL/PR*-изображением *системы* [2.4.2].

диаграмма системы

англ. : system diagram
исп. : diagrama de sistema
фр. : diagramme de système

Диаграмма системы является *SDL/GR*-изображением *системы* [2.4.2].

работа

англ. : task
исп. : tarea
фр. : tâche

Работа является действием, выполняемым в *переходе;* она содержит либо последовательность *предложений присваивания*, либо *неформальный текст.* Интерпретация *работы* зависит от информации, имеющейся в *системе*, и может воздействовать на эту информацию [2.7.1].

область работы

англ. : task area
исп. : área de tarea
фр. : zone de tâche

Область работы является *SDL/GR*-изображением *работы* [2.7.1].

терм

англ. : term
исп. : término
фр. : terme

Синтаксически *терм* эквивалентен *выражению. Термы* используются только в *аксиомах* и выделены в отдельное понятие в интересах большей ясности [5.2.3, 5.3.3].

символ расширения текста

англ. : text extension symbol
исп. : sibolo de ampliación de texto
фр. : symbole d'extension de texte

Символ расширения текста является контейнером текста, принадлежащего тому *графическому символу*, к которому присоединен *символ расширения текста.* Текст в *символе расширения текста* следует за текстом, находящимся в том *символе*, к которому данный текст присоединен [2.2.7].

временно́й

англ. : time
исп. : tiempo; time
фр. : temps (time)

Временно́й является *сортом*, определенным в предопределенном *частичном определении типа; значения* этого *сорта* обозначаются так же, как значения *сорта "вещественный".* Предопределенными *операциями*, использующими *временно́й* и *длительность*, являются + и — [5.5.4.1, 5.6.12].

таймер

англ. : timer
исп. : temporizador
фр. : temporisateur

Таймер является объектом, которым владеет *экземпляр процесса.* Таймер может быть либо *активным*, либо *неактивным. Активный таймер* возвращает в специфицированный момент времени *сигнал от таймера* тому *экземпляру процесса*, который владеет данным *таймером.* См. также *установка* и *сброс* [2.8, 5.5.4.5].

переход

англ. : transition
исп. : transición
фр. : transition

Переход является последовательностью *действий*, которая возникает, когда *экземпляр процесса* меняет одно *состояние* на другое [2.6.7.1].

область перехода

англ. : transition area
исп. : área de transición
фр. : zone de transition

Область перехода является *SDL/GR*-изображением *перехода* [2.6.7.1].

цепочка перехода

англ. : transition string
исп. : cadena de transicion
фр. : chaîne de transition

Цепочка перехода состоит из нуля или большего числа *действий* [2.6.7.1].

область цепочки перехода

англ. : transition string area
исп. : área de cadena de transición
фр. : zone de chaîne de transition

Область цепочки перехода является *SDL/GR*-изображением *цепочки перехода* [2.6.7.1].

тип

англ. : type
исп. : tipo
фр. : type

Тип является совокупностью свойств элементов. Примерами классов *типов* в *SDL* служат *блоки, каналы, маршруты сигналов, сигналы* и *системы* [1.3.1].

определение типа

англ. : type definition
исп. : definición de tipo
фр. : définition de type

Определение типа определяет свойства некоторого *типа* [1.3.1].

неопределенный

англ. : undefined
исп. : indefinido
фр. : indéfini (undefined)

Неопределенный является "специальным" *значением* каждого *сорта*, указывающим что *переменной* этого *сорта* пока еще не было *присвоено* нормальное *значение*. См. *доступ* [5.5.2.2].

совокупность допустимых входных сигналов

англ. : valid input signal set
исп. : conjunto de señales de entrada válidas
фр. : ensemble de signaux d'entrée valides

Совокупностью допустимых входных сигналов некоторого *процесса* является список всех внешних *сигналов*, обрабатываемых хоть одним *вводом* в *процессе*. Он состоит из *сигналов* тех *маршрутов сигналов*, которые ведут к данному *процессу*. Сравните с *полной совокупностью допустимых входных сигналов* [2.4.4, 2.5.2].

допустимая спецификация

англ. : valid specification
исп. : especificación válida
фр. : spécification valide

Допустимой спецификацией является *спецификация*, которая следует *конкретному синтаксису* и статическим *правилам корректности* [1.3.3].

значение

англ. : value
исп. : valor
фр. : valeur

Значение сорта является одним из значений, которые ассоциируются с *переменной* этого *сорта* и которые могут быть использованы *операцией*, требующей *значение* этого *сорта*. *Значение* является результатом интерпретации *выражения* [2.3.3, 5.1.3].

переменная

англ. : variable
исп. : variable
фр. : variable

Переменная является объектом, которым владеет *экземпляр процесса* или *экземпляр процедуры;* с *переменной* может быть связано некоторое *значение* с помощью предложения присваивания. Когда к *переменной* осуществляется *доступ*, она выдает последнее присвоенное ей *значение* [2.3.2].

определение переменной

англ. : variable definition
исп. : definición de variable
фр. : définition de variable

Определение переменной свидетельствует о том, что *имена* перечисленных *переменных* будут доступны в *процессе*, *процедуре* или *сервисе*, содержащем определение [2.6.1.1].

определение обозревания

англ. : view definition
исп. : definición de visión
фр. : définition de visibilité

Определение обозревания определяет *идентификатор переменной* в другом *процессе*, в котором эта *переменная* имеет атрибут раскрытия. Это обеспечивает доступ к значению этой переменной обозревающему процессу [2.6.1.2].

обозревающее выражение

англ. : view expression
исп. : expresión de visión
фр. : expression de vue

Обозревающее выражение используется в *выражении* для получения текущего *значения обозреваемой переменной* [5.5.4.4].

область видимости

англ. : visibility
исп. : visibilidad
фр. : visibilité

Областью видимости идентификатора являются те *единицы подразделения*, в которых он может быть использован. Никакие два определения, принадлежащие одному и тому же *классу объектов* и находящиеся в одной и той же *единице подразделения*, не могут иметь одно и то же *имя* [2.2.2].

правила корректности

англ. : well-formedness rules
исп. : reglas de formación correcta
фр. : règles de bonne formation

Правила корректности являются ограничениями в *конкретном синтаксисе*, определяющими статические условия, не выраженные явно синтаксическими правилами [1.4.1, 1.4.2].

ПРИЛОЖЕНИЕ B
(к Рекомендации Z.100)

Сводное описание абстрактного синтаксиса

Идентификатор :: *Квалификатор Имя*

Квалификатор = *Квалификационный-шаг* +

Квалификационный-шаг =
Квалификатор-системы |
Квалификатор-блока |
Квалификатор-подструктуры-блока |
Квалификатор-сигнала |
Квалификатор-процесса |
Квалификатор-процедуры |
Квалификатор-сорта

Квалификатор-системы :: *Имя-системы*

Квалификатор-блока :: *Имя-блока*

Квалификатор-подструктуры-блока :: *Имя-подструктуры-блока*

Квалификатор-процесса :: *Имя-процесса*

Квалификатор-процедуры :: *Имя-процедуры*

Квалификатор-сигнала :: *Имя-сигнала*

Квалификатор-сорта :: *Имя-сорта*

Имя :: *Элементарный знак*

Неформальный текст :: ...

Определение-системы ::
Имя-системы
Определение-блока-set
Определение-канала-set
Определение-сигнала-set
Определение-типа-данных
Определение-подтипа-set

Имя-системы = *Имя*

Определение-блока ::
Имя-блока
Определение-процесса-set
Определение-сигнала-set
Соединение-канала-с-маршрутом-set
Определение-маршрута-сигнала-set
Определение-типа-данных
Определение-подтипа-set
[*Определение-подструктуры-блока*]

Имя-блока = *Имя*

Определение-процесса ::
Имя-процесса
Число-экземпляров
*Формальные-параметры-процесса**
Определение-процедуры-set
Определение-сигнала-set
Определение-типа-данных
Определение-подтипа-set
Определение-переменной-set
Определение-обозревания-set
Определение-таймера-set
Граф-процесса

Число-экземпляров :: *Intg Intg*

Имя-процесса = *Имя*

Граф-процесса ::
Стартовая-вершина-процесса
Вершина-состояния-set

Формальные-параметры-процесса ::
Имя-переменной
Идентификатор-ссылки-на-сорт

Определение-процедуры ::
Имя-процедуры
*Формальные-параметры-процедуры**
Определение-процедуры-set
Определение-типа-данных
Определение-подтипа-set
Определение-переменной-set
Граф-процедуры

Имя-процедуры = *Имя*

Формальные-параметры-процедуры =
Входной-параметр |
Входной-выходной-параметр

Входной-параметр ::
Имя-переменной
Идентификатор-ссылки-на-сорт

Входной-выходной-параметр ::
Имя-переменной
Идентификатор-ссылки-на-сорт

Граф-процедуры ::
Стартовая-вершина-процедуры
Вершина-состояния-set

Стартовая-вершина-процедуры :: *Переход*

Определение-канала ::
Имя-канала
Путь-канала
[*Путь-канала*]

Путь-канала :: *Исходный-блок*

Приемный-блок
*Идентификатор-сигнала-***set**

Исходный-блок = *Идентификатор-блока* |
ENVIRONMENT

Приемный-блок = *Идентификатор-блока* |
ENVIRONMENT

Идентификатор-блока = *Идентификатор*

Идентификатор сигнала = *Идентификатор*

Имя-канала = *Имя*

Определение-маршрута-сигнала :: *Имя-маршрута-сигнала*
Путь-маршрута-сигнала
[*Путь-маршрута-сигнала*]

Путь-маршрута-сигнала :: *Исходный-процесс*
Приемный-процесс
*Идентификатор-сигнала-***set**

Исходный-процесс = *Идентификатор-процесса* |
ENVIRONMENT

Приемный-процесс = *Идентификатор-процесса* |
ENVIRONMENT

Имя-маршрута-сигнала = *Имя*

Соединение-канала-с-маршрутом :: *Идентификатор-канала*
*Идентификатор-маршрута-сигнала-***set**

Идентификатор-маршрута-сигнала = *Идентификатор*

Определение-сигнала :: *Имя-сигнала*
*Идентификатор-ссылки-на-сорт******
[*Уточнение-сигнала*]

Имя-сигнала = *Имя*

Определение-переменной :: *Имя-переменной*
Идентификатор-ссылки-на-сорт
[**REVEALED**]

Имя-переменной = *Имя*

Определение-обозревания :: *Идентификатор-переменной*
Идентификатор-ссылки-на-сорт

Стартовая-вершина-процесса :: *Переход*

Вершина-состояния :: *Имя-состояния*
Совокупность-сохраняемых-сигналов
Входная-вершина-**set**

Имя-состояния = *Имя*

Входная-вершина :: *Идентификатор-сигнала*
[*Идентификатор-переменной*]*
Переход

Идентификатор-переменной = *Идентификатор*

Совокупность-сохраняемых-сигналов :: *Идентификатор-сигнала*-**set**

Переход :: *Вершина-графа**
(Терминатор | Вершина-принятия-решения)

Вершина-графа :: *Вершина-работы* |
Выходная-вершина |
Вершина-запроса-на-создание |
Вершина-вызова |
Вершина-установки |
Вершина-сброса

Терминатор :: *Вершина-следующего-состояния* |
Стоп-вершина |
Вершина-возврата

Вершина-следующего-состояния :: *Имя-состояния*

Вершина-возврата :: *( )*

Стоп-вершина :: *( )*

Вершина-работы :: *Предложение-присваивания* |
Неформальный-текст

Вершина-запроса-на-создание :: *Идентификатор-процесса*
[*Выражение*]*

Идентификатор-процесса = *Идентификатор*

Вершина-вызова :: *Идентификатор-процедуры*
[*Выражение*]*

Идентификатор-процедуры = *Идентификатор*

Вершина-принятия-решения :: *Вопрос-принятия-решения*
Ответ-принятия-решения-**set**
[*Иначе-ответ*]

Вопрос-принятия-решения = *Выражение* |

Неформальный текст

Ответ-принятия-решения :: *(Условие-на-диапазон |*
Неформальный-текст) Переход

Иначе-ответ :: *Переход*

Выходная-вершина :: *Идентификатор-сигнала*
*[Выражение]**
*[Приемник-сигнала]**
Направить-через

Приемник-сигнала = *Выражение*

Направить-через = *Идентификатор-маршрута-сигнала-***set**

Определение-таймера :: *Имя-таймера Идентификатор-ссылки-на-сорт**

Имя-таймера = *Имя*

Вершина-установки :: *Выражение-время*
Идентификатор-таймера
*Выражение**

Вершина-сброса :: *Идентификатор-таймера*
*Выражение**

Идентификатор-таймера = *Идентификатор*

Выражение-время = *Выражение*

Определение-подструктуры-блока :: *Имя-подструктуры-блока*
*Определение-подблока-***set**
*Соединение-канала-***set**
*Определение-канала-***set**
*Определение-сигнала-***set**
Определение-типа-данных
*Определение-подтипа-***set**

Имя-подструктуры-блока = *Имя*

Определение-подблока = *Определение-блока*

Соединение-канала :: *Идентификатор-канала*
*Идентификатор-подканала-***set**

Идентификатор-подканала = *Идентификатор-канала*

Идентификатор-канала = *Идентификатор*

Уточнение-сигнала :: *Определение-подсигнала-***set**

Определение-подсигнала :: **[REVERSE]** *Определение-сигнала*

Определение-типа-данных ::
Имя-типа
Объединение-типов
Сорта
*Сигнатура-***set**
Равенства

Объединение-типов = *Идентификатор-типа-***set**

Идентификатор-типа = *Идентификатор*

Сорта = *Имя-сорта-***set**

Имя-типа = *Имя*

Имя-сорта = *Имя*

Равенства = *Равенство-***set**

Сигнатура =
Сигнатура-литерала |
Сигнатура-операции

Сигнатура-литерала ::
Имя-операции-литерала
Результат

Сигнатура-операции ::
Имя-операции
Список-аргументов
Результат

Список-аргументов = *Идентификатор-ссылки-на-сорт* +

Результат = *Идентификатор-ссылки-на-сорт*

Идентификатор-ссылки-на-сорт =
Идентификатор-сорта |
Идентификатор-подтипа

Имя-операции-литерала = *Имя*

Имя-операции = *Имя*

Идентификатор-сорта = *Идентификатор*

Равенство =
Неквантифицированное-равенство |
Квантифицированные-равенства |
Условное-равенство |
Неформальный-текст

Неквантифицированное-равенство ::
Терм
Терм

Квантифицированные-равенства :: *Имя-значения-set*
Идентификатор-сорта
Равенства

Имя-значения = *Имя*

Терм = *Пассивный-терм* |
Составной-терм |
Терм-ошибка

Составной-терм :: *Идентификатор-значения* |
Идентификатор-операции Терм + |
Условный-составной-терм

Идентификатор-операции = *Идентификатор*

Идентификатор-значения = *Идентификатор*

Пассивный-терм :: *Идентификатор-операции-литерала* |
Идентификатор-операции Пассивный-терм |
Условный-пассивный-терм

Идентификатор-операции-литерала = *Идентификатор*

Условное-равенство :: *Ограничение-set*
Ограничиваемое-равенство

Ограничение = *Неквантифицированное-равенство*

Ограничиваемое-равенство = *Неквантифицированное-равенство*

Условный-составной-терм = *Условный-терм*

Условный-пассивный-терм = *Условный-терм*

Условный-терм :: *Условие*
Следствие
Альтернатива

Условие = *Терм*

Следствие = *Терм*

Альтернатива = *Терм*

Терм-ошибка :: ( )

Идентификатор-подтипа = *Идентификатор*

Определение подтипа :: *Имя-подтипа*
Идентификатор-родительского-сорта
Условие-на-диапазон

Имя подтипа = *Имя*

Идентификатор-родительского-сорта = *Идентификатор-сорта*

Условие-на-диапазон :: *Идентификатор-операции-OR*
*Элемент-условия-***set**

Элемент-условия = *Открытый-диапазон* | *Замкнутый-диапазон*

Открытый диапазон :: *Идентификатор-операции*
Пассивное-выражение

Замкнутый-диапазон :: *Идентификатор-операции-AND*
Открытый-диапазон
Открытый-диапазон

Идентификатор-операции-OR = *Идентификатор*

Идентификатор-операции-AND = *Идентификатор*

Выражение = *Пассивное-выражение* |
Активное-выражение

Пассивное-выражение :: *Пассивный-терм*

Доступ-к-переменной = *Идентификатор-переменной*

Активное-выражение = *Доступ-к-переменной* |
Условное-выражение |
Применение-операции |
Императивная-операция

Императивная-операция = *Выражение-"сейчас"*
Выражение-PId
Обозревающее-выражение
Выражение-активности-таймера

Выражение-"сейчас" :: ( )

Выражение-PId = *Выражение "сам"* |
Выражение-"родитель" |
Выражение-"потомок" |
Выражение-"отправитель" |

Выражение-"сам" :: ( )

Выражение-"родитель" :: ( )

Выражение-"потомок" :: ( )

Выражение-"отправитель" :: ( )

Обозревающее-выражение :: *Идентификатор-переменной*
Выражение

Выражение-активности-таймера :: *Идентификатор-таймера*
*Выражение**

Условное-выражение :: *Булевское-выражение*
Выражение-следствие
Выражение-альтернатива

Булевское-выражение = *Выражение*

Выражение-следствие = *Выражение*

Выражение-альтернатива = *Выражение*

Применение-операции :: *Идентификатор-операции*
Выражение +

Предложение-присваивания :: *Идентификатор-переменной*
Выражение

ПРИЛОЖЕНИЕ C1
(к Рекомендации Z.100)

Сводное описание конкретного графического синтаксиса

C.1.1 Введение

C.1.1.1 *Метаязык*

Для графической грамматики метаязык, описанный в Рекомендации по SDL, § 1.5.2, расширен с помощью нижеследующих метасимволов:

a) **contains** (содержит);
b) **is associated with** (ассоциирован с);
c) **is followed by** (за ним следует);
d) **is connected to** (присоединен к);
e) **set** (совокупность).

Метасимвол **set** является постфиксной операцией, воздействующей на непосредственно предшествующие ему синтаксические элементы, заключенные в фигурные скобки, и обозначает (неупорядоченную) совокупность объектов. Таким объектом может быть произвольный синтаксический элемент. В этом случае объект должен быть расширен, прежде чем будет применен метасимвол **set**. Пример.

{< область текста системы >} * {< диаграмма макроса >} * < область взаимодействия блоков > } **set**

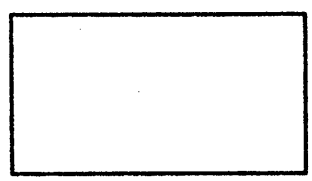
является совокупностью, содержащей ноль или более < областей системного текста>, ноль или более < диаграмм макроса> и одиу <область взаимодействия блоков >.

Все остальные метасимволы являются инфиксными операциями, для которых левым аргументом является нетерминальный графический символ. Правым аргументом является либо группа синтаксических элементов, помещенная внутри фигурных скобок, либо единственный синтаксический элемент. Если в правой части правила порождения на первом месте стоит нетерминальный графический символ и имеются еще несколько инфиксных операций, то графический нетерминальный символ является левым аргументом каждой из этих инфиксных операций. Графическим нетерминальным символом является нетерминальный объект, в котором имеется слово "символ" непосредственно перед знаком "больше чем" >.

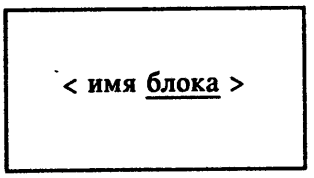
Метасимвол **contains** обозначает, что его правый аргумент должен быть помещен внутрь его левого аргумента и присоединенного к нему <символа расширения текста >, если таковой имеется. Пример:

<ссылка на блок > :: =
<символ блока > **contains** <имя блока >

< символ блока > :: =



означает следующее:



Метасимвол is associated with означает, что его правый аргумент логически ассоциируется с левым аргументом (как если бы он "содержался" в этом аргументе; однозначность "ассоциирования" обеспечивается правилами вычерчивания).

Метасимвол is followed by означает, что его правый аргумент следует (как логически, так и на чертеже) за его левым аргументом.

Метасимвол is connected to означает, что его правый аргумент присоединен (как логически, так и на чертеже) к его левому аргументу.

C1.1.2 *Общие правила*

C1.1.2.1 *Членение диаграмм*

Нижеследующее определение членения диаграмм не является частью *Конкретной графической грамматики*, но метаязык здесь используется тот же.

< страница > :: =
< символ рамки > contains
< область заголовка > < область номера страницы >
{ < синтаксическая единица > } *

< область заголовка > :: =
< неявный символ текста > contains < заголовок >

< область номера страницы > :: =
< неявный символ текста > contains [< номер страницы > [(< число страниц >)]]

< номер страницы > :: =
< имя литерала >

< число страниц > :: =
< имя натурального литерала >

< страница > является начальным нетерминальным объектом и поэтому ни в одном правиле порождения нет на него ссылок. Диаграмма может быть расчленена по нескольким < страницам > ; в этом случае < символ кадра >, ограничивающий диаграмму и < заголовок > диаграммы, заменяется на < символ кадра > и < заголовок > для каждой < страницы >.

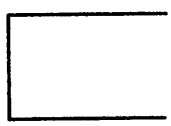
< неявный символ текста > не изображается, но подразумевается с тем, чтобы четко отделить друг от друга

< область заголовка> и < область номера страницы >. < область заголовка > помещается в левом верхнем углу < символа кадра >. < область номера страницы > помещается в правом верхнем углу < символа кадра >. < заголовок > и < синтаксическая единица > зависят от типа диаграммы.

C1.1.2.2 *Комментарий*

< область комментария > :: =
< символ комментария > contains < текст >
is connected to < символ пунктирной линии ассоциации >

< символ комментария > :: =



< символ пунктирной линии ассоциации > :: =

Один конец < символа пунктирной линии ассоциации > должен быть присоединен к середине вертикального отрезка < символа комментария >.

< символ комментария > может быть присоединен к любому графическому символу с помощью < символа пунктирной линии ассоциации >. < символ комментария > считается замкнутым символом за счет замыкания (в воображении) прямоугольника. Он содержит комментирующий текст, касающийся графического символа.

C.1.1.2.3 *Расширение текста*

< область расширения текста > :: =
< символ расширения текста > contains < текст >
is connected to < символ сплошной линии ассоциации >

< символ расширения текста > :: =
< символ комментария >

< символ сплошной линии ассоциации > :: =

< символ расширения текста > может быть присоединен к любому графическому символу с помощью < символа сплошной линии ассоциации >. < символ расширения текста > считается замкнутым за счет замыкания (в воображении) прямоугольника.

Один конец < символа сплошной линии ассоциации > должен быть присоединен к середине вертикального отрезка < символа расширения текста >.

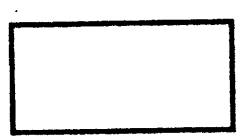
Текст, содержащийся в < символе расширения текста >, является продолжением текста, расположенного внутри графического символа, и считается содержащимся внутри этого символа.

C1.2 *Определение системы*

```
< конкретное определение системы > :: =
        {< определение системы > | < диаграмма системы >} {< удаленное определение >} *

< удаленное определение > :: =
        < определение >
    |   < диаграмма >

< диаграмма > :: =
        < диаграмма блока >
    |   < диаграмма процесса >
    |   < диаграмма процедуры >
    |   < диаграмма подструктуры блока >
    |   < диаграмма подструктуры канала >
    |   < диаграмма сервиса >
    |   < диаграмма макроса >
```

C1.3 *Диаграмма системы*

```
< диаграмма системы > :: =
        < символ кадра > contains
        {< заголовок системы >
        { { < область текста системы >} *
        {< диаграмма макроса >} *
                < область взаимодействия блоков >} set

< символ кадра > :: =
```



```
< заголовок системы > :: =
        SYSTEM < имя системы >

< область текста системы > :: =
        < символ текста > contains
        { < определение сигнала >
            |   < определение списка сигналов >
            |   < определение данных >
            |   < определение макроса >
            |   < определение отбора > } *
```

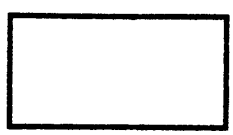
< символ текста > :: =

< область взаимодействия блоков > :: =
{< область блока > | < область определения канала >} +

< область блока > :: =
< графическая ссылка на блок >
| < диаграмма блока >

< графическая ссылка на блок > :: =
< символ блока > **contains** < имя блока >

< символ блока > :: =

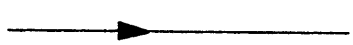


< область определения канала > :: =
< символ канала >
is associated with {< имя канала >
{[{ < идентификатор канала > | < идентификатор блока > }]
< область списка сигналов >
[< область списка сигналов >] } **set** }
is connected to { < область блока >
{< область блока > | < символ кадра >}
[< область ассоциирования подструктуры канала >] } **set**

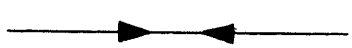
< идентификатор канала > идентифицирует внешний канал, присоединенный к < диаграмме подструктуры блока >, ограниченной < символом кадра >. < идентификатор блока > идентифицирует внешний блок, являющийся конечной точкой для < диаграммы подструктуры канала >, ограниченной < символом кадра >.

