МЕЖДУНАРОДНЫЙ СОЮЗ ЭЛЕКТРОСВЯЗИ

МККТТ

МЕЖДУНАРОДНЫЙ
КОНСУЛЬТАТИВНЫЙ КОМИТЕТ
ПО ТЕЛЕГРАФИИ И ТЕЛЕФОНИИ

СИНЯЯ КНИГА

ТОМ X — ВЫПУСК X.3

ПРИЛОЖЕНИЕ F.1 К РЕКОМЕНДАЦИИ Z.100: ФОРМАЛЬНОЕ ОПРЕДЕЛЕНИЕ ЯЗЫКА SDL

ВВЕДЕНИЕ

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.3

ПРИЛОЖЕНИЕ F.1 К РЕКОМЕНДАЦИИ Z.100: ФОРМАЛЬНОЕ ОПРЕДЕЛЕНИЕ ЯЗЫКА SDL

ВВЕДЕНИЕ

IX ПЛЕНАРНАЯ АССАМБЛЕЯ
МЕЛЬБУРН, 14–25 НОЯБРЯ 1988 ГОДА

ISBN 92-61-03774-7

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 систем 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 служб. 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.3 СИНЕЙ КНИГИ

Приложение F.1 к Рекомендации Z.100

ПРЕДВАРИТЕЛЬНЫЕ ЗАМЕЧАНИЯ

1 Вопросы, порученные каждой Исследовательской комиссии на исследовательский период 1989—1992 годов, содержатся во Вкладе № 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.

Содержание

ВЫПУСК X.3

Приложение F.1 к Рекомендации Z.100

ФОРМАЛЬНОЕ ОПРЕДЕЛЕНИЕ ЯЗЫКА SDL

1 Предисловие

В настоящем Формальном определении SDL дается определение языка, дополняющее определение, приведенное в тексте рекомендации. Это приложение предназначено для тех, кому необходимо очень точное и детальное определение SDL, например лицам, обеспечивающим поддержку языка SDL, и разработчикам средств SDL.

Формальное определение состоит из трех томов:

Приложение F.1 (настоящий том)

В данном Приложении излагаются мотивировка и общая структура, содержится руководство по использованию Формального Определения, а также описывается используемая нотация.

Приложение F.2 Содержит определение статических свойств SDL.

Приложение F.3 Содержит определение динамических свойств SDL.

2 Мотивировка

Естественные языки, как правило, неоднозначны и неполны, то есть некоторым предложениям языка может быть дано несколько интерпретаций, независимо от того, читает текст компьютер или человек.

Определение или спецификация формальны, если их смысловое содержание (семантика) однозначно и полно. Поскольку для этой цели нельзя использовать естественные языки, были разработаны специальные языки, известные как языки спецификации (например, SDL и LOTOS). Язык реализации — CHILL или PASCAL — также может быть использован в качестве языка спецификации (так, например, компилятор формально специфицирует семантику другого языка), но часто бывает важно отделить несущественные для понимания детали реализации от семантики спецификации.

Формальные языки, специально предназначенные для определения языков, известны как метаязыки. Например, форма Бэкуса—Наура (БНФ) — это метаязык, специально предназначенный для формального определения синтаксиса языков программирования.

Несмотря на свою неоднозначность, естественные языки обычно более удобны для чтения, чем формальные языки, и с их помощью легче дать логическое обоснование, указывающее рамки, в которых должна пониматься формальная спецификация. В связи с этим определения часто приводятся одновременно как на естественном, так и на формальном языке спецификации.

В настоящем приложении дается формальное определение SDL. Если те или иные свойства SDL-понятия, содержащиеся в данном документе, противоречат свойствам, определенным в Z.100, причем это понятие содержательно определено в Z.100, то должно быть принято определение из Z.100, а данное формальное определение подлежит исправлению.

2.1 Метаязык

Метаязыком, используемым в данном Формальном Определении, является Мета-IV [1]. Этот язык выбран в силу следующих причин:

- Он опирается на мощные и хорошо исследованные математические основания.
- В нем есть очень удобные и мощные средства манипулирования объектами.
- В нем используется нотация, подобная используемой в программировании, что означает его ориентированность на прикладных и системных программистов.
- В настоящее время он стандартизуется в рамках Европейского сообщества.
- Он хорошо изложен в книгах, научных трудах и журналах и был использован в пособии МККТТ "Формальное Определение CHILL " [2], в котором также содержится сводка нотации Мета-IV.

- Средства Мета-IV доступны и позволяют проводить проверку синтаксиса, анализ видимости, генерацию документов, перекрестные ссылки и т.д.

В разделе 5 дается неформальное введение в ту часть Мета-IV, которая используется в Формальном Определении. Полное определение Мета-IV приведено в [1].

3 Техника моделирования

При рассмотрении того, что понимается под "семантикой SDL", целесообразно (с концептуальной точки зрения) разбить определение языка на несколько частей:

- Определение синтаксических правил.
- Определение статических семантических правил (так называемых условий корректности), устанавливающих, например, какие имена допустимо использовать в данном месте, какие типы значений допустимо присваивать переменным и т.д.
- Определение семантики конструкций языка при их интерпретации (динамическая семантика).

Нет необходимости включать в Формальное Определение синтаксические правила, поскольку имеющиеся в Z.100 БНФ-правила и Синтаксические диаграммы уже служат формальными определениями синтаксических правил, что означает, что исходной информацией для Формального Определения является синтаксически корректная SDL-спецификация. Исходная информация представляется посредством Абстрактного Синтаксиса. Этот абстрактный синтаксис основывается на дереве грамматического разбора конкретного текстового синтаксиса SDL (БНФ-правилах), в котором опущены не относящиеся к делу детали, такие как разделители и лексические правила. Поэтому настоящий Абстрактный Синтаксис отличается от используемого в рекомендациях Абстрактного Синтаксиса из Z.100, являющегося обобщением понятия модели SDL.

Например, правило в Абстрактном Синтаксисе:

1 *Цепперех* :: *Предлдейств* + [*Предлтерм*]

означает, что *Цепочка перехода* состоит из непустого списка *Предложений действия* и необязательного *Предложения терминатора* (курсив встречается также в правиле вывода).

Полное множество правил вывода (так называемые Определения Доменов), определяющее SDL-синтаксис в абстрактной форме, обозначается AS_0. В некотором отношении оно определяет синтаксис языка на более низком уровне, чем содержащиеся в Z.100 синтаксические правила, поскольку в отличие от AS_0 конкретный текстовый синтаксис в Z.100 содержит много семантической информации (он чувствителен к контексту). Следует отметить, что AS_0 является абстракцией конкретного текстового синтаксиса. Конкретный графический синтаксис не был использован скорее из экономии времени и места, чем из-за трудностей в осуществлении.

Например, список сигналов определяется в Z.100 следующим образом:

< список сигналов > :: = < экземпляр сигнала > {,< экземпляр сигнала >}
< экземпляр сигнала > :: = < идентификатор сигнала > | (< идентификатор списка сигналов >) | < идентификатор таймера >,

а в AS_0 соответствующие определения имеют следующий вид:

2 *Списоксиген* :: *Экземплсиген* +
3 *Экземплсиген* = *Ид* | *Идспискасиген.*

Список сигналов состоит из списка *Экземпляров сигналов*. *Экземпляр сигнала* — это либо идентификатор, либо идентификатор списка сигналов. В отличие от чувствительного к контексту БНФ-вывода < экземпляр сигнала > в AS_0 не делается различия между идентификатором сигнала и идентификатором таймера, поскольку синтаксически они оба являются идентификаторами, в то время как списки сигналов выделяются с помощью скобок.

Исходной точкой для ФО является синтаксически корректная SDL-спецификация. Задачи Формального Определения таковы:

- Определение условий корректности для SDL-спецификаций. Эта задача, упоминаемая далее как Статическая Семантика, составляет содержание Приложения F.2.

- Определение динамических свойств SDL-спецификаций. Эта задача, упоминаемая далее как Динамическая Семантика, составляет содержание Приложения F.3.

Эти этапы показаны на рис. 1. Результат, получающийся из Статической Семантики (то есть AS_1), объясняется ниже.

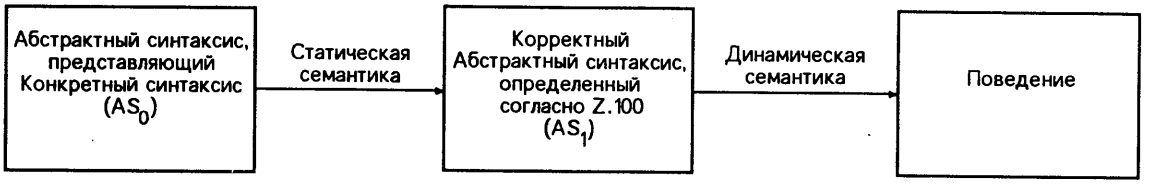

Рисунок 1. Назначение Статической Семантики и Динамической Семантики

Этап перевода конкретного текстового синтаксиса на AS_0 не определяется формально, а выводится из соответствия между именами в двух синтаксисах, как было ранее проиллюстрировано на примере *Списка сигналов.*

3.1 Статическая Семантика

В Z.100 динамическая семантика различных конструкций определяется в терминах Абстрактного Синтаксиса. В общих по содержанию подразделах Z.100 (*Конкретная текстовая грамматика* и *Конкретная графическая грамматика*) определяются конкретные синтаксические правила, приводятся подходящие условия корректности и устанавливается соответствие между конкретными синтаксическими правилами и абстрактным синтаксисом. Это определяется с помощью Мета-IV (в общих подразделах *Абстрактная грамматика*). Тот же абстрактный синтаксис используется в Формальном Определении (где он обозначается AS_1). Сводка этого абстрактного синтаксиса имеется в Приложении B к Z.100.

В дополнение к определению условий корректности Статическая Семантика должна определять, как AS_0-представление спецификации преобразуется в AS_1-представление, то есть, если задано AS_0-представление, Статическая Семантика возвращает AS_1-представление в том случае, когда AS_0-представление корректно. Статическую Семантику можно рассматривать как "абстрактный компилятор", для которого AS_0-представление является исходным языком, а AS_1-представление — объектным языком.

В дополнение к AS_0 и AS_1 Статическая Семантика использует некоторые внутренние вспомогательные домены, называемые Семантическими Доменами. В них хранится информация о заданном элементе, которая может быть затребована в любом месте. Например, когда преобразуется определение процесса, в Семантических Доменах сохраняется информация о его формальных параметрах, и эта информация извлекается во время преобразования Запроса на Создание. Для этой цели можно было бы использовать AS_0-домены, поскольку Семантические Домены так или иначе выводятся из AS_0, однако, когда требуется информация об определенном элементе (скажем, об определении процесса), встречающемся в каком-либо месте на дереве, представление посредством дерева будет неудобным. Поэтому Семантические Домены — это обычно таблицы, моделирующие отображения.

