МЕЖДУНАРОДНЫЙ СОЮЗ ЭЛЕКТРОСВЯЗИ

МККТТ

МЕЖДУНАРОДНЫЙ
КОНСУЛЬТАТИВНЫЙ КОМИТЕТ
ПО ТЕЛЕГРАФИИ И ТЕЛЕФОНИИ

СИНЯЯ КНИГА

ТОМ X — ВЫПУСК X.5

ПРИЛОЖЕНИЕ F.3 К РЕКОМЕНДАЦИИ Z.100: ФОРМАЛЬНОЕ ОПРЕДЕЛЕНИЕ ЯЗЫКА SDL

ДИНАМИЧЕСКАЯ СЕМАНТИКА

IX ПЛЕНАРНАЯ АССАМБЛЕЯ
МЕЛЬБУРН, 14–25 НОЯБРЯ 1988 ГОДА

PAGE INTENTIONALLY LEFT BLANK

PAGE LAISSEE EN BLANC INTENTIONNELLEMENT

МЕЖДУНАРОДНЫЙ СОЮЗ ЭЛЕКТРОСВЯЗИ

МККТТ

МЕЖДУНАРОДНЫЙ
КОНСУЛЬТАТИВНЫЙ КОМИТЕТ
ПО ТЕЛЕГРАФИИ И ТЕЛЕФОНИИ

СИНЯЯ КНИГА

ТОМ X — ВЫПУСК X.5

ПРИЛОЖЕНИЕ F.3 К РЕКОМЕНДАЦИИ Z.100: ФОРМАЛЬНОЕ ОПРЕДЕЛЕНИЕ ЯЗЫКА SDL

ДИНАМИЧЕСКАЯ СЕМАНТИКА

IX ПЛЕНАРНАЯ АССАМБЛЕЯ
МЕЛЬБУРН, 14–25 НОЯБРЯ 1988 ГОДА

ISBN 92-61-03794-1

Printed in Russia

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 интегрального обслуживания (ЦСИО) — Общая структура, услуги и возможности обслуживания. Рекомендации I.110—I.257 (Исследовательская комиссия XVIII).

ВЫПУСК III.8 — Цифровая сеть интегрального обслуживания (ЦСИО) — Общесетевые аспекты и функции, интерфейсы "пользователь — сеть" ЦСИО. 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 измерительной аппаратуре. 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 с системами подвижной спутниковой связи. 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 аттестационных испытаний для Рекомендаций по телетексу. 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 предписания. 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 линейного оборудования. Рекомендации серии L (Исследовательская комиссия VI).

Том X

ВЫПУСК X.1 — Язык функциональной спецификации и описания. Критерии применения методов формальных описаний. Рекомендация Z.100 с Приложениями A, B, C и E, Рекоменда-ция 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.5 СИНЕЙ КНИГИ

Приложение F.3 к Рекомендации Z.100

ПРИМЕЧАНИЕ

Ввиду особого характера семантики SDL данный Выпуск до настоящего времени был опубликован только на английском языке.

ПРЕДВАРИТЕЛЬНЫЕ ЗАМЕЧАНИЯ

1 Вопросы, порученные каждой Исследовательской комиссии на исследовательский период 1989—1992 гг., содержатся в Документе № 1 для данной Исследовательской комиссии.

2 В настоящем Выпуске термин «администрация» используется для краткости и обозначает как администрацию электросвязи, так и признанную частную эксплуатационную организацию.

PAGE INTENTIONALLY LEFT BLANK

PAGE LAISSEE EN BLANC INTENTIONNELLEMENT

Содержание

PAGE INTENTIONALLY LEFT BLANK

PAGE LAISSEE EN BLANC INTENTIONNELLEMENT

ВЫПУСК X.5

Приложение F.3 к Рекомендации Z.100

ФОРМАЛЬНОЕ ОПРЕДЕЛЕНИЕ ЯЗЫКА SDL
ДИНАМИЧЕСКАЯ СЕМАНТИКА

PAGE INTENTIONALLY LEFT BLANK

PAGE LAISSEE EN BLANC INTENTIONNELLEMENT

Введение

В настоящем Формальном Определении SDL дается определение динамических свойств языка SDL. Описание всей структуры Формального Определения и объяснение используемой нотации содержатся в Приложении F.1: Введение в Формальное Определение.

SDL-система интерпретируется как ряд параллельных процессов. Коммуникация между ними является синхронной, подобно коммуникации CSP (Communicating Sequential Processes — Сообщающиеся Последовательные процессы). Линии на рисунке указывают на наличие такой коммуникации. Процесс-*системы* создает экземпляры других процессов: один экземпляр процесса-*обозревания* и процесса-*таймера*, один экземпляр процесса-*пути* для каждого из различных путей транспортировки SDL-выхода, один экземпляр взаимодействующей пары (*sdl-процесс, входной-порт*) для каждого действительного экземпляра SDL-процесса. Таким образом, в модели используются шесть различных типов мета-процессов:

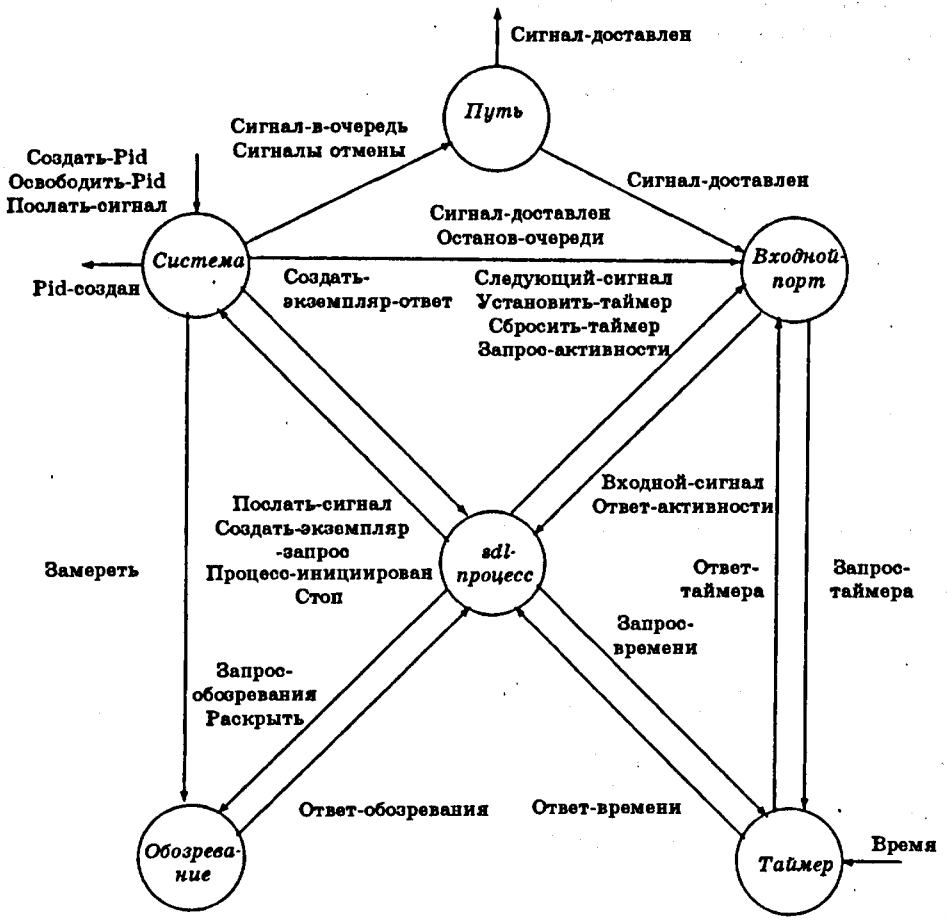

РИСУНОК 1.— Структура модели интерпретации

Указанными процессами являются:

1. *Система*

   Управление маршрутизацией сигналов и создание *sdl-процессов*.

2. *Путь*

   Обработка недетерминированных задержек в каналах и маршрутах сигналов. Следует отметить, что к одной задержке в экземпляре *пути* прибавляются все возможные задержки в маршрутах сигналов и в каналах, через которые проходит один экземпляр сигнала.

3. *Таймер*

   Слежение за текущим временем и обращения с тайм-аутами. Когда *sdl-процесс* обрабатывает выражение NOW, он запрашивает у *таймера* значение времени.

   Предполагается, что внешняя среда посылает в *таймер* сигнал времени через постоянные интервалы. Этот механизм называется процессом-импульсов-сигналов-времени (tick-process). Следует отметить, что неформальная модель процесса-импульсов-сигналов-времени не образует раздел динамической семантики, а используется только для пояснений.

4. *Обозревание*

   Слежение за всеми раскрытыми переменными. Каждый раз, когда *sdl-процесс* модифицирует раскрытую переменную, он посылает *обозреванию* новую переменную. Когда процесс использует выражение VIEW (ОБОЗРЕВАНИЕ), он запрашивает от *обозревания* текущее значение.

5. *sdl-процесс*

   Интерпритация поведения SDL-процесса.

6. *Входной-порт*

   Управление образованием очереди сигналов в SDL-процессе. Для каждого экземпляра *sdl-процесса* существует ровно один экземпляр *входного-порта*. Сигналы принимаются *sdl-процессом* всегда в его *входном-порте*.

1 Домены для коммуникации Процессов

1.1 *sdl-process ↔ system* *(sdl-процесс ↔ система)*

1 *Process-Initiated* :: *Port*
2 *Port* = *Π(input-port)*

Когда *sdl-процесс* создан, он отвечает *Процесс-Инициирован (Process-Initiated)*, если он готов интерпретировать свой граф процесса. Переносимые данные являются CSP-экземпляром входного-порта, запускаемого экземпляром процесса.

3 *Create-Instance-Request* :: *Process-identifier*$_1$ *Value-List*
4 *Create-Instance-Answer* :: [*Offspring-Value*]
5 *Offspring-Value* = *Pid-Value*
6 *Pid-Value* = *Value*
7 *Value* = *Ground-term*$_1$

Когда процесс интерпретирует вершину запроса на создание, он будет выдавать *системе* сигнал *Запрос-Создания-Экземпляра (Create-Instance-Request)*. Данные, которые переносятся, представляют идентификатор запускаемого процесса и список фактических параметров. *Система* будет отвечать выдачей в обратном направлении сигнала *Создать-Экземпляр-Ответа (Create-Instance-Answer)*, который переносит Pid-Значение стартовавшего процесса. Если процесс не мог стартовать, то происходит возврат nil.

8 *Send-Signal* :: *Signal-identifier*$_1$ *Value-List* [*Pid-Value*] *Direct-via*$_1$

Когда экземпляр *sdl-процесса* интерпретирует выходную вершину, он будет выдавать *Послать-Сигнал (Send-Signal)*. При этом передаются идентификатор посылаемого SDL-сигнала, список приданных к сигналу необязательных значений, необязательный экземпляр процесса-получателя, необязательный набор идентификаторов каналов или идентификаторов маршрутов сигналов.

9 *Stop* :: ()

Когда экземпляр *sdl-процесса* интерпретирует стоп-вершину, он передает сигнал *Стоп (Stop) системе*, которая определяет, являются ли экземпляры действующими или недействующими («замершими»).

1.2 *sdl-process* ↔ *input-port* (*Sdl-процесс*↔ *входной порт*)

1 *Next-Signal* :: *Signal-identifier*$_1$-set
2 *Input-Signal* :: *Signal-identifier*$_1$ *Value-List Sender-Value*

sdl-процесс выдает *Следующий-Сигнал (Next-Signal)* своему *входному-порту*, а входной-порт отвечает (если очередь не пуста) выдачей *Входного-Сигнала (Input-Signal)*. Сигналы, которые должны оставаться в очереди (сохраняемый набор), обозначаются как *Идентификатор-сигнала*$_1$-set (*Signal-identifier*$_1$-set).

3 *Set-Timer* :: *Timer-identifier*$_1$ *Timeout-value Arglist Equivalent-test*
4 *Reset-Timer* :: *Timer-identifier*$_1$ *Arglist Equivalent-test*
5 *Timeout-value* = *Value*
6 *Arglist* = *Value**
7 *Equivalent-test* :: *Ground-term*$_1$ *Ground-term*$_1$ → *Bool*

(Когда *входной-порт* принимает сигнал *Установить-таймер (Set-Timer)* от *sdl-процесса*, интерпретирующего действия по установке таймера, он запускает таймер по истечении времени, определенному *значением-Тайм-аута (Timeout-value)*. Таймер также имеет приданный ему список значений, который вместе *с идентификатором*$_1$*-Таймера* идентифицирует экземпляр таймера. *Входной-порт* проверяет, относятся ли два запроса (то есть *Установить-Таймер (Set-Timer), Сбросить-Таймер (Reset-Timer)* или *Запрос-Активности (Active-Request)*) к одному экземпляру таймера, сравнивая их *идентификаторы-Таймера* и применяя функцию *Теста-эквивалентности (Equivalent-test)* к элементам в двух ассоциированных *списках аргументов (Arglist)*. Функция *Теста-эквивалентности (Equivalent-test)* берет в качестве двух аргументов *Пассивные-терм*$_1$(*ы*) (*Ground-term*$_1$*s*) и возвращает (указывается с помощью «→») *Bool (Булевский)*.

8 *Active-Request* :: *Timer-identifier*$_1$ *Arglist Equivalent-test*
9 *Active-Answer* :: *Bool*

sdl-процесс посылает во *входной-порт* сигнал *запрос-Активности* для определения активности или неактивности таймера, идентифицированного *идентификатором-Таймера*$_1$. *Список аргументов* и *Тест-эквивалентности* объяснены выше.

1.3 *sdl-process ↔ view* (*sdl-процесс ↔ обозревание*)

1 *Reveal* :: *Variable-identifier*$_1$ (*Value* | UNDEFINED) *Pid-Value*

Когда *sdl-процесс* модифицирует раскрытую переменную, он будет выдавать сигнал *Раскрыть (Reveal)*. *Reveal* переносит идентификатор раскрытой переменной, новое значение переменной и *Pid-значение* «self» («сам»).

2 *View-Request* :: *Variable-identifier*$_1$ *Pid-Value*
3 *View-Answer* :: (*Value* | UNDEFINED)

Когда *sdl-процесс* обозревает переменную, он будет выдавать *Запрос-Обозревания (View-Request)*. *Запрос обозревания* переносит идентификатор обозреваемой переменной и *Pid-Значение* экземпляра, который раскрывает ее. В ответ на *обозревание* выдается *Ответ-Обозревания (View-Answer)*, переносящий запрашиваемое значение.

1.4 *sdl-process, input-port ↔ timer* (*sdl-процесс, входной-порт ↔ таймер*)

1 *Time-Request* :: ()
2 *Time-Answer* :: *Value*

Когда *sdl-процесс* оценивает значение выражения NOW (СЕЙЧАС), он посылает *Запрос-Времени (Time-Request)*. *Таймер* отвечает передачей *Ответа-Времени (Time-Answer)*, который переносит значение текущего времени.

Входной-порт постоянно контролирует окончание времени действия таймеров. Для этого ему необходимо знать реальное время *таймера*. Данная коммуникация является такой же, как коммуникация между *sdl-процессом* и *таймером*.

1.5 *system ↔ environment* (*система ↔ внешняя среда*)

1 *Create-Pid* :: *Port*
2 *Pid-Created* :: *Pid-Value*
3 *Release-Pid* :: *Pid-Value*

Поскольку относительно внешней среды должно быть высказано как можно меньше предположений, то для создания экземпляров во внешней среде определена конкретная схема. Она значительно проще схемы создания процессов в системе. Когда во внешней среде создается экземпляр процесса, то с помощью сигнала *Создать-Pid (Create-Pid)* системе передается *входной-порт* CSP-имени. Система отвечает выдачей ассоциированного *Pid-значения* SDL во внешнюю среду, переносимого с помощью *Pid-Создан (Pid-Created)*. Когда экземпляр процесса перестает существовать во внешней среде, система будет получать из внешней среды *Освободить-Pid (Release-Pid)* с *Pid-Значением* SDL остановленного процесса. Основным назначением данной схемы является согласование управления в системе Pid-Значений во внешней среде.

1.6 *system ↔ view* (*система ↔ обозревание*)

1 *Die* :: *Pid-Value*

Когда SDL-процесс останавливается, *обозревание* вводит *Замереть (Die)*, так что экземпляр может быть удален из его внутреннего отображения возможных раскрытых переменных.

1.7 *system ↔ path* (*система ↔ путь*)

1 *Queue-Signal* :: *Signal-identifier*$_1$ *Value-List Pid-Value Port*

Система передает Сигнал, выдавая *Сигнал-В-Очередь (Queue-Signal)* экземпляру пути, который соответствует выбранному маршруту от отправителя до получателя. *Сигнал-В-Очередь* передает идентификатор сигнала, значения, переносимые сигналом, Pid-значение, указывающее на отправителя, и значение CSP-экземпляра принимающего *входного-порта*.

2 *Discard-Signals* :: *Port*

Когда sdl-процесс останавливается, *система* требует, чтобы все *пути* аннулировали сигналы, направляемые к *входному-порту* остановившегося *sdl-процесса*. Это происходит в результате выдачи *Сигналов-Отмены (Discard-Signals)*.

1.8 *system, path ↔ input-port* (*система, путь ↔ входной-порт*)

1 *Signal-Delivered* :: *Signal-identifier*$_1$ *Value-List Sender-Value*
2 *Value-List* = [*Value*]*
3 *Sender-Value* = *Pid-Value*

Путь направляет сигнал во *входной-порт*, когда он свободен. *Система* направляет сигнал непосредственно во *входной-порт*, если отправитель и получатель находятся в одном блоке.

1.9 *system ↔ input-port* (*система ↔ входной-порт*)

1 *Stop-Queue* :: ()

Когда экземпляр *sdl-процесса* останавливается, *система* выдает *Останов-Очереди (Stop-Queue)*, вынуждая остановиться свой *входной-порт*.

1.10 *timer ↔ tick* (*таймер ↔ импульсы сигналов времени*)

1 *Time* :: ()

Процесс-импульсов-сигналов-времени (tick-process) не моделируется формально. Это процесс, согласно которому системе через постоянные интервалы времени передаются импульсы сигналов времени, формируя таким образом основу для процесса-таймера. Импульсы сигналов времени должны рассматриваться в качестве части входного потока, преобразуемого SDL-системой в выходной-поток.

Entity-dict содержит информацию обо всех идентификаторах SDL, относящихся к процессам, то есть *Entity-dict* используется всякий раз, когда процессу требуется информация об идентификаторе. Первоначально он получается из AS_1. Каждый процесс имеет свою собственную версию *Entity-dict*.

1 *Entity-dict* = (*Identifier*$_1$ SIGNAL) $\overrightarrow{m}$ *SignalDD* ∪
(*Identifier*$_1$ PROCEDURE) $\overrightarrow{m}$ *ProcedureDD* ∪
(*Identifier*$_1$ TYPE) $\overrightarrow{m}$ *TypeDD* ∪
(*Identifier*$_1$ SORT) $\overrightarrow{m}$ (*SyntypeDD* | *SortDD*) ∪
(*Identifier*$_1$ PROCESS) $\overrightarrow{m}$ *ProcessDD* ∪
(*Identifier*$_1$ VALUE) $\overrightarrow{m}$ (*VarDD* | *OperatorDD*) ∪
ENVIRONMENT $\overrightarrow{m}$ *Reachabilities* ∪
EXPIREDF $\overrightarrow{m}$ *Is-expired* ∪
PIDSORT $\overrightarrow{m}$ *Sort-identifier*$_1$ ∪
NULLVALUE $\overrightarrow{m}$ *Literal-operator-identifier*$_1$ ∪
TRUEVALUE $\overrightarrow{m}$ *Literal-operator-identifier*$_1$ ∪
FALSEVALUE $\overrightarrow{m}$ *Literal-operator-identifier*$_1$ ∪
SCOPEUNIT $\overrightarrow{m}$ *Qualifier*$_1$ ∪
PORT $\overrightarrow{m}$ *П*(*input-port*) ∪
SELF $\overrightarrow{m}$ *П*(*input-port*) ∪
PARENT $\overrightarrow{m}$ *П*(*input-port*)

Entity-dict состоит из отображения пары *Идентификаторов*$_1$ и ассоциированного с ними класса объекта в дескрипторы. Класс объекта — это SIGNAL (СИГНАЛ), PROCEDURE (ПРОЦЕДУРА), TYPE (ТИП), SORT (СОРТ), PROCESS (ПРОЦЕСС) или VALUE (ЗНАЧЕНИЕ).

Кроме того, *Entity-dict* содержит информацию о том, как могут быть маршрутированы сигналы, ведущие к внешней среде системы или исходящие из нее. Объект *ENVIRONMENT (ВНЕШНЯЯ СРЕДА)* рассмотрен ниже.

Дескриптор — это дескриптор сигнала, процедуры, типа, подтипа, процесса, сорта, переменной, литерала или операции. Следует отметить, что некоторые объекты идентификаторов SDL (например, каналы и блоки) исключены.

Кроме того, *Entity-dict* содержит некоторые дополнительные объекты, распознаваемые основной системой и/или sdl-процессами. Доступ к этим объектам осуществляется посредством *Выделенных* значений *(Quot*-values*)*:

ENVIRONMENT — В результате применения к *Entity-dict* получают *Достижимости (Reachabilities)*, ведущие к внешней среде или исходящие от нее.

EXPIREDF — В результате применения к *Entity-dict* получают функцию, используемую процессором *таймера*.

PIDSORT — В результате применения к *Entity-dict* получают идентификатор AS_1 PiD-сорта.

NULLVALUE — В результате применения к *Entity-dict* получают идентификатор AS_1 PiD-литерала null (нуль).

TRUEVALUE — В результате применения к *Entity-dict* получают идентификатор AS_1 булевского литерала true (истина).

FALSEVALUE — В результате применения к *Entity-dict* получают идентификатор AS_1 булевского литерала false (ложь).

SCOPEUNIT — В результате применения к *Entity-dict* получают квалификатор, обозначающий текущую структурную единицу.

PORT — В результате применения к *Entity-dict* получают П-значение входного порта sdl-процесса.

SELF — В результате применения к *Entity-dict* получают П-значение sdl-процесса, использующего *Entity-dict*.

PARENT — В результате применения к *Entity-dict* получают П-значение родителя sdl-процесса, использующего *Entity-dict*.

2.1 Дескриптор Сигнала

1 *SignalDD* :: *Sort-reference-identifier*$_1$*

SignalDD — это дескриптор сигнала. Он содержит список идентификаторов сортов или подтипов, приданных сигналу.

2.2 Дескриптор Процедуры

1 *ProcedureDD* :: *FormparmDD** *Procedure-graph*$_1$
2 *FormparmDD* = *InparmDD* | *InoutparmDD*
3 *InparmDD* :: *Variable-identifier*$_1$
4 *InoutparmDD* :: *Variable-identifier*$_1$

ProcedureDD — это дескриптор процедуры. Он содержит список дескрипторов формальных параметров и граф процедуры. Формальный параметр — это либо параметр IN, либо параметр IN/OUT, содержащие *идентификатор-Переменной*$_1$ *(Variable-identifier*$_1$*)*.

2.3 Дескриптор Типа

1 *TypeDD* :: *Sortmap Equations*$_1$
2 *Sortmap* = *Sort-identifier*$_1$ $\Rightarrow$ *Term-class-set*
3 *Term-class* = (*Ground-term*$_1$ | *Error-term*$_1$)*-set*

TypeDD — это дескриптор определения типа данных. Дескриптор содержит отображение *(Sort-map)* всех *идентификаторов-Сорта*$_1$ (*Sort-identifier*$_1$s), видимых в структурной единице, объемлющей определение типа данных, в набор эквивалентных классов, существующих для сорта. Эквивалентный класс *(Term-class)* — это набор пассивных термов, возможно объединенных с термом-ошибкой.

TypeDD содержит также *Равенства*$_1$ (*Equations*$_1$), из которых получаются эквивалентные классы.

2.4 Дескриптор Сорта

1 *SortDD* :: *Type-identifier*$_1$
2 *SyntypeDD* :: *Parent-sort-identifier*$_1$ *Range-condition*$_1$

SortDD и *SyntypeDD* — это дескрипторы новых типов и подтипов соответственно. Дескриптор нового типа содержит идентификатор объемлющего определения типа данных, поскольку все свойства новых типов содержатся в данном дескрипторе.

Дескриптор подтипа содержит также идентификатор родительского нового типа и условие на диапазон в AS_1.

2.5 Дескриптор Процесса

1	*ProcessDD*	::	*ParameterDD* Initial Maximum Process-graph*$_1$ *Reachabilities*
2	*Reachabilities*	=	*Reachability-set*
3	*ParameterDD*	=	*Variable-identifier*$_1$
4	*Initial*	=	*Intg*
5	*Maximum*	=	*Intg*
6	*Reachability*	=	(*Process-identifier*$_1$ | ENVIRONMENT) *Signal-identifier*$_1$*-set Path*
7	*Path*	=	*Path-identifier**
8	*Path-identifier*	=	*Identifier*$_1$

ProcessDD — это дескриптор процесса. Он содержит список параметров (*ParameterDD*), количество созданных экземпляров процесса во время запуска системы (*Initial*), максимальное количество разрешенных процессов (*Maximum*), граф процесса и *Достижимости* (*Reachabilities*). *Достижимость* определяет *Идентификатор-процесса*$_1$, который можно получить из процесса передачи сигнала в *Идентификатор-сигнала*$_1$-set (*Signal-identifier*$_1$-set), используя определенный *Путь*. *Путь* идентифицируется списком идентификаторов маршрутов сигналов и каналов (*Path-identifiers*). *Путь* является пустым, если *Идентификатор-процесса*$_1$ является одновременно отправителем и получателем. Дескриптор формального параметра — это *Идентификатор-переменной*$_1$ параметра.

2.6 Дескриптор Переменной

1	*VarDD*	::	*Variable-identifier*$_1$ *Sort-reference-identifier*$_1$ [REVEALED] [ref *Stg*]

VarDD — это дескриптор переменной. Он содержит идентификатор переменной, идентификатор сорта или подтипа, атрибут REVEALED (РАСКРЫТАЯ) и дополнительно ссылку на память. Для переменных обозревания нет дескриптора, так как *Определения-обозревания*$_1$ (*View-definition*$_1$s) не содержат (существенной) информации. Всякий раз, когда вызывается процедура, *Entity-dict* переписывается с дескрипторами, представляющими формальные параметры и локальные определения. Для параметров IN/OUT дескрипторы содержат ассоциированные фактические параметры и ссылку на версию памяти, где находится версия *Идентификатора-переменной*$_1$, то есть из-за возможности рекурсивных процедур в SDL может иметься несколько версий одного и того же *Идентификатора-переменной*$_1$, по одному на каждое рекурсивное обращение, и, следовательно, несколько версий памяти.

2.7 Дескриптор Операции и Литерала

1	*OperatorDD*	::	*Argument-list Result*
2	*Argument-list*	=	*Sort-reference-identifier*$_1$*
3	*Result*	=	*Sort-reference-identifier*$_1$

OperatorDD — это дескриптор операции или литерала. Он содержит список сортов или подтипов аргументов и сорт или подтип результата.