< символ канала > :: =
< символ канала 1 >
| < символ канала 2 >
| < символ канала 3 >

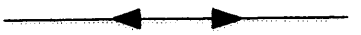
< символ канала 1 > :: =



< символ канала 2 > :: =

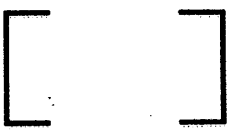


< символ канала 3 > :: =



< область списка сигналов > :: =
< символ списка сигналов > **contains** < список сигналов >

< символ списка сигналов > :: =



C1.4 *Диаграмма блока*

< диаграмма блока > :: =
< символ кадра >
contains { < заголовок блока >
{{ < область текста блока >} *
{ < диаграмма макроса > } *
[< область взаимодействия процессов >]
[< область подструктуры блока >] } **set**}
is associated with { < идентификатор канала > } *

< идентификатор канала > идентифицирует канал, присоединенный к маршруту сигнала в < диаграмме блока >. Этот канал помещается вне < символа кадра > впритык к той точке, в которой заканчивается маршрут сигнала на < символе кадра >. Если < диаграмма блока > не содержит < области взаимодействия процессов >, то она должна содержать < область подструктуры блока >.

< заголовок блока > :: =
BLOCK { < имя блока > | < идентификатор блока >}

< область текста блока > :: =
< область текста системы >

< область взаимодействия процессов > :: =
{ < область процесса >
| < область линии создания >
| < область определения маршрута сигнала > } +

< область процесса > :: =
< графическая ссылка на процесс >
| < диаграмма процесса >

< графическая ссылка на процесс > :: =
< символ процесса > **contains** { < имя процесса > [< число экземпляров >]}

< символ процесса >

< область линии создания > :: =
< символ линии создания >
is connected to { < область процесса > < область процесса >}

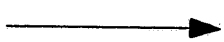
< символ линии создания > :: =

< область определения маршрута сигнала > .: =
< символ маршрута сигнала >
is associated with { < имя маршрута сигнала >
{ [< идентификатор канала >]
< область списка сигналов >
[< область списка сигналов>] } set}
is connected to
{< область процесса >} < область процесса > | < символ кадра >}} set

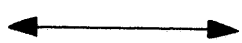
Если < символ маршрута сигнала > подсоединен к < символу кадра >, то < идентификатор канала > идентифицирует тот канал, к которому подсоединен маршрут сигнала.

< символ маршрута сигнала > :: =
< символ маршрута сигнала 1 >
| < символ маршрута сигнала 2 >

< символ маршрута сигнала 1 > :: =



< символ маршрута сигнала 2 > :: =



C1.5 *Диаграмма процесса*

< диаграмма процесса > :: =
< символ кадра >
contains { < заголовок процесса >
< область текста процесса >} *
{ < область процедуры >} *
{ < диаграмма макроса >} *
{ < область графа процесса > | < область взаимодействия сервисов >}} set}
[is associated with { < идентификатор маршрута сигнала >} +]

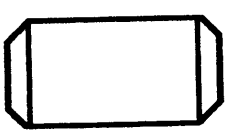
< Идентификатор маршрута сигнала > идентифицирует внешний маршрут сигнала, присоединенный к маршруту сигнала внутри < диаграммы процесса >. Он помещается вне < символа кадра > впритык к той точке, в которой заканчивается маршрут сигнала на < символе кадра >.

< заголовок процесса > :: =
PROCESS {< имя процесса > | < идентификатор процесса >}
[< число экземпляров > < конец >] [< формальные параметры >]

< область текста процесса > :: =
< символ текста > contains [< совокупность допустимых входных сигналов >]
{ < определение сигнала >
| < определение списка сигналов >
| < определение переменной >
| < определение обозревания >
| < определение импорта >
| < определение данных >
| < макроопределение >
| < определение таймера >
| < определение отбора >} *

< графическая ссылка на процедуры > :: =
< символ процедуры > contains < имя процедуры >

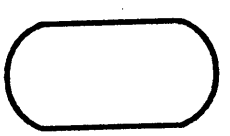
< символ процедуры > :: =



< область графа процесса > :: =
< область старта > { < область состояния > | < область входного коннектора > } *

< область старта > :: =
< символ старта > is followed by < область перехода >

< символ старта > :: =



< область перехода > :: =
[< область цепочки перехода >] is followed by
{ < область состояния >
| < область следующего состояния >
| < область принятия решения >
| < символ стоп >
| < область слияния >
| < область выходного коннектора >
| < символ возврата >
| < область необязательного перехода >}

< область слияния > :: =
< символ слияния > is connected to < символ линии потока >

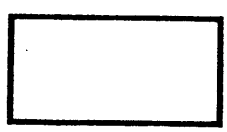
< символ слияния > :: =
< символ линии потока >

< символ линии потока > :: =

< область цепочки перехода > :: =
{< область работы >
| < область вывода >
| < область приоритетного вывода >
| < область установки >
| < область сброса >
| < область экспорта >
| < область запроса на создание >
| < область вызова процедуры >}
[is followed by < область цепочки перехода >]

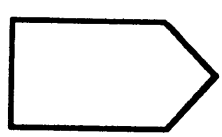
< область работы > :: =
< символ работы > contains < тело работы >

< символ работы > :: =



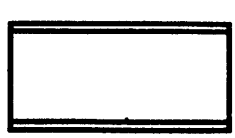
< область вывода > :: =
< символ вывода > contains < тело вывода >

< символ вывода > :: =



< область запроса на создание > :: =
< символ запроса на создание > contains < тело создания >

< символ запроса на создание > :: =

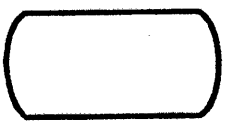


< область вызова процедуры > :: =
< символ вызова процедуры > contains < тело вызова процедуры >

< символ вызова процедуры > :: =

< область состояния > :: =
< символ состояния > contains < список состояний > is associated with
{ < область ассоциирования ввода >
| < область ассоциирования приоритетного ввода >
| < область ассоциирования непрерывного сигнала >
| < область ассоциирования сохранения > } *

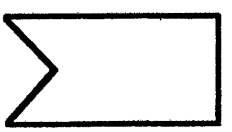
< символ состояния > :: =



< область ассоциирования ввода > :: =
< символ сплошной линии ассоциации > is connected to < область ввода >

< область ввода > :: =
< символ ввода > contains < список ввода >
is followed by { [< область разрешающего условия >] < область перехода > }

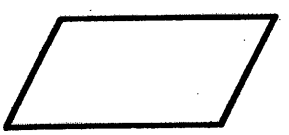
< символ ввода > :: =



< область ассоциирования сохранения > :: =
< символ сплошной линии ассоциации > is connected to < область сохранения >

< область сохранения > :: =
< символ сохранения > contains < список сохранения >

< символ сохранения > :: =



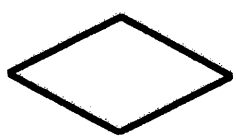
< область входного коннектора > :: =
< символ входного коннектора > contains < имя коннектора >
is followed by < область перехода >

< символ входного коннектора > :: =

< область следующего состояния > :: =
< символ состояния > contains < тело следующего состояния >

< область принятия решения > :: =
< символ принятия решения > contains < вопрос >
is followed by {{ < раздел графического ответа > < раздел графического иначе > } set
| { < раздел графического ответа > { < раздел графического ответа >} + [< раздел графического иначе >] {set}

< символ принятия решения > :: =



< графический ответ > :: =
< ответ > | (< ответ >)
< раздел графического ответа > :: =
< символ линии потока > is associated with < графический ответ >
is followed by < область перехода >

< раздел графического иначе > :: =
< символ линии потока > is associated with ELSE
is followed by < область перехода >

< область установки > :: =
< символ работы > contains < установка >

< область сброса > :: =
< символ работы > contains < сброс >

< символ стоп > :: =

< область выходного коннектора > :: =
< символ выходного коннектора > contains < имя коннектора >

< символ выходного коннектора > :: =
< символ входного коннектора >

C1.6 *Диаграмма процедуры*

< диаграмма процедуры > :: =
< символ кадра > contains { < заголовок процедуры >
{{ < область текста процедуры > } *
{< область процедуры >} *
{< диаграмма макроса > } *
< область графа процедуры > } set}

< заголовок процедуры > :: =
PROCEDURE {< имя процедуры > | < идентификатор процедуры >}
[< формальные параметры процедуры >]

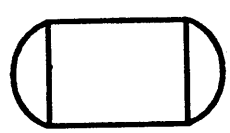
< область процедуры > :: =
< графическая ссылка на процедуру >
| < диаграмма процедуры >

< область текста процедуры > :: =
< символ текста > contains
{ < определение переменной >
| < определение данных >
| < макроопределение >
| < определение отбора > } *

< область графа процедуры > :: =
< область старта процедуры > { < область состояния > | < область входного коннектора > } *

< область старта процедуры > :: =
< символ старта процедуры > is followed by < область перехода >

< символ старта процедуры > :: =



< символ возврата > :: =

C1.7 *Подструктура блока*

< область подструктуры блока > :: =
< графическая ссылка на подструктуру блока >
| < диаграмма подструктуры блока >

< графическая ссылка на подструктуру блока > :: =
< символ подструктуры блока > **contains** < имя подструктуры блока >

< символ подструктуры блока > :: =
< символ блока >

< диаграмма подструктуры блока > :: =
< символ кадра >
contains { < заголовок подструктуры блока >
{{ < область текста подструктуры блока > } *
{ < диаграмма макроса > } *
< область взаимодействия блоков > } **set** }
is associated with { < идентификатор канала > } *

< идентификатор канала > идентифицирует канал, присоединенный к подканалу в < диаграмме подструктуры блока >. Он помещается вне < символа кадра > впритык к точке окончания подканала на <символе кадра >.

< заголовок подструктуры блока > :: =
SUBSTRUCTURE { < имя подструктуры блока > | < идентификатор подструктуры блока > }

< область текста подструктуры блока > :: =
< область текста системы >

C1.8 *Подструктура канала*

< область ассоциирования подструктуры канала > :: =
< символ пунктирной линии ассоциации >
is connected to < область подструктуры канала >

< область подструктуры канала > :: =
< графическая ссылка на подструктуру канала >
| < диаграмма подструктуры канала >

< графическая ссылка на подструктуру канала > :: =
< символ подструктуры канала > contains < имя подструктуры канала >

< символ подструктуры канала > :: =
< символ блока >

< диаграмма подструктуры канала > :: =
< символ кадра >
contains { < заголовок подструктуры канала >
{{ < область текста подструктуры канала > } *
{ < диаграмма макроса > } *
< область взаимодействия блоков > } set }
is associated with { < идентификатор блока > | ENV } +

< идентификатор блока > или ENV идентифицирует конечную точку канала, подвергшегося разбиению. < идентификатор блока > помещается вне < символа кадра > впритык к точке окончания ассоциированного подканала на < символе кадра >.

< заголовок подструктуры канала > :: =
SUBSTRUCTURE { < имя подструктуры канала >
| < идентификатор подструктуры канала > }

< область текста подструктуры канала > :: =
< область текста системы >

C1.9 *Макрос*

C1.9.1 *Диаграмма макроса*

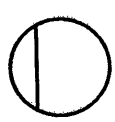
< диаграмма макроса > :: =
< символ кадра > contains {< заголовок макроса >< область тела макроса >}

< заголовок макроса > :: =
MACRODEFINITION < имя макроса > [< формальные макропараметры >]

< область тела макроса > :: =
{{ < любая область > *
< любая область > [is connected to < порт1 тела макроса >] } set
| { < любая область > is connected to < порт2 тела макроса >
< любая область > is connected to < порт2 тела макроса >
{< любая область > [connected to < порт2 тела макроса >]} *} set

< символ входной точки в макрос > :: =

< символ выходной точки из макроса > :: =



< порт1 тела макроса > :: =
< символ выходной точки > [is associated with < метка макроса >]
is connected to { < символ кадра >
| < символ входной точки в макрос >
| < символ выходной точки из макроса >}

< порт2 тела макроса > :: =
< символ выходной точки > is associated with < метка макроса >
is connected to < символ кадра >
| < символ входной точки в макрос >
| < символ выходной точки из макроса >

< метка макроса > :: =
< имя >

< символ выходной точки > :: =
< символ фиктивной выходной точки >
| < символ линии потока >
| < символ канала >
| < символ маршрута сигнала >
| < символ сплошной линии ассоциации >
| < символ пунктирной линии ассоциации >
| < символ линии создания >

< символ фиктивной выходной точки > :: =
< символ сплошной линии ассоциации >

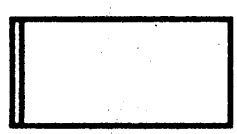
< любая область > :: =
< область текста системы >
| < область взаимодействия блоков >
| < область списка сигналов >
| < область блока >
| < область текста блока >
| < область взаимодействия процессов >
| < графическая ссылка на процедуру >
| < область процедуры >
| < область текста процесса >
| < область графа процесса >
| < область слияния >
| < область цепочки перехода >
| < область состояния >
| < область ввода >
| < область сохранения >
| < область расширения текста >
| < область ассоциирования подструктуры канала >
| < область подструктуры канала >
| < область подструктуры блока >
| < область приоритетного ввода >
| < область непрерывного сигнала >
| < область входного коннектора >
| < область следующего состояния >
| < область процесса >
| < область определения канала >
| < область линии создания >
| < область определения маршрута сигнала >
| < графическая ссылка на процесс >
| < диаграмма процесса >
| < область старта >
| < область вывода >
| < область установки >
| < область сброса >
| < область экспорта >
| < область приоритетного вывода >
| < область работы >
| < область запроса на создание >
| < область вызова процедуры >
| < область принятия решения >

| < область выходного коннектора >
| < область текста процедуры >
| < область графа процедуры >
| < область старта процедуры >
| < область текста подструктуры блока >
| < область взаимодействия блоков >
| < область сервиса >
| < область определения маршрута сигнала сервиса >
| < область текста сервиса >
| < область графа сервиса >
| < область старта сервиса >
| < область комментария >
| < область вызова макроса >

C1.9.2 *Вызов макроса*

< область вызова макроса > :: =
< символ вызова макроса > contains { < имя макроса > [< тело вызова макроса >]}
[is connected to
{ < порт1 вызова макроса >| < порт 2 вызова макроса > { < порт2 вызова макроса >} + }]

< символ вызова макроса > :: =



< порт1 вызова макроса > :: =
< символ входной точки > [is associated with < метка макроса >]
is connected to < любая область >

< порт2 вызова макроса > :: =
< символ входной точки > is associated with < метка макроса >
is connected to < любая область >

< символ входа > :: =
< символ фиктивной входной точки >
| < символ линии потока >
| < символ канала >
| < символ маршрута сигнала >
| < символ сплошной линии ассоциации >
| < символ пунктирной линии ассоциации >
| < символ линии создания >

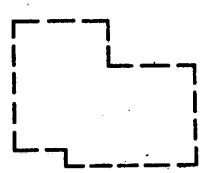
< символ фиктивной входной точки > :: =
< символ сплошной линии ассоциации >

C1.10 *Родовые системы*

C1.10.1 *Определение отбора*

< область отбора > :: =
< символ отбора > contains
{SELECT IF (< булевское простое выражение >)
{ < область блока >
| < область подструктуры канала >
| < область текста системы >
| < область текста блока >
| < область текста процесса >
| < область текста процедуры >
| < область текста подструктуры блока >
| < область текста подструктуры канала >
| < область текста сервиса >
| < диаграмма макроса >
| < область отбора >
| < область процесса >
| < область определения маршрута сигнала >
| < область линии создания >
| < область процедуры >
| < область сервиса >
| < область определения маршрута сигнала сервиса > } +}

< символом отбора > является многоугольник со сплошными углами, сторонами которого являются отрезки пунктирной линии. Например,



C1.10.2 *Выбираемый переход*

< область выбираемого перехода > :: =
< символ выбираемого перехода > contains {< вопрос альтернативы >}
is followed by {< выбираемая выходная точка1 >} < выбираемая выходная точка1 > | < выбираемая выходная точка 2 >
{<выбираемая выходная точка1> } * } set.
< символ выбираемого перехода > :: =

< выбираемая выходная точка1 > :: =
< символ линии потока > is associated with < графический ответ >
is followed by { < область перехода >|< область слияния >}

< выбираемая выходная точка2 > :: =
< символ линии потока > is associated with ELSE
is followed by { < область перехода > | < область слияния > }

C1.11 *Сервис*

C1.11.1 *Разложение на сервисы*

< область взаимодействия сервисов > :: =
{ < область сервиса > | < область определения маршрута сигнала > } +

< область сервиса > :: =
< графическая ссылка на сервис >
| < диаграмма сервиса >

< графическая ссылка на сервис > :: =
< символ сервиса > contains < имя сервиса >

< символ сервиса > :: =

< область определения маршрута сигнала сервиса > :: =
< символ маршрута сигнала >
is associated with { < имя маршрута сигнала сервиса > [< идентификатор маршрута сигнала >]
< область списка сигналов > [< область списка сигналов >] } set
is connected to { < область сервиса >
{ < область сервиса > | < символ кадра > } } set

Если < символ маршрута сигнала > присоединен к < символу кадра >, то это означает, что < идентификатор маршрута сигнала > идентифицирует внешний маршрут сигнала, к которому подсоединен данный маршрут сигнала.

C1.11.2 *Диаграмма сервиса*

< диаграмма сервиса > :: =
< символ кадра > contains
{ < заголовок сервиса >
{{ < область текста сервиса > } *
{ < область процедуры > } *
{ < диаграмма макроса > } *
< область графа сервиса > } set }

< заголовок сервиса > :: =
SERVICE { < имя сервиса > | < идентификатор сервиса > }

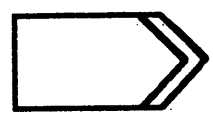
< область текста сервиса > :: =
< символ текста > **contains**
{ < определение переменной >
| < определение обозревания >
| < определение импорта >
| < определение данных >
| < макроопределение >
| < определение таймера >
| < определение отбора > } *

< область графа сервиса > :: =
< область графа процесса >

< область ассоциирования приоритетного ввода > :: =
< символ сплошной линии ассоциации > **is connected to** < область приоритетного ввода >

< область приоритетного ввода > :: =
< символ приоритетного ввода > **contains** < приоритетный входной список >

< символ приоритетного ввода > :: =



< область приоритетного вывода > :: =
< символ приоритетного вывода > **contains** < тело приоритетного вывода >

< символ приоритетного вывода > :: =

C1.12 *Непрерывный сигнал*

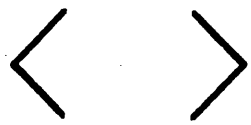
< область ассоциирования непрерывного сигнала > :: =
< символ сплошной линии ассоциации > **is connected to** < область непрерывного сигнала >

< область непрерывного сигнала > :: =
< символ разрешающего условия >
contains {< булевское выражение > [< конец > **PRIORITY** < имя целого литерала >]}
is followed by < область перехода >

C1.13 *Разрешающее условие*

< область разрешающего условия > :: =
< символ разрешающего условия > contains < булевское выражение >

< символ разрешающего условия > :: =



C1.14 *Экспорт*

< область экспорта > :: =

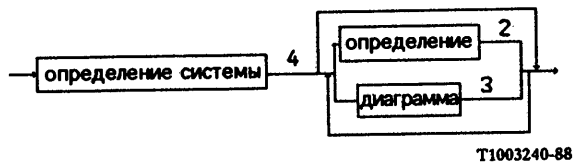
< символ работы > contains < экспорт >

ПРИЛОЖЕНИЕ C2

(к Рекомендации Z.100)

Сводное описание синтаксиса SDL/PR

1 *система*

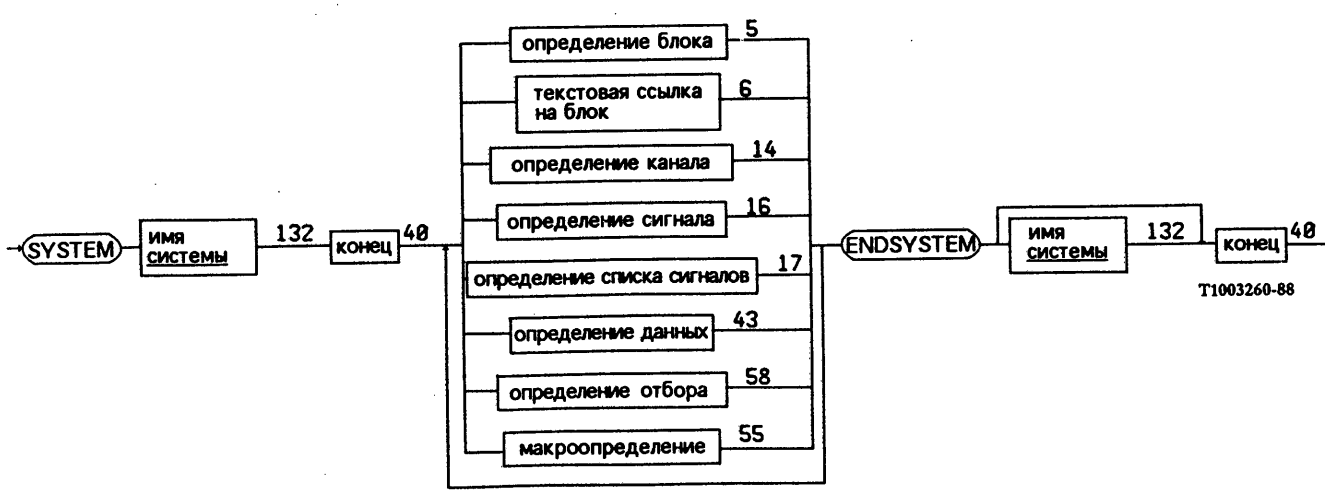


2 *определение*

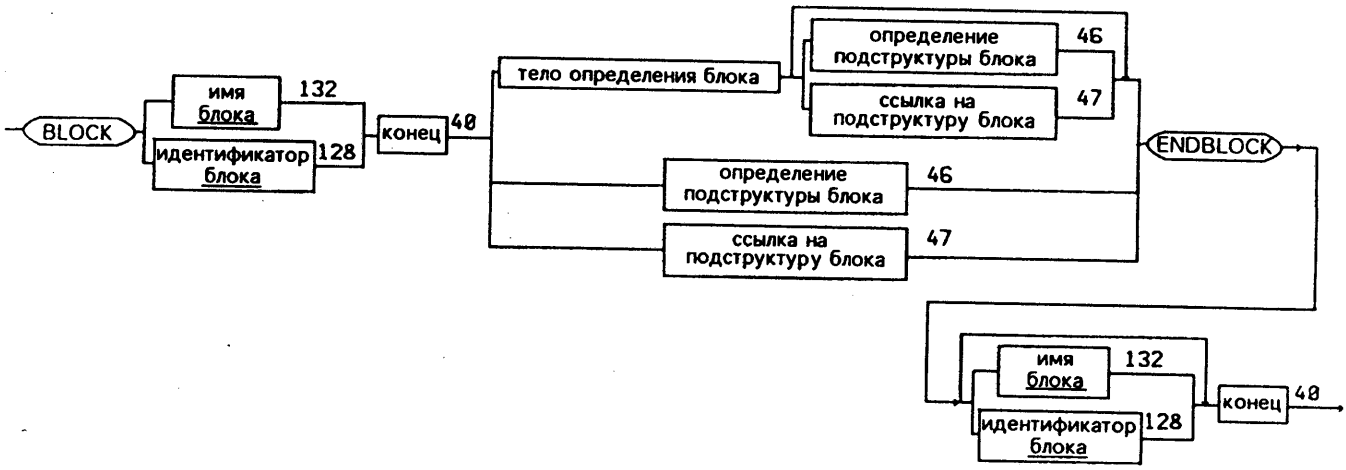
3 *диаграмма*

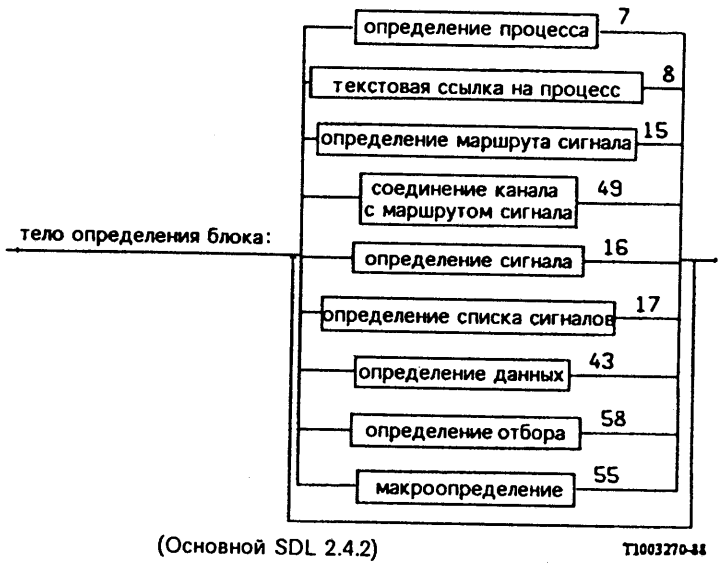
Диаграмма определена в сводном описании графического синтаксиса.