Например, в Семантических Доменах имеется отображение (описываемое далее в разделе 5.4.7) идентификаторов в некоторый дескриптор, содержащий информацию об этих идентификаторах:

4 *Предписдескрипт* = *Квал* $\vec{m}$ *Дескр,*

где *Квал* — это представление идентификатора, используемое в Формальном Определении внутренним образом, а *Дескр* — любой дескриптор. Дескриптором может быть, к примеру, дескриптор процесса:

```
5 Дескр                    = ДПроцесса| . . .
6 ДПроцесса                :: ДПараметра* Совокупдопвход Совокупвыход,
```

что означает, что Дескриптор *Процесса* содержит список Дескрипторов *Параметров*, информацию о *Совокупности допустимых входных* сигналов и информацию о *Выходных* сигналах. Определения этих трех (под)дескрипторов здесь не приводятся.

Само преобразование осуществляется набором функций Мета-IV, использующих три Домена, AS_0, AS_1 и Семантические Домены.

3.2 Динамическая Семантика

Задачей Динамической Семантики является определение поведения SDL-спецификации в форме AS_1.

Динамическая Семантика разбита на три основных раздела:

- Модель основной системы (абстрактная SDL-машина)
- Интерпретация графов процессов.
- Преобразование AS_1 в более удобное представление; то есть строится отображение (Семантический Домен), содержащее информацию, требуемую во время интерпретации, такую как информация о сорте переменной, возможных путях связи между процессами, классах эквивалентности типов и т.д. Отображение называется *Предпис-элемент* (или, более точно, домен отображения называется *Предпис-элемент*).

В Динамической Семантике параллельность в SDL моделируется посредством **Мета-процессов**; то есть параллельное выполнение Мета-процессов в Мета-IV моделирует параллельное выполнение процессов в SDL.

Используется шесть различных типов Мета-процессов:

- *система*

  управление маршрутизацией сигналов и создание *sdl-процессов;*

- *путь*

  обработка недетерминированных задержек в каналах;

- *таймер*

  слежение за текущим временем и обращения с тайм-аутами;

- *обозревание*

  слежение за всеми раскрытыми переменными;

- *sdl-процесс*

  интерпретация поведения SDL-процесса;

- *входной-порт*

  управление образованием очереди сигналов в SDL-процессе. Для каждого экземпляра *sdl-процесса* существует ровно один экземпляр *входного-порта.*

Четыре типа Мета-процессов (*система, путь, таймер* и *обозревание*) могут в целом рассматриваться как моделирующие основную систему.

У Мета-процессов нет совместно используемых данных; они взаимодействуют посредством передачи значений, переносимых экземплярами (объектами) **Коммуникационных Доменов** (отвечающих концепции сигналов в SDL).

Коммуникационные Домены определяются таким же образом, как и другие домены; например объекты Коммуникационного Домена *Входной-Сигнал* направляются в экземпляр *sdl-процесса* из приданного ему экземпляра *входного-порта.* Коммуникационный Домен определяется, например так:

7 *Входной-Сигнал* :: *Идентификатор-Сигнала*, [*Значение*]* *Значение-"Отправитель"*

Экземпляры *Входного-Сигнала* переносят идентификатор отправленного SDL-сигнала, список значений, переносимых SDL-сигналом, и PID-значение отправителя.

На рис. 2 приведена полная "Схема взаимодействия Мета-процессов". Механизм коммуникаций синхронный, а используемая нотация известна как CSP (см. [3] и [4]) (Communicating Sequential Processes).

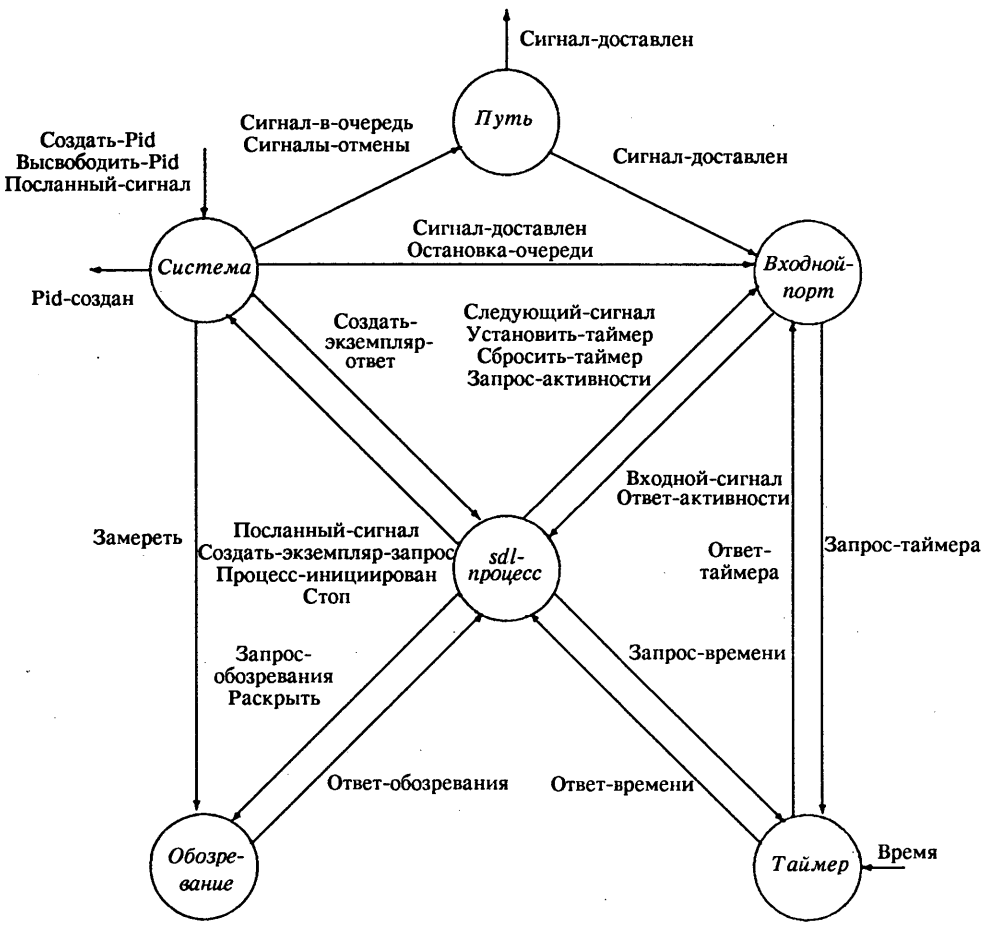

Рисунок 2. Схема коммуникаций

3.3 Пример

На рис. 3 показана коммуникация между мета-процессами в формальном определении следующего (частичного) SDL-процесса, когда сигнал ("b") поступает из окружающей среды и процесс отвечает посылкой сигнала ("a") обратно в окружающую среду:

```
...
state S;
  input b;
  output a
...
```

Коммуникация неформально проиллюстрирована диаграммой последовательности сообщений. Путь (1) и Путь (2) обозначают два экземпляра процессора *путь*, соответствующих путям из окружающей среды к sdl-процессу (Путь(1)) и обратно (Путь(2)).

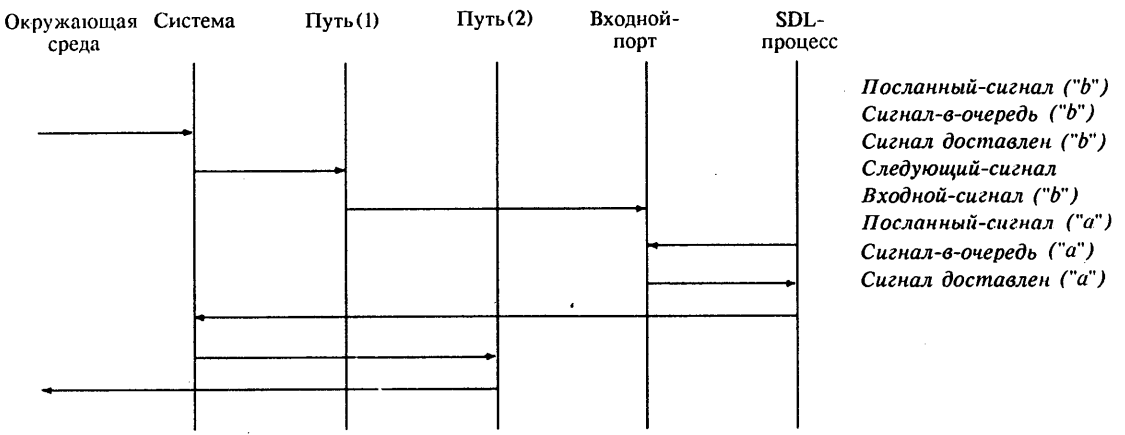

Рисунок 3. Пример коммуникации между мета-процессами

3.4 Физическая структура Формального Определения

Статическая Семантика (Приложение F.2) разбита на три основные части:

1. Определения Доменов для AS_0.
2. Определения Доменов для Семантических Доменов.
3. Функции Мета-IV, проверяющие условия корректности и определяющие, как AS_0 преобразуется в AS_1.

Определения Доменов для AS_1, используемые в частях 2 и 3, даются в Z.100 и сведены в Приложении B к Z.100. В формальном определении они не повторяются.

В Приложение F.2 также входят перекрестные указатели имен функций Мета-IV и имен доменов (в них обоих определяется вхождение и применяемые вхождения) и перекрестный указатель применяемых условий корректности.

Динамическая Семантика (Приложение F.3) разбита на пять основных разделов:

1. Определения Доменов для Коммуникационных Доменов.
2. Определения Доменов для Семантических Доменов (*Предпис-элемент*).
3. Определения Мета-процессов и присоединенные функции для модели основной системы.
4. Определения Мета-процессов и присоединенные функции для интерпретации SDL-процесса.
5. Создание внутреннего домена *Предпис-элемент*. *Предпис-элемент* используется SDL-процессами и поэтому создается до начала интерпретации SDL-процессов.

В Приложении F.3, как и в Приложении F.2, также содержится ряд указателей, охватывающих имена доменов, имена функций, имена Мета-процессов, условия ошибок и т.д.

Объем материала (особенно в Приложении F.2) на первый взгляд может показаться пугающим. Однако более половины объема занимают аннотации для Доменов и определений функций и процессов.

План, по которому излагается определение функции или процесса, таков:

1. Сначала определение функции или процесса специфицируется посредством
   (a) заголовка, определяющего имя процесса или функции и имена их формальных параметров;
   (b) тела (алгоритма) процесса или функции; и
   (c) *клаузулы типа,* специфицирующей тип (домен) формальных параметров и тип результата (если он имеется).
2. Затем по пунктам (на английском языке) следуют аннотации, относящиеся к определению процесса или функции:

Назначение	Излагается назначение функции или процесса.
Параметры	Излагается назначение каждого формального параметра функции или процесса.
Результат	Описывается возвращаемый объект (если такой имеется).
Алгоритм	Построчно объясняется алгоритм, используемый в функции или процессе.

