This electronic version (PDF) was scanned by the International Telecommunication Union (ITU) Library & Archives Service from an original paper document in the ITU Library & Archives collections.

La présente version électronique (PDF) a été numérisée par le Service de la bibliothèque et des archives de l'Union internationale des télécommunications (UIT) à partir d'un document papier original des collections de ce service.

Esta versión electrónica (PDF) ha sido escaneada por el Servicio de Biblioteca y Archivos de la Unión Internacional de Telecomunicaciones (UIT) a partir de un documento impreso original de las colecciones del Servicio de Biblioteca y Archivos de la UIT.

هذه النسخة الإلكترونية (PDF) نتاج تصوير بالمسح الضوئي أجراه قسم المكتبة والمحفوظات في الاتحاد الدولي للاتصالات (ITU) من نسخة أصلية قورة ثقيو ملانقً.

此电子版（PDF版本）由国际电信联盟（ITU）图书馆和档案室利用存于该处的纸质文件扫描提供。

Настоящий электронный вариант (PDF) был подготовлен в библиотечно-архивной службе Международного союза электросвязи путем сканирования исходного документа в бумажной форме из библиотечно-архивной службы МСЭ.

© International Telecommunication Union

МЕЖДУНАРОДНЫЙ СОЮЗ ЭЛЕКТРОСВЯЗИ

МККТТ

МЕЖДУНАРОДНЫЙ
КОНСУЛЬТАТИВНЫЙ КОМИТЕТ
ПО ТЕЛЕГРАФИИ И ТЕЛЕФОНИИ

СИНЯЯ КНИГА

ТОМ X — ВЫПУСК X.4

ПРИЛОЖЕНИЕ F.2 К РЕКОМЕНДАЦИИ Z.100: ФОРМАЛЬНОЕ ОПИСАНИЕ SDL

СТАТИЧЕСКИЕ СЕМАНТИКИ

IX ПЛЕНАРНАЯ АССАМБЛЕЯ
МЕЛЬБУРН 14-25 НОЯБРЯ 1988

Женева 1989

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.4

ПРИЛОЖЕНИЕ F.2 К РЕКОМЕНДАЦИИ Z.100: ФОРМАЛЬНОЕ ОПИСАНИЕ SDL

СТАТИЧЕСКИЕ СЕМАНТИКИ

IX ПЛЕНАРНАЯ АССАМБЛЕЯ
МЕЛЬБУРН 14-25 НОЯБРЯ 1988

Женева 1989

ISBN 92-61-03784-4

PAGE INTENTIONALLY LEFT BLANK

PAGE LAISSEE EN BLANC INTENTIONNELLEMENT

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

СИНЯЯ КНИГА

Том I

ВЫПУСК I.1 — Протоколы и отчеты Пленарной Ассамблеи.
Перечень исследовательских комиссий и изучаемых вопросов.

ВЫПУСК I.2 — Пожелания и резолюции.
Рекомендации по организации и процедурам работы МККТТ (Серия А).

ВЫПУСК I.3 — Термины и определения. Аббревиатуры и сокращения. Рекомендации по средствам выражения (Серия В) и общей статистике телекоммуникаций (Серия С).

ВЫПУСК I.4 — Указатель синей книги.

Том II

ВЫПУСК II.1 — Общие принципы тарификации — Таксация и расчеты в международных службах электросвязи. Рекомендации серии D (Исследовательская Комиссия III).

ВЫПУСК II.2 — Телефонная служба и ЦСИС — Эксплуатация, нумерация, маршрутизация и подвижная служба. Рекомендации E.100-E.333 (Исследовательская Комиссия II).

ВЫПУСК II.3 — Телефонная служба и ЦСИС — Качество обслуживания, управление сетью и расчет нагрузки. Рекомендации E.401-E.880 (Исследовательская Комиссия II).

ВЫПУСК II.4 — Телеграфная и подвижная службы — Эксплуатация и качество обслуживания. Рекомендации F.1-F.140 (Исследовательская Комиссия I).

ВЫПУСК II.5 — Телематические службы, службы передачи данных и конференц—связи. — Эксплуатация и качество обслуживания. Рекомендации F.160-F.353, F.600, F.601,F.710-F.730 (Исследовательская Комиссия I).

ВЫПУСК II.6 — Службы обработки сообщений и справочные службы. — Эксплуатация и определение службы. Рекомендации F.400-F.422,F.500 (Исследовательская Комиссия I).

Том III

ВЫПУСК III.1 — Общие характеристики международных телефонных соединений и каналов. Рекомендации G.101-G.181 (Исследовательские Комиссии XII и XV).

ВЫПУСК III.2 — Международные аналоговые системы передачи. Рекомендации G.211-G.544 (Исследовательская Комиссия XV).

ВЫПУСК III.3 — Среда передачи — Характеристики. Рекомендации G.601-G.654 (Исследовательская Комиссия XV).

ВЫПУСК III.4 — Общие принципы цифровых систем передачи; оконечное оборудование. Рекомендации G.700-G.772 (Исследовательские Комиссии XV и XVIII).

ВЫПУСК III.5 — Цифровые сети, цифровые участки и цифровые линейные системы. Рекомендации G.801-G.956 (Исследовательские Комиссии XV и XVIII).

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

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

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

ВЫПУСК III.9 — Цифровая сеть с интеграцией служб (ЦСИС) — Межсетевые стыки и принципы технической эксплуатации. Рекомендации I.500-I.505 (Исследовательская Комиссия XVIII).

Том IV

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

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

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

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

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

Том VI

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

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

ВЫПУСК VI.3 — Требования к системам 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-Q.921 (Исследовательская Комиссия XI).

ВЫПУСК VI.11 — Цифровая абонентская система сигнализации № 1 (ЦАС 1), сетевой уровень, управление сетью пользователей. Рекомендации Q.930-Q.940 (Исследовательская Комиссия XI).

ВЫПУСК VI.12 — Государственная сухопутная подвижная сеть. Взаимодействие с ЦСИС и КТСОП. Рекомендации Q.1000-Q.1032 (Исследовательская Комиссия XI).

ВЫПУСК VI.13 — Государственная сухопутная подвижная сеть. Подсистема подвижного применения и стыки. Рекомендации Q.1051-Q.1063 (Исследовательская Комиссия XI).

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

Том VII

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

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

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

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

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

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

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

Том VIII

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

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

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

ВЫПУСК VIII.4 — Сети передачи данных: взаимосвязь открытых систем (ВОС) — модель и система обозначений, определения 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). Прокладка и защита кабеля и других элементов линейных сооружений. Рекомендации серии L (Исследовательская Комиссия VI).

Том X

ВЫПУСК X.1 — Язык функциональной спецификации и описания (SDL). Критерии применения методов формального описания (FDT). Рекомендации Z.100 и Приложения A, B, C и E. Рекомендация Z.110 (Исследовательская Комиссия X).

ВЫПУСК X.2 — Приложение D к Рекомендации Z.100: руководство пользователя SDL (Исследовательская Комиссия X).

ВЫПУСК X.3 — Приложение F.1 к Рекомендации Z.100: формальное описание SDL. Введение (Исследовательская Комиссия X).

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

ВЫПУСК X.5 — Приложение F.3 к Рекомендации Z.100: формальное описание SLD. Динамическая семантика (Исследовательская Комиссия X).

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

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

СОДЕРЖАНИЕ ВЫПУСКА X.4 ГОЛУБОЙ КНИГИ

Приложение F.2 к Рекомендации Z.100

Формальное описание SDL. Статические семантики 1

ЗАМЕЧАНИЯ

В связи с узкой специализацией семантик SDL этот выпуск публикуется только на английском языке.

REMARQUE

Etant donne la nature tres speciale de la semantique du LDS, ce fascicule est publie uniquement en anglais.

OBSERVACION

Debido a la naturaleza especializada de la semantica del LED, este fasciculo solo est publica en ingles.

ПРЕДВАРИТЕЛЬНЫЕ ЗАМЕЧАНИЯ

1 Вопросы, порученные каждой Исследовательской комиссии на исследовательский период 1989-1992, содержатся в документе N 1 Исследовательской комиссии.

2 В данном выпуске для краткости термин «Администрация» используется для обозначения как администрации электросвязи, так и признанной частной эксплуатационной организации.

PAGE INTENTIONALLY LEFT BLANK

PAGE LAISSEE EN BLANC INTENTIONNELLEMENT

Содержание

1 Абстрактный синтаксис, представляющий конкретный синтаксис **2**

1.1 Базисный SDL 2

1.2 Понятие структурирования 10

1.3 Дополнительные понятия 11

1.4 Данные 12

2 Внутренние домены **20**

2.1 Описан*ие Descriptordict* 21

2.2 Описание *Quotdict* 28

2.3 Другие домены 30

3 Преобразование AS_0 в AS_1 **32**

3.1 Основные функции 32

3.2 Замена ссылок на определения 35

3.3 Удаленние определений отбора 42

3.4 Преобразование определений 51

3.4.1 Определения блоков 54

3.4.2 Определения каналов 57

3.4.3 Определения процессов 61

3.4.4 Определения сигналов 65

3.4.5 Определения процедур 66

3.4.6 Генераторы сортов 70

3.4.7 Определения сорта 71

3.4.8 Определения таймеров 116

3.4.9 Определения переменных 116

3.4.10 Определение обозревания 119

3.4.11 Определения импорта 121

3.4.12 Определения маршрута сигнала 122

3.4.13 Определение списка сигналов Signallist 123

3.4.14 Сосотояния соединения 125

3.5 Преобразование выражений 134

3.5.1 Идентификаторы 137

3.5.2 Символьные строки 142

3.5.3 Операторы 142

3.5.4 Выражения условия 154

3.5.5 Инфиксные и префиксные операторы 155

3.6 Обработка обозревания 155

3.7 Преобразование процедурных графов и графов процессов 161

3.7.1 Вставка терминаторов в ответы 165

3.7.2 Построение Labeldict 169
3.7.3 Построение Statedict 172
3.7.4 Раскрытие ввода со звездочкой, сохранени со звездочкой и неявных переходов 174
3.7.5 Распространение непрерывных сигналов и условия разрешения 180
3.7.6 Преобразование состояний и вершин ввода 188
3.7.7 Преобразование строк перехода 195
3.7.8 Преобразование предложений действия 198
3.8 Общие функции создания в AS_1 223
3.9 Расширение сервисов 233
3.10 Создание неявных каналов и маршрутов сигналов 241
3.10.1 Создание неявных каналов 244
3.10.2 Создание неявных маршрутов сигналов 249
3.10.3 Создание неявных операторов соединения 253
3.11 Сервисные функции 256
3.12 Генерация вспомогательной информации 258
3.13 Отображения глобальных констант 260
3.13.1 Соотношения между именами AS_0 и AS_1 260
3.13.2 Соотношения между именами импорта/экспорта и неявными именами сигналов 260
3.14 Неформальные функции 261

4 Уточнения к Z.100 262

ВЫПУСК X.4

Приложение F.2 к Рекомендации Z.100

ФОРМАЛЬНОЕ ОПИСАНИЕ SDL
СТАТИЧЕСКИЕ СЕМАНТИКИ

PAGE INTENTIONALLY LEFT BLANK

PAGE LAISSEE EN BLANC INTENTIONNELLEMENT

Введение

Эта часть Формального Определения определяет статические свойства SDL. Для описания общей структуры Формального Определения и для объяснения используемых обозначений обращаться к Приложению F.1: Введение в Формальное Определение.

Приложение F.2 определяет следующее:

- Используемые корректные условия. Однако, корректные условия, которые вызывают уточнение равенств, для удобства согласуются с конструкцией *Entity-dict* в приложении F.3: Динамические Семантики (см. введение в раздел 5 Приложения F.3). Даже если условия находятся в Динамических Семантиках, условия являются статическими свойствами, и они, таким образом, применяются прежде, чем осуществляется любая интерпретация процессов SDL.

  Корректными условиями являются следующие:

  1. Проверка того, что ответы при принятии решения не перекрываются.
  2. Проверка того, что вложенный блок не делает недействительными свойства сорта, определенного в описывающем блоке.
  3. Проверка того, что никакой литерал не совпадает с ошибочным символом.
  4. Проверка того, что Булевские символы TRUE и FALSE обозначают различные величины.

- Связь между конкретным синтаксисом и абстрактным синтаксисом такая, как определено в Z.100 (обозначаемое как AS_1). AS_1 не повторяется в Формальном Описании. Резюме можно найти в Приложении B Z.100.

  Для того, чтобы отличить объекты AS_1 от других объектов в функциях META-IV, к каждому имени AS_1 добавляется индекс "$_1$".

  Например, определение *Identifier*, как определено в §2.2.2 Z.100, имеет вид:

  Identifier :: *Qualifier Name*

  тогда как соответствующее определение в Формальном Определении выглядит так

  $Identifier_1$ :: $Qualifier_1$ $Name_1$

1 Абстрактный синтаксис, представляющий конкретный синтаксис

Ниже определяется домен AS_0, который отражает конкретный синтаксис. Хотя AS_0 представляет конкретный синтаксис в абстрактной форме, AS_0 в некотором отношении действительно определяет синтаксис языка на более низком уровне, чем синтаксические правила Z.100. Это происходит благодаря тому, что контекстно-зависимая информация, содержащаяся в синтаксических правилах, не отражается в AS_0 (для объяснения см. раздел 3 в Приложении F.1).

В отличие от AS_1 сокращения имен домена используются в AS_0 для того, чтобы уменьшить проблемы, связанные с ограниченной шириной физической строки. Примечания, связанные с определениями домена, определяют полные имена, выделяя *курсивом* буквы, которые также появляются в сокращениях.

Для того, чтобы отличить объекты AS_0 от других объектов в функциях META-IV, каждый домен AS_0 отмечается индексом "$_0$".

В связи с тем, что AS_0 является Абстрактным Синтаксисом, происходящим от текстовой грамматики, следующие позиции не покрываются Формальным Определением:

- Лексические правила (включая макрос)
  Большинство этих правил формально определены в §2.2.1 Z.100.
- Синтаксические правила (AS_0 не определяет, где поместить ключевые слова и разделители, определяются только объекты, из которых состоит определение системы).
  Синтаксические правила формально определяются в Z.100 в подразделе *текстовая грамматика.*
- Соотношение между текстовой грамматикой и графической грамматикой
  В большинстве случаев это соотношение является прямым, т.е. <диаграмма> соответствует <определению>, "is followed by" в графической грамматике или буквенно "is followed by" в текстуальной грамматике и т.д.

Домены определяются так же, как соответствующие синтаксические правила определены в Z.100, например, первым определением домена в AS_0 является Id_0 и первым продуктом БНФ в Z.100 является <идентификатор>.

1.1 Базисный SDL

1	Id_0	::	$Qualifier_0$ $Name_0$
2	$Qualifier_0$	=	$Qualifierelem_0$*
3	$Qualifierelem_0$	=	$Scopeunit_0$ $Name_0$
4	$Scopeunit_0$	=	SYSTEM | BLOCK | PROCESS | SERVICE | PROCEDURE | SUBSTRUCTURE | TYPE | SIGNAL
5	$Name_0$	::	$Char^+$ [EXCLAMATIONMARK]

- Идентификатор (*Id*) состоит из квалификатора (*Qualifier*) и имени (*Name*)
- Квалификатор (*Qualifier*) является, возможно пустым, списком элементов квалификатора (*Qualifier elem*).
- Элемент квалификатора (*Qualifier elem*) является типом блок подразделения (*Scope unit*) и именем (*Name*).
- Тип блок подразделения (*Scope unit*) является либо системой, либо блоком, либо процессом, либо сервисом, либо процедурой, либо подструктурой блок/канал, либо частичным определением типа данных, либо уточнением сигнала.
- Имя (*Name*) состоит из непустого списка знаков, сопровождаемых необязательным восклицательным знаком.
  Для удобства имена (*Names*) в AS_0 покрывают несколько отличный класс имен чем тот, который определен синтаксическими правилами БНФ в Z.100.
  Используются следующие предположения для AS_0:
  - Необязательный восклицательный знак, сопровождающий ($Char^+$), может использоваться только, если это разрешено синтаксически в соответствии с правилами БНФ.

– Написание является последовательностью заглавных знаковых, которая согласуется с лексическими правилами, определенными в Z.100.

6 *String*$_0$ :: *Char**

- Знаковая строка (*String*) состоит из, возможно пустого, списка знаков.

7 *Text*$_0$ :: *String*$_0$

- Неформальный текст (*Text*) содержит знаковую строку (*String*). В контекстном выражении неформальный текст (*Text*) не может быть отличен от знаковой строки (*String*). Следовательно, в AS$_0$ текст (*Text*) появляется только в работе (*Task*), как альтернатива предложениям присваивания.

8 *Sys*$_0$ :: *Sysdef*$_0$ *Remotedef*$_0$

- Система (*Sys*) состоит из определения системы (*Sysdef*) и удаленных определений (*Remotedef*).

9 *Remotedef*$_0$ = *Refdecl*$_0$*
10 *Refdecl*$_0$ = *Blockdef*$_0$ | *Prdef*$_0$ | *Servicedef*$_0$ | *Procdef*$_0$ | *Chansubdef*$_0$

- Удаленное определение (*Remotedef*) является списком деклараций, на которые есть ссылки (*Refdecl*).
- Декларация, на которую есть ссылки (*Refdecl*), является либо определением блока (*Blockdef*), либо определением процесса (*Prdef*), либо определением сервиса (*Servicedef*), либо определением процедуры (*Procdef*), либо определением подструктуры канала (*Chansubdef*).

Подструктуры блока также могут быть удаленными. Однако, так как синтаксически невозможно отличить подструктуру удаленного блока от подструктуры удаленного канала, подструктура канала должна быть преобразована в подструктуру блока (см. *remove-references*), если контекст показывает, что удаленное определение действительно является подструктурой блока.

11 *Sysdef*$_0$ :: *Sysname*$_0$ *Sysdecl*$_0$* [*Tailname*$_0$]
12 *Sysname*$_0$ = *Name*$_0$
13 *Sysdecl*$_0$ = *Blockdef*$_0$ | *Blockref*$_0$ | *Chandef*$_0$ | *Sigdef*$_0$ | *Signallistdef*$_0$ | *Datadef*$_0$ | *Select*$_0$

- Определение системы (*Sysdef*) состоит из имени системы (*Sysname*), списка деклараций системы (*Sysdecl*) и необязательного оконечного имени (*Tailname*), завершающего определение системы (*Sysdef*).
- Имя системы (*Sysname*) является именем (*Name*).
- Декларация системы (*Sysdecl*) является либо определением блока (*Blockdef*), либо ссылкой на блок (*Blockref*), либо определением канала (*Chandef*), либо определением сигнала (*Sigdef*), либо определением списка сигналов (*Signallistdef*), либо определением данных (*Datadef*), либо определением отбора (*Select*).

14 *Blockref*$_0$ :: *Blockname*$_0$
15 *Blockdef*$_0$ :: *Blockid*$_0$ *Blockdecl*$_0$* [*Blocksub*$_0$] [*Tailid*$_0$]
16 *Blockid*$_0$ = *Id*$_0$
17 *Tailid*$_0$ = *Id*$_0$
18 *Blockdecl*$_0$ = *Sigdef*$_0$ | *Prdef*$_0$ | *Prref*$_0$ | *Datadef*$_0$ | *Connect*$_0$ | *Sigroutedef*$_0$ | *Signallistdef*$_0$ | *Select*$_0$
19 *Blockname*$_0$ = *Name*$_0$

- Ссылка на блок (*Blockref*) содержит имя блока (*Blockname*).
- Определение блока (*Blockdef*) содержит идентификатор блока (*Blockid*), список деклараций блока (*Blockdecl*), необязательное определение подструктуры блока (*Blocksub*) и необязательный конечный идентификатор (*Tailid*), завершающий определение блока (*Blockdef*).

Заметим, что синтаксически невозможно отличить имя (*Name*) от идентификатора (*Id*) в конкретном синтаксисе. Следовательно, AS_0 содержит идентификаторы (*Id*) на тех местах, где допустимы как имена (*Names*), так и идентификаторы (*Id*).

- Идентификатор блока (*Blockid*) является идентификатором (*Id*).
- Конечный идентификатор (*Tailid*) является идентификатором (*Id*).
- Декларация блока (*Blockdecl*) является либо определением сигнала (*Sigdef*), либо определением процесса (*Prdef*), либо ссылкой на процесс (*Prref*), либо определением данных (*Datadef*), либо соединением канала к маршруту сигналов (*Connect*), либо определением маршрута сигналов (*Sigroutedef*), либо определением списка сигналов (*Signallistdef*), либо отбором (*Select*).
- Имя блока (*Blockname*) является именем (*Name*).

20	$Prref_0$	::	$Prname_0$ [$Instances_0$]
21	$Prname_0$	=	$Name_0$
22	$Prdef_0$	::	$Prid_0$ [$Instances_0$] $Parm_0$* [$Inputset_0$] $Prdecl_0$* $Processbody_0$ [$Tailid_0$]
23	$Prid_0$	=	Id_0
24	$Prdecl_0$	=	$Vardef_0$ | $Viewdef_0$ | $Datadef_0$ | $Sigdef_0$ | $Signallistdef_0$ | $Importdef_0$ | $Procdef_0$ | $Procref_0$ | $Timerdef_0$ | $Select_0$
25	$Procref_0$	::	$Procname_0$
26	$Inputset_0$	::	[$Signallist_0$]
27	$Processbody_0$	=	$Body_0$ | $Decomposition_0$
28	$Parm_0$	::	$Varname_0$+ $Sortid_0$
29	$Instances_0$	::	[$Initial_0$] [$Maximum_0$]
30	$Initial_0$	=	$Expr_0$
31	$Maximum_0$	=	$Expr_0$

- Ссылка на процесс (*Prref*) содержит имя процесса (*Prname*) и необязательный номер спецификации экземпляров (*Instances*).
- Имя процесса (*Prname*) является именем (*Name*).
- Определение процесса (*Prdef*) состоит из идентификатора процесса (*Prid*), необязательного номера спецификации экземпляров (*Instances*), списка формальных параметров процесса (*Parm*), необязательного множества допустимых входных сигналов (*Inputset*), списка деклараций процесса (*Prdecl*), тела процесса (*Processbody*), и необязательного конечного идентификатора (*Tailid*), завершающего определение процесса.
- Идентификатор процесса (*Prid*) является идентификатором (*Id*).
- Декларация процесса (*Prdecl*) является либо определением переменной (*Vardef*),либо определением обозревания (*Viewdef*), либо определением данных (*Datadef*), либо определением сигнала (*Sigdef*), либо определением списка сигналов (*Signallistdef*), либо определением импорта (*Importdef*), либо определением процедуры (*Procdef*), либо ссылкой на процедуру (*Procref*), либо определением таймера (*Timerdef*), либо отбором (*Select*).
- Ссылка на процедуру (*Procref*) состоит из имени процедуры (*Procname*).
- Множество допустимых входных сигналов (*Inputset*) содержит, возможно пустой, (если nil), список сигналов (*Signallist*).
- Тело процесса (*Processbody*) является либо графом тела (*Body*), либо декомпозицией сервиса (*Decomposition*).
- Формальные параметры (*Parm*) процесса состоят из списка имен переменных (*Varname*) и идентификатора сорта (*Sortid*).
- Номер спецификации экземпляров (*Instances*) состоит из необязательного начального номера (*Initial*) и необязательного максимального номера (*Maximum*).
- Начальное число (*Initial*) является простым выражением (*Expr*).
- Максимальный номер (*Maximum*) является простым выражением (*Expr*).