4 *определение системы* (Основной SDL 2.4.1)



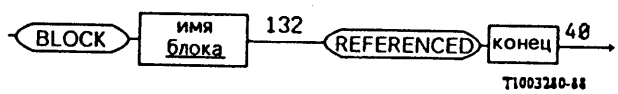
5 *определение блока* (Основной SDL 2.4.2)



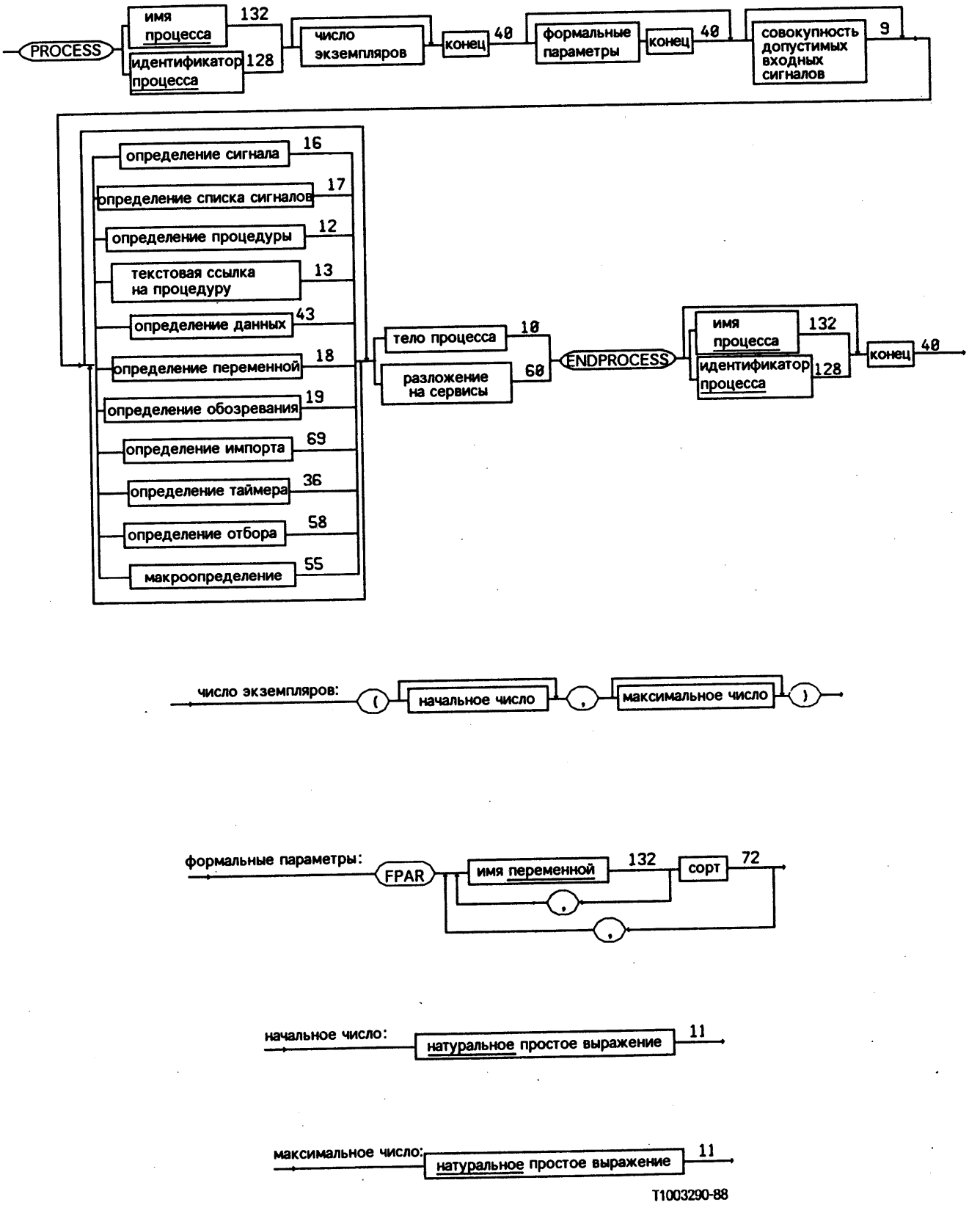


(Основной SDL 2.4.2) T1003270-88

6 *текстовая ссылка на блок* (Основной SDL 2.4.2)

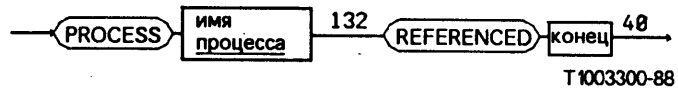


T1003280-88


PROCESS
имя процесса
132
идентификатор процесса
128
число экземпляров
конец
40
формальные параметры
конец
40
совокупность допустимых входных сигналов
9
определение сигнала
16
определение списка сигналов
17
определение процедуры
12
текстовая ссылка на процедуру
13
определение данных
43
определение переменной
18
определение обозревания
19
определение импорта
69
определение таймера
36
определение отбора
58
макроопределение
55
тело процесса
10
разложение на сервисы
60
ENDPROCESS
имя процесса
132
идентификатор процесса
128
конец
40
число экземпляров:
(
начальное число
,
максимальное число
)
формальные параметры:
FPAR
имя переменной
132
сорт
72
,
,
начальное число:
натуральное простое выражение
11
максимальное число:
натуральное простое выражение
11

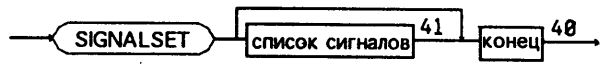
T1003290-88

8 *текстовая ссылка на процесс* (Основной SDL 2.4.3)



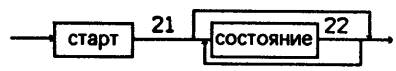
T1003300-88

9 *совокупность допустимых входных сигналов*



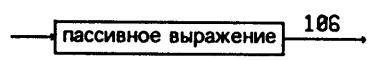
T1003310-88

10 *тело процесса* (Основной SDL 2.4.3)



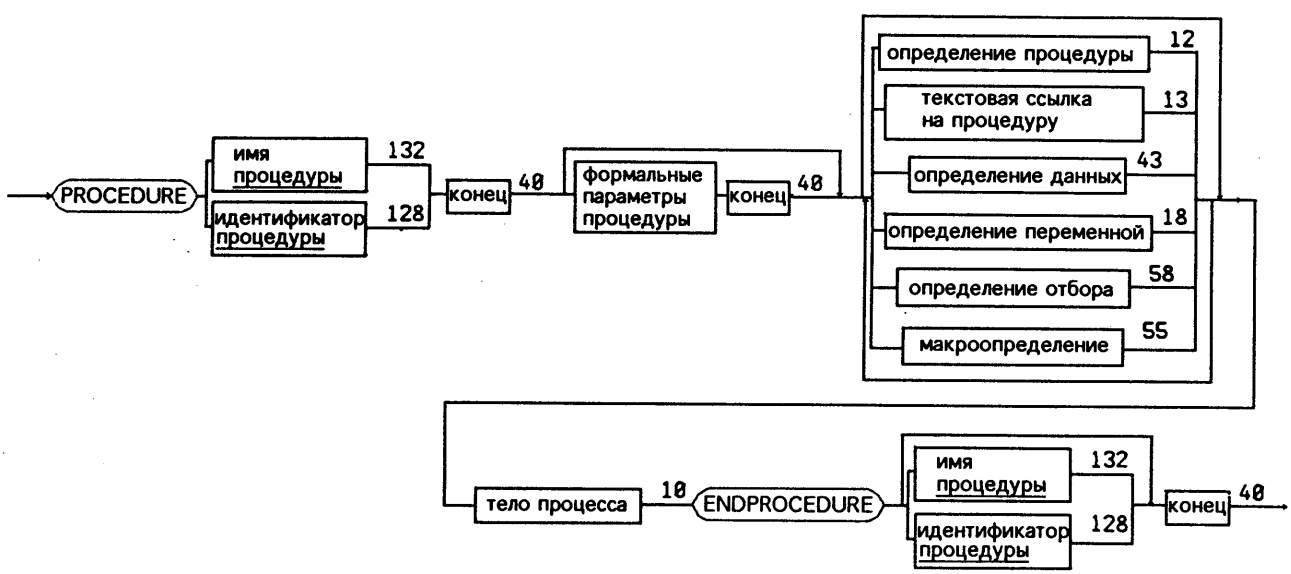
T1003320-88

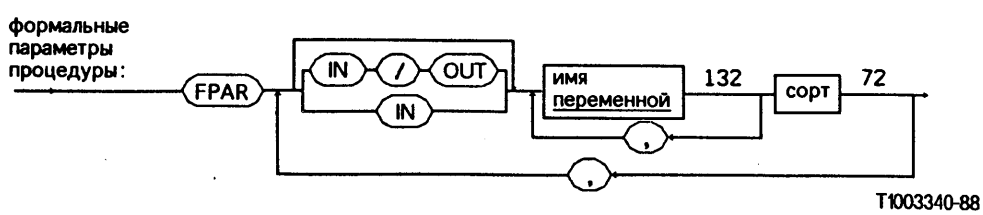
11 *простое выражение*



T1003330-88

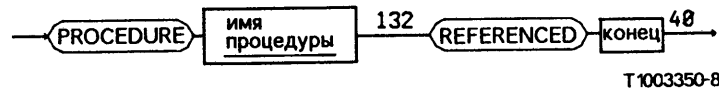
12 *определение процедуры* (Основной SDL 2.3.4)



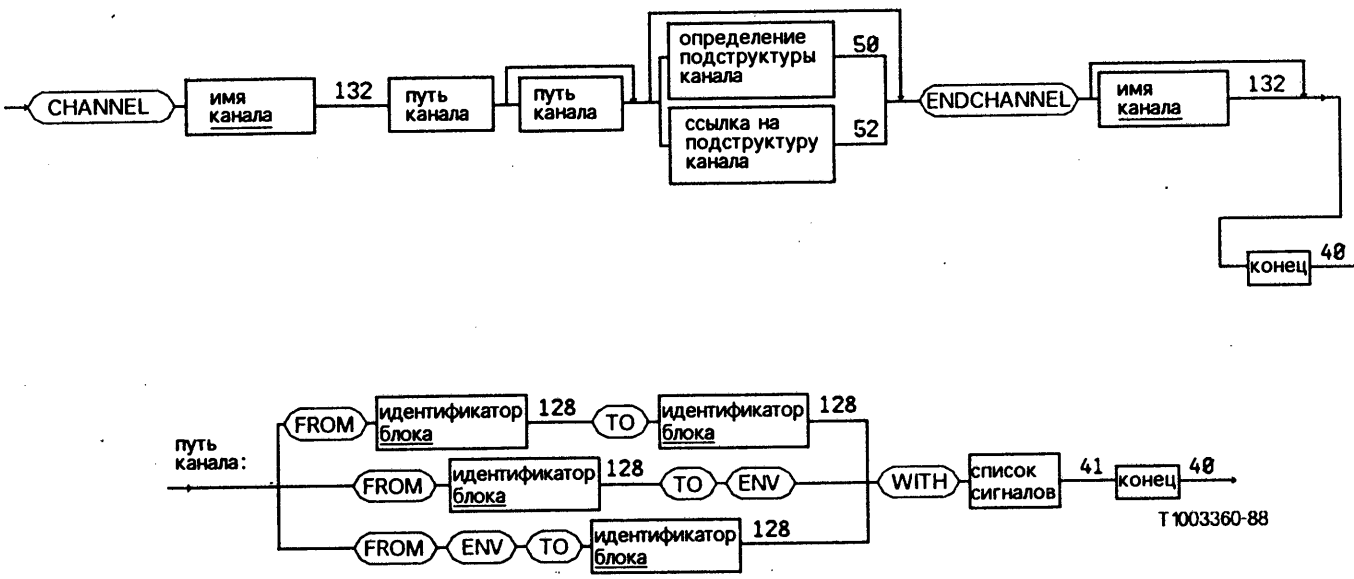


T1003340-88

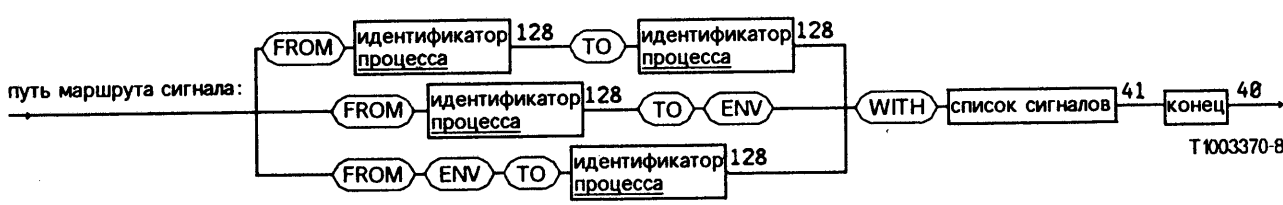
13 *текстовая ссылка на процедуру* (Основной SDL 2.3.4)



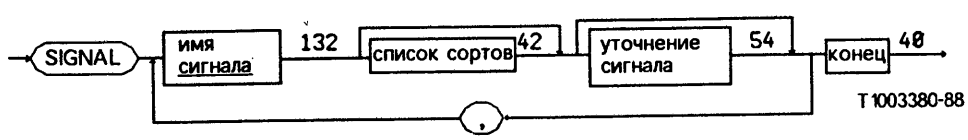
14 *определение канала* (Основной SDL 2.5.1)



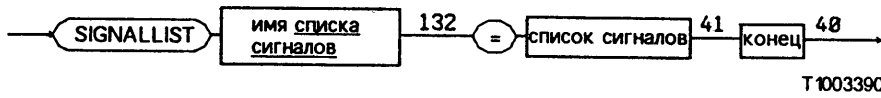
15 *определение маршрута сигнала* (Основной SDL 2.5.2)



16 *определение сигнала* (Основной SDL 2.5.4)

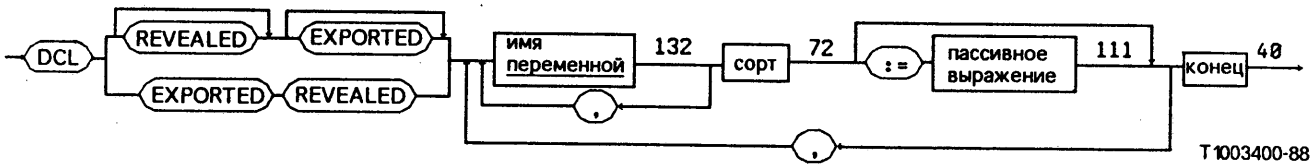


17 *определение списка сигналов* (Основной SDL 2.5.5)



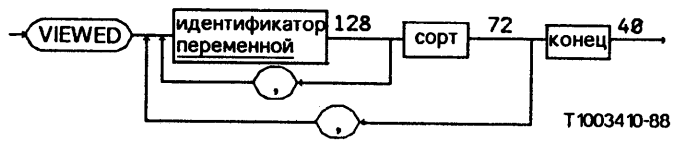
T1003390-88

18 *определение переменной* (Основной SDL 2.6.1.1)



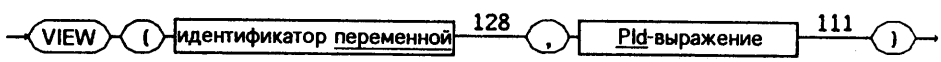
T1003400-88

19 *определение обозревания* (Основной SDL 2.6.1.2)



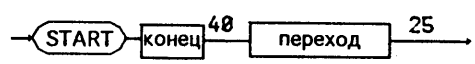
T1003410-88

20 *обозревающее выражение* (Данные 5.5.4.4)

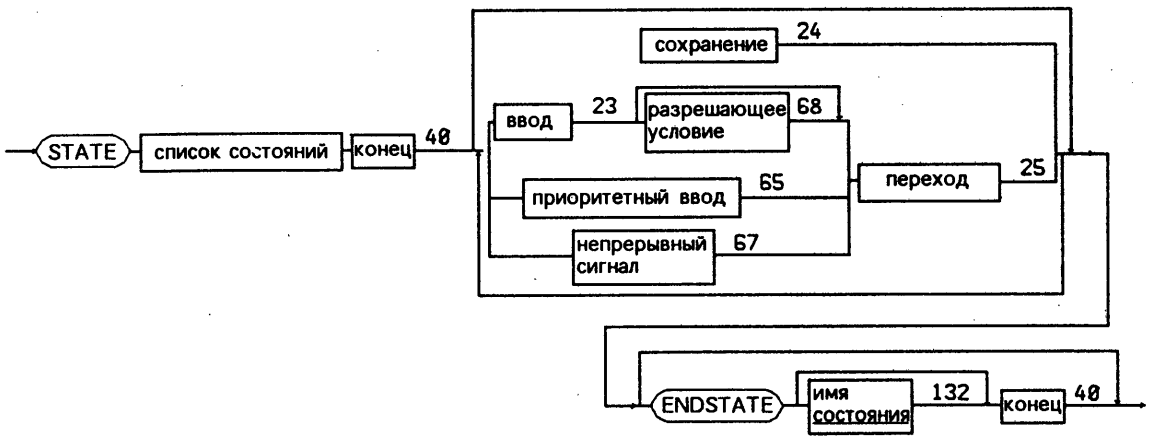


T1003420-88

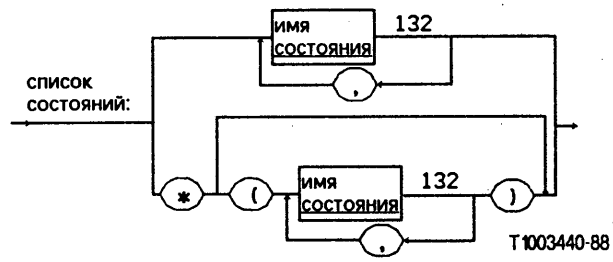
21 *старт* (Основной SDL 2.6.2)



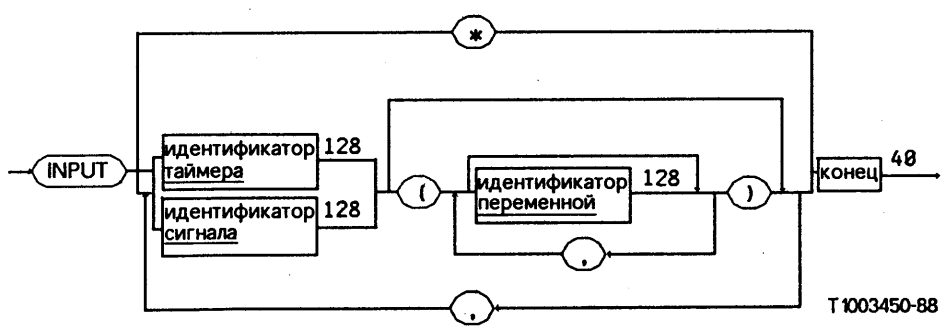
T1003430-88



(Дополнительные понятия 4.4)

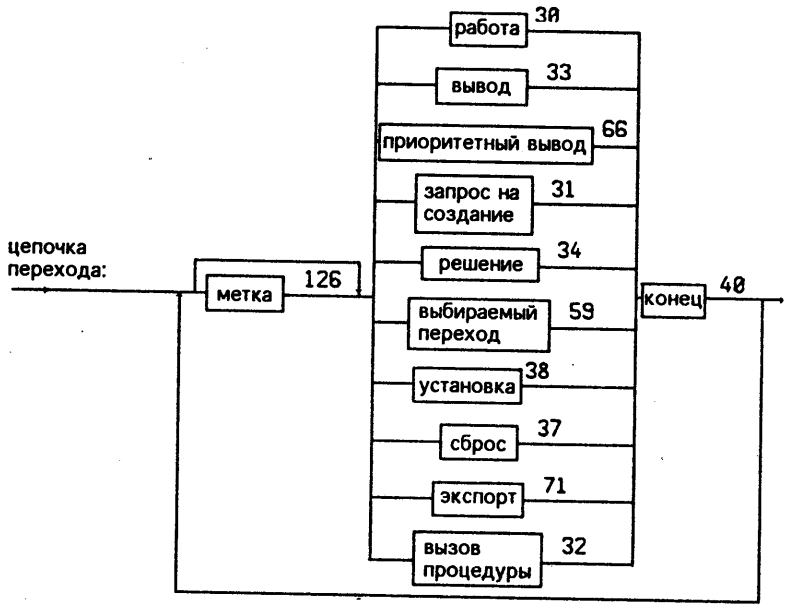


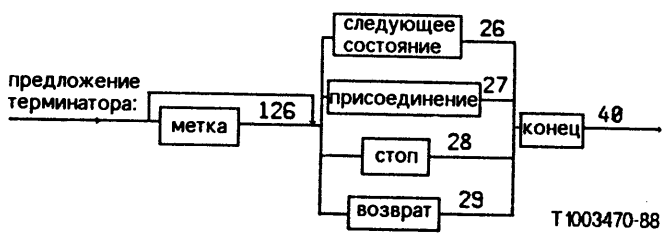
23 *ввод* (Основной SDL 2.6.4)



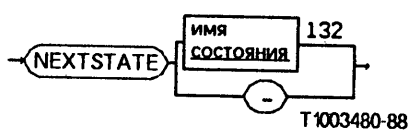
24 *сохранение* (Основной SDL 2.6.5)

25 *переход* (Основной SDL 2.6.7)





26 *следующее состояние* (Основной SDL 2.6.7.2.1)



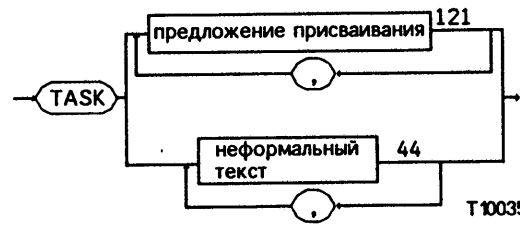
27 *присоединение* (Основной SDL 2.6.7.2.2)

28 *стоп* (Основной SDL 2.6.7.2.3)

29 *возврат* (Основной SDL 2.6.7.2.4)

T1003510-88

30 *работа* (Основной SDL 2.7.1)

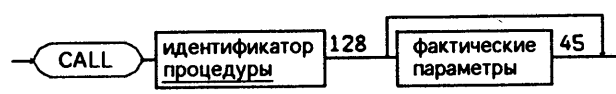


T1003520-88

31 *запрос на создание* (Основной SDL 2.7.2)

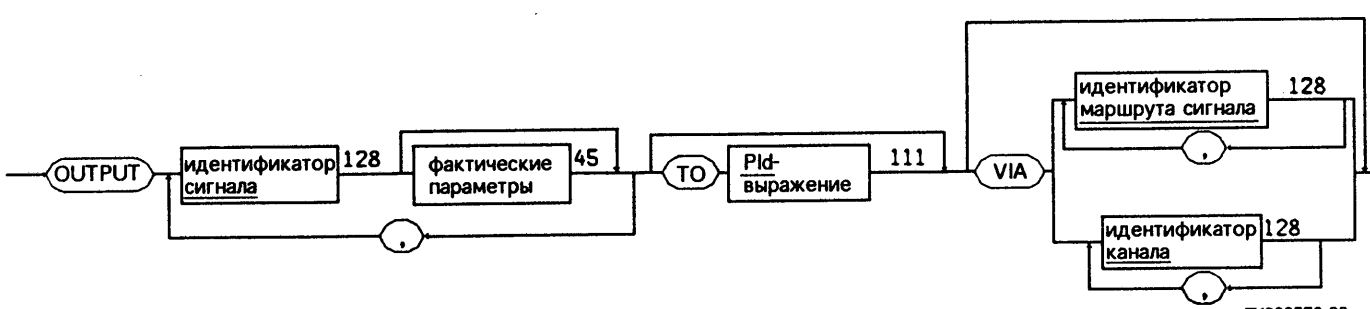
T1003530-88

32 *вызов процедуры* (Основной SDL 2.7.3)



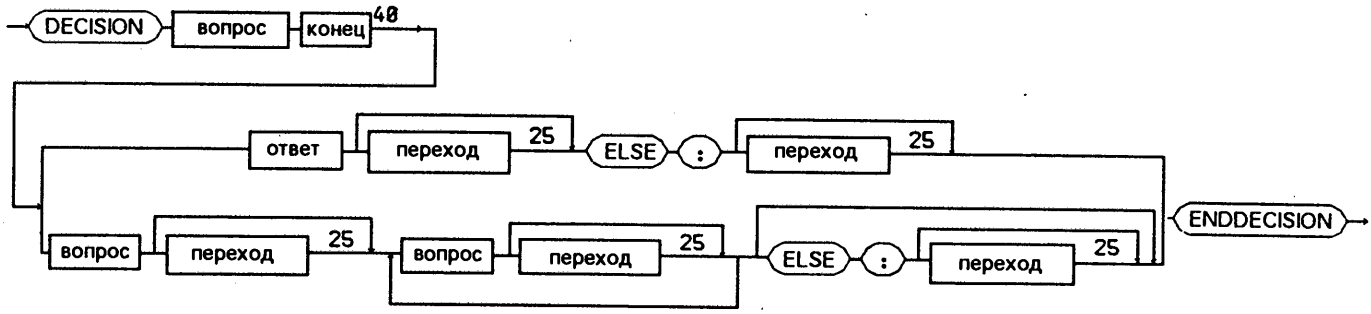
T1003540-88

33 *вывод* (Основной SDL 2.7.4)