Пример

Самая внешняя из функций в Приложении F.2 (*определение-SDL*), связывающая Статическую Семантику (*преобразовать-систему*) и Динамическую Семантику (посредством запуска Мета-процесса *система*), такова:

определение-SDL (*внешпарам, опрсист, предопрсорта*) $\triangleq$

```
1  (let (as1 , вспомогинф) = преобразовать-систему (опрсист, предопрсорта, внешпар) in
2  if as1  = nil then
3     undefined
4   else
5     (let выделподмн = выбрать-соглас-подмнож (as1 , внешпар) in
6      start система (as1 , выделподм, вспомогинф) ) )
```

type: *Внешняя-Информация Сист*$_0$ *Опрданных*$_0$ + $\Rightarrow$

Назначение Определяет свойства SDL

Параметры

внешпарам	Некоторая *Внешняя-Информация* (см. Приложение F.2, раздел 2.3).
опрсист	AS_0-дерево, представляющее SDL-систему
предопрсорта	Предопределенные данные в форме AS_0.

Алгоритм

Строка 1	Преобразовать систему в абстрактно-синтаксическую форму (форму AS_1).
Строка 2	Если обнаружены статические ошибки (то есть если не может быть выведено никакого AS_1-представления), то поведение не определено.
Строка 4	Если статические ошибки не обнаружены, то следует:
Строка 5	Выбрать множество *Идентификаторов-Блоков*$_1$, обозначающее согласованное подмножество,
Строка 6	Создать экземпляр системы, то есть создать Мета-IV-процесс с поведением, как у основной системы.

4 Как пользоваться Формальным Определением

4.1 Пользователи SDL

Формальное Определение не предполагается в качестве справочного пособия для пользователей по SDL. Те, кто впервые знакомится с SDL, могут получить общее представление о концепциях языка (и их логическом обосновании) из Руководства для пользователей (Приложение D к Z.100), тогда как сама Рекомендация Z.100 служит справочным пособием по SDL. Однако в некоторых случаях Z.100 может оказаться недостаточно. Например,

- если некоторые свойства отсутствуют (например, некоторые ожидаемые статические условия), если одни сформулированные свойства противоречат другим или
- если трудно понять точный смысл некоторых сформулированных свойств, или
- если некоторые из свойств трудно найти (из-за отсутствия в Z.100 перекрестного указателя), или
- если пользователь хочет глубже понять более сложный материал, такой как абстрактная SDL-машина, когда и как выбирать согласованное подмножество, разрешение по контексту, механизм наследования и т.п.

В таких случаях Формальное Определение может быть полезным вспомогательным документом. Для начала, конечно, пользователь должен получить представление о структуре Формального Определения, о том, как организованы функции и для чего используются Домены. Также требуется некоторое значение нотации Мета-IV, но, поскольку функции широко аннотированы, Мета-IV можно понимать, читая определения функций в сочетании с аннотациями, предварительно ознакомившись с введением в Мета-IV (см. раздел 5, ниже). При работе с Формальным Определением можно пользоваться преимуществами, которые дают содержание и перекрестные указатели.

4.2 Системные программисты

Как упоминалось ранее, подход, связанный с Мета-IV, позволяет системным программистам производить реализацию систематически (то есть контекстный анализатор, интерпретатор и т.п.) из Мета-IV-спецификации. Для SDL контекстный анализатор можно построить по Приложению F.2, а интерпретатор — по Приложению F.3. В качестве основы для интерпретации рекомендуется использовать AS_1-представление (генерируемое контекстным анализатором). Причина этого состоит в том, что в AS_0 отсутствует контекстная информация об идентификаторах (в AS_0 они обычно не квалифицируются), и в том, что динамическую семантику спецификации в AS_0-форме трудно вывести из-за большого числа сокращений в SDL (особенно для таких концепций, как типы данных).

Следует отметить, что при этом реализация производится систематически, а не механически.

Необходимо учитывать следующие моменты:

- Для представления идеальных типов данных (доменов) в Мета-IV, таких как отображения, списки и множества, используемые в AS_0, AS_1 и Семантических Доменах, должны быть выбраны подходящие типы данных.
- В соответствии с правилами видимости в SDL (относительно того, что идентификаторы могут использоваться до того, как они определены) в Статической Семантике используется (для удобства) так называемая "неподвижная точка уравнения" (см. раздел 3.1 Приложения F.2). Но при реализации Семантические Домены могут создаваться постепенно, путем многократного просмотра AS_0-дерева (например, дескрипторы для сигналов должны быть созданы раньше, чем дескрипторы для каналов, поскольку в определениях каналов имеются ссылки на сигналы).
- Подход, связанный с инициальной алгеброй, предполагает, что в Формальном Определении осуществляются манипуляции с бесконечными объектами. В AS_1 также содержатся бесконечные объекты. Поэтому необходимо слегка модифицировать AS_1 и наложить ограничения на использование типов данных либо применить некоторый аппарат абстракции, с тем чтобы эти объекты можно было закодировать.

5 Введение в Мета-IV

Настоящий раздел содержит неформальное введение в Мета-IV. В нем также описано, как Мета-IV использовался в Формальном Определении, то есть Мета-IV объясняется в терминах Формального Определения (сокращенно ФО), что означает, что изложены лишь те части Мета-IV, которые были использованы в ФО.

5.1 Общая структура

Формальное определение (ФО) состоит из:

- Множества определений процессов и функций, определяющих семантику SDL. Процессы (называемые в Мета-IV и в ФО процессорами) используются для моделирования параллельности и поэтому употребляются только в Динамической Семантике. Синтаксически определения процессоров выглядят как определения функций (за исключением ключевого слова **processor**, следующего за именем процессора), поэтому последующее описание понятия функции применимо также к процессорам.

- Множества определений доменов, определяющих тип объектов, которыми манипулируют функции. Термины, обозначающие различные группы определений доменов, вводятся для их логической классификации. Имеются AS_0-домены, обозначающие представление конкретного синтаксиса, AS_1-домены, обозначающие абстрактный синтаксис SDL, и множество доменов *Предпис* и *Предпис-элемент*, обозначающих "внутренние" вспомогательные домены (семантические домены) соответственно в Статической и Динамической Семантике. В данном разделе мы часто будем использовать "значение", как синоним объекта, и "тип", как синоним домена.

- Множества определений глобальных констант. В ФО таких определений всего два. Они содержатся в разделе 3.13 Статической Семантики. Для понимания ФО они не существенны.

Определения могут быть расположены в любом порядке; имена, вводимые в определениях, могут быть использованы в тексте до того, как они определены.

5.2 Определения Функций

Определение функции состоит из трех частей:

1. Заголовка, начинающегося с имени функции, за которым следуют один или два списка формальных параметров. Каждый список формальных параметров заключен в скобки. Формально не имеет значения, как параметры разбиваются на два списка. Часто некоторые параметры помещаются в отдельный (второй) список, если они имеют второстепенное значение при вычислении. Например, семантические домены, часто используемые функциями, приводятся в отдельном списке параметров.

2. Тела функции, которое может быть либо выражением, либо списком предложений. Функция не обязательно должна иметь результат (см. ниже).

3. Клаузулы типа, специфицирующей тип формальных параметров и тип результата. Сначала специфицируется тип первого списка параметров, а затем второго (если он имеется), причем первый список отделяется от второго стрелкой ($\rightarrow$ или $\Rightarrow$), после чего следует еще одна стрелка и за ней — результат.

Пример

$f(a, b)(d) \triangleq$

1 /* *выражение* */

type: *ДомX ДомY* $\rightarrow$ *ДомZ* $\rightarrow$ *Дом W*

В этом примере

f — имя функции

a, b, d — формальные параметры; *a* и *b* содержатся в первом списке формальных параметров, а *d* содержится во втором списке параметров. Типом *a* является *ДомX*, типом *b* является *ДомY* и типом *d* является *ДомZ*. Типом результата является *ДомW*. Домены *ДомX*, *ДомY*, *ДомZ* и *ДомW* должны быть определены в некоторых определениях доменов.

Если формальные параметры или результат используются не в соответствии с клаузулой типа, то значит в Мета-IV-спецификации имеется ошибка. В вышеприведенном примере для обозначения некоторого Мета-IV-выражения, опущенного в целях экономии места, используется неформальный Мета-IV-текст (текст, заключенный в /* */). Неформальный Мета-IV-текст аналогичен неформальному тексту в SDL. Он широко используется в примерах данного раздела.

Обычно различают аппликативные и императивные функции. Аппликативными являются функции, которые не обращаются к элементам глобального состояния (переменным), то есть результат таких функций зависит только от значений фактических параметров, к которым они применяются. Телом аппликативной функции может быть только выражение, поскольку предложения вызывают некоторые изменения состояния. Аппликативные функции всегда должны иметь результат. Императивными являются функции, обращающиеся к глобальному состоянию или даже изменяющие его (функции с побочными эффектами). Если функция императивная, то в клаузуле типа при специфицировании результата используется стрелка $\Rightarrow$ вместо $\rightarrow$.

Например,

$f(a, b)(d) \triangleq$

1 /* *выражение, обращающееся к глобальному состоянию, или последовательность предложений*/

type: *ДомX ДомY* $\rightarrow$ *ДомZ* $\Rightarrow$ *ДомW*.

В ФО Статическая Семантика и создание внутреннего Домена *Предпис-элемент* в Динамической Семантике аппликативны.

5.3 Определения переменных

Глобальные переменные в определениях процессоров определяются на самом внешнем уровне. Они видимы для всех функций, используемых процессором, определяющим эти переменные, несмотря на то что функции обычно определяются вне определений процессоров. Однако функции, совместно используемой несколькими процессорами, запрещен доступ к переменным. Если существует несколько экземпляров данного процессора, то существует и несколько экземпляров переменной, определенной процессором. (Совместно используемых переменных нет.)

Определения переменных вводятся указанием ключевого слова dcl, за которым следует список, составленный из имен переменных, необязательного начального выражения и, наконец, тип каждой переменной.

Пример:

dcl v1 := 5 type *Цел*
dcl v2 type *ДомD*.

Здесь мы определили две переменные v1 и v2, причем переменная v1 типа целый и инициализирована значением 5, v2 типа *ДомD*. Отметим, что синтаксически переменные всегда можно отличить от прочих имен, поскольку они не выделяются курсивом. Альтернативный синтаксис определений переменных таков:

dcl v1 := 5 type *Цел*
  v2 type *ДомD*.