32	*Procdef*$_0$	::	*Procid*$_0$ *Procparm*$_0$* *Procdecl*$_0$* *Body*$_0$ [*Tailid*$_0$]
33	*Procid*$_0$	=	*Id*$_0$
34	*Procdecl*$_0$	=	*Vardef*$_0$ | *Datadef*$_0$ | *Procdef*$_0$ | *Procref*$_0$ | *Select*$_0$
35	*Procname*$_0$	=	*Name*$_0$
36	*Procparm*$_0$	=	*Inoutparm*$_0$ | *Inparm*$_0$
37	*Inoutparm*$_0$	::	*Varname*$_0$$^+$ *Sortid*$_0$
38	*Inparm*$_0$	::	*Varname*$_0$$^+$ *Sortid*$_0$

- Определение процедуры (*Procdef*) состоит из идентификатора процедуры (*Procid*), списка формальных параметров процедуры (*Procparm*),списка деклараций процедуры (*Procdecl*),тела процедуры (*Body*) и необязательного конечного идентификатора (*Tailid*),завершающего определение процедуры.
- Идентификатор процедуры (*Procid*) является идентификатором (*Id*).
- Декларация процедуры (*Procdecl*) является либо определением переменной (*Vardef*),либо определением данных (*Datadef*), либо определением процедуры (*Procdef*), либо ссылкой на процедуру (*Procref*), либо отбором (*Select*).
- Имя процедуры (*Procname*) является именем (*Name*).
- Формальный параметр процедуры (*Procparm*) является либо параметром переменной *In/Out* (*Inoutparm*), либо параметром переменной *In* (*Inparm*).
- Параметр *In/Out* (*Inoutparm*) состоит из списка имен переменной (*Varname*) и идентификатора сорта (*Sortid*).
- Параметр *In* (*Inparm*) состоит из списка имен переменной (*Varname*) и идентификатора сорта (*Sortid*).

39	*Chandef*$_0$	::	*Channame*$_0$ *Chanpath*$_0$ [*Chanpath*$_0$] [*Chansub*$_0$] [*Tailname*$_0$]
40	*Chanpath*$_0$	::	*Orig*$_0$ *Dest*$_0$ *Signallist*$_0$
41	*Channame*$_0$	=	*Name*$_0$
42	*Orig*$_0$	=	*Blockid*$_0$ | ENV
43	*Dest*$_0$	=	*Blockid*$_0$ | ENV
44	*Tailname*$_0$	=	*Name*$_0$

- Определение канала (*Chandef*) состоит из имени канала (*Channame*), одного или двух путей канала связи (*Chanpath*), необязательного определения подструктуры канала (*Chansub*) и необязательного конечного имени (*Tailname*), завершающего определение канала. Путь канала связи (*Chanpath*) состоит из оконечной точки источника (*Orig*), оконечной точки назначения (*Dest*) и списка сигналов (*Signallist*).
- Оконечная точка источника (*Orig*) является или идентификатором блока (*Blockid*), или окружающей средой (ENV).
- Оконечная точка назначения (*Dest*) является или идентификатором блока (*Blockid*), или окружающей средой (ENV).
- Имя, которое заканчивает определение (*Name*), является конечным именем (*Tailname*).

45	*Sigroutedef*$_0$	::	*Sigroutename*$_0$ *Sigroutepath*$_0$ [*Sigroutepath*$_0$]
46	*Sigroutepath*$_0$	::	*Origin*$_0$ *Destination*$_0$ *Signallist*$_0$
47	*Origin*$_0$	=	*Id*$_0$ | ENV
48	*Destination*$_0$	=	*Id*$_0$ | ENV
49	*Sigroutename*$_0$	=	*Name*$_0$

- Определение маршрута сигнала (*Sigroutedef*) содержит имя маршрута сигнала (*Sigroutname*) и один или два пути связи маршрута сигнала (*Sigroutepath*).
- Путь маршрута сигнала (*Sigroutepath*) содержит исходный процесс (*Origin*) или сервис, приемный процесс (*Destination*) или сервис и список сигнала (*Signallist*).
- Исходный процесс (*Origin*) или сервис является либо идентификатором (*Id*), либо окружающей средой (ENVIRONMENT).
- Приемный процесс (*Destination*) является либо идентификатором процесса (*Id*), либо окружающей средой (ENVVIRONMENT).
- Имя маршрута сигнала (*Sigroutename*) является именем (*Name*).

50	*Connect*$_0$	::	*Connectpoint*$_0$ *Id*$_0^+$
51	*Connectpoint*$_0$	=	(*Id*$_0$ | ENV)

- Соединение (*Connect*) содержит точку соединения (*Connectpoint*) и список идентификаторов (*Id*), к которым она присоединена. Заметим, что *Connect*$_0$ обозначает соединение оконечной точки канала, соединение канала к маршруту, соединение сигнального маршрута и канальное соединение, так как эти четыре типа соединений неразличимы синтаксически, если они появляются в определении отбора.
- Точка соединения (*Connectpoint*) является идентификатором (*Id*) или, в случае соединения оконечной точки канала, окружающей средой ENV.

52	*Sigdef*$_0$	::	*Sigelem*$_0^+$
53	*Sigelem*$_0$	::	*Signame*$_0$ *Sortid*$_0$* [*Refinement*$_0$]
54	*Signame*$_0$	=	*Name*$_0$
55	*Sortid*$_0$	=	*Id*$_0$

- Определение сигнала (*Sigdef*) состоит из списка элементов сигнала (*Sigelem*).
- Элемент сигнала (*Sigelem*) состоит из имени сигнала (*Signame*), списка идентификаторов сорта (*Sortid*) и необязательной части уточнения (*Refinement*).
- Имя сигнала (*Signame*) является именем (*Name*).
- Идентификатор сорта (*Sortid*) является идентификатором (*Id*).

56	*Signallist*$_0$	::	*Signallistname*$_0$ *Signallist*$_0$
57	*Signallistname*$_0$	=	*Name*$_0$
58	*Signallist*$_0$	=	(*Signallistid*$_0$ | *Sigid*$_0$)$^+$
59	*Signallistid*$_0$	::	*Id*$_0$
60	*Sigid*$_0$	=	*Id*$_0$

- Определение списка сигналов (*Signallistdef*) состоит из имени списка сигнала (*Signallistname*) и списка сигналов (*Signallist*).
- Имя списка сигналов (*Signallistname*) является именем (*Name*).
- Список сигналов (*Signallist*) является непустым списком идентификаторов списка сигнала (Signallistid) и идентификаторов сигнала (*Sigid*).
- Идентификатор списка сигналов (*Signallistid*) содержит идентификатор (*Id*).
- Идентификатор сигналов (*Sigid*) является идентификатором (*Id*).

Заметим, что поскольку идентификатор списка сигналов (*Signallistid*) является деревом, всегда возможно отличить идентификаторы списка сигналов от идентификаторов сигнала (так же, как в случае конкретного синтаксиса).

61	*Vardef*$_0$	::	[REVEALED] [EXPORTED] *Vardefelem*$_0^+$
62	*Vardefelem*$_0$	::	*Varname*$_0^+$ *Sortid*$_0$ [*Value*$_0$]
63	*Varname*$_0$	=	*Name*$_0$

- Определение переменной (*Vardef*) состоит из необязательного атрибута раскрытия (REVEALED), необязательного атрибута экспорта (EXPORTED) и списка элементов определения переменной (*Vardefelem*).
- Элемент определения переменной (*Vardefelem*) состоит из списка имен переменной (*Varname*), идентификатора сорта (*Sortid*) и необязательного начального значения (*Value*).
- Имя переменной (*Varname*) является именем (*Name*).

64	*Viewdef*$_0$	::	*Viewdefelem*$_0^+$
65	*Viewdefelem*$_0$	::	*Varid*$_0^+$ *Sortid*$_0$
66	*Varid*$_0$	=	*Id*$_0$

- Определение обозревания (*Viewdef*) состоит из списка элементов определения обозревания (*Viewdefelem*).
- Элемент определения обозревания (*Viewdefelem*) состоит из списка обозревания идентификаторов переменной (*Varid*) и идентификатора сорта (*Sortid*).
- Идентификатор переменной (*Varid*) является идентификатором (*Id*).

67 *Body*$_0$ :: *Transition*$_0$ *Statebody*$_0$*
68 *Statebody*$_0$ :: (*Statenamelist*$_0$ | *Starredlist*$_0$) *Statespec*$_0$* [*Tailname*$_0$]
69 *Statenamelist*$_0$ :: *Name*$_0$
70 *Statename*$_0$ = *Name*$_0$
71 *Starredlist*$_0$ :: [*Statenamelist*$_0$]

- Тело (*Body*) состоит из начального перехода (*Transition*) и списка тел состояния (*Statebody*).
- Состояние тела (*Statebody*) содержит список имен состояния (*Statenamelist*) или список имен со звездочкой (*Starredlist*), с последующим списком спецификаций стимула состояния (*Statespec*) и необязательное конечное имя (*Tailname*), завершающее тело состояния (*Statebody*).
- Список имен состояния (*Statenamelist*) состоит из списка имен состояния (*Statename*).
- Имя состояния (*Statename*) является именем (*Name*).
- Список имен со звездочкой (*Starredlist*) содержит, возможно пустой (если nil), список имен состояния (*Statenamelist*).

72 *Statespec*$_0$ = *Savespec*$_0$ | *Inputspec*$_0$ | *Contspec*$_0$ | *Priinput*$_0$

- Спецификация стимула состояния (*Statespec*) является или спецификацией сохранения (*Savespec*), или спецификацией ввода (*Inputspec*), или спецификацией непрерывного сигнала (*Contspec*), или спецификацией ввода приоритета (*Priinput*).

73 *Savespec*$_0$ :: *Signallist*$_0$ | *Starred*$_0$
74 *Inputspec*$_0$ :: (*Starred*$_0$ | *Inputvars*$_0^+$) *Enabling*$_0$ *Transition*$_0$
75 *Starred*$_0$ :: ( )
76 *Inputvars*$_0$ :: *Sigid*$_0$ [*Varid*$_0$]*

- Спецификация сохранения (*Savespec*) содержит или список сигналов (*Signallist*) или звездочку.
- Спецификация ввода (*Inputspec*) содержит или звездочку (т.е. дерево со звездочкой (*Starred*), обозначающее звездочку), или непустой список переменных сигналов ввода (*Inputvars*) с последующим условием разрешения (*Enabling*), сопровождаемым переходом (*Transition*).
- Переменные сигналов ввода (*Inputvars*) содержат идентификатор сигнала (*Sigid*) и список необязательных идентификаторов переменной (*Varid*).

77 *Transition*$_0$ :: *Actstmt*$_0$* [*Termstmt*$_0$]
78 *Actstmt*$_0$ :: [*Label*$_0$] *Act*$_0$
79 *Label*$_0$ = *Name*$_0$

- Переход (*Transition*) содержит список предложений действия (*Actstmt*) и необязательное предложение окончания (*Termstmt*).
  Если предложение окончания опущено, то список предложений действия непустой, и если этот список пустой, то предложение действия присутствует (оно не требует проверки, так как является контекстно независимым конкретным синтаксическим правилом).
- Предложение действия (*Actstmt*) содержит необязательную метку (*Label*) и действие (*Act*).
- Метка (*Label*) является именем (*Name*).

80 Act_0 = $Task_0$ | $Output_0$ | $Create_0$ | $Decision_0$ | $Export_0$ | $Option_0$ | $Call_0$ | $Prioutput_0$ | Set_0 | $Reset_0$

- Действие (*Act*) является либо работой (*Task*), либо выводом (*Output*), либо запросом на создание (*Create*), либо принятием решения (*Decision*), либо экспортом (*Export*), либо необязательным переходом (*Option*), либо вызовом процедуры (*Call*), либо приоритетным выводом (*Prioutput*), либо установкой таймера (*Set*), либо сбросом таймера (*Reset*).

81 $Termstmt_0$ :: [$Label_0$] $Terminator_0$
82 $Terminator_0$ = $Nextstate_0$ | $Join_0$ | $Stop_0$ | $Return_0$
83 $Nextstate_0$ :: [$Statename_0$]
84 $Join_0$ :: $Label_0$
85 $Stop_0$ :: ( )
86 $Return_0$ :: ( )

- Предложение терминатора (*Termstmt*) содержит необязательную метку (*Label*) и терминатор (*Terminator*).
- Терминатор (*Terminator*) является либо следующим состоянием (*Nextstate*), либо перескоком (*Join*), либо символом Стоп (*Stop*), либо возвратом (*Return*).
- Следующее состояние (*Nextstate*) содержит необязательное имя состояния (*Statename*) (nil, если следующее состояние "черточка").
- Перескок (*Join*) содержит метку (*Label*).
- Символ Стоп (*Stop*) не содержит дополнительной информации.
- Возврат (*Return*) не содержит дополнительной информации.

87 $Task_0$ :: $Assignstmt_0^+$ | $Text_0^+$

- Работа (*Task*) содержит либо список предложений присваивания (*Assignstmt*), либо список неформальных текстов (*Text*).

88 $Create_0$ :: $Prid_0$ $Actparmlist_0$
89 $Actparmlist_0$ = [$Expr_0$]*

- Запрос на создание (*Create*) содержит идентификатор процесса (*Prid*) и список фактических параметров (*Actparmlist*).
- Список фактических параметров (*Actparmlist*) является списком необязательных выражений (*Expr*).

90 $Call_0$ :: $Procid_0$ $Actparmlist_0$

- Вызов процедуры (*Call*) содержит идентификатор процедуры (*Procid*) и список фактических параметров (*Actparmlist*).

91 $Output_0$ :: $Outputsig_0^+$ [$Piexpr_0$] Via_0
92 $Outputsig_0$ :: $Sigid_0$ $Actparmlist_0$
93 $Piexpr_0$ = $Expr_0$ | $Scopeexpr_0$
94 Via_0 = Id_0*
95 $Scopeexpr_0$ :: $Scope_0$ $Expr_0$
96 $Scope_0$ = $Qualifier_0$

- Вывод (*Output*) содержит список сигналов вывода (*Outputsig*), необязательное выражение *PId* (*Piexpr*) и *Via*.
- Сигнал вывода (*Outputsig*) состоит из идентификатора сигнала (*Sigid*) и списка фактических параметров (*Actparmlist*).
- *PId* выражение (*Piexpr*) является либо выражением (*Expr*), либо выражением подразделения (*Scopeexpr*).
- Направить через (*Via*) является, возможно пустым, списком идентификаторов (*Id*).

- Выражение подразделения (*Scopeexpr*) не имеет соответствующей конструкции в конкретном синтаксисе. Это полезная конструкция, которая введена в AS_0 для того, чтобы облегчить трансформацию сервиса.

  Когда сервисы преобразованы, они сливаются в один граф процесса AS_0. Кроме перехода, содержащего вывод, принятия решений, вытекающих из разрешающего условия и непрерывных сигналов, каждый переход в результирующем графе процесса AS_0 может быть трансформирован в контекст определенного сервиса. Подразделение (*Scope*) в выражении подразделения (*Scopeexpr*) является квалификатором (*Qualifier*), который указывает сервис, который "владеет" выражением (*Expr*), появляющимся в принятии решений в разрешающем условии.

  Выражения подразделения (*Scopeexprs*) генерируются во время расширения разрешающего условия и непрерывного сигнала.

97	*Decision*$_0$	::	*Question*$_0$ *Answer*$_0^+$ [*Elsepart*$_0$]
98	*Question*$_0$	=	*Expr*$_0$ | *Scopeexpr*$_0$
99	*Elsepart*$_0$	::	[*Transition*$_0$]
100	*Answer*$_0$	::	*Conditionlist*$_0$ [*Transition*$_0$]

- Принятие решения (*Decision*) содержит вопрос (*Question*), непустой список ответов (*Answers*) и необязательную часть "иначе" (*Else*).
- Вопрос (*Question*) является либо выражением (*Expr*) или выражением подразделения (*Scopeexpr*) (объяснено выше).
- Часть "*else*" содержит необязательный переход (*Transition*).
- Ответ (*Answer*) содержит список условий (*Conditionlist*) и необязательный переход (*Transition*).

101	*Timerdef*$_0$	::	*Timerelem*$_0^+$
102	*Timerelem*$_0$	::	*Timername*$_0$ *Sortid*$_0$
103	*Timername*$_0$	=	*Name*$_0$

- Определение таймера (*Timerdef*) состоит из списка элементов таймера (*Timerelem*).
- Элемент таймера (*Timerelem*) содержит имя таймера (*Timername*) и список идентификаторов сорта (*Sortid*).
- Имя таймера (*Timername*) является именем (*Name*).

104	*Reset*$_0$	::	*Resetelem*$_0^+$
105	*Resetelem*$_0$	::	*Timerid*$_0$ *Expr*$_0^*$
106	*Timerid*$_0$	=	*Id*$_0$

- Сброс таймера (*Reset*) содержит список элементов сброса таймера (*Resetelem*).
- Элемент сброса таймера (*Resetelem*) содержит идентификатор таймера (*Timerid*) и, возможно пустой, список выражений (*Expr*).
- Идентификатор таймера (*Timerid*) является идентификатором (*Id*).

107	*Set*$_0$	::	*Setelem*$_0^+$
108	*Setelem*$_0$	::	*Expr*$_0$ *Timerid*$_0$ *Expr*$_0^*$

- Установка таймера (*Set*) содержит список элементов установки таймера (*Setelem*).
- Элемент установки таймера (*Setelem*) содержит выражение окончания времени (*Expr*), идентификатор таймера (*Timerid*) и, возможно пустой, список выражений (*Expr*).

1.2 Понятие структурирования

1	*Blocksub*$_0$	=	*Blocksubdef*$_0$ | *Blocksubref*$_0$
2	*Blocksubdef*$_0$	::	[*Blocksubid*$_0$] *Blocksubdecl*$_0^+$ [*Tailid*$_0$]
3	*Blocksubid*$_0$	=	*Id*$_0$
4	*Blocksubdecl*$_0$	=	*Signallistdef*$_0$ | *Connect*$_0$ |*Blockdef*$_0$ | *Blockref*$_0$ | *Chandef*$_0$ | *Sigdef*$_0$ | *Datadef*$_0$ | *Select*$_0$
5	*Blocksubref*$_0$	::	*Blocksubname*$_0$
6	*Blocksubname*$_0$	=	*Name*$_0$

- Подструктура блока (*Blocksub*) является либо определением подструктуры блока (*Blocksubdef*), либо ссылкой на подструктуру блока (*Blocksubref*).
- Определение подструктуры блока (*Blocksubdef*) состоит из идентификатора подструктуры блока (*Blocksubid*), списка деклараций подструктуры блока (*Blocksubdecl*) и необязательного идентификатора окончания (*Tailid*), завершающего определение подструктуры блока.
- Идентификатор подструктуры блока (*Blocksubid*) является идентификатором (*Id*).
- Декларация подструктуры блока (*Blocksubdecl*) является либо определением списка сигналов (*Signallistdef*), либо соединением подструктуры блока (*Connect*), либо определением блока (*Blockdef*), либо ссылкой на блок (*Blockref*), либо определением канала (*Chandef*), либо определением сигнала (*Sigdef*), либо определением данных (*Datadef*), либо отбором (*Select*).
- Ссылка на подструктуру блока (*Blocksubref*) содержит имя подструктуры блока (*Blocksubname*).
- Имя подструктуры блока (*Blocksubname*) является именем (*Name*).

7	*Chansub*$_0$	=	*Chansubdef*$_0$ | *Chansubref*$_0$
8	*Chansubdef*$_0$	::	*Chansubid*$_0$ *Chansubdecl*$_0^+$ [*Tailid*$_0$]
9	*Chansubid*$_0$	=	[*Id*$_0$]
10	*Chansubdecl*$_0$	=	*Chandef*$_0$ | *Blockdef*$_0$ | *Sigdef*$_0$ | *Blockref*$_0$ *Signallistdef*$_0$ | *Datadef*$_0$ | *Connect*$_0$ | *Select*$_0$
11	*Chansubref*$_0$	::	*Chansubname*$_0$
12	*Chansubname*$_0$	=	*Name*$_0$

- Подструктура канала (*Chansub*) является либо определением подструктуры канала (*Chansubdef*), либо ссылкой на подструктуру канала (*Chansubref*).
- Определение подструктуры канала (*Chansubdef*) состоит из идентификатора подструктуры канала (*Chansubid*), списка деклараций подструктуры канала (*Chansubdecl*) и необязательного идентификатора окончания (*Tailid*), завершающего определение подструктуры канала.
- Идентификатор подструктуры канала (*Chansubid*) является необязательным идентификатором (*Id*).
- Декларация подструктуры канала (*Chansubdecl*) является либо определением канала (*Chandef*), либо определением блока (*Blockdef*), либо определением сигнала (*Sigdef*), либо ссылкой на блок (*Blockref*), либо определением списка сигналов (*Signallistdef*), либо определением данных (*Datadef*), либо соединением оконечной точки канала (*Connect*), либо отбором (*Select*).
- Ссылка на подструктуру канала (*Chansubref*) содержит имя подструктуры канала (*Chansubname*).
- Имя подструктуры канала (*Chansubname*) является именем (*Name*).