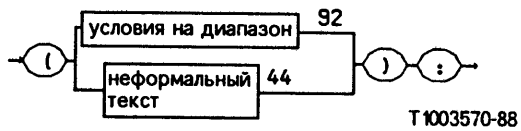


T1003550-88

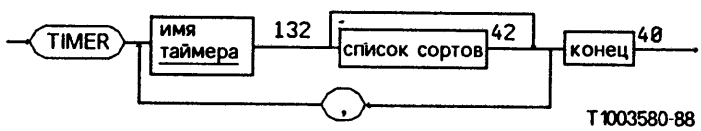
34 *решение* (Основной SDL 2.7.5)



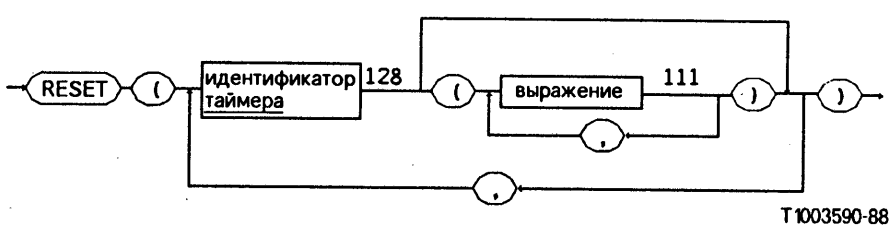
35 *ответ* (Основной SDL 2.7.5)



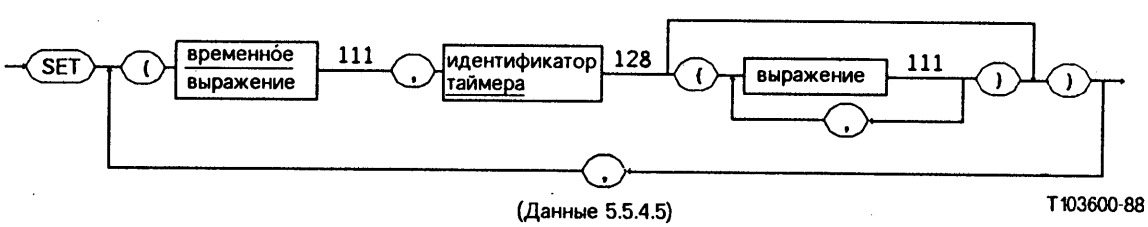
36 *определение таймера* (Основной SDL 2.8)



37 *сброс* (Основной SDL 2.8)

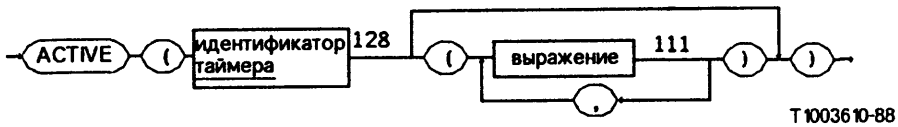


38 *установка* (Основной SDL 2.8)



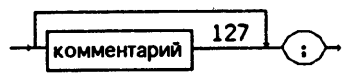
(Данные 5.5.4.5)

39 *выражение активности таймера* (Данные 5.5.4.5)



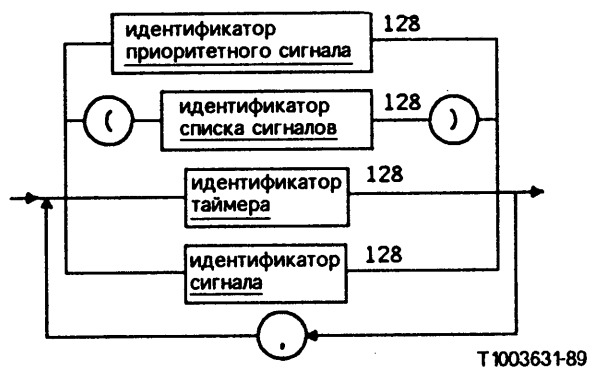
T1003610-88

40 *конец*



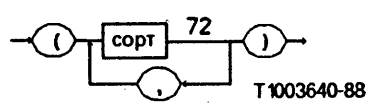
T1003620-88

41 *список сигналов*



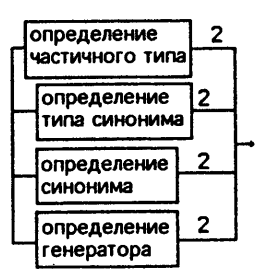
T1003631-89

42 *список сортов*



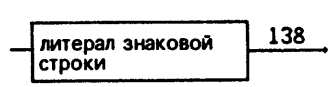
T1003640-88

43 *определение данных*



T1003650-88

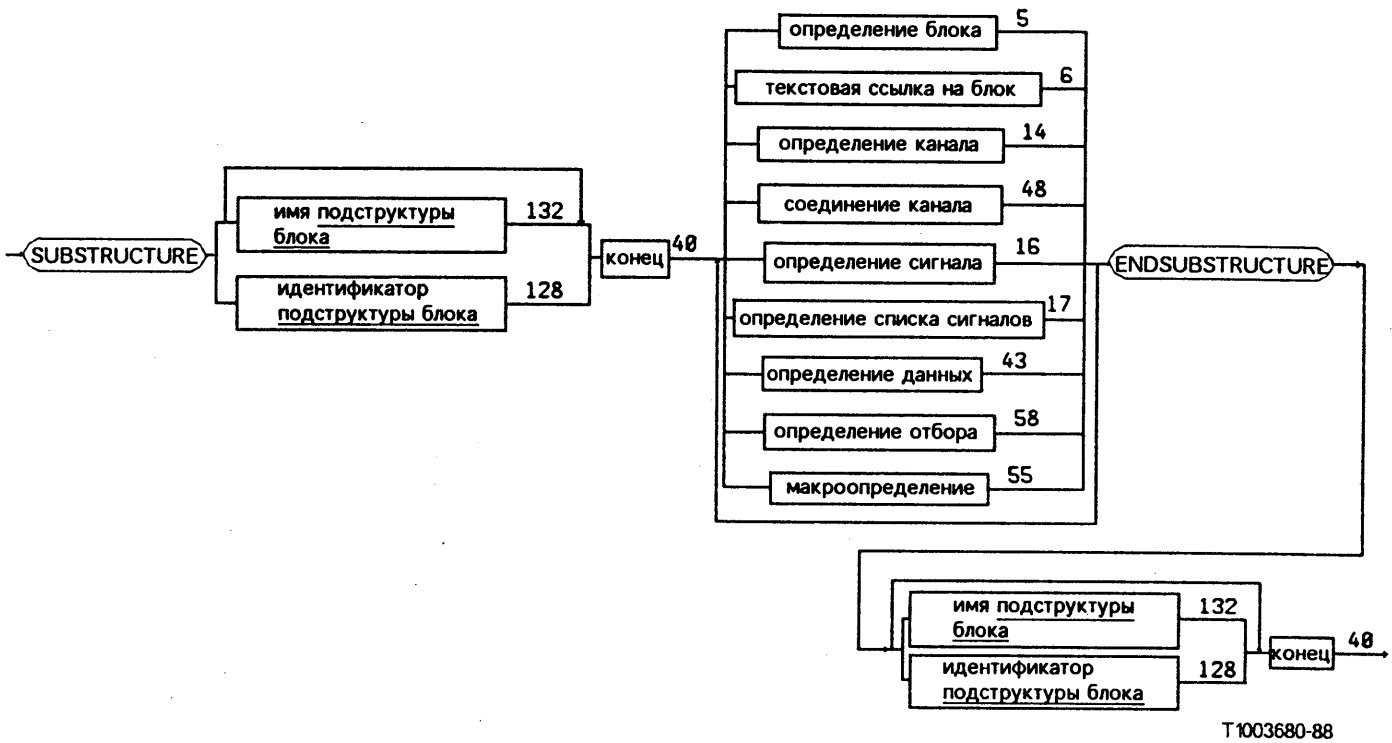
44 *неформальный текст*



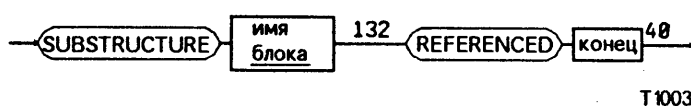
T1003660-88

45 *фактические параметры*

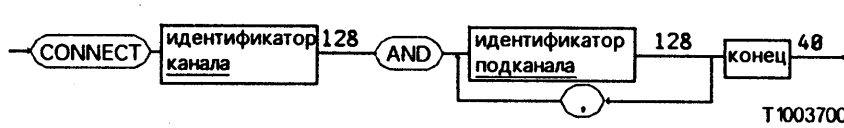
46 *определение подструктуры блока* (Концепции структурирования 3.2.2)



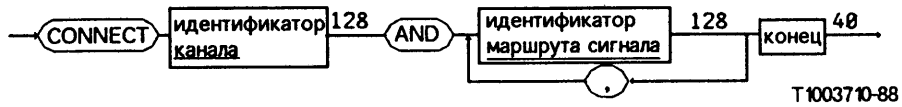
47 *ссылка на подструктуру блока* (Концепции структурирования 3.2.2)



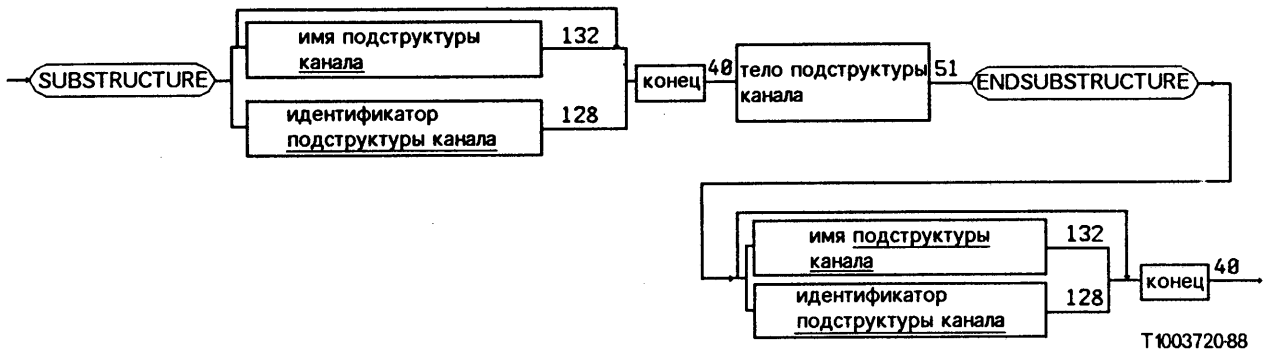
48 *соединение канала* (Концепции структурирования 3.2.2)



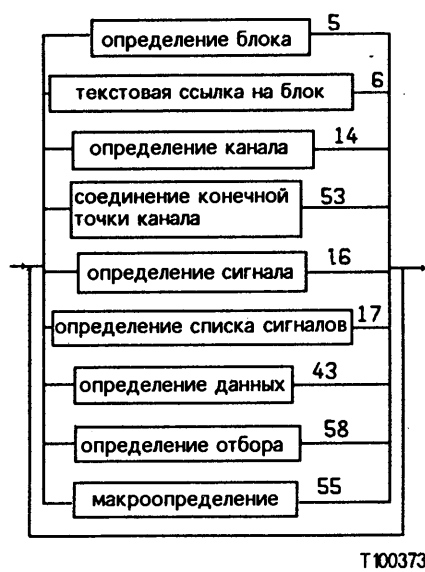
49 *соединение канала с маршрутом сигнала* (Концепции структурирования 3.2.2)



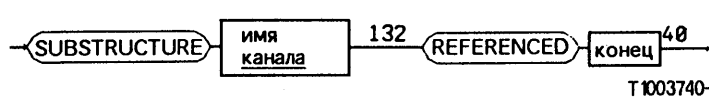
50 *определение подструктуры канала* (Концепции структурирования 3.2.3)



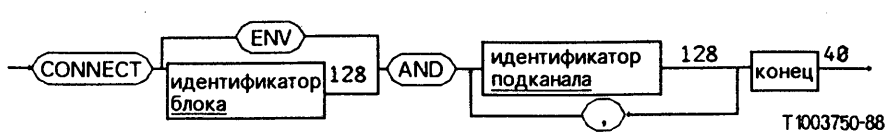
51 *тело подструктуры канала* (Концепции структурирования 3.2.3)



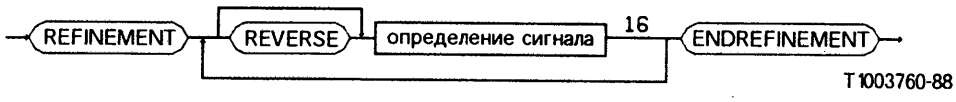
52 *ссылка на подструктуру канала* (Основной SDL 2.5.1)



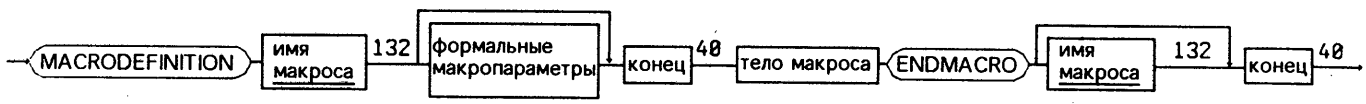
53 *соединение конечной точки канала* (Концепции структурирования 3.2.3)

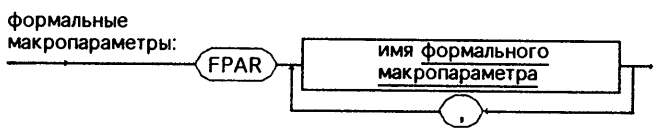


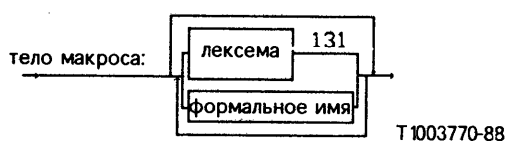
54 *уточнение сигнала* (Концепции структурирования 3.3)



55 *макроопределение* (Дополнительные понятия 4.2.2)

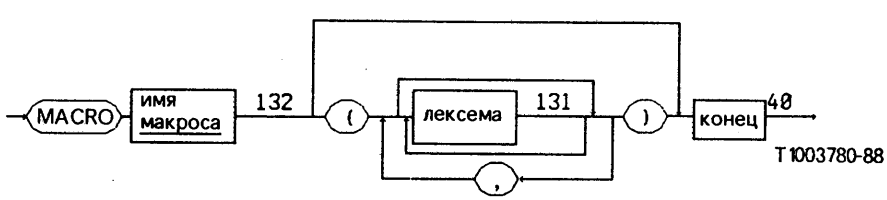




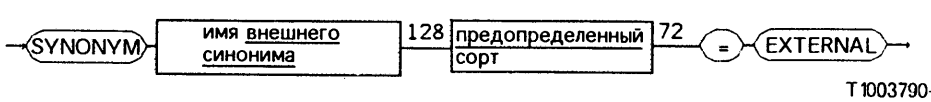


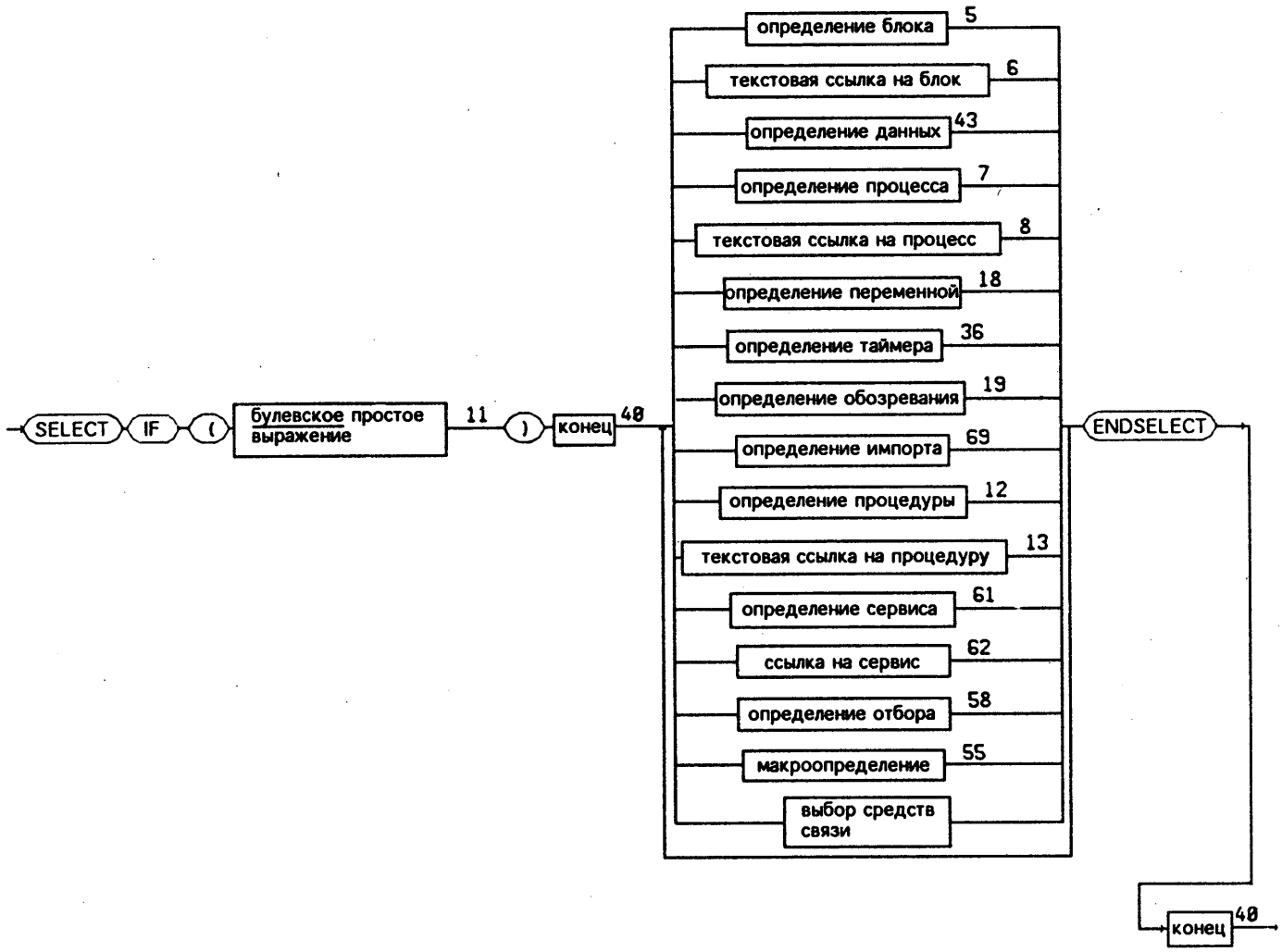
(Дополнительные понятия 4.2.3)

56 *вызов макроса* (Дополнительные понятия 4.2.3)



57 *определение внешнего синонима* (Дополнительные понятия 4.3.1)

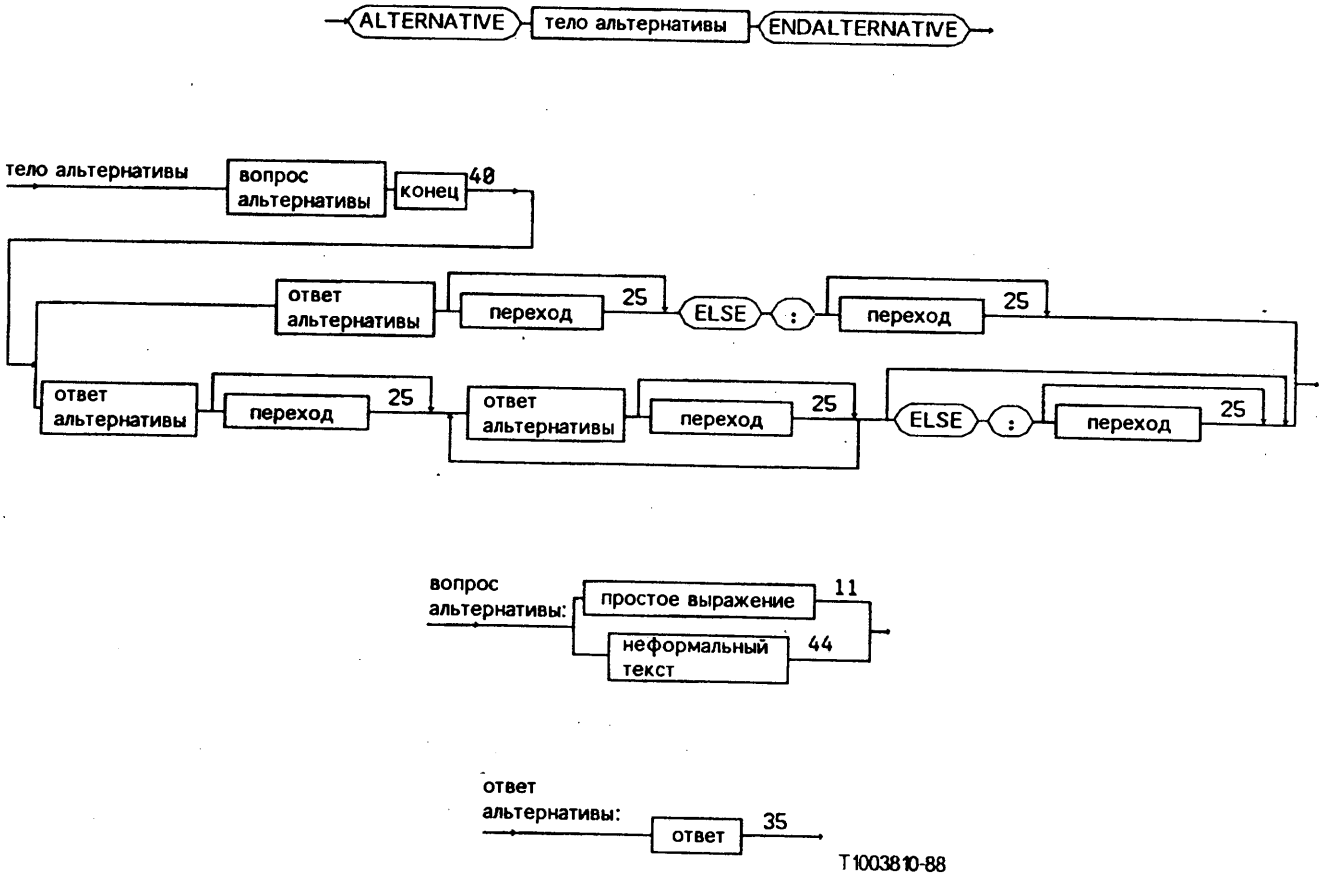



SELECT
IF
(
булевское простое выражение
11
)
конец
40
определение блока
5
текстовая ссылка на блок
6
определение данных
43
определение процесса
7
текстовая ссылка на процесс
8
определение переменной
18
определение таймера
36
определение обозревания
19
определение импорта
69
определение процедуры
12
текстовая ссылка на процедуру
13
определение сервиса
61
ссылка на сервис
62
определение отбора
58
макроопределение
55
выбор средств связи
ENDSELECT
конец
40

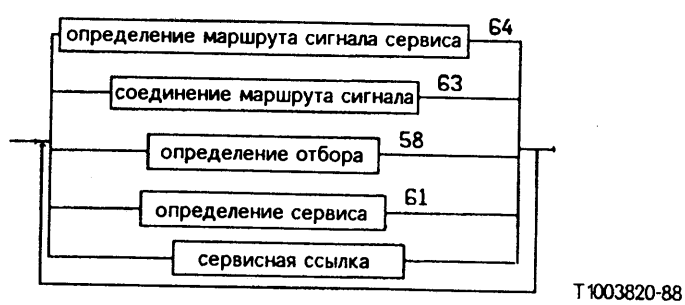

выбор средств связи:
соединение канала
48
определение канала
14
определение маршрута сигнала
2
определение маршрута сигнала сервиса
2
соединение канала с маршрутом сигнала
49
соединение конечной точки канала
53
соединение маршрута сигнала
63
определение сигнала
16
определение списка сигналов
17