Доступ к значению, ассоциированному с переменной, осуществляется применением операции извлечения содержимого (разыменования), которая задается ключевым словом c.

Пример

f ( ) $\triangleq$

1 **cv1 + cv2**

type: ( ) ⇒*Цел.*

5.4 Домены

Домены обычно определяются в начале документа. Имена доменов можно синтаксически отличить от прочих имен по тому, что первая буква в них прописная. Домен определяется указанием имени домена со следующим за ним символом "::" (или "=" в случае имени синонима, как объясняется в разделе 5.4.1) и следующим затем выражением домена, выражающим его свойства (для ознакомления с нотацией доменов см. также § 1.5.1 Z.100).

Пример:

8 *Выходная-вершина*$_1$ :: *Идентификатор-сигнала*$_1$
[*Выражение*$_1$]*
[*Получатель-сигнала*$_1$]
Направить-через$_1$.

Этот пример взят из абстрактного синтаксиса SDL (для ясности все имена из AS_1 снабжены в ФО нижним индексом "$_1$"). В нем определяется именованное дерево, то есть тип данных вида записи, в котором именем типа записи является *Выходная-вершина*$_1$, а поля имеют типы *Идентификатор-сигнала*$_1$, [*Выражение*$_1$]*, [*Получатель-сигнала*$_1$] и *Направить-через*$_1$.

Самой важной операцией для именованных деревьев является операция mk- (сделать), используемая для образования и разложения объектов-деревьев (то есть значений записей).

Например, если *идсиги* обозначает объект домена *Идентификатор-сигнала*$_1$, имя *списвыраж* обозначает объект домена [*Выражение*$_1$]*, имя *получ* обозначает объект типа [*Получатель-сигнала*$_1$] и имя *через* обозначает объект домена *Направить-через*$_1$, то объект домена *Выходная-вершина*$_1$ конструируется путем написания

mk-*Выходная-вершина*$_1$ (*идсиги, списвыраж, получ, через*)

и в таком виде может быть использован в Мета-IV-выражениях. Отметим, что порядок, в котором специфицированы аргументы в операции mk-, существен. То же относится и к вызовам функций. Аналогично, если имеется объект домена *Выходная-вершина*$_1$ с именем *выходверш* и нужно получить доступ к полям, то имена полей можно ввести, разлагая этот объект (здесь выбраны те же имена, что и выше):

let mk-*Выходная-вершина*$_1$ (*идсинг, списвыраж, получ, через*) = *выходверш* **in**
/* *некоторое выражение, использующее поля*/

Имена для обозначения полей в объекте *выходверш* введены посредством конструкции **let**. Использование конструкции **let** является общим способом для введения имен объектов (не только в комбинации с операцией mk-). Конструкция **let** излагается далее в разделе 5.5.

Если некоторые поля в выражении не используются, соответствующие имена в разложении можно опустить. Например, если в выражении не используется *идсиги*, можно написать

let mk-*Выходная-вершина*$_1$ (, *списвыраж, получ, через*) = *выходверш* **in**
/**некоторое выражение, использующее списвыраж и получ*/.

Если в выражении нужно использовать только *Идентификатор-Сигнала*$_1$, альтернативной возможностью является применение операции выборки поля s-:

let *идсиг* = s-*Идентификатор-Сигнала*$_1$ (*выходверш* |) **in**
/* *некоторое выражение, использующее идсиг**/.

Выборка поля может быть использована только в том случае, если поле однозначно определяется по имени домена.

Формальные параметры можно разложить (то есть ввести имена для содержащихся в них элементов) в заголовке функции, а не в ее теле, если этим достигается большая наглядность. Например,

ввести-верш-созд (**mk**-*Вершина-запроса-на-создание*$_1$ (*pid, свыраж*)) (*предпис*) $\triangleq$

1 /* *тело ввести-верш-созд**/

type: *Вершина-запроса-на-создание*$_1$ → *Предпис-элемент* ⇒

эквивалентно

ввести-верш-созд (*вершсозд*) (*предпис*) $\triangleq$

1 (**let** **mk**-*Вершина-запроса-на-создание* (*pid, свыраж*) = *вершсозд* **in**
2 /* *тело ввести-верш-созд**/)

type: *Вершина-запроса-на-создание*$_1$ → *Предпис-элемент* ⇒

Отметим, что в этом примере имеется также второй список параметров, содержащий формальный параметр *предпис* домена *Предпис-элемент*.

5.4.1 Синонимы

Операцию выборки поля можно использовать только в том случае, если поле в определении домена представлено именем. Если, например, мы хотим применить выборку ко второму полю объектов домена *Выходная-вершина*$_1$, то *Выходная-вершина*$_1$ должна быть определена несколько по-иному:

9 *Выходная-вершина*$_1$ :: *Идентификатор-сигнала*$_1$
Списокзначений
[*Получатель-сигнала*$_1$]
Направить-через$_1$
10 Списокзначений = [*Выражение*$_1$]*.

Такая *Выходная-вершина*$_1$ является в точности тем же самым доменом, что и ранее определенная *Выходная-вершина*$_1$. Отличие состоит только в том, что второе поле получило имя, то есть мы определили синоним или сокращение для выражения домена [*Выражение*$_1$]* (символ " = " используется при определении синонимов). Зачастую синонимы выявляются по другим причинам, например, если одно выражение домена используется в нескольких местах, или для наглядности. Например, в абстрактном синтаксисе SDL *Имя-канала*$_1$, *Имя-блока*$_1$, *Имя-процесса*$_1$ и т.д. являются синонимами домена *Имя*$_1$. При этом они несут информацию для читателя о том, что объекты, представляемые различными *Именами*$_1$, принадлежат определенным классам объектов. Другим типичным случаем является случай длинного списка альтернатив. Так, например, абстрактный синтаксис для *Выражения*$_1$

11 *Выражение*$_1$ = *Пассивное-выражение*$_1$ |
Активное-выражение$_1$

12 *Активное-выражение*$_1$ = *Доступ-к-переменной*$_1$ |
Условное-выражение$_1$ |
Применение-операции$_1$ |
Императивная-операция$_1$

13 *Императивная-операция*$_1$ = *Выражение-"сейчас"*$_1$ |
Pid-выражение$_1$ |
Обозревающее-выражение$_1$ |
Выражение-активности-таймера$_1$

лучше отражает, как группируются различные типы выражений, чем

14 *Выражение*$_1$ = *Пассивное-выражение*$_1$ |
Доступ-к-переменной$_1$ |
Условное-выражение$_1$ |
Применение-операции$_1$ |
Выражение-"сейчас"$_1$ |
Pid-выражение$_1$ |
Обозревающее-выражение$_1$ |
Выражение-активности-таймера$_1$.

5.4.2 Неименованные деревья

В некоторых случаях нет необходимости именовать определение дерева. Неименованные деревья широко используются в ФО, но они анонимны, поскольку часто явно не определяются.

Пример

Первой строкой в определении *Предпис-элемент* в Динамической Семантике является

15 *Предпис-элемент* = (*Идентификатор*$_1$ *Классэлемента* $\vec{m}$ *Дескрэлемента*

в ней выражается, что *Предпис-элемент* включает отображение из двух доменов. *Идентификатор*$_1$ и *Классэлемента*, в некоторый дескриптор (*Дескрэлемента*). Эти два домена составляют неименованное дерево. Если бы использовалось именованное дерево, это определение следовало переписать так:

16 *Предпис-элемент* = *Пара* $\vec{m}$ *Дескрэлемента*
17 *Пара* :: *Идентификатор*$_1$ *Классэлемента*

Пример

Достижимость в динамической семантике определяется следующим образом:

18 *Достижимость* = (*Идентификатор-процесса*$_1$ | ENVIRONMENT)
Идентификатор-сигнала$_1$-set *Путь*.

Здесь определен синоним для неименованного дерева, содержащего три поля:

1. Поле, которое может содержать либо идентификатор процесса, либо выделенный литерал ENVIRONMENT.

2. Поле, содержащее множество идентификаторов сигналов.

3. Поле домена *Путь*.

Как показано выше, скобки в определениях доменов используются и для определения неименованных деревьев, и для выделения групп альтернатив.

Пример

Функция *сделать-формальные-параметры* в Динамической Семантике определяется так:

сделать-формальные-параметры (*спарм, уровень*) $\triangleq$

1 /* *Тело, которое здесь не приводится*/

type: *Формальный-параметр-процедуры*$_1$ ***** *Квалификатор*$_1$ → *ДФормпарм*Предпис-элемент.*

Эта функция возвращает два объекта, *ДФормпарм** и *Предпис-элемент;* это означает, что фактически она возвращает неименованное дерево, состоящее из двух объектов.

Операцию mk- нельзя применять к неименованным деревьям. Их образование и разложение осуществляются заключением полей в скобки.

Пример

образования объекта *Достижимости,* в котором *a* обозначает *Идентификатор-Процесса*$_1$, *b* — множество идентификаторов сигналов и *d* — *Путь*:

(*a, b, d*)

Если из соображений наглядности желательно обозначить объект именем (с именем обращаться проще, чем с (*a, b, d*), особенно если (*a, b, d*) используется в выражении несколько раз), снова можно воспользоваться конструкцией let, так что выражение

/* *некоторое выражение, использующее "*(*a, b, d*)*"*/

эквивалентно

(let *достичь* = (*a, b, d*) in
/**некоторое выражение, использующее "достичь"* */).

Конструкция let используется также для разложения объектов неименованных деревьев. Например, разложение объекта *Достижимости* с именем *достичь* в случае, когда по какой-то причине множество идентификаторов сигналов не используется, таково:

let (*a,* , *d*) = *достичь* in

/* *некоторое выражение, использующее a и d**/.

Когда вызывается функция, то обычно неименованные деревья, являющиеся результатом вызова функции, разлагают, то есть:

let (*списокпарм, списокпутей*) = *сделать-формальные-параметры* (...,...) in
/* *некоторое выражение, использующее результаты функции, списокпарм и списокпутей**/

эквивалентно

let *инфпарм* =*сделать-формальные-параметры* (...,...) in
let (*списокпарм, списокпутей* |) =*инфпарм* in
/* *некоторое выражение, использующее результаты функции, списокпарм и списокпутей**/.