3 Основная Система

3.1 Процессор Системы

Данный процессор является точкой входа интерпретации для SDL-описания. Все другие процессы начинаются (прямо или косвенно) от этого процесса. Он начинается *с определения-SDL (definition-of-SDL)*, описанного в Приложении F.2: Статическая Семантика.

system **processor** $(as_1 tree, subset, auxinf) \triangleq$ (3.1.1)

```
 1  (let (timeinf, terminf, expiredf, delayf) = auxinf in
 2   dcl instancemap := [] type Π(sdl-process) ⇒
 3       ((ENVIRONMENT | Process-identifier1) Pid-Value);
 4   dcl queuemap := [] type Pid-Value-set ⇒Port;
 5   dcl pidno := [] type Process-identifier1 ⇒N0;
 6   dcl pathmap := [] type Channel-identifier1* ⇒Π(path);
 7   dcl pidset := {} type Pid-Value-set;
 8   trap exit with error in
 9   (let entitydict = extract-dict(as1 tree, subset, expiredf, terminf) in
10    start view();
11    start timer(timeinf);
12    start-initial-processes(entitydict);
13    pathd(delayf)(entitydict);
14    handle-inputs(entitydict)))
```

type: *System-definition*$_1$ *Block-identifier*$_1$-**set** *Auxiliary-information* ⇒

Назначение Интерпретировать SDL-систему.

Параметры

as_1tree — AS$_1$ - определение системы.

subset — Выбрано согласованное подмножество.

auxinf — Содержит следующее (см. строку 1):

timeinf — Информация, затребованная процессором таймера. Она содержит функцию, модифицирующую текущее NOW при каждом импульсе сигнала времени в процессоре *таймера*, и стартовое значение времени системы. Домен определен в Приложении F.2 и описывается далее в процессоре *таймера*.

terminf — Замкнутое выражение, содержащее идентификатор AS, Pid-сорта, Pid-литерала null (нуль) и булевские литералы True (Истина) и False (Ложь).

expiredf — Функция, вырабатывающая по истечении времени таймера значение true.

delayf — Функция, вырабатывающая произвольным образом булевское значение. Используется в процессоре-*пути* для моделирования задержки в каналах.

Алгоритм

Строка 2 — Пусть istancemap обозначает отображение экземпляров csp-процессоров в составной домен *Идентификатора-процесса*$_1$ или ВНЕШНЮЮ СРЕДУ и *Pid-значение*. Отображение реализуется как пустое. Первоначально оно используется для маршрутизации сигналов и создания новых экземпляров.

Строка 4 — Пусть queuemap обозначает отображение классов эквивалентности *Pid-значений* во *входной-порт sdl-процессов*. Это отображение реализуется как пустое. Queuemap используется для тех же целей, что и instancemap.

Строка 5 — Пусть pidno обозначает отображение проверки того, что не превышено максимальное количество экземпляров определения-процесса. Отображение реализуется как пустое.

Строка 6 Пусть pathmap обозначает отображение путей с задержками в csp-экземпляры процессора-*пути*. Путь с задержкой — это список каналов, которые проходит экземпляр сигнала при интерпретации выходной-вершины. Необходимо различать возможные пути с задержками при одной и той же последовательности каналов.

Строка 7 Пусть pidset обозначает первоначально пустое множество *Pid-значений*.

Строка 11 Запустить *таймер* с фактическими параметрами для обработки NOW (описывается далее в процессоре *таймера*).

Строка 12 Запустить *sdl-процессы*.

Строка 13 Запустить один экземпляр процессора для каждого коммуникационного пути в системе.

Строка 14 Обработать все другие коммуникации.

start-initial-processes(*entitydict*) ≙ (3.1.2)

```
1  (let pset = {id | (id, PROCESS) ∈ dom entitydict} in
2   for all p ∈ pset do
3   (let mk-ProcessDD(, no, , , ) = entitydict((p, PROCESS)) in
4    (for i = 1 to no do
5     handle-create-instance-request(p, nil, nil)(entitydict))))
```

type: *Entity-dict* ⇒

Назначение Запустить sdl-процессы.

Алгоритм

Строка 3 Пусть *no* обозначает число экземпляров, запускаемых процессом.

Строка 4 Запустить требуемое количество экземпляров.

pathd(*delayf*)(*entitydict*) ≙ (3.1.3)

```
1  (let rs = entitydict(ENVIRONMENT) in
2   for all reach ∈ rs do
3   (let (, , p) = reach in
4    let p' = ⟨p[i] | 1 ≤ i ≤ len p − 1⟩ in
5    (if p' ∉ dom pathmap
6       then (def cspp : start path(delayf);
7             pathmap := c pathmap + [p' ↦ cspp])
8       else I));
9   for all (pd, PROCESS) ∈ dom entitydict do
10  (let mk-ProcessDD(, , , , rs) = entitydict((pd, PROCESS)) in
11   for all reach ∈ rs do
12   (let (d, , p) = reach in
13    let p' = ⟨p[i] | 2 ≤ i ≤ (d = ENVIRONMENT
14                                → len p,
15                              T → len p − 1)⟩ in
16    if len p' > 0 ∧ p ∉ dom pathmap then
17      (def cspp : start path(delayf);
18       pathmap := c pathmap + [p' ↦ cspp])
19     else
20      I)))
```

type: (() ⇒ *Bool*) → *Entity-dict* ⇒

Назначение Запустить экземпляр процессора *пути* для каждой взаимодействующей пары *Идентификатора-процесса*$_1$ и пути в системе. Модифицировать отображение путей в csp-экземпляры.

Параметры

delayf — Функция, вырабатывающая произвольным образом значение *Bool* (Булевский). Используется в процессоре *пути* для моделирования задержки в каналах.

Алгоритм

Строка 1 — Пусть *rs* обозначает *набор-достижимости* (*reachability-set*) (процессов) внешней среды. Эта информация базируется на каналах, ведущих из внешней среды в систему, и извлекается из *entitydict*.

Строки 2—8 — Запустить экземпляр процессора для каждой достижимости в наборе (set); *pcsp* обозначает csp-экземпляр запускаемого процессора.

Строка 4 — Пусть *p'* обозначает путь, вызывающий задержку (то есть исключающий последний элемент, являющийся маршрутом сигналов).

Строка 7 — Модифицировать соответственно pathmap.

Строки 9—18 — Повторить данную схему для исходных процессов в системе.

Строка 13 — Определить часть пути с задержками как начинающуюся со второго элемента, так как первый элемент — это маршрут сигналов, и заканчивающуюся вторым последним элементом if в системе, так как в этом случае последний элемент — это маршрут сигналов.

Строка 16 — Запустить только процессор, если остальной путь — не пустой (вызывает задержку).

handle-inputs(*entitydict*) ≙ (3.1.4)

```
1  (cycle {input mk-Send-Signal(si, vl, r, p) from se
2             ⇒ handle-send-signal(si, vl, r, p, se)(entitydict),
3          input mk-Create-Instance-Request(prid, vl) from se
4             ⇒ handle-create-instance-request(prid, vl, se)(entitydict),
5          input mk-Stop() from se
6             ⇒ handle-stop(se),
7          input mk-Create-Pid(port) from se
8             ⇒ handle-create-from-environment(port, se)(entitydict),
9          input mk-Release-Pid(p) from se
10            ⇒ handle-stops-in-environment(p, se)})
```

type: *Entity-dict* ⇒

Назначение — Обработать всю коммуникацию *системы* после инициализаций.

Алгоритм

Строка 1 — Запускаем на все время цикл. При каждом прохождении этого цикла будет подробно уточняться один из упомянутых входов (на недетерминированной основе). Обработка каждого входа описана в конкретной функции-обработки (handle-function).

handle-stops-in-environment(*p*, *se*) ≙ (3.1.5)

```
1  (def class s.t. p ∈ class ∧ class ∈ dom c queuemap;
2   def q : c queuemap(class);
3   instancemap := c instancemap \ {se};
4   queuemap := c queuemap \ {class};
5   discard-signals-to-port(q))
```

type: *Pid-Value Π* ⇒

Назначение Обработка останова «процессов» во внешней среде путем модификации отображений в системе.

Параметры

p Pid-Значение остановленного «процесса».

se Csp-экземпляр «ОТПРАВИТЕЛЯ» («SENDER»).

Алгоритм

Строки 3—4 Удалить «процесс» и его «входной порт» из отображений действующих экземпляров.

Строка 5 Обработать устранение сигналов к останавливающемуся процессу во внешней среде, окружающей коммуникационные пути.

handle-create-from-environment(*port*, *se*)(*entitydict*) $\triangleq$ (3.1.6)

```
1  (def (pid, pidclass) : getpid(entitydict);
2   instancemap := c instancemap + [se ↦ (ENVIRONMENT, pid)];
3   queuemap := c queuemap + [pidclass ↦ port];
4   output mk-Pid-Created(pid) to se)
```

type: $\Pi\ \Pi \to$ *Entity-dict* $\Rightarrow$

Назначение Обработать процесс создания Pid-Значений во внешней среде. Модифицировать отображения в системе и передать Pid-Значение во внешнюю среду. Коммуникация не соответствует полностью коммуникации обработки вершин-СОЗДАТЬ (CREATE-nodes) в системе. Однако нельзя допускать, что внешняя среда содержит вершины-СОЗДАТЬ (!). Основная цель состоит в том, чтобы при сохраняющейся согласованной модели высказать как можно меньше предположений о внешней среде.

Параметры

port Csp-экземпляр входного порта «отправителя». Предполагается, что внешняя среда имеет входной-порт, так как это способ реализации асинхронной коммуникации.

se Это csp-экземпляр «отправителя».

Алгоритм

Строки 2—3 Модифицировать отображения действующих экземпляров «процессом», связанным с помощью внешней среды.

Строка 4 Возвратить *Pid-Значение* во внешнюю среду.

handle-send-signal(*si*, *vl*, *r*, *p*, *se*)(*entitydict*) ≙ (3.1.7)

```
1  (def (sid, sp) : c instancemap(se);
2   (let re = if is-Identifier1(sid)
3                then s-Reachabilities(entitydict((sid, PROCESS)))
4                else entitydict(ENVIRONMENT) in
5    let re' = {(, s', ) ∈ re | si ∈ s'} in
6    let re'' = if p = {} then re' else {(, ,p') ∈ re' | p ∩ elems p' ≠ {}} in
7    def rp : (r ≠ nil
8                → {(rident, r) ∈ rng c instancemap | (rident, , ) ∈ re''},
9              T → {(rident, ) ∈ rng c instancemap | (rident, , ) ∈ re''});
10   (card (rp) = 0
11      → exit("§2.7.4:Получатель не найден"),
12    card (rp) > 1
13      → exit("§2.7.4: Найдено много получателей"),
14    T → (let {(rident, ri)} = rp in
15         let (rident', , path) ∈ re'' be s.t. rident' = rident in
16         (def class s.t. ri ∈ class ∧ class ∈ dom queuemap;
17          def rcsp : c queuemap(class);
18          (let reduced-path = delaying-path(path, sid, rident) in
19           if reduced-path = ⟨⟩
20             then output mk-Signal-Delivered(si, vl, sp) to rcsp
21             else (def path' : c pathmap(reduced-path);
22                   output mk-Queue-Signal(si, vl, sp, rcsp) to path'))))))))
```

type: *Signal-identifier*$_1$ *Value-List* [*Pid-Value*] *Direct-via*$_1$
Π(*sdl-process*) → *Entity-dict* ⇒

Назначение Маршрутизация сигналов.

Параметры

si	Передаваемый сигнал.
vl	Необязательный список значений, переносимых сигналом.
r	Необязательное *Pid-Значение*, обозначающее получателя (из TO-клаузулы).
p	Необязательный список путей (из VIA-клаузулы).
se	Csp-экземпляр передающего *sdl-процесса*.

Алгоритм

Строка 1	Пусть *sid* и *sp* обозначают *Идентификатор-процесса*$_1$ и *Pid-Значение* отправителя.
Строка 2	Проверить, исходит ли сигнал от внешней среды (строка 4) или от процесса в системе (строка 3). В обоих случаях *re* обозначает *набор-Достижимости* (*Reachability-set*) отправителя. Остающаяся функция последовательно ограничивает достижимость отправителя (до строки 9).
Строка 5	Ограничиться теми достижимостями, которые могут передавать фактический сигнал *si*.
Строка 6—6	Ограничить *p*, образуемое путями из VIA-клаузулы.
Строка 6	Ограничений нет, если VIA-клаузула отсутствует.
Строка 8	Проверить пути от окружающей среды.
Строка 6	Ограничить набор-достижимости теми членами, которые упоминаются членом *p* из VIA-клаузулы в их пути.
Строка 7	Пусть *rp* обозначает множество потенциальных получателей. Членами *rp* являются взаимосвязанные пары (*Process-identifier*$_1$, *Pid-Value*).
Строка 8	Обработать случай, где была дана TO-клаузула. Тогда взаимосвязанная пара должна обозначать действующий процесс, где *rident* — это достижимость.
Строка 9	Обработать случай без TO-клаузулы. В этом случае член *Pid-Value* (*Pid-Значение*) взаимодействующей пары остается неспецифицированным.

Строки 10—14	Проверить количество найденных получателей.
Строка 11	Определить ошибку отсутствующего (ненайденного) (достижимого и действующего) получателя.
Строка 13	Определить ошибку нескольких получателей (индетерминированность Выходной-вершины (OUTPUT-node)).
Строка 14	Указание на успешный результат: *rp* содержит один и только один член.
Строка 15	Выбрать *путь*, ведущий к одному получателю. Этот выбор — недетерминированный, если подканалы, ведущие к одному и тому же процессу, могут переносить один и тот же сигнал.
Строка 16	Пусть *rcsp* обозначает csp-экземпляр входного порта *sdl-процесса*-получателя.
Строка 18	Пусть *сокращенный-путь* (*reduced-path*) обозначает часть *Пути*, вызывающего задержку (каналы).
Строка 19	Если сигнал не проходит по каналам (в пределах одного и того же блока), то для процессора *входного-порта* получателя сигнал является выходным.
Строка 21	Пусть *path'* обозначает csp-экземпляр соответствующего процессора *пути*.
Строка 22	Вывести сигнал к выбранному процессору *пути*.

delaying-path(*path*, *sid*, *rid*) $\triangleq$ **(3.1.8)**

```
1  (len path ≤ 1
2     → ⟨⟩,
3   sid = ENVIRONMENT
4     → ⟨path[i] | 1 ≤ i ≤ len path − 1⟩,
5   rid = ENVIRONMENT
6     → tl path,
7   T → ⟨path[i] | 2 ≤ i ≤ len path − 1⟩)
```

type: *Path* (ENVIRONMENT | *Process-identifier*$_1$)
(ENVIRONMENT | *Process-identifier*$_1$) → *Path*

Назначение Сократить коммуникационный путь до пути-*задержки* (*delaying*-path).

Параметры

path	Весь путь от отправителя до получателя.
sid	Отправитель (тождество).
rid	Получатель (тождество).

Результат Путь задержки.

Алгоритм

Строка 1	Если путь пустой или состоит из одного идентификатора маршрута сигналов, возвратить его немодифицированным.
Строка 3	Если сигнал исходит от внешней среды, удалить идентификатор маршрута сигнала, оканчивающего *Путь*.
Строка 5	Если сигнал передается во внешнюю среду, тогда удалить идентификатор маршрута сигналов, начинающего *Путь*.
Строка 7	Если сигнал передан от одного блока к другому, удалить начальный и конечный идентификаторы маршрутов сигналов.

handle-create-instance-request(*prid*, *vl*, *se*)(*entitydict*) ≙ (3.1.9)

```
 1  (if prid ∉ dom c pidno
 2     then pidno := c pidno + [prid ↦ 0]
 3     else I;
 4   (let mk-ProcessDD(il, , maximum, , ) = entitydict((prid, PROCESS)) in
 5    let vl' = if vl = nil then ⟨nil | 1 ≤ i ≤ len il⟩ else vl in
 6    def parent : if se = nil then nil else s-Pid-Value(c instancemap(se));
 7    def exceed : (maximum = c pidno(prid));
 8    def (newpid, pidclass) : getpid(entitydict);
 9    if ¬exceed then
10      (def csppid : start sdl-process(parent, newpid, vl', prid)(entitydict);
11       (input mk-Process-Initiated(qcsppid) from csppid
12            ⇒  (instancemap := c instancemap + [csppid ↦ (prid, newpid)];
13                queuemap := c queuemap + [pidclass ↦ qcsppid];
14                pidno := c pidno + [prid ↦ c pidno(prid) + 1])))
15     else
16      I;
17    if se ≠ nil
18      then output mk-Create-Instance-Answer(if exceed then nil else newpid) to se
19      else I))
```

type: *Process-identifier*$_1$ [*Value-List*] [*Π*(*sdl-process*)] → *Entity-dict* ⇒

Назначение Оперирование созданием *sdl-процессов*.

Параметры

prid	*Идентификатор-процесса*$_1$ запускаемого процесса.
vl	Необязательный список фактических параметров (=nil при обращении во время инициализации системы).
se	Необязательный родитель (=nil при обращении во время инициализации системы).

Алгоритм

Строка 1	Инициировать отображение для *prid*, равного нулю, с нулем экземпляров.
Строка 6	Пусть *parent* (*родитель*) обозначает родительское значение, переносимое к новому экземпляру.
Строка 8	Создать единственное *Pid-Значение*.
Строка 7	Выполнить проверку превышения максимального числа экземпляров определения процесса.
Строка 10	Самозапуск экземпляра-*sdl-процесса*.
Строка 11	Ожидание подтверждения инициализации от запущенного *sdl-процесса*.
Строка 12	Прибавить *sdl-процесс* к отображению *sdl-процессов*.
Строка 13	Прибавить *входной-порт* к отображению *входных-портов*.
Строка 14	Изменить текущее количество экземпляров для определения процесса.
Строка 18	Направить обратно ответ «ОТПРАВИТЕЛЮ» («SENDER») с *Pid-Значением* нового процесса, если создание было вызвано *вершиной-Создать*$_1$ (*Create-node*$_1$). Если максимальное число экземпляров превышено, то возвращается nil.

getpid(*entitydict*) ≜ (3.1.10)

```
1  (let pidsortid = entitydict(PIDSORT) in
2   let mk-SortDD(tid) = entitydict((pidsortid, SORT)) in
3   let mk-TypeDD(sortmap, ) = entitydict((tid, TYPE)) in
4   let classes = sortmap(pidsortid) in
5   let nullterm = entitydict(NULLVALUE) in
6   def class s.t. class ∈ classes ∧ (nullterm ∉ class) ∧ ¬(∃term ∈ class)(term ∈ c pidset);
7   let pid ∈ class in
8   pidset := c pidset ∪ class;
9   return (pid, class))
```

type: *Entity-dict* ⇒ *Pid-Value Pid-Value-set*

Назначение Выбрать еще не используемое *Pid-Значение*. Операция Unuquel, определенная для Pid-сорта в Рекомендации Z.100, показывает, что существует бесконечное число *Pid-значений*. Значениями для Pid-сорта являются null (нуль), uniquel (null, uniquel (uniquel (null)) и т. д. Множество Pid-термов (значений) содержится в *entitydict*.

Следует отметить, что модель предполагает наличие также в процессоре-системы уникальных (unique) *Pid-Значений* для внешней среды. В противном случае трудно представить, как можно использовать *Pid-Значения* для адресации процессов во внешней среде.

Результат Неиспользуемое *Pid-значение* и класс эквивалентности, к которому оно принадлежит.

Алгоритм

Строка 1 Выбрать *Идентификатор*$_1$ Pid-сорта из *entitydict*.

Строка 2 Выбрать идентификатор типа, определенный на уровне системы.

Строка 3 Выбрать sortmap (отображение сорта), содержащее классы эквивалентности сорта, определенного на уровне системы.

Строка 4 Выбрать классы эквивалентности Pid-сорта. Следует отметить, что для Pid-сорта каждый класс эквивалентности содержит ровно один пассивный-терм.

Строка 5 Выбрать представление AS$_1$ терма NULL (НУЛЬ).

Строка 6 Взять класс эквивалентности, не представленный в pidset и отличающийся от терма NULL.

Строка 7 Взять пассивный-терм в данном классе эквивалентности.

Строка 8 Добавить это новое значение к множеству *Pid-Значений*.

handle-stop(*se*) ≜ (3.1.11)

```
1  (def (prid, p) : c instancemap(se);
2   def class s.t. p ∈ class ∧ class ∈ dom queuemap;
3   def q : c queuemap(class);
4   instancemap := c instancemap \ {se};
5   queuemap := c queuemap \ {class};
6   pidno := c pidno + [prid ↦ c pidno(prid) − 1];
7   discard-signals-to-port(q);
8   output mk-Stop-Queue() to q;
9   output mk-Die(p) to view)
```

type: *П*(*sdl-process*) ⇒

Назначение Оперирование со *СТОП-вершина*$_1$*ми* (*STOP-Node*$_1$*s*).

Параметры

se Сер-экземпляр остановленного *sdl-процесса*.

Алгоритм

Строка 4	Вычесть процесс из отображения действующих экземпляров.
Строка 5	Вычесть соответствующий *входной-порт* из его отображения.
Строка 6	Изменить текущее количество экземпляров *prid*, вычитая остановленный процесс.
Строка 7	Отменить сигналы на путях к остановленному процессу.
Строка 8	Запросить останов *входного-порта*.
Строка 9	Запросить *обозревание* (*view*), чтобы модифицировать отображение раскрытых переменных.

discard-signals-to-port(*q*) ≜ (3.1.12)

```
1  (for all r ∈ rng c pathmap do
2   output mk-Discard-Signals(q) to r)
```

type: *Port* ⇒

Назначение Выход ко всем экземплярам-*пути* с указанием аннулировать экземпляры сигналов, ожидающие передачи к одному *входному-порту*.

Параметры

q	Csp-экземпляр *входного-порта*.

3.2 Процессор Обозревания

view **processor** () ≜ (3.2.1)

```
1  (dcl viewmap := [] type (Pid-Value Variable-identifier1) ⇸ (Value | UNDEFINED);
2   trap exit with error in
3   (cycle {input mk-Reveal(id, value, pid) from sdl-process
4              ⇒  viewmap := c viewmap + [(pid, id) ↦ value],
5           input mk-View-Request(id, revealpid) from viewpid
6              ⇒  (let entry = (revealpid, id) in
7                  if entry ∈ dom c viewmap
8                     then output mk-View-Answer(c viewmap(entry)) to viewpid
9                     else exit("§5.5.4.4: Процесс раскрытия не является действующим")),
10          input mk-Die(pid) from system
11             ⇒  (for all (pid, id) ∈ dom c viewmap do
12                 viewmap := c viewmap \ {(pid, id)})}))
```

type: () ⇒

Назначение Интерпретация понятий VIEW (ОБОЗРЕВАНИЕ) и REVEAL (РАСКРЫТИЕ).

Алгоритм

Строка 1	Пусть *viewmap* обозначает отображение пары *Pid-Значение* и *Идентификатор-Переменной*$_1$ в раскрытое *Значение*.
Строка 3	Обработать *Reveal*-вход (вход-*Раскрытия*).
Строка 4	Модифицировать отображение новым входом.
Строка 5	Обработать VIEW из *sdl-процесса*.
Строка 8	Возвратить значение *sdl-процессу*.
Строка 9	Определить ошибку нераскрываемой переменной.
Строка 10	Обработать сообщение остановленного *sdl-процесса*.
Строка 11	Вычесть все раскрытые переменные остановленного процесса из отображения.

3.3 Процессор Пути

path **processor** (*delayf*) $\triangleq$ (3.3.1)

```
1  (dcl pqueue := ⟨⟩ type (Signal-identifier1 Value-List Pid-Value Port)*;
2   cycle {input mk-Queue-Signal(si, vl, sp, rcsp) from system
3              ⇒  (pqueue := c pqueue ↷ ((si, vl, sp, rcsp))),
4          input mk-Discard-Signals(q) from system
5              ⇒  (pqueue := (c pqueue[i] | 1 ≤ i ≤ len c pqueue ∧
6                                          (def(_,_,r) : c pqueue[i];
7                                           return r ≠ q))),
8          (delayf() ∧ c pqueue ≠ ⟨⟩)
9          (output mk-Signal-Delivered(s-Signal-identifier1(hd c pqueue),
10                                     s-Value-List(hd c pqueue),
11                                     s-Pid-Value(hd c pqueue)) to s-Port(hd c pqueue))
12           ⇒ (pqueue := tl c pqueue)})
```

type: (() ⇒ *Bool*) ⇒

Назначение Интерпретация потенциальной задержки в пути. Экземпляр существует для каждого значения пути в наборе-*Достижимостей* (*Reachability*-set) в *ProcessDD*.

Параметры

delayf Функция, произвольным образом вырабатывающая значение Булевский. Используется для моделирования задержки в каналах.

Алгоритм

Строка 3 Ввести сигнал в очередь пути.

Строка 4 Обработать удаление сигналов, направленных в конкретный *входной-порт q*. Используется, когда останавливается *sdl-процесс входного-порта*.

Строка 5 Пусть новая очередь pqueue равна старой очереди, за исключением направляемых в *q* элементов.

Строка 8 Данная клаузула моделирует недетерминированную задержку в пути. Выход (**output**) защищен предикатом: он имеет место только в том случае, если *delayf* вырабатывает true, а очередь pqueue не пуста. Конкретный синтаксис таков:

```
(<предикат>)(<коммуникационное событие>)
    => <оператор>
```

Индетерминированность выражается в терминах императивной функции *delayf*, определение которой не входит в данное формальное определение. Если предикат имеется, то сигнал выдается экземпляру *входного-порта*.

Строка 12 Удалить выходной сигнал из очереди.

3.4 Процессор Входного-Порта

Данный процессор реализует неограниченные буферы *sdl-процессов* и таймеров. Неограниченный буфер отображается в процессоре в качестве очереди переменных.

input-port **processor** (*ppid*, *expiredf*) ≙ (3.4.1)

```
1  (dcl queue := ⟨⟩ type (Signal-identifier1 Value-List Pid-Value)*;
2   dcl waiting := false type Bool;
3   dcl pendingset := {} type Signal-identifier1-set;
4   dcl timers := [] type (Timer-identifier1 Arglist) ➘(Π(sdl-process) Value Equivalent-test);
5   cycle {input mk-Signal-Delivered(sid, vl, se) from p
6              ⇒  handle-queue-insert(sid, vl, se, p),
7           input mk-Next-Signal(saveset) from p
8              ⇒  handle-queue-extract(saveset, ⟨⟩, c queue, p),
9           input mk-Stop-Queue() from p
10             ⇒  stop,
11          input mk-Set-Timer(tid, v, al, et) from p
12             ⇒  handle-set-timer(tid, v, al, et, p),
13          input mk-Reset-Timer(tid, al, et) from p
14             ⇒  handle-reset-timer(tid, al, et),
15          input mk-Active-Request(tid, al, et) from p
16             ⇒  handle-active-request(tid, al, et, p),
17          (output mk-Time-Request() to timer;
18           handle-time-request(ppid, expiredf))})
```

type: *Pid-Value Is-expired* ⇒

Назначение Интерпретация входного-порта *sdl-процесса*. Экземпляр существует для каждого экземпляра *sdl-процесса*.