T1003800-88

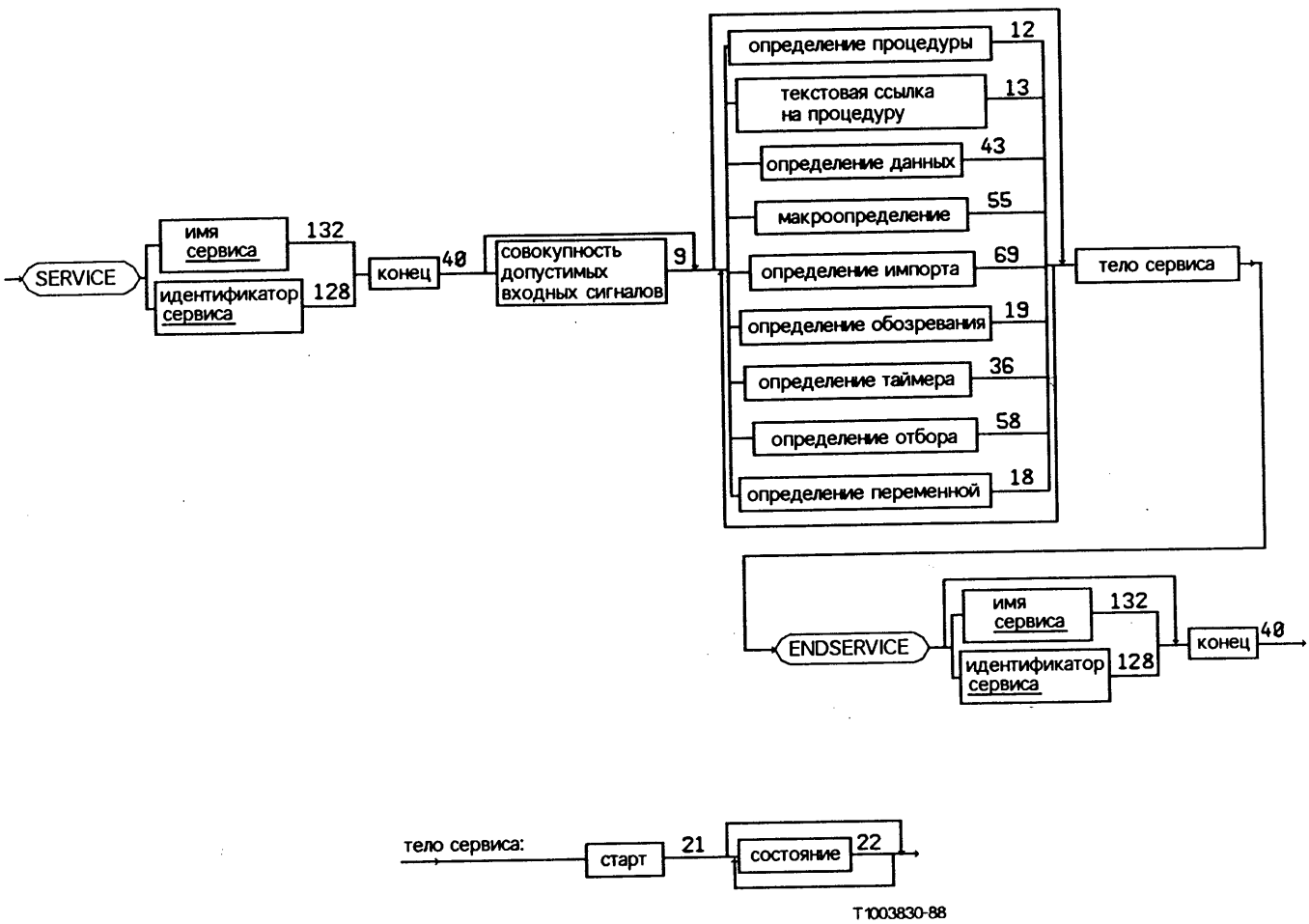
59 *выбираемый переход* (Дополнительные понятия 4.3.4)



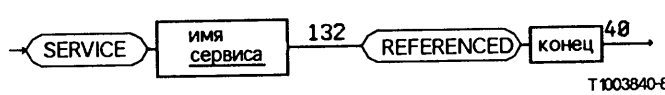
60 *разложение на сервисы* (Дополнительные понятия 4.10.1)



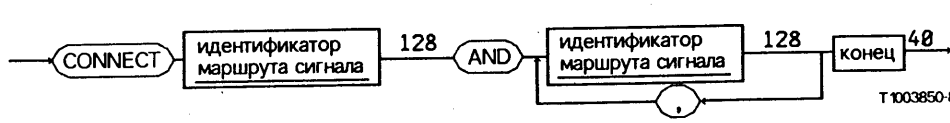
61 *определение сервиса* (Дополнительные понятия 4.10.2)



62 *ссылка на сервис* (Дополнительные понятия 4.10.1)

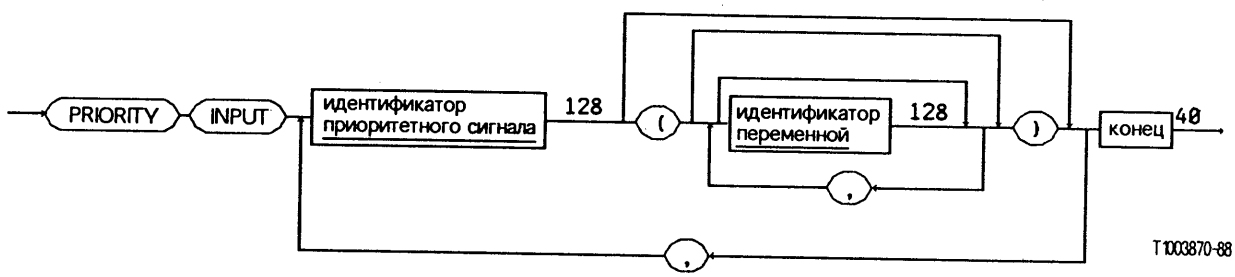


63 *соединение маршрута сигнала* (Дополнительные понятия 4.10.1)

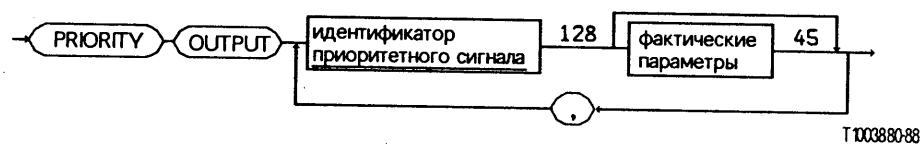


64 *определение маршрута сигнала сервиса* (Дополнительные понятия 4.10.1)

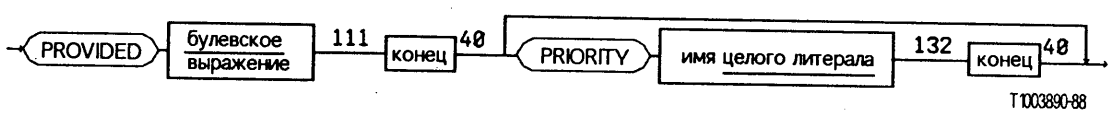
65 *приоритетный ввод* (Дополнительные понятия 4.10.2)



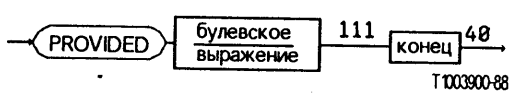
66 *приоритетный вывод* (Дополнительные понятия 4.10.2)



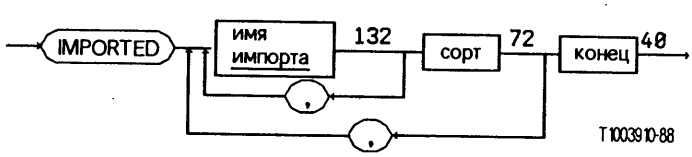
67 *непрерывный сигнал* (Дополнительные понятия 4.11)



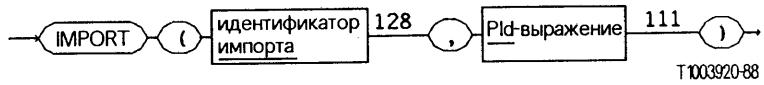
68 *разрешающее условие* (Дополнительные понятия 4.12)



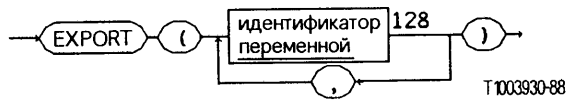
69 *определение импорта* (Дополнительные понятия 4.13)



70 *импортирующее выражение* (Дополнительные понятия 4.13)

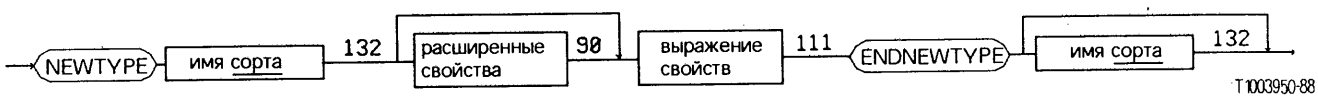


71 *экспорт* (Дополнительные понятия 4.13)

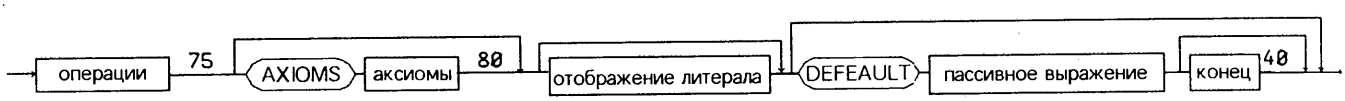


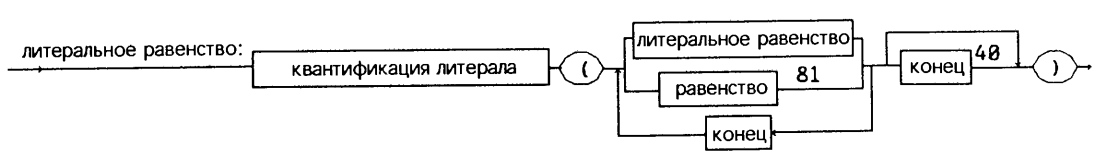
72 *сорт* (Данные 5.2.2)

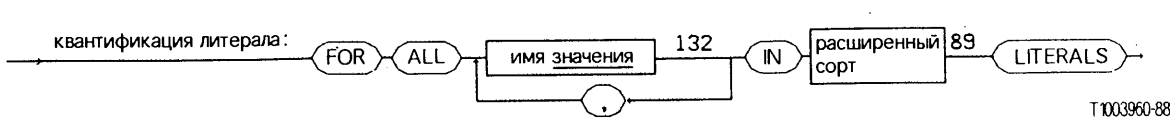
73 *определение частичного типа* (Данные 5.2.1)



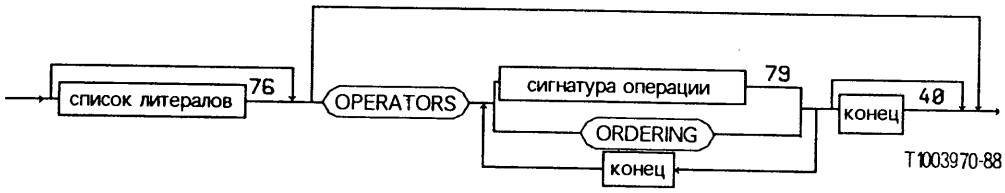
74 *выражение свойств* (Данные 5.2.1 и 5.5.3.3)





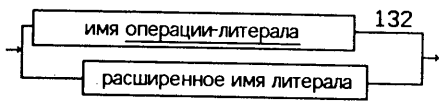


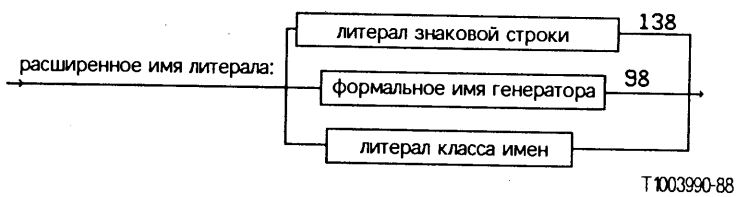
75 *операции* (Данные 5.2.2 и 5.4.1.8)



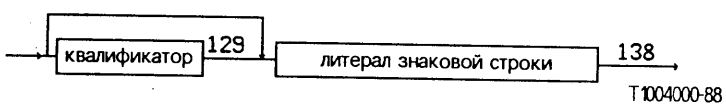
76 *список литералов* (Данные 5.2.2)

77 *сигнатура литерала* (Данные 5.2.2 и 5.4.1.8)

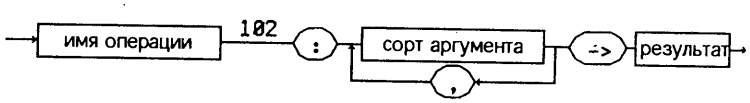


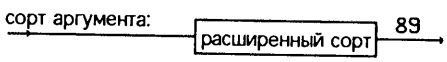


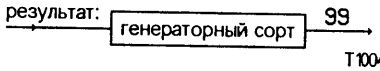
78 *идентификатор литерала знаковой строки*



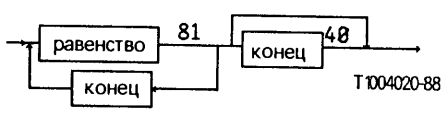
79 *сигнатура операции* (Данные 5.2.2)

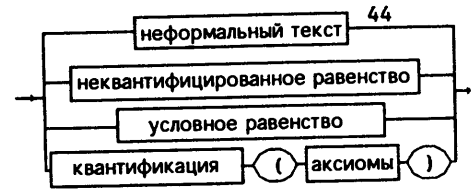


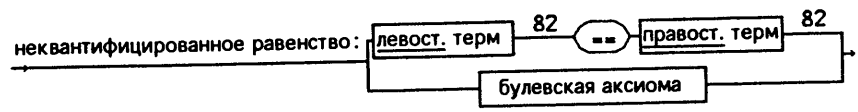


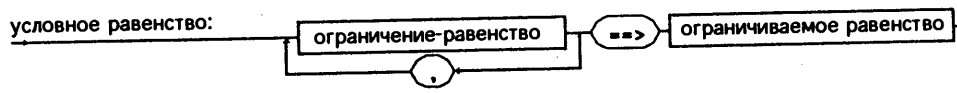


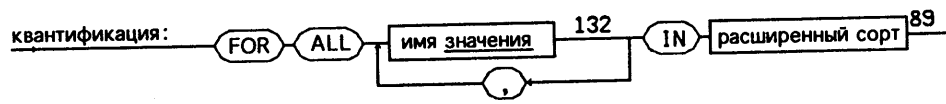
80 *аксиомы* (Данные 5.2.3 и 5.2.4)

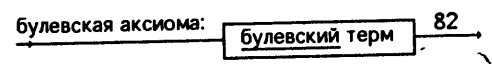




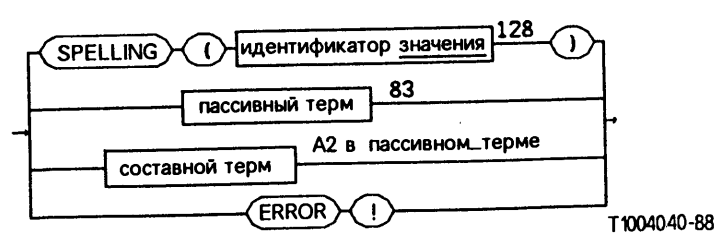




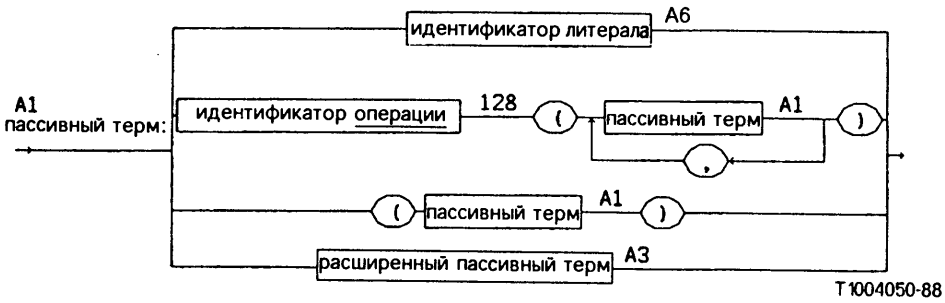


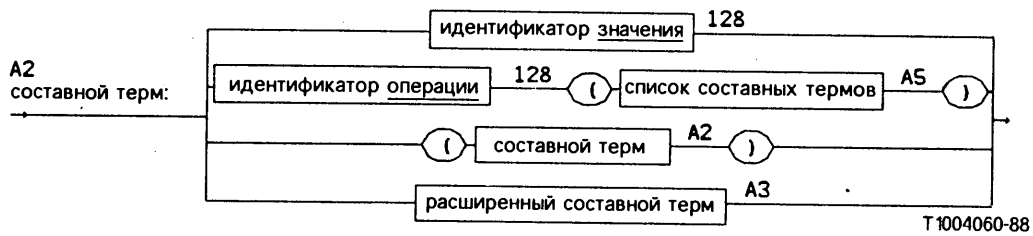


T1004030-88

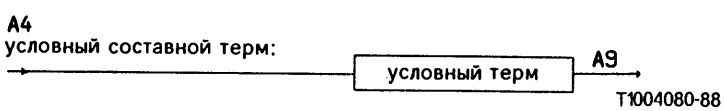


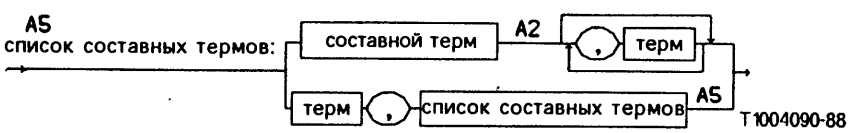
T1004040-88

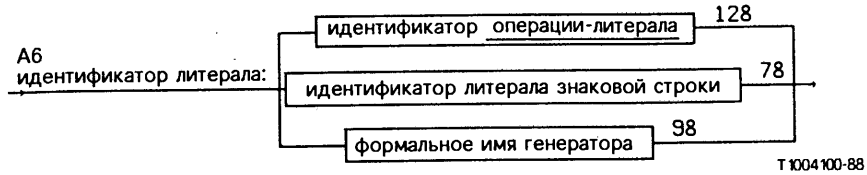

A1
пассивный терм:
идентификатор операции
128
(
пассивный терм
A1
)
,
идентификатор литерала
A6
(
пассивный терм
A1
)
расширенный пассивный терм
A3
T1004050-88

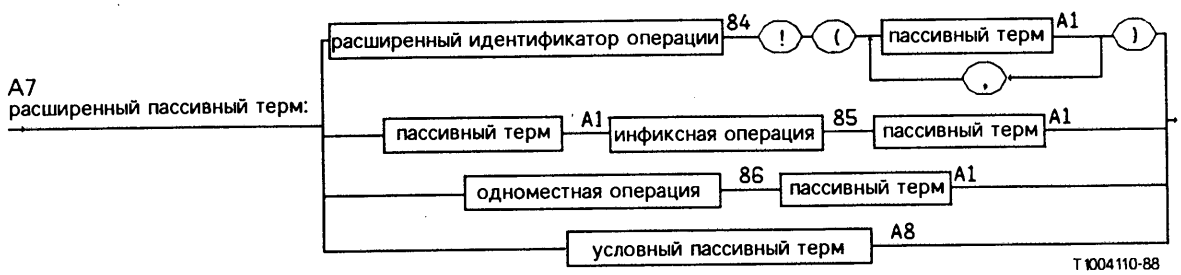

A2
составной терм:
идентификатор значения
128
идентификатор операции
128
(
список составных термов
A5
)
(
составной терм
A2
)
расширенный составной терм
A3
T1004060-88

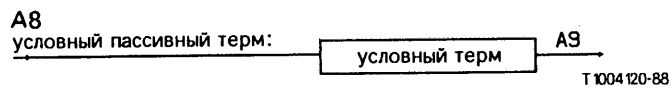

A3
расширенный составной терм:
расширенный идентификатор операции
84
!
(
список составных термов
A5
)
составной терм
A2
инфиксная операция
85
терм
82
терм
82
инфиксная операция
85
составной терм
A2
одноместная операция
86
составной терм
A2
условный составной терм
A4
T1004070-88

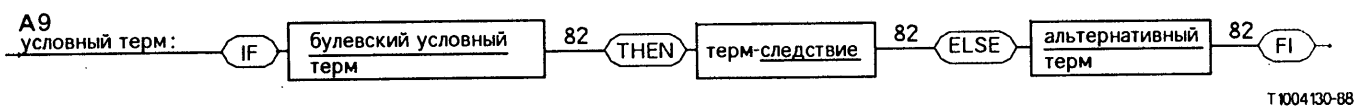

A4
условный составной терм:
условный терм
A9
T1004080-88


A5
список составных термов:
составной терм
A2
,
терм
терм
,
список составных термов
A5
T1004090-88

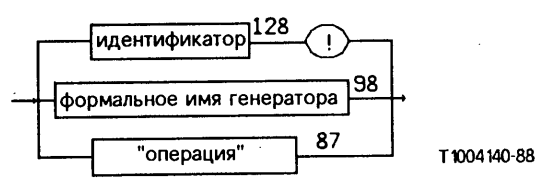


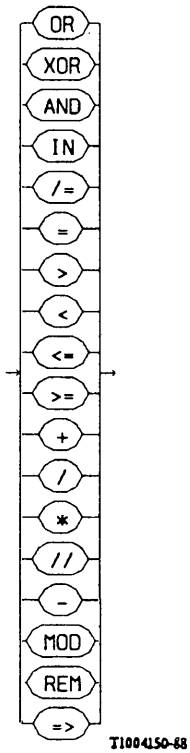




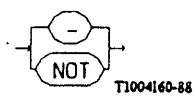


84 *расширенный идентификатор операции*

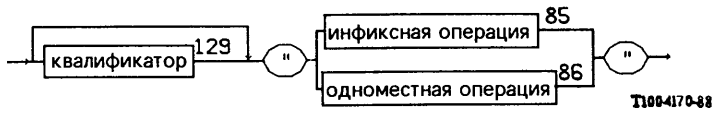




86 *одноместная операция*

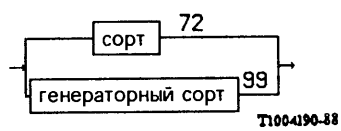


87 *"операция"*

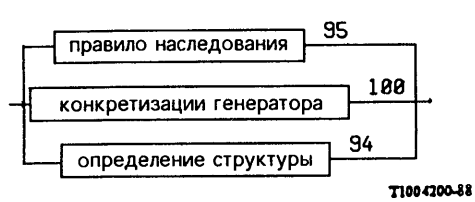


88 *расширенное имя операции* (Данные 5.4.1)

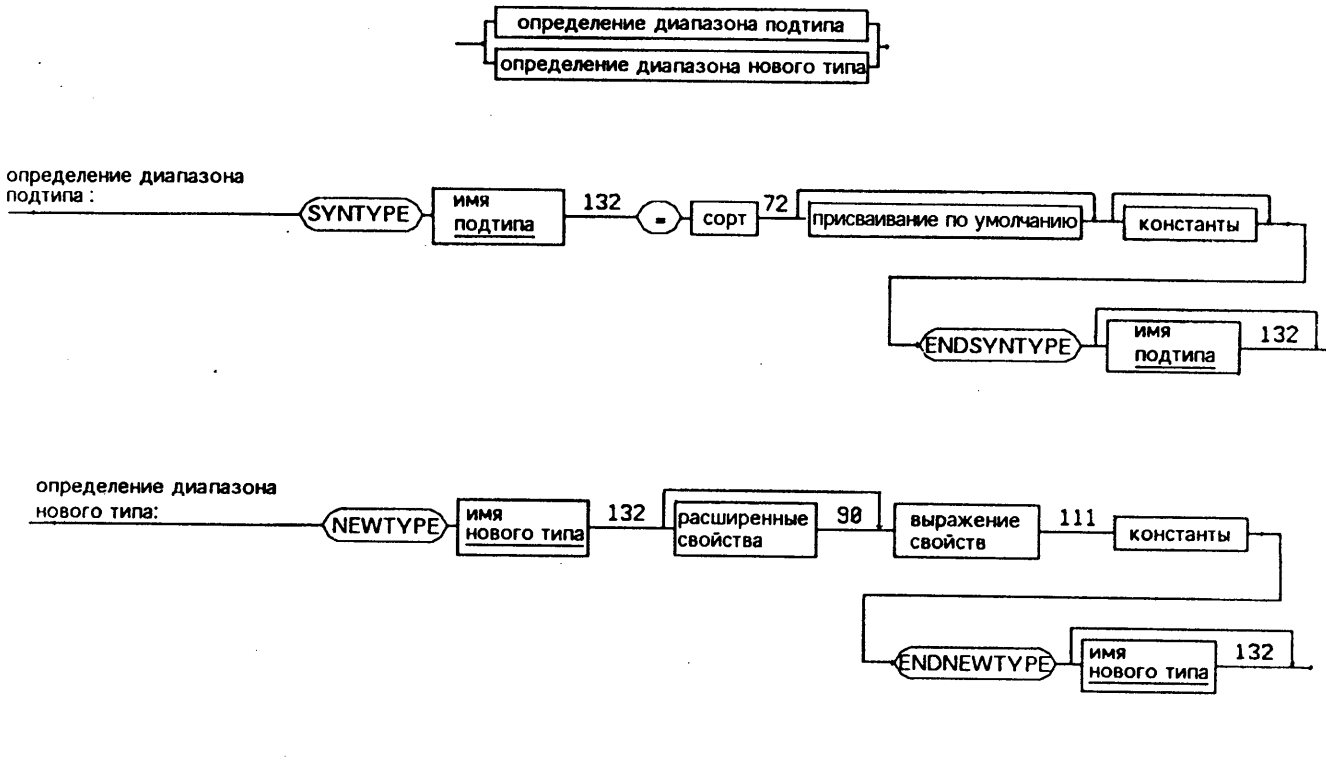
89 *расширенный сорт* (Данные 5.4.1.9)