5.4.3 Конструкции с разветвлением

В некоторых случаях должно быть возможно отличать друг от друга несколько объектов-деревьев. Например, объектом ранее определенного синонима *Императивная-операция*$_1$ является либо *Выражение-"ссйчас*$_1$*"* либо *Pid-выражение*$_1$, либо *Обозревающее-выраже-*

ние$_1$ и т.п. Если имеется *Императивная-операция*$_1$, то прежде чем его можно будет вычислить, следует определить его тип. С этой целью можно воспользоваться выражением/предложением case. Например, функция, вычисляющая императивные SDL-выражения, может иметь следующий вид:

вычисл-императив-выражение (*выраж*) $\triangleq$

```
1   cases выраж:
2   (mk-Выражение-"сейчас"1 ()
3      →вычисл-выражение-"сейчас" (),
4   mk-Обозревающее-выражение1 (идоб, pidвыраж)
5      →вычисл-обозревающее-выражение (идоб, pidвыраж),
6   mk-Выражение-активности-таймера 1 (идт, списфактпар)
7      →вычисл-выражение-таймера (идт, списфактпар),
8   T →вычисл-pid-выражение (выраж) )
```

type: *Императивная-операция*$_1$ ⇒

Отметим, что разветвление происходит в зависимости от типа *Императивной-операции*, а не от фактических значений полей в дереве. T обозначает клаузулу "в остальных случаях", используемую здесь потому, что последняя альтернатива в *Императивной-операции*$_1$ (*Pid-выражение*$_1$) является синонимом, представляющим четыре другие альтернативы, которые мы не хотим тут различать. Вычисление этих альтернатив составляется на *вычисл-pid-выражение.*

Это же можно сделать другим способом, используя булевскую операцию is-, возвращающую true, если объект, заданный в качестве аргумента, принадлежит определенному домену, например:

вычисл-императив-выражение (*выраж*) $\triangleq$

```
1   if is-Выражение-"сейчас"1 (выраж) then
2      вычисл-выражение-"сейчас"()
3    else
4      if is-Обозревающее-выражение1 (выраж) then
5         вычисл-обозревающее-выражение (s-Идентификатор-переменной1 (выраж), s-
          Выражение1 (выраж) )
6       else
7       if is-Выражение-активности-таймера1 (выраж) then
8            (let mk-выражение-активности таймера1 (идт, списфактпар) = выраж in
9            вычисл-выражение-таймера (идт, списфактпар) )
10         else
11           вычисл-pid-выражение (выраж)
```

type: *Императивная-операция*$_1$ ⇒

Отметим, что здесь проиллюстрирован доступ к полям как посредством разложения (строка 8), так и посредством выборки поля (строка 5).

Как и в большинстве других языков программирования и спецификаций, требуется, чтобы альтернативы и выражении/предложении case были "константами" (каковыми они будут, если разветвление происходит по типу дерева); это означает, что если альтернативы имеют динамическую природу (как переменные или формальные параметры), то должна использоваться конструкция if-then-else. Однако для конструкции if-then-else имеется и другая нотация, так называемое условное выражение Мак-Карти, более удобное в случае большого количества альтернатив:

вычисл-императив-выражение (*выраж*) $\triangleq$

```
1  (is-Выражение-"сейчас"1 (выраж)
2     →вычисл-выражение-"сейчас"(),
3  is-Обозревающее-выражение1 (выраж)
4        → (let mk-Обозревающее-выражение (идоб, pidвыраж) = выраж in
5         вычисл-обозревающее-выражение (идоб, pidвыраж)),
6  is-Выражение-таймера1 (выраж)
7     → (let mk-Выражение-таймера1 (идт, списфактпар) = выраж in
8        вычисл-выражение-таймера (идт, списфактпар)),
9  T → вычисл-pid-выражение (выраж))
```

type: *Императивная-операция*$_1$ $\Rightarrow$

Отметим, что некоторые имена функций в ФО также начинаются с "is-". Эти случаи легко отличить от случая операции "is-", поскольку имена функций не набираются жирным шрифтом.

5.4.4. Элементарные домены

Мета-IV обеспечивает ряд предопределенных элементарных доменов. Их нотация и ассоциированные с ними операции описываются далее.

5.4.4.1 Булевский

Мета-IV-имя *Бул* обозначает домен значений истинности, то есть множество { true, false } .

Операции для Булевского:

Нотация	тип	операция
$\neg$	*Бул* $\rightarrow$ *Бул*	отрицание
$\wedge$	*Бул* $\rightarrow$ *Бул*	и
$\vee$	*Бул* $\rightarrow$ *Бул*	или
$\supset$	*Бул* $\rightarrow$ *Бул*	импликация
$=$	*Бул Бул* $\rightarrow$ *Бул*	равно
$\neq$	*Бул Бул* $\rightarrow$ *Бул*	не равно

Пример

В терминах выражений Мета-IV свойства операций *Бул*, $\neg$, $\wedge$, $\vee$ и $\supset$ могут быть выражены следующим образом:

$\neg a$ = (if a then false else true)
$a \vee b$ = (if a then true else b)
$a \wedge b$ = (if a then b else false)
$a \supset b$ = (if a then b else true).

5.4.4.2 Целый

Для целых значений предопределены три имени доменов:

- Имя *Цел* обозначает домен всех целых значений, то есть множество { ... −2, −1,0, 1,2...} .
- Имя N_0 обозначает домен неотрицательных целых значений, то есть множество { 0, 1,2, . . . } .
- Имя N_1 обозначает домен положительных целых значений, то есть множество { 1,2, . . . }.

Операции для Целого:

Нотация	тип	операция
−	*Цел* → *Цел*	отрицание
−	*Цел Цел* → *Цел*	вычитание
+	*Цел Цел* → *Цел*	сложение
*	*Цел Цел* → *Цел*	умножение
/	*Цел Цел* → *Цел*	деление в целых
mod	N_0 N_1 → N_0	остаток от деления
=	*Цел Цел* → *Бул*	равно
≠	*Цел Цел* → *Бул*	не равно
<	*Цел Цел* → *Бул*	меньше
≤	*Цел Цел* → *Бул*	меньше или равно
>	*Цел Цел* → *Бул*	больше
≥	*Цел Цел* → *Бул*	больше или равно

5.4.4.3 Знаковый

Мета-IV-имя *Знак* обозначает домен значений знаков ASCII. Для печатаемых знаков существует представление объектов, получаемое заключением в кавычки соответствующих знаков, например "a", "Z", " ".

Операции для Знакового:

Нотация	тип	операция
=	*Знак Знак* → *Бул*	равно
≠	*Знак Знак* → *Бул*	не равно
<	*Знак Знак* → *Бул*	меньше
≤	*Знак Знак* → *Бул*	меньше или равно
>	*Знак Знак* → *Бул*	больше
≥	*Знак Знак* → *Бул*	больше или равно

Операции сравнения применяются к соответствующим числовым значениям ASCII.

Из соображений наглядности, объекты домена *Знак* + могут представляться взятой в кавычки последовательностью знаков, поэтому, например, "abc" есть то же самое, что и < "a", "b", "c" > (см. раздел 5.4.6).

5.4.4.4 Выделенный

Мета-IV-имя *Выдел* обозначает домен выделенных объектов. Это попарно различные элементарные объекты, представляемые в виде произвольных последовательностей набранных жирным шрифтом прописных букв и цифр, например **ENVIRONMENT**, **REVERSE**.

Операции для Выделенных:

Нотация	тип	операция
=	*Выдел Выдел* → *Бул*	равно
≠	*Выдел Выдел* → *Бул*	не равно

В отличие от других доменов, когда в данном контексте допустимы лишь некоторые определенные объекты *Выдел*, сами эти объекты могут встречаться в определениях доменов. Например, в абстрактном синтаксисе Z.100 *Блок-отправитель*$_1$ определяется так:

1 *Блок-отправитель*$_1$ = *Идентификатор-блока*$_1$ | **ENVIRONMENT**

По-другому *Блок-отправитель*$_1$ можно было бы определить, используя *Выдел*:

2 *Блок-отправитель*$_1$ = *Идентификатор-блока*$_1$ | *Выдел*

Однако в определении домена точнее использовать **ENVIRONMENT**, поскольку этот объект является единственным возможным значением *Выдел* в данном контексте.

5.4.4.5 Маркер (Элементарный знак)

Мета-IV-*имя Маркер* обозначает домен маркеров. Этот домен можно рассматривать как состоящий из потенциально бесконечного множества попарно различных элементарных объектов, для которых не требуется никакого представления.

Операции для Маркеров:

Нотация	тип	операция
=	*Маркер Маркер*→ *Бул*	равно
≠	*Маркер Маркер*→ *Бул*	не равно

Пример

Имя$_1$ в абстрактном синтаксисе Z.100 определяется так:

1 *Имя*$_1$ :: *Маркер*

Единственным свойством, требуемым от *Имен*$_1$ при интерпретации, является возможность их сравнения. *Имя*$_1$, таким образом, состоит из значения Маркера (конкретное написание имен несущественно).

5.4.4.6 Эллипсис

Домен Эллипсис (представленный посредством...) обозначает неспецифицированную конструкцию. Он используется в определениях доменов или в выражениях

- в случаях, когда конкретное выражение или домен не существенны для семантики, или
- когда обработка домена или выражения оставлена за рамками спецификации.

Пример

Неформальный-текст$_1$ в абстрактном синтаксисе Z.100 определяется так:

1 *Неформальный-текст*$_1$ ::

Неформальный-текст$_1$ не может быть проинтерпретирован с помощью Мета-IV. Таким образом, *Неформальный-текст*$_1$ содержит некоторый не специфицируемый в дальнейшем объект.

5.4.5 Домены множеств

Домен множеств конструируется добавлением к домену элементов ключевого слова -set (дефис важен). Например:

2 *Вершина-состояния*$_1$ :: *Имя-состояния*$_1$
Совокупность-сохраняемых-сигналов$_1$
Выходцая-вершина$_1$-set

3 *Совокупность-сохраняемых-сигналов*$_1$ :: *Идентификатор-Сигнала*$_1$-set

выражает, что объект домена *Вершина-состояния*$_1$ состоит из имени состояния, совокупности сохраняемых сигналов, которая является множеством идентификаторов сигналов, и множества Входных-вершин.

Значения множеств могут быть сконструированы с помощью прямого конструктора множества, представляющего список выражений в фигурных скобках, то есть

{ 1, 3, 5, 1 }

обозначает объект домена *Цел*-set и содержит три значения *Цел.* 1, 3, 5. Более употребителен так называемый косвенный конструктор множества, когда множество включает все элементы, удовлетворяющие некоторому условию (предикату). Например, посредством

$$\{ i \in Цел \mid 0 \leq i \leq 5 \wedge i \bmod 2 = 0 \}$$

определяется множество

{ 0, 1, 2, 3, 4, 5, 6, 8, 10, 12, 14, 16, . . . } .

Это читается так: множество значений слева от вертикальной черты (возможно, квалифицированных значением или доменом), для которых выполняется выражение справа от вертикальной черты.

Пустое множество обозначается посредством { }.