Параметры

pcsp — Pid-значение обслуживаемого *sdl-процесса*.

expiredf — Функция, вырабатывающая значение True, если время данного таймера истекло.

Алгоритм

Строка 1 — Пусть queue (очередь) обозначает неограниченный буфер *sdl-процесса*. Каждый вход содержит *Идентификатор-сигнала*$_1$, возможно, пустой список *Значений* и *Pid-Значение*, обозначающее «ОТПРАВИТЕЛЯ» («SENDER»).

Строка 2 — Пусть waiting (ожидание) обозначает, ожидает ли *sdl-процесс* ответ после запроса *Следующего-Сигнала*, на который ответить сразу нельзя, так как очередь была пустой или же потому, что все сигналы, находящиеся в очереди, были элементами параметра saveset (сохраняемый-набор). В случае ожидания запроса pendingset (см. ниже) содержит saveset ожидания запроса.

Строка 3 — Пусть pendingset обозначает saveset, связанный с ожиданием запроса от *sdl-процесса*, как определено в состоянии ожидания.

Строка 4 — Пусть таймеры обозначают отображение для действующих таймеров. *Π (sdl-process)* отображения обозначает *sdl-процесс*, устанавливающий таймер. *Значение* отображения обозначает время, которое истекает, и оно равно нулю после истечения (то есть, когда время в очереди). Для сравнения значений из *Arglists* (например, для сравнения элемента из отображения таймеров с сигналом в очереди в функции *handle-queue-extract* (*обработка-выбора-из-очереди*)) используется *Тест-эквивалентности* (*Equivalence-test*). *Значение* содержит истекаемое время. Таймер вычитается из отображения, когда он возвращается в *sdl-процесс* в качестве сигнала.

Строка 5 — Это вход основного цикла *входного-порта*.

Строка 7 Примечание: на этот ввод нельзя ответить немедленно. Причиной ввода ожидания для переменных и установки-ожидания (pendingset) является конструкция SAVE (СОХРАНИТЬ). В случае структуры неизменяемой очереди может использоваться защита-входа, чтобы исключить коммуникацию *Следующего-Сигнала* в случае пустой очереди.

Строка 17 Ввести один выход в данную схему. Это повторный запрос фактического времени от *таймера*.

handle-queue-insert(*sid*, *vl*, *se*, *pcsp*) ≙ (3.4.2)

```
1  (queue := c queue ⌢ ⟨(sid, vl, se)⟩;
2   if ¬waiting then
3      handle-queue-extract(c pendingset, ⟨⟩, c queue, pcsp)
4   else
5   I)
```

type: *Signal-identifier*, *Value-List Pid-Value* Π(*sdl-process*) ⇒

Назначение Ввести сигнал в очередь.

Параметры

sid Вводимый сигнал.

vl Необязательный список его значений.

se Отправитель.

pcsp CSP-экземпляр обслуживаемого *sdl-процесса*.

Алгоритм

Строка 1 Конкатенация сигнала с очередью.

Строки 2—3 Проверить, не ждет ли *Следующий-Сигнал*. Если ждет, то извлечь элемент из очереди. Это может привести к *Вводу-Сигнала* в *sdl-процесс*.

handle-queue-extract(*saveset*, *qf*, *qa*, *pcsp*) ≙ (3.4.3)

```
1   (if qa ≠ ⟨⟩ then
2      (let s = hd qa in
3       let (sid, vl, se) = s in
4       (if sid ∈ saveset
5          then handle-queue-extract(saveset, qf ⌢ ⟨s⟩, tl qa, pcsp)
6          else (output mk-Input-Signal(sid, vl, se) to pcsp;
7                queue := qf ⌢ tl qa;
8                waiting := false;
9                if (sid, ) ∈ dom c timers then
10                 if (∃a, et)((sid, a) ∈ dom c timers ∧
11                                (, , et) = c timers(sid, a) ∧
12                                same-argument-values(a, vl, et)) then
13                   (def (a, et) s.t. (sid, a) ∈ dom c timers ∧
14                        (, , et) = c timers(sid, a) ∧ same-argument-values(a, vl, et);
15                    timers := c timers \ {(sid, a)})
16                 else
17                  I
18               else
19                I)))
20     else
21      (pendingset := saveset;
22       waiting := true))
```

type: *Signal-identifier*$_1$-set
(*Signal-identifier*$_1$ *Value-List Pid-Value*)*
(*Signal-identifier*$_1$ *Value-List Pid-Value*)* *Π*(*sdl-process*) ⇒

Назначение Извлечь один элемент из очереди и послать его *sdl-процессу*, если *sdl-процесс* готов его принять.

Параметры

saveset	Набор сигналов, которые в данной ситуации не извлекаются из очереди.
qf	Часть очереди, уже проверенная.
qa	Часть очереди, требующая проверки.
pcsp	CSP-экземпляр обслуживаемого *sdl-процесса*.

Алгоритм

Строка 1	Останов безуспешного извлечения, если *qa* пуста.
Строка 2	В противном случае взять первый элемент из *qa* и
Строка 3	произвести декомпозицию очереди на *Идентификатор-сигнала*$_1$, список *Значений* и «ОТПРАВИТЕЛЯ».
Строка 5	Если сигнал находится в *saveset* (*сохраняемый-набор*), тогда произвести его конкатенацию с проверенной очередью и повторить поиск в остальной части *qa*.
Строка 6	Вывести *Следующий-Сигнал* в *sdl-процесс*, если *si* не находится в *сохраняемом-наборе*. Модифицировать очередь путем конкатенации *qf* и остальной части *qa*. Установить флаг для нежудущих запросов и в итоге
Строка 9	изменить отображение таймеров, если выбранный сигнал был таймером.
Строка 13	Удаляемый таймер должен иметь тот же идентификатор, что и сигнал *si*, а сравнение с *et* должно привести к заключению, что список аргументов из таймера эквивалентен списку аргументов из отображения таймеров.
Строка 21	В случае неуспеха пометить ждущий запрос с фактическим сохраняемым набором.

same-argument-values$(a, vl, et) \triangleq$ (3.4.4)

```
1  (len a = len vl ∧
2   (∀i ∈ Ind a)(et(a[i], vl[i])))
```

type: *Arglist Arglist Equivalent-test* → *Bool*

Назначение Проверить эквивалентность двух списков *Терм*$_1$*ов* (Term$_1$s) согласно определению функции *et*.

Параметры

a	Один список для проверки.
vl	Другой список.
et	Функция теста-эквивалентности.

Алгоритм

Строка 1	Длина двух списков должна быть одинаковой.
Строка 2	Для каждого индекса тест должен быть успешным.

handle-set-timer$(tid, v, al, et, p) \triangleq$ (3.4.5)

```
1  (handle-reset-timer(tid, al, et);
2   timers := c timers + [(tid, al) ↦ (p, v, et)])
```

type: *Timer-identifier*$_1$ *Value Arglist Equivalent-test* Π(*sdl-process*) ⇒

Назначение Установить таймер, модифицируя отображения таймеров.

Параметры

tid	Идентификатор таймера.
v	Время истечения действия таймера.
al	Список аргументов таймера.
et	Соответствующая функция теста-эквивалентности, которая может применяться к каждому элементу списка аргументов.
p	*sdl-процесс*, устанавливающий таймер.

Алгоритм

Строка 1	Сброс возможно существующего таймера с тем же идентификатором и списком-аргументов.
Строка 2	Модифицировать отображение таймеров.

handle-reset-timer$(tid, al, et) \triangleq$ (3.4.6)

```
 1  (for all (t, a) ∈ dom c timers do
 2   (if (∃a)((tid, a) ∈ dom c timers ∧
 3            same-argument-values(a, al, et)) then
 4      (def (, e, ) : c timers(tid, al);
 5       (timers := c timers \ {(t, a)};
 6        if c e = nil
 7          then (handle-remove-timer-from-queue(tid, al, et, (), c queue))
 8          else I))
 9      else
10       I))
```

type: *Timer-identifier*$_1$ *Arglist Equivalent-test* ⇒

Назначение Сброс таймера модификацией отображения таймеров и очереди.

Параметры

tid Идентификатор таймера.

al Список аргументов таймера.

et Соответствующая функция теста-эквивалентности, которая может применяться к каждому элементу списка аргументов.

Алгоритм

Строка 2 Выбрать соответствующий таймер с таким списком аргументов, в котором (*tid*, *a*) является доменом для отображения таймеров и *a* соответствует *al* при применении теста-эквивалентности *et*.

Строка 5 Вычесть (*tid*, *a*) из отображения таймеров.

Строка 7 Удалить *tid* из очереди, если он здесь имеется (отмечен в поле *Значения* диапазона для отображения таймеров).

handle-remove-timer-from-queue(*sid*, *al*, *et*, *qf*, *qa*) ≙ (3.4.7)

```
1  (let (si, vl, ) = hd qa in
2   if si = sid ∧ same-argument-values(vl, al, et)
3     then queue := qf ⌢ tl qa
4     else handle-remove-timer-from-queue(sid, al, et, qf ⌢ ⟨hd qa⟩, tl qa))
```

type: *Signal-identifier*$_1$ *Arglist Equivalent-test*
(*Signal-identifier*$_1$ [*Value**] *Pid-Value*)*
(*Signal-identifier*$_1$ [*Value**] *Pid-Value*)* ⇒

Назначение Удалить один элемент из очереди.

Параметры

sid Удаляемый сигнал.

al Список аргументов таймера.

et Соответствующая функция теста-эквивалентности, которая может применяться к каждому элементу списка аргументов.

qf Проверенная часть очереди.

qa Часть очереди, требующая проверки.

Алгоритм

Строка 1 Пусть *si* обозначает *Идентификатор-сигнала*$_1$ первого элемента в *qa*.

Строка 3 Если *si* является удаляемым сигналом, тогда модифицировать очередь, часть которой *qf* подвергается конкатенации с остальной частью *qa*, в противном случае

Строка 4 продолжить поиск остальной части *qa*. Следует отметить, что *sid* должен всегда присутствовать в очереди в самом последнем случае обращения к функции, так что не требуется тестирования окончания рекурсии!

handle-active-request(*tid*, *al*, *et*, *pcsp*) ≙ (3.4.8)

```
1  (def stat : (∃a)((tid, a) ∈ dom c timers ∧
2                   same-argument-values(al, a, et));
3   output mk-Active-Answer(stat) to pcsp)
```

type: *Timer-identifier*$_1$ *Arglist Equivalent-test* Π(*sdl-process*) ⇒

Назначение Дать ответ на ACTIVE (АКТИВНЫЙ) на основе отображения таймеров.

Параметры

tid	Идентификатор таймера.
al	Список аргументов таймера.
et	Соответствующая функция теста-эквивалентности, которая может применяться к каждому элементу списка аргументов.
pcsp	CSP-экземпляр обслуживаемого *sdl-процесса*.

Алгоритм

Строка 1	Пусть *stat* означает true, если специфицированный таймер находится в домене отображения таймеров; в противном случае *stat* означает false.
Строка 3	Использовать данное значение в качестве параметра выхода к *sdl-процессу*.

handle-time-request(*ppid*, *expiredf*) $\triangleq$ (3.4.9)

```
1  (input mk-Time-Answer(t) from timer
2      ⇒  (for all (tid, al) ∈ dom c timers do
3           (def (p, expt, et) : c timers(tid, al);
4            if expt ≠ nil ∧ expiredf(expt, t) then
5              (timers := c timers + [(tid, al) ↦ (p, nil, et)];
6               handle-queue-insert(tid, al, ppid, p))
7            else
8             I)))
```

type: *Value Is-expired* ⇒

Назначение Сравнение с реальным временем, проводимое для всех установленных таймеров.

Параметры

ppid	PId-значение обслуживаемого *sdl-процесса*.
expiredf	Функция (конструкция из Приложения F.2), вырабатывающая True, по истечении времени действия данного таймера.

Алгоритм

Строка 1	Получение реального времени от *таймера* в *t*.
Строка 2	Запуск проверки множества таймеров.
Строка 4	Проверить, находится ли таймер в очереди или же время его действия истекло.
Строка 5	В этом случае изменить отображение таймеров, находящихся «в очереди».
Строка 6	Ввести таймер в очередь, при этом для обслуживаемого *sdl-процесса* «SENDER» равен «SELF».

3.5 Процессор Таймера

Данный процессор введен в SDL для интерпретации понятия всемирного времени. Это выражается в очень простой связи с внешним процессором *импульсов сигналов времени* (*tick*-процессором).

timer **processor** (*timeinf*) ≙ (3.5.1)

```
1  (let (timef, startt) = timeinf in
2   dcl time-now := startt type Value;
3   cycle {input mk-Time() from tick
4              ⇒  time-now := timef(c time-now),
5          input mk-Time-Request() from p
6              ⇒  output mk-Time-Answer(c time-now) to p})
```

type: *Time-information* ⇒

Назначение Интерпретация обработки таймера в основной системе.

Параметры Объект *timeinf* содержит две компоненты (строка 1) из Приложения F.2:

timef Функция, вызываемая из внешней среды при каждом импульсе сигнала времени («tick»). Функция *timef* решает две задачи: интерпретирует «+» сорта Временной и анализирует временные значения в системе (то есть значение увеличения в NOW для каждого импульса сигнала времени).

startt Начальное значение NOW.

Алгоритм

Строка 2 Пусть time-now обозначает (только одно) всемирное время системы. Для корректного описания понятия времени в SDL используется модель, содержащая стартовое время интерпретации (*startt*) и изменение (функция *timef*) времени.

Строка 4 Изменить время.

Строка 6 Возвратить NOW.

3.6 Неформальный Процессор Импульсов сигналов времени

tick **processor** () ≙ (3.6.1)

```
1  (cycle (output mk-Time() to timer;
2          /* неформально моделирует интервал между последовательными импульсами (ticks)
           сигналов времени */))
```

type: () ⇒

4 SDL-Процесс

В данном разделе описана интерпретация экземпляра SDL-процесса с помощью *sdl-процесса* процессора META-IV. Определение SDL-процесса содержится в entity-dict. Управление всеми связями между процессами и другими связями осуществляет основная система.

Каждый экземпляр SDL-процесса имеет индивидуальную память следующего типа:

1 *Stg* = $Identifier_1 \xrightarrow{m} (Value \mid \mathsf{UNDEFINED})$

4.1 sdl-процесс

sdl-процесс процессора META-IV создается *системой* процессоров и ее фактическими параметрами, что дает возможность иметь представление о ее окружении, о самой системе и о SDL-процессе, который должен ею интерпретироваться. Если SDL-процесс интерпретирован, экземпляры *sdl-процесса* прекращают свое существование.

sdl-process **processor** (*parentp*, *selfp*, *actparml*, *process-id*)(*dict*) $\triangleq$ (4.1.1)

```
 1  (let mk-Identifier1(qual, nm) = process-id in
 2   let nullterm = dict(NULLVALUE) in
 3   def dict1 : dict + [SCOPEUNIT ↦ qual ⌢ ⟨mk-Process-qualifier1(nm)⟩] +
 4       [PORT ↦ start input-port(selfp, dict(EXPIREDF))] +
 5       [SELF ↦ selfp] +
 6       [PARENT ↦ (parentp = nil
 7                          → nullterm,
 8                      T → parentp)];
 9   dcl sender := nullterm type Pid-Value;
10   dcl offspring := nullterm type Pid-Value;
11   dcl stg := [] type Stg;
12   (trap exit() with error in
13    (trap exit(STOP) with output mk-Stop() to system in
14     (let mk-ProcessDD(formparml, , , graph, ) = dict1((process-id, PROCESS)) in
15      (def dict2 : dict1 + [(id, VALUE) ↦ mk-VarDD(id, sort, rev, stg) |
16                            (id, VALUE) ∈ dom dict1 ∧ is-VarDD(dict1((id, VALUE)))∧
17                            s-Qualifier1(id) = dict1(SCOPEUNIT) ∧
18                            mk-VarDD(, sort, rev, ) = dict1((id, VALUE))];
19       init-process-decls(dict2);
20       init-process-parms(formparml, actparml)(dict2);
21       output mk-Process-Initiated(dict2(PORT)) to system;
22       int-process-graph(graph)(dict2))))))
```

type: [*Pid-Value*] *Pid-Value Value-List* $Process\text{-}identifier_1 \rightarrow Entity\text{-}dict \Rightarrow$

Назначение Интерпретация sdl-процесса.

Параметры

parentp	PID-значение процесса в SDL, который создал данный процесс.
selfp	PID-значение данного процесса в SDL.
actparml	Список фактических параметров.
process-id	SDL-идентификатор данного процесса.

Алгоритм

Строка 2	Выбрать PID-значение NULL версии AS_1.

Строки 3—6	Выполнить приращение для dict таким образом, чтобы:
Строка 3	SCOPEUNIT обозначала текущую структурную единицу. SCOPEUNIT изменяется всякий раз, когда интерпретация SDL-процесса касается новой структурной единицы.
Строка 4	PORT обозначает CSP-имя входного порта. Фактические параметры для создания процессора *входного-порта* представляют PID-значение данного процесса в SDL и функцию тестирования истечения времени действия таймера. Входной порт — это процесс, обрабатывающий посылаемые процессу сигналы и оперирующий таймерами.
Строка 5	SELF обозначает PID-значение самого процесса в SDL.
Строка 6	PARENT обозначает PID-значение в SDL родительского процесса, если такой имеется. Если процесс создан при инициализации системы, то родительский процесс отсутствует.
Строки 9—10	Объявление переменных sender и offspring (потомок) с их инициализацией *null-term* (*нуль-термом*).
Строка 11	Объявление переменной stg, которая должна представлять индивидуальную память для данного sdl-процесса и которая в начальный момент пуста.
Строка 12	Ловушка для любого выхода по ошибке.
Строка 13	Ловит (обрабатывает) exit (STOP), посылая в систему сигнал останова, и совершает его.
Строки 15—18	*dict* меняется таким образом, чтобы для каждой локальной переменной имелась ссылка на индивидуальную память. Объявленная переменная и формальные параметры рассматриваются как локальные переменные.
Строка 19	Увеличить объем памяти согласно определениям sdl-процесса.
Строка 20	Увеличить объем памяти согласно содержимому фактических параметров.
Строка 21	Процесс инициирован и процессор *системы* информирован об этом, а также ему известно CSP-имя процессора *входного-порта* в результате выдачи в *систему* сигнала *Process-Initiated* (*Процесс-Инициирован*).
Строка 22	Интерпретация sdl-процесса.

$init\text{-}process\text{-}decls(dict) \triangleq$ (4.1.2)

```
1  (for all (id, VALUE) ∈ dom dict do
2   if is-VarDD(dict((id, VALUE))) ∧ s-Qualifier1(id) = dict(SCOPEUNIT)
3     then update-stg(id, nil)(dict)
4     else I)
```

type: *Entity-dict* ⇒

Назначение Модификация памяти объявлениями переменных, связанных с интерпретируемым sdl-процессом.

Алгоритм

Строка 1	Для всех идентификаторов с атрибутом VALUE, которые
Строка 2	являются дескрипторами переменных и объявлены в данном процессе,
Строка 3	память инициируется с помощью nil. Дополнительное инициирование, касающееся объявления переменных, преобразуется в AS_1 в присваивания в работе, предваряющей стартовый переход.

init-process-parms(*formparml*, *actparml*)(*dict*) ≜ (4.1.3)

```
1  (for all i ∈ Ind formparml do
2   update-stg(formparml[i], actparml[i])(dict))
```

type: *ParameterDD** *Value-List* → *Entity-dict* ⇒

Назначение Модификация индивидуальной памяти процесса формальными параметрами процесса и дополнительно используемыми фактическими параметрами.

Параметры

formparml	Список формальных параметров.
actparml	Список фактических параметров.

Алгоритм

Строки 1—2	В индивидуальную память заносится переменная, обозначенная каждым формальным параметром, и значение связанного с ней фактического параметра. Проверка диапазона значений осуществляет *update-stg*.

4.2 Интерпретация графа-процесса

В данном разделе описывается интерпретация графа-процесса, при которой функции интерпретации выполняются отдельно для вершины-графа каждого типа.

int-process-graph(*graph*)(*dict*) ≜ (4.2.1)

```
1  (let mk-Process-graph1(mk-Process-start-node1(trans), stateset) = graph in
2   (tixe [statenm ↦ int-state-node(statenode)(dict) |
3          statenode ∈ stateset ∧ s-State-name1(statenode) = statenm] in
4    (let mk-Transition1(nodel, termordec) = trans in
5     int-transition(nodel, termordec)(dict))))
```

type: *Process-graph*$_1$ → *Entity-dict* ⇒

Назначение Интерпретация графа-процесса в SDL.

Параметры

graph	Граф процесса.

Алгоритм

Строка 1	Разбиение графа на начальный переход и множество состояний.
Строка 2	Ловит (обрабатывает) все exit (имя-состояния) из *int-state-node* и *int-transition*, интерпретируя связанный с ними *State-node*$_1$. Конструкция tixe очень удобна для моделирования «goto», используемого в вершинах следующих состояний. За ключевым словом tixe следует отображение имен состояний в вызов int-state-node с вершиной-состояния, связанной с именем состояния в качестве фактического параметра. Если exit (statenm$_1$) встречается в структуре конструкции-tixe (то есть либо в области отображения tixe (int-state-node), либо в int-transition), то интерпретация процесса продолжается, и интерпретируется вершина-состояния с именем statenm$_1$.
Строка 4	Интерпретация начального-перехода.

int-state-node(mk-*State-node*$_1$(, mk-*Save-signalset*$_1$(*saveset*), *inputset*))(*dict*) ≜ (4.2.2)

```
 1  (output mk-Next-Signal(saveset) to dict(PORT);
 2   input mk-Input-Signal(sid, actparml, sender') from dict(PORT)
 3      ⇒  (sender := sender';
 4          (let {inpnode} = {inp ∈ inputset | s-Signal-identifier1(inp) = sid} in
 5           let mk-Input-node1(, formparml, trans) = inpnode in
 6           for all i ∈ ind formparml do
 7           (if formparml[i] ≠ nil
 8              then update-stg(formparml, actparml[i])(dict)
 9              else I);
10           (let mk-Transition1(nodel, termordec) = trans in
11            int-transition(nodel, termordec)(dict)))))
```

type: *State-node*$_1$ → *Entity-dict* ⇒

Назначение Запрос от входного порта нового сигнала, прием сигнала и интерпретация соответствующего перехода.

Параметры

state-node Состоит из *saveset*, являющегося сохраняемым входным-портом сигнала, и *inputset*, являющегося совокупностью сигналов и соответствующих переходов.

Алгоритм

Строка 1 Запросить входной-порт выдать сигнал, не находящийся в *saveset*, и сохранить все сигналы, находящиеся в saveset.

Строка 2 Получить сигнал, состоящий из идентификатора-сигнала, списка фактических параметров и Pid-значения (в SDL) отправителя.

Строка 3 У отправителя переменной процесса имеет место коррекция на значение отправителя только что полученного сигнала.

Строка 4 Выбрать входную вершину, имеющую тот же идентификатор сигнала, что и у полученного сигнала.

Строка 5 Разложить выбранный вход на список формальных параметров сигнала и соответствующий переход.

Строки 6—9 Для всех формальных параметров выполнить следующее: если формальный параметр (отличный от nil)имеется, то занести в память соответствующую переменную и значение фактического параметра.

Строка 11 Интерпретировать выбранный переход.

int-transition(*nodel*, *termordec*)(*dict*) ≜ (4.2.3)

```
 1  (if nodel = ⟨⟩
 2     then cases termordec:
 3             (mk-Nextstate-node1(nm)
 4                → exit(nm),
 5              mk-Stop-node1()
 6                → exit(STOP),
 7              mk-Return-node1()
 8                → exit(RETURN),
 9              mk-Decision-node1(,,)
10                → int-decision-node(termordec)(dict))
11     else (int-graph-node(hd nodel)(dict);
12           int-transition(tl nodel, termordec)(dict)))
```

type: *Graph-node*$_1$* (*Terminator*$_1$ | *Decision-node*$_1$) → *Entity-dict* ⇒

Назначение Интерпретация перехода.

Параметры

nodel Список неинтерпретированных вершин графа.

termordec Вершина-терминатор или вершина-принятия-решения.

Алгоритм

Строка 2 Если список вершин пуст, то интерпретируется *termordec*.

Строка 3 Вершина следующего состояния интерпретируется выходом с именем следующего состояния.

Строка 5 Стоп-вершина при выходе с STOP.

Строка 7 Вершина-принятия-решения при выходе с RETURN.

Строка 9 Вершина-принятия-решения при обращении к функции *int-decision-node*.

Строка 10 Если список вершин не пуст, то первая вершина интерпретируется функцией *int-graph-node*.

Строка 11 Остальная часть перехода интерпретируется рекурсивно.

$$int\text{-}decision\text{-}node(\text{mk-}Decision\text{-}node_1(quest, answset, elseansw))(dict) \triangleq \quad (4.2.4)$$

```
1  (let answset' = matching-answer(quest, answset)(dict) in
2   (if answset' ≠ {}
3      then (let {mk-Decision-answer1(, trans)} = answset' in
4              let mk-Transition1(nodel, termordec) = trans in
5              int-transition(nodel, termordec)(dict))
6      else (elseansw ≠ nil
7              → (let mk-Else-answer1(trans) = elseansw in
8                  let mk-Transition1(nodel, termordec) = trans in
9                  int-transition(nodel, termordec)(dict)),
10            T → exit("§2.7.5: Нет сопоставимого ответа")))
```

type: $Decision\text{-}node_1 \rightarrow Entity\text{-}dict \Rightarrow$

Назначение Интерпретировать вершину-принятия-решения на основе вопроса по выбору ответа из множества ответов и интерпретации соответствующего перехода. В случае отсутствия сопоставимого ответа в множестве ответов интерпретируется else-переход. Если нет сопоставимого ответа на вопрос и else-ответа, имеет место ошибка.

Параметры

quest Вопрос принятия решения.

answset Множество ответов и соответствующих переходов.

elseansw Необязательный else-переход.

Алгоритм

Строка 1 Выбор множества сопоставимых ответов при обращении к *matching-answer*.

Строки 2—5 Если выбранное множество ответов непустое, тогда оно содержит только один ответ (в статической семантике проверяется отсутствие перекрытий ответов) и интерпретируется переход, связанный с выбранным ответом.

Строки 6—9 Если сопоставимых ответов не найдено, то интерпретируется переход, связанный с else-ответом.

Строка 10 Если сопоставимых ответов не найдено и нет else-ответа, то имеет место ошибка.

```
matching-answer(quest, answset)(dict) ≜                                    (4.2.5)
 1  (let gterm = dict(TRUEVALUE) in
 2   {mk-Decision-answer1(valsetortext,) ∈ answset |
 3    (is-Range-condition1(valsetortext) ∧ is-Expression1(quest)
 4       → (let mk-Range-condition1(orid, cset) = valsetortext in
 5          let operator1 = make-valuetest-operator(quest, orid, cset) in
 6          is-equivalent((trap exit() with false in
 7                         eval-expression(operator1)(dict)), gterm, dict(SCOPEUNIT))(dict)),
 8     T → text-equality(quest, valsetortext))})
```

type: *Decision-question*$_1$ *Decision-answer*$_1$-**set** → *Entity-dict* → *Decision-answer*$_1$-**set**

Назначение Найти множество ответов из данного множества ответов, которое можно сопоставить с данным вопросом.