13	*Refinement*$_0$	::	*Subsignal*$_0^+$
14	*Subsignal*$_0$	::	[REVERSE] *Sigdef*$_0$

- Часть уточнения (*Refinement*) содержит список определений подсигналов (*Subsignal*).
- Определение подсигнала (*Subsignal*) содержит флаг, указывающий, является ли подсигнал идущим в обратном направлении, и определение сигнала (*Sigdef*).

1.3 Дополнительные понятия

1 *Select*$_0$ :: *Expr*$_0$ *Decl*$_0$+
2 *Decl*$_0$ = *Blockdef*$_0$ | *Blockref*$_0$ | *Chandef*$_0$ | *Sigdef*$_0$ | *Prdef*$_0$ | *Prref*$_0$ | *Procdef*$_0$ | *Procref*$_0$ | *Servicedef*$_0$ | *Servicerefo* | *Datadef*$_0$ | *Sigroutedef*$_0$ | *Signallistdef*$_0$ | *Vardef*$_0$ | *Importdef*$_0$ | *Viewdef*$_0$ | *Timerdef*$_0$ | *Connect*$_0$ | *Select*$_0$

- Определение отбора (*Select*) состоит из выражения (*Exp*) и списка деклараций (*Decl*).
- Декларация (*Decl*) - это либо определение блока (*Blockdef*), ссылка на блок (*Blockref*), определение канала (*Chandef*), определение сигнала (*Sigdef*), определение процесса (*Prdef*), ссылка на процесс (*Prref*), определение процедуры (*Procdef*), ссылка на процедуру (*Procref*), определение сервиса (*Servicedef*), ссылка на сервис (*Servicerefo*), определение данных (*Datadef*), определение маршрута сигнала (*Sigroutedef*), определение списка сигналов (*Signallistdef*), определение переменных (*Vardef*), определение импорта (*Importdef*), определение обозревания (*Viewdef*), определение таймера (*Timerdef*), соединение (*Connect*) либо определение отбора (*Select*).

3 *Option*$_0$ :: *Question*$_0$ *Answer*$_0$ + [*Elsepart*$_0$]

- Выбор перехода (*Option*) содержит вопрос (*Question*), список ответов (*Answer*) и необязательную часть *Else* (*Elsepart*).

4 *Decomposition*$_0$ :: *Decompositiondecl*$_0$ +
5 *Decompositiondecl*$_0$ = *Sigrouteddef*$_0$ | *Connect*$_0$ | *Servicedef*$_0$ | *Servisereft*$_0$ | *Select*$_0$

- Декомпозиция (*Decompositon*) содержит список деклараций декомпозиции (*Decompositiondecl*).
- Декларация декомпозиции (*Decompositon*) - это либо определение маршрута сигнала (*Sigroutdef*), соединение маршрута сигнала (*Connect*), определение сервиса (*Servicedef*), ссылка на сервис (*Servicerefo*), либо определение отбора (*Select*).

6 *Servicedef*$_0$ :: *Serviceid*$_0$ [*Inputset*$_0$] *Servicedecl*$_0$ * *Body*$_0$ [*Tailid*$_0$]
7 *Servicedecl*$_0$ = *Vardef*$_0$ | *Procdef*$_0$ | *Procref*$_0$ | *Viewdef*$_0$ | *Importdef*$_0$ | *Timerdef*$_0$ | *Datadef*$_0$ | *Select*$_0$
8 *Serviceid*$_0$ = *Id*$_0$
9 *Servicerefo* :: *Servicename*$_0$
10 *Servicename*$_0$ = *Name*$_0$

- Определение сервиса (*Servicedef*) состоит из идентификатора сервиса (*Serviceid*), необязательной допустимой совокупности входных сигналов (*Inputset*), списка деклараций сервиса (*Servicedecl*), тела сервиса (*Body*) и необязательного конечного идентификатора (*Tailid*), заканчивающего определение сервиса (*Servicedef*).
- Декларация сервиса (*Servicedecl*) - это либо определение переменной (*Vardef*), определение процедуры (*Procdef*), ссылка на процедуру (*Procref*), определение обозревания (*Viewdef*), определение импорта (*Importdef*), определение таймера (*Timerdef*), определение данных (*Datadef*) либо определение отбора (*Select*).
- Идентификатор сервиса (*Serviceid*) - это идентификатор (*Id*).
- Ссылка на сервис (*Servicref*) содержит имя сервиса (*Servicename*).
- Имя сервиса (*Servicename*) - это имя (*Name*)

11 *Priinput*$_0$ :: *Inputvars*$_0$$^+$ *Transition*$_0$
12 *Prioutput*$_0$ :: *Outputsig*$_0$$^+$

- Приоритетность ввода (*Priinput*) состоит из списка переменных сигналов ввода (*Inputvars*) и перехода (*Transition*).
- Приоритетность вывода (*Prioutput*) состоит из списка сигналов вывода (*Outputsig*).

13 *Contspec*$_0$ :: *Expr*$_0$ *Priority*$_0$ *Transition*$_0$
14 *Priority*$_0$ = [*Name*$_0$]
15 *Enabling*$_0$ = [*Expr*$_0$]

- Определение непрерывного сигнала (*Contspec*) состоит из выражения (*Expr*), приоритета (*Priority*) и перехода (*Transition*).
- Приоритет - это необязательное имя целого литерала (*Name*).
- Разрешающее условие (*Enabling*) - это необязательное выражение (*Expr*).

16 *Export*$_0$ :: *Varid*$_0$$^+$

- Экспорт (*Export*) содержит список идентификаторов экспортируемых переменных (*Varid*).

17 *Importdef*$_0$ :: *Importelem*$_0$ $^+$
18 *Importelem*$_0$ :: *Varnemes*$_0$ $^+$ *Sortid*$_0$
19 *Imortexpr*$_0$ :: *Varid*$_0$ [*Expr*$_0$]

- Определение импорта (*Importdef*) содержит список элементов импорта (*Importelem*).
- Элемент импорта (*Importelem*) - это список импортируемых имен переменных (*Varname*) и идентификатора сорта (*Sortid*).
- Выражение импорта (*Importexpr*) содержит идентификатор импортируемой переменной (*Varid*) и необязательное выражение *PId* (*Expr*).

1.4 Данные

1 *Partialtypedef*$_0$ :: *Sortname*$_0$ [*Extproperties*$_0$] *Properties*$_0$ [*Conditionlist*$_0$] [*Tailname*$_0$]
2 *Sortname*$_0$ = *Name*$_0$

- Частичное определение типа (*Partialtypedef*) состоит из имени сорта (*Sortname*), некоторых необязательных дополнительных свойств (*Extproperties*), некоторых свойств (*Properties*), необязательного списка условий в диапазоне (*Conditionlist*) и необязательного имени сорта (*Tailname*), завершающего определение.

  Необходимо заметить, что в отличие от синтаксических правил Z.100, это определение также покрывает подтипы, где подразумевается частичное определение типа (т.е. в случае, когда *Conditionlist*$_0$ отлично от нуля nil).
- Имя сорта (*Sortname*) есть имя (*Name*).

3 *Properties*$_0$ :: *Literal** *Op*$_0$* *Axiom*$_0$* *Mappingaxiom*$_0$* [*Initialvalue*$_0$]

- Свойства (*Properties*) состоят из списка сигнатур литералов (*Literal*), списка сигнатур операторов (*Op*), списка аксиом (*Axioms*), списка литералов аксиом отображения (*Mapping axioms*) и необязательного начального значения (*Initialvalue*) (по умолчанию).

4 *Literal*$_0$ = *Name*$_0$ | *String*$_0$ | *Nmclass*$_0$

- Сигнатура литерала есть или имя (*Name*), строка символов (*String*) или имя класса (*Nameclass*).

5 *Op*$_0$ = *Ordering* | *Opspec*$_0$
6 *Opspec*$_0$ :: *Operatorname*$_0$ *Sortid*$_0$ $^+$ *Sortid*$_0$
7 *Operatorname*$_0$ = *Name*$_0$ | *Quotedop*$_0$

- Сигнатура оператора (*Op*) - это или упорядочение (*Ordering*) или спецификация оператора (*Opspec*).
- Спецификация оператора (*Opspec*) состоит из имени оператора (*Operatorname*), списка аргументов идентификаторов сорта (*Sortid*) и идентификатора сорта результата (*Sortid*).
- Имя оператора (*Op*) есть имя (*Name*) или оператор в кавычках (*Quoted*).

8 *Axiom*$_0$ = *Unquantequation*$_0$ | *Condequation*$_0$ | *Quantequation*$_0$
9 *Unquantequation*$_0$ = *Equation*$_0$ | *Term*$_0$

- Аксиома (*Axiom*) есть либо неквантифицированное равенство (*Unquantequation*), либо условное равенство (*Condequation*), либо квантифицированное равенство (*Quantequation*), либо терм (*Term*).
- Неквантифицированное равенство (*Unquantequation*) есть либо равенство (*Equation*), либо терм (*Term*).

10 *Equation*$_0$ :: *Term*$_0$ *Term*$_0$

- Равенство (*Equation*) состоит из левого терма (*Term*) и правого терма (*Term*).

11 *Quantequation*$_0$ :: *Valuename*$_0$ $^+$ *Sortid*$_0$ *Axiom*$_0$ $^+$
12 *Valuename*$_0$ = *Name*$_0$

- Квантифицированное равенство (*Quantequation*) состоит из списка имен значений (*Valuename*), идентификатора сорта (*Sortid*) и списка аксиом (*Axiom*).
- Имя значения (*Valuename*) есть имя (*Name*).

13 *Term*$_0$ = *Id*$_0$ | *Operatorterm*$_0$ | *Condterm*$_0$ | *Stringterm*$_0$
Monadterm$_0$ | *Infixterm*$_0$ | *Errorterm*$_0$ |*Spellingterm*$_0$

- Терм (*Term*) есть или идентификатор (*Id*), терм оператора (*Opratorterm*), условный терм (*Condterm*), терм символьной строки (*Stringterm*), одноместный терм (*Monadterm*), инфиксный терм (*Infixterm*), терм-ошибка (*Errorterm*) или считывающий терм (*Spellingterm*).

  Необходимо заметить, что в AS$_0$ не может быть сделано никаких различий между разными типами операторов в термах и выражениях, которые отличаются только семантически. Например, (расширенный составной терм) и (расширенный пассивный терм), как определено в §5.4 Z.100, не могут быть различены синтаксически (т.е. без связи имен с определениями), что означает, что они покрываются одной и той же областью определения в AS$_0$.

14 *Operatorterm*$_0$ :: (*Id*$_0$ | *Qualop*$_0$) *Term*$_0$ $^+$

- Терм оператора (*Operatorterm*) состоит из идентификатора (*Id*) или квалифицированного оператора (*Qualop*) и следующего за ним списка термов аргумента (*Term*).

15 *Condterm*$_0$ :: *Term*$_0$ *Term*$_0$ *Term*$_0$

- Условный терм (*Condterm*) состоит из терма условий (*Term*), терма следствия (*Term*) и терма альтернативы (*Term*).

16 *Stringterm*$_0$ :: *Qualifier*$_0$ *String*$_0$

- Терм символьной строки (*Stringterm*) состоит из квалификатора (*Qualifier*) и символьной строки (*String*).

17 *Monadterm*$_0$ :: (NOT | MINUS) *Term*$_0$

- Одноместный терм (*Monadic*) состоит из одного из операторов "NOT" или "-", предшествующих терму аргумента (*Term*).

18 *Infixterm*$_0$ :: *Term*$_0$ *Infixop*$_0$ *Term*$_0$
19 *Infixop*$_0$ = IMPLY | OR | XOR | AND | IN | MOD | REM | PLUS | MINUS | CONC | MULT| DIV | *Relop*$_0$
20 *Relop*$_0$ = NE | EQ | GT | LT | LE | GE

- Инфиксный терм (*Infixterm*) состоит из двух термов аргумента (*Term*) и инфиксного оператора (*Infix*).
- Инфиксный оператор есть либо "=>", "OR", "XOR", "AND", "IN", "MOD", "REM", "+", "-", "//", "*", "/", либо один из операторов отношения (*Relop*).
- Оператор отношения (*Relop*) есть либо "/=","=",">", "<", "<=" либо ">=".

Необходимо заметить, что группировка любых "поддеревьев" отражает предшествующие правила, как определено в §5.4.2.2 Z.100.

21 *Errorterm*$_0$ :: ()

- Терм ошибки (*Errorterm*) не содержит добавочной информации.

22 *Condequation*$_0$ :: *Unquantequation*$_0^+$ *Unquantequation*$_0$

- Условное равенствое (*Condequation*) состоит из списка ограничений неквантифицированных равенств (*Unquantequation*) и ограничивающего неквантифицированного равенства (*Unquantequation*).

23 *Extproperties*$_0$ = *Struc*$_0$ | *Inherited*$_0$ | *Geninstlist*$_0$

- Расширенные свойства (*Extproperties*) есть либо свойства структуры (*Struc*), наследуемые свойства (*Inherited*), либо свойства списка реализации генератора (*Geninstlist*).

24 *Syntypedef*$_0$ :: *Syntypename*$_0$ *Parentid*$_0$ [*Initialvalue*$_0$] [*Conditionlist*$_0$] [*Tailname*$_0$]
25 *Syntypename*$_0$ = *Name*$_0$

- Определение подтипа (*Syntypedef*) состоит из имени подтипа (*Syntypename*), родительского идентификатора (*Parentid*), необязательного начального значения (*Initialvalue*), необязательного диапазона списка условий (*Conditionlist*) и имени подтипа (*Tailname*), завершающего определение.
- Имя подтипа (*Syntype*) есть имя (*Name*).

26 *Conditionlist*$_0$ = *Condition*$_0$ $^+$

27 *Condition*$_0$ :: [*Value*$_0$ | *Relop*$_0$] *Value*$_0$

28 *Value* = *Expr*$_0$

- Диапазон списока условий (*Conditionlist*) есть список условий на диапазон (*Condition*).
- Диапазон условий состоит из необязательного значения (*Value*) или оператора отношений (*Relop*) и сопровождается значением (*Value*).
- Значение (*Value*) есть выражение (*Expression*).

29	*Struc*$_0$	::	*Fieldspec*$_0$$^+$
30	*Fieldspec*$_0$	::	*Fieldname*$_0$ $^+$ *Sortid*$_0$
31	*Fieldname*$_0$	=	*Name*$_0$

- Свойства структуры (*Struc*) состоят из списка спецификаций полей (*Fieldspec*).
- Спецификация поля (*Fieldspec*) состоит из списка имен поля (*Fieldname*) и идентификатора сорта (*Sortid*).
- Имя поля (*Fieldname*) есть имя (*Name*).

32	*Inherited*	::	*Parentid*$_0$ *Literalrenaming*$_0$ (ALL | *Operatorrenaming*$_0$)
33	*Parentid*$_0$	=	*Id*$_0$
34	*Literalrenaming*$_0$	=	*Literalpair*$_0$*
35	*Literalpair*$_0$	::	*Newliteral*$_0$ *Oldliteral*$_0$
36	*Newliteral*$_0$	=	*Name*$_0$ | *String*$_0$
37	*Oldliteral*$_0$	=	*Name*$_0$ | *String*$_0$
38	*Operatorrenaming*$_0$	=	*Operatorpair*$_0$*
39	*Operatorpair*$_0$	=	*Newoperator*$_0$ *Oldoperator*$_0$
40	*Newoperator*$_0$	=	[*Opetratorname*$_0$]
41	*Oldoperator*$_0$	=	*Operatorname*$_0$

- Свойства наследования (*Inherited*) состоят из родительского идентификатора (*Parentid*), переименования литералов (*Literalrenaming*) и наследуемых операторов, которые есть или операторы ALL, либо переименование оператора (*Operatorrenaming*).
- Родительский идентификатор (*Parentid*) есть идентификатор (*Identifier*).
- Переименование литерала (*Literalrenaming*) есть список пар литералов (*Literalpair*).
- Пара литералов (*Literalpair*) состоит из нового имени литерала (*Newliteral*) и старого (родительского) имени литерала (*Oldliteral*).
- Новое имя литерала (*Newliteral*) есть или имя (*Name*) или символьная строка (*String*).
- Старое имя литерала (*Oldliteral*) есть или имя (*Name*) или символьная строка (*String*).
- Переименование оператора (*Operatorrenaming*) есть список пар операторов (*Operatorpair*).
- Пара операторов (*Opertorpair*) состоит из нового имени оператора (*Newoperator*) и старого имени оператора (*Oldoperator*).
- Новое имя оператора (*Newoperator*) есть имя оператора (*Operatorname*), если оно определено.
- Старое (родительское) имя оператора (*Oldoperator*) есть имя оператора (*Operatorname*).

42	*Sortgenerator*$_0$	::	*Generatorname*$_0$ *Genparm*$_0$$^+$ [*Geninstlist*]*Properties*$_0$ [*Tailname*$_0$]
43	*Generatorname*$_0$	=	*Name*$_0$
44	*Genparm*$_0$	=	*Sortparm*$_0$ | *Termparm*$_0$ | *Litparm*$_0$ | *Opparm*$_0$
45	*Sortparm*$_0$	::	*Name*$_0$$^+$
46	*Termparm*$_0$	::	*Name*$_0$$^+$
47	*Litparm*$_0$	::	*Name*$_0$$^+$
48	*Opparm*$_0$	::	*Name*$_0$$^+$

- Генератор сорта (*Sortgenerator*) состоит из имени генератора (*Generatorname*), списка формальных параметров генератора (*Genparam*), возможно пустого списка реализаций генератора (*Geninstlist*), некоторых свойств (*Properties*) и имени генератора сорта (*Tailname*), завершающего определение.
- Имя генератора (*Generatorname*) есть имя (*Name*).
- Формальный параметр генератора (*Genparam*) есть или параметр сорта (*Sortparm*), параметр терма (*Termparm*), параметр литерала (*Litparm*) или параметр оператора (*Opparm*).
- Параметр сорта (*Sortparam*) состоит из списка имен (*Name*).

- Параметр терма (*Termparm*) состоит из списка имен (*Name*).
- Параметр литерала (*Litparm*) состоит из списка имен (*Name*).
- Параметр оператора (*Opparm*) состоит из списка имен (*Name*).

49	*Geninstlist*$_0$	::	*Geninst*$_0^+$
50	*Geninst*$_0$	::	*Generatorid*$_0$ *Genactparm*$_0^+$
51	*Generatorid*$_0$	=	*Id*$_0$
52	*Genactparm*$_0$	=	*Term*$_0$ | *Quotedop*$_0$ | *Nmclass*$_0$

- Список реализаций генератора (*Geninstlist*) состоит из списка реализаций генератора (*Geninst*).
- Реализация генератора (*Geninst*) состоит из идентификатора генератора (*Generatorid*) и списка фактических параметров генератора (*Genactparm*).
- Идентификатор генератора (*Generatorid*) есть идентификатор (*Id*).
- Фактический параметр генератора (*Genactparm*) есть терм (Term) или оператор в кавычках (*Quotedop*) или имя класса (*Nmclass*). Т.е. идентификаторы сорта (*Sortid*) и имена операторов (*Operatornam*) синтаксически есть термы (*Term*), подобно другим типам фактических параметров. Операторы в кавычках (*Quotedop*) и имя класса (*Nmclass*), напротив, не могут формировать собственный терм (*Term*) и, следовательно, должны быть явно определены как альтернативы *Genactparm*$_0$.

53	*Synonymdef*$_0$	::	*Synonymname*$_0$ [*Sortid*$_0$] [*Initialvalue*$_0$]
54	*Synonymname*$_0$	=	*Name*$_0$

- Определение синонима (*Synonymdef*) состоит из имени синонима (*Synonymname*) и необязательного идентификатора сорта и необязательного начального (по умолчанию) значения (*Initialvalue*).
- Имя синонима (*Synonymname*) есть имя (*Name*).

55	*Nmclass*$_0$	::	*Regularexp*$_0$
56	*Regularexp*$_0$	=	*Partregexp*$_0$ | *Orregexp*$_0$ | *Andregexp*$_0$
57	*Orregexp*$_0$	::	*Regularexp*$_0$ *Partregexp*$_0$
58	*Andregexp*$_0$	::	*Regularexp*$_0$ *Partregexp*$_0$
59	*Partregexp*$_0$	=	*Rngregexp*$_0$ | *Singregexp*$_0$ | *Parenregexp*$_0$
60	*Rngregexp*$_0$	::	*String*$_0$ *String*$_0$ [*Regexpexp*$_0$]
61	*Singregexp*$_0$	::	*String*$_0$ [*Regexpexp*$_0$]
62	*Parenregexp*$_0$	::	*Regularexp*$_0$ [*Regexpexp*$_0$]
63	*Regexpexp*$_0$	=	MULT | PLUS | *Name*$_0$

- Имя класса (*Nmclass*) состоит из регулярного выражения (*Regularexp*).
- Регулярное выражение (*Regularexp*) есть либо частичное регулярное выражение (*Partregexp*) или инфиксное регулярное выражение *OR* (*Orregexp*) или инфиксное регулярное выражение *AND* (*Andregexp*).
- Инфиксное регулярное выражение *OR* (*Orregexp*) состоит из регулярного выражения (*Regularexp*) и частичного регулярного выражения (*Partregexp*).
- Инфиксное регулярное выражение *AND* (*Andregexp*) состоит из регулярного выражения (*Regularexp*) и частичного регулярного выражения (*Partregexp*).
- Частичное регулярное выражение (*Partregexp*) есть или регулярное выражение диапазона (*Rngregexp*), или одиночное регулярное выражения (*Singregexp*), или родительское регулярное выражение (*Parenregexp*).
- Регулярное выражение диапазона (*Rngregexp*) состоит из двух символьных строк и необязательного выражения повторения (*Regexpexp*).
- Одиночное регулярное выражение (*Singregexp*) состоит из символьной строки и необязательного выражения повторения (*Regexpexp*).