В нижеследующем изложении семантики операций на множествах *s* обозначает множество { 1, 3, 5 } :

$\in$	Операция принадлежности. Проверяет, принадлежит ли данный элемент домена элементов данному множеству, то есть $1 \in s \equiv$ **true** и $2 \in s \equiv$ **false**.
$\notin$	Проверяет, исключен ли данный элемент домена элементов из данного множества, то есть $1 \notin s \equiv$ **false** и $2 \notin s \equiv$ **true**.
$\cup$	Операция объединения. Объединяет два множества, то есть $\{ 2, 3 \} \cup s \equiv \{ 1, 2, 3, 5 \}$ и $s \cup s \equiv s$.
$\cap$	Операция пересечения. Возвращает пересечение двух множеств, то есть $\{ 2, 3 \} \cap s \equiv \{ 3 \}$ и $\{ \} \cap s \equiv \{ \}$.
\	Операция разности. Исключает из одного множества значений другое множество, то есть $s \setminus \{ 1, 2 \} \equiv \{ 3, 5 \}$ и $\{ 1,2 \} \setminus s \equiv \{ 2 \}$.
$\subseteq$	Операция собственного подмножества. Проверяет, содержится ли строго одно множество в другом, то есть $\{ 1, 5 \} \subseteq s \equiv$ **true**, $s \subseteq \{ 1, 5 \} \equiv$ **false** и $s \subseteq s \equiv$ **false**.
$\subset$	Операция подмножества. Проверяет, содержится ли одно множество в другом, то есть $\{ 1, 5 \} \subset s \equiv$ **true**, $s \subset \{ 1, 5 \} \equiv$ **false** и $s \subset s \equiv$ **true**.
card	Операция мощности. Возвращает число элементов множества, то есть **card** $s \equiv 3$ и **card** $\{ \} \equiv 0$.
union	Операция распределенного объединения. Его аргументом является набор множеств, а результатом — объединение всех множеств, содержащихся в аргументе, так что **union** $\{ s, \{ 5, 6 \}, \{ 1, 5, 8 \} \} \equiv s \cup \{ 5,6 \} \cup \{ 1, 5, 8 \} \equiv \{ 1, 3, 5, 6, 8 \}$.
$=, \neq$	Проверяют совпадение и несовпадение множеств.

Пример

В терминах Мета-IV-выражений свойства операций на множествах, $\notin$, $\cup$, $\cap$, $\subseteq$ $\subset$, **card** и **union** могут быть проиллюстрированы следующим образом:

элемент $\notin$ *s1* = (¬(*элемент* $\in$ *s1*))
s1 $\cup$ *s2* = {*элемент* | *элемент* $\in$ *s1* $\vee$ *элемент* $\in$ *s2*}
s1 $\cap$ *s2* = {*элемент* | *элемент* $\in$ *s1* $\wedge$ *элемент* $\in$ *s2*}
s1 $\setminus$ *s2* = {*элемент* | *элемент* $\in$ *s1* $\wedge$ *элемент* $\notin$ *s2*}
s1 $\subseteq$ *s2* = ($\forall$*элемент* $\in$ *s1*)(*элемент* $\in$ *s2*) $\wedge$ *s1* $\neq$ *s2*
s1 $\subset$ *s2* = ($\forall$*элемент* $\in$ *s1*)(*элемент* $\in$ *s2*)
card *s1* = (**if** *s1* = {}
then 0
else (**let** *элемент* $\in$ *s1* **in**
1 + **card** (*s1* $\setminus$ {*элемент*})))
union *s1* $\equiv$ {*элемент* | ($\exists$*множество* $\in$ *s1*)(*элемент* $\in$ *множество*)}

Кванторы ($\forall$ и $\exists$) объяснены в разделе 5.6.

5.4.6 Домены списков

Домен списков или кортежей конструируется добавлением к домену элементов символа "*", если допускается пустой список, или символа "+" в противном случае.

Пример

4 *Определение-сигнала*$_1$ :: *Имя-сигнала*$_1$
Идентификатор-ссылки-на-сорт$_1$ *

Это выражение домена означает, что определение сигнала состоит из имени сигнала и возможно пустого списка идентификаторов сортов.

Значение списка может быть сконструировано с помощью прямого конструктора кортежа. Им является список выражений в угловых скобках, то есть

⟨ 11, 12, 11, 13, 14 ⟩

обозначает объект домена *Цел*$^+$ (или *Цел**) и содержит 5 упорядоченных элементов.

Пустой список обозначается посредством ⟨⟩ .

Существуют также косвенные конструкторы списков, аналогичные имеющимся для множеств. Например, в Динамической Семантике в функции *ввести-выход-верш* конструируется кортеж (*сзнач*), содержащий значения всех фактических параметров (*свыраж*) в выходной вершине:

let *сзнач* = ⟨ *вычисл-выражение* (*свыраж* [*i*] (*предпис*) | 1 $\leq$ i $\geq$ **len** *свыраж*⟩ **in**,

что соответствует явной нумерации всех элементов в списке:

let *сзнач* = ⟨ *вычисл-выражение* (*свыраж* [1]) (*предпис*),
вычисл-выражение (*свыраж* [2]) (*предпис*),
вычисл-выражение (*свыраж* [3]) (*предпис*),
. . . ⟩ **in**.

Отметим, что скобки кортежа (⟨ и⟩) отличаются по форме от операций сравнения ⟨ и⟩ .

В нижеследующем изложении семантики операций на списках *l* обозначает список ⟨ 11, 12, 11, 13, 14⟩ :

hd Возвращает первый элемент списка, то есть **hd** *l* $\equiv$ 11.
Аргументом **hd** не должен быть пустой список (⟨⟩).

tl	Возвращает список, в котором удален первый элемент, то есть tl $l \equiv \langle 12, 11, 13, 14 \rangle$.
$[i]$	Возвращает i-й элемент списка; то есть $l\,[3] \equiv 11$ и $l\,[5] \equiv 14$. Значение индекса должно быть не меньше 1 и не больше длины списка.
len	Возвращает длину списка; то есть len $l \equiv 5$.
elems	Возвращает множество, состоящее из элементов, составляющих список; то есть elems $l \equiv \{ 11, 12, 13, 14 \}$.
ind	Возвращает множество целых объектов, являющихся значениями индексов списка; то есть ind $l \equiv \{1, 2, 3, 4, 5\}$.
$\frown$	Осуществляет конкатенацию двух списков; то есть $l \frown \langle 0,1 \rangle \equiv \langle 11, 12, 11, 13, 14, 0, 1 \rangle$.
conc	Осуществляет конкатенацию всех списков, являющихся элементами списка, заданного в качестве аргумента; то есть conc $\langle\langle 0,7 \rangle, l, \langle 9 \rangle\rangle \equiv \langle 0, 7, 11, 12, 11, 13, 14, 9 \rangle$.
$=, \neq$	Проверяют совпадение и несовпадение списков.

Пример

В терминах Мета-IV-выражений свойства операций на списках hd, tl, ind, elems и conc могут быть проиллюстрированы следующим образом:

hd l = (**if** $l = \langle\rangle$ **then undefined else** $l[1]$)
tl $l = \langle l[i] \mid 2 \leq i \leq$ **len** $l \rangle$
ind $l = \{i \mid 1 \leq i \leq$ **len** $l\}$
elems $l = \{l[i] \mid i \in$ **ind** $l\}$
conc l = (**if** $l = \langle\rangle$ **then** $\langle\rangle$ **else hd** $l \frown$ **conc tl** l)

5.4.7 Домены отображений

Домен отображения (то есть таблица) конструируется указанием домена входных объектов, со следующей за ним операцией $\vec{m}$, за которым в свою очередь следует домен объектов, в который происходит отображение (то есть домен результирующих значений).

Пример

5 *Предпис-элемент* = (*Идентификатор*$_1$ *Классэлемента*) $\vec{m}$ *Дескрэлемента* ∪
ENVIRONMENT $\vec{m}$ *Достижимость*-**set** ∪
EXPIREDF $\vec{m}$ *Срок-истек* ∪
PIDSORT $\vec{m}$ *Идентификатор*$_1$ ∪
NULLVALUE $\vec{m}$ *Идентификатор*$_1$ ∪
TRUEVALUE $\vec{m}$ *Идентификатор*$_1$ ∪
FALSEVALUE $\vec{m}$ *Идентификатор*$_1$.

Выше приведено полное определение отображения *Предпис-элемент*. В нем продемонстрировано, как используется операция → и как с помощью операции слияния доменов ∪ конструируются составные отображения, то есть если дано отображение из домена *Предпис-элемент*, то:

- в таблице отображения отыскивается объект неименованного дерева (*Идентификатор*$_1$ *Классэлемента*) и результатом будет объект домена *Дескрэлемента*.
- для значения из домена *Выдел*, **ENVIRONMENT**, результатом будет объект домена *Достижимость*-**set**,
- для значения из домена *Выдел*, **EXPIREDF**, результатом будет объект домена *Срок-истек*,
- для значения из домена *Выдел*, **PIDSORT**, результатом будет объект домена *Идентификатор*$_1$,
- для значения из домена *Выдел*, **NULLVALUE**, результатом будет объект домена *Идентификатор*$_1$,

- для значения из домена *Выдел*, TRUEVALUE, результатом будет объект домена *Идентификатор*$_1$ либо
- для значения из домена *Выдел*, FALSEVALUE, результатом будет объект домена *Идентификатор*$_1$.

Значение допустимо лишь в том случае, если оно было предварительно помещено в таблицу отображения, в отличие от функций, для которых соответствие между значениями аргумента и результата фиксировано и определяется при определении функции.

Таблицы отображения могут быть сконструированы с помощью прямого конструктора отображения, представляющего список пар из входного и результирующего значений, заключенный в квадратные скобки, то есть

[1 ↦ D,
  2 ↦ AA,
  4 ↦ BB,
  9 ↦ ABC,
  5 ↦ XYZ]

обозначает значение (таблицу) отображения из домена *Цел* $\vec{m}$ *Выдел*.

Могут быть сконструированы и косвенно заданные отображения. Например, отображение

$[a \mapsto b \mid a \in N_1 \ \Lambda \ a * a = b]$

эквивалентно бесконечному отображению

[1 ↦ 1,
 2 ↦ 4,
 3 ↦ 9,
 . . . ↦ . . .].

В нижеследующем изложении семантики операций на отображениях m обозначает первое из явно специфицированных выше отображений:

m (*входноезначение*) Возвращает результирующее значение отображения, то есть $m\ (1) \equiv$ D и $m\,(9) \equiv$ ABC.

\+ Накладывает таблицу одного отображения на таблицу другого отображения. Эта операция не коммутативна, то есть

m + [0 ↦ XX, 1 ↦ B] ≡

[0 ↦ XX, 1 ↦ B,2 ↦ AA,4 ↦ BB,9 ↦ ABC,5 ↦ XYZ],

в то время как

[0 ↦ XX, 1 ↦ B] + m ≡

[0 ↦ XX, 1 ↦ D,2 ↦ AA,4 ↦ BB,9 ↦ ABC,5 ↦ XYZ].