Параметры

quest Вопрос принятия решения.

answset Множество ответов и соответствующих переходов.

Результат Сопоставимый ответ и связанный с ним переход.

Алгоритм

Строка 1 Выбрать версию AS_1 литерала TRUE версии AS_0.

Строки 2—8 Построить множество ответов из *answset*, выбранных предикатами в строках 3—8.

Строки 3—6 Если ответ или вопрос не являются неформальными, то производится операция тестирования значения с целью проверки сопоставимости или отсутствия ее для вопроса и ответа.

Строка 8 Если ответ или вопрос являются неформальными, то проверяется совпадение с помощью text-equality (текстовое-совпадение).

```
make-valuetest-operator(exp1, orid, cset) ≜                               (4.2.6)
 1  (let v ∈ cset in
 2   let op = cases v:
 3            (mk-Closed-range1(aop, c1, c2)
 4               → (let mk-Open-range1(relop, co1) = c1 in
 5                  let t1 = mk-Operator-application1(relop, ⟨co1, exp1⟩),
 6                      t2 = make-valuetest-operator(exp1, orid, {c2}) in
 7                  mk-Operator-application1(aop, ⟨t1, t2⟩)),
 8             mk-Open-range1(relop, c1)
 9               → mk-Operator-application1(relop, ⟨exp1, c1⟩)) in
10   if card cset = 1 then
11     op
12    else
13     (let op' = make-valuetest-operator(exp1, orid, cset − {v}) in
14      mk-Operator-application1(orid, ⟨op, op'⟩)))
```

type: *Expression*$_1$ *Operator-identifier*$_1$ *Condition*$_1$-**set** → *Expression*$_1$

Назначение Произвести операцию, которая может проверить, выполняет ли выражение exp_1 условия, налагаемые идентифицированной операцией и *orid*, а также множество условий на диапазон (*cset*).

Параметры

exp_1 Тестируемое выражение.

orid — Идентифицирует операцию, используемую для наложения условия на *cset*.

cset — Множество условий на диапазон, которые должны выполняться для выражения exp_1.

Результат — Операция, с помощью которой можно проверить, сопоставимо ли значение выражения exp_1 с условиями, определяемыми операцией, идентифицированной в *orid* и *cset*.

Алгоритм

Строка 1 — Выбрать условие на диапазон, *v*, в *cset*.

Строка 3 — Если *v* — это *Closed-range*$_1$ (*Замкнутый-диапазон*$_1$), то он разлагается на операцию «и» (*aop*) и два условия c_1 и c_2 *Open-range*$_1$ (*Открытого-диапазона*$_1$).

Строка 5 — Пусть t_1 обозначает операцию проверки значений для c_1.

Строка 6 — Пусть t_2 обозначает операцию проверки значений для c_2.

Строка 7 — Образовать *Operator-application*$_1$ (*Применение-операции*$_1$), при котором применяются t_1 и t_2 по отношению к *aop*, и пусть *op* обозначает это применение операции, то есть образовать операцию «AND» («<=» (col, expl), t2).

Строка 8 — Образовать *Operator-application*$_1$ (*Применение-операции*$_1$), при котором применяются exp_1 и c_1 по отношению к *relop*, и назовем его *v*, то есть образовать операцию «<=» (expl, c1).

Строка 11 — Если *v* — последний элемент в *cset*, то *op* возвращается.

Строка 13 — Произвести операцию проверки значений для остальной части *cset* и назвать ее *op'*.

Строка 14 — Образовать *Operator-application*$_1$ где *orid* применяется по отношению к применениям операций *op* и *op'*.

text-equality(*exp-text*, *valueset-text*) ≜ (4.2.7)

```
1  (/* Этот неформальный Meta-IV-текст обозначает текст совпадения */;
2  /* между неформальным вопросом и/или неформальным ответом */)
```

type: (*Informal-text*$_1$ | *Expression*$_1$) (*Informal-text*$_1$ | *Range-condition*$_1$) → *Bool*

int-graph-node(*graphnode*)(*dict*) ≜ (4.2.8)

```
1  ( cases graphnode:
2    (mk-Task-node1(silt)          → int-task-node(silt)(dict),
3     mk-Output-node1(,,,)         → int-output-node(graphnode)(dict),
4     mk-Create-request-node1(,)   → int-create-node(graphnode)(dict),
5     mk-Call-node1(,)             → int-call-node(graphnode)(dict),
6     mk-Set-node1(,,)             → int-set-node(graphnode)(dict),
7     mk-Reset-node1(,)            → int-reset-node(graphnode)(dict)))
```

type: *Graph-node*$_1$ → *Entity-dict* ⇒

Назначение — Интерпретация вершины графа.

Параметры

graphnode — Интерпретируемая вершина графа.

int-task-node(*silt*)(*dict*) ≜ (4.2.9)

```
1  ( cases silt:
2    (mk-Assignment-statement1(,) → int-assign-stmt(silt)(dict),
3     mk-Informal-text1()         → int-informal-text(silt)))
```

type: (*Assignment-statement*$_1$ | *Informal-text*$_1$) → *Entity-dict* ⇒

Назначение Интерпретация вершины-работы.

Параметры

sllt Оператор присваивания или неформальный текст.

Алгоритм

Строка 1 *Sllt* интерпретируется как присваивание или как неформальный текст.

$$int\text{-}set\text{-}node(\text{mk-}Set\text{-}node_1(texp, tid, exprl))(dict) \triangleq \tag{4.2.10}$$

```
1  (def val : eval-expression(texp)(dict);
2   def vall : ⟨eval-expression(exprl[i])(dict) | 1 ≤ i ≤ len exprl⟩;
3   let mk-SignalDD(sortl) = dict((tid, SIGNAL)) in
4   def vall' : ⟨reduce-term(sortl[i], vall[i], dict(SCOPEUNIT))(dict) | 1 ≤ i ≤ len vall⟩;
5   def f(t1, t2) : is-equivalent(t1, t2, dict(SCOPEUNIT))(dict);
6   if (∀i ∈ ind vall)(range-check(sortl[i], vall'[i])(dict))
7     then output mk-Set-Timer(tid, val, vall', f) to dict(PORT)
8     else exit("§5.4.1.9: Значение вне диапазона Подтипа"))
```

type: $Set\text{-}node_1 \rightarrow Entity\text{-}dict \Rightarrow$

Назначение Интерпретирует set-node (вершину-установки) путем проверки фактических параметров сигнала-таймера для ошибок-диапазона и затем выдает сигнал установить-таймер во входной-порт.

Параметры

texp Выражение таймера, значение которого обозначает время, установленное в таймере.

tid Идентификатор установленного таймера.

exprl Фактические параметры таймера.

Алгоритм

Строка 1 Оценить выражение таймера и список фактических параметров.

Строка 4 См. *reduce-term* (*терм-сокращения*).

Строка 5 Выполнить функцию *isequivalent*, используемую в процессоре *входного-порта*, чтобы проверить, установлен ли (уже) данный таймер с одинаковыми фактическими параметрами.

Строка 6 Проверить, находятся ли значения фактических параметров таймера в диапазоне связанных с ними сортов.

$$int\text{-}reset\text{-}node(\text{mk-}Reset\text{-}node_1(tid, exprl))(dict) \triangleq \tag{4.2.11}$$

```
1  (def vall : ⟨eval-expression(exprl[i])(dict) | 1 ≤ i ≤ len exprl⟩;
2   let mk-SignalDD(sortl) = dict((tid, SIGNAL)) in
3   def vall' : ⟨reduce-term(sortl[i], vall[i], dict(SCOPEUNIT))(dict) | 1 ≤ i ≤ len vall⟩;
4   def f(t1, t2) : is-equivalent(t1, t2, dict(SCOPEUNIT))(dict);
5   if (∀i ∈ ind vall')(range-check(sortl[i], vall'[i])(dict))
6     then output mk-Reset-Timer(tid, vall', f) to dict(PORT)
7     else exit("§5.4.1.9: Значение вне диапазона Подтипа"))
```

type: $Reset\text{-}node_1 \rightarrow Entity\text{-}dict \Rightarrow$

Назначение Интерпретировать вершину-сброса путем проверки, находятся ли значения фактических параметров сигнала таймера в диапазоне их сортов, и если находятся, то выдать сигнал *Reset-Timer* (*Сбросить-Таймер*) процессору *входного-порта*.

Параметры

tid	Идентификатор сбрасываемого таймера.
exprl	Фактические параметры таймера.

Алгоритм

Строка 1	Оценить список фактических параметров таймера.
Строка 3	См. *Reduce-term*.
Строка 4	Выполнить функцию is-equivalent (эквивалентен), используемую в процессоре *входного-порта*, чтобы проверить, установлен ли (уже) данный таймер с одинаковыми фактическими параметрами.
Строка 5	Проверить, находятся ли значения фактических параметров таймера в диапазоне связанных с ними сортов.

$$\mathit{int\text{-}assign\text{-}stmt}(\mathbf{mk}\text{-}\mathit{Assignment\text{-}statement}_1(\mathit{vid}, \mathit{exp}))(\mathit{dict}) \triangleq \tag{4.2.12}$$

```
1  (def val : eval-expression(exp)(dict);
2   update-stg(vid, val)(dict))
```

type: $\mathit{Assignment\text{-}statement}_1 \to \mathit{Entity\text{-}dict} \Rightarrow$

Назначение Переменная присваивается значению выражения.

Параметры

vid	Переменная.
exp	Выражение.

Алгоритм

Строка 1	Оценить значение выражения.
Строка 2	Занести в память *vid* и значение выражения.

$$\mathit{int\text{-}informal\text{-}text}(\mathbf{mk}\text{-}\mathit{Informal\text{-}text}_1()) \triangleq \tag{4.2.13}$$

```
1  (/* Этот неформальный Meta-IV-текст обозначает интерпретацию неформального текста */)
```

type: $\mathit{Informal\text{-}text}_1 \Rightarrow$

$$\mathit{int\text{-}output\text{-}node}(\mathbf{mk}\text{-}\mathit{Output\text{-}node}_1(\mathit{sid}_1, \mathit{exprl}, \mathit{dest}, \mathit{via}))(\mathit{dict}) \triangleq \tag{4.2.14}$$

```
1  (def vall : ⟨eval-expression(exprl[i])(dict) | 1 ≤ i ≤ len exprl⟩;
2   def pidval : eval-expression(dest)(dict);
3   let mk-SignalDD(sortl) = dict((sid1, SIGNAL)) in
4   def vall' : ⟨reduce-term(sortl[i], vall[i], dict(SCOPEUNIT))(dict) | 1 ≤ i ≤ len vall⟩;
5   if (∀i ∈ ind vall')(range-check(sortl[i], vall'[i])(dict))
6      then output mk-Send-Signal(sid1, vall', pidval, via) to system
7      else exit("§5.4.1.9: Значение вне диапазона Подтипа"))
```

type: $\mathit{Output\text{-}node}_1 \to \mathit{Entity\text{-}dict} \Rightarrow$

Назначение Интерпретация выходной вершины путем проверки, находятся ли значения фактических параметров в диапазоне их сортов, и, если находятся, выдать процессору системы Send-Signal.

Параметры

sid_1	Идентификатор посылаемого сигнала.
exprl	Фактические параметры сигнала.
dest	PID-выражение, обозначающее процесс, к которому должен быть направлен сигнал.
via	Множество идентификаторов путей, обозначающее путь следования сигнала.

Алгоритм

Строка 1	Оценить список фактических параметров и pid-значение.
Строка 4	См. *reduce-term*.
Строка 5	Проверить, находятся ли значения фактических параметров сигнала в диапазоне связанных с ними сортов.

int-create-node(mk-*Create-request-node*$_1$(*pid*, *exprl*))(*dict*) ≜ (4.2.15)

```
1  (def vall : ⟨eval-expression(exprl[i])(dict) | 1 ≤ i ≤ len exprl⟩;
2   let mk-ProcessDD(formparms,,,,) = dict((pid, PROCESS)) in
3   let sortl = ⟨s-Sort-reference-identifier1(dict((formparms[i], VALUE))) | 1 ≤ i ≤ len formparms⟩ in
4   def vall' : ⟨reduce-term(sortl[i], vall[i], dict(SCOPEUNIT))(dict) | 1 ≤ i ≤ len vall⟩;
5   output mk-Create-Instance-Request(pid, vall') to system;
6   input mk-Create-Instance-Answer(offspring') from system
7      ⇒  if offspring' = nil then
8            (let nullterm = dict(NULLVALUE) in
9             offspring := nullterm)
10          else
11           offspring := offspring')
```

type: *Create-request-node*$_1$ → *Entity-dict* ⇒

Назначение Интерпретация вершины-создать.

Параметры

pid	Идентификатор создаваемого процесса.
exprl	Список фактических параметров.

Алгоритм

Строка 1	Оценить значение каждого фактического параметра.
Строка 2	Установить список Sort-reference-identifiers (идентификаторов-ссылок-на-Сорт) формальных параметров.
Строка 4	См. *reduce-term*.
Строка 5	Выдать процессору системы Create-Instance-Request (Запрос-Создания-Экземпляра).
Строка 6	Ввести либо PID-значение создаваемого процесса, либо nil, если процесс не мог быть создан.
Строка 9	Если процесс не мог быть создан, то *nullterm* (*нулевому-терму*) присваивается offspring (потомок). Новый процесс не может быть создан, если уже имеется максимальное число экземпляров этого процесса.
Строка 11	Если процесс не мог быть создан, то PID-значению, полученному от процессора системы, присваивается offspring (потомок).

int-call-node(mk-*Call-node*$_1$(*prd-id*, *exprl*))(*dict*) $\triangleq$ (4.2.16)

```
1  (dcl newstg := [] type Stg;
2   let mk-ProcedureDD(formparms, graph) = dict((prd-id, PROCEDURE)) in
3   let mk-Identifier1(qual, nm) = prd-id in
4   let newlevel = qual ⌢ (mk-Procedure-qualifier1(nm)) in
5   let decl-parm-set = {(mk-Identifier1(l, ), VALUE) ∈ dom dict | l = newlevel} in
6   let dict1 = establ-dyn-dict(formparms, exprl, newstg, decl-parm-set)(dict) in
7   let dict2 = dict1 + [SCOPEUNIT ↦ newlevel] in
8   (trap exit(RETURN) with I in
9    int-procedure-graph(graph)(dict2)))
```

type: *Call-node*$_1$ → *Entity-dict* ⇒

Назначение Интерпретация вершины вызова процедуры.

Параметры

prd-id	Идентификатор вызываемой процедуры.
exprl	Фактические параметры для вызова процедуры.

Алгоритм

Строка 1	Объявить новую переменную пустой памяти, используемой в качестве индивидуальной памяти для процедуры.
Строка 2	Установить список формальных параметров процедуры и граф процедуры.
Строки 3—4	Вычислить уровень области видимости процедуры.
Строка 5	Выбрать множество переменных, определенных или используемых на данном уровне в качестве формальных параметров.
Строка 6	Сделать новый dict и изменить новую память согласно новым определениям, формальным и фактическим параметрам путем обращения к *establ-dyn-dict*.
Строки 7—9	Ввести новый уровень и взять exit (RETURN) из интерпретации графа процедуры, не выполняя никаких действий.

int-procedure-graph(*graph*)(*dict*) $\triangleq$ (4.2.17)

```
1  (let mk-Procedure-graph1(mk-Procedure-start-node1(trans), stateset) = graph in
2   tixe [statenm ↦ int-state-node(statenode)(dict) |
3         statenode ∈ stateset ∧ s-State-name1(statenode) = statenm] in
4   let mk-Transition1(nodel, termordec) = trans in
5   int-transition(nodel, termordec)(dict))
```

type: *Procedure-graph*$_1$ → *Entity-dict* ⇒

Назначение Интерпретация графа процедуры.

Параметры

graph	Граф процедуры.

Алгоритм

Строка 1	Разбиение графа на начальный переход и множество состояний.
Строка 2	Взять все exit (*statenm*) из int-state-node (вершины-состояния-интерпретации) и int-transition (интерпретации-перехода) путем интерпретации соответствующей вершины-состояния (объяснение tixe-конструкции приведено в аннотациях к int-process-graph (интерпретация-графа-процесса)).
Строка 4	Интерпретация начального-перехода.

4.3 Вспомогательные функции

Определения вспомогательных функций, используемых в предыдущих разделах, приводятся ниже. Вспомогательные функции оценивают значения выражений, выполняют проверку диапазонов, управляют индивидуальной памятью и динамической частью в entity-dict (см. раздел 2.6).

eval-expression(*exp*)(*dict*) ≜ (4.3.1)

```
 1  (if exp = nil then
 2     nil
 3   else
 4    cases exp:
 5     (mk-Identifier1(,)
 6        → eval-variable-identifier(exp)(dict),
 7      mk-Ground-expression1()
 8        → eval-ground-expression(exp)(dict),
 9      mk-Operator-application1(,)
10        → eval-operator-application(exp)(dict),
11      mk-Conditional-expression1(e1, e2, e3)
12        → eval-conditional-expression(e1, e2, e3)(dict),
13      mk-View-expression1(,)
14        → eval-view-expression(exp)(dict),
15      mk-Timer-active-expression1(,)
16        → eval-active-expression(exp)(dict),
17      mk-Now-expression1()
18        → eval-now-expression(),
19      mk-Self-expression1()
20        → mk-Ground-term1(dict(SELF)),
21      mk-Parent-expression1()
22        → mk-Ground-term1(dict(PARENT)),
23      mk-Offspring-expression1()
24        → mk-Ground-term1(c offspring),
25      mk-Sender-expression1()
26        → mk-Ground-term1(c sender)))
```

type: [*Expression*$_1$] → *Entity-dict* ⇒ [*Value*]

Назначение Оценка значения выражения в AS_1.

Параметры

Выражение в AS_1.

Результат Значение выражения.

Алгоритм

Строка 1 Если выражение имеет значение nil, то результатом является nil-значение.

Строка 19 Если выражениями являются Self (Сам процесс), Parent (Родитель), Offspring (Потомок) или Sender (Отправитель), то возвращается пассивный-терм содержимого этих выражений.

eval-variable-identifier(*id*)(*dict*) ≜ (4.3.2)

```
1  (let mk-VarDD(vid, , , stg) = dict((id, VALUE)) in
2   if c stg(vid) = UNDEFINED
3      then exit("§5.5.2.2: Значение переменной, к которой осуществлен доступ, не определено"))
4      else c stg(vid))
```

type: *Identifier*$_1$ → *Entity-dict* ⇒ *Value*

Назначение Оценка идентификатора переменной.

Параметры

id Идентификатор переменной.

Результат Содержимое, если имеется, данной переменной.

Алгоритм

Строка 1 Получение идентификатора, на который произведена ссылка.

Строка 3 Если содержимое памяти для идентификатора, на который произведена ссылка, не определено, то имеет место ошибка.

Строка 4 Содержимое памяти для идентификатора, на который произведена ссылка, возвращено.

eval-ground-expression(mk-*Ground-expression*$_1$(*ex*))(*dict*) ≜ (4.3.3)

```
1  (let mk-Ground-term1(e) = ex in
2   if is-Identifier1(e) then
3      ex
4     else
5      if is-Conditional-term1(e) then
6        (let mk-Conditional-term1(bex, ex1, ex2) = e in
7         eval-conditional-expression(bex, ex1, ex2)(dict))
8       else
9        (let (opid, arglist) = e in
10        let mk-OperatorDD(sortlist, sort) = dict((opid, VALUE)) in
11        if (∀i ∈ ind arglist)(range-check(sortlist[i], arglist[i])(dict)) then
12          (let arglist' = ⟨eval-expression(arglist[i])(dict) | 1 ≤ i ≤ len arglist⟩ in
13           let t = mk-Ground-term1((opid, arglist')) in
14           if range-check(sort, t)(dict)
15              then t
16              else exit("§5.4.1.9: Значение вне диапазона Подтипа"))
17         else
18          exit("§5.4.1.9: Значение вне диапазона Подтипа")))
```

type: *Ground-expression*$_1$ → *Entity-dict* → *Value*

Назначение Оценка значения пассивного выражения.

Параметры

ex Пассивное выражение.

Результат Значение пассивного выражения, заданное в качестве идентификатора операции, и оценочный список аргументов.

Алгоритм

Строка 2 Если пассивный терм состоит из идентификатора, то этот терм возвращается.

Строка 6 Если пассивный терм — это условный терм, то он разлагается и вычисляется.

Строка 9 Если пассивный терм не является ни идентификатором, ни условным выражением, то он должен быть применением операции.

Строка 11 Список аргументов операции проверяется на наличие ошибок диапазонов в соответствии со связанным с ним списком сортов. Если такой ошибки не обнаружено, то идентификатор операции и оценочный список сортов компонуются в пассивный терм. В противном случае имеет место ошибка диапазона.

Строка 14 Пассивный терм проверяется на наличие ошибки диапазона в соответствии с сортом операции. Если такой ошибки не обнаружено, то пассивный терм возвращается.

eval-operator-application(mk-*Operator-application*$_1$(*opid*, *expl*))(*dict*) ≜ (4.3.4)

```
1  (def vall : ⟨eval-expression(expl[i])(dict) | 1 ≤ i ≤ len expl⟩;
2   let term = mk-Ground-term1((opid, vall)) in
3   let mk-OperatorDD(sortl, result) = dict((opid, VALUE)) in
4   if (∀i ∈ Ind sortl)(range-check(sortl[i], vall[i])(dict)) ∧
5      range-check(result, term)(dict)
6      then term
7      else exit("§5.4.1.9: Значение вне диапазона Подтипа"))
```

type: *Operator-application*$_1$ → *Entity-dict* => *Value*

Назначение Оценка применения операции.

Параметры

opid Идентификатор операции.

expl Список аргументов для применения операции.

Результат Значение применения операции, то есть пассивный терм идентификатора операции и оценочный список аргументов.

Алгоритм

Строка 1 Оценить список аргументов.

Строка 2 Образовать пассивный терм идентификатора операции и оценочный список аргументов.

Строка 3 Просмотреть обозначения операций в *Entity-dict*.

Строка 4 Проверить, находятся ли оценочные аргументы в диапазоне связанных с ними сортов.

Строка 5 Проверить, находится ли терм из *строки 2* в диапазоне сорта результата.

Строка 6 Если ошибка диапазона не обнаружена, терм из *строки 2* возвращается.

eval-view-expression(mk-*View-expression*$_1$(id_1, *exp*))(*dict*) ≜ (4.3.5)

```
1  (def pid : eval-expression(exp)(dict);
2   def pid' : reduce-term(dict(PIDSORT), pid, dict(SCOPEUNIT))(dict);
3   output mk-View-Request(id1, pid') to view;
4   (input mk-View-Answer(val) from view
5      ⇒ if val = UNDEFINED
6           then exit("§5.5.2.2: Обозреваемое значение не определено")
7           else val))
```

type: *View-expression*$_1$ → *Entity-dict* ⇒ *Value*

Назначение Оценка значения выражения VIEW.

eval-conditional-expression(exp_1, exp_2, exp_3)(*dict*) ≙ (4.3.6)

```
1  (let trueterm = dict(TRUEVALUE) in
2   let falseterm = dict(FALSEVALUE) in
3   if is-equivalent(eval-expression(exp1)(dict), trueterm, dict(SCOPEUNIT))(dict) then
4     eval-expression(exp2)(dict)
5    else
6     if is-equivalent(eval-expression(exp1)(dict), falseterm, dict(SCOPEUNIT))(dict) then
7       eval-expression(exp3)(dict)
8      else
9       exit("§5.5.2.3: Условие должно проверяться по отношению к TRUE (ИСТИНА) или FALSE (ЛОЖЬ)"))
```

type: *Expression*$_1$ *Expression*$_1$ *Expression*$_1$ → *Entity-dict* ⇒ *Value*

Назначение Оценка значения условного выражения.

Параметры

exp_1	Условное выражение.
exp_2	Выражение следствия.
exp_3	Выражение альтернативы.

Результат Значения выражения следствия и выражения альтернативы зависят от условного выражения.

Алгоритм

Строка 1	Выбрать терм в AS$_1$ для TRUE.
Строка 2	Выбрать терм в AS$_1$ для FALSE.
Строка 3	Если true-терм равен условному, то
Строка 4	оценить значение выражения следствия, иначе
Строка 6	если false-терм равен условному, то
Строка 7	оценить значение выражения альтернативы, иначе
Строка 9	имеет место ошибка (возможна только в том случае, если сорт Булевский имеет дополнительные значения).

eval-active-expression(mk-*Timer-active-expression*$_1$(*timer*, *exprl*))(*dict*) ≙ (4.3.7)

```
1  (let mk-Identifier1(qual, ) = timer in
2   let mk-SignalDD(sortl) = dict((timer, SIGNAL)) in
3   let trueterm = dict(TRUEVALUE),
4       falseterm = dict(FALSEVALUE) in
5   def vall : ⟨eval-expression(exprl[i])(dict) | 1 ≤ i ≤ len exprl⟩;
6   def vall' : ⟨reduce-term(sortl[i], vall[i], dict(SCOPEUNIT))(dict) | 1 ≤ i ≤ len vall⟩;
7   let f(t1, t2) = is-equivalent(t1, t2, qual)(dict) in
8   output mk-Active-Request(timer, vall', f) to dict(PORT);
9   (input mk-Active-Answer(b) from dict(PORT)
10        ⇒ if b then mk-Ground-term1(trueterm) else mk-Ground-term1(falseterm)))
```

type: *Timer-active-expression*$_1$ → *Entity-dict* ⇒ *Value*

Назначение Проверка активности данного таймера.

Параметры

timer	Идентификатор таймера.
exprl	Параметры таймера.

Результат Если данный таймер активен, в AS_1 имеется значение TRUE; в противном случае в AS_1 — значение FALSE.

Алгоритм

Строка 1	Установить список сортов таймера.
Строки 3—4	Выбрать значение TRUE и FALSE в AS_1.
Строка 5	Оценить параметры таймера.
Строка 6	См. *reduce-term*.
Строка 7	Определение функции тестирования эквивалентности между *val1'* и параметрами потенциально активного таймера.
Строка 8	Передать *Active-Request* (*Запрос-Активности*) во входной порт.
Строка 9	Получить *Active-Answer* (*Ответ-Активности*) от входного порта с параметром *b*, обозначающим «активность» таймера.
Строка 10	Возвратить TRUE и FALSE версии AS_1 в зависимости от *b*.

eval-now-expression() $\triangleq$ (4.3.8)

```
1  (output mk-Time-Request() to timer;
2   (input mk-Time-Answer(val) from timer
3        ⇒  val))
```

type: () ⇒ *Value*

Назначение Оценка значения выражения now.

Результат Значение, обозначающее now (см. процессор *Таймера*).