90 *расширенные свойства* (Данные 5.4.1)

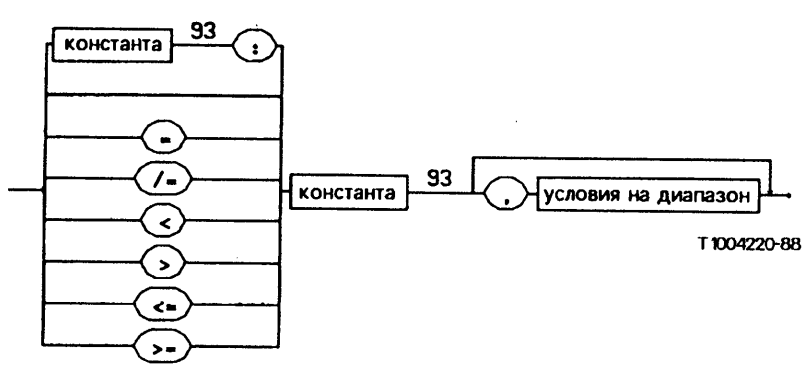


91 *определение подтипа* (Данные 5.4.1.9)



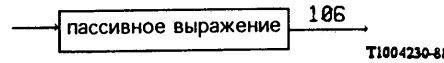
T1004210-88

92 *условия на диапазон* (Данные 5.4.1.9.1)

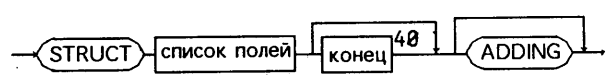


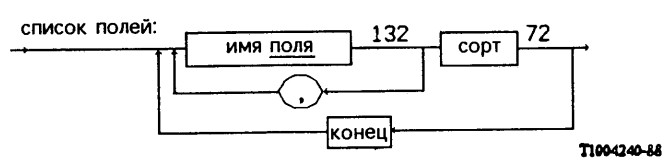
T1004220-88

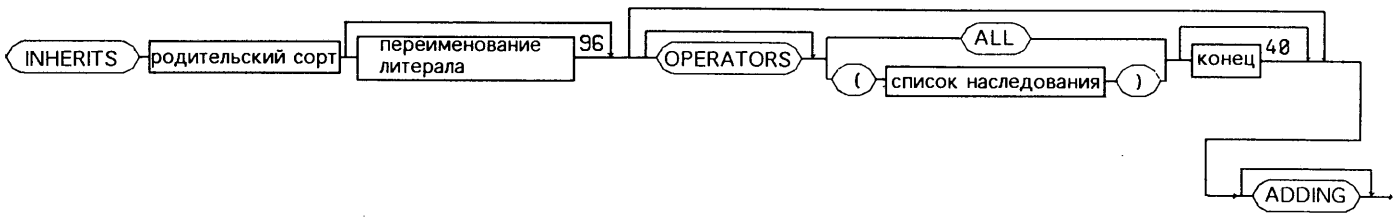
93 *константа* (Данные 5.4.1.9)

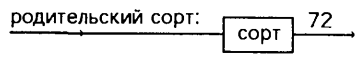


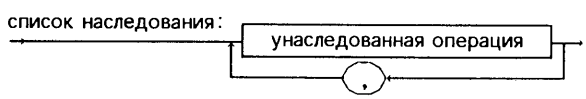
94 *определение структуры* (Данные 5.4.1.10)

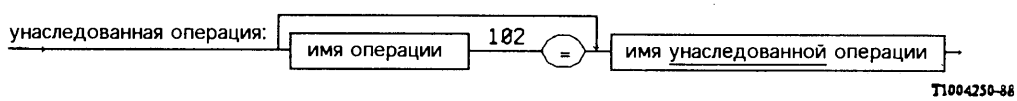




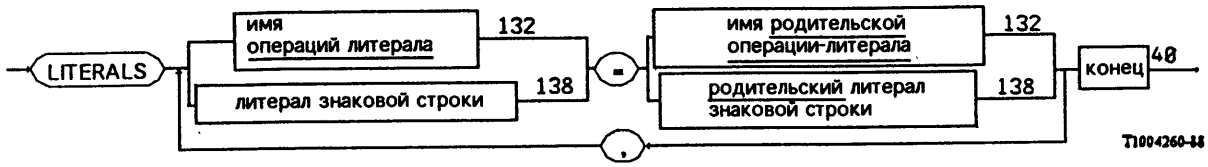

INHERITS
родительский сорт
переименование литерала
96
OPERATORS
ALL
(
список наследования
)
конец
40
ADDING


родительский сорт:
сорт
72

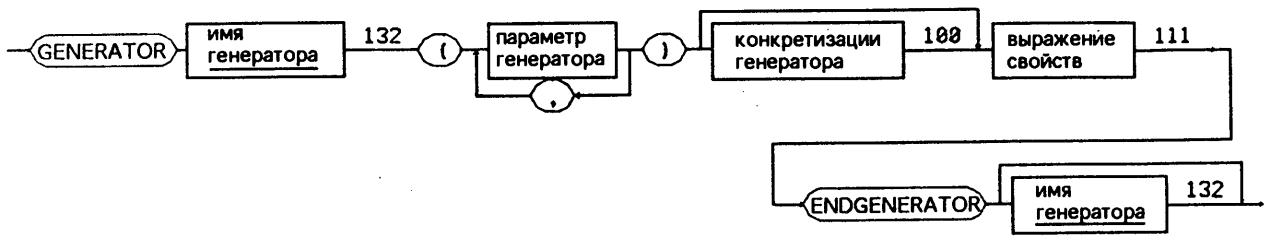

список наследования:
унаследованная операция
,

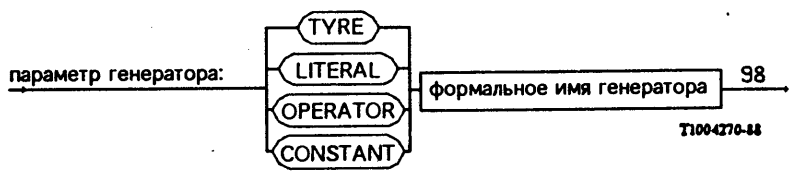

унаследованная операция:
имя операции
102
=
имя унаследованной операции
T1004250-88

96 *переименование литерала* (Данные 5.4.1.11)

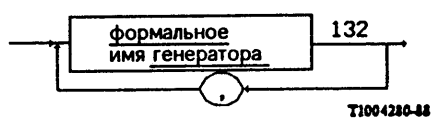


97 *определение генератора* (Данные 5.4.1.12.1)

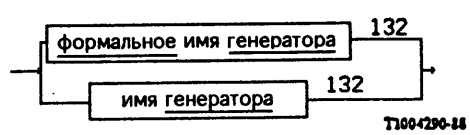




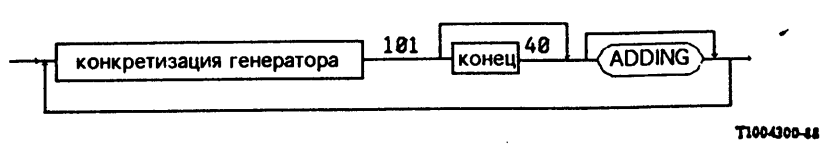
98 *формальное имя генератора* (Данные 5.4.1.12.1)



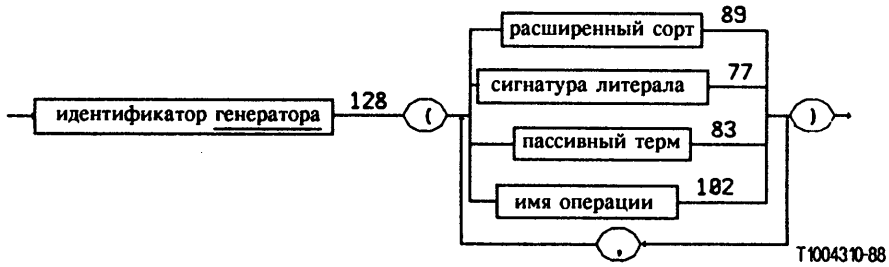
99 *генераторный сорт* (Данные 5.4.1.12.1)



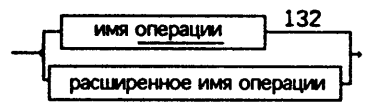
100 *конкретизации генераторов* (Данные 5.4.1.12.2)

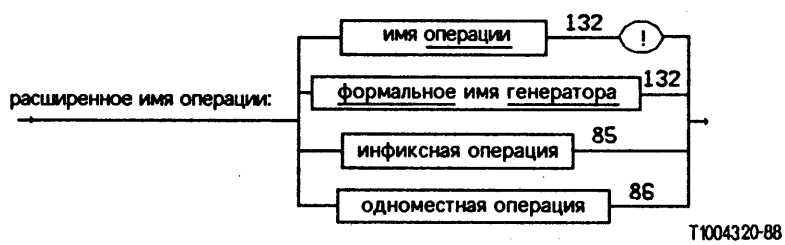


101 *конкретизация генератора* (Данные 5.4.1.12.2)

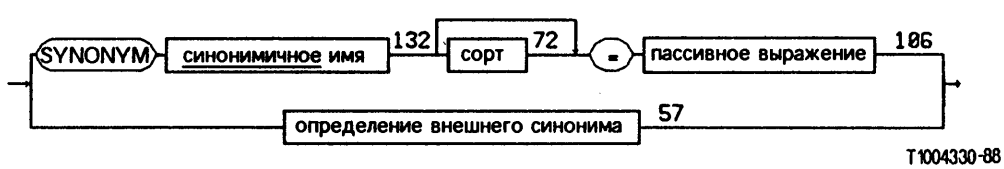


102 *имя операции*



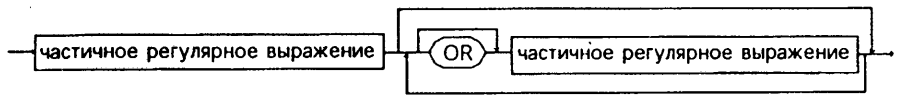


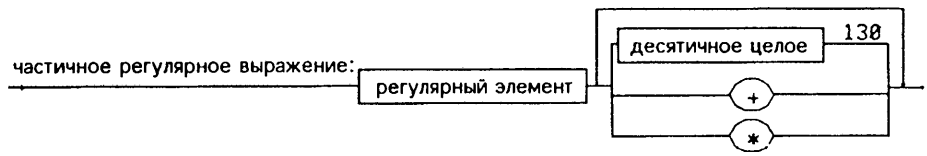
103 *определение синонима* (Данные 5.4.1.13)

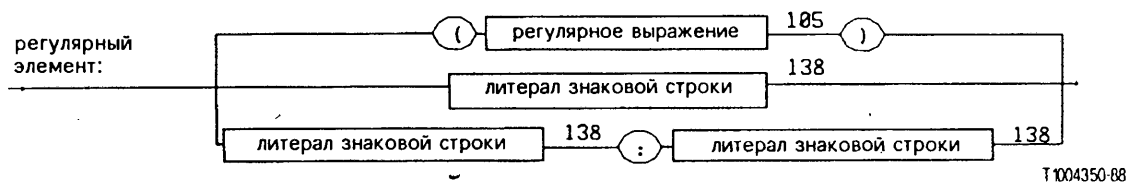


104 *литерал класса имен* (Данные 5.4.1.14)

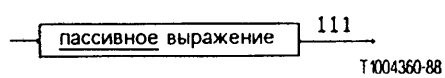
105 *регулярное выражение* (Данные 5.4.1.14)



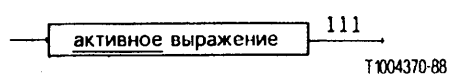




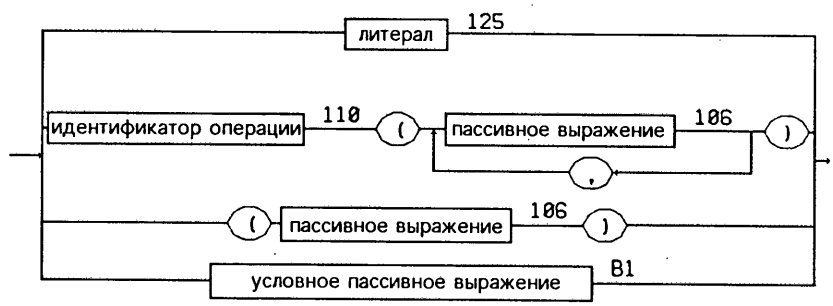
106 *пассивное выражение* (Данные 5.4.2.1)



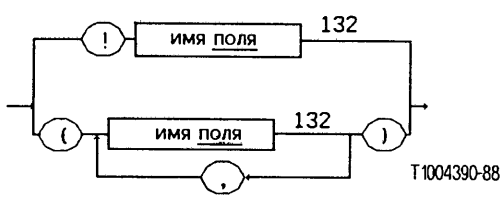
107 *активное выражение* (Данные 5.4.2.2)



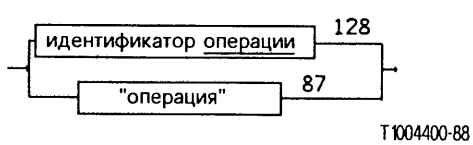
108 *пассивное первичное выражение* (Данные 5.4.2.2)



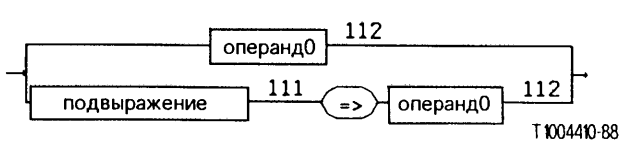
109 *отбор поля*



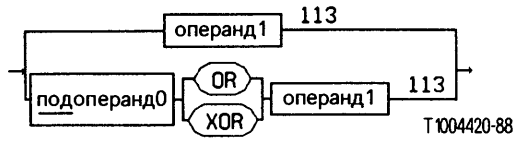
110 *идентификатор операции*



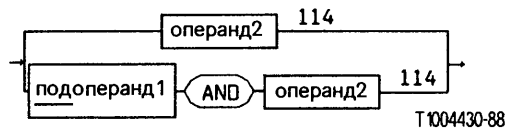
111 *выражение* (Данные 5.5.2.1)



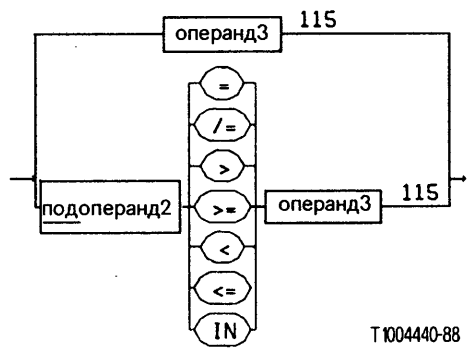
112 *операнд0* (Данные 5.5.2.1)



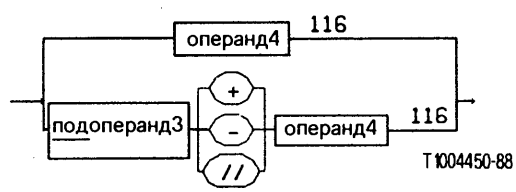
113 *операнд1* (Данные 5.5.2.1)



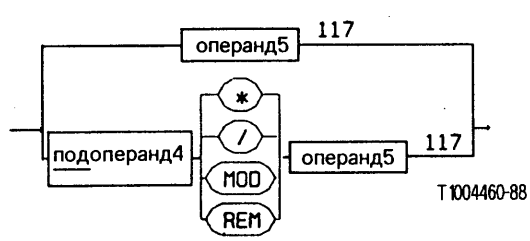
114 *операнд2* (Данные 5.5.2.1)



115 *операнд3* (Данные 5.5.2.1)

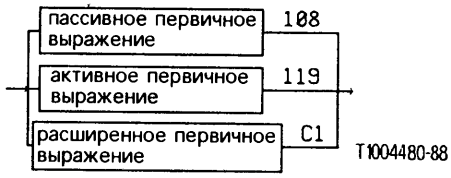


116 *операнд4* (Данные 5.5.2.1)



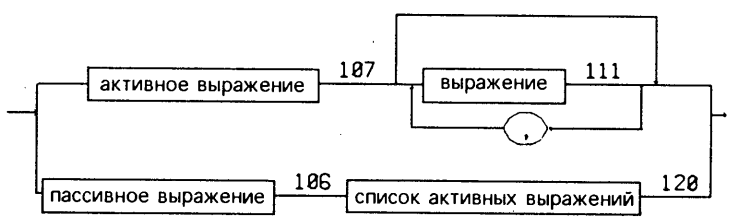
117 *операнд5* (Данные 5.5.2.1)

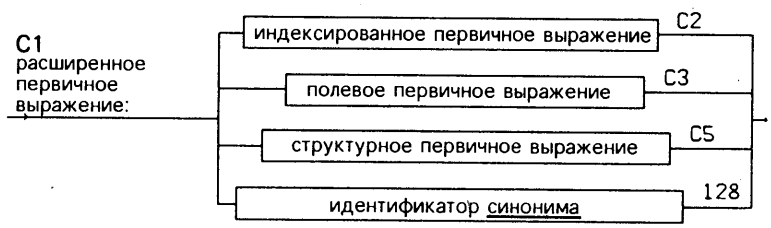
118 *первичное выражение*

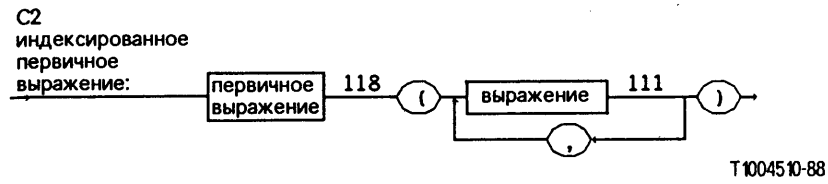


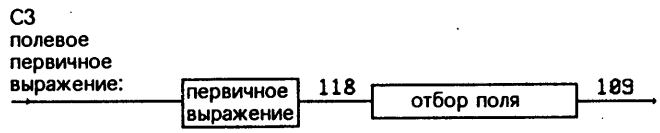
119 *активное первичное выражение*

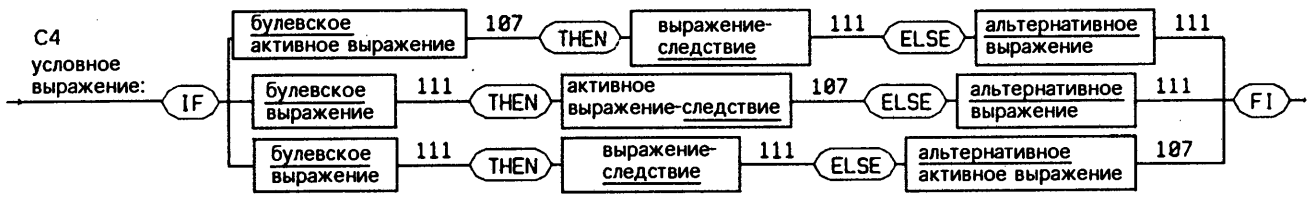
120 *список активных выражений* (Данные 5.5.2.1)

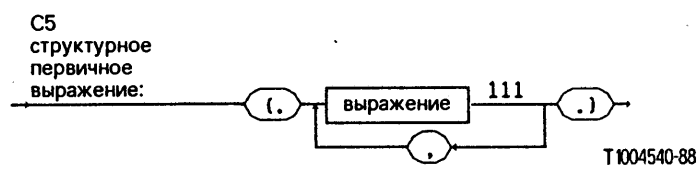




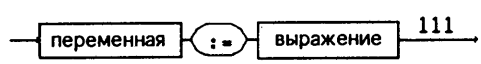


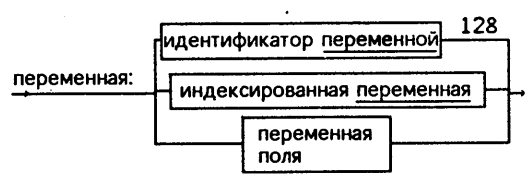


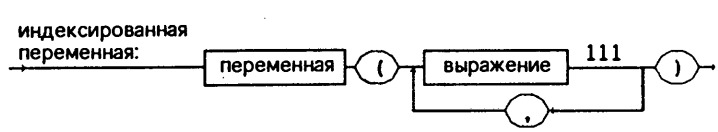


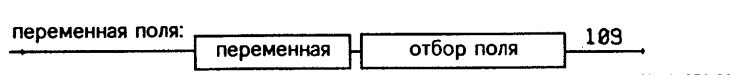


121 *предложение присваивания*

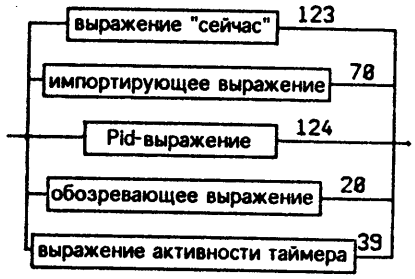




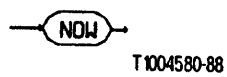




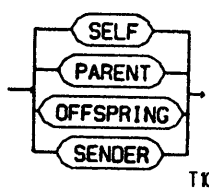
122 *императивная операция* (Данные 5.5.4)



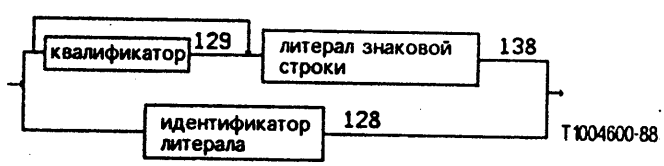
123 *выражение "сейчас"*



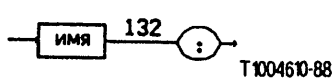
124 *PId-выражение*



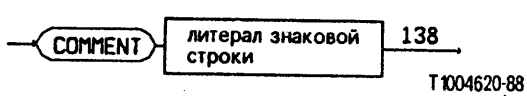
125 *литерал*



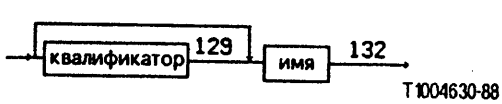
126 *метка*



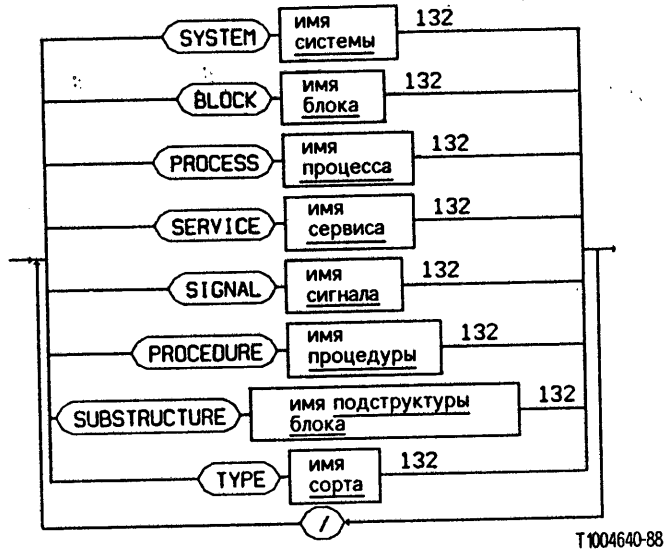
127 *комментарий*



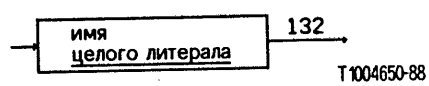
128 *идентификатор*



129 *квалификатор*



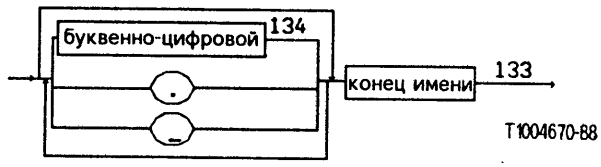
130 *десятичное целое*



Лексические правила диаграмм синтаксиса

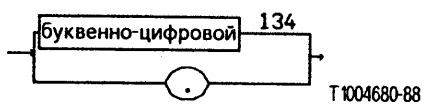
131 *лексема*

132 *имя*

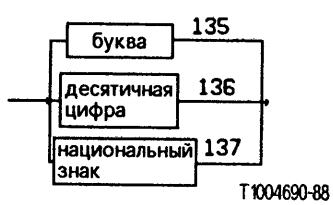


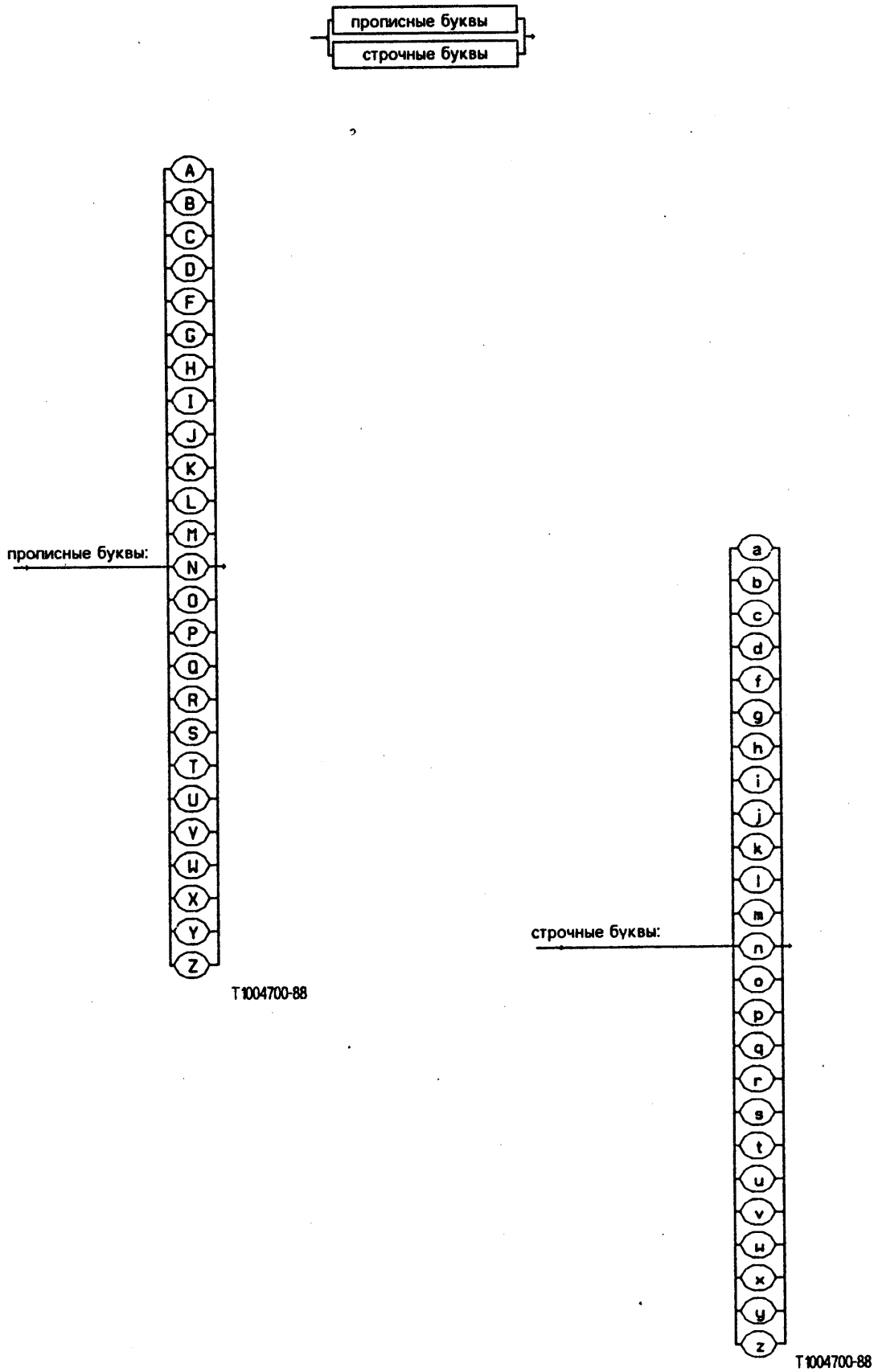
Имя должно содержать хотя бы один буквенно-цифровой знак.