- Родительское регулярное выражение (*Parenregexp*) состоит из регулярного выражения (*Regularexp*) и необязательного регулярного выражения повторения (*Regexpexp*) .

- Регулярное выражение повторения (*Regexpexp*) есть или "*" или "+" или имя (*Name*).

64 *Ordering* :: ()

- Упорядочение (*Ordering*) не содержит дополнительной информации.

65 *Mappingaxiom*$_0$ :: *Valuename*$_0^+$ *Sortid*$_0$ *Literalaxiom*$_0$
66 *Literalaxiom*$_0$ = *Axiom*$_0$ | *Mappingaxiom*$_0$
67 *Spellingterm*$_0$ :: *Id*$_0$

- Аксиома отображения (*Mappingaxiom*) состоит из списка имен значений (*Valuename*), идентификатора сорта (*Sortid*) и списка литеральных аксиом (*Literalaxiom*).

- Литеральная аксиома есть либо аксиома (*Axiom*), либо аксиома отображения (*Mappingaxiom*).

- Считывающий терм *Spellingterm* содержит идентификатор (*Id*).

68 *Expr*$_0$ = *Id*$_0$ | *Stringterm*$_0$ | *Condexpr*$_0$ |
Operatorapp$_0$ | *Monadexpr*$_0$ | *Infixexpr*$_0$ |
Impoperator$_0$ | *Selectexpr*$_0$ | *Tupleexpr*$_0$

- Выражение (*Expr*) есть либо идентификатор (*Id*), либо терм символьной строки (*Stringterm*), либо условное выражение (*Condexpr*), применение оператора (*Operatorapp*), либо одноместное выражение (*Monadexpr*), либо инфиксное выражение (*Infixexpr*), либо императивный оператор (*Impoperator*), либо выражение выбора поля (*Selectexpr*), либо выражение кортежа (*Tupleexpr*).

69 *Monadexpr*$_0$ :: (NOT | MINUS) *Expr*$_0$

- Одноместное выражение (*Monadexpr*) состоит из одного из операторов "NOT" или "-" и следующего за ним выражения аргумента (*Expr*).

70 *Infixexpr*$_0$ :: *Expr*$_0$ *Infixop*$_0$ *Expr*$_0$

- Инфиксное выражение (*Infixexpr*) состоит из двух выражений аргумента (*Expr*) и инфиксного оператора (*Infixop*).

71 *Selectexpr*$_0$ :: *Expr*$_0$ *Name*$_0$

- Выражение выбора поля (*Selectexpr*) состоит из выражения (*Expr*) и имени поля (*Name*).

72 *Tupleexpr*$_0$ :: *Qualifier*$_0$ *Expr*$_0^+$

- Выражение кортежа (*Tupleexpr*) содержит квалификатор (*Qualifier*) и список выражений (*Expr*).

73 *Condexpr*$_0$ :: *Expr*$_0$ *Expr*$_0$ *Expr*$_0$

- Условное выражение (*Condexpr*) состоит из условного выражения (*Expr*), выражения следствия (*Expr*) и альтернативного выражения (*Expr*).

74 *Datadef*$_0$ = *Partialtypedef*$_0$ | *Syntypedef*$_0$ | *Sortgenerator*$_0$ | *Synonymdef*$_0$

- Определение данных (*Datadef*) есть либо частичное определение типа (*Partialtypedef*), определение подтипа (*Syntypedef*), генератор сорта (*Sortgenerator*), либо определение синонима (*Synonymdef*).

75 *Exprlist*$_0$ = *Expr*$_0$*

- Список выражений (*Exprlist*) есть список выражений (*Expr*).

76 *Operatorapp*$_0$ :: (*Expr*$_0$ | *Qualop*$_0$) *Exprlist*$_0$
77 *Qualop*$_0$ :: *Qualifier*$_0$ *Quotedop*$_0$
78 *Quotedop*$_0$ :: *Infixop*$_0$ | NOT

- Применение оператора (*Operatorapp*) содержит выражение (*Expr*) либо квалифицированный оператор (*Qualop*) и последующий список выражений (*Exprlist*). Выражение (*Expr*) в применении оператора (*Operatorapp*) может быть либо идентификатором (*Id*) (обозначающим идентификатор оператора (*Id*)), либо представлять <активное выражение>, как определено в §5.4.2.4 и §5.4.2.5 в Z.100.
- Квалифицированный оператор (*Qualop*) содержит квалификатор (*Qualifier*) и оператор в кавычках (*Quotedop*).
- Оператор в кавычках (*Quotedop*) есть либо инфиксный оператор (*Infixop*), либо "NOT".

79 *Assignstmt*$_0$ :: *Variable*$_0$ *Expr*$_0$
80 *Variable*$_0$ = *Varid*$_0$ | *Indexedvar*$_0$ | *Fieldvar*$_0$
81 *Indexedvar*$_0$ :: *Variable*$_0$ *Exprlist*$_0$
82 *Fieldvar*$_0$ :: *Variable*$_0$ *Name*$_0$

- Предложение присваивания (*Assignstmt*) содержит переменную (*Variable*) и выражение (*Expr*).
- Переменная (*Variable*) есть либо идентификатор переменной (*Varid*), либо индексированная переменная (*Indexedvar*), либо переменная поля (*Fieldvar*).
- Индексированная переменная (*Indexedvar*) состоит из переменной (*Variable*) и списка выражений (*Exprlist*). Необходимо заметить, что индексированная переменная (*Indexedvar*) может обозначать переменную поля (*Fieldvar*), например:

  v(a) := ...

  может (в зависимости от контекста) записана другим способом

  v ! a := ...

- Переменная поля (*Fieldvar*) состоит из переменной (*Variable*) и имени поля (*Name*).

83 *Viewexpr*$_0$ :: *Varid*$_0$ *Expr*$_0$

- Обозревающее выражение (*Viewexp*) состоит из идентификатора обозревающей переменной (*Varid*) и выражения (*Expr*).

84 *Initialvalue*$_0$ = *Expr*$_0$

- Начальное (по умолчанию) значение (*Initialvalue*) есть выражение (*Expr*).

85 *Impoperator*$_0$ = *Importexpr*$_0$ | *Viewexpr*$_0$ | *Nowexpr*$_0$ | *Activeexpr*$_0$ | *Parentexpr*$_0$ | *Offspringexpr*$_0$ | *Senderexpr*$_0$ | *Selfexpr*$_0$
86 *Nowexpr*$_0$ :: ()
87 *Selfexpr*$_0$ :: ()
88 *Parentexpr*$_0$ :: ()
89 *Offspringexpr*$_0$ :: ()
90 *Senderexpr*$_0$ :: ()
91 *Activeexpr*$_0$ :: *Timerid*$_0$ *Exprlist*$_0$

- Императивная операция (*Impoperator*) есть либо импортирующее выражение (*Importexpr*), обозревающее выражение (*Viewexpr*), выражение "сейчас" (*Nowexpr*), выражение активного таймера (*Activeexpr*), родительского выражение (*Parentexpr*), выражение потомка (*Offspringexpr*), выражение отправителя (*Senderexpr*), либо выражение "сам" (*Selfexpr*).
- Выражение "сейчас" (*Nowexpr*) не содержит дополнительной информации.
- Выражение "сам" (*Selfexpr*) не содержит дополнительной информации.
- "Родительское" выражение (*Parentexpr*) не содержит дополнительной информации.
- Выражение "потомка" (*Offspringexpr*) не содержит дополнительной информации.
- Выражение "отправителя" (*Senderexpr*) не содержит дополнительной информации.
- Выражение активного таймера (*Activeexpr*) содержит идентификатор таймера (*Timerid*) и список выражений (*Exprlist*).

2 Внутренние домены

Семантические домены определяют домен составного объекта, который содержит некоторую полученную (контекстную) информацию (атрибуты), требующуюся во время преобразования. Имеется различие между информацией, относящейся к конкретным объектам, такой как информация об оконечных точках канала, сорте переменных, сорте списка сигналов и т.д., и "общих" данных таких, как информация, о которой блок подразделения окружает список определений, совокупность имен для неявных переменных и т.д. Два вида информации (*Descriptordict* и *Quotdict*) объединяются в один объект *Dict* такой, что только еще один формальный параметр или еще один результат требуется для функционирования при использовании или при его создании, соответственно.

1 *Dict* = *Descriptordict* ∪ *Quotdict*

Когда объект *Dict* задан как формальный параметр, то он называется условно *dict* и включается во второй список аргументов в заголовках функций.

Ниже детально описываются эти два домена.

2.1 Описание *Descriptordict*

Descriptordict- это отображение идентификаторов на свои дискрипторы.

1	*Descriptordict*	=	*Qual* ⇸ *Descr*
2	*Qual*	=	(*Qualelem* | *Operatorqualelem* | *Importqualelem* | *Viewqualelem*)+

Qual обозначает внутреннее представление идентификаторов AS_0. В SDL только идентификаторы AS_0 являются уникальными внутри одного и того же класса объектов, так что для того, чтобы включить все идентификаторы в одну и туже таблицу отображений, класс объекта должен быть частью входов таблицы. Следовательно, *Qual* состоят из списка элементов квалификатора, при этом конечный элемент квалификатора в списке указывает на класс объекта и имя объекта. *Operatorqualelem*, *Importqualelem* и *Viewqualelem* обрабатываются специальным образом, так как требуется дополнительная информация для того, чтобы гарантировать уникальность этих объектов.

пример:

Блок BL в системе SYS имеет *Qual* в *Descriptordict*, который есть

⟨(SYSTEM, **mk**-$Name_0$("SYS",nil)),(BLOCK, **mk**-$Name_0$("BL",nil))⟩

Сигнал SIG в блоке BL имеет *Qual*, который есть

⟨(SYSTEM, **mk**-$Name_0$("SYS",nil)),(BLOCK, **mk**-$Name_0$("BL",nil)),(SIGNAL, **mk**-$Name_0$("SIG",nil))⟩

3	*Qualelem*	=	*Kind* ($Name_0$ | $String_0$)
4	*Kind*	=	$Scopeunit_0$ | *Entity*
5	*Entity*	=	SIGNALROUTE | CHANNEL | SIGNALLIST | GENERATOR | VALUE | LITERAL
6	*Operatorqualelem*	=	OPERATOR (($Name_0$ | $Quotedop_0$) $Sortqual^+$ *Sortqual*)
7	*Importqualelem*	=	IMPORT ($Name_0$ *Sortqual*)
8	*Viewqualelem*	=	VIEW *Qual*

Qualelem есть пара типов *Scopeunit* и имени. Когда он появляется в качестве конечного *Qualelem* в списке (в *Qual*), то он обозначает класс объекта и имя объекта. *Kind*, который является *Entity*, всегда обозначает класс объекта, т.е. он всегда появляется в качестве оконечного *Qualelem* в *Qual*. Всегда есть специальные *Qualelem*, *Operatorqualelem*, *Importqualelem* и *Viewqualelem*. Класс объекта VALUE обозначает переменные, синонимы и идентификаторы значения.

Информация (т.е.*Operatorqualelem*), которая делает оператор уникальным (в "частичном типе данных" блоков подразделения) есть имя ($Name_0$) или оператор в кавычках ($Quotedop_0$), сорта аргументов ($Sortqual^+$) и сорт результата (*Sortqual*).

Информация (т.е. *Importqualelem*), которая делает импортированный объект уникальным в блоке подразделения, есть имя ($Name_0$) импортированного объекта вместе с его сортом (*Sortqual*).

Информация (т.е. *Viewqualelem*), которая делает обозреваемую переменную уникальной в процессе определения, есть идентификатор (*Qual*) обозреваемой переменной.

9	*Descr*	=	*SystemD* | *BlockD* | *ChannelD* | *SignalD* | *TimerD* | *SignalrouteD* | *SignallistD* | *ProcedureD* | *ProcessD* | *SortD* | *SyntypeD* | *GeneratorD* | *SynD* | *VarD* | *ImportD* | *ViewD* | *LiteralD* | *OperatorD* | *ValueidD* |

Descr в *Descriptordict* может быть дескриптором одного из: система (или самый удаленный уровень), блок, канал, таймер, маршрут сигнала, список сигналов, процедура, процесс, сорт, подтип, генератор, синоним, переменная, импортируемая переменная, обозреваемая переменная, литерал, оператор, аксиомная переменная (идентификатор значения), сервис, подструктура блока, подструктура канала или рекурсивный дескриптор.

Следующее утверждение Meta-IV об отношении между *Quals* и связанных с ними *Descr* в *Dict* всегда применимо:

is-consistent-Dict(*dict*) ≜ (2.1.1)

```
(∀qual ∈ dom dict)
 ((let (kind,) = qual[len qual] in
   cases kind:
     (SYSTEM
       → is-SystemD(dict(qual)),
     BLOCK
       → is-BlockD(dict(qual)),
     CHANNEL
       → is-ChannelD(dict(qual)),
     SIGNAL
       → is-SignalD(dict(qual)) ∨ is-TimerD(dict(qual)),
     SIGNALROUTE
       → is-Signalroute(dict(qual)),
     SIGNALLIST
       → is-SignallistD(dict(qual)) ∨ is-ErrorD(dict(qual)),
     PROCERDURE
       → is-ProcedureD(dict(qual)),
     PROCESS
       → is-ProcessD(dict(qual)),
     TYPE
       → is-SortD(dict(qual)) ∨ is-SyntypeD(dict(qual)),
     VALUE
       → is-VarD(dict(qual)) ∨ is-SynD(dict(qual)) ∨
         is-ValueidD(dict(qual)) ∨ is-ErrorD(dict(qual)),
     GENERATOR
       → is-GeneratorD(dict(qual)) ∨ is-ErrorD(dict(qual)),
     SUBSTRUCTURE
       → is-BlocksubD(dict(qual)) ∨ is-ChannelsubD(dict(qual)),
     LITERAL
       → is-LiteralD(dict(qual)),
     SERVICE
       → is-ServiceD(dict(qual)),
     OPERATOR
       → is-OperatorD(dict(qual)),
     IMPORT
       → is-ImportD(dict(qual)),
     VIEW
       → is-ViewD(dict(qual)))))
```

запись: *Dict* → *Bool*

Приведенная выше функция определяется только в пояснительных целях.

10 *SystemD* :: ()
11 *BlocksubD* :: ()

SystemD это есть дескриптор уровня системы.

BlocksubD есть дескриптор подструктуры блока. Эти дескрипторы присутствуют только потому, что уровень системы и подструктуры блока являются определяющими объектами блоков подразделения и, следовательно, их имена используются в квалификаторах идентификаторов. (Все полностью определенные квалификаторы могут быть взяты в домене *Dict* как блоки подразделения).

12 *ChannelsubD* :: *Blockqual*

ChannelsubD есть дескриптор подструктуры канала.

Blockqual В AS_1 подструктура канала представляется в качестве подструктуры блока определения синтетического блока. *Blockqual* представляет идентификатор этой подструктуры блока. Каждый раз, когда используется идентификатор AS_0, относящийся к подструктуре канала в его квалификационной части, квалификатор модифицируется в соответствии с *Blockqual*.

13 *ServiceD* :: *Transition*$_0$ *Statedict Labeldict Validinputset Priinputset*

14 *Priinputset* = *Signalqual*-set

ServiceD есть дескриптор сервиса SDL.

Transition$_0$ Есть внутренний переход , взятый из определения тела (*Body*) сервиса (*Service*).

Statedict, Labeldict Есть *Statedict* и *Labeldict* для сервиса. (См. описание *Quotdict*).

Validinputset Есть полная совокупность входных сигналов для сервиса (включая таймеры).

Priinputset Есть совокупность приоритетных входных сигналов для сервиса.

15 *BlockD* :: *Exportchannels Importchannels Explicitroutes BlockconnectionD*
16 *Exportchannels* = *ExpimpchanD*
17 *Importchannels* = *ExpimpchanD*
18 *ExpimpchanD* = *NameclosureD* $\rightarrow$ (*Otherend Channame*$_0$)*
19 *Otherend* = *Blockqual* | ENV
20 *Explicitroutes* = *Bool*
21 *BlockconnectionD* = *Channelqual* $\rightarrow$ *Qual*-set
22 *Channelqual* = *Qual*

BlockD есть дескриптор блока.

Exportchannels Содержит информацию о неявных каналах, входящих в блок или исходящих из него, так как в блоке в процессе содержатся экспортные переменные.

Importchannels Содержит информацию о неявных каналах, входящих в блок или исходящих из него, так как в блоке в процессе содержатся импортные переменные. Для неявных каналов, содержащихся в *importchannels*, информация, встречавшаяся в других дескрипторах блока, выводится из *Exportchannels*.

ExpimpchanD Есть таблица связок ввода-вывода (см. определение домена *NameclosureD*) при связи с другой конечной точкой канала (*Otherend*) и двунаправленного канала, который придается к такой связке. Первый список сигналов в канале содержит сигнал xtQUERY, второй список сигналов в канале содержит сигнал xtREPLY.

Explicitroutes Истинно, если явные маршруты сигналов определены для блока. *Explicitroutes* используется для получения полной совокупности допустимых входных сигналов, необходимых для содержащихся процессов.

BlockconnectionD Есть отношение между каналами и маршрутами сигналов, связанными с каналом. *BlockconnectionD* используется для замещения идентификаторов канала идентификаторами маршрута сигнала в конструкции VIA вложенных процессов и для замещения идентификаторов канала, имеющих подструктуру, соответствующими новыми идентификаторами каналов в соединениях.

Channelqual Есть *Qual* канала.

23	*ChannelD*	:: *Endpoint Endpoint Signalqual*-set *Signalqual*-set [*Newchannels*]
24	*Endpoint*	= *Blockqual* | ENV
25	*Signalqual*	= *Qual*
26	*Blockqual*	= *Qual*
27	*Newchannels*	= *Endpoint Channameo Endpoint Channameo*

ChannelD есть дескриптор канала.

Endpoint Есть конечная точка, которая является или идентификатором блока, или средой.

Signalqual-set Есть полная совокупность идентификаторов сигналов, передаваемых от источника (*Origin*) приемнику (*Destination*). В том случае, когда рассматривается двунаправленный канал, представляется другое множество сигналов, обозначающее сигналы, переданные от второго блока *Endpoint* в первый блок *Endpoint*.

Newchannels Если канал содержит подструктуру канала, то канал представляется двумя каналами в AS_1. *Newchannels* содержит имена и исходящие конечные точки этих двух каналов, эта информация используется тогда, когда старый идентификатор канала должен заменяться новым идентификатором канала в множестве VIA.

28	*SignalD*	:: *Sortqual** *Signalqual*-set *Signalqual*-set
29	*Sortqual*	= *Qual*

SignalD есть дескриптор сигнала.

*Sortqual** есть сорт значений, передаваемых сигналом.

Signalqual-set Два множества подсигналов, первое из которых - сигналы, передаваемые в том же направлении, что и "родительский" сигнал, а второе - сигналы, передаваемые в противоположном направлении.

30	*TimerD*	:: *Sortqual** *Newqual*

TimerD это есть дескриптор таймера.

*Sortqual** есть сорт значений, передаваемых таймером.

Newqual Обозначает идентификатор, который должен быть использован для AS_1. Его имя в *Newqual* изменено, если таймер определен в сервисе.

31 *SignalrouteD* :: *Originprocess Destinationprocess Signalqual*-set *Signalqual*-set

32 *Originprocess* = *Processqual* | ENV
33 *Destinationprocess* = *Processqual* | ENV

SignalrouteD есть дескриптор маршрута сигнала.

Originprocess Есть исходящая конечная точка, которая является или идентификатором процесса, или средой.

Destinationprocess Есть конечная точка назначения, которая является или идентификатором процесса, или средой.

Signalqual-set Есть множество идентификаторов сигналов, передаваемых от источника (*Origin*) к приемнику (*Destination*). В случае, когда используется двунаправленный маршрут сигнала, используется другое множество сигналов, обозначающее, что сигналы переданы от процесса *Destination* процессу *Origin*.

34 *SignallistD* :: $Signallist_0$

SignallistD есть дескриптор списка сигналов. Он содержит список идентификаторов сигналов AS_0 (и идентификаторы списка сигналов), прикрепленные к списку сигналов.

35 *ProcedureD* :: *FormparmD* Newqual*
36 *FormparmD* = *InDescr* | *InoutDescr*
37 *InDescr* :: *Sortqual*
38 *InoutDescr* :: *Sortqual*
39 *Newqual* = *Qual*

ProcedureD есть дескриптор процедуры.

*FormparmD** Список специальных дескрипторов, используемый, когда запрашивается процедура, чтобы проверить соответствие сортов фактических параметров, т.е. дескрипторы содержат характеристики соответствующих формальных параметров.

InDescr есть параметр переменной IN.

InoutDescr есть параметр переменной IN/OUT.

Newqual Обозначает идентификатор, который должен быть использован в AS_1. Его имя в *Newqual* изменено, если процедура определена в сервисе.

40 *ProcessD* :: *ParameterD* Validinputset Outputset ProcessconnectionD*
41 *Validinputset* = *Signalqual*-set
42 *Outputset* = *Signalqual*-set
43 *ParameterD* = *Sortqual*
44 *ProcessconnectionD* = *Qual* ↠ *Qual*-set

ProcessD есть дескриптор процесса.

*ParameterD** Список дескрипторов, используемых, когда процесс создан, чтобы проверить соответствие сортов фактических параметров, т.е. дескрипторы содержат сорта соответствующих формальных параметров.

Signalqual-set Совокупности допустимых входных сигналов и допустимых выходных сигналов, соответственно (допустимые выходные сигналы берутся из графа процесса).

ProcessconnectionD Есть отношение между маршрутами сигналов и маршрутами сигналов сервиса процесса. Если маршруты сигналов сервиса не специфицируются, тогда таблица является пустой. Используется для проверки VIA в сервисах.

45 *SortD* :: *Equations*$_1$ [*Parentqual*] [*Expression*$_1$] *Newqual*
46 *Parentqual* = *Sortqual*

SortD есть дескриптор частичного определения типа. (новый тип - *newtype*)

Equations$_1$ Равенства AS$_1$ определены для сорта. Эти равенства используются, когда другой сорт наследует этот сорт.