Алгоритм

Строка 1	Передать процессору таймера *Time-Request* (*Запрос-Времени*).
Строка 2	Получить *Time-Answer* (*Ответ-Времени*) со значением now.

establ-dyn-dict(*formparml*, *exprl*, *stg*, *decl-parm-set*)(*dict*) $\triangleq$ (4.3.9)

```
1  (if decl-parm-set ≠ {} then
2     (let (id, VALUE) ∈ decl-parm-set in
3      def dict1 : (mk-InoutparmDD(id) ∈ elems formparml
4                     → (let i ∈ ind formparml be s.t. formparml[i] = mk-InoutparmDD(id) in
5                        let mk-VarDD(vid, sid, rev, stg') = dict((exprl[i], VALUE)) in
6                        [(id, VALUE) ↦ mk-VarDD(vid, sid, rev, stg')]),
7                  mk-InparmDD(id) ∈ elems formparml
8                     → (let i ∈ ind formparml be s.t. formparml[i] = mk-InparmDD(id) in
9                        let mk-VarDD(vid, sid, rev, ) = dict((id, VALUE)) in
10                       let dict' = [(id, VALUE) ↦ mk-VarDD(vid, sid, rev, stg)] in
11                       update-stg(vid, eval-expression(exprl[i])(dict))(dict + dict');
12                       dict'),
13                T → (let mk-VarDD(vid, sid, rev, ) = dict((id, VALUE)) in
14                       let dict' = [(id, VALUE) ↦ mk-VarDD(vid, sid, rev, stg)] in
15                       update-stg(vid, nil)(dict + dict');
16                       dict'));
17      return establ-dyn-dict(formparml, exprl, stg, decl-parm-set \ {(id, VALUE)})(dict) + dict1)
18    else
19     dict)
```

type: *FormparmDD** *Expression*$_1$* **ref** *Stg* (*Identifier*$_1$ VALUE)-**set** → *Entity-dict* ⇒ *Entity-dict*

Назначение Выполнить необходимые изменения в динамической части entity-dict при интерпретации вызова процедуры. Кроме того, в соответствии с определенными в процедуре переменными и формальными входными параметрами модифицируется память.

Параметры

formparml	Список формальных параметров.
actparml	Список фактических параметров.
stg	Ссылка на новую память.
dcl-parm-set	Множество входов *dict*, для которых *dict* и память должны быть модифицированы.

Результат Модифицированный *Entity-dict*.

Алгоритм

Строка 1	Рекурсия останавливается, если набор входов dict пустой.
Строка 2	Взять один из входов dict.
Строка 3	Если он соответствует одному из формальных входных/выходных параметров, тогда
Строки 4—5	отыскать в *dict* соответствующий фактический параметр и его дескриптор.
Строка 6	Пусть переменная формального параметра описывается дескриптором переменной фактического параметра.
Строка 7	Если он соответствует одному из формальных входных параметров,
Строка 8	отыскать индекс формального параметра.
Строки 9—10	Изменить дескрипторы переменных формального параметра для ссылки на новую память.
Строка 11	Модифицировать память для переменной значением фактического параметра, использующего новый дескриптор.
Строка 13	Если он не соответствует ни формальному входному параметру, ни формальному входному/выходному параметру, тогда
Строки 13—16	он анализируется подобно формальному входному параметру, но память модифицируется с помощью nil (UNDEFINED).
Строка 17	Обратиться к *establ-dyn-dict* для остальной части *dcl-parm-set* и переписать результат с только что сконструированным входом.
Строка 19	Останов рекурсии, если для проверки больше нет определений или параметров, и возвратить прежний dict.

(4.3.10)

```
update-stg(id, val)(dict) ≜

 1  (let mk-VarDD(vid, sid, revealed, stg') = dict((id, VALUE)) in
 2   def val' : if val = nil then
 3                 UNDEFINED
 4                else
 5                 reduce-term(sid, val, dict(SCOPEUNIT))(dict);
 6   if range-check(sid, val')(dict)
 7     then (stg' := c stg' + [vid ↦ val'];
 8           if revealed = REVEALED then
 9             (output mk-Reveal(vid, val', dict(SELF)) to view)
10           else
11             I)
12      else exit("§5.4.1.9: Значение вне диапазона Подтипа"))

type: Identifier, Value → Entity-dict ⇒
```

Назначение Модификация памяти для используемого идентификатора переменной используемым значением и раскрытие переменной, если она объявлена раскрытой.

Параметры

id	Идентификатор переменной, для которой должна быть модифицирована память.
val	Значение, модифицирующее память.

Алгоритм

Строка 1	Отыскать дескриптор идентификатора переменной.
Строка 2	Если *val* не равна нулю, она должна быть изменена, чтобы соответствовать идентификатору сорта переменной (см. *reduce-term*). Если величина равна нулю, то память модифицируется с помощью UNDEFINED.
Строки 7—8	Память, на которую имеется ссылка, переписывается при новой паре значение—переменная.
Строка 9	Для раскрытых переменных процессору обозревания посылается Reveal (Раскрыть) с идентификатором переменной, «сокращенным» значением и PID-значением этого процесса.
Строка 12	Если «сокращенное» значение не находится в диапазоне сорта переменной, то имеет место ошибка диапазона.

range-check(*sort-id*, *value*)(*dict*) ≙ (4.3.11)

```
1  if value ∈ {nil, UNDEFINED} then
2    true
3   else
4    ( cases dict((sort-id, SORT)):
5      (mk-SyntypeDD(, mk-Range-condition1(orid, condset))
6         → (let operator1 = make-valuetest-operator(value, orid, condset) in
7            let value' = eval-ground-expression(operator1)(dict) in
8            let trueterm = dict(TRUEVALUE) in
9            is-equivalent(eval-expression(value')(dict), trueterm, dict(SCOPEUNIT))(dict)),
10      T → true))
```

type: *Sort-reference-identifier*$_1$ [*Value*] → *Entity-dict* → *Bool*

Назначение Проверить, находится ли значение в диапазоне своего сорта.

Параметры

sort-id	Идентификатор сорта.
value	Проверяемое значение.

Результат True, если значение находится в диапазоне, в противном случае false.

Алгоритм

Строка 1	nil или UNDEFINED в диапазоне всех сортов.
Строка 4	Отыскать описание сорта.
Строка 6	Если сорт — это подтип, то использовать *orid*, *condset* и *value* для выполнения SDL-операции (*operator*$_1$) по проверке диапазона. См. операцию *make-valuetest-operator*.
Строка 8	Выбрать true версии AS_1.

5 Образование *Entity-dict* и обработка Абстрактных Типов Данных

Данный раздел содержит функции, образующие *entitydict* (см. определение домена *Entity-dict*). *Entitydict* используется *sdl-процессом*, а также процессом *Системы*, создающим также объект при обращении к функции входа *extract-dict*.

Данный раздел имеет в своем составе следующие подразделы:

1. Создание простых замкнутых дескрипторов, таких как дескрипторы подтипов, переменных, сигналов и т. д. Созданы также дескрипторы для процессов (*ProcessDD*), но с пустым множеством *Reachability*.

   Дескрипторы элементов создаются независимо от наличия этих элементов в структурной единице, содержащейся в согласованном подмножестве. Причиной этого является проверка согласованности типов данных всех структурных единиц.

2. Создание дескрипторов для *Data-type-definition*$_1$*в* (*Определений-типов-данных*$_1$) (*TypeDD*). Дескриптор создается для каждой структурной единицы до создания дескрипторов для сортов (*SortDD*).

3. Выбор согласованного подмножества.

4. Создание *Reachabilities* (Достижимостей) для процессов (то есть создание всех возможных коммуникационных путей для процессов).

Выбор согласованного подмножества происходит после создания дескрипторов для всех элементов, но до создания *Reachabilities* путем удаления дескрипторов процессов, не входящих в согласованное подмножество. Конструкцию *entitydict* можно рассматривать как некоторый промежуточный уровень между статической и динамической семантиками. Особые ситуации, приведенные в данном разделе (проверки согласованного подмножества и согласованности абстрактных типов данных), могут рассматриваться как некоторые дополнительные условия статического характера Динамической Семантики, так как:

- Проверка выбора согласованного уточненного подмножества требует построения *Reachabilities*.

  Более точно, выбор согласованного (уточненного) подмножества не относится к особой ситуации, поскольку не является частью спецификации в SDL, но для проверки его свойств проводятся проверки согласованности множества идентификаторов блоков, отражающих согласованность подмножества.

- Проверки согласованности эквивалентных классов и взаимного исключения ответов по принятию решений нельзя легко выразить в терминах AS$_1$, то есть наличие этих (статических) проверок в Динамической Семантике обусловлено конструкцией эквивалентных классов.

$$extract\text{-}dict(as_1 tree, blockset, expiredf, terminf) \triangleq \tag{5.1}$$

```
1  (let (as1pid, as1null, as1true, as1false) = terminf in
2   let d = [EXPIREDF   ↦ expiredf,
3            PIDSORT    ↦ as1pid,
4            NULLVALUE  ↦ as1null,
5            TRUEVALUE  ↦ as1true,
6            FALSEVALUE ↦ as1false] in
7   (let mk-System-definition1(nm, bset, cset, sigset, tp, synset) = as1tree in
8    let level = (mk-System-qualifier1(nm)) in
9    let leafprocesses = select-consistent-subset(bset, blockset, level) in
10   let dict' = extract-sortdict(tp, level)(d) in
11   let dict'' = merge {make-entity(entity, level)(dict') | entity ∈ (sigset ∪ synset ∪ bset)} in
12   let d' = dict'' + [(id, q) ↦ d((id, q)) | (id, q) ∈ dom d ∧ (q = PROCESS ⊃ id ∈ leafprocesses)] in
13   make-structure-paths(bset, cset, level)(d')))
```

type : *System-definition*$_1$ *Block-identifier*$_1$-**set** *Is-expired Term-information* → *Entity-dict*

Назначение Образовать *entitydict*, используемый *sdl-процессами* и процессом *системы*. Объект конструируется процессом *системы* и задается в виде фактического параметра всякий раз, когда запускается новый *sdl-процесс*.

Параметры

as_1 tree Представление абстрактного синтаксиса системы, то есть объекта домена *System-definition*$_1$ (*Определение-системы*$_1$).

blockset (Предполагаемое) согласованное подмножество представлено совокупностью идентификаторов блоков и идентификаторов подструктур блоков. Хотя структурная единица системы также содержится в согласованном подмножестве, она не включена в *blockset*.

expiredf Функция, вырабатывающая true по истечении времени действия данного таймера.

terminf Некоторые идентификаторы в AS_1, используемые основной системой.

Результат Объект домена *Entity-dict*.

Алгоритм

Строка 1 Разложить *Term-information* (*Информацию-терма*) (определена в Приложении F.2), содержащую *Идентификатор*$_1$*ы* PiD-сорта, литералы NULL, TRUE и FALSE.

Строки 2—6 Создать начальный *entitydict*, в котором имеются функция истечения времени и информация терма.

Строка 8 Образовать квалификатор, обозначающий уровень системы.

Строка 9 Проверить корректность согласованного подмножества и выбрать идентификаторы процессов, содержащихся в согласованном подмножестве.

Строка 10 Создать дескрипторы для *Data-type-definition*$_1$ (*Определения-типа-данных*$_1$), литералов и операций, определенных на уровне системы.

Строка 11 Произвести действие *entitydict* по отношению к сигналам (*sigset*), подтипам (*synset*) и блокам (*bset*).

Строка 12 Удалить дескрипторы процессов, не находящиеся в согласованном подмножестве.

Строка 13 Ввести *Reachabilities* во все дескрипторы процессов (*ProcessDD*); функция *make-structure-paths* (*сделать пути-структуры*) возвращает *entitydict* на то место, где они были введены.

make-entity(*entity*, *level*)(*dict*) ≜ (5.1.1)

```
1  cases entity:
2   (mk-Timer-definition1(nm, sortlist)
3      → dict + [(mk-Identifier1(level, nm), SIGNAL) ↦ mk-SignalDD(sortlist)],
4    mk-Signal-definition1(,,)
5      → dict + make-signal-dict(entity, level),
6    mk-Process-definition1(,,,,,,,,,,)
7      → make-process-dict(entity, level)(dict),
8    mk-Procedure-definition1(,,,,,,)
9      → make-procedure-dict(entity, level)(dict),
10   mk-Variable-definition1(nm, sort, rev)
11     → dict + [(mk-Identifier1(level, nm), VALUE) ↦ mk-VarDD(, sort, rev, )],
12   mk-Syn-type-definition1(nm, psort, ran)
13     → dict + [(mk-Identifier1(level, nm), SORT) ↦ mk-SyntypeDD(psort, ran)],
14   mk-Block-definition1(,,,,,,,)
15     → make-block-dict(entity, level)(dict),
16   T → dict)
```

type: $Decl_1$ $Qualifier_1$ → *Entity-dict* → *Entity-dict*

Назначение Возвратить *Entity-dict* (*dict*), который изменен и содержит новую информацию для элемента.

Параметры

entity	Определение элемента в AS_1.
level	Квалификатор, обозначающий содержащую определение структурную единицу.

Алгоритм Создать новую информацию (вклад) для имеющегося элемента. Следует отметить, что таймер анализируется как обычный сигнал и что для переменных обозревания не требуется дескриптор (в операторе выбора нет $View\text{-}definition_1$ (*Определения-Обозревания*$_1$)).

make-signal-dict(mk-*Signal-definition*$_1$(*nm*, *sortlist*, *refinement*), *level*) ≜ (5.1.2)

```
1  (let d = [(mk-Identifier1(level, nm), SIGNAL) ↦ mk-SignalDD(sortlist)] in
2   if refinement = nil then
3     d
4   else
5    (let mk-Signal-refinement1(subsigset) = refinement in
6     let level' = level ⌢ (mk-Signal-qualifier1(nm)) in
7      d + merge {make-signal-dict(s-Signal-definition1(sdef), level') | sdef ∈ subsigset}))
```

type: *Signal-definition*$_1$ $Qualifier_1$ → *Entity-dict*

Назначение Сделать вклад *entitydict* для сигнала и его подсигналов. Отметим, что дескриптор сигнала не сообщает, является ли сигнал подсигналом или не является таковым. Это происходит потому, что если только на обоих концах линии связи выбраны одни и те же сигналы, то выбор подсигнала корректен, независимо от того, являются ли сигналы подсигналами или не являются таковыми.

Параметры

Signal-definition$_1$	Определение сигнала в AS_1, состоящего из	
	nm	имени сигнала,
	sortlist	сортов значений, переносимых сигналом.

refinement Часть уточнения сигнала.

level Квалификатор, обозначающий структурную единицу, где определен сигнал.

Алгоритм

Строка 1 Сделать вклад для сигнала и

Строки 5—7 сделать вклад для подсигналов с квалификатором, обозначающим структурную единицу, являющуюся определением сигнала.

make-process-dict(*pdef*, *level*)(*dict*) $\triangleq$ (5.1.3)

```
1  (let mk-Process-definition1(nm, inst, f, pset, sigset, tp, synset, vset, , tset, graph) = pdef in
2   let mk-Number-of-instances1(init, maxi) = inst,
3       pid = mk-Identifier1(level, nm),
4       level' = level ⌢ (mk-Process-qualifier1(nm)) in
5   let parm = ⟨mk-Identifier1(level', s-Variable-name1(f[i])) | 1 ≤ i ≤ len f⟩ in
6   let parmd = [(parm[i], VALUE) ↦ mk-VarDD(, s-Sort-reference-identifier1(f[i]), nil, ) |
7                1 ≤ i ≤ len f] in
8   let dict' = extract-sortdict(tp, level')(dict + parmd) in
9   let dict'' = merge {make-entity(entity, level')(dict') |
10                      entity ∈ (pset ∪ sigset ∪ synset ∪ vset ∪ tset)} in
11  let mk-Process-graph1(, stateset) = graph in
12  let nodeset = union {statenodeset | mk-State-node1(, , statenodeset) ∈ stateset} in
13  let insigset = {sigid | mk-Input-node1(sigid, , ) ∈ nodeset} in
14  let localreach = (pid, insigset, ⟨⟩) in
15  if is-wf-decision-answers(graph, level)(dict) then
16    dict'' + [(pid, PROCESS) ↦ mk-ProcessDD(parm, init, maxi, graph, {localreach})]
17   else
18    exit("§2.7.5: Ответы в действиях по принятию решения не являются взаимно исключающими")
```

type: *Process-definition*$_1$ *Qualifier*$_1$ $\rightarrow$ *Entity-dict* $\rightarrow$ *Entity-dict*

Назначение Возвратить вклад *entitydict* для процесса и всех его определений.

Параметры

pdef Определение процесса в AS_1.

level Квалификатор, обозначающий структурную единицу, где определен процесс.

Алгоритм

Строка 2 Выбрать начальное число экземпляров (*init*) и максимальное число экземпляров (*maxi*).

Строка 3 Образовать *Идентификатор*$_1$, обозначающий процесс.

Строка 4 Образовать квалификатор, обозначающий структурную единицу, являющуюся процессом.

Строка 5 Образовать *Идентификатор*$_1$*ы* для формальных параметров.

Строка 6 Образовать вклад *Entity-dict* для формальных параметров. Заметим, что они анализируются как обычные переменные.

Строка 8 Сделать *entitydict*, корректируемый дескриптором, для *Data-type-definition*$_1$ (*Определения-типа-данных*$_1$), определенного в процессе.

Строки 9—10 Сделать вклад для содержащих процедуру определений (*pset*), определений сигналов (*sigset*), определений подтипов (*synset*), определений переменных (*vset*), определений таймеров (*tset*).

Строка 11 Пусть *stateset* обозначает множество состояний процесса.

Строка 12 Пусть *nodeset* обозначает множество входных вершин процесса.

Строка 13 Пусть *insigset* обозначает множество сигналов, получаемых во входной вершине процесса.

Строка 14 Пусть *localreach* обозначает *Reachability*, используемую для маршрутизации сигналов к экземплярам процесса данного типа.

Строка 15 Действия по принятию решений, содержащиеся в графе процесса, должны содержать взаимно исключающие ответы

Строка 16 и корректировать образованный *entitydict* дескриптором самого процесса. Отметим, что на этой стадии совокупность *Reachability* для процесса содержит только *Reachability*, используемую для маршрутизации сигналов к экземплярам процесса данного типа.

make-procedure-dict(*procdef*, *level*)(*dict*) $\triangleq$ (5.1.4)

```
1  (let mk-Procedure-definition1(nm, fp, pset, tp, sset, vset, graph) = procdef in
2   let level' = level ⌢ (mk-Procedure-qualifier1(nm)) in
3   let (fparml, fdict) = make-formal-parameters(fp, level) in
4   let pid = mk-Identifier1(level, nm) in
5   let dict' = extract-sortdict(tp, level')(dict + fdict) in
6   let dict'' = merge {make-entity(entity, level')(dict') | entity ∈ (pset ∪ sset ∪ vset)} in
7   if is-wf-decision-answers(graph, level)(dict) then
8     dict'' + [(pid, PROCEDURE) ↦ mk-ProcedureDD(fparml, graph)]
9    else
10    exit("§2.7.5: Ответы в действиях по принятию решения не являются взаимно исключающими")
```

type: *Procedure-definition*$_1$ *Qualifier*$_1$ $\rightarrow$ *Entity-dict* $\rightarrow$ *Entity-dict*

Назначение Возвратить вклад *entitydict* для процедуры и всех ее определений.

Параметры

procdef Определение процедуры в AS_1.

level Квалификатор, обозначающий структурную единицу, где определена процедура.

Алгоритм

Строка 2 Образовать квалификатор, обозначающий структурную единицу процедуры.

Строка 3 Выработать информацию о том, являются ли формальными параметрами IN или IN/OUT (*fparml*) и дескрипторы *entitydict* формальных параметров (*fdict*).

Строка 4 Образовать квалификатор, обозначающий структурную единицу, являющуюся процедурой.

Строка 5 То же, что и для процесса (см. выше).

Строка 6 Образовать дескрипторы для содержащих процедуры определений (*pset*), определений подтипов (*sset*), определений переменных (*vset*).

Строка 7 Действия по принятию решений, содержащиеся в графе процедуры, должны содержать взаимно исключающие ответы.

Строка 8 Образовать дескриптор для самой процедуры.

make-formal-parameters(*parml*, *level*) ≜ (5.1.5)

```
1  (if parml = ⟨⟩ then
2     (⟨⟩, [])
3   else
4   (let (parmrest, drest) = make-formal-parameters(tl parml, level) in
5    let id = mk-Identifier1(level, s-Variable-name1(hd parml)) in
6    let (p, d) =
7        cases hd parml:
8         (mk-In-parameter1(, sort)
9            → (mk-InparmDD(id), [(id, VALUE) ↦ mk-VarDD(, sort, nil, )]),
10        mk-Inout-parameter1(, )
11           → (mk-InoutparmDD(id), [])) in
12  (⟨p⟩ ⌢ parmrest, d + drest)))
```

type: *Procedure-formal-parameter*$_1$* *Qualifier*$_1$ → *FormparmDD** *Entity-dict*

Назначение Образовать (рекурсивно) и возвратить список дескрипторов, содержащих информацию о том, являются ли формальными параметрами параметры IN или IN/OUT, а также возвратить дескрипторы *entity* для них.

Параметры

parml	Формальные параметры процедуры в AS_1.
level	Квалификатор, обозначающий структурную единицу процедуры.

Алгоритм

Строка 1	При нахождении в конце списка формальных параметров возврата нет.
Строка 4	Образовать дескрипторы для остального списка.
Строка 5	Сделать *Идентификатор*$_1$ для первого параметра в списке.
Строки 6—10	Сделать дескриптор параметра и вклад *entitydict* для первого параметра, соединенного с остальными параметрами списка, и возвратить дескрипторы *entitydict* для первого параметра, соединенного с остальными параметрами списка.

make-block-dict(*bdef*, *level*)(*dict*) ≜ (5.1.6)

```
1  (let mk-Block-definition1(bnm, pdefs, sigdefs, , , datatype, syntype, sub) = bdef in
2   let level' = level ⌢ (mk-Block-qualifier1(bnm)) in
3   let sortd = extract-sortdict(datatype, level')(dict) in
4   let dict' = sortd + merge {make-entity(entity, level')(sortd) | entity ∈ (sigdefs ∪ syntype ∪ pdefs)} in
5   if sub = nil then
6     dict'
7    else
8     (let mk-Block-substructure-definition1(snm, bdefs, , , sdefs, tp, syndefs) = sub in
9      let level'' = level' ⌢ (mk-Block-substructure-qualifier1(snm)) in
10     let sortd' = extract-sortdict(tp, level'')(dict') in
11     sortd' + merge {make-entity(entity, level'')(sortd') | entity ∈ (bdefs ∪ sdefs ∪ syndefs)}))
```

type: *Block-definition*$_1$ *Qualifier*$_1$ → *Entity-dict* → *Entity-dict*

Назначение Образовать и возвратить дескрипторы *entitydict* для элементов, определенных в блоке. Заметим, что здесь не рассматриваются объемлемые определения маршрутов сигналов, каналов, соединений и т. д.

Параметры

bdef	Определение блока в AS_1.
level	Определение квалификатора для блока.

Алгоритм

Строка 1	Разложить определение блока.
Строка 2	Образовать квалификатор, обозначающий блок.
Строка 3	Скорректировать *entitydict* для включения *Data-type-definition*$_1$ (*Определения-типа-данных*$_1$), определенного в блоке.
Строка 4	Скорректировать *entitydict* для включения сигналов (*sigdefs*), подтипов (*syntype*) и процессов (*pdefs*), определенных в блоке.
Строка 5	Если подструктура блока не специфицирована, тогда возвратить вклад *Entity-dict* для блока.
Строка 8	Разложить подструктуру блока.
Строка 9	Образовать квалификатор, обозначающий уровень подструктуры блока.
Строка 10	Скорректировать *entitydict* для включения *Data-type-definition*$_1$, определенного в подструктуре блока.
Строка 11	Возвратить указанный скорректированный *entitydict*, объединенный с вкладами от блоков (*bdefs*), сигналов (*sdefs*), подтипов (*syndefs*).

is-wf-decision-answers(*graph*, *level*)(*dict*) $\triangleq$ (5.1.7)

```
1  (let (starttrans, stateset) =
2        cases graph:
3         (mk-Procedure-graph1(init, stset) → (init, stset),
4          mk-Process-graph1(init, stset)   → (init, stset)) in
5   let trans = s-Transition1(starttrans) in
6   is-wf-transition-answers(trans, level)(dict) ∧
7   (∀mk-State-node1(, , inputs) ∈ stateset)
8    ((∀input ∈ inputs)(is-wf-transition-answers(s-Transition1(input), level)(dict))))
```

type: (*Procedure-graph*$_1$ | *Process-graph*$_1$) *Qualifier*$_1$ → *Entity-dict* → *Bool*

Назначение Проверить, что ответы в действиях по принятию решений в графе процедуры или процесса являются взаимно исключающими.

Параметры

graph	Граф процедуры или процесса.
level	*Квалификатор*$_1$, обозначающий процедуру или процесс.

Результат True в случае успеха.

Алгоритм

Строки 1—4	Пусть *starttrans* обозначает стартовую вершину графа, а *stateset* — состояния графа.
Строка 5	Пусть *trans* обозначает начальный переход.
Строка 6	Ответы в случаях принятия решений в начальных переходах должны быть взаимно исключающими и
Строка 7	для каждого состояния каждая входная вершина должна храниться в состоянии.
Строка 8	Переход во входной вершине содержит взаимно исключающие ответы при принятии решений.

```
is-wf-transition-answers(mk-Transition1(trans, ), level)(dict) ≜                    (5.1.8)
 1  (∀mk-Decision-node1(, answerset, elsetrans) ∈ elems trans)
 2   ((elsetrans ≠ nil ⊃ is-wf-transition-answers(elsetrans, level)(dict)) ∧
 3    (∀mk-Decision-answer1(answer1, trans1), mk-Decision-answer1(answer2, trans2) ∈ answerset)
 4     (is-wf-transition-answers(trans1, level)(dict) ∧
 5      is-wf-transition-answers(trans2, level)(dict) ∧
 6      (answer1 ≠ answer2 ∧ is-Range-condition1(answer1) ∧ is-Range-condition1(answer2) ⊃
 7       (let mk-Range-condition1(orid, cset1) = answer1,
 8            mk-Range-condition1(, cset2) = answer2 in
 9        (∀term ∈ Ground-expression1)
10         ((let dict' = dict + [SCOPEUNIT ↦ level] in
11           trap exit with true in
12           let answerterm1 = eval-ground-expression(make-valuetest-operator(term, orid, cset1))(dict'),
13               answerterm2 = eval-ground-expression(make-valuetest-operator(term, orid, cset2))(dict') in
14           ¬is-equivalent(answerterm1, answerterm2, level)(dict)))))))
```

type: *Transition*$_1$ *Qualifier*$_1$ → *Entity-dict* → *Bool*

Назначение Проверить, чтобы каждое действие по принятию решения в переходе содержало взаимно исключающие ответы.

Параметры

trans Действия при переходе.

level Квалификатор, обозначающий объемлющую структурную единицу.

Алгоритм

Строка 1 Для каждой вершины принятия решения в списке действий должны выполняться действия, указанные в строках 2—14.