133 *конец имени*

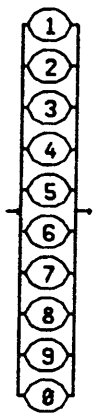


134 *буквенно-цифровой*



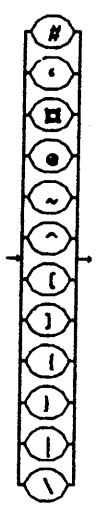

прописные буквы
строчные буквы
прописные буквы:
A
B
C
D
F
G
H
I
J
K
L
M
N
O
P
Q
R
S
T
U
V
W
X
Y
Z
T1004700-88
строчные буквы:
a
b
c
d
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x
y
z
T1004700-88

136 *десятичная цифра*

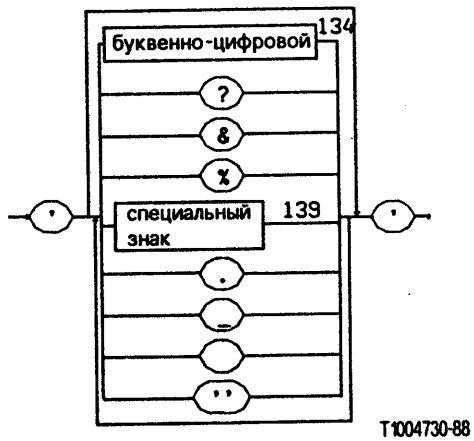


T1004710-88

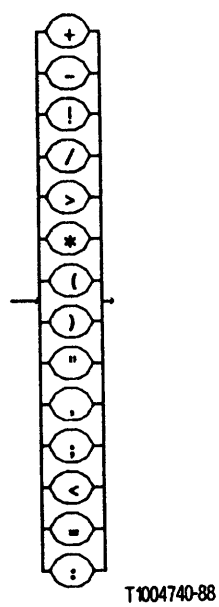
137 *национальный знак*

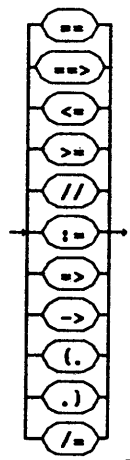


T1004720-88



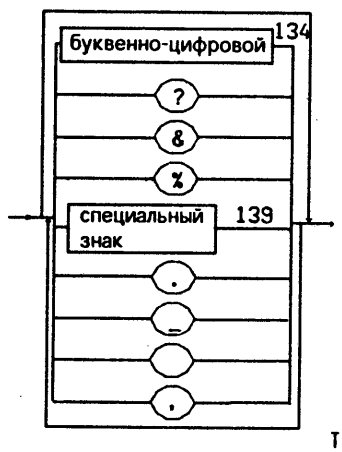
139 *специальный знак*





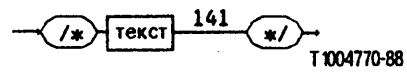
T1004750-88

141 *текст*



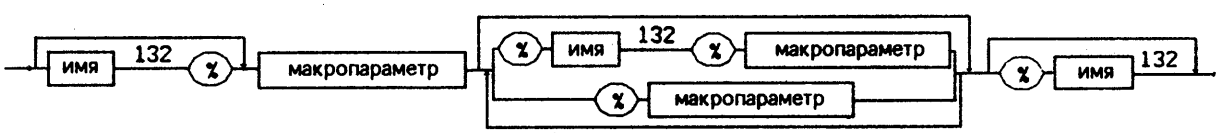
T1004760-88

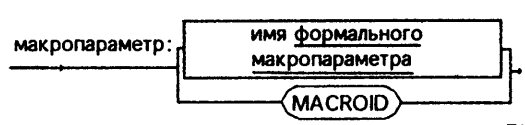
142 *замечание*



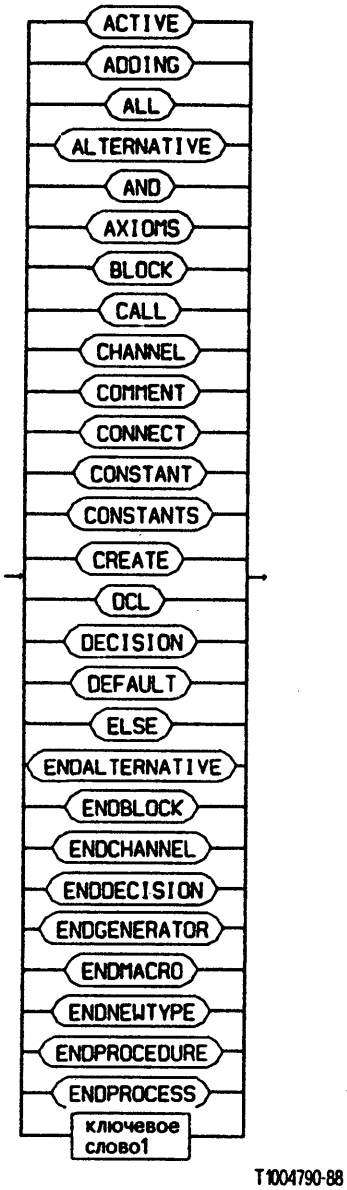
T1004770-88

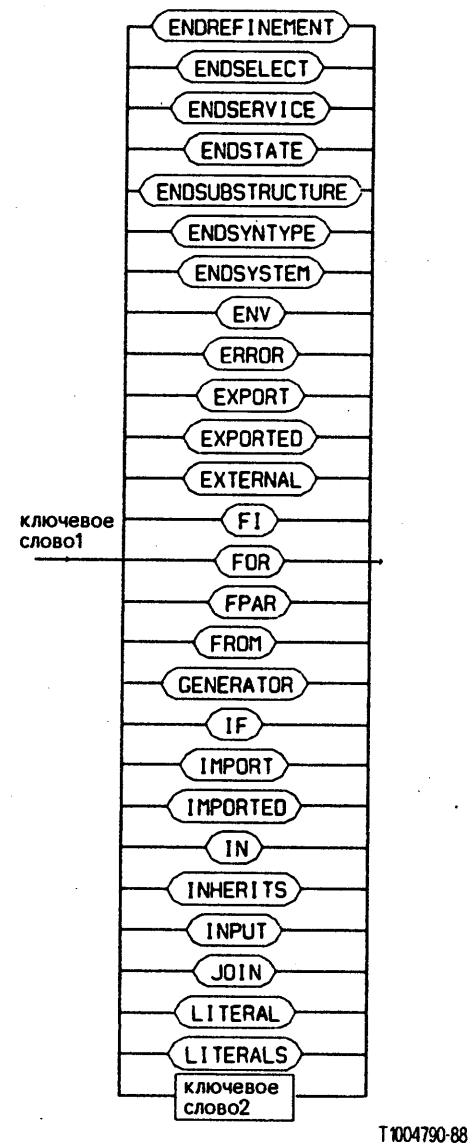
143 *формальное имя*

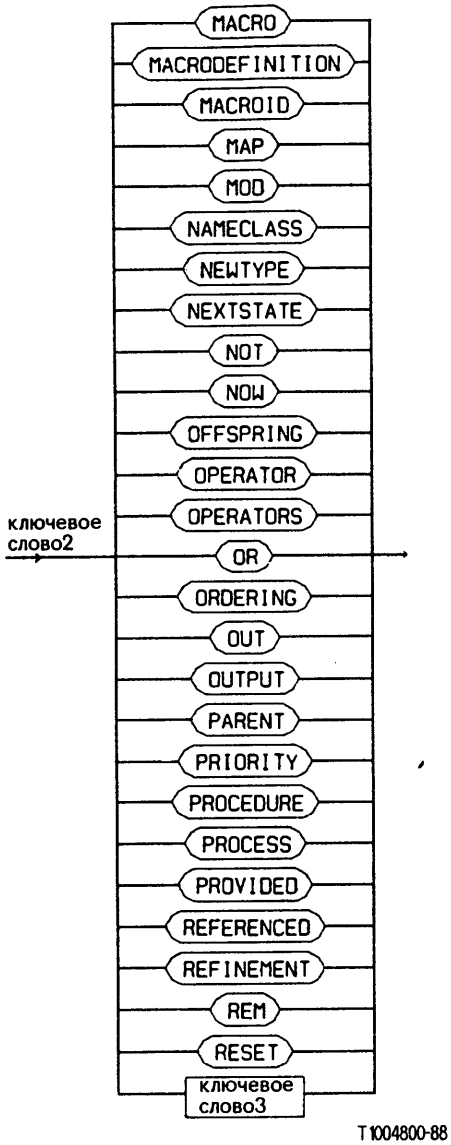


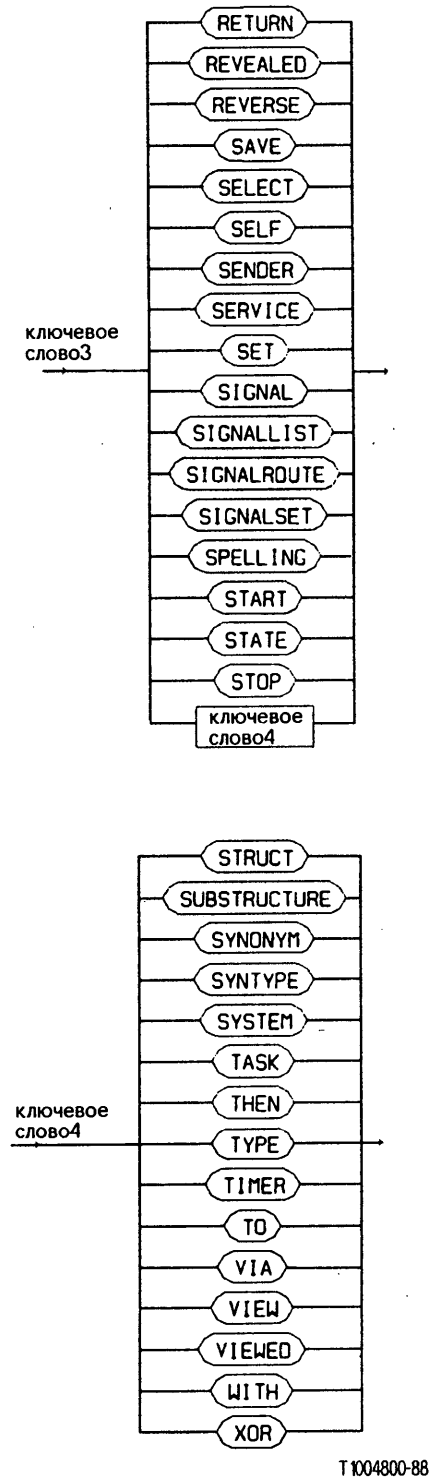


T1004780-88


ACTIVE
ADDING
ALL
ALTERNATIVE
AND
AXIOMS
BLOCK
CALL
CHANNEL
COMMENT
CONNECT
CONSTANT
CONSTANTS
CREATE
DCL
DECISION
DEFAULT
ELSE
ENDALTERNATIVE
ENDBLOCK
ENDCHANNEL
ENDDECISION
ENDGENERATOR
ENDMACRO
ENDNEWTYPE
ENDPROCEDURE
ENDPROCESS
ключевое слово1
T1004790-88


ключевое слово1
ENDREFINEMENT
ENDSELECT
ENDSERVICE
ENDSTATE
ENDSUBSTRUCTURE
ENDSYNTYPE
ENDSYSTEM
ENV
ERROR
EXPORT
EXPORTED
EXTERNAL
FI
FOR
FPAR
FROM
GENERATOR
IF
IMPORT
IMPORTED
IN
INHERITS
INPUT
JOIN
LITERAL
LITERALS
ключевое слово2
T1004790-88


MACRO
MACRODEFINITION
MACROID
MAP
MOD
NAMECLASS
NEWTYPE
NEXTSTATE
NOT
NOW
OFFSPRING
OPERATOR
OPERATORS
OR
ORDERING
OUT
OUTPUT
PARENT
PRIORITY
PROCEDURE
PROCESS
PROVIDED
REFERENCED
REFINEMENT
REM
RESET
ключевое слово3
ключевое слово2
T1004800-88


RETURN
REVEALED
REVERSE
SAVE
SELECT
SELF
SENDER
SERVICE
SET
SIGNAL
SIGNALLIST
SIGNALROUTE
SIGNALSET
SPELLING
START
STATE
STOP
ключевое слово4
ключевое слово3
STRUCT
SUBSTRUCTURE
SYNONYM
SYNTYPE
SYSTEM
TASK
THEN
TYPE
TIMER
TO
VIA
VIEW
VIEWED
WITH
XOR
ключевое слово4
T1004800-88

ПРИЛОЖЕНИЕ E

(к Рекомендации Z.100)

Представление, ориентированное на состояния и изобразительные элементы

E.1 *Введение*

SDL опирается на "обобщенную" модель Конечного автомата (КА), то есть КА обобщен такими объектами, как переменные, ресурсы и т.д. Автомат находится в некотором состоянии. При получении сигнала автомат совершает некоторый переход, при котором выполняются относящиеся к данному состоянию действия (например, распределение и/или перераспределение ресурсов, управление ресурсами, посылка сигналов, принятие решений и т.д.). Поэтому динамическое поведение обобщенного КА может быть описано для каждого перехода в процедурном виде в форме описания последовательности действий, выполняемых над объектами.

В результате перехода из состояния автомат входит в новое состояние. Состояние обобщенного автомата может быть охарактеризовано с помощью объектов, связанных с состоянием, дополнительной информацией об объектах (например, значений переменных, состояния ресурсов, взаимосвязи между ресурсами) и сигналов, которые могут быть получены в данном состоянии. Например, "состояние-ожидания-первой-цифры" при телефонном вызове может быть охарактеризовано следующим образом:

Вызывающий: трубка снята

Генератор тона номеронабирателя: посылка тона номеронабирателя

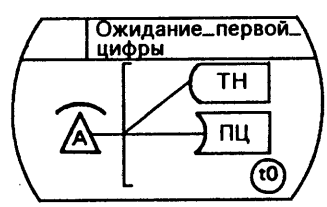
Приемник цифр: готов к приему

Таймер: управление временем непрерывного сигнала

Маршрут: вызывающий соединен с генератором тона номеронабирателя, приемником цифр и т.д.

Очевидно, каждое состояние может быть определено статически с помощью объектов и дополнительной информации (пояснительного текста), связанной с данным состоянием.

Для определения объектов, связанных с каждым состоянием, SDL/GP расширен за счет добавления изобразительных элементов. Определение состояний в терминах изобразительных элементов называют картинами состояний. Символ состояния, используемый в SDL/GP, может содержать картину состояния. Это является необязательной частью SDL/GP. На рис. E-1 приведен пример определения "состояния-ожидания-первой-цифры".



T1001720-87

РИСУНОК E-1

Пример определения состояния в терминах изобразительных элементов

Во многих случаях действия, которые должны быть выполнены над объектами при переходе, могут быть выведены с учетом различия в определениях состояний до и после перехода. Например, если некоторый ресурс появляется только после перехода, это означает, что во время перехода требуется выполнить действие по выделению этого ресурса. Поэтому, если приведено детальное описание состояния, то обычно полный состав действий, выполняемых обобщенным КА во время перехода, может быть выведен из описаний состояния, предшествующего переходу и следующего за ним. Однако последовательность выполнения действий может не быть выводимой из различия в описаниях состояний. Поэтому в тех случаях, когда в диаграммах SDL последовательность выполнения действий менее важна, явного описания тех действий, которые могут быть выведены из различия в определениях состояний не требуется. В противном случае желательно приводить явное описание последовательности действий.

Диаграмма SDL, в которой переход описан исключительно с помощью явных символов действий, называется версией SDL/GP, ориентированной на переход.

Диаграмма SDL/GP, в которой состояния описаны с использованием изобразительных элементов, а действия при переходе минимизированы, называют версией SDL/GP, ориентированной на состояния, или SDL, ориентированной на состояния с изобразительными элементами (SDL/PE). Картины состояний могут быть с успехом применены при определении некоторых конкретных систем, в результате чего получаются более компактные, декларативные диаграммы процессов с меньшим словесным описанием.

Возможна и комбинированная версия. Таким образом имеются три следующие версии SDL/GP:

a) Версия, ориентированная на переход
   - Цепочка перехода описана только явными символами действий.
   - Как указано ранее, это является процедурным описанием обобщенного КА.
   - Данная версия удобна в том случае, если последовательность действий более существенна, чем детальное описание состояний.

b) Версия, ориентированная на состояния
   - Состояние описано только изобразительными элементами.
     Это изображение называется картиной состояний.
   - Последовательность действий при переходе неявно заключена в разнице между состояниями, предшествующими переходу и следующим за ним.
   - Как указано ранее, это является декларативной спецификацией обобщенного КА.
   - Данная версия удобна в том случае, если последовательность действий в каждом переходе представляет незначительный интерес и желательно иметь наглядное разъяснение или компактное изображение.

c) Комбинированная версия
   - Комбинированная версия удобна в том случае, если исследуются как последовательность действий в каждом переходе, так и детальное описание состояний.

Примеры трех версий приведены на рис. Е-2, Е-3 и Е-4.

РИСУНОК Е-2

Версия, ориентированная на переход

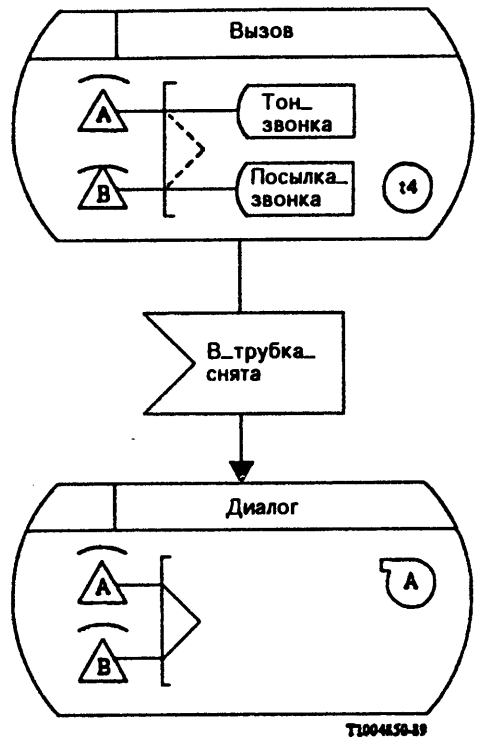


РИСУНОК E-3

Версия, ориентированная на состояния

РИСУНОК E-4

Комбинированная версия

E.2 *Изобразительные элементы в SDL/GP*

Синтаксис и семантика SDL, определенные в Рекомендации Z.100, применимы к изобразительным элементам. Однако указанные синтаксис и семантика расширены следующим образом:

Изобразительные элементы отражают различные объекты. Набор изобразительных элементов в принципе неограничен, так как для каждого нового применения SDL можно вводить новые изобразительные элементы. Однако для применения в системе коммутации электросвязи и сигнализации следующий набор изобразительных элементов считается достаточным:

- граница функционального блока (левая или правая);
- оконечное оборудование (различное);
- приемник сигналов;
- передатчик сигналов;
- комбинированный приемо-передатчик сигналов;
- управляющий таймер;
- коммутируемый путь (соединенный, резервированный);
- модули коммутации;
- идет начисление платы;
- управляющие элементы;
- символ неопределенности.

Стандартные символы, рекомендованные для указанных изобразительных элементов, приведены в разделе E.2.2.

E.2.1 *Правила интерпретации*

1) Символ состояния может содержать картину состояния. Картина состояния определяет состояние с помощью изобразительных элементов и пояснительного текста.

2) Каждый изобразительный элемент в картине состояния отражает какой-то объект, связанный с этим состоянием, например:
   - ресурсы;
   - переменные;
   - внешние и внутренние границы;
   - взаимосвязи между объектами;
   - сигналы, которые могут быть получены в данном состоянии;
   - и т.д.

3) С каждым изобразительным элементом может быть связан пояснительный текст. Этот текст может использоваться, чтобы разъяснить:
   - детальное имя ресурса;
   - состояние ресурса;
   - значение переменной;
   - сигналы, касающиеся объекта;
   - и т.д.

4) Граница функционального блока:
   a) Граница функционального блока используется для определения, является ли изобразительный элемент "внутренним" или "внешним" по отношению к процессу. Внутренние изобразительные элементы отображают объекты, владельцем которых является данный процесс. Внешние изобразительные элементы отображают объекты, владельцем которых является какой-то другой из рассматриваемых процессов.
   b) Правило a) применяется также для различения внутреннего и внешнего пояснительных текстов. С этой целью в правиле a) следует заменить слова "изобразительный элемент" на слова "пояснительный текст".

5) Правило интерпретация перехода:
   Когда процесс переходит из одного состояния в другое, выполняемая при этом полная совокупность действий является комбинацией:
   - действий, направленных на изменение всех связанных с переходом объектов; при этом имеются в виду действия, которые могут быть выведены из различий в описании состояний;
   - действий, явно описанных в переходе, например выводов или работ.

Таким образом,

a) Если некоторый изобразительный элемент изображает ресурс, то его отсутствие в некотором состоянии и наличие в последующем состоянии означает, что этот ресурс был выделен во всех переходах, соединяющих два состояния. То же самое можно изобразить аналогичным образом с помощью работы (работ), показывающей (их) выделение ресурса при переходе (дах).

b) Если в правиле a) слова "наличие" и "отсутствие" взаимно заменены друг другом, то слово "выделен" следует заменить словом "устранен".

c) Если в правиле a) слова "изобразительный элемент" заменены словами "внешний изобразительный элемент", то слово "работа" должно быть заменено словами "выходной сигнал" (запрашивающим тот процесс, который владеет ресурсом, выделить этот ресурс); слово "работа" может быть заменено просто на "входной сигнал" от процесса, сообщающий, что процесс выделил ресурс.

d) Если в правиле a) слова "наличие" и "отсутствие" взаимно заменены и, кроме того, слова "изобразительный элемент" заменены словами "внешний изобразительный элемент", то имеет место правило c), в котором слово "выделить" должно быть заменено словом "устранен".

e) Правила a), b), c) и d) сохраняют силу для случая наличия или отсутствия в картинах состояний пояснительного текста; для этого надо в этих правилах заменить слова "изобразительный элемент" словами "пояснительный текст".