Parentqual Родительский сорт сорта. Если *Parentqual* отсутствует nil, то сорт не имеет "родителя". *Parentqual* используется для проверки рекурсивности определений сорта.

Expression$_1$ Есть необязательное выражение AS$_1$, соответствующее необязательному начальному значению переменной, которая может быть специфицирована в частичном определении сорта.

Newqual Обозначает идентификатор, который должен быть использован в AS$_1$. Его имя в *Newqual* изменено, если частичный тип данных определен в сервисе.

47 *SyntypeD* :: *Parentqual Newqual* [*Expression*$_1$] *Range-condition*$_1$

SyntypeD есть дескриптор подтипа.

Parentqual Есть родительский сорт, т.е., если родитель, определенный в *AS*$_0$, есть подтип, он является родителем этого подтипа.

Newqual Обозначает идентификатор, который должен быть использован в AS$_1$. Его имя в *Newqual* изменено, если процедура определена в сервисе.

Expression$_1$ Есть необязательное выражение AS$_1$, соответствующее необязательному начальному значению переменной, которая может быть специфицирована в определении подтипа.

Range-condition$_1$ Есть диапазон условий AS$_1$, который используется для генерации значений, присваиваемых по умолчанию переменным подтипа.

48 *GeneratorD* :: *Genparm*$_0^+$ *Properties*$_0$

GeneratorD есть дескриптор генератора сорта данных.

Genparm$_0^+$ Список формальных параметров *AS*$_0$, которые берутся из заголовка генератора.

Properties$_0$ Тело *AS*$_0$ генератора.

49 *SynD* :: *Sortqual Expr*$_0$

SynD есть дескриптор синонима.

Sortqual Сорт синонима. Если в *AS*$_0$ сорт отсутствует, *Sortqual* извлекается из сорта выражения, содержащегося в определении синонима.

Expr$_0$ Выражение синонима *AS*$_0$.

50 *VarD* :: *Sortqal* [REVEALED] [EXPORTED] [*Expression*$_1$] *Newqual*

VarD Является дескриптором переменной.

Sortqual	Сорт переменной
REVEALED	Необязательный атрибут REVEALED, как в AS_0
EXPORTED	Необязательный атрибут EXPORTED, как в AS_0
[*Expression*$_1$]	Вариант AS_1 необязательных начальных выражений, специфицированных при определении переменной.
Newqual	*Qual*, который должен быть использован в AS_1. Его имя в *Newqual* отлично от первоначального имени, если переменная определена в сервисе.

51 *ImportD* :: *Sortqual*

ImportD есть дескриптор переменной импорта, содержащий сорт переменных импорта.

52 *ViewD* :: *Qual*

ViewD есть дескриптор переменной обозревания, содержащей *Qual* соответствующей переменной с атрибутом раскрытия.

53 *LiteralD* :: *Result*
54 *Result* = *Sortqual*

LiteralD есть дескриптор литерала, содержащего сорт литерала.

55 *OperatorD* :: *Sortqual*$^+$ *Result Newqual Explicit*
56 *Explicit* = *Bool*

OperatorD есть дескриптор оператора.

Sortqual$^+$	список идентификаторов сорта, соответствующих аргументам оператора
Result	идентификатор сорта, соответствующего результату
Newqual	Уникальный идентификатор оператора, используемый в AS_1
Explicit	Этот флаг принимает значение можно (false), если оператор неявно наследуется из другого сорта.

57 *ValueidD* :: *Mapvalue Sortqual*-set *Explicit*
58 *Mapvalue* = [*Qual*]

ValueidD есть дескриптор переменной аксиомы (идентификатор значения).

Mapvalue	Если идентификатор введен при помощи квантификации литерала, то дескриптор содержит текущее значение литерала. В квантифицированных аксиомах идентификатор значения замещается этим значением литерала.
Sortqual-set	Множество *Quals* сортов, которые в любой определенный момент времени доступны для идентификаторов значения. Если идентификатор значения введен при помощи явной квантификации, тогда множество содержит только один сорт.
Explicit	Флаг, указывающий был ли введен идентификатор значения при помощи явной квантификации.

59 *ErrorD* :: ()

ErrorD используется для обнаружения рекурсивных определений синонимов, списков сигналов и генераторов. Во время выполнения определений этих конструкций их дескрипторы заменяются на *ErrorD* так, чтобы любые применения в их собственных определениях могли быть обнаружены. Так же используется для отмены маскирования синонимов, которые не могут быть использованы в простых выражениях выбранных определений.

2.2 Описание *Quotdict*

Следующие записи содержат вспомогательную информацию, которая включена в домен *Dict* только с практической точки зрения (для того, чтобы сократить число параметров и результатов функций Мета IV).

1	*Quotdict*	=	SCOPEUNIT ⇸ *Qual* ∪ GLOBALNAMES ⇸ *Globalnames* ∪ LABELDICT ⇸ *Labeldict* ∪ STATEDICT ⇸ *Statedict* ∪ OUTSIGNALS ⇸ *Signalqual*-**set** ∪ DATATYPEDEF ⇸ *Data-type-definition*$_1$ ∪ SERVICES ⇸ (*Statetuplemap Servicetuple*) ∪ IMPLIED ⇸ *NameclosureD*-**set** ∪ IMPORTLIST ⇸ (*Name*$_0$ *Qual*[*Expr*$_0$])*
2	*Globalnames*	=	*Emptyqid Formuniquenm Formuniquenm*
3	*Labeldict*	=	*Label*$_0$ ⇸ *Transition*$_0$
4	*Statedict*	=	*Statename*$_0$ ⇸ (*StateD* | *ContenablestateD*)
5	*StateD*	::	*Speclist* [*Importstateinf*]
6	*Importstateinf*	=	[*Statename*$_0$] [*Graph-node*$_1$ | *Decision-node*$_1$]
7	*ContenablestateD*	::	*Transition*$_0$
8	*Statetuplemap*	=	*Statename*$_0$$^+$ ⇸ *Statename*$_0$
9	*Servicetuple*	=	*Servicequal**
10	*Speclist*	=	*Spec**
11	*Spec*	=	*Qual Statespec*$_0$
12	*Servicequal*	=	*Qual*
13	*Emptyqid*	=	*Qual*
14	*Formuniquenm*	=	*Name*$_0$

SCOPEUNIT — Содержит *Qual*, обозначающий текущий уровень, т.е. *Qual* обозначает идентификатор вложенной единицы подразделения.

GLOBALNAMES — Содержит дескриптор *Globalnames*, содержащий имена объектов, которые требуются для преобразования допустимого условия и непрерывного сигнала.

emptyqid — Сигнал, который посылается процессами к себе же в непрерывных сигналах и допустимых условиях. Объявляется в AS_1 на системном уровне.

formuniquenm — Имена двух переменных, используемых в непрерывных сигналах. Первое из них изменяется при каждом переходе в состояние непрерывного сигнала и его новое значение передается с сигналом *emptyqid*, формируя таким образом каждый раз новое уникальное значение. Второе имя сохраняет значение, принятое с сигналом *emptyqid*. Объявляются в AS_1 в каждом процессе.

LABELDICT — *Labeldict* сохраняет информацию о любой метке (имени коннектора), используемой в процессе, процедуре или теле сервиса. Он образуется перед осуществлением действительного преобразования тела и содержит *Transition*$_0$*s*, которые следуют за этой меткой (в коннекторах).

STATEDICT — *Statedict* содержит состояния тела. Преобразование состояний основывается на *Statedict*. Если дескриптор состояния есть *ContenableStateD*, то он обозначает состояние, которое содержит непрерывный сигнал или допустимое условие. Такие состояния не имеют явного представления в AS_1 (они представлены номером искусственного состояния). Вершина следующего состояния замещается посредством цепочки перехода, ведущей к искусственным состояниям.
Если дескриптор состояния есть *StateD*, то он обозначает состояние, которое имеет представление в AS_1. Он содержит:

Speclist — Список входных дескрипторов состояния. Каждый входной дескриптор (*Spec*) содержит:

Qual — Квалификатор (*Qualifier*), обозначающий контекст, в котором должен быть выражен следующий переход. Так как сервисы объединены в один *Statedict* до того как тела состояний преобразованы в AS_1, то *Qual* различается для различных *Specs*, если *Statedict* получен при декомпозиции сервиса. В других случаях *Qual* обозначает идентификатор вложенного процесса или процедуры.

Statespec$_0$ — Входная вершина AS_0.

Importstateinf — Если состояние является неявным состоянием, полученным из импортируемого выражения, то присутствует *Importstateinf*. Он включает имя предыдущего состояния (*Statename*$_0$) такое, что вершины следующих состояний с черточкой, следующие за импортируемым выражением, могут быть соответственно заменены. Если он отсутствует (nil), то импортируемое выражение образуется из начального перехода. Он также содержит вершину AS_1, где появляется выражение, содержащее импортируемое выражение. Если выражение содержит более одного импортируемого выражения, то только один из дескрипторов состояния импорта содержит такую вершину (другие отсутствуют nil).
В отличие от нормальных состояний дескрипторы *Statedict* для состояний импорта создаются, тогда как состояния преобразовываются.

OUTSIGNALS — Содержит множество сигналов. Используется для выведения выходных сигналов из тела процесса.

DATATYPEDEF — Содержит AS_1 *Data-type-definition*$_1$, ассоциированный с каждым блоком подразделения. Когда встречается частичное определение типа, *Data-type-definition*$_1$ модифицируется новым сортом, новыми операторами и равенствами.

SERVICES — Данная информация используется во время преобразования *Statedict*, образованного из декомпозиции сервиса. Все сервисы упорядочиваются в кортеже *Servicetuple*, который вместе с *Statetuplemap* используется для ассоциирования уникальных имен состояний AS_1 со старыми именами состояний, встречающихся в различных сервисах.

IMPLIED — Множество пар сорта переменной и неявного имени переменной, собранное во время преобразования определения процесса. Т.е. во время преобразования аксиом собираются неявно квантифицированные имена, во время преобразования действий собираются неявно импортированные имена переменных. Заметим, что *NameclosureD* не относятся концептуально к *NameclosureD*, которые встречаются в *Exportmap*, хотя они относятся к тому же домену.

IMPORTLIST — Во время преобразования выражения собирается информация об импортируемых выражениях, встречающихся в выражении. В результирующем выражении AS_1 выражение импорта заменяется новым неявным именем. После преобразования создается ряд состояний (одно на каждое выражение импорта) (и добавляются к

Statedict). Переход, следующий за новым состоянием, создается действием, в котором использовались выражения, неявное имя ($Name_0$), идентификатор переменной импорта (*Qual*); необязательное выражение Pid используются в выражении импорта ($Expr_0$). Длина кортежа, обозначенного IMPORTLIST, равна числу выражений импорта, встречающихся в конструкции.

2.3 Другие домены

1 $Decl_1$ = *Block-definition*$_1$ | *Channel-definition*$_1$ | *Signal-definition*$_1$ | *Signal-route-definition*$_1$ | *Procedure-definition*$_1$ | *Process-definition*$_1$ | *Signal-route-definition*$_1$ | *Syn-type-definition*$_1$ | *Data-type-definition*$_1$ | *Variable-definition*$_1$ | *View-definition*$_1$ | *Timer-definition*$_1$

2 *Context* = CONSTANT | AXIOMS | MAPPING | EXPRESSION

Эти домены (синонимы) вводятся для того, чтобы избежать громоздких повторений при спецификации типов функций:

$Decl_1$ — Сокращенная запись для обозначения определения AS_1, в которой много функций "преобразования определения" создается в качестве результата.

Context — Сокращенная запись для обозначения возможных контекстов, в которую преобразовываются выражения. Контекст может быть константой (например, при преобразовании ответов), аксиомой, секцией отображения аксиом или чем-либо другим.

3 *Processqual* = *Qual*

4 *Operatorqual* = *Qual*

Эти имена доменов используются в некоторых случаях для указания того, что в данном контексте *Qual* (идентификатор) относится к специальному типу (т.е. процесс или оператор).

5 *External-Information* = ...

External-Information содержит дополнительную информацию, необходимую для того, чтобы ввести семантику в SDL. Эта информация вносится извне, и она, следовательно, придается в качестве параметра к *definition-of-SDL* (вместе с определением системы и предопределенными сортами). Так как SDL не определяет того, каким образом информация структурирована (называемые фактические родовые параметры), то функции Meta-IV, которые использует *External-Information*, определены неформально.

External-Information содержит следующее.

- Информацию о фактических родовых параметрах (соответствующие формальные параметры являются внешними синонимами и неформальными текстами в необязательной вершине).
- Информация о фактическом подмножестве параметров, чтобы указать какую последовательность подмножеств следует выбрать, т.е. выбирается множество *Block-identifier*$_1$*s* из *External-Information* (см. *definition-of-SDL* и *select-consistent-subset*).
- Информация о недетерминированной задержке в каналах (т.е. "случайной" функции).
- Информация о времени начала и единица времени (используемая при моделировании реального времени).

6 *Auxiliary-Information* = *Time-information Term-Information Is-expiredF DelayF*

7 *Time-information* = (*Ground-term*$_1$ → *Ground-term*$_1$) *Ground-term*$_1$

8	*Term-Information*	= *Sort-identifier*$_1$ *Literal-operator-identifier*$_1$ *Literal-operator-identifier*$_1$ *Literal-operator-identifier*$_1$
9	*Is-expiredF*	= *Ground-term*$_1$ *Ground-term*$_1$ → *Bool*
10	*DelayF*	= () ⇒ Bool

Auxiliary-Information содержит некоторую информацию, которая, независимо от *System-definition*$_1$ и последовательного подмножества, требуется для интерпретации. *Auxiliary-Information* образуется в Статических Семантиках и придается в качестве параметра процессору System, когда он начинает функционировать (см. функцию *definition-of-SDL*). *Auxiliary-Information* состоит из

Time-information — Функция в *Time-information* используется в Динамических Семантиках для обновления текущего времени. Данный литерал AS_1 сорта TIME возвращает другой литерал сорта времени. Кроме того, *Time-information* содержит *Ground-term*$_1$, обозначающий начальное время, которое определяется вне системы SDL (т.е. оно берется из *External-Information*).

Term-Information — AS_1 не содержит никакой информации о проверке идентификаторов. Однако, четыре идентификатора AS_1 должны быть известны (т.е. различаться) в Динамических Семантиках. Такими идентификаторами являются PID, NULL, TRUE и FALSE. *Term-information* обозначает эти идентификаторы и они создаются в Статических Семантиках.

Is-expiredF — Данная функция создается в Статических Семантиках и используется Динамическими Семантиками для проверки просрочен ли данный таймер. Заданы два литерала AS_1 сорта времени, **true** возвращается, если значение первого параметра больше или равно второму параметру.

DelayF — Данная функция используется для моделирования недетерминированной задержки в каналах (см. процессор пути *path* в Приложении F.3). Эта функция получается из *External-Information* и она имеет императивный характер, предполагающий, что она может зависеть от некоторых внешних физических параметров.

3 Преобразование AS_0 в AS_1

3.1 Основные функции

Этот раздел содержит две функции:

definition-of-SDL Которая является самой удаленной функцией, привлеченной извне. Когда задана SDL система (в форме AS_0) как параметры, она определяет семантики (статические или динамические) системы. Она также формирует связь между статическими и динамическими семантиками.

transform-system Которая является входной функцией для статических семантик. Когда задана SDL система (в форме AS_0) как параметр, она возвращает спецификацию на абстрактную синтаксическую форму SDL (в форме AS_1), если форма AS_0 статически корректна.

definition-of-SDL (*extparms, systemdef, predefsorts*) ≜ (3.1.1)

```
1  (let (as1, auxinf) = transform-system(systemdef, predefsorts, extparms) in
2   if as1 = nil then
3       undefined
4     else
5       (let subsetcut = select-consistent-subset (as1, extparms) in
6        start system(as1, subsetcut, auxinf)))
```

запись: *External-Information* Sys_0 $Datadef_0^+$ ⇒

Цель Определить свойства SDL

Параметры

extparms Некоторая внешняя информация (*External-Information*) (см. раздел 2.3).

systemdef Представление в виде AS_0-дерева системы SDL

predefsorts Предопределенные данные в форме AS_0.

Алгоритм

Строка 1 Преобразовать систему в абстрактную синтаксическую форму (в форме AS_1).

Строка 2 Если найдены статические ошибки (т.е. если никакое представление в форме AS_1 не может быть выведено), то поведение не определено

Строка 4 Если статические ошибки не найдены, то

Строка 5 Выбрать множество *Block-identifier*$_1$*s*, обозначающее согласованное подмножество

Строка 6 Создать экземпляр системы, т.е. создать процесс Meta-IV, который ведет себя как нижележащая система.

transform-system (genericsystem, predefdatasorts, extparms) ≜ (3.1.2)

```
1  (let mk-Sys0(sysdef,refdeflist) = apply-generic-parameters(genericsystem, extparms) in
2   let mk-Sysdef0(snm, decllist, tnm) = sysdef in
3   let (as0 global, globalentities) = as0-global-entities() in
4   let as1 datadef = mk-Data-type-definition1 (name-to-name1(create-unique-name()),{},{},{},{}) in
5   (trap exit with (nil,nil) in
6    (let d = [⟨(SYSTEM, snm)⟩ ↦ mk-SystemD(),
7              SCOPEUNIT       ↦ ⟨(SYSTEM,snm,)⟩
8              GLOBALNAMES     ↦ globalentities,
9              DATATYPEDEF     ↦ as1 datadef] in
10   let decllist' = replace-references(decllist, refdeflist)(⟨(SYSTEM, snm)⟩) in
11   let predefdict be s.t. (,predefdict) = transform-decllist(predefdatasorts)(predefdict + d) in
12   let decllist" = remove-select(decllist', {})(predefdict) in
13   if (∃fulldict ∈ Dict)
14     ((trap exit with false in
15       (let (dict) = transform-decllist(decllist'' ⌢ ⟨as0global⟩ ⌢ predefdatasorts)(fulldict + d) in
16        fulldict = dict))) then
17     (let (as1 dcl, dict) be s.t. (as1 dcl, dict) =
18          transform-decllist(decllist" ⌢ ⟨as0global⟩ ⌢ predefdatasorts)(dict + d) in
19      let as1 = make-as1 tree(SYSTEM, snm,as1 dcl, nil, nil, nil)(dict) in
20      let auxinf = generate -auxiliary-information(extparms)(dict) in
21      (tnm ∉ {snm, nil}
22          → exit("§2.2.2; Завершающее имя в определении системы отлично от определяющего имени"),
23      ¬ (∃b ∈ elems decllist')(is-Blockdef0(b))
24          → exit("§2.4.2.: Определение системы должно содержать по крайней мере один блок"),
25      T  → (as1, auxinf)))
26    else
27     (nil,nil))))
```

запись: *Sys0 Datadef0⁺ External-Information* → [*Sysytem-definition1*] [*Auxiliary-Information*]

Цель Преобразовать систему SDL в форме AS_0 в соответствующую форму AS_1

Параметры

genericsystem	Общее определение системы
predefdatasort	Список предопределенных данных также в форме AS_0
extparameters	Внешняя информация (см. раздел 2.3)

Результат Определение системы в форме AS_1 и некоторая информация, которая будет использована в Приложении F.3. Если определение системы не является корректным, nil возвращается для обоих результатов.

Алгоритм

Строка 1 Преобразовать общее определение системы в определение конкретной системы, преобразуя выражение для каждого внешнего синонима, определенного в системе, и трансформируя каждый неформальный текст в ответы выбираемых определений в AS_0 *Conditionlist0*. SDL не определяет, как сделать эти преобразования.

Строка 2 Пусть *snm* обозначает имя системы, *decllist* обозначает список определений и *tnm* обозначает завершающее имя

Строка 3 Создать определение AS_0 и дескриптор *Globalnames* (см. *Quotdict*), соответствующие глобальному сигналу emptyq.

Строка 4 Создать начальное *Data-type-definition1* для блоков подразделения системы. После просмотра списка определений (строка 17) оно содержит информацию о всех частичных типах данных, определенных на системном уровне.

Строка 5 Если *transform-decllist* или другие функции отловлены, спецификация не является корректной.

Строки 6-9 Создать начальный *Dict*, состоящий из системного дескриптора, входа SCOPEUNIT, который указывает уровень системы, глобальных имен и текущего *Data-type-definition*$_1$.

Строка 10 Вставить все удаленные определения (*refdeflist*) в список определений системы. Новый список определений (*decllist'*) не содержит ссылок

Строка 11 Преобразовать предопределенные данные для того, чтобы создать *Dict*, состоящий только из сортов предопределенных данных. Результат используется только во время перемещения утверждений SELECT (строка 12), т.е. предопределенные сорта всегда трактуются как обычные определения, как это показано в строке 17. Это читается так: Пусть *predefdict*) будет таким, что один из результатов (другой результат это определения AS$_1$, которые здесь не используются) преобразования предопределенных сортов в подразделении *predefdict* (т.е. с семантической информацией всех предопределенных сортов) есть сам *predefdict*. Такой *Dict* действительно всегда существует, так как предопределенные сорта полагаются корректными.

Строка 12 Удалить все утверждения SELECT из системы. Так как выбираемые вершины не содержат определений, они могут быть удалены при присвоении.

Строки 13-16 Если существует *Dict* (*fulldict*) такой, что преобразование *decllist"* в подразделение *fulldict* может быть сделано без ошибок (т.е. без применения *exit* в *transform-decllist*) и такой, что результатом является сам *fulldict*, тогда *decllist"* является корректным.

Строка 17 Если это так, тогда преобразовать *decllist"* в подразделении такого *Dict*. Другой результат (*as*$_1$*dcl*) является множеством определений AS$_1$. Этот путь использования *Dict* является существенным в моделировании и следует заметить, что:

- Он используется для того, чтобы обойти тот факт, что ссылка на имена в SDL может осуществляться раньше их текстовых определений.
- Функции Meta-IV, которые преобразовывают определения, имеют полную доступность к *Dict* (всем дескрипторам). Например, *transform-blockdef* возвращает *Block-definition*$_1$ и дескриптор *Dict* для блока, хотя дескриптор блока уже содержится в *Dict*, который задается как параметр функции. *Dict*, который задается как параметр, используется как свойства доступа объектов, используемые блоками, но сам дескриптор блока в параметре *Dict* не может быть использован (другими словами нельзя гарантировать, что решение *Dict* существует для корректной спецификации, т.е. спецификация Meta-IV может быть неверной).
- Во время преобразования определений возвращенные дескрипторы являются составными и в конце концов (когда все определения преобразованы), результат *transform-decllist* является таким же, как параметр.
- *dict* в строке 17 является только *Descriptordict*. *Quotdict* ничего не дает для результирующего *Dict*.
- Строки 11 и 17 в *transform-system* являются единственными во всем формальном описании, где конструкция **be such that** (быть таким, что) использована в таком виде.

Строка 19 Создать *System-definition*$_1$

Строка 20 Сгенерировать *Auxiliary-Information* для использования при интерпретации

Строка 21 Если завершающее имя определено, оно должно совпадать с системным именем.

Строка 23 Должно быть по крайней мере одно определение блока в системе.

Строка 25 Возвратить представление системы в AS$_1$ и вспомогательную информацию

as₀-global-entities() ≜ (3.1.3)

```
1  (let emptyqnm = create-unique-name(),
2      formunique1 = create-unique-name(),
3      formunique2 = create-unique-name() in
4  let emptyqid = mk-Id0(◊, emptyqnm) in
5  let globalnames = (emptyqid, formunique1, formunique2) in
6  let intgid = mk-Id0(◊, mk-Name0("INTEGER", nil) in
7  let as0def = mk-Sigdef0(⟨mk-Sigelem0(emptyqnm, ⟨ intgid ⟩, nil)⟩) in
8  (as0def, globalnames))
```

запись: () → *Sigdef₀ Globalnames*

Цель — Создать глобальное определение в AS_0, чтобы поместить на системный уровень, и создать смыкание *Globalnames* (см. определение *Quotdict*) для общего использования при преобразовании. Определение в AS_0 определяет сигнал *emptyq*, используемый в разрешающем условии, и непрерывный сигнал

Алгоритм

Строка 1	Создать уникальное имя для сигнала *emptyq*.
Строки 2-3	Создать уникальные имена для двух переменных, используемых в соединении с непрерывным сигналом. Эти переменные определяются в каждом процессе.
Строка 4	Создать идентификатор в AS_0 для сигнала *emptyq*.
Строка 5	Создать смыкание *Globalnames*.
Строка 6	Создать идентификатор в AS_0 для сорта *integer*.
Строка 7	Создать определение в AS_0 для непрерывного сигнала.
Строка 8	Вернуть определение в форме AS_0 и смыкание *Globalnames*.

3.2 Замена ссылок на определения

В этом разделе ссылки заменяются удаленными определениями. Входная функция *replace-references* берет в качестве аргумента определения с системного уровня и список удаленных определений. Результатом является список определений без ссылок.

replace-references (*deflist, remotelist*) (*scopeunit*) ≜ (3.2.1)

```
1  if (∀i1,i2 ∈ ind remotelist)
2     (i1 ≠ i2 ∧
3       cases (remotelist[i1], remotelist[i2]):
4          ((mk-Blockdef0(mk-Id0(q1,nm1),,,), mk-Blockdef0(mk-Id0(q2,nm2),,,)),
5           (mk-Prdef0(mk-Id0(q1,nm1),,,,,,), mk-Prdef0(mk-Id0(q2,nm2),,,,,,)),
6           (mk-Procdef0(mk-Id0(q1,nm1),,,,), mk-Procdef0(mk-Id0(q2,nm2),,,,)),
7           (mk-Servicedef0(mk-Id0(q1,nm1),,,,), mk-Servicedef0(mk-Id0(q2,nm2),,,,)),
8           (mk-Chansubdef0(mk-Id0(q1,nm1),,), mk-Chansubdef0(mk-Id0(q2,nm2),,))
9             → nm1 = nm2 ⊃ (q1 ≠ q2 ∧ q1 ≠ ◊ ∧ q2 ≠ ◊),
10         T  → true )) then
11   (let (deflist', remoteset) = remove-references (deflist, elems remotelist, false) (scopeunit) in
12    if remoteset = {} then
13      deflist'
14     else
15      exit("§2.4.1: В системном определении не ссылаются на удаленные определения"))
16 else
17  exit("§ 2.4.1: Удаленные определения не являются уникальными"))
```

запись: $Decl_0^+$ $Decl_0^*$ → *Qual* → $Decl_0^+$

Цель — Заменить все ссылки в системе на удаленные определения

Параметры

deflist	Список определений для системного уровня.
remotelist	Список удаленных определений.

Результат Список определений системного уровня без ссылок.

Алгоритм

Строки 1-10	Для любых двух различных удаленных определений блоков, определений процессов, определений процедур, определений сервиса или определений подструктур должно выполняться условие, что если они имеют одинаковое имя в идентификаторе, то их квалификаторы должны быть непустыми и отличными.
Строки 11-15	Заменить ссылки их определениями. *deflist'* является окончательным списком. При замене всех ссылок не должно остаться ни одного удаленного определения.

remove-references(dlist, remoteset, chansub)(unit) ≜ (3.2.2)

```
1   if dlist = ⟨⟩ then
2     (⟨⟩, remoteset)
3    else
4     (let match-id (nm, id) = s-Name0(id) = nm ∧ s-Qualifier0(id) ∈ {⟨⟩,unit} in
5       cases hd dlist :
6        (mk-Blockref0 (name)
7          → if (∃blkdef ∈ remoteset)(is-Blockdef0(blkdef) ∧ match-id(name, s-Blockid0(blkdef))) then
8              (let blkdef ∈ remoteset be s.t. is-Blockdef0(blkdef) ∧ match-id(name, s-Blockid0(blkdef)) in
9               let mk-Blockdef0 (bid, decll, sub, tid) = blkdef in
10              if tid ∉ {nil, bid} then
11                exit("§2.4.1 Открывающий и заканчивающий идентификаторы в определении различны")
12               else
13                (let blkdef' =
14                     mk-Blockdef0(mk-Id0 (⟨⟩, name), decll, sub, mk-Id0(⟨⟩, name)) in
15                remove-references(⟨ blkdef' ⟩ ⌢ tl dlist, remoteset \ {blkdef}, chansub)(unit)))
16               else
17                exit("§2.4.1 Нет удаленных определений, соответствующих ссылке")
18       mk-Prref0(name, inst)
19         → if (∃prdef ∈ remoteset)(is-Prdef0(prdef) ∧ match-id(name, s-Prid0(prdef))) then
20             (let prdef ∈ remoteset be s.t. is-Prdef0(prdef) ∧ match-id(name, s-Prid0(prdef)) in
21              let mk-Prdef0(pid, inst', b, d, e, f, tid) = prdef in
22              let inst'' = select-remote-number-of-instances(inst, inst') in
23              if tid ∉ {nil, pid} then
24                exit("§2.4.1 Открывающий и заканчивающий идентификаторы в определении различны")
25               else
26                (let prdef' =
27                     mk-Prdef0(mk-Id0(⟨⟩, name), inst'', b, d, e, f, mk-Id0(⟨⟩, name)) in
28                remove-references(⟨ prdef' ⟩ ⌢ tl dlist, remoteset \ {prdef}, chansub)(unit)))
29               else
30                exit("§2.4.1 Нет удаленных определений, соответствующих ссылке")
31       mk-Procref0(name)
32         → if (∃procdef ∈ remoteset)(is-Procdef0(procdef) ∧ match-id(name, s-Procid0(procdef))) then
33             (let procdef ∈ remoteset be s.t. is-Procdef0(procdef) ∧ match-id(name, s-Procid0(procdef)) in
34              let mk-Procdef0(pid, parm, decll, body, tid) = procdef in
35              if tid ∉ {nil, pid} then
36                exit("§2.4.1 Открывающий и заканчивающий идентификаторы в определении различны")
37               else
38                (let procdef' =
39       mk-Procdef0(mk-Id0(⟨⟩, name), parm, decll, body, mk-Id0(⟨⟩, name)) in
40              remove-references(⟨ procdef' ⟩ ⌢ tl dlist, remoteset \ {procdef}, chansub)(unit)))
41               else
42                exit("§2.4.1 Нет удаленных определений, соответствующих ссылке"),
43       mk-Servicerefo(name)
44         → if (∃servicedef ∈ remoteset)(is-Servicedef0(serevicedef) ∧
45                                   match-id(name, s-ServiceId0(servicedef))) then
46             (let servicedef ∈ remoteset be s.t. is-Servicedef0(servicedef) ∧
47                 match-id(name, s-ServiceId0(servicedef)) in
48              let mk-Servicedef0(sid, a, b, d, tid) = servicedef in
49              if tid ∉ {nil, sid} then
50                exit("§2.4.1 Открывающий и заканчивающий идентификаторы в определении различны")
51               else
52                (let servicedef' =
53                     mk-Serelcedef0(mk-Id0(⟨⟩, name), a, b, d, mk-Id0(⟨⟩, name)) in
54                remote-references(⟨ servicedef' ⟩ ⌢ tl dlist, remoteset \{servicedef}, chansub)(unit)))
55               else
56                exit("§2.4.1 Нет удаленных определений, соответствующих ссылке"),
```

```
57      mk- Chansubref0(name)
58        → if (∃chansubdef ∈ remoteset)(is-Chansubdef0(chansubdef) ∧
59                                       s-Chansubid0(chansubdef) ≠ nil ∧
60                                       match-id(name, s-Chansubid0(chansubdef))) then
61          (let chansubdef ∈ remoteset be s.t. is-Chansubdef0(chansubdef) ∧
62             match-id(name, s-Chansubid0(chansubdef)) in
63           let mk-Chansubdef0(cid, decll, tid) = chansubdef in
64           if tid ∉ {nil, cid} then
65              exit("§2.4.1 Открывающий и заканчивающий идентификаторы различны")
66             else
67              (let chansubdef' =
68                  mk-Chansubdef0(mk-Id0(⟨⟩, name), decll, mk-Id0(⟨⟩, name)) in
69              remove-references(⟨chansubdef'⟩ ⌢ tl dlist, remoteset \{chansubdef}, true)(unit)))
70        else
71          exit("§2.4.1 Нет удаленных определений, соответствующих ссылке"),
72      mk-Blocksubref0(name)
73        → if (∃chansubdef ∈ remoteset)(is-Chansubdef0(chansubdef) ∧
74                                       s-Chansubid0(chansubdef) ≠ nil ∧
75                                       match-id(name, s-Chansubid0(chansubdef))) then
76          (let chansubdef ∈ remoteset be s.t. is-Chansubdef0(chansubdef) ∧
77              match-id(name, s-Chansubid0(chansubdef)) in
78           let mk-Chansubdef0(bid, decll, tid) = chansubdef in
79           if tid ∉ {nil, bid} then
80              exit("§2.4.1 Открывающий и заканчивающий идентификаторы различны")
81             else
82              (let blksubdef' =
83                  mk-Blocksubdef0(mk-Id0(⟨⟩, name), decll, mk-Id0(⟨⟩, name)) in
84              remove-references(⟨blksubdef'⟩ ⌢ tl dlist, remoteset \{chansubdef}, chansub)(unit)))
85        else
86          exit("§2.4.1 Нет удаленных определений, соответствующих ссылке"),
87      mk-Decomposition0(list)
88        → (let (list', rrest) = remove-references (list, remoteset, false)(unit) in
89          (mk-Decomposition0(list'), rrest)),
90      mk-Blockdef0(mk-Id0(q, nm), decll, sub, tid)
91        → (let (decll', rrest) = remove-references(decll, remoteset, false)(unit ⌢ ⟨(BLOCK, nm)⟩) in
92          let (sub', rrest') =
93            if sub = nil
94              then (nil, rrest)
95              else remove-references(⟨ sub ⟩, rrest, false)(unit ⌢ ⟨(BLOCK, nm)⟩) in
96          let (drest', rrest") = remove-references(tl dlist, rrest', chansub)(unit) in
97          (⟨ mk-Blockdef0(mk-Id0(q, nm), decll', sub', tid)⟩ ⌢ drest', rrest")),
98      mk-Blocksubdef0(id, decll, tid)
99        → (let (, scnm) = unit [len unit] in
100         let mk-Id0(q, nm) = if id = nil then mk-Id0(⟨⟩, scnm) else id in
101         let unit' = unit ⌢ ⟨(SUBSTRUCTURE, nm)⟩ in
102         let (decll', rrest) = remove-references(decll, remoteset, false)(unit') in
103         (mk-Blocksubdef0(mk-Id0(q, nm), decll', tid), rrest)),
104     mk-Chandef0(nm, a, b, sub, tnm)
105       → (let (sub', rrest) =
106           if sub = nil
107             then (nil, remoteset)
108             else remove-references(⟨ sub ⟩, remoteset, chansub)(unit) in
109         let (drest', rrest') = remove-references(tl dlist, rrest, chansub)(unit) in
110         (⟨ mk-Chandef0(nm, a, b, sub', tnm)⟩ ⌢ drest', rrest')),
```

```
111     mk-Chansubdef0(id, decll, tid)
112       → (let (, scnm) = unit[len unit] in
113           let mk-Id0(q,nm) = if id = nil then mk-Id0(◊, scnm) else id in
114           let unit' = unit ⇝ ⟨(SUBSTRUCTURE, nm)⟩ in
115           let (decll', rrest) = remove-references(decll, remoteset, true)(unit') in
116           (mk-Chansubdef0(mk-Id0(q,nm), decll', tid), rrest)),
117     mk-Prdef0(mk-Id0(q,nm), a, b, d, decll, body, tid)
118       → (let (decll', rrest) =
119             remove-references(decll, remoteset, false)(unit ⇝ ⟨(PROCESS, nm)⟩) in
120           let (body', rrest') =
121             if is-Body0(body) then
122               (body, rrest)
123             else
124               remove-references(⟨body⟩, rrest, false)(unit ⇝ ⟨(PROCESS, nm)⟩) in
125           let (drest', rrest") = remove-references(tl dlist, rrest', chansub)(unit) in
126           (⟨mk-Prdef0(mk-Id0(q,nm), a, b, d, decll', body', tid)⟩ ⇝ drest', rrest")),
127     mk-Procdef0(mk-Id0(q,nm), p, decll, body, tid)
128       → (let (decll', rrest) =
129             remove-references(decll, remoteset, false)(unit ⇝ ⟨(PROCEDURE, nm)⟩) in
130           let (drest', rrest') = remove-references(tl dlist, rrest, chansub)(unit) in
131           (⟨mk-Procdef0(mk-Id0(q,nm), p, decll', body, tid)⟩ ⇝ drest', rrest')),
132     mk-Servicedef0(mk-Id0(q,nm), a, decll, body, tid)
133       → (let (decll', rrest) =
134             remove-references(decll, remoteset, false)(unit ⇝ ⟨(SERVICE, nm)⟩) in
135           let (drest', rrest') = remove-references(tl dlist, rrest, chansub)(unit) in
136           (⟨mk-Servicedef0(mk-Id0(q,nm), a, decll', body, tid)⟩ ⇝ drest', rrest')),
137     mk-Select0(expr, decll)
138       → if is-wf-entities(decll, chansub)(unit) then
139           (let (decll', rrest) = remove-references (decll, remoteset, chansub)(unit) in
140            let (drest', rrest') = remove-references(tl dlist, rrest, chansub)(unit) in
141            (⟨mk-Select0(expr, decll')⟩ ⇝ drest', rrest'))
142           else
143            exit("§4.3.3 Определения в отборе не допустимы в данной единице подразделения"),
144     T → (let (drest, rrest = remove-references(tl dlist, remoteset, chansub)(unit) in
145           (⟨hd dlist⟩ ⇝ drest,rrest))))
```

запись: (*Decl*$_0$* | *Blocksubdef*$_0$ | *Chansubdef*$_0$ | *Decomposition*$_0$) *Refdecl*$_0$* *Bool* → *Qual* →
(*Decl*$_0$* | *Blocksubdef*$_0$ | *Chansubdef*$_0$ | *Decomposition*$_0$) *Refdecl*$_0$*

Цель Заменить каждую ссылку в списке определений удаленным определением

Параметры

dlist	Список определений в AS$_0$.
remoteset	Множество удаленных определений.
chansub	Имеет значение **true** (истинно) только, если *dlist* обозначает список определений канальной подструктуры. Этот параметр используется для проверки корректности *Connect*$_0$s (см. *is-wf-entities*).
unit	Контекст (представляемый квалификатором), в котором присутствует список определений.

Результат Список определений в AS$_0$, не содержащий ссылок, и множество удаленных определений, на которые нет ссылок. удалены

Алгоритм

Строка 1 Если множество определений (*dlist*) пусто, не возвращать определение и модифицированное множество удаленных определений. (*remove-refereces* рекурсивно возвращает полный список определений)

Строка 4	Определить функцию использования, которая возвращает значение true (истинно), если *nm* ($Name_0$) в ссылке соответствует *id* (из определения), т.е., если имя в *id* равно $Name_0$ в ссылке и квалификатор либо пуст, либо равен блоку подразделения, в котором размещена ссылка.
Строка 5	Рассмотреть первое определение в списке определений.
Строка 6	Если определение является ссылкой на блок, то должно существовать удаленное определение блока, которое имеет такое же имя в его идентификаторе, что и ссылка на блок (проверяемая при помощи *match-Id*, см. строку 4), и для которого квалификатор в его идентификаторе либо пуст, либо равен блоку подразделения, в котором размещена ссылка.
Строка 8	Пусть *blkdef* обозначает определение этого блока.
Строки 10-11	Завершающий идентификатор в удаленном определении блока должен быть либо опущен, либо равен начальному идентификатору.
Строка 13	Перестроить определение блока с пустыми квалификаторами в начальном и конечном идентификаторах
Строка 15	Добавить его к списку определений так, чтобы вложенные определения в блоке были связаны в следующем рекурсивном уровне, и удалить определение блока из множества удаленных определений.
Строки 18-30	Та же схема, как для определений блока, на которые есть ссылки, как показано выше за исключением строки 22, где $Instances_0$ для ссылки сравнивается с $Instances_0$ для удаленного определения
Строки 31-42	Та же схема, как для определений блока, на которые есть ссылки, как показано выше.
Строки 43-56	Та же схема, как для определений блока, на которые есть ссылки, как показано выше.
Строки 57-71	Так же схема, как для определений блока, на которые есть ссылки, как показано выше. В строке 59 проверяется присутствует ли идентификатор канальной подструктуры, если определение канальной подструктуры является удаленным.
Строки 72-86	Та же схема, как для определений блока, на которые есть ссылки, как показано выше за исключением того, что определения подструктуры блока синтаксически появляются, как более общие определения канальной подструктуры так, что подструктура блока конвертируется.
Строки 87-89	Если определение является декомпозицией, то заменить ссылки в списке определений, содержащиеся в декомпозиции.
Строка 90	Если определение является определением блока, то
Строка 91	Удалить ссылки в списке определений, содержащемся в блоке.
Строка 92	Удалить ссылки в подструктуре блока.
Строка 96	Удалить ссылки в оставшейся части списка определений, содержащей определение блока, и
Строка 97	Вернуть модифицированное определение блока, связанное с другими модифицированными определениями, и вернуть модифицированный список удаленных определений.
Строки 98-103	Удалить ссылки в определении подструктуры блока.
Строки 104-110	Удалить ссылки в определении канала.
Строки 111-116	Удалить ссылки в определении канальной подструктуры.
Строки 117-126	Удалить ссылки в определении процесса. Если процесс содержит декомпозицию (проверяемую в строке 20), то так же должны быть заменены ссылки в декомпозиции.
Строки 127-131	Удалить ссылки в определении процедуры.
Строки 132-136	Удалить ссылки в определении сервиса.
Строки 137-141	Удалить ссылки, содержащиеся в определении отбора, и проверить, чтобы каждое определение в выборе являлось разрешенным во вложенной единице подразделения.
Строка 144	В остальных случаях определение возвращается без изменений.

select-remote-number-of-instances(*inst*, *reminst*) ≜ (3.2.3)

```
1  (inst = nil ∧ reminst ≠ nil
2    → reminst,
3   reminst = inst ∨ reminst = nil
4    → inst,
5  T → exit("§ 2.4.4 Удаленное число экземпляров спецификации не соответствует ссылке процесса"))
```

запись: [*Instances*$_0$] [*Instances*$_0$] → [*Instances*$_0$]

Цель Выбрать *Instances*$_0$ для процесса в случае, когда имеется ссылка на процесс.

Параметры

inst *Instances*$_0$ специфицировано для ссылки процесса.
reminst *Instances*$_0$ специфицировано для удаленного определения.

Алгоритм

Строка 1 Если *Instances*$_0$ специфицировано для удаленного определения, только тогда вернуть эту спецификацию *Instances*$_0$
Строка 3 Если *Instances*$_0$ для ссылки процесса равно *Instances*$_0$ для удаленного определения или *Instances*$_0$ специфицировано для ссылки, только тогда вернуть эту спецификацию *Instances*$_0$.
Строка 5 В противном случае спецификация *Instances*$_0$ для процесса является противоречивой.

is-wf-entities(*decllist*, *ischannelsub*) (*level*) ≜ (3.2.4)

```
1  if decllist = ⟨⟩ then
2    true
3   else
4    (let (q, ) = level[len level] in
5     is-wf-entities(tl decllist, ischannelsub)(level) ∧
6      cases hd decllist :
7       (mk-Blockdef0(,,,),
8        mk-Chandef0(,,,,)
9         → q ∈ {SYSTEM, SUBSTRUCTURE},
10       mk-Connect0(id,)
11        → q ∈ {SUBSTRUCTURE,BLOCK} ∧ (id = ENV ⊃ (q = SUBSTRUCTURE ∧ ischannelsub)),
12       mk-Sigroutedef0(,,),
13       mk-Prdef0(,,,,,,)
14        → q = BLOCK,
15       mk-Signallistdef0(,),
16       mk-Sigdef0()
17        → q ∉ {SERVICE,PROCEDURE},
18       mk-Procdef0(,,,,),
19       mk-Vardef0(,,)
20        → q ∈ {SERVICE, PROCESS, PROCEDURE},
21       mk-Importdef0(),
22       mk-Viewdef0(),
23       mk-Timerdef0()
24        → q ∈ {SERVICE, PROCESS},
25      T → true))
```

запись: *Decl*$_0$* *Bool* → *Qual* → *Bool*

Цель Проверить, что список определений, полученный из определения отбора, является синтаксически допустимым в данном контексте. Тем не менее декомпозиция здесь не рассматривается.