Исключает из отображения заданное множество входных значений, так что

$m \setminus \{ 1, 2, 3 \}$ есть

[4 ↦ BB, 9 ↦ ABC, 5 ↦ XYZ].

dom Возвращает множество, содержащее в точности все входные значения данного отображения, то есть

dom $m \equiv \{ 1, 2, 4, 5, 9 \}$.

rng Возвращает множество, содержащее в точности все результирующие значения данного отображения, то есть

rng $m \equiv \{$ D, AA, BB, ABC, XYZ $\}$.

=, ≠ Проверяют совпадение и несовпадение отображений.

merge Отправляясь от данного множества отображений, возвращает отображение, конструируемое путем слияния всех таблиц отображений из этого множества, то есть

$\{m, [0 \mapsto \mathbf{WE}], [10 \mapsto \mathbf{D}]\} \equiv$

$[0 \mapsto \mathbf{WE}, 10 \mapsto \mathbf{D}, 1 \mapsto \mathbf{D}, 2 \mapsto \mathbf{AA}, 4 \mapsto \mathbf{BB}, 9 \mapsto \mathbf{ABC}, 5 \mapsto \mathbf{XYZ}]$.

Если какие-то из содержащихся в данном множестве отображений имеют перекрывающиеся входные значения, то в качестве результирующих значений берутся любые возможные.

Пустое отображение обозначается так: [ ] (две близко расположенные квадратные скобки).

Пример

В терминах Мета-IV-выражений свойства операций на отображениях, \, + и **merge** могут быть проиллюстрированы следующим образом:

$m1 \setminus s = [a \mapsto b \mid a \in \mathbf{dom}\, m1 \setminus s \wedge m1(a) = b]$
$m1 + m2 = [a \mapsto b \mid (a \in \mathbf{dom}\, m2 \wedge m2(a) = b) \vee (a \in \mathbf{dom}\, m1 \setminus \mathbf{dom}\, m2 \wedge m1(a) = b)]$

merge $m1 =$ (**if** $m1 = \{\}$
then []
else (**let** *элемент* $\in m1$ **in**
элемент + **merge** $m1 \setminus \{$*элемент*$\}$))

5.4.8 Pid-домены

Pid-домен (соответствующий сорту Pid в SDL) конструируется посредством символа П. Он может (необязательно) квалифицироваться типом процессора, с тем чтобы указать, какие Pid-значения обозначаются доменом, например:

6 *Сигналы-Отмены* :: *П* (*входной-порт*)

Домен *Сигналы-Отмены* (определяемый в Динамической Семантике) содержит Pid-объекты, квалифицируемые типом процессора *входной-порт*. Pid-значения Мета-IV не следует путать с Pid-значениями SDL, являющимися в SDL *Пассивными-термами*$_1$, так что домен Pid-значений SDL будет определяться в Динамической Семантике следующим образом:

7 *Pid-значение* = *Значение*
8 *Значение* = *Пассивный-терм*$_1$.

Pid-значения Мета-IV создаются применением предложения/выражения **start**. Оно соответствует запросу на создание в SDL. Например, когда процессор *система* создает экземпляр процессора *таймер* с фактическим параметром *фтаймера*, это выглядит так:

пример

start *таймер* (*фтаймера*).

Если в выражении используется конструкция **start**, то она создает экземпляр процессора и возвращает Мета-IV-Pid-значение этого экземпляра (соответствующее значению OFFSPRING в SDL). Например, когда в процессоре *sdl-процесс* процессор *входной-порт* вводится конструкцией **start**:

start *входной-порт* (*псам*, *предпис* (**EXPIRED**)),

то создается экземпляр процессора *входной-порт* и результирующее Мета-IV-Pid-значение используется *sdl-процессом* для идентификации *входного-порта*. Созданному экземпляру передаются параметры *псам* и *предпис* (**EXPIRED**).

Коммуникация осуществляется синхронизированными коммуникационными примитивами **input** и **output**. В конструкции **output** можно выбрать коммуникацию либо с конкретным экземпляром процессора, либо с неспецифицированным экземпляром конкретного типа процессора.

Пример

output mk-*Некоторое-дерево (некоторое значение, некотдругое значение,...)* **to** *p*.

где *p* либо обозначает Pid-значение, либо является именем типа процессора. Значения, посылаемые процессором, обычно заключаются в объект именованного дерева (некоторого коммуникационного домена), и поэтому такие деревья отвечают понятию сигнала в SDL, то есть *Некоторое-дерево* может рассматриваться как сигнал.

В конструкции input указывается коммуникационный объект, который должен быть принят, а также действия, осуществляемые после того, как объект принят. Кроме того, можно специфицировать имя, которое после приема объекта будет обозначать либо Pid-значение процессора-отправителя (соответствующее SENDER в SDL), либо будет ограничивать множество возможных отправителей.

Пример

input mk-*Некоторое-дерево (a, b, d)* **from** *p*
⇒ /* *некоторые предложения или выражение* */.

После приема *Некоторого-дерева a, b* и *d* будут значениями, переданными посредством *Некоторого-дерева*, а для *p* есть три возможные интерпретации:

- Если *p* является именем типа процессора, то коммуникационный объект должен быть принят от экземпляра данного типа процессора.
- Если *p* — это имя, которое еще не определено, то его вхождение является определяющим вхождением для имени и оно доступно в выражении или предложениях, которые следуют за клаузулой input. Оно обозначает Мета-IV-Pid-значение отправителя.
- Если *p* — это выражение, то оно должно быть типа П, и коммуникационный объект должен быть принят от экземпляра процессора, обозначаемого этим выражением.

Если может быть принят один из нескольких коммуникационных объектов, то указывается набор input-конструкций, разделяемых запятыми, а весь набор заключается в фигурные скобки:

{ **input** mk-*Некоторое-дерево (a, b, d)* **from** *p*
⇒ ; / *некоторые предложения, либо выражение* */,
input mk-*Некоторое-другое-дерево (a, b, d)* **from** *p*
⇒ /* *некоторые предложения, либо выражение* */}

В некоторых случаях желательно указывать, что может быть осуществлен либо ввод, либо вывод, в зависимости от того, какая из коммуникаций возможна первой (что не требуется в SDL, поскольку в SDL коммуникации асинхронны). В этих случаях output-конструкции включаются во множество актов коммуникации:

{ **input** mk-*Некоторое-дерево (a, b, d)* **from** *p*
⇒ /* *некоторые предложения, либо выражение* */,
input mk-*Некоторое-другое-дерево (a, b, d)* **from** *p*
⇒ /* *некоторые предложения, либо выражение* */,
output mk-*Нечто* /* *выражение* */, /* *выражение* */ **to** *pi* }.

Часто, если коммуникация должна осуществляться повторно, используется конструкция cycle вместе с конструкциями input и output:

cycle { **input** mk-*Некоторое-дерево (a, b, d)* **from** *p*
⇒/* *некоторые предложения, либо выражение* */,
input mk- *Некоторое-другое-дерево (a, b, d)* **from** *p*
⇒ /* *некоторые предложения, либо выражение* */,
output mk-*Нечто* (/* *выражение* */, /* *выражение* */) **to** *pi* } ,

что означает, что после акта коммуникации экземпляр процессора произведет необходимые действия, а затем будет ожидать очередной акт коммуникации.

5.4.9 Домены ссылок

Когда объявляется Мета-IV-переменная,

dcl v **type** *Цел,*

в Мета-IV-памяти выделяется участок, а переменная (v) будет обозначать ссылку на этот участок. Как было показано ранее, для доступа к содержимому участка используется операция c (операция разыменования). Если переменная используется без операции разыменования, то результатом будет значение домена ref, то есть ссылка на местоположение участка. ref-домены специфируются ключевым словом ref, за которым следует соответствующий домен. Например,

9 *ДПерем* :: *Идентификатор-переменной*$_1$ *Идентификатор-сорта-ссылок*$_1$
[**REVEALED**] **ref** *Пмт.*

Дескриптор переменной включает ссылку на домен *Пмт.* Дескриптор *ДПерем* определяется в Динамической Семантике и описывается далее в соответствующих аннотациях.

5.4.10 Необязательные домены

Квадратные скобки, широко используемые в определениях доменов, обозначают необязательность вхождений.

Пример

10 *Определение-сигнала*$_1$ :: *Имя-сигнала*$_1$
Идентификатор-сорта-ссылок$_1$*
[*Уточнение-сигнала*$_1$].

Квадратные скобки обозначают, что среди объектов дерева *Определение-сигнала*$_1$ объект домена *Уточнение-сигнала* может как присутствовать, так и отсутствовать. Если он отсутствует, то соответствующее поле будет содержать не имеющее типа значение nil.

Пример

```
(let mk-Определение-сигнала (имя, сорт, уточнение) =/* некоторый объект Определения-
   сигнала1 */ in
if уточнение = nil then
   /* некоторые действия */
 else
 (let mk-Уточнение-сигнала (. . .)-уточнение in
   /*некоторые другие действия, использующие уточнение сигнала*/)).
```

5.5 Конструкции let и def

Как было показано ранее, конструкция let может использоваться для образования и разложения объектов. Более широко конструкция let используется, когда нужно, чтобы некоторое имя обозначало конкретный объект (часто только для того, чтобы избежать появления слишком сложных и трудно читаемых выражений). Имена, встречающиеся в конструкции let слева от знака равенства, являются определяющими вхождениями (за исключением имен доменов, которые всегда должны определяться где-либо в определени-

ях доменов). Введенное имя может также использоваться в правой части от знака равенства (в этом случае имя определяется рекурсивно), и в выражении, следующем за конструкцией **let**. В приведенном ниже примере *имя1* видимо (то есть может быть использовано) в /**выражении1**/, /**выражении2**/, /**выражении3**/ и /**выражении4**/, *имя2* видимо в ;/ *выражении2* */, /**выражении3**/ и /**выражении4**/, *и имя3* видимо в /**выражении3** */ и /* *выражении4**/. Для ограничения видимости имен, вводимых посредством let, конструкция let заключается в скобки. В вышеприведенном примере уточнение сигнала составляет выражение, которое начинается с левой скобки по причине использования конструкции let.

Последовательность let-конструкций можно указывать двумя способами:

let *имя1* = /* *выражение1* */ **in**
let *имя2* = /* *выражение2* */ **in**
let *имя3* =/* *выражение3* */ **in**
/* *выражение4**/

либо

let *имя1* = /* *выражение1* */,
имя2 =/ **выражение2* */,
имя3 =/* *выражение3* */ **in**
/* *выражение4* */.

Первая форма использования трех let-конструкций обычно применяется в ФО, когда существен порядок, то есть если в /* *выражении2* */ используется *имя1* и в /**выражении3* */ используется *имя2*, тогда как вторая форма используется, когда различные let-конструкции независимы.