6) Для каждой данной диаграммы процесса конкретные изобразительные элементы (или конкретная комбинация изобразительных элементов) должны помещаться всегда в одно и то же место картины состояния, если они имеются в данном состоянии. Благодаря этому наличие или отсутствие конкретного изобразительного элемента (или комбинации) внутри символа состояния может быть легко распознаваемо сравнением данной картины состояния с картинами других состояний процесса.

7) Если в картине состояния имеется передатчик сигнала, то его пояснительный текст идентифицирует сигнал, посылаемый в течение последующего перехода.

8) Если в картине состояния имеется передатчик непрерывного сигнала (например, тон звонка), то его пояснительный текст идентифицирует сигнал, который начинает посылаться в течение последующего перехода и в данном состоянии.

9) Те действия при переходе, которые не могут быть выведены из разницы описаний предшествующего и последующего состояний, должны быть явно описаны в переходе. Например, если некоторого ресурса в экспортируемой переменной нет в предшествующем и последующем состояниях, то необходимые действия лучше описать в переходе.

E.2.2 *Символы, рекомендуемые в качестве изобразительных элементов*

При использовании изобразительных элементов каждое состояние изображается символом состояния, содержащем картину состояния; формат символа состояния приведен на рис. E-5:

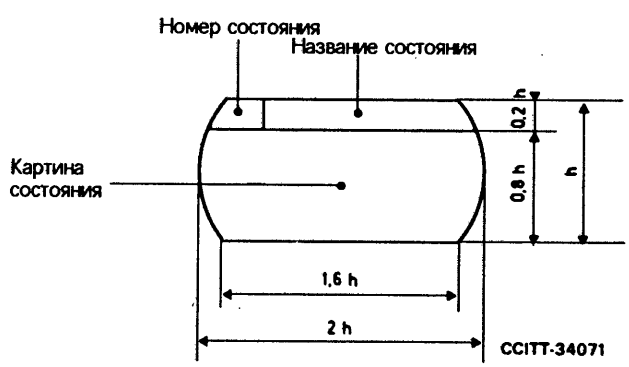


РИСУНОК E-5

Рекомендуемый формат символа состояния, содержащего картину состояния

Для применения SDL/GP к описанию системы, описывающей обработку процесса вызова электросвязи, включая протоколы сигнализаций, сетевые службы и процессы сигнальной взаимосвязи, был рекомендован основной комплект изобразительных элементов. Многие из этих изобразительных элементов могут быть использованы в приложениях SDL/GP к процессам, отличным от обработки процесса вызова.

Рекомендуемые символы для основного комплекта изобразительных элементов изображены на рис. E-6, а рекомендуемые пропорции символов изобразительных элементов приведены на рис. E-7.

Примеры использования основного комплекта изобразительных элементов приведены на рис. E-8.

E.2.3 *Специальные соглашения и интерпретации, используемые в SDL/GP, ориентированном на состояния*

В этом разделе определяются несколько специальных соглашений и интерпретаций, касающихся SDL/GP, ориентированного на состояния. В их число входят:

— Специальное правило интерпретации диаграммы процесса, которое требуется по отношению к так называемому ПРАВИЛУ ИНТЕРПРЕТАЦИИ ПЕРЕХОДА (см. § E.2.1, правило 5).

— Уникальное расположение изобразительного элемента (или изобразительного элемента и пояснительного текста) в картине состояния, которое требуется при использовании изобразительных элементов (см. § E.2.1, правило 6).

— Специальная интерпретация, которая требуется по отношению к переменным, изображенным с помощью внешних изобразительных элементов и внешнего пояснительного текста, являющихся доверяемыми переменными, связанными с другими процессами.

E.3 *Критерии выбора изобразительных элементов*

Выбор символов для конкретных изобразительных элементов был основан на следующих заключениях и общем критерии выбора. Они должны быть учтены при разработке дополнительных символов изобразительных элементов для более широкого применения SDL.

1) Простота репродуцирования

Для того чтобы обеспечить удобное репродуцирование SDL-диаграмм с помощью репродуцирующих методов красящих линий или голубой печати, а также с помощью фотокопирования и фотопечати, символы изобразительных элементов должны состоять из ясных линий без теней и какой-либо окраски.

2) Простота понимания

a) Соответствие. — Очертание каждого символа должно соответствовать тому понятию, которое изображает этот символ.

b) Различие. — При выборе основного комплекта символов необходимо предусмотреть, чтобы один символ можно было отличить от другого символа, входящего в данный комплект.

c) Близость. — Очертания изобразительных элементов, представляющих различные, но связанные друг с другом устройства (например, передатчики и приемники), должны быть близки друг другу каким-то очевидным образом.

d) Связь сокращенного пояснительного текста с символом. — Предполагается, что в некоторых случаях сокращенный текст будет некоторым образом ассоциирован с изобразительным элементом, с тем чтобы указать класс изобразительных элементов; например буквы МЧК, ассоциируемые с символом приемника, будут указывать, что должны поступать многочастотные коды сигналов. В этих случаях изобразительные элементы должны содержать замкнутые области, допускающие использование очень небольшого числа буквенно-цифровых знаков.

e) Ограниченность набора. — Для обеспечения легкости изучения изобразительного метода общее число символов должно быть сведено к минимуму.

РИСУНОК E-6

Рекомендуемые символы для основного комплекта понятий, представляемых изобразительными элементами

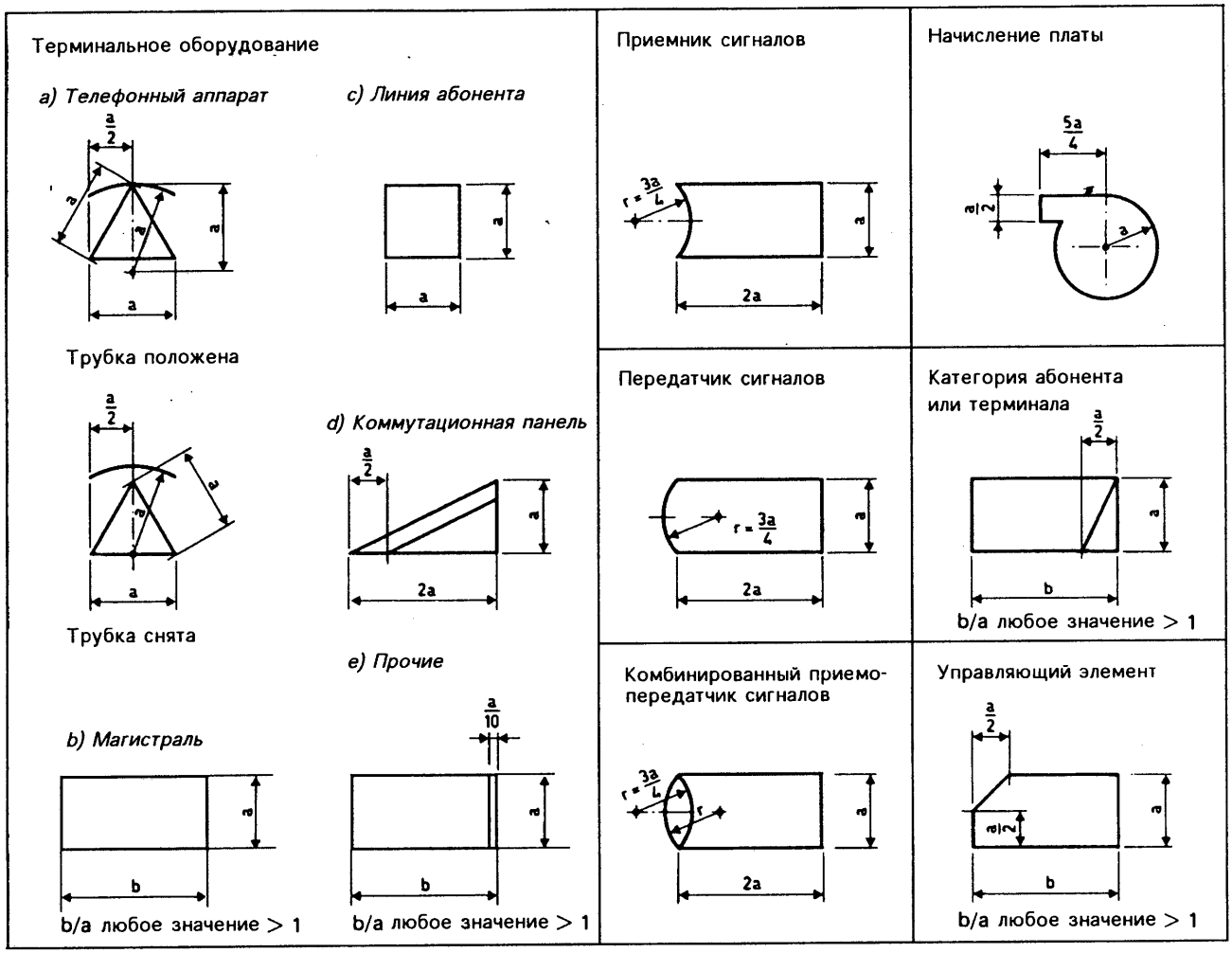
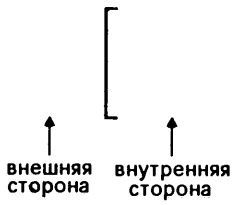
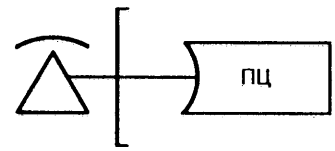
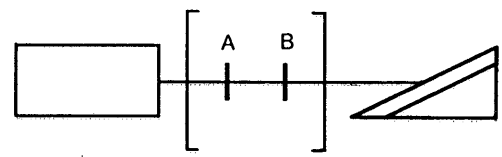
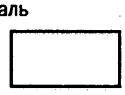
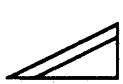
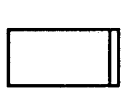
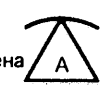
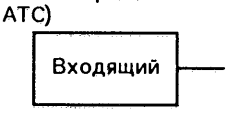
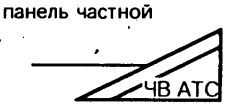
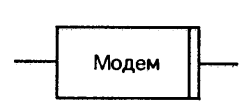


РИСУНОК Е-7

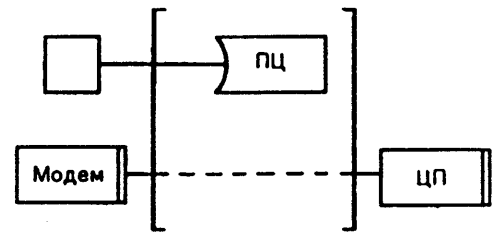
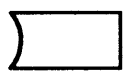
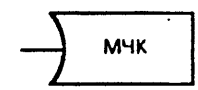
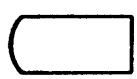
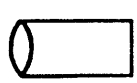
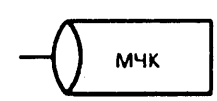
Рекомендуемые пропорции основного комплекта изобразительных элементов

№	Картинный элемент	Комментарий	Примеры
1.	*Граница функционального блока (ФБ)*  	Для разграничения объектов, находящихся внутри и вне границы ФБ. Данный процесс может изменить состояния только тех объектов, которые находятся внутри границы.	1.1 Приемник цифр, находящийся внутри границ ФБ, присоединен к телефонному аппарату, находящемуся вне границ ФБ.  1.2 Магистраль, находящаяся вне границ ФБ, присоединена через двухступенчатый коммутатор к коммутационной панели, находящейся вне границ. 
2.	*Терминальное оборудование* a) Телефонный аппарат трубка положена  трубка снята  b) Магистраль  c) Линия абонента [за исключением a)]  d) Коммутационная панель  e) Прочие 	Может оказаться полезным изобразить терминальное оборудование (например, телефонный аппарат и коммутационную панель), находящееся вне границ ФБ; это может улучшить понимание работы процесса.	2.1 Телефонная трубка A положена  2.2 Телефонная трубка B снята  2.3 Соединитель входящей магистрали (от территориальной АТС)  2.4 Выходная линия абонента к линии другой стороны соединения.  2.5 Коммутационная панель частной внутренней АТС.  2.6 Модем 

CCITT-20880

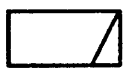
РИСУНОК E-8

Примеры использования основного комплекта изобразительных элементов

№	Картинный элемент	Комментарий	Примеры
3.	*Коммутируемый путь* a) соединен ——— b) резервирован _ _ _ _	Чтобы изобразить соединения между терминальным оборудованием и/или устройствами сигнализации, вовлеченными в процесс.	3.1 Линия абонента, присоединенная к приемнику цифр, а также модем, имеющий резервированную линию, ведущую к центральному процессору (ЦП) 
4.	*Приемник сигналов* 	Чтобы изобразить природу полученных сигналов и особенно тех, которые пересекают границу функционального блока.	4.1 Приемник многочастотного кода сигналов 
5.	*Передатчик сигналов* 	Для спецификации процесса, посылающего сигнал, и для описания посылаемых сигналов и особенно тех, которые должны пересечь границу функционального блока.	5.1 Генератор тона звонка 
6.	*Комбинированный приемо-передатчик сигналов* 	По соглашению этот прибор обьединяет функции приемника и передатчика сигналов.	6.1 Приемо-передатчик МЧК 
7.	*Таймер, управляющий процессом* 	Изображается таймер, работающий в данном состоянии.	7.1 Таймер t_3 работает  7.2 Родовой таймер t_s работает  здесь s = 1,2,. . ., n определяет различные служебные тоны.

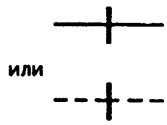
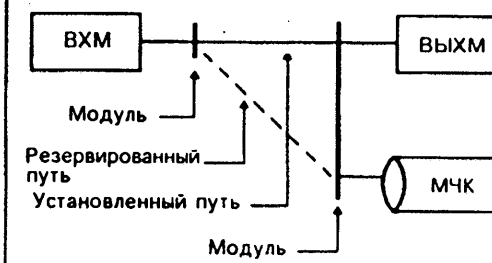
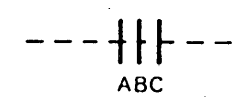
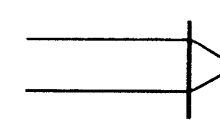
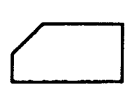
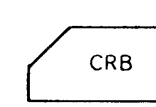
CCITT-20890

РИСУНОК E-8 (*продолжение*)

№	Картинный элемент	Комментарий	Примеры
8.	Идет начисление платы 	Пояснительный текст, находящийся внутри элемента, указывает пользователя, на имя которого в данный момент начисляется плата.	8.1 В настоящий момент происходит начисление платы на имя абонента А. 
9.	*Категория абонента или терминала (и идентифицирующая информация)* 	С помощью этого элемента удобно указывать изменения категории абонента или терминала кажой из сторон в многостороннем соединении.	9.1 Сторона С имеет генераторную категорию № 2. 
10.	*Символ неопределенности* # 	Этот символ заменяет преднамеренно неспецифицированную информацию, которая в других картинах состояний изображена однозначно. В некоторых случаях с помощью этого символа два или более состояний могут быть с успехом слиты в одно, что обеспечивает повышение общей интеллектуальности диаграмм.	10.1 Телефонная трубка либо снята, либо положена.  10.2 В данном состоянии был послан неопределенный МЧК сигнала 

CCITT-20900

РИСУНОК Е-8 (*продолжение*)

№	Картинный элемент	Комментарий	Примеры
11.	*Модуль коммутации*  или	Чтобы показать, какие коммутационные модули вовлечены в процесс. *Примечание.* - Горизонтальная линия является картинным элементом коммутируемого пути, который может быть соединен или резервирован. Вертикальную линию можно использовать для изображения либо всего модуля коммутации (когда внутренняя структура модуля не требуется), либо одну из ступеней коммутации внутри модуля коммутации.	11.1. Путь, соединенный с помощью одного модуля коммутации ЛСС (Линии соединения сети). ЛСС - Линия соединения сети  11.2 Пути, соединенные и резервированные с помощью двухступенчатых модулей коммутации.  ВХМ - Входная магистраль ВЫХМ - Выходная магистраль МЧК - Многочастотный код *Примечание.*- В этом приеме ВХМ присоединена к ВЫХМ, но не присоединена к приемо-передатчику МЧК. 11.3 Путь, соединенный с помощью трехступенчатого модуля коммутации RSN.  11.4 Путь, резервированный с помощью трехступенчатого модуля коммутации ABC.  11.5 Путь, соединенный с помощью перекрестной сети. 
12.	*Управляющий элемент (приданный процессу)* 	Чтобы изобразить, какое управляющее оборудование вовлечено в процесс (особенно те модули, для которых должны быть указаны размеры). Этот символ можно использовать для указания того, что процессу были выделены какие-то конкретные программные элементы.	12.1 Буфер регистрации вызова 

CCITT-20910

РИСУНОК Е-8 (*окончание*)

КРИТЕРИИ ИСПОЛЬЗОВАНИЯ И ПРИМЕНИМОСТИ МЕТОДОВ ФОРМАЛЬНЫХ ОПИСАНИЙ[1]

1 Обеспечение методов формальных описаний (FDT)

Ввиду сложности и широкого распространения Рекомендаций возникает настоятельная необходимость в использовании развитых методов разработки и внедрения этих Рекомендаций.

Методы формальных описаний представляют собой важное направление в развитии таких методов.

В некоторых областях использование FDT все еще является новым делом и требуется поэтапная процедура, обеспечивающая их введение.

2 Методы формальных описаний

2.1 *Определения*

Метод формальных описаний (FDT) является методом спецификации, опирающимся на язык описаний, который использует строгие однозначные правила как в отношении разработки выражений на этом языке (формальный синтаксис), так и в отношении интерпретации смысла этих выражений (формальная семантика). FDT предназначен для разработки, спецификации, внедрения и проверки выполнения Рекомендаций (или их частей).

Описание на естественном языке является примером метода неформальных описаний, использующей один из языков МККТТ, на которых публикуются Рекомендации. Этот метод может быть дополнен математической и прочей принятой нотацией, рисунками и т.п.

2.2 *Назначение метода формальных описаний*

Назначением метода формальных описаний (FDT) является достижение точной и недвусмысленной спецификации. Кроме того, FDT предназначен для достижения таких целей, как обеспечение:

— основы для анализа спецификации на корректность, эффективность и т.д.;

— основы для определения полноты спецификаций;

— основы для проверки спецификации в отношении требований к Рекомендации;

— основы для определения того, насколько реализация соответствует Рекомендациям;

— основы для определения содержательного соответствия между Рекомендациями;

— основы для обеспечения реализации.

При нынешнем состоянии этого искусства для удовлетворения всех этих целей в некоторых областях потребуется не один метод формального описания.

2.3 *Преимущества метода формальных описаний*

Применение метода формальных описаний может обеспечить такие преимущества, как:

— улучшение качества Рекомендаций, что в свою очередь, приведет к снижению производственных расходов как МККТТ, так и пользователей Рекомендаций;
— снижение зависимости от естественных языков по обмене техническими концепциями в многоязычной среде;
— ускорение разработки конкретных реализаций за счет использования средств, опирающихся на свойства метода формальных описаний;
— облегчение реализаций, что приводит к улучшению окончательного продукта.

---

[1] Текст настоящей Рекомендации издан также в качестве Резолюции ИСО (ISO/IEC JTC1/N145).

Содержащиеся в документе JTC 1 утверждения относительно предпочтительности определений при наличии нескольких определений в данной Рекомендации опущены.

2.4 *Трудности использования метода формальных описаний (FDT)*

FDT является перспективным методом, еще не получившим широкого распространения. Кроме того, при разработке метода формальных описаний и формальных описаний (ФО) ощущается нехватка ресурсов, а также знаний и опыта в Исследовательских комиссиях МККТТ, чтобы достичь как технических достоинств формально описанных Рекомендаций, так и взаимопонимания в их оценке.

2.5 *Решения*

Разработка учебных и образовательных материалов обеспечит широкое понимание трудных мест FDT. Очевидно, что для этого потребуется время.

3 Разработка и стандартизация метода формальных описаний

Следует избегать ненужного распространения FDT. Прежде чем внедрять новый метод должны быть удовлетворены следующие критерии:

— подтверждена потребность в FDT;

— представлено доказательство того, что эта потребность нуждается в модели, существенно отличающейся от используемой в имеющихся FDT; и

— представлены доказательства полезности и возможностей нового FDT.

4 Разработка и приемка формальных описаний

4.1 В будущем в формальных описаниях Рекомендаций должны использоваться либо уже стандартизованные FDT, либо FDT, находящиеся в процессе стандартизации.

4.2 Предполагается, что разработка формального описания любой конкретной Рекомендации входит в компетенцию Исследовательской комиссии (при консультациях с ИСО для выработки совместных стандартов). Если для новой Рекомендации должно быть разработано формальное описание, то оно должно развиваться как можно глубже и в те же временны́е сроки, что и вся остальная Рекомендация.

4.3 Для эволюционного внедрения формального описания (ФО) в Рекомендации можно предусмотреть три фазы. В компетенцию Исследовательской комиссии входит определение того, которая из фаз должна быть осуществлена первой в отношении некоторого ФО и возможное эволюционирование ФО к следующей фазе. Для некоторого конкретного формального описания не обязательно прохождение через все нижеописываемые три фазы и, более того, вообще не обязательно какое бы то ни было развитие ФО.

Фаза 1

Эта фаза характеризуется отсутствием широкораспространенного знания FDT и опыта формальных описаний; Исследовательские комиссии могут не обладать достаточными ресурсами для разработки или оценки формальных описаний.

Разработка Рекомендаций должна опираться на согласованный подход на естественном языке, ведущий к Рекомендациям, в которых и описание на естественном языке обеспечивает нужную строгость.

Разработка формальных описаний Рекомендаций Исследовательскими комиссиями поощряется, поскольку такие усилия могут повысить качество Рекомендации за счет обнаружения ее недостатков, могут облегчить к тому же понимание читателями и будут способствовать постепенному введению FDT.

Разработанное Исследовательской комиссией формальное описание, которое может считаться достоверным представлением значительной части Рекомендации или всей Рекомендации, должно публиковаться в качестве приложения к Рекомендации.

Вместе с тем Исследовательские комиссии должны разрабатывать и представлять учебные материалы по FDT для обеспечения их широкого внедрения в МККТТ и связанных организациях.

Фаза 2

Для этой фазы характерным является то обстоятельство, что знание FDT и опыт формальных описаний уже широко распространены; Исследовательские комиссии могут обеспечить достаточные ресурсы для поддержки разработок по формальным описаниям. Тем не менее еще нельзя утверждать, что достаточное число членов МККТТ может знакомиться с формальными описаниями, с тем чтобы они могли освоить предлагаемые формально описанные Рекомендации.

Разработка Рекомендаций должна опираться на согласованный подход на естественных языках, ведущих к Рекомендациям, для которых описание на естественном языке является строгим стандартом. Однако эти разработки должны сопровождаться разработкой формальных описаний этих стандартов с целью улучшения и обеспечения структуры, полноты, согласованности и корректности описаний на естественных языках.

Разработанное Исследовательской комиссией формальное описание, являющееся достоверным представлением значительной части Рекомендации или всей Рекомендации, должно издаваться в качестве приложения к Рекомендации.

Тем временем должна продолжаться учебная работа.

Фаза 3

Для этой фазы характерным является допущение, что значение FDT широко распространено; члены МККТТ могут предоставить достаточные ресурсы как для разработки, так и для изучения формальных описаний и, кроме того, существует уверенность, что применение FDT не ограничит свободу реализаций.

Исследовательские комиссии при разработке своих Рекомендаций должны постоянно пользоваться FDT; формальные описания, наряду с описанием на естественных языках, являются составной частью Рекомендации.

Если между описанием на естественном языке и формальным описанием или между двумя формальными описаниями возникает расхождение, то оно должно разрешаться либо за счет изменения или улучшения описания на естественном языке, либо изменения или улучшения формального описания без оказания какого-либо предпочтения одному из них.

4.4 Вышеописанные процедуры пофазной разработки формальных описаний направлены на поддержку введения прогрессивного ФО в процессы стандартизации, а не на то, чтобы помешать этому прогрессу. Однако, если учесть недостаточность, а то и отсутствие фактического опыта в реализации этих процедур, любая Исследовательская комиссия приглашается к выделению одного или нескольких показательных примеров и к тщательному обследованию каждого из них в рамках указанных процедур. Если при этом возникнут трудности с процедурами, то об этом должна быть поставлена в известность Исследовательская комиссия, ответственная за Методы формальных описаний, и по возможности должны быть предложены модификации процедур, направленные на преодоление этих трудностей.

Printed in USSR • 1991 — ISBN 92-61-03754-2