Строка 2 Переход в части else должен содержать взаимно исключающие ответы и

Строка 3 для каждых двух ветвей в решении должны выполняться действия, указанные в строках 4—14.

Строки 4—5 Переходы в двух ветвях должны содержать взаимно исключающие ответы и

Строка 6 если имеются различные ветви и обе содержат в ответах формальный текст, тогда

Строки 7—8 пусть *orid* обозначает *Идентификатор*$_1$ операции OR (ИЛИ), *cset*1 обозначает условия на диапазон для одной из двух ветвей, а *cset*2 — условия на диапазон для другой ветви.

Строка 9 Для каждого пассивного терма (*term*) должно выполняться следующее:

Строки 12—13 пассивный терм (*answerterm*1), производный от *term* и *cset*1 (первое множество условий) (строка 12), не должен быть в одном и том же эквивалентном классе, что и пассивный терм (*answerterm*2), производный от *term* и второго множества условий (*cset*2). Конструкция *make-valuetest-operator* возвращает *Пассивное-выражение* (*Ground-expression*), которое вычисляется с помощью *eval-ground-expression*, образуя пассивный терм. Любые *выходы* из *eval-ground-expression* улавливаются (строка 11), так как до тех пор, пока интерпретируется принятие решения, не должны проводиться неявные проверки диапазонов подтипами.

5.2 Обработка Абстрактных Типов Данных

Данный раздел содержит функции обработки абстрактных типов данных. Функциями входа являются:

extract-sortdict Применяется в процесе образования *entitydict* и создает дескрипторы типов, дескрипторы сортов, дескрипторы литералов и дескрипторы операций.

reduce-term Используется при передаче терма в другую структурную единицу, как, например, в фактических параметрах, присвоении нелокальной переменной и т. д.

is-equivalent Используется при сравнении двух термов, например, в условных выражениях, проверке диапазона и вершинах принятия решения.

(5.2.1)

extract-sortdict(*typedef*, *l*)(*dict*) ≜

```
1  (let mk-Data-type-definition1(tnm, union, sorts, signatureset, eqs) = typedef in
2   let tid = mk-Identifier1(l, tnm) in
3   let (psmap, eqs') =
4       if union = {} then
5         ([], {})
6        else
7         (let tid' ∈ union in
8          let mk-TypeDD(pmap, equa) = dict((tid', TYPE)) in
9          (pmap, equa)) in
10   let literald =
11       [mk-Identifier1(l, s-Literal-operator-name1(lit)) ↦ mk-OperatorDD(⟨⟩, s-Result1(lit)) |
12        lit ∈ signatureset ∧ is-Literal-signature1(lit)],
13       operatord =
14       [mk-Identifier1(l, s-Operator-name1(op)) ↦ mk-OperatorDD(s-Argument-list1(op), s-Result1(op)) |
15        op ∈ signatureset ∧ is-Operator-signature1(op)] in
16   let dict' = dict + [(id, VALUE) ↦ literald(id) | id ∈ dom literald] +
17       [(id, VALUE) ↦ operatord(id) | id ∈ dom operatord] in
18   let sortd = [(id, SORT) ↦ mk-SortDD(tid) | id ∈ sorts] in
19   let sortset = {mk-Identifier1(l, nm) | nm ∈ (sorts ∪ dom psmap)},
20       sortmap = [sort ↦ make-equivalent-classes(sort)(dict') | sort ∈ sortset] in
21   let equations = eqs ∪ eqs' in
22   let sortmap' = eval-equations(sortmap, equations)(dict) in
23   let dict'' = dict' + sortd + [(tid, TYPE) ↦ mk-TypeDD(sortmap', equations)] in
24   if (∃{mk-Ground-term1(t), mk-Error-term1()} ⊂ union rng sortmap')(is-Identifier1(t)) then
25     exit("§5.4.1.7: Литерал равен терму-ошибке")
26    else
27     if is-wf-values(psmap, sortmap')(dict'') then
28       dict''
29      else
30       exit("Z.100 §5.2.1: Порождение или сокращение эквивалентных классов объемлющей структурной единицы"))
```

type: *Data-type-definition*$_1$ *Qualifier*$_1$ → *Entity-dict* → *Entity-dict*

Назначение Модификация entitydict, содержащего дескриптор для *Data-type-definition*$_1$ и для содержащихся в нем сортов, операций и литералов.

Параметры

typedef *Data-type-definition*$_1$ (*Определение-типа-данных*$_1$).

l Уровень, на котором он определен.

Результат Модифицированный *entitydict*.

Алгоритм

Строка 2 Образовать *Идентификатор*$_1$ типа данных.

Строки 3—9 Выбрать *Sortmap* (*Отображение сорта*) и *Equations*$_1$ (*Равенства*$_1$) определения родительского (объемлющего) типа данных. Если в *Type-union*$_1$ (*Объединение-типов*$_1$) нет *Type-identifier*$_1$ (*Идентификатора-типа*$_1$), тогда это уровень системы, в противном случае родительские *Sortmap* и *Equations*$_1$ находятся в дескрипторе родителя.

Строки 10—12 Образовать дескрипторы для всех литералов, определенных в определении типа данных. Они рассматриваются как операции без каких-либо аргументов.

Строки 13—16 Образовать дескрипторы для всех операций, определенных в определении типа данных, добавить их к существующему *Entity-dict* и предоставить им класс объектов VALUE (ЗНАЧЕНИЕ).

Строка 18 Образовать дескрипторы *(sortmap)* для всех сортов, определенных в определении типа данных.

Строки 19—20 Образовать начальное *Sortmap*, состоящее из количества эквивалентных классов для каждого *сорта*, равного количеству термов для *сорта*, то есть каждый эквивалентный класс содержит один и только один терм. Домен *sortmap* — это локально определенные сорта (*sorts*) и сорта объемлющей структурной единицы (dom *psmap*).

Строка 22 Модифицировать *Sortmap* согласно равенствам.

Строки 24—25 Пассивные термы, являющиеся идентификаторами литералов, и ошибочный терм не должны принадлежать к одному эквивалентному классу.

Строка 23 Скорректировать *entitydict* с помощью дескрипторов локальных сортов и определения типов данных.

Строка 27 Если тип данных согласуется с типом данных объемлющей структурной единицы (то есть значения не изменены), тогда возвратить скорректированный *entitydict*.

is-equivalent(*lterm*, *rterm*, *level*)(*dict*) ≜ **(5.2.2)**

```
1  (let (id, TYPE) ∈ dom dict be s.t. s-Qualifier1(id) = level in
2   let mk-TypeDD(sortmap, ) = dict((id, TYPE)) in
3   let termsets = union rng sortmap in
4   let ltermset ∈ termsets be s.t. lterm ∈ ltermset,
5       rtermset ∈ termsets be s.t. rterm ∈ rtermset in
6   if mk-Error-term1() ∈ (ltermset ∪ rtermset)
7     then exit("§5.4.1.7: Применение операции эквивалентно терму-ошибке")
8     else ltermset = rtermset)
```

type: $Ground\text{-}term_1\ Ground\text{-}term_1\ Qualifier_1 \rightarrow Entity\text{-}dict \rightarrow Bool$

Назначение Проверить принадлежность двух термов к одному эквивалентному классу.

Параметры

lterm, rterm Два терма, подвергаемые проверке на эквивалентность.

level Квалификатор, обозначающий структурную единицу, в контексте которой должна проводиться проверка.

Результат True, если они эквивалентны.

Алгоритм

Строка 1 Выбрать *Идентификатор-типа*$_1$ структурной единицы, обозначенный *level*.

Строки 2—3 Образовать множество всех эквивалентных классов для всех сортов, доступных в структурной единице. (Во всей системе все эквивалентные классы разъединены.)

Строки 4—5 Выбрать эквивалентный класс для *lterm* и эквивалентный класс для *rterm*.

Строка 6 Ни один из этих эквивалентных классов не может содержать терм-ошибку.

Строка 8 Возвратить *true*, если *lterm* и *rterm* принадлежат к одному эквивалентному классу.

reduce-term(*sortid*, *term*, *level*)(*dict*) ≜ (5.2.3)

```
1  if term = nil then
2    nil
3  else
4   (let sortid' = if is-SortDD(dict((sortid, SORT))) then
5                    sortid
6                  else
7                    s-Parent-sort-identifier1(dict((sortid, SORT))) in
8    let mk-SortDD(tpid) = dict((sortid', SORT)) in
9    let mk-TypeDD(sortmap, ) = dict((tpid, TYPE)) in
10   let (tpid', q) ∈ dom dict be s.t. q = TYPE ∧ s-Qualifier1(tpid') = level in
11   let mk-TypeDD(sortmap', ) = dict((tpid', q)) in
12   let vset ∈ sortmap'(sortid') be s.t. term ∈ vset in
13   let vset' ∈ sortmap(sortid') be s.t. vset' ⊆ vset in
14   let term' ∈ vset' in
15   term')
```

type: *Sort-reference-identifier*$_1$ [*Ground-term*$_1$] *Qualifier*$_1$ → *Entity-dict* → [*Ground-term*$_1$]

Назначение Преобразовать терм в другой терм того же эквивалентного класса так, чтобы выбранный терм содержал только литералы и операции, определенные в структурной единице, определяющей *sortid*. Это преобразование требуется всякий раз при передаче значения к необъемлемой структурной единице. (Для простоты преобразование делается каждый раз, когда значение передается (может передаваться) в другую структурную единицу, то есть в присваивание, вычисление фактических параметров и т. д.)

Параметры

sortid	*Идентификатор-ссылки-на-сорт*$_1$, обозначающий сорт преобразуемого терма.
term	Преобразуемый *Term*$_1$ (*Терм*$_1$).
level	Структурная единица, в которой используется *терм*.

Результат Новый терм. Результатом является nil, если *терм* является nil (*term* является nil, если *терм-сокращения* используется в функции вычисления фактических параметров, где фактический параметр не специфицируется).

Алгоритм

Строка 1	Если *Term*$_1$ не специфицируется, то возвратить nil.
Строки 4—7	Если *sortid* обозначает *Syntype-identifier*$_1$ (*Идентификатор-подтипа*$_1$) (*SyntypeDD*), тогда выбрать родительский *Sort-identifier*$_1$ (*Идентификатор-сорта*$_1$).
Строка 8	Выбрать *Type-identifier*$_1$ (*Идентификатор-типа*$_1$) (*tpid*), определяющий сорт.
Строка 9	Выбрать Отображение сорта (*sortmap*) типа.
Строки 10—11	Выбрать *Отображение сорта* (*sortmap'*) типа, определенного в структурной единице, где используется *терм*.
Строка 12	Выбрать эквивалентный класс, содержащий *терм* среди эквивалентных классов, принадлежащих типу, определенному в структурной единице, где используется *терм*.
Строка 13	Выбрать эквивалентный класс определяющей структурной единицы сорта (*sortid'*), содержащегося в другом эквивалентном классе.
Строка 14	Возвратить произвольный элемент из этого эквивалентного класса.

is-wf-values(*survmap*, *vmap*)(*dict*) ≜ (5.2.4)

```
1  if survmap = [] then
2    is-wf-bool-and-pid(dict, vmap(dict(PIDSORT)))
3   else
4    (∀id ∈ dom survmap)
5     ((let survset = survmap(id),
6           vset = vmap(id) in
7       (∀class ∈ vset)((∃!class' ∈ survset)(class' ⊆ class))))
```

type: *Sortmap Sortmap* → *Entity-dict* → *Bool*

Назначение Проверить, является ли множество эквивалентных классов объемлющей структурной единицы тем же самым множеством для объемлемой структурной единицы.

Параметры

survmap — *Отображение-сорта*, обозначающее значения объемлющей структурной единицы.

vmap — *Отображение-сорта*, обозначающее значения объемлемой структурной единицы.

Результат True в случае успеха.

Алгоритм

Строки 1—2 — Если *Sortmap* (*Отображение-сорта*) объемлющей структурной единицы пусто, то это — уровень системы, и значения True и False (Булевского) сорта должны принадлежать различным эквивалентным классам и должно иметься бесконечное число Pid-значений. В случае вложенной структурной единицы эти проверки относятся к особому случаю более общего условия, проверяемого с помощью *is-wf-values*.

Строки 4—7 — Для всех *Идентификаторов-сорта*$_1$, принадлежащих к объемлющей структурной единице, условие должно выполняться.

Строка 5 — Выбрать эквивалентные классы, относящиеся к объемлющей структурной единице имеющегося сорта.

Строка 6 — Выбрать эквивалентные классы, относящиеся к объемлемой структурной единице имеющегося сорта.

Строка 7 — Для каждого эквивалентного класса в объемлемой структурной единице необходимо, чтобы она содержала все термы единственного эквивалентного класса объемлющей структурной единицы.

make-equivalent-classes(*sort*)(*dict*) ≜ (5.2.5)

```
1  (let termset = {term ∈ Ground-term1 | is-of-this-sort(sort, term)(dict)} in
2   let classes = {{term} | term ∈ termset} ∪ {{mk-Error-term1()}} in
3   [sort ↦ classes])
```

type: *Sort-identifier*$_1$ → *Entity-dict* → *Sortmap*

Назначение Образовать все возможные *Терм*$_1$*ы* для *Идентификатора-сорта*$_1$, обозначенного как *sort*, где *Терм*$_1$*ы* вводятся в различные эквивалентные классы.

Результат Вклад *Sortmap* (*Отображения-сорта*) для *sort* (*сорта*).

Алгоритм

Строка 1 — Образовать множество *Ground-Term*$_1$*s* (*Пассивных-терм*$_1$*ов*), состоящее из всех возможных термов сорта.

Строка 2 — Поместить все термы в множества различных эквивалентных классов, а терм-ошибку — в его собственный эквивалентный класс.

is-of-this-sort(*sort*, *t*)(*dict*) ≜ (5.2.6)

```
1  (let mk-Ground-term1(term) = t in
2   if is-Identifier1(term) then
3     (let entry = (term, VALUE) in
4      entry ∈ dom dict ∧
5      is-OperatorDD(dict(entry)) ∧
6      s-Argument-list(dict(entry)) = ⟨⟩ ∧
7      s-Result(dict(entry)) = sort)
8    else
9     if is-Conditional-term1(term) then
10      false
11     else
12      (let (id, arglist) = term in
13       let entry = (id, VALUE) in
14       if entry ∈ dom dict ∧ is-OperatorDD(dict(entry)) then
15         (let mk-OperatorDD(sortlist, result) = dict(entry) in
16          len arglist = len sortlist ∧
17          result = sort ∧
18          (∀i ∈ Ind arglist)(is-of-this-sort(sortlist[i], arglist[i])(dict)))
19        else
20         false))
```

type: *Sort-identifier*$_1$ *Ground-term*$_1$ → *Entity-dict* → *Bool*

Назначение Проверить, является ли данный *Term*$_1$ *t* термом сорта, обозначенного как *sort*.

Алгоритм

Строка 2 Если терм является идентификатором, то

Строка 4 идентификатор должен находиться в *entitydict*

Строки 5—6 в качестве литерала (то есть как операция без аргументов),

Строка 7 а сорт результата должен быть *sort*.

Строка 9 Если терм является условным термом, то он не представляет значение (но представляет следствие и альтернативу в условном терме).

Строка 12 Если терм — это терм операции и

Строка 14 если идентификатор терма может находиться в *entitydict* и обозначать операцию, тогда

Строка 16 количество аргументов дескриптора должно быть равным количеству аргументов, имеющихся в терме,

Строка 17 сорт результата, содержащийся в дескрипторе, должен быть *sort*,

Строка 18 а каждый терм аргумента, находящегося в *term*, должен быть сортом, соответствующим списку сортов аргументов, содержащемуся в дескрипторе.

eval-equations(*sortmap*, *equations*)(*dict*) ≜ (5.2.7)

```
1  (let trueterm = dict(TRUEVALUE),
2       falseterm = dict(FALSEVALUE) in
3   let quanteq = {eq ∈ equations | is-Quantified-equations1(eq)} in
4   let rest = equations \ quanteq in
5   let unquant = union {eval-quantified-equation(sortmap, eq) | eq ∈ quanteq} in
6   let rest' = expand-conditional-term-in-equations(rest ∪ unquant, trueterm, falseterm) in
7   let rest'' =
8       union {if is-Conditional-equation1(eq)
9                 then expand-conditional-term-in-conditions({eq}, trueterm, falseterm)
10                else {eq} | eq ∈ rest'} in
11  let unquanteqs = {eq ∈ rest'' | is-Unquantified-equation1(eq)},
12      condeqs = {eq ∈ rest'' | is-Conditional-equation1(eq)} in
13  let sortmap' = eval-unquantified-equations(sortmap, unquanteqs) in
14  eval-conditional-equations(sortmap', condeqs))
```

type: *Sortmap Equations*$_1$ → *Entity-dict* → *Sortmap*

Назначение Преобразовать число эквивалентных классов сортов, распознаваемых в данной структурной единице, в соответствии с множеством равенств.

Параметры

sortmap — *Отображение-сорта*, содержащее эквивалентные классы, которые должны быть преобразованы.

equations — Множество равенств.

Результат Модифицированное *Sortmap* (*Отображение-сортов*).

Алгоритм

Строки 1—2 — Выбрать из *Entity-dict* представления в AS$_1$ булевских литералов True и False.

Строка 3 — Выбрать квантифицируемые равенства.

Строка 5 — Преобразовать множество квантифицированных равенств во множество неквантифицированных равенств.

Строка 6 — Преобразовать все условные термы модифицированного множества равенств (за исключением тех, которые содержатся в условиях условных равенств) во множество условных равенств.

Строки 7—10 — Преобразовать все условные равенства, содержащие условные термы во множестве условий, во множество условных равенств без каких-либо условных термов, содержащихся в условиях (см. пример в тексте вслед за функцией *expand-conditional-term-in-conditions*).

Строки 11—12 — Разделить полученное множество равенств (*rest''*) на множество неквантифицированных равенств и множество условных равенств.

Строка 13 — Модифицировать *sortmap* в соответствии с множеством неквантифицированных равенств.

Строка 14 — Возвратить *отображение сорта*, являющееся *sortmap*, модифицированным в соответствии с множеством условных равенств.

eval-unquantified-equations(*sortmap*, *equations*) ≜ (5.2.8)

```
 1  (if equations = {} then
 2     sortmap
 3    else
 4     (let eq ∈ equations in
 5      let mk-Unquantified-equation1(lterm, rterm) = eq in
 6      let sort ∈ dom sortmap be s.t. (∃termset ∈ sortmap(sort))(lterm ∈ termset) in
 7      let termset1 be s.t. termset1 ∈ sortmap(sort) ∧ lterm ∈ termset1 in
 8      let termset2 be s.t. termset2 ∈ sortmap(sort) ∧ rterm ∈ termset2 in
 9      if termset1 = termset2 then
10        eval-unquantified-equations(sortmap, equations \ {eq})
11       else
12        (let newset = sortmap(sort) \ {termset1, termset2} ∪ {termset1 ∪ termset2} in
13         let sortmap' = sortmap + [sort ↦ newset] in
14         let sortmap'' = eval-deduced-equivalence(sortmap') in
15         eval-unquantified-equations(sortmap'', equations \ {eq}))))
```

type: *Sortmap Equations*$_1$ → *Sortmap*

Назначение Модифицировать *Отображение сортов* (эквивалентные классы) в соответствии с *equations*.

Параметры

sortmap	Модифицируемое *отображение сортов*.
equations	Множество неквантифицированных равенств.

Алгоритм

Строка 1	Возврат модифицированного параметра *sortmap* после выполнения.
Строки 4—5	Выбор двух *термов* из одного из (остальных) равенств.
Строка 6	Выбрать сорт терма *lterm* (то же самое, что и сорт *rterm*).
Строка 7	Выбрать эквивалентный класс, содержащий *lterm*.
Строка 8	Выбрать эквивалентный класс, содержащий *rterm*.
Строка 9	Если термы обозначают тот же эквивалентный класс, тогда не корректировать *sortmap*, иначе
Строка 12	определить новое множество эквивалентных классов, в котором отождествляются два эквивалентных класса.
Строка 13	Модифицировать параметр *sortmap*, содержащий новое множество эквивалентных классов.
Строка 14	Сократить количество эквивалентных классов, используя полученную из равенства информацию.
Строка 15	Повторить все действия для остальных равенств.

eval-deduced-equivalence(*sortmap*) ≜ (5.2.9)

```
 1  if (∃class1, class2, class3 ∈ union rng sortmap)
 2     (class1 ≠ class2 ∧
 3      (∃term1, term2 ∈ class3)((∃term ∈ class1)(replace-term(term, term1, term2) ∈ class2))) then
 4    (let (class1, class2, class3) be s.t. {class1, class2, class3} ⊂ union rng sortmap ∧
 5          class1 ≠ class2 ∧
 6          (∃term1, term2 ∈ class3)((∃term ∈ class1)(replace-term(term, term1, term2) ∈ class2)) in
 7     let sort be s.t. {class1, class2} ⊂ rng sortmap(sort) in
 8     let classes = sortmap(sort) in
 9     let classes' = classes \ {class1, class2} ∪ {class1 ∪ class2} in
10     let sortmap' = sortmap + [sort ↦ classes'] in
11     eval-deduced-equivalence(sortmap'))
12   else
13    sortmap
```

type: *Sortmap* → *Sortmap*

Назначение Сократить количество эквивалентных классов сортов, используя информацию о том, что два терма сорта находятся в одном эквивалентном классе.

Параметры

sortmap — *Отображение сорта*, содержащего модифицируемые эквивалентные классы.

Результат *Отображение сорта*, где сокращено количество эквивалентных классов для некоторых сортов.

Алгоритм

Строка 1 — Если существуют три эквивалентных класса *class*1, *class*2, *class*3 в *Отображении сорта*, таких что *class*1 и *class*2 не пересекаются (*class*3 может быть равен *class*1 или *class*2 или может обозначать другой эквивалентный класс, и даже класс другого сорта), и существуют два терма (*term*1 и *term*2) в *class*3 (*классе*3), таких что при замене *term*1 на *term*2 в терме (*term*), взятом из *class*1, получается терм в *class*2, тогда

Строки 4—13 — *class*1 и *class*2 сливаются в один эквивалентный класс.

Строки 4—6 — Пусть *class*1, *class*2, *class*3 обозначают три таких эквивалентных класса.

Строка 7 — Пусть *sort* обозначает сорт классов *class*1 и *class*2; *class*1 и *class*2 не могут быть классами разного сорта, поскольку в этом случае не удовлетворяются положения алгоритма, изложенные в строках 1—3.

Строки 8—10 — Образовать новый *sortmap*, где соединяются два эквивалентных класса сорта.

Строка 11 — Повторять все действия (с модифицированным *sortmap*) до тех пор, пока больше нельзя соединять эквивалентные классы.

replace-term(*term*, *oldterm*, *newterm*) $\triangleq$ (5.2.10)

```
1  if term = oldterm then
2    newterm
3  else
4   if is-Identifier1(term) then
5     term
6    else
7     (let (opid, arglist) = term in
8      if (∃i ∈ ind arglist)(replace-term(arglist[i], oldterm, newterm) ≠ arglist[i]) then
9        (let i ∈ ind arglist be s.t. replace-term(arglist[i], oldterm, newterm) ≠ arglist[i] in
10        let arglist' = ⟨arglist[n] | 1 ≤ n < i⟩ ⌢
11            ⟨replace-term(arglist[i], oldterm, newterm)⟩ ⌢
12            ⟨arglist[n] | i < n ≤ len arglist⟩ in
13        (opid, arglist'))
14       else
15        term)
```

type: *Ground-term*$_1$ *Ground-term*$_1$ *Ground-term*$_1$ $\rightarrow$ *Ground-term*$_1$

Назначение Замена вхождения *oldterm* (*старого терма*) в терм *newterm* (новым термом) и возврат модифицированного терма.

Алгоритм

Строка 1	Если входной терм равен *oldterm*, возвратить новый терм.
Строка 4	Если терм является идентификатором (и отличается от *oldterm*), тогда никакая замена не производится, в противном случае
Строка 7	терм является термом операции (условные термы не могут обладать вхождением, поскольку *терм* берется из эквивалентного класса). Пусть *op* обозначает идентификатор операции и пусть *arglist* обозначает список аргументов.
Строка 8	Если существует аргумент, содержащий *oldterm*, тогда
Строка 9	пусть *i* обозначает индекс аргумента, содержащего *oldterm*.
Строки 10—12	Образовать список аргументов, где вхождение *oldterm* в элемент *i* заменено на *newterm*.
Строка 13	Возвратить модифицированный терм.
Строка 15	Если *oldterm* не включен в список аргументов, то терм не меняется.

eval-quantified-equation(*sortmap*, *quanteqs*) $\triangleq$ (5.2.11)

```
1  (let mk-Quantified-equations1(nmset, sortid, equations) = quanteqs in
2   let nm ∈ nmset in
3   let mk-Identifier1(level, snm) = sortid in
4   let valueid = mk-Identifier1(level ⌢ (mk-Sort-qualifier1(snm)), nm) in
5   let allterms = union sortmap(sortid) \ {mk-Error-term1()} in
6   let equations' = union {union {insert-term(sortmap, eq, valueid, term) | term ∈ allterms} |
7                           eq ∈ equations} in
8   if nmset = {nm} then
9     equations'
10   else
11    (let quanteq = mk-Quantified-equations1(nmset \ {nm}, sortid, equations') in
12     eval-quantified-equation(sortmap, quanteq)))
```

type: *Sortmap Quantified-equations*$_1$ $\rightarrow$ *Equations*$_1$

Назначение Разложить квантифицированное равенство на множество неквантифицированных равенств.

Параметры

sortmap	Sortmap (Отображение сорта) определения объемлющего типа данных, когда термы (еще) находятся в различных эквивалентных классах.
quanteqs	Квантифицированные равенства.

Результат Полученное множество неквантифицированных равенств.

Алгоритм

Строка 2	Взять одно из имен значений в квантифицированном равенстве.
Строка 4	Сделать идентификатор значения, соответствующий имени значения.
Строка 5	Сделать множество (*allterms*) всех возможных термов (за исключением *Error-term*$_1$ (*Терма-ошибки*$_1$)) для квантифицирующего сорта.
Строки 6—7	Образовать множество неквантифицированных равенств, содержащихся в квантифицированном равенстве, заменить идентификатор значения во множестве равенств на каждый терм в *allterms*.
Строка 8	Если в равенствах заменено имя каждого значения, тогда возвратить равенства (*equations'*), иначе
Строки 11—12	делать то же самое для остальных имен значений в квантифицированном равенстве.

insert-term(*sortmap*, *equation*, *vid*, *term*) $\triangleq$ **(5.2.12)**

```
1  cases equation:
2   (mk-Unquantified-equation1(term1, term2)
3      → {mk-Unquantified-equation1(insert-term-in-term(term1, vid, term),
4                                   insert-term-in-term(term2, vid, term))},
5    mk-Quantified-equations1(, , )
6      → (let equations = eval-quantified-equation(sortmap, equation) in
7         union {insert-term(sortmap, eq, vid, term) | eq ∈ equations}),
8    mk-Conditional-equation1(eqs, eq)
9      → (let mk-Unquantified-equation1(term1, term2) = eq,
10           eqs' = union {insert-term(sortmap, e, vid, term) | e ∈ eqs} in
11        let eq' = mk-Unquantified-equation1(insert-term-in-term(term1, vid, term),
12                                            insert-term-in-term(term2, vid, term)) in
13        {mk-Conditional-equation1(eqs', eq')}),
14   T → {equation})
```

type: *Sortmap Equation*$_1$ *Value-identifier*$_1$ *Ground-term*$_1$ → *Equations*$_1$

Назначение Заменить имя значения на *Пассивный-терм*$_1$ в равенстве, объемлемом квантифицированным равенством.