Параметры

decllist	Проверяемый список определений.
ischannelsub	Флаг, который имеет значение **true** (истинно), если *decllist* принадлежит канальной подструктуре.

Результат Истинно, если допустимо

Алгоритм

Строка 1	После просмотра вернуть значение **true**.
Строка 4	Извлечь тип единицы подразделения (*q*) из *Qual*, обозначающего текущую единицу подразделения.
Строки 5-25	Список определений является корректным, если первое определение является корректным (строки 7-25), и оставшаяся часть списка определений является корректной (строка 5).
Строки 7-8	Если первое определение является определением блока или определением канала, то единица подразделения контекста должна быть системой или подструктурой блока, или подструктурой канала.
Строка 10	Если это соединение, то единица подразделения должна быть либо подструктурой блока, либо подструктурой канала, либо блоком.
Строки 12-13	Если это определение маршрута сигнала в определении процесса, то единица подразделения должна быть блоком.
Строки 15-16	Если это определение списка сигналов или определение сигнала, то единица подразделения не должна быть сервисом или процедурой.
Строки 18-19	Если это определение процедуры или определение переменной, то единица подразделения должна быть либо сервисом, либо процессом, либо процедурой.
Строки 21-23	Если это определение импорта, или определение обозревания, или определение таймера, то единица подразделения должна быть сервисом либо процессом.

3.3 Удаление определений отбора

В этом разделе рассматриваются определения отбора. Входная функция является *remove-select*, используемый в *transform-system*. Для каждого списка определений выполняются следующие шаги:

1. Собрать имена сортов, которые определены в списке определений, включая сорта, определенные в содержащемся определении отбора. Добавить эти сорта к сортам, собранным в окружающей единице подразделения. Это необходимо для того, чтобы предотвратить беспорядок, если предопределенные сорта будут переопределены.

2. Собрать все синонимы, определенные в списке определений или в списках определений для содержащих их определений отбора. Поместить дескриптор *ErrorD* в *Dict* для этих синонимов, указывая, что они не могут быть пока использованы. Идентифицируются те синонимы, которые определены при помощи любых собранных (видимых) сортов или которые имеют более одного определения (они являются также многократно определенными, если они определены в различных определениях отбора). Они совсем не могут использоваться в определениях отбора, так как их использование может приводить к разным результатам в зависимости от порядка выбора. Заметим, что эти синонимы не могут использоваться только во время удаления определений отбора. После чего они, очевидно, не являются многократно определенными.

3. Удалить определения отбора. Каждый раз, когда определение отбора замещается его содержащимся списком определений, дескрипторы для синонимов в списке определений изменяются с *ErrorD* на *SynD* за исключением тех, которые не разрешены для использования.

4. Когда все определения отбора удалены, дескрипторы *SynD* выполнены для всех синонимов в окончательном списке определений (также для синонимов, которые ранее не могли быть использованы) и собраны сорта, определенные в окончательном списке определений.

5. Определения отбора во вложенном блоке подразделения удаляются, используя обновленный dict и используя собранные сорта.

Так как возможности переходов не должны содержать определений, они могут быть оценены во время преобразования тела состояния в AS_1.

remove-select(*decllist, sursorts*) (*dict*) ≜ (3.3.1)

1 (**let** *sorts* = *sursorts* ∪ *collect-sorts*(*decllist*) **in**
2 **let** (*allsyn, illsyn*) = *collect-illegal-synonyms* (*decllist, sorts*) **in**
3 **let** *dict'* = [*dict*(SCOPEUNIT) ⇀ ⟨(VALUE, *nm*)⟩ ↦ **mk**-*ErrorD*() | *nm* ∈ *allsyn*] **in**
4 **let** *decllist'* = *remove-select-in-decllist* (*decllist, illsyn*) (*dict'*) **in**
5 **let** *dict''* = *repeat-collecting-synonyms* (*decllist'*, {}) (*dict*) **in**
6 **let** *sorts'* = *sursorts* ∪ *collect-sorts*(*decllist'*) **in**
7 ⟨*remove-select-in-enclosed-scopeunit*(*decllist*[*i*]), *sort'*) (*dict''*) | 1 ≤ *i* ≤ **len** *decllist* ⟩)

запись: ($Decl_0$ | $Blocksubdef_0$ | $Decomposition_0$)* $Name_0$-**set** → *Dict* →
($Decl_0$ | $Blocksubdef_0$ | $Decomposition_0$)*

Цель Расширить определения отбора для единицы подразделения и для всех определений отбора в единице подразделения, содержащихся в списке определений.

Параметры

decllist Список определений, содержащий определения отбора. Подструктура блока и декомпозиция сервиса для удобства также рассматриваются как определения в этом контексте.

sursorts Множество сортов (включая предопределенные), которые потенциально видимы в списке определений (т.е. возможно зависящие от того выбираются ли они содержащемся определением отбора). Эта информация требуется для того, чтобы достичь корректной обработки в случае, когда предопределенные данные переопределяются.

dict *Dict*, который содержит только дескрипторы для предопределенных сортов и для видимых (но не локальных) синонимов предопределенных сортов.

Результат Список определений, в котором все определения отбора расширены.

Алгоритм

Строка 1 Собрать сорта, которые определены в списке определений или определены в определении отбора.

Строка 2 Пусть *allsyn* обозначает синонимы, которые определены в списке определений или в содержащемся определении отбора. Пусть *illsyn* (которое является подмножеством *allsyn*) обозначает синонимы, которые не могут быть использованы при удалении определений отбора для этого списка определений.

Строка 3 Поместить дескриптор *ErrorD* в *Dict*, чтобы указать, что локально определенные синонимы пока не могут быть использованы.

Строка 4 Удалить определения отбора.

Строка 5 Поместить дескриптор *SynD* в *Dict* для синонимов в окончательный список определений.

Строка 6 Собрать снова сорты, используя на этот раз окончательный список определений.

Строка 7 Удалить определения отбора в содержащихся блоках подразделения.

remove-select-in-decllist(*decllist*, *illsyn*)(*dict*) ≜ (3.3.2)

```
1  (let dict' = repeat-collecting-synonyms(decllis, illsyn)(dict) in
2   if (∃d ∈ elems decllist)(is-Select0(d)) then
3     (if (∃d ∈ elems decllist)(is-Select0(d) ∧ is-wf-simple-expr(s-Expr0(d)(dict')) then
4       (let d ∈ elems decllist be s.t. is-Select0(d) ∧ is-wf-simple-expr(s-Expr0(d)(dict') in
5        let mk-Select0(expr, decllist') = d in
6        let selected = eval-simple-expr, "BOOLEAN")(dict') in
7        let decllist" = ⟨d' ∈ elems decllist | d' ≠ d ⟩ in
8        if selected then
9          remove-select-in-decllist(decllist' ⌢ decllist", illsyn)(dict)
10        else
11         remove-select-in-decllist(decllist", illsyn)(dict))
12      else
13       exit("4.3.2.: Простое выражение не является предопределенным сортом или содержит не определенные идентификаторы"))
14   else
15    decllist)
```

запись: (*Decl*$_0$ | *Blocksubdef*$_0$ | *Decomposition*$_0$)* *Name*$_0$-**set** → *Dict* →
(*Decl*$_0$ | *Blocksubdef*$_0$ | *Decomposition*$_0$)*

Цель Удалить определения отбора для списка определений.

Параметры

decllist	Список определений, содержащий определения отбора.
illsyn	Синонимы, которые не могут быть использованы в простых выражениях определений отбора.
dict	*Dict*, который содержит только дескрипторы для предопределенных сортов и для видимых синонимов (*SynD* или *ErrorD*).

Результат Список определений, в котором все определения отбора удалены.

Алгоритм

Строка 1	Обновить *dict* дескрипторами для синонимов (предопределенных сортов), которые определены в списке определений (но не содержатся в определении отбора)
Строка 2	Если (все еще) существует определение отбора в списке определений, то
Строка 3	Если выражение в определении отбора является корректным, то
Строка 4	Пусть *d* обозначает определение отбора
Строка 6	Оценить простое выражение в *d*.
Строка 7	Создать список определений, в котором *d* удален.
Строка 8	Если определения в определении отбора выбраны, то
Строка 9	Продолжить удаление определений отбора при том, что список определений расширен за счет включения определений из определения отбора.
Строка 11	Если определение отбора не выбрано, то продолжить со списком определений, в котором определение отбора удалено.
Строка 15	Если все определения отбора в списке определений удалены, то вернуть список определений.

eval-simple-expr(*expr*, *sort*)(*dict*) ≜ (3.3.3)

```
1  (let predefqual = get-predef-sort(sort) (dict) in
2   let (as1tree,,) = transform-expr(expr, CONSTANT, {predefqual}) (dict) in
3   eval-expr(as1tree, sort)(dict))
```

запись: *Expr*$_0$ *Char*$^+$ → *Dict* → (*intg* | *Bool*)

Цель Оценить простое выражение

Параметры

expr	Простое выражение AS_0
sort	Проверка сорта выражения. Сорт может быть либо "Булевский", либо " Целое".

Результат META-IV пишет *Intg*, если сорт SDL - "Целое", META-IV определяет тип *Bool*, если сорт SDL - " Булевский".

Алгоритм

Строка 1	Создать *Qual* сорта.
Строка 2	Преобразовать выражение в AS_1, отмечая, что это выражение является (должно быть) постоянным (пассивным) выражением, и что это выражение является (должно быть) специфированного сорта.
Строка 3	Оценить выражение AS_1 (эта функция определена неформально)

repeat-collecting-synonyms(*decllist, illsyn*)(*dict*) ≜ (3.3.4)

1 (**let** *dict'* = *collect-synonyms*(*decllisl, illsyn*)(*dict*) **in**
2 **if** *dict'* = *dict* **then** *dict'* **else** *repeat-collecting-synonyms*(*decllisl, illsyn*)(*dict'*))

запись: $Decl_0$* $Name_0$-**set** → *Dict* → *Dict*

Цель Объединить дескрипторы для всех корректных синонимов (т.е. синонимов, которые на данной стадии не зависят от синонима, определенного в определении отбора)

Параметры

decllist	Список определений, который предстоит просмотреть
illsyn	Синонимы из *decllist*, которые не могут быть использованы, т.е. их дескриптор *ErrorD* не должен быть заменен на дескриптор *SynD*.

Результат Dict заполняется дескрипторами корректных синонимов

Алгоритм

Строка 1	Собрать дескрипторы корректных синонимов в списке определений *decllist* в ходе просмотра *decllist*.
Строка 2	Если нет больше синонимов, которые можно преобразовать к *dict'*, то возвращается *dict*, иначе
Строка 2	Просмотреть список определений еще раз. Поскольку синонимы могут быть взаимозависимы, то еще некоторые определения синонимов могут теперь стать корректными (в контексте *dict'*)

collect-synonyms(*decllist, illsyn*)(*dict*) ≜ (3.3.5)

```
 1  if decllist = then
 2    dict
 3   else
 4    (if is-Synonymdef0(hd decllist) then
 5       (let mk-Synonymdef0(nm, sort, expr) = hd decllist in
 6        let qualset = {qual ∈ dom dict | is-SortD(dict(qual))} in
 7        let sortset =
 8            if sort = nil then
 9              qualset
10             else
11              (trap exit with{} in
12               {get-parent(get-visible-qual(sort, TYPE)(dict)(dict)}) in
13        let synqual = dict(SCOPEUNIT) ⇀ ⟨(VALUE, nm)⟩ in
14        let (as1tree, qset,) =
15            if sortset = {} then
16              (nil, {},)
17             else
18              (trap exit with (nil, {}) in
19               transform-expr(expr, CONSTANT, sortset)(dict + [synqual ↦ mk-ErrorD()])) in
20        if as1tree = nil ∨ card qset ≠ 1 ∨ nm ∈ illsyn then
21          collect-synonyms(tl decllist, illsyn)(dict)
22         else
23          (let sortqual ∈ qset in
24           let d = [synqual ↦ mk-SynD(sortqual, expr)] in
25           collect-synonyms(tl decllist, illsyn)(dict + d)))
26   else
27    collect-synonyms(tl decllist, illsyn)(dict))
```

запись: $Decl_0$* $Name_0$-set → *Dict* → *Dict*

Цель Просмотреть список определений и объединить дескрипторы для всех корректных синонимов

Параметры

decllist	Список определений, содержащий определения синонимов
illsyn	Синонимы из *decllist*, которые не могут быть использованы, т.е. их дескриптор *ErrorD* не должен быть заменен на дескриптор *SynD*.
dict	*Dict*, содержащий дескрипторы для предопределенных сортов и дескрипторы видимых синонимов.

Результат *Dict* с измененными дескрипторами *SynD* корректных синонимов

Алгоритм

Строка 1	Если просмотр закончен, вернуть (измененный) *Dict*.
Строка 4	Если следующее определение в списке - определение синонима, то
Строка 6	Извлечь *Quals* (идентификаторы) предопределенных сортов из *dict*.
Строки 7-12	Если сорт описан в определении синонима, то множество допустимых сортов выражения содержит только этот сорт (если описанный сорт не является корректным, то это множество пусто), иначе мно- жество допустимых сортов включает все предопределенные сорта.
Строка 13	Пусть *synqual* обозначает *Qual* синонима.
Строки 14-19	Попытаться преобразовать выражение в AS_1. Если это не удастся, то или это статическая ошибка (будет рассмотрена ниже), или выражение содержит синонимы, которые еще не были рассмотрены (они могут быть определены либо в определении отбора, либо в tl *decllist*).

Строка 20	Если это не удалось, т.е. если выражение AS_1 *as₁tree* не существует, или выражение неопределено (мощность возвращенного множества сортов отлична от 1), или синоним - один из тех, которые не могут быть использованы, тогда продолжить просмотр списка определений, иначе
Строка 23	Пусть *sortqual* обозначает сорт выражения.
Строки 24-25	Модифицировать *dict* дескрипторами синонимов и продолжить просмотр списка определений.
Строка 27	Если первое определение в списке определений не является определением синонима, то продолжить просмотр списка определений.

is-wf-simple-expr(*expr*)(*dict*) ≜ (3.3.6)

```
1  (let boolqual = get-predef-sort("BOOLEAN")(dict) in
2   trap exit with false in
3   (let (,,) =transform-expr(expr, CONSTANT, {boolqual})(dict) in
4    true))
```

запись: $Expr_0 \rightarrow Dict \rightarrow Bool$

Цель Проверить, правильно ли построено простое логическое выражение. Если не правильно, то может быть потому, что оно определено в терминах синонимов, которые не были включены еще в *dict*.

Параметры

expr	Выражение AS_0, которое нужно проверить

Результат True, если правильно построено

Алгоритм

Строка 1	Извлечь *Qual* (идентификатор) булевского сорта. Множество допустимых сортов для выражений только включает этот сорт.
Строки 2-3	Если преобразующая функция отловлена, то возвращается false, иначе
Строка 4	Возвращается true.

remove-select-in-enclosed-scopeunit(*decl, sorts*)(*dict*) ≜ (3.3.7)

```
(let level = dict(SCOPEUNIT) in
 cases decl:
   (mk-Blockdef0(mk-Id0(q, nm), decll, blksub, tid)
     → (let dict' = dict + [SCOPEUNIT ↦ level ⌢ ⟨(BLOCK, nm)⟩] in
         (let (decll', blksub') =
             if blksub = nil then
                (remove-select(decll, sort)(dict'), nil)
              else
                (let dlist = remove-select(decll ⌢ ⟨blksub⟩, sorts)(dict') in
                 let i be s.t. is-Blocksubdef0(dlist[i] in
                 (⟨dlist[n] | 1 ≤ n ≤ len dlist ∧ n ≠ i, dlist[i]⟩)) in
         (mk-Blockdef0(mk-Id0(q, nm), decll', blksub', tid))),
   mk-Servicedef0(mk-Id0(q, nm), sigl, decll, body, tid)
     → (let dict' = dict + [SCOPEUNIT ↦ level ⌢ ⟨(SERVICE, nm)⟩] in
         let decll' = remove-select(decll, sorts)(dict') in
         mk-Servicedef0(mk-Id0(q, nm), sigl, decll', body, tid)),
   mk-Chandef0(nm, p1, p2, chansub, tnm)
     → if chansub = nil then
           decl
         else
           (let mk-Chansubdef0(id,decll,tid) = chansub in
            let nm' = if id = nil then nm else s-Name0(id) in
            let dict' = dict + [SCOPEUNIT ↦ level ⌢ ⟨(SUBSTRUCTURE, nm')⟩] in
            let decll' = remove-select(decll, sorts)(dict') in
            mk-Chandef0(nm, p1, p2, mk-Chansubdef0(id,decll,tid), tnm)),
   mk-Prdef0(mk-Id0(q, nm), inst, parm, inpset, decll, body, tid)
     → (let dict' = dict + [SCOPEUNIT ↦ level ⌢ ⟨(PROCESS, nm')⟩] in
         let (decll', body') =
             if is-Body0(body) then
               (remove-select(decll, sorts)(dict'), body)
              else
               (let dlist = remove-select(decll ⌢ ⟨body⟩, sorts)(dict') in
                let i be s.t. is-Decomposition0(dlist[i]) in
                (⟨dlist[n] | 1 ≤ n ≤ len dlist ∧ n ≠ i⟩, dlist[i])) in
         mk-Prdef0(mk-Id0(q, nm), inst, parm, inpset, decll', body', tid)),
   mk-Blocksubdef0(id, decll, tid)
     → (let (, nm) = level[len level] in
         let nm' = if id = nil then nm else s-Name0(id) in
         let dict' = dict + [SCOPEUNIT ↦ level ⌢ ⟨(SUBSTRUCTURE, nm')⟩] in
         let decll' = remove-select(decll, sorts)(dict') in
         mk-Blocksubdef0(id, decll', tid)),
   mk-Decomposition0(decll)
     → (let decll' = remove-select(decll, sorts)(dict) in
         if (∀d ∈ elems decll')(is-Sigroutedef0(d) ∨
                                (is-Connect0(d) ∧ is-Id0(s-Connectpoint0(d))) ∨
                                is-Servicedef0(d)) then
           mk-Decomposition0(decll')
         else
           exit("§4.3.3: Выбранное определение недопустимо в этой единице подразделения")
   T → decl))
```

запись: (*Decl*$_0$ | *Blocksubdef*$_0$ | *Decomposition*$_0$) *Name*$_0$-**set** → *Dict* → (*Decl*$_0$ | *Blocksubdef*$_0$ | *Decomposition*$_0$)

Цель Удалить определения отбора во вложенных единицах подразделения.

Параметры

decl Определение, которое может быть единицей подразделения.

Алгоритм

Строка 1 — Обозначить контекст (в котором встречается определение) как уровень (*level*).

Строки 3-12 — Если это определение - определение блока, то поместить определения отбора в список определений блока (строка 7). Если блок содержит подструктуру, то он также рассматривается как определение. Определения отбора перемещаются в контексте блока, т.е. SCOPEUNIT - это *Dict*, измененный для обозначения блока.

Строки 13-41 — Проделать то же для других альтернативных единиц подразделения в том случае, если определение является определением блока.

Строки 42-49 — Если определение является декомпозицией (которая может содержать определения отбора, хотя это не единица подразделения), тогда проверить, что конечный список определений (после перемещения определений отбора) содержит только определения маршрутов сигналов сервиса, соединения маршрутов сигналов сервиса и определение сервиса. *Connectpoint*$_0$ в соединении маршрута сигнала сервиса не должна быть ENV (этот случай не проверяется синтаксически, если соединение содержится в определении отбора).

collect-illegal-synonyms(*decllist, sorts*) ≜ (3.3.8)

```
1  (if decllist = ◊ then
2      ( {},{} )
3     else
4      (let (restsyn, restillsyn) = collect-illegal-sysonyms(tl decllist, sorts) in
5       let (synset, illsynset) =
6        cases hd decllist :
7            (mk-Synonymdef0(nm, sort, )
8             → if (sort ≠ nil ∧ s-Name0(sort) ∈ sorts ∧ s-Qualifier0(sort) ≠ ⟨(SYSTEM,)⟩) then
9                 ({nm}, {nm})
10               else
11                ({nm}, {}),
12           mk-Select0(, dlist)
13            → collect-illegal-synonyms(dlist, sorts),
14           T → ({}, {})) in
15     (restsyn ∪ synset, restillsyn ∪ illsynset ∪ (restsyn ∩ synset ))))
```

запись : *Decl*$_0$* *Name*$_0$-set → *Name*$_0$-set *Name*$_0$-set

Цель — Собрать синонимы, которые (потенциально) определены в списке определений, и определить те из них, которые не могут быть использованы ни в каком простом выражении определений отбора.

Параметры

decllist — Список определений

sorts — Сорта, которые (потенциально) определены в списке определений.

Результат — Множество потенциально определенных синонимов и синонимы, которые не могут быть использованы.

Алгоритм

Строка 1 — Когда просмотр списка определений закончен, возвратить пустые множества

Строка 4 — Собрать множества синонимов в оставшемся списке определений.

Строка 5 — Пусть *synset* обозначает синонимы, определенные в этом определении, и пусть *illsynset* обозначает те из них, которые не могут быть использованы.

Строки 7-11 Если определение есть определение синонима, тогда имя синонима - *synset*, это имя не может использоваться, если имя сорта есть одно из сортов отображения и квалификатор не обозначает системный уровень. Отметим, что *Dict* фактически содержит только дескрипторы для предопределенных сортов. Эта проверка, следовательно, важна только, если предопределенный сорт был переопределен.

Строка 12 Если определение есть определение отбора, тогда собрать множества синонимов для содержащихся определений.