Существует несколько различных форм let-конструкции. Ранее уже было показано ее использование для разложения объектов. Другие формы ее использования таковы:

let *имя* ∈ *множествоилиимя1* **in**
/* *некоторое выражение, использующее имя* */
let *имя* **be s.t.** /* *условие, использующее имя* */ **in**
/* *некоторое выражение, использующее имя* */
let *имя* ∈ *множествоилиимя2* **be s.t.** /* *условие, использующее имя* */ **in**
/* *некоторое выражение, использующее имя* */
let *имя* (*параметры*) = /* *тело функции* */ **in**
/* *некоторое выражение, применяющее имя* */.

Первая форма читается так: Извлечь произвольное значение, принадлежащее множеству или домену, обозначенному *множествоилиимя1*, и обозначить это значение *именем*.

Вторая форма читается так: Сконструировать значение, то есть пусть *имя* будет таким, чтобы для значения выполнялось указанное условие.

Третья форма — это комбинация двух предыдущих, когда применяются оба ограничения. Если не существует ни одного такого значения, то спецификация ошибочна.

Четвертая форма читается так: Сконструировать локальную функцию (названную *именем*), имеющую некоторые формальные параметры (*праметры*), и тело.

Пример

Определение квадратного корня из числа 3:

let $r \in$ *Веществ* **be s.t.** $r > 0 \ \Lambda r * r = 3$ **in**

Пример

Определение функции, задающей факториал, где *n*—формальный параметр:

let *факт* (*n*) = if *n*< 0 then error else if *n* = 0 then 1 else *n* * *факт* (*n*—*1*) in.

При определении имени объекта, конструируемого со ссылкой на глобальное состояние (то есть если имя определяется в терминах императивного выражения), вместо обозначения let используется обозначение def, то есть ключевое слово let заменяется ключевым словом def, знак равенства заменяется двоеточием, а ключевое слово in заменяется точкой с запятой (поскольку конструкция def используется в контексте предложений, см. раздел 5.7). Например, если требуется обозначить именем значение созданного экземпляра процессора, пишем:

(def *pid*:start *входной-порт (некоторое значение)*;
/* *некоторые предложения, использующие pid-значение* */)

либо, если нужно разложить результат императивной функции, пишем:

(def mk-*Некоторое-дерево (a, b) : некоторая-императивная-функция* (. . .);
/* *некоторые предложения, использующие a и b* */).

Существует также def-версия конструкции "является таким, что":

(def *r* ∈ *Веществ* s.t. r. > 0 Λ *r* * *r* = c *v*1;
/**некоторые предложения, использующие r* */,

где def использовано потому, что при вычислении r используется переменная (v1). Она читается так: Определить такое *Вещественное* значение *r*, при котором квадрат *r* равен содержимому переменной v1.

Следует отметить, что имена, вводимые посредством let и def, не являются переменными. Они суть имена, представляющие конкретные значения, и присваивание таким образом новых значений не допускается.

5.6 Квантификация

В Мета-IV есть также математические кванторы — квантор общности, представляемый символом ∀, квантор существования, представляемый символом ∃, и квантор единственности, представляемый символом ∃ !. Эти кванторы могут использоваться в квантифицированных выражениях, возвращающих булевское значение "истина", если удовлетворено указанное условие (предикат) на объект.

Пример

идентификаторы-определенные-на-уровне-системы (*p*) ≜

1 (∀ mk-*Идентификатор*$_1$ (*q*), ∈ *p*) (len *q* = 1)

type: *Идентификатор*$_1$-set → *Бул.*

Эта функция возвращает значение "истина" только в том случае, если для каждого идентификатора (*Идентификатор*$_1$) во множестве *p* длина его квалификатора (*q*) равна 1 (во вторую пару скобок заключено предикатное выражение).

Пример

один-идентификатор-определен-на-уровне-системы (p) ≜

1 (∃ mk-*Идентификатор*$_1$ (*q*), ∈ *p*) (len *q* = 1)

type: *Идентификатор*$_1$-set → Бул.

Эта функция возвращает значение "истина" только в том случае, если во множестве *p* существует хотя бы один идентификатор (*Идентификатор*$_1$), длина квалификатора (*q*) которого равна 1.

Пример

ровно-один-идентификатор-определен-на-уровне-системы (p) $\triangleq$

1 ($\exists$! mk-*Идентификатор*$_1$ (q,) $\in p$) (**len** q = 1)

type: *Идентификатор*$_1$-**set** → Бул.

Эта функция возвращает значение "истина" только в том случае, если во множестве p существует ровно один идентификатор (*Идентификатор*$_1$), длина квалификатора (q) которого равна 1.

Вместо того чтобы разлагать идентификатор в квантификации, его можно разложить в предикатном выражении:

идентификаторы-определенные-на-уровне-системы (p) $\triangleq$

1 ($\forall$ $p' \in p$)
2 ((**let** mk-*Идентификатор*$_1$ (q,) = p' **in**
3 **len** q = 1))

type: *Идентификатор*$_1$-**set** → Бул.

Отметим, что апостроф и черточка являются допустимыми знаками в именах Мета-IV.

5.7 Вспомогательные предложения

- Тождественное предложение.
  Ключевое слово I указывает на пустое предложение, то есть на предложение, которое ничего не делает.

- Неопределенное предложение/выражение.
  Ключевое слово **undefined** указывает на то, что не может быть дано никакой семантики.

- Предложение возврата.
  Ключевое слово **return**, за которым следует выражение, завершает выполнение императивной функции; результатом будет данное выражение.

- Предложение/выражение-ошибка.
  В ФО ключевое слово **error** указывает на динамическую SDL-ошибку.

- Предложение присваивания.
  Аналогично имеющемуся в SDL. При присваивании значений переменным операция разыменования (c) не применяется.

- Предложения for и while.
  Те же (хорошо известные) понятия, что и в CHILL.
  Предложения, которые должны повторяться, заключаются в скобки.

- Предложение/выражение захватывание выходов и прерываний. Ловит (обрабатывает) прерывания, вызываемые предложением/выражением выхода. Если в предложении выхода указан аргумент, то ловушка действует только в том случае, если выражение в ней согласуется со значением указанного аргумента. Особая версия механизма прерываний — конструкция — использовалась в функциях *ввести-граф-процесса* и *ввести-граф-процедуры.* Конструкция tixe объясняется в соответствующих аннотациях.

5.8 Отклонения от нотации, используемой в формальном определении CHILL

- В формальном определении CHILL имена предопределенных доменов набираются жирным шрифтом прописными буквами (например, **BOOL**, **INTG**), а имена, обозначающие семантически домены, могут содержать только прописные буквы.

В формальном определении SDL все имена доменов набираются курсивом, начинаются с прописной буквы и содержат хотя бы одну строчную.

- В формальном определении CHILL все объекты конечные.

  В формальном определении SDL объекты могут быть бесконечными. Семантика применения некоторых операций над бесконечными объектами плохо определена. Например, такие операции, как операции равенства и вычисления мощности, не следует применять к потенциально бесконечным объектам.

  Кроме того, в Приложении F.2 в *преобразовать-процесс* использована специальная константа *бесконечность* для представления "неограниченного числа экземпляров" в AS_1.

- В формальном определении SDL расширена нотация Мета-IV, с тем чтобы включить в себя элементарный домен *Знак* и объекты-знаковые строки (см. раздел 5.4.4.3).

- В Приложении F.3 в процессоре *путь* использована так называемая "защита выхода". Это понятие описано как в аннотациях, относящихся к процессору *Путь*, так и в [4].

5.9 Пример: игра Демон, специфицированная на Мета-IV

Ниже показано, как Мета-IV может быть использован для определения семантики игры Демон. За дальнейшими деталями, касающимися игры Демон, можно обратиться к §2.9 Z.100.

Коммуникации *демон → монитор* и *монитор → игра*

11 *Удар* :: ()

Коммуникация *пользователь → монитор*

12 *Новаяигра* :: ()

Коммуникация *игра → монитор*

13 *Игразакончена* :: *П*

Коммуникация *монитор → игра*

14 *Потвигразакончена* :: ()

Коммуникация *игра → пользователь*

15 *Идигры* :: ()
16 *Выигрыш* :: ()
17 *Проигрыш* :: ()
18 *Счет* :: *Цел*

Коммуникация *пользователь → игра*

19 *Попытка* :: ()
20 *Результат* :: ()
21 *Конецигры* :: ()

введ-игры-демон () $\triangleq$
1 **start** *монитор* ()

type: () $\Rightarrow$ ()

монитор **processor** () $\triangleq$

```
 1  (dcl множпольз := {} type П-set,
 2       множигр := {} type П-set;
 3   cycle (input mk-Новаяигра() from отправитель
 4             ⇒  if отправитель ∉ c множпольз then
 5                    (def потомок : start игра (отправитель);
 6                     множигр := c множигр ∪ {потомок};
 7                     множпольз := c множпольз ∪ {отправитель})
 8                  else
 9                    I,
10          input mk-Игразакончена (игрок) from отправитель
11             ⇒  (множигр := c множигр \ {отправитель};
12                 множпольз := c множпольз \ {игрок};
13                 output mk- Подтвигразакончена () to отправитель),
14          input mk-Толчок() from демон
15             ⇒  for all pid ∈ множигр do
16                 output mk- Удар () to pid))
```

type: () ⇒

игра **processor** (*игрок*) $\triangleq$

```
 1  (dcl счетч := 0 type Цел;
 2   dcl четный := true type Бул;
 3   output mk-Идигры () to игрок;
 4   cycle (input mk- Попытка() from пользователь
 5             ⇒  if c четный
 6                   then (output mk- Выигрыш() to игрок;
 7                         счетч := c счетч + 1)
 8                   else (output mk-Проигрыш() to игрок;
 9                         счетч := c счетч − 1),
10          input mk-Результат() from пользователь
11             ⇒  output mk-Счет (счетч) to игрок,
12          input mk-Конецигры() from пользователь
13             ⇒  (output mk-Игразакончена (игрок) to монитор;
14                 input mk- Подтвигразакончена () from монитор
15                    ⇒  stop),
16          input mk-Удар () from монитор
17             ⇒ четный := ¬c четный))
```

type: П ⇒ ()

Библиография

[1] Dines Bjørner and Cliff B. Jones
Formal specification and software development
Prentice-Hall Publ. 1982.

[2] The Formal Definition of CHILL
CCITT Manual
ITU, Geneva 1981.

[3] P. Folkjær, D. Bjørner
A formal model of a generalized CSP-like language,
IFIP 8th World Computer Conference
Proceedings, North-Holland Publ. 1980.

[4] C.A.R. Hoare
Communicating Sequential Processes
Prentice-Hall 1985.

Printed in USSR • 1991 — ISBN 92-61-03774-7