Параметры

sortmap	*Sortmap*, используемое, когда равенство (в свою очередь) содержит квантифицированные равенства.
equation	Модифицируемое равенство.
vid	Идентификатор значения, который должен быть заменен.
term	*Term*$_1$ (*Терм*$_1$), который должен быть заменен на *vid*.

Результат Множество равенств, содержащее модифицированное равенство. Если равенство — это квантифицированное равенство, то множество может состоять из нескольких равенств.

Алгоритм

Строки 2—4	Если имеем неквантифицированное равенство, тогда заменить *vid* на *term* в двух содержащихся термах (*term*1, *term*2).

Строки 5—7 Если имеем квантифицированное равенство, то сначала разложить его на множество неквантифицированных равенств, а затем заменить идентификатор значения в каждом равенстве множества.

Строки 8—13 Если имеем условное равенство, то заменить идентификатор значения на терм в каждом равенстве в ограничении и в ограниченном равенстве, образовать и возвратить множество, содержащее модифицированное условное равенство.

Строка 14 Если имеем неформальный текст, то его не обрабатываем.

$insert\text{-}term\text{-}in\text{-}term(term, vid, vterm) \triangleq$ (5.2.13)

```
 1  (if is-Ground-term1(term) ∨ is-Error-term1(term) then
 2     term
 3    else
 4     (let mk-Composite-term1(term') = term in
 5      if is-Identifier1(term') then
 6        if term' = vid then vterm else term
 7       else
 8        if is-Conditional-term1(term') then
 9          (let mk-Conditional-term1(cond, t1, t2) = term' in
10           let cond' = insert-term-in-term(cond, vid, vterm),
11               t1' = insert-term-in-term(t1, vid, vterm),
12               t2' = insert-term-in-term(t2, vid, vterm) in
13           let term'' = mk-Conditional-term1(cond', t1', t2') in
14           if is-Ground-term1(cond') ∧ is-Ground-term1(t1') ∧ is-Ground-term1(t2') then
15             mk-Ground-term1(term'')
16            else
17             mk-Composite-term1(term''))
18         else
19          (let (opid, arglist) = term' in
20           let arglist' =
21               ⟨insert-term-in-term(arglist[i], vid, vterm) | 1 ≤ i ≤ len arglist⟩ in
22           if (∃arg ∈ elems arglist)(is-Composite-term1(arg)) then
23             mk-Composite-term1((opid, arglist'))
24            else
25             mk-Ground-term1((opid, arglist')))))
```

type : $Term_1\ Value\text{-}identifier_1\ Ground\text{-}term_1 \rightarrow Term_1$

Назначение Заменить идентификатор значения (*vid*) на (пассивный) терм (*vterm*) в терме (*term*).

Параметры

term $Term_1$ ($Терм_1$), у которого должен быть заменен идентификатор значения.

vid Заменяемый идентификатор значения.

vterm $Term_1$ ($Терм_1$), который должен быть введен вместо идентификатора значения.

Результат Модифицированный терм.

Алгоритм

Строка 1 Если имеем пассивный терм или терм-ошибку, то не модифицировать его.

Строки 5—6 Если имеем идентификатор, равный *vid*, тогда возвратить новый терм, иначе не модифицировать его.

Строки 8—13 Если имеем условный терм, тогда создать условный терм, где включения *vid* в три содержащихся терма заменяется на *vterm*.

Строки 14—17 Если все три содержащихся терма стали пассивными термами, тогда возвратить новый условный терм в качестве пассивного терма, иначе возвратить его в качестве составного терма.

Строки 19—25 В противном случае *терм* должен быть термом операции, в ходе которой *vid* в термах аргументов заменяется на *vterm*, и, если все модифицированные термы аргументов становятся пассивными термами, возвратить новый терм-операцию в качестве пассивного терма, в противном случае возвратить его как составной терм.

expand-conditional-term-in-equations(*equations*, *trueterm*, *falseterm*) ≙ **(5.2.14)**

```
1  (if equations = {} then
2     {}
3   else
4     (let eq ∈ equations in
5      let (condset, eq') =
6          cases eq:
7           (mk-Unquantified-equation1(,)
8              → ({}, eq),
9            mk-Conditional-equation1(condeq, eqs)
10             → (condeq, eqs)) in
11     let mk-Unquantified-equation1(t1, t2) = eq' in
12     let (t1', t1'', cond1) = expand-conditional-in-terms(t1),
13         (t2', t2'', cond2) = expand-conditional-in-terms(t2) in
14     if cond1 = nil ∧ cond2 = nil then
15       {eq} ∪ expand-conditional-term-in-equations(equations \ {eq}, trueterm, falseterm)
16      else
17       (let (cond, term, nterm1, nterm2) be s.t. (cond, term, nterm1, nterm2) ∈
18            {(cond2, t1, t2', t2''), (cond1, t2, t1', t1'')} ∧ cond ≠ nil in
19        let eq1 = mk-Unquantified-equation1(cond, trueterm),
20            eq2 = mk-Unquantified-equation1(cond, falseterm) in
21        let condeq1 =
22            mk-Conditional-equation1(condset ∪ {eq1}, mk-Unquantified-equation1(term, nterm1)),
23            condeq2 =
24            mk-Conditional-equation1(condset ∪ {eq2}, mk-Unquantified-equation1(term, nterm2)) in
25        let equations' = equations ∪ {condeq1, condeq2} \ {eq} in
26        expand-conditional-term-in-equations(equations', trueterm, falseterm))))
```

type: *Equations*$_1$ *Literal-operator-identifier*$_1$ *Literal-operator-identifier*$_1$ → *Equations*$_1$

Назначение Заменить каждый *Conditional-term*$_1$ (*Условный-терм*$_1$) на два *Conditional-equation*$_1$*s* (*Условных-равенства*$_1$).

Пример:

равенство

если a, тогда b, иначе c==d

разлагается на

a == True (Истина) ==> b == d;
a == False (Ложь) ==> c == d;

Параметры

equations Множество заменяемых равенств.

trueterm, *falseterm* Два пассивных терма, обозначающих булевские True (Истина) и False (Ложь).

Результат Модифицированное множество равенств, не содержащих *Conditional-terms* (*Условные-термы*).

Алгоритм

Строка 1 Если множество равенств пусто, то ничего не возвращается.

Строки 4—9 Взять равенство из множества и извлечь множество ограничений (*cond-set*) и ограниченное равенство (*eq'*). Если это неквантифицированное равенство, то множество ограничений пусто.

Строки 12—13 Модифицировать термы в ограниченном равенстве; *cond*1 и *cond*2 — проверяемые условия. Условие равно nil, если терм не содержит никаких условных термов; $t1'$, $t2'$ являются исходными термами ($t1$, $t2$), в которых условный терм заменяется частью «then» («тогда») условного терма, а $t1''$, $t2''$ являются исходными термами, в которых условный терм заменяется частью «else» («иначе») условного терма.

Строки 14—15 Если ни один из двух термов не содержал условных термов, тогда равенство не менять и продолжить действия с другим равенством из *equations*.

Строка 17 Выбрать один из двух рассматриваемых термов. При этом обращении второй терм не будет меняться.

Строки 19—20 Образовать два неквантифицированных равенства, которые должны выполняться для двух модифицированных равенств.

Строки 21—23 Образовать два условных равенства, где *eq*1 по отношению к *eq*2 представляет собой дополнительное условие; (*condeq*1) содержит равенство, где один из исходных термов ($t1$ или $t2$) заменяется на терм, содержащий часть «then» («тогда»), а (*condeq*2) содержит равенство, где один из исходных термов заменяется термом, содержащим часть «else» («иначе»).

Строка 26 Включить два новых условных равенства во множество остальных рассматриваемых равенств (так как один из термов в *eq* не разложен и так как разложенный терм может содержать дополнительные условные термы).

expand-conditional-in-terms$_1(t)$ $\triangleq$ (5.2.15)

```
 1  (if is-Error-term1(t) then
 2     (t, t, nil)
 3   else
 4     (let mk-Ground-term1(term) = t in
 5       cases term:
 6        (mk-Identifier1(,)
 7           → (t, t, nil),
 8         mk-Conditional-term1(cond, t1, t2)
 9           → (t1, t2, cond),
10         (id, arglist)
11           → if (∃arg ∈ elems arglist)
12                ((let (,,cond) =
13                     expand-conditional-in-terms(arg) in
14                  cond ≠ nil)) then
15               (let (i, t1, t2, cond) be s.t. i ∈ inds arglist ∧
16                    cond ≠ nil ∧
17                    expand-conditional-in-terms(arglist[i]) = (t1, t2, cond) in
18                let arglist' =
19                    ⟨arglist[n] | 1 ≤ n < i⟩ ⌢ ⟨t1⟩ ⌢ ⟨arglist[n] | i < n ≤ len arglist⟩,
20                    arglist'' =
21                    ⟨arglist[n] | 1 ≤ n < i⟩ ⌢ ⟨t2⟩ ⌢ ⟨arglist[n] | i < n ≤ len arglist⟩ in
22                (mk-Ground-term1((id, arglist')), mk-Ground-term1((id, arglist'')), cond))
23              else
24               (t, t, nil))))
```

type: $Term_1 \rightarrow Term_1\ Term_1\ [Ground\text{-}term_1]$

Назначение Разделить терм (t) на три терма. Если t не содержит условного терма, тогда первые два терма не принимаются во внимание, а третий терм есть *nil*. В противном случае результатом является t, модифицируемый и содержащий часть «then» («тогда»), модифицируемый и содержащий часть «else» («иначе»), и булевский условный терм.

Результат Три новых терма.

Алгоритм

Строки 1—6 Если имеем терм-ошибку, тогда не модифицировать его и указать, что он не содержит условного терма при возвращении nil в качестве условного терма.

Строка 8 Если имеем условный терм, то возвратить три его части.

Строки 10—14 Если имеем терм операции и один из его аргументов содержит условный терм, тогда

Строки 15—17 взять терм аргумента, содержащий условный терм, и разделить его; i является позицией в списке аргументов.

Строки 18—20 Образовать списки аргументов, соответствующие части «then» ($arglist'$) и части «else» («иначе») ($arglist''$) и

Строка 22 возвратить два терма операции, соответствующих части «then», части «else», и булевское условие в условный терм в аргументе.

expand-conditional-term-in-conditions(*equations*, *trueterm*, *falseterm*) ≜ (5.2.16)

```
1  (if equations = {} then
2     {}
3   else
4    (let eq ∈ equations in
5     let mk-Conditional-equation1(condset, eq') = eq in
6     if (∃cond ∈ condset)
7        ((let mk-Unquantified-equation1(t1, t2) = cond in
8          let (,,cond1) =
9              expand-conditional-in-terms(t1),
10             (,,cond2) =
11             expand-conditional-in-terms(t2) in
12         cond1 ≠ nil ∨ cond2 ≠ nil)) then
13       (let (condeq, cond, term, nterm1, nterm2) be s.t. condeq ∈ condset ∧
14          (let mk-Unquantified-equation1(t1, t2) =
15              condeq in
16           let (t1', t1'', cond1) =
17              expand-conditional-in-terms(t1),
18              (t2', t2'', cond2) =
19              expand-conditional-in-terms(t2) in
20           (cond, term, nterm1, nterm2) = (if cond1 = nil
21                                             then (cond2, t1, t2', t2'')
22                                             else (cond1, t2, t1', t1''))) in
23        let eq1 = mk-Unquantified-equation1(cond, trueterm),
24            eq2 = mk-Unquantified-equation1(cond, falseterm) in
25        let condset' = condset \ {condeq} ∪ {eq1, mk-Unquantified-equation1(term, nterm1)},
26            condset'' = condset \ {condeq} ∪ {eq2, mk-Unquantified-equation1(term, nterm2)} in
27        let equations' = equations \ {eq} ∪ {mk-Conditional-equation1(condset', eq'),
28                                            mk-Conditional-equation1(condset'', eq')} in
29        expand-conditional-term-in-conditions(equations', trueterm, falseterm))
30      else
31       {eq} ∪ expand-conditional-term-in-conditions(equations \ {eq}, trueterm, falseterm)))
```

type: *Conditional-equation*$_1$ *Literal-operator-identifier*$_1$ *Literal-operator-identifier*$_1$ → *Equations*$_1$

Назначение Разделить условные равенства из *equations* на два условных равенства, если они содержат какие-либо условные термы в *Restriction*$_1$ (*Ограничении*$_1$).

Пример:

равенство

если b тогда c иначе d == e ==> f == g

разлагается на

b == True (Истина), c == e ==> f == g;
b == False (Ложь), d == e ==> f == g

Параметры

equations	Множество условных равенств.
treuterm, *falseterm*	Два пассивных терма, обозначающих булевские True (Истина) и False (Ложь).

Результат Разложенное множество равенств.

Алгоритм

Строка 1	Возвратить пустое множество после выполнения.
Строки 4—12	Взять из множества условное равенство и если оно не содержит условного терма в части ограничения, то продолжить рассмотрение остальных равенств из множества (строка 31).
Строки 13—21	Выбрать из множества ограничений неквантифицированное равенство, содержащее условный терм (*condeq*), условие в условном терме (*cond*), версию «then» («тогда») терма в неквантифицированном равенстве, содержащем условный терм (*nterm*1), версию «else» («иначе») терма в неквантифицированном равенстве, содержащем условный терм (*nterm*2), и другой терм неквантифицированного равенства (*term*).
Строки 23—24	Образовать два дополнительных ограничения, включаемых в соответствующие множества ограничений.
Строки 25—26	Образовать два модифицированных множества ограничений.
Строка 27	Заменить прежнее условное равенство на два новых условных равенства во множестве равенств.
Строка 29	Повторить все действия с модифицированным множеством равенств.

eval-conditional-equations(*sortmap*, *condequations*) ≜ (5.2.17)

```
1  if (∃condeq ∈ condequations)(restriction-holds(condeq, sortmap)) then
2    (let condeq ∈ condequations be s.t. restriction-holds(condeq, sortmap) in
3     let mk-Conditional-equation1(, eq) = condeq in
4     let sortmap' = eval-unquantified-equations(sortmap, {eq}) in
5     eval-conditional-equations(sortmap', condequations \ {condeq}))
6   else
7    sortmap
```

type: *Sortmap Conditional-equation*$_1$-**set** → *Sortmap*

Назначение Сократить количество эквивалентных классов в *Sortmap* в соответствии с условными равенствами структурной единицы.

Параметры

sortmap	*Sortmap* (*Отображение сорта*).
condequations	Множество условных равенств.

Результат Модифицированное *Отображение сорта*.

Алгоритм

Строка 1	Если существует выполняемое условное равенство, тогда
Строка 2	пусть *condeq* обозначает условное равенство, которое выполняется.
Строки 3—4	Изменить в *Отображении сорта* приоритеты, выраженные в ограниченном равенстве (*eq*).
Строка 5	Повторять все действия до тех пор, пока в оставшемся множестве не будет больше условных равенств, которые выполняются.

restriction-holds(mk-*Conditional-equation*$_1$(*eqs*,), *sortmap*) ≜ (5.2.18)

```
1  (let termpairs = {{term1, term2} | (∃eq ∈ eqs)(mk-Unquantified-equation1(term1, term2) = eq)} in
2   (∀pairs ∈ termpairs)((∃class ∈ union rng sortmap)(pairs ⊂ class)))
```

type: *Conditional-equation*$_1$ *Sortmap* → *Bool*

Назначение Проверить, выполняется ли множество ограничений условного равенства.

Параметры

eqs	Множество ограничений.
sortmap	*Отображение сорта*, используемое для проверки выполнения ограничений.

Результат True (Истина) в случае успеха.

Алгоритм

Строка 1	Образовать множество пар термов, каждая из которых содержит терм с левой стороны и терм с правой стороны ограничения во множестве ограничений.
Строка 2	Ограничения выполняются, если для каждого ограничения выполняется следующее условие: правосторонний терм находится в том же эквивалентном классе, что и левосторонний терм.

(5.2.19)

$$is\text{-}wf\text{-}bool\text{-}and\text{-}pid(dict, pidvalueset) \triangleq$$

```
1  (let trueterm = dict(TRUEVALUE),
2       falseterm = dict(FALSEVALUE),
3       mk-Identifier1(level, ) = trueterm in
4   (∀s ∈ Sortmap)(s ⊂ pidvalueset ⊃ (∃n ∈ N1)(n > card s)) ∧
5   ¬is-equivalent(trueterm, falseterm, level)(dict))
```

type: *Entity-dict Term-class-set* → *Bool*

Назначение Проверить, принадлежит ли булевское True к тому же эквивалентному классу, что и булевское False, и существует ли бесконечное число PiD-значений.

Результат True, если эти условия удовлетворяются.

Алгоритм

Строки 1—2	Образовать пассивные термы, соответствующие литералам True (Истина) и False (Ложь).
Строка 3	Пусть *level* (*уровень*) обозначает его квалификатор.
Строка 4	Множество PiD-значений должно быть бесконечным, то есть для каждого (конечного) подмножества *s* множества *pidvalueset* должно существовать значение *n* из множества значений N_1, такое что *n* больше мощности подмножества.
Строка 5	Булевский литерал True (Истина) не должен принадлежать к тому же эквивалентному классу, что и булевский литерал False (Ложь).

5.3 Выбор Согласованного Подмножества

select-consistent-subset(*bset*, *subset*, *level*) ≜ (5.3.1)

```
1  if bset = {} then
2    {}
3   else
4    (let block ∈ bset in
5     let rest = select-consistent-subset(bset \ {block}, subset, level) in
6     let mk-Block-definition1(bnm, pdefs, , , , , sub) = block in
7     let pset = {mk-Identifier1(level ⌢ (mk-Block-qualifier1(bnm)), s-Process-name1(pdef)) |
8                 pdef ∈ pdefs} in
9     let bid = mk-Identifier1(level, bnm) in
10    if bid ∉ subset then
11      exit("§3.2.1: Подблок не находится в согласованном подмножестве")
12     else
13      if sub = nil then
14        rest ∪ pset
15       else
16        (let mk-Block-substructure-definition1(subnm, bset', , , , ) = sub in
17         let level' =
18             level ⌢ (mk-Block-qualifier1(bnm), mk-Block-substructure-qualifier1(subnm)) in
19         if mk-Identifier1(level, subnm) ∈ subset then
20           rest ∪ select-consistent-subset(bset', subset, level')
21          else
22           if pset = {} then
23             exit("§3.2.1: Блок-лист не содержит процессов")
24            else
25             rest ∪ pset))
```

type: *Block-definition*$_1$-set *Block-identifier*$_1$-set *Qualifier*$_1$ → *Process-identifier*$_1$-set

Назначение Проверить, что данное множество идентификаторов блоков и идентификаторов подструктур блоков обозначает согласованное подмножество, и возвратить идентификаторы процессов, содержащихся в согласованном подмножестве. Функция рекурсивно проходит через определение системы.

Параметры

bset Множество определений блоков в определении системы или в определении подструктуры блока.

subset (Предполагаемое) согласованное подмножество представлено множеством идентификаторов блоков и идентификаторов подструктур блоков.

level *Квалификатор*$_1$ структурной единицы, содержащей блоки (*bset*).

Алгоритм

Строка 1 Возвратить пустое множество идентификаторов процесса при выполнении.

Строка 4 Пусть *block* (*блок*) обозначает определение следующего рассматриваемого блока.

Строка 5 Выбрать согласованное подмножество для остальных определений блоков.

Строка 6 Пусть *bnm* обозначает имя блока, *pdefs* — множество определений процессов и *sub* — дополнительное определение подструктуры блока.

Строка 7 Пусть *pset* обозначает множество *Идентификаторов-процесса*$_1$, соответствующих *pdefs*.

Строки 9—11 Блок (или подблок) должен быть в согласованном подмножестве.

Строка 13 Если в блоке нет подструктуры, тогда

Строка 14 процессы блока находятся в согласованном подмножестве.

Строка 16 Если подструктура специфицирована, то пусть *subnm* обозначает ее имя, а *bset'* обозначает определения ее блоков.

Строки 17—20 Если подструктура находится в согласованном подмножестве, тогда рассмотреть блоки подструктуры, иначе

Строки 22—23 должно существовать по крайней мере одно определение процесса в блоке.

5.4 Образование Коммуникационных Путей

Данный раздел содержит функции изменения дескриптора каждого процесса (*ProcessDD*), содержащего множество *Reachabilities* (*Достижимостей*).

В каждой структурной единице, содержащей каналы между двумя блоками, образуются входящие пути каналов, исходящие пути, объединенные пути и в итоге модифицируются дескрипторы процессов, связанные с процессами, содержащимися в блоке от исходящих путей. Входящие и исходящие пути (частичные пути) содержат перед их объединением канал в одной из конечных точек и маршрут сигнала в другой конечной точке. Промежуточные идентификаторы являются идентификаторами всех подканалов; *make-structure-paths* является входной функцией, применяемой в *extract-dict*.

make-structure-paths(*bset*, *cset*, *level*)(*dict*) $\triangleq$ (5.4.1)

```
1  (if cset = {} then
2     dict
3   else
4    (let ch ∈ cset in
5     let mk-Channel-definition1(nm, mk-Channel-path1(b1, b2, ), ) = ch in
6     if (b1 = ENVIRONMENT ∨ b2 = ENVIRONMENT) ∧ ¬is-System-qualifier1(level[len level]) then
7       make-structure-paths(bset, cset \ {ch}, level)(dict)
8      else
9       (let chid = mk-Identifier1(level, nm) in
10       let (reachset1, dict') = out-going-paths(chid, b1, bset, ⟨⟩)(dict) in
11       let (reachset1', dict'') = out-going-paths(chid, b2, bset, ⟨⟩)(dict') in
12       let reachset2 = in-coming-paths(chid, b2, bset, ⟨⟩)(dict) in
13       let reachset2' = in-coming-paths(chid, b1, bset, ⟨⟩)(dict) in
14       if is-consistent-refinement(reachset1, reachset2) ∧
15         is-consistent-refinement(reachset2, reachset2') then
16          (let d = update-processd(reachset1, reachset2)(dict'') in
17           let d' = update-processd(reachset1', reachset2')(d) in
18           make-structure-paths(bset, cset \ {ch}, level)(d'))
19         else
20          exit("Z.100 §3.3 : Запрещенное уточнение канала"))))
```

type : *Block-definition*$_1$-**set** *Channel-definition*$_1$-**set** *Qualifier*$_1$ $\rightarrow$ *Entity-dict* $\rightarrow$ *Entity-dict*

Назначение Для всех каналов в структурной единице, соединенных с двумя блоками или соединенных с внешней средой системы, модифицировать *Reachabilities* для процессов передачи сигналов через каналы.

Параметры

bset Определение блоков.

cset Определение каналов структурной единицы.

level Квалификатор, обозначающий структурную единицу.

Результат *Entitydict*, в котором модифицируются соответствующие дескрипторы *ProcessDD*.

Алгоритм

Строка 1 Возвратить модифицированный *entitydict* при выполнении.

Строки 4—5 Взять определение канала из остального множества определений.

Строки 6—7 Если канал является подканалом, тогда не выполнять никаких действий, так как подканалами занимаются *in-coming-path* (*входящий-путь*) и *outgoing-path* (*исходящий-путь*).

Строки 10—11 Выбрать *Reachabilities* (*Достижимости*), содержащие те процессы, которые могут вести передачу через каналы и содержат соответствующий *Path* (*Путь*) и *entitydict*, корректируемый информацией *Reachabilities*, соответствующей локальным коммуникационным путям в $b1$ (строка 10) относительно $b2$ (строка 11).

Строки 12—13 Выбрать *Reachabilities*, содержащие те процессы в блоке $b2$ относительно $b1$, которые могут осуществлять прием через каналы и содержат соответствующий *Path* (*Путь*).

Строка 14 Для обоих направлений должны быть согласованными любые выборы подмножеств уточнений.

Строка 16 Скорректировать дескрипторы процессов в *reachset*1 относительно *reachset'* с помощью *Reachabilities*, содержащих возможных получателей, которые выводятся из *reachset*2 относительно *reachset*2'.

Строка 18 Проделать то же самое для остальных определений каналов.

$$\mathit{is\text{-}consistent\text{-}refinement}(\mathit{reachset1}, \mathit{reachset2}) \triangleq \tag{5.4.2}$$

```
1  (let sigset1 = {sig | (∃(, sset, ) ∈ reachset1)(sig ∈ sset)},
2       sigset2 = {sig | (∃(, sset, ) ∈ reachset2)(sig ∈ sset)} in
3   let env1 = card reachset1 = 1 ∧ (∃(p, , ) ∈ reachset1)(p = ENVIRONMENT),
4       env2 = card reachset2 = 1 ∧ (∃(p, , ) ∈ reachset2)(p = ENVIRONMENT) in
5   ¬(∃mk-Identifier1(qual1, ), mk-Identifier1(qual2, nm2) ∈ sigset1)
6     (len qual1 > len qual2 ∧ qual2 ⌢ ⟨mk-Signal-qualifier1(nm2)⟩ = ⟨qual1[i] | 1 ≤ i ≤ len qual2 + 1⟩) ∧
7   ¬(∃mk-Identifier1(qual1, ), mk-Identifier1(qual2, nm2) ∈ sigset2)
8     (len qual1 > len qual2 ∧ qual2 ⌢ ⟨mk-Signal-qualifier1(nm2)⟩ = ⟨qual1[i] | 1 ≤ i ≤ len qual2 + 1⟩) ∧
9   (env1 ∨ env2 ∨ sigset1 = sigset2))
```

type: *Reachabilities Reachabilities* → *Bool*

Назначение Проверить, что сигналы в маршрутах сигналов каждой конечной точки канала не содержат сигналов на разных уровнях уточнения одного и того же сигнала и что множество сигналов от исходящей конечной точки канала то же, что и множество сигналов во входящей конечной точке.

Параметры

*reachset*1 *Reachabilities* (*Достижимости*) для исходящего конца канала.

*reachset*2 *Reachabilities* (*Достижимости*) для входящего конца канала.

Результат **True** (**Истина**), если удовлетворяются вышеприведенные условия.

Алгоритм

Строка 1 Пусть *sigset* 1 обозначает множество сигналов на исходящем конца канала.

Строка 2 Пусть *sigset*2 обозначает множество сигналов на входящем конце канала.

Строка 3 Пусть *env*1 есть **true**, если исходящий конец канала является внешней средой системы.

Строка 4 Пусть *env*2 есть **true**, если входящий конец канала является внешней средой системы.