Строка 15 Окончательное множество определенных имен синонимов есть имена оставшихся определений, объединенные с именами этого определения. Окончательное множество неиспользуемых синонимов есть неиспользуемые синонимы для оставшихся определений, объединенные с именами для этого определения, объединенные с именами, определенными в оставшихся из определения и в этом определении.

collect-sorts (*declist*) ≜ (3.3.9)

```
 1  (if decllist = ⟨⟩ then
 2     {}
 3   else
 4    (collect-sorts(tl decllist)
 5     cases hd decllist :
 6      (mk-Syntypedef0(nm,,,,),
 7       mk-Partialtypedef0(nm,,,,)
 8        → {nm},
 9       mk-Select0(, dlist)
10        → collect-sorts(dlist),
11      T → {})))
```

запись: $Decl_0^* \rightarrow Name_0$-**set**

Цель Собрать имена сорта для сортов, которые потенциально определены в списке определений.

Параметры

declist Список определений

Результат Множество, состоящее из потенциально определенных имен сорта (были ли они фактически определены зависит от определений отбора).

Алгоритм

Строка 1 Если просмотрен весь список определений, возвращается пустое множество.

Строки 6-7 Если определение есть частичное определение типа или определение подтипа, тогда извлекается имя сорта.

Строка 9 Если определение есть определение отбора, тогда собрать потенциально определенные имена сортов в содержащемся списке определений.

3.4 Преобразование определений

transform-decllist(*decllist*)(*dict*) = ≜ (3.4.1)

```
1  (if decllist = ◊ then
2      ({},[])
3     else
4      (let (as1 dcl, d) = transform-decl(hd decllist)(dict) in
5       let (as1 dcll, d') = transform-decllist(tl decllist)(dict) in
6       if dom d ∩ dom d' ≠ {} then
7         exit("§2.2.2: Два определения в одной единице подразделения и одном классе объектов определяют одно имя")
8       else
9        (as1 dcl ∪ as1 dcll, d + d')))
```

запись: $Decl_0^* \rightarrow Dict \rightarrow Decl_1$-set $Dict$

Цель Преобразовать список определений AS_0 в множество определений AS_1.

Параметры Список определений AS_0.

Результат Определения AS_1 и требования *Dict* из списка определений AS_0. Отметим, что это не полный *dict*, который возвращается, в отличие от функций преобразования выражения и графа. Это означает, что ни какой возвращенный дескриптор (кроме *ValueidD*, который используется строго локально в аксиомах и параметрах генератора и *ErrorD*, который используется локально для отлавливания рекурсивных определений), не влияет на содержание *dict*. Эквивалентность между возвращенными дескрипторами и *dict* осуществляется посредством конструкции be such that в функции *transform-system*, как упоминалось ранее.

Алгоритм

Строка 4 Преобразовать первое определение из списка определений.

Строка 5 Преобразовать остальные определения.

Строка 6 Пара имя-класс объектов (отраженные в *Qual*) первого определения должны быть отделены от пар, представляющих остальные определения.

Строка 9 Возвратить требования *Dict* и определения AS_1

transform-decl(decl)(dict) = ≜ (3.4.2)

```
1  ( cases decl :
2      (mk-Blockdef0(,,,)
3        → transform-blockdef(decl)(dict),
4      mk-Chandef0(,,,,)
5        → transform-channeldef(decl)(dict),
6      mk-Prdef0(,,,,,)
7        → transform-processdef(decl)(dict),
8      mk- Sigdef0()
9        → transform-signaldef(decl)(dict),
10     mk-Procdef0(,,,,)
11       → transform-proceduredef(decl)(dict),
12     mk-Partialtypedef0(,,,,)
13       → transform-partial-typedef(decl)(dict),
14     mk-Syntypedef0(,,,,)
15       → transform-syntype(decl)(dict),
16     mk-Sortgenerator0(,,,,)
17       → transform-sortgenerator(decl)(dict),
18     mk-Synonymdef0(,,)
19       → transform-synonymdef(decl)(dict),
20     mk-Vardef0(,,)
21       → transform-vardef(decl)(dict),
22     mk-Viewdef0()
23       → transform-viewdef(decl)(dict),
24     mk-Importdef0()
25       → transform-importdef(decl)(dict),
26     mk-Sigroutedef0(,,)
27       → if is-BlockD(dict(dict(LEVEL)))
28           then transform-signalroutedef(decl)(dict)
29           else transform-servicesigroutedef(decl)(dict),
30     mk-Signallistdef0(,)
31       → transform-signallistdef(decl)(dict),
32     mk-Timerdef0()
33       → transform-timerdef(decl)(dict),
34     mk-Servicedef0(,,,,)
35       → build-service-descriptor(decl)(dict),
36    T → ({},[]))
```

запись: $Decl_0$ → *Dict* → $Decl_1$-**set** *Dict*

Цель Преобразовать определение AS_0 в определение AS_1.

Результат См.*transform-decllist.*

Алгоритм Преобразовать любое

Строка 1	Определение блока или
Строка 4	Определение канала или
Строка 6	Определение процесса или
Строка 8	Определение сигнала или
Строка 10	Определение процедуры или
Строка 12	Частичное определение типа данных или
Строка 14	Определение подтипа или
Строка 16	Генератор сорта или
Строка 18	Определение синонима или
Строка 20	Определение переменной или
Строка 22	Определение обозревания или
Строка 24	Определение импорта или

Строки 26-29	Определение маршрута сигнала или маршрута сигнала сервиса, зависящее от вложенной единицы подразделения или
Строка 30	Определение списка сигналов
Строка 32	Определение таймера
Строка 29	Определение маршрута сигнала сервиса или
Строка 34	Определение сервиса
Строка 19	Ничего, так как другой вид определений (т.е. соединения) обрабатываются в другом месте.

3.4.1 Определения блоков

transform-blockdef(mk-*Blockdef*$_0$(*bid, decllist, subdef, tid*))(*dict*) ≜ (3.4.1.1)

```
(let mk-Id0(q, bnm) = bid in
 let bqual = dict(SCOPEUNIT) ⇝ ⟨(BLOCK, bnm)⟩ in
 let dict' = dict + [SCOPEUNIT ↦ bqual] +
    [DATATYPEDEF ↦ initialdatadef(dict)] in
 let explicit = (∃d ∈ elems decllist)(is-Sigroutedef0(d)) in
 let (chandefl, routedefl, connects, exp, imp) =
     implicit-channels-and-signal-routes(decllist)(dict') in
 let (outerchannels, cdi) = transform-decllist(chandefl)(dict) in
 let (as1 dcl, di) = transform-decllist(decllist ⇝ routedefl)(dict') in
 let (as1 connect, connectmap) = transform-block-connect(decllist, {}, [])(dict') in
 if ((∃l ∈ elems decllist)(is-Prdef0(l)) ∨ subdef ≠ nil) ∧ q = ◊ ∧ tid ∈ {bid, nil} then
    if subdef = nil then
       (let descr = [bqual ↦ mk- BlockD(exp, imp, explicit, connectmap)] in
        let as1 block = make-as1 tree(BLOCK, bnm, as1 dcl ⇝ as1 connect, nil, nil, nil)(dict) in
        (outerchannels ∪ {as1 block}, descr + di + cdi))
    else
       (let mk-Blocksubdef0(subid, subdecll, taillid) = subdef in
        let mk-Id0(q', bnm') = if subid = nil then mk-Id0(◊, bnm) else subid in
        let chansubqual = dict(SCOPEUNIT) ⇝ ⟨(SUBSTRUCTURE, bnm')⟩ in
        let subqual = if chansubqual ∈ dom dict then
                          chansubqual
                       else
                          bqual ⇝ ⟨(SUBSTRUCTURE, bnm' )⟩ in
        let dict" = [SCOPEUNIT ↦ subqual,
                     DATATYPEDEF ↦ initialdatadef(dict')] in
        let (as1 subdcl, subdi) = transform-decllist(subdecll)(dict") in
        let as1 connect' = transform-substructure-connect(subdecll ⇝ connects, {},[])(dict") in
        let as1 tree = make-as1 tree(SUBSTRUCTURE, bnm',
                                     as1 subdcl ∪ as1 connect', nil, nil, nil)(subdi) in
        let as1 block' = make-as1 tree(BLOCK, as1 tree, nil, nil)(dict) in
        let descr = [bqual ↦ mk-BlockD(exp, imp, explicit, connectmap)] +
          (if chansubqual ∈ dom dict then [] else [subqual ↦ mk-BlocksubD()]) in
        (q' ≠ ◊
          → exit ("§2.4.1: Определяющие имена могут быть квалифицированы только в удаленных описаниях"),
        taillid ∉ {nil, subid}
          → exit ("§2.2.2: Конечное имя в определении подструктуры блока отличается от определяющего имени"),
        ¬(∃d ∈ elems subdecll)(is-Blockdef0(d))
          → exit ("§3.2.2: Подструктура блока не содержит определение блока")
        T→ (outerchannels ∪ {as1 block'}, descr +  di + subdi + cdi)))
 else
  (¬(∃l ∈ elems decllist)(is-Prdef0(l) ∨ subdef ≠ nil)
       → exit ("§2.4.3: Блок должен содержать либо один или более процессов, либо описание подструктуры"),
    q ≠ ◊
       → exit ("§2.4.1: Определяющие имена могут быть квалифицированы только в удаленных описаниях"),
    T → exit ("§2.2.2: Конечное имя в определении подструктуры блока отличается от определяющего имени")))
```

запись : *Blockdef*$_0$ → *Dict* → *Decl*$_1$-**set** *Dict*

Цель Преобразовать определение блока AS_0 в определение блока AS_1.

Параметры Определение блока AS_1, содержащее

bid	Не квалифицированный идентификатор блока
decllist	Его список определений
subdef	Его необязательная подструктура
tid	Его не квалифицированный конечный идентификатор

Результат См. *transform-decllist.*

Алгоритм

Строки 2-3	Построить *Qual*, обозначающий идентификатор блока (*bqual*) и модифицировать SCOPEUNIT входа Dict, чтобы обозначить контекст блока. Построить начальный *Data-type-definition*$_1$ для этого блока.
Строка 5	Пусть *explicit* истинно (**true**), если явные маршруты явных сигналов определены для блока.
Строка 6	Создать неявные определения каналов AS_0, подразумеваемые при экспорте и импорте из и в блок, а так же таблицами *ExpimpchanD* (см. определение домена *ExpimpchanD*), которое используется, чтобы обеспечить взаимодействие определений каналов с подструктурой и окружающей средой.
Строка 8	Преобразовать неявные определения каналов в AS_1.
Строка 9	Преобразовать содержащийся список определения с модифицированным SCOPEUNIT входа *dict*, чтобы обозначить блок.
Строка 10	В блоке преобразовать канал в соединения маршрута сигнала. Возвратить также *BlockconnectionD* для блока.
Строка 11	Если подструктура пропущена, тогда в блоке должно быть по крайней мере одно определение процесса. Идентификатор блока не должен квалифицироваться, и, если конечный идентификатор определяется, то он должен быть эквивалентен идентификатору блока.
Строки 12-15	Если блок не содержит подструктуру блока, то возвратить определение AS_1 неявных каналов (*outerchannels*), блока (*as*$_1$*block*) и требование *Dict* для неявных каналов (*cdi*), блока (*descr*) и содержащихся определений (*di*).
Строка 17	Произвести декомпозицию подструктуры блока.
Строка 18	Если имя подструктуры блока опускается, то оно является таким же, как имя блока. Пусть *bnm* обозначает имя подструктуры блока
Строки 19-20	Пусть *subqual* обозначает *Qual* подструктуры. Если подструктура блока извлекается из подструктуры канала, то *subqual* обозначает *Qual* подструктуры канала.
Строка 24	Модифицировать SCOPEUNIT и DATATYPEDEF, как в строке 3.
Строка 26	Преобразовать список определения подструктуры.
Строка 27	Создать определение AS_1 блока (*as*$_1$*block*),
Строка 28	Создать связи AS_1 из связей AS_0 в *subdecll.*
Строка 30	Определение AS_1 подструктуры блока (*as*$_1$*tree*),
Строки 31-32	Вход *Dict* блока и, если подструктура не обозначает подструктуру канала, тогда также вход *Dict* для подструктуры блока (*subqual*).
Строка 33	Имя (идентификатор) подструктуры блока не должно быть квалифицировано (так как оно не является удаленным определением).
Строка 35	Если завершающий id определен, то он должен быть таким же, как идентификатор подструктуры (*subid*).
Строка 37	Подструктура должна содержать по крайней мере одно определение блока.
Строка 39	Возвратить определение AS_1, как в строке 15, и требования *Dict* для неявных каналов (*cdi*) для блока, подструктуры (*descr*), содержащихся в блоке определений (*di*) и определений, содержащихся в подструктуре (*subdi*).
Строка 41	В блоке должно быть по крайней мере одно определение процесса, если подструктура не определяется, и
Строка 33	Идентификатор блока не должен быть квалифицированным и
Строка 45	Если конечное имя определяется, тогда оно должно быть равно имени блока

initialdatadef(dict) ≜ (3.4.1.2)

1 (**let** **mk**-*Data-type-defenition*$_1$(*unm*,,,,) = *dict*(DATATYPEDEF),

2 *qual* = *dict*(SCOPEUNIT) ⌢ ⟨(TYPE, *unm*)⟩ **in**

3 **let** *typeid* = *make-as*$_1$*-identifier*(*qual*)(*dict*) **in**

4 **mk**-*Data-type-definition*$_1$(*create-unique-name*(), {*typeid*},{},{},{}))

запись: *Dict* → *Data-type-definition*$_1$

Цель Построить определение типа данных AS_1 для единицы подразделения, где идентификатор, представляющий определение типа данных для единицы подразделения окружения, включается в тип единицы. Литералы, операции и равенства добавляются в определение типа данных во время преобразования частичных определений типа данных в единице подразделения

Алгоритм

Строка 1 Извлечь определение типа данных AS_1 единицы подразделения окружения.

Строка 2 Построить *Qual* идентификатора типа единицы подразделения окружения.

Строка 3 Построить идентификатор типа AS_1.

Строка 4 Возвратить новое определение типа данных, которое имеет уникальное имя, и в котором идентификатор типа единицы подразделения окружения включается в тип, и которое не содержит литералов, операций или равенств

3.4.2 Определение каналов

```
transform-channeldef (mk-Chandef0(cnm, chanpath, ochanpath, csub, tnm))(dict)≜          (3.4.2.1)
 1  (let mk-Chanpath0(ori, dest, siglist) = chanpath in
 2   let orid = get-visible-qual(ori, BLOCK)(dict),
 3       destid = get-visible-qual(dest, BLOCK)(dict),
 4       sigidset = transform-signallist(siglist)(dict),
 5       sigidset' =
 6       if ochanpath = nil then
 7         {}
 8        else
 9         (let mk-Chanpath0(orig', dest', osiglist) = ochanpath in
10          let orid' = get-visible-qual(orig', BLOCK)(dict),
11              destid' = get-visible-qual(dest', BLOCK)(dict) in
12          if orid = destid' ∧ destid = orid' then
13            transform -signallist(osiglist)(dict)
14           else
15            exit("§2.5: Второй путь в определении канала должен обозначать обратное направление первого пути") in
16 if orid = destid then
17   exit("§2.5: Окончания каналов должны быть различны")
18  else
19   (let as1orid = if ori = ENV then ENVIRONMENT else make-as1-identifier(orid)(dict),
20        as1destid = if dest = ENV then
21                        ENVIRONMENT
22                       else
23                        make-as1-identifier(destid)(dict),
24        as1sigs = make-as1idset(sigidset)(dict),
25        as1sigs' = make-as1idset(sigidset')(dict),
26        cqual = dict(SCOPEUNIT) ⇝ ⟨(CHANNEL, cnm)⟩ in
27      (¬ is-local(orid)(dict) ∨ ¬ is-local(destid)(dict)
28        → exit("§2.5: Окончание канала определяется в другой единице подразделения, отличной от канала"),
29      tnm ∉ {cnm, nil}
30        → exit("§2.2.2: Конечное имя в определении канала отличается от определяющего имени"),
31      csub ≠ nil
32        → tranaform-channel-sub(cauh, cnm, sigidset, sigidset', orid, destid)(dict),
33      T → (let descr = [cqual ↦ mk-ChannelD(orid, destid, sigidset, sigidset', nil)] in
34            let path1 = mk-Channel-path1(as1orid, as1destid, as1sigs),
35                path2 = if as1sigs' = {} then
36                            nil
37                            else
38                            mk-Channel-path1(as1destid, as1orid, as1sigs') in
39            ({mk-Channel-definition1(name-to-name1(cnm), path1, path2)}, descr)))))
```

запись: $Chandef_0 \rightarrow Dict \rightarrow Decl_1$-**set** $Dict$

Цель Преобразовать определения канала в AS_1

Результат Определение канала AS_1 и требование *dict*, содержащее канальный дескриптор и дескриптор подструктуры канала (если он есть)

Параметры Определение канала AS_0, содержащее

cnm	Имя канала
chanpath	Первый путь
ochanpath	Второй необязательный путь канала
csub	Необязательная подструктура канала
tnm	Имя, заканчивающее определения

Алгоритм

Строка 1 Произвести декомпозицию первого пути

Строки 2-15	Составить *Qual* из исходящего блока *(orid)*, блока назначения (*destid*), сигналов (*sigidset*) и сигналов, передаваемых в обратном направлении (*sigidset'*).
Строки 9-15	Если канал используется в двух направлениях, исходящая оконечная точка одного из направлений должна быть той же, что и оконечная точка назначения другого направления.
Строка 16	Исходящий блок не должен быть тем же, что и блок назначения.
Строки 19-25	Составить идентификаторы $Identifier_1$ исходящего блока ($as_1 orid$), блока назначения ($as_1 destid$), сигналов ($as_1 sigidset$) и сигналов, передаваемых в обратном направлении ($as_1 sigidset'$).
Строка 26	Модифицировать SCOPEUNIT для обозначения идентификатора канала.
Строки 27-28	Исходящий блок и блок назначения должны быть определены на том же уровне, где определен канал, и
Строка 29	Имя, заканчивающее определение канала, должно быть эквивалентно имени канала, если оно определено
Строка 31	Если есть подструктура канала, то преобразовать подструктуру канала, иначе
Строка 33	Сформировать требование *Dict* для канала.
Строки 34-35	Сформировать два пути AS_1.
Строка 39	Вернуть определение канала AS_1 и его требование *Dict*.

transform-channel-sub(*csub, nm, sigset, osigset, endpoint*1, *endpoint*2)(*dict*)≜ (3.4.2.2)

```
1  (let mk-Chansubdef0(subid, decllist, tailid) = csub in
2   let mk-Id0(q, nm') = if subid = nil then mk-Id0(⟨⟩, nm) else subid in
3   let mk-Id0(qtail, nmtail) = if tailid = nil then mk-Id0(q, nm') else tailid in
4   if q ≠ ⟨⟩ ∨ qtail ≠ ⟨⟩ then
5     exit("§2.4.1 Завершающее имя в определении подструктуры канала может быть квалифицировано только в удаленных определениях")
6    else
7     if nmtail ≠ nm' then
8       exit("§2.2.2  Завершающее имя в определении подструктуры канала отличается от определяющего имени")
9      else
10      (let level = dict(SCOPEUNIT) in
11       let newbnm = create-unique-name() in
12       let newbqual = level ⌢ ⟨(BLOCK, newbnm)⟩ in
13       let newc1 = create-unique-name(),
14           newc2 = create-unique-name() in
15       let path1 = mk-Chanpath0(as0-id(endpoint1), as0-id(newbqual),⟨as0-id(q) | q∈ sigset⟩),
16           opath1 =
17           if osigset = {}
18             then nil
19             else mk-Chanpath0(as0-id(newbqual), as0-id(endpoint1), ⟨ as0-id(q) | q∈ osigset⟩),
20          path2 = mk-Chanpath0(as0-id(newbqual), as0-id(endpoint2), ⟨as0-id(q) | q∈ sigset⟩),
21          opath2 =
22          if osigset = {}
23            then nil
24            else mk-Chanpath0(as0-id(endpoint2), as0-id(newbqual), ⟨as0-id(q) | q∈ osigset⟩) in
25       let chandef1 = mk-Chandef0(newc1, path1, opath1, nil, newc1),
26           chandef2 = mk-Chandef0(newc2, path2, opath2, nil, newc2) in
27       let substructurequal = level ⌢ ⟨(SUBSTRUCTURE, nm')⟩ in
28       let dict' = dict + [SCOPEUNIT ↦ substructurequal] in
29       let decllist' = replace-connects(decllist, newc1, endpoint1, newc2, endpoint2)(dict') in
30       let newchannels = (endpoint1, newc1, endpoint2, newc2) in
31       let blockdef =
32         mk-Blockdef0(mk-Id0(⟨⟩,newbnm),⟨⟩,mk-Blocksubdef0(mk-Id0(⟨⟩,nm'),decllist',nil),nil) in
33       let channelqual = level ⌢ ⟨(CHANNEL, nm)⟩ in
34       let di = [channelqual      ↦ mk-ChannelD(endpoint1, endpoint2, sigset, osigset, newchannels),
```

35 *substructurequal*↦ **mk-***ChannelsubD*(*newbqual*)] **in**

36 **let** (*as*$_1$ *declset*, *dict*') = *transform-decllist*(⟨*chandef*1, *chandef*2, *blockdef*⟩)(*dict*) **in**

37 (*as*$_1$ *declset*, *dict*' + *di*)))

запись: *Chansubdef*$_0$ *Name*$_0$ *Signalqual*-**set** *Signalqual*-**set** *Endpoint Endpoint*→ *Dict*→ *Decl*$_1$-**set** *Dict*

Цель Преобразовать определение подструктуры канала в два определения каналов и определение блока

Параметры

csub	Определение подструктуры канала AS_0
nm	Имя вложенного определения канала
siglist	Сигналы, переданные вложенным каналом.
osiglist	Необязательные сигналы, идущие в противоположном направлении.
*endpoint*1	Блок, из которого передается *siglist*.
*endpoint*2	Блок, в который передается *siglist*. *Osiglist* сигналы передаются от *endpoint*2 в *endpoint*1.

Результат См. *transform-decllist*

Алгоритм

Строка 1	Пусть *subid* обозначает идентификатор подструктуры канала, *decllist* список определений, вложенных в подструктуру, и *tailid* идентификатор, заканчивающий определение подструктуры.
Строка 2	Если идентификатор(имя) не определяется для подструктуры, то имя