Строки 5—6 Для каждых двух исходящих сигналов должно выполняться следующее: они не должны быть подсигналами друг друга.

Строки 7—8 Для каждых двух входящих сигналов должно выполняться следующее: они не должны быть подсигналами друг друга.

Строка 9 До тех пор, пока одна из конечных точек является внешней средой системы, множество исходящих сигналов должно быть равным множеству входящих сигналов.

out-going-paths(*chid*, *b*, *bset*, *path*)(*dict*) ≜

```
1  if b = ENVIRONMENT then
2    ({(ENVIRONMENT, ⟨chid⟩)}, dict)
3    else
4     (let mk-Identifier1(level, bnm) = b in
5      let bdef ∈ bset be s.t. s-Block-name1(bdef) = bnm in
6      let mk-Block-definition1(,,, connects, srdefs,,, sub) = bdef in
7      let path' = path ⌢ ⟨chid⟩ in
8      let bqual = level ⌢ ⟨mk-Block-qualifier1(bnm)⟩ in
9      if (∃(mk-Identifier1(qual,), PROCESS) ∈ dom dict)(qual = bqual) then
10       (let mk-Channel-to-route-connection1(ch, routeset) ∈ connects be s.t. ch = chid in
11        let (rset, dict') = make-out-reaches(routeset, srdefs, path')(dict) in
12        (rset, dict'))
13       else
14        (let mk-Block-substructure-definition1(, bset', connects', cset,,,) = sub in
15         let mk-Channel-connection1(cid, cidset) ∈ connects' be s.t. cid = chid in
16         let dict' = make-structure-paths(bset', cset, level)(dict) in
17         make-out-connect-paths(cidset, cset, bset', path')(dict')))
```

type: *Channel-identifier*$_1$ (*Block-identifier*$_1$ | ENVIRONMENT)
Block-identifier$_1$-**set** *Path* →
Entity-dict → *Reachability-set Entity-dict*

Назначение

Образовать *Reachabilities* (*Достижимости*), соответствующие сигналам, выходящим из блока через данный канал (*chid*). Канал — это часть *Paths* (*Путей*) в *Reachabilities*. Образованные (временно) *Reachabilities* отличаются от *Reachabilities*, находящихся в дескрипторах процессов, поскольку *Path* является только частичным коммуникационным путем (часть получателя опущена), а *Идентификатор-процесса*$_1$ в *Reachabilities* является процессом передачи. Дополнительная функция *incoming-path* (*входящий-путь*) образует «обратную» *Reachability* (*Достижимость*), где *Path* является исходящей частью, и в функции *update-processd* две *Reachabilities* сливаются в вид *Reachabilities*, которые вводятся в дескриптор процесса передачи; *outgoing-path* (*исходящий путь*) также корректирует дескрипторы процесса, но только с помощью *Reachabilities*, локальных по отношению к блоку, откуда исходят каналы. Поскольку несколько каналов могут исходить от одного и того же блока, дескрипторы процесса могут корректироваться несколько раз с помощью тех же локальных *Reachabilities*, однако это не имеет значения, если *Reachabilities* представляют собой множество.

Параметры

chid	Идентификатор канала, из которого функция образует *Paths* (*Пути*).
b	Идентификатор блока, из которого исходит канал.
bset	Множество определений блоков, среди которых может находиться блок.
path	Уже образованный путь, ведущий из объемлющего блока (если канал не является подканалом, то путь содержит только *chid*).

Результат

(Временные) *Reachabilities* и *entitydict*, модифицированный коммуникационными путями, локальными по отношению к блоку.

Алгоритм

Строки 1—2	Если исходящая конечная точка является внешней средой системы, тогда возвратить *Reachability* (*Достижимость*), содержащую ENVIRONMENT в качестве исходящей конечной точки неизменяемого *dict*.
Строки 4—6	Выбрать из *bset* определение блока, соответствующее идентификатору блока *b*.
Строка 7	Прибавить идентификатор канала (*chid*) к *Path* (*Пути*) (обозначающему путь от канала, отличного от подканала, к *chid*).

Строка 8 — Пусть *bqual* обозначает квалификатор объектов, определенных в блоке *bnm*.

Строки 9—12 — Если процессы в блоке выбраны, тогда возвратить *Reachabilities* (*Достижимости*), содержащие процессы (*rset*), и *Entity-dict*, скорректированный *Reachabilities* для процессов, которые могут вести передачу к другим процессам в блоке.

Строки 14—15 — Разложить определение подструктуры блока и точку соединения, с которой соединен канал (*chid*).

Строка 16 — Скорректировать *Entity-dict* с помощью *Reachabilities*, содержащих *Paths* (*Пути*), локальные по отношению к подструктуре блока. Отметим, что дескрипторы процесса модифицируются (без последствий) несколько раз теми же *Reachabilities*, если с блоком соединено несколько каналов.

Строка 17 — Продолжить создание *Reachabilities*, вводя подблоки, соединенные с подканалами *cidset*.

make-out-connect-paths(*cidset*, *cset*, *bset*, *path*)(*dict*) $\triangleq$ **(5.4.4)**

```
1  (if cidset = {} then
2     ({}, dict)
3   else
4    (let cid ∈ cidset in
5     let (reachsetrest, dictrest) = make-out-connect-paths(cidset \ {cid}, cset, bset, path)(dict) in
6     let cdef ∈ cset be s.t. s-Channel-name1(cdef) = s-Name1(cid) in
7     let mk-Channel-definition1(, mk-Channel-path1(b1, b2, ), ) = cdef in
8     let block = if b2 = ENVIRONMENT then b1 else b2 in
9     let (rset, dict') = out-going-paths(cid, block, bset, path)(dictrest) in
10    (reachsetrest ∪ rset, dict')))
```

type: *Channel-identifier*$_1$-**set** *Channel-definition*$_1$-**set** *Block-definition*$_1$-**set** *Path* → *Entity-dict* → *Reachability*-**set** *Entity-dict*

Назначение — Образовать временные *Reachabilities*, соответствующие сигналам, переносимым подканалами подструктуры блока. Дополнительной функцией является *make-in-connect-paths*.

Параметры

cidset — Множество подканалов.

cset — Множество определений каналов подструктуры блоков.

bset — Множество определений блоков для подструктуры блоков.

path — (Частичный) *Path* (*Путь*), ведущей от подструктуры блоков к первому каналу, отличному от подканала.

Результат — Тот же, что и для *out-going-paths* (*исходящих путей*).

Алгоритм

Строка 1 — При выполнении не возвращаются ни *Reachabilities*, ни сигналы. Возвращается неизмененный *Entity-dict*. (Результат создается рекурсивно.)

Строки 4—5 — Взять канал (*cid*) из множества подканалов и образовать (рекурсивно) *Reachabilities* для остальных подканалов.

Строки 6—7 — Выбрать определение канала, соответствующее имеющемуся подканалу.

Строка 8 — Выбрать блок-отправитель подканала.

Строка 9 — Образовать временные *Reachabilities* (*rset*), в которых процессы — это те процессы, которые посылают информацию через канал (*cid*) и содержатся в блоке (*block*), а *Paths* (*Пути*) — это *path*, скорректированный остальной частью пути от канала до маршрутов сигналов, связанных с процессами. Кроме того, образовать *Entity-dict*, который обновляется *достижимостями*, локальными по отношению к блоку (*block*).

Строка 10 Возвратить *Reachabilities* (*Достижимости*) и *Entity-dict* (как описано выше) для имеющегося подканала, объединенных с теми же конструкциями для всех других подканалов.

in-coming-paths(*cid*, *block*, *bset*, *path*)(*dict*) ≜ (5.4.5)

```
1  if block = ENVIRONMENT then
2    {(ENVIRONMENT, ⟨cid⟩)}
3   else
4    (let mk-Identifier1(qual, bnm) = block in
5     let bdef ∈ bset be s.t. s-Block-name1(bdef) = bnm in
6     let mk-Block-definition1(,,, connects, srdefs,,, sub) = bdef in
7     let path' = path ⌢ ⟨cid⟩ in
8     let bqual = qual ⌢ ⟨mk-Block-qualifier1(bnm)⟩ in
9     if (∃(mk-Identifier1(qual,), PROCESS) ∈ dom dict)(qual = bqual) then
10      (let mk-Channel-to-route-connection1(ch, routeset) ∈ connects be s.t. ch = cid in
11       make-in-reaches(routeset, srdefs, path'))
12     else
13      (let mk-Block-substructure-definition1(, bset', connects', cdefs,,,) = sub in
14       let mk-Channel-connection1(ch, cset) ∈ connects' be s.t. ch = cid in
15       make-in-connect-paths(cset, cdefs, bset', path')(dict)))
```

type: *Channel-identifier*$_1$ (*Block-identifier*$_1$ | ENVIRONMENT)
Block-identifier$_1$-**set** *Path* → *Entity-dict* → *Reachability-set*

Назначение Образовать и возвратить *Reachabilities*, в которых процессы — это процессы, содержащиеся в данном блоке. В противоположность *out-going-paths* (*исходящим-путям*) *in-coming-paths* (*входящие-пути*) образуют «действительные» *Reachabilities*, так как они содержат процессы-получатели. *Paths* (*Пути*) в *Reachabilities* являются частичными и обозначают пути от первого канала, отличного от подканала, к процессу-получателю. «Другой конец» пути образуется в дополнительной функции *out-going-paths*.

Параметры

cid	Канал, для которого образованы *Reachabilities* (*Достижимости*).
block	Приемный блок канала.
bset	Множество определений блоков, среди которых можно найти *block* (*блок*).
path	Множество возможных путей.

Алгоритм

Строки 1—2	Если конечная точка приема является внешней средой системы, тогда возвратить множество, содержащее только одну *Reachability* (*Достижимость*), где получателем является внешняя среда.
Строки 4—6	Выбрать и разложить определение блока, соответствующее идентификатору блока *Block*.
Строка 7	Сложить идентификатор канала с путем, который будет использован внутри блока.
Строка 8	Пусть *bqual* обозначает квалификатор объектов, определенных в блоке *bnm*.
Строка 9	Если существует дескриптор процесса, определенного в блоке, тогда подструктура не выбирается.
Строки 10—11	Выбрать и возвратить *Reachabilities*, соответствующие маршрутам сигналов (*routeset*), связанных с каналом (*ch*).

Строки 13—14 Разложить определение подструктуры блока и соединение канала, связывающее канал (*chid*) с подструктурой блока.

Строка 15 Продолжить создание *Reachabilities* (*Достижимостей*), вводя подблоки (*bset'*), соединенные с подканалами *cset*.

make-in-connect-paths(*cidset*, *cset*, *bset*, *path*)(*dict*) ≜ (5.4.6)

```
 1  if cidset = {} then
 2    {}
 3   else
 4    (let cid ∈ cidset in
 5     let reachsetrest = make-in-connect-paths(cidset \ {cid}, cset, bset, path)(dict) in
 6     let cdef ∈ cset be s.t. s-Channel-name1(cdef) = s-Name1(cid) in
 7     let mk-Channel-definition1(, mk-Channel-path1(b1, b2, ), ) = cdef in
 8     let block = if b2 = ENVIRONMENT then b1 else b2 in
 9     let inblockreach = in-coming-paths(cid, block, bset, path)(dict) in
10     reachsetrest ∪ inblockreach)
```

type: *Channel-identifier*$_1$-**set** *Channel-definition*$_1$-**set**
Block-identifier$_1$-**set** *Path* → *Entity-dict* → *Reachability-set*

Назначение Образовать *Reachabilities*, соответствующие сигналам, переносимым подканалами блочной подструктуры. Дополнительной функцией является *make-out-connect-paths*.

Параметры

cidset Множество подканалов.

cset Множество определений каналов подструктуры блока.

bset Множество определений блоков подструктуры блока.

path (Частичный) *Path* (*Путь*), ведущий от первого канала, отличного от подканала, к подструктуре блока.

Алгоритм

Строка 1 При выполнении *Reachabilities* не возвращаются (результат получается рекурсивно).

Строки 4—5 Взять идентификатор подканала из множества подканалов и создать (рекурсивно) *Reachabilities* для оставшейся части подканалов.

Строки 6—7 Взять определение канала из *cset*, соответствующее имеющемуся идентификатору подканала. Отметим, что информация о том, какие сигналы переносит канал, не используется, так как имеются маршруты сигналов, определяющие, какие сигналы действительно переносятся каналом.

Строка 8 Выбрать блок-получатель.

Строка 9 Образовать *Reachabilities* (*inblockreach*), в которых процессы — это процессы, принимающие информацию через канал (*cid*) и содержащиеся в блоке (*block*), сигналы — это сигналы, получаемые процессами через канал (*cid*), а соответствующие маршруты сигналов и *Paths* (*Пути*) — это *path*, скорректированный остальной частью пути от канала до маршрутов сигналов, связанных с процессами.

Строка 10 Возвратить *Reachabilities* (как описано выше) для имеющегося подканала, объединенных с теми же конструкциями для всех других подканалов.

make-in-reaches(*routeset*, *srdefs*, *path*) ≜ (5.4.7)

```
1  if routeset = {} then
2    ({}, {})
3   else
4    (let route ∈ routeset in
5     let reachrest = make-in-reaches(routeset \ {route}, srdefs, path) in
6     let mk-Identifier1(, rnm) = route in
7     let mk-Signal-route-definition1(nm, path1, path2) ∈ srdefs be s.t. nm = rnm in
8     let mk-Signal-route-path1(e1, e2, sigset) = path1 in
9     let (signalset, dest) =
10        if e1 = ENVIRONMENT then
11          (sigset, e2)
12         else
13          if path2 = nil then
14            ({}, e1)
15           else
16            (let mk-Signal-route-path1(, , sigset') = path2 in
17             (sigset', e1)) in
18     reachrest ∪ {(dest, signalset, path ⌢ ⟨route⟩)})
```

type: *Signal-route-identifier*$_1$-**set** *Signal-route-definition*$_1$-**set** *Path* → *Reachabilities*

Назначение Образовать *Reachabilities* (*Достижимости*) для частичного *Path* (*Пути*), ведущего к точке соединения маршрутов сигналов. Имеется столько образованных *Reachabilities*, сколько имеется идентификаторов маршрутов сигналов в точке соединения маршрутов сигналов. Дополнительной функцией обработки исходящих сигналов является *make-out-reaches*.

Параметры

routeset	Множество идентификаторов маршрутов сигналов для соединения маршрутов сигналов.
srdefs	Соответствующие им определения маршрутов сигналов.
path	*Paths* (*Пути*), к которым прибавляются маршруты сигналов.

Результат Образованные *Reachabilities*.

Алгоритм

Строка 1	После выполнения никаких действий по возврату не производится (результат получается рекурсивно).
Строки 4—5	Взять *Signal-route-identifier*$_1$ (*Идентификатор-маршрута-сигнала*$_1$) из *routeset* и образовать *Reachabilities* для остальной части идентификаторов маршрутов сигналов.
Строки 6—8	Выбрать и разложить определение маршрута сигнала, соответствующее идентификатору маршрута сигнала.
Строки 9—17	Выбрать входящие сигналы (*signalset*) и процесс-получатель из определения маршрутов сигналов.
Строка 18	Возвратить *Reachability*, соответствующую имеющемуся идентификатору маршрута сигнала, соединенную с *Reachabilities*, соответствующими остальной части идентификаторов маршрутов сигналов.

```
make-out-reaches(routeset, routedefs, path)(dict) ≜                                    (5.4.8)
 1  (if routedefs = {} then
 2     ({}, dict)
 3   else
 4    (let route ∈ routedefs in
 5     let (restr, restd) = make-out-reaches(routeset, routedefs \ {route}, path)(dict) in
 6     let mk-Signal-route-definition1(rnm, mk-Signal-route-path1(p1, p2, sset), path2) = route in
 7     if p1 = ENVIRONMENT ∨ p2 = ENVIRONMENT then
 8       if (∃id ∈ routeset)(s-Name1(id) = rnm) then
 9         (let id ∈ routeset be s.t. s-Name1(id) = rnm in
10          if path2 = nil then
11            if p1 = ENVIRONMENT then
12              (restr, restd)
13            else
14              (restr ∪ {(p1, sset, ⟨id⟩ ⌢ path)}, restd)
15          else
16           (let mk-Signal-route-path1(, , sset') = path2 in
17            let (originp, sset'') =
18                if p1 = ENVIRONMENT then
19                  (p2, sset')
20                else
21                  (p1, sset) in
22            (restr ∪ {(originp, sset'', ⟨id⟩ ⌢ path)}, restd)))
23       else
24        (restr, restd)
25     else
26      (let mk-Identifier1(level, ) = p1 in
27       (restr, make-local-reach(mk-Identifier1(level, rnm), route)(restd)))))
```

type: *Signal-route-identifier*$_1$-**set** *Signal-route-definition*$_1$-**set** *Path* → *Entity-dict* → *Reachability*-**set** *Entity-dict*

Назначение

Образовать *Reachabilities* (*Достижимости*) для частичного *Path* (*Пути*), исходящего из точки соединения маршрутов сигналов. Образованных *Reachabilities* столько, сколько идентификаторов маршрутов сигналов содержится в точке соединения маршрутов сигналов. Дополнительной функцией, обрабатывающей входящие сигналы, является *make-in-reaches*. Кроме того, модифицировать дескрипторы процесса в *Entity-dict достижимостями*, соответствующими маршрутам сигналов между процессами. Если блок имеет несколько соединений маршрутов сигналов, дескрипторы процесса модифицируются несколько раз.

Параметры

routeset	Множество идентификаторов маршрутов сигналов для точки соединения.
routedefs	Множество определений маршрутов сигналов блока.
path	Частичный *Path* (*Путь*), исходящий из точки соединения.

Результат

Образованные *Reachabilities* и *Entity-dict*, модифицированные *достижимостями*, соответствующими маршрутам сигналов между процессами.

Алгоритм

Строка 1	Рассматривается каждое определение маршрута сигнала в блоке. При выполнении возврат *Reachabilities* не происходит, возвращается неизменный *Entity-dict*.
Строка 4	Взять определение маршрута сигнала из *routeset* и образовать *Reachabilities* (*restr*) и модифицированный *Entity-dict* (*restd*) для оставшейся части определений маршрутов сигналов.
Строки 7—24	Рассмотреть полностью случай, когда маршрут сигнала связан с каналом.

Строки 8 и 24 Если маршрут сигнала не упоминается в этой точке соединения (представлен в *routeset*), тогда возвратить информацию, относящуюся к остальным определениям маршрутов сигналов (то есть никаких действий с имеющимся определением маршрута сигнала не производить).

Строка 9 Выбрать идентификатор маршрута сигнала из *routeset*.

Строки 10—14 Если маршрут сигнала однонаправленный, тогда, если маршрут сигнала «входящий» (строки 11—12), не производить никаких действий с имеющимся определением маршрута сигнала, иначе возвратить *Reachabilities* (*Достижимости*) от остальной части определений маршрутов сигналов, объединенные с *Reachability*, содержащей процесс-отправитель (*p*1), сигналами, переносимыми маршрутом сигналов (*sset*), и *Path* (*Путем*), где прибавляется идентификатор маршрута сигнала (*id*). Также возвратить возможный модифицированный *Entity-dict*.

Строки 16—22 Если маршрут сигнала двунаправленный, тогда выделить процесс-отправитель (*originp*) из соответствующего $Signal\text{-}route\text{-}path_1$ (*Пути-маршрута-сигнала*$_1$), а затем выполнить действия строки 14.

Строки 26—27 Если маршрут сигнала связывает процессы, тогда новая *Reachability* не создается, но дескрипторы процессов в *Entity-dict* модифицируются *достижимостями* коммуникационных путей между двумя процессами (обрабатываемыми в *make-local-reach*).

$make\text{-}local\text{-}reach(id, \text{mk-}Signal\text{-}route\text{-}definition_1(rnm, path1, path2))(dict) \triangleq$ (5.4.9)

```
1  (let mk-Signal-route-path1(p1, p2, sset) = path1 in
2   let mk-ProcessDD(parm, init, mazi, graph, inrset) = dict((p1, PROCESS)) in
3   let reach = (p2, sset, ⟨id⟩) in
4   let dict' = dict + [(p1, PROCESS) ↦ mk-ProcessDD(parm, init, mazi, graph, inrset ∪ {reach})] in
5   if path2 = nil then
6     dict'
7   else
8     make-local-reach(id, mk-Signal-route-definition1(rnm, path2, nil))(dict'))
```

type: $Signal\text{-}route\text{-}identifier_1\ Signal\text{-}route\text{-}definition_1 \to Entity\text{-}dict \to Entity\text{-}dict$

Назначение Модифицировать один или два дескриптора процессов в *Entity-dict* с помощью *Reachabilities* для другой конечной точки процесса. Два дескриптора процесса модифицируются, если только маршрут сигнала является двунаправленным.

Параметры

id Идентификатор маршрута сигнала.

$Signal\text{-}route\text{-}definition_1$ Определение маршрута сигнала, содержащее

rnm имя маршрута сигнала,

*path*1 первый $Signal\text{-}route\text{-}path_1$ (*Путь-маршрута-сигнала*$_1$),

*path*2 второй (дополнительный) $Signal\text{-}route\text{-}path_1$.

Результат Модифицированный *Entity-dict*.

Алгоритм

Строки 1—4 Модифицированный *Entity-dict* с помощью *Reachabilities* одного из направлений. *Reachability* (*Достижимость*), прибавляемая к дескриптору процесса для процесса-отправителя (*p*1), содержит процесс-получатель (*p*2), сигналы (*sset*) и *Path* (*Путь*), содержащий только идентификатор маршрута сигнала (*id*).

Строки 5—8 Если маршрут сигнала однонаправленный, тогда возвратить модифицированный *Entity-dict*, иначе делать то же самое, когда маршрут сигнала рассматривается как однонаправленный и содержащийся $Signal\text{-}route\text{-}path_1$ (*Путь-маршрута-сигнала*$_1$) является структурой, которая пока не обработана.

```
update-processd(outrset, inrset)(dict) ≜                                    (5.4.10)
 1  if outrset = {} then
 2    dict
 3   else
 4    (let outreach ∈ outrset in
 5     let (pid, sigset, path) = outreach in
 6     let inrset' =
 7         {inr ∈ inrset | (let (, , path') = inr in
 8                          path'[len path'] = hd path)} in
 9     let mk-ProcessDD(parmd, init, mazi, graph, rset) = dict((pid, PROCESS)) in
10     let reachabilityset = extract-reachabilities(sigset, path, inrset') in
11     let dict' = dict +
12         [(pid, PROCESS) ↦ mk-ProcessDD(parmd, init, mazi, graph, rset ∪ reachabilityset)] in
13     update-processd(outrset \ {outreach}, inrset)(dict'))

type: Reachability-set Reachability-set → Entity-dict → Entity-dict
```

Назначение Модифицировать дескрипторы процессов в *Entity-dict* с помощью *reachabilities*, получаемых из *Reachabilities* (*Достижимостей*), содержащих исходящие (частичные) *Paths* (*Пути*), и из *Reachabilities*, содержащих входящие (частичные) *Paths*.

Параметры

outrset	*Reachabilities*, содержащие исходящие *Paths* (*Пути*).
inrset	*Reachabilities*, содержащие входящие *Paths* (*Пути*).

Результат Модифицированный Entity-dict.

Алгоритм

Строка 1	Каждая *Reachability* (*Достижимость*), содержащая исходящие *Paths* (*Пути*), проверяется. Когда просмотрено все множество, возвратить (модифицированный) *Entity-dict*.
Строки 4—5	Взять *reachability* из *outrset*.
Строка 6	Выбрать те входящие *Reachabilities*, которые содержат продолжение *Path* (*Пути*) в имеющейся *Reachability*, то есть, выбрать те *Reachabilities*, которые имеют тот же идентификатор канала в конце *Path* (*Пути*), что и идентификатор канала в начале имеющегося *Пути* (*path*).
Строка 9	Разложить дескриптор процесса процесса-отправителя.
Строка 10	Пройти через все входящие *Reachabilities*, чтобы образовать возможные (полные) *Reachabilities*.
Строки 11—13	Модифицировать *Entity-dict* новыми *Reachabilities* (*Достижимостями*) и использовать этот модифицированный *Entity-dict* при обработке остальных исходящих *Reachabilities*.

```
extract-reachabilities(sigset, path, inrset) ≜                              (5.4.11)
 1  if inrset = {} then
 2    {}
 3   else
 4    (let inr ∈ inrset in
 5     let (pid, sigset', path') = inr in
 6     let reach = if sigset ∩ sigset' = {} then
 7                   {}
 8                 else
 9                   {(pid, sigset ∩ sigset', path ⌢ tl path')} in
10     {reach} ∪ extract-reachabilities(sigset, path, inrset \ {inr}))

type: Signal-identifier1-set Path Reachability-set → Reachability-set
```

Назначение Образование *Reachabilities* (*Достижимостей*) от исходящего (частичного) *Path* (*Пути*) и множества входящих *Reachabilities*.

Параметры

sigset Множество сигналов в исходящем *Path* (*Пути*).

path Исходящий *Path* (*Путь*).

inrset Множество входящих *Reachabilities*.

Результат Образованные *Reachabilities*.

Алгоритм

Строка 1 Когда произведен проход через множество *Reachabilities* (*Достижимостей*), никаких возвратов не производится.

Строки 4—5 Взять входящую *Reachability* из *inrset*.

Строки 5—6 Образовать *Reachability*, которая пуста, если входящая *Reachability* не имеет сигналов, общих с исходящим *Path* (*Путем*), в противном случае она содержит получателя (*pid*), пересечение сигналов, возможных во входящей *Reachability* и в исходящем *Path* (*Пути*), и полный *Path* (*Путь*), образуемый конкатенацией исходящего пути (*path*) с входящим путем (*path*), за исключением первого элемента (то есть идентификатора канала, который также включается в качестве последнего элемента в исходящем пути).

Строка 10 Возвратить образованную *Reachability* вместе с *Reachabilities*, образованными из остальных входящих *Reachabilities*.

Указатель процессоров

Указатель переменных

Сообщения об ошибках

§ 2.7.4: Найдено много получателей 13
§ 2.7.4: Получатель не найден 13
§ 2.7.5: Ответы в действиях по принятию решений не являются взаимно исключающими 47, 48
§ 2.7.5: Нет сопоставимого ответа 30

§ 3.2.1: Блок-лист не содержит процессов 69
§ 3.2.1: Подблок не находится в согласованном подмножестве 69
§ 3.3: Запрещенное уточнение канала 70

§ 5.2.1: Порождение или сокращение эквивалентных классов объемлющей структурной единицы 52
§ 5.4.1.7: Литерал равен терму-ошибке 52
§ 5.4.1.7: Применение операции эквивалентно терму-ошибке 53
§ 5.4.1.9: Значение вне диапазона Подтипа 33, 34, 38, 39, 42
§ 5.5.2.2: Обозреваемое значение не определено 39
§ 5.5.2.2: Значение переменной, к которой осуществлен доступ, не определено 38
§ 5.5.2.3: Условие должно проверяться по отношению к TRUE (ИСТИНА) или FALSE (ЛОЖЬ) 40
§ 5.5.4.4: Процесс раскрытия не является действующим 17

ISBN 92-61-03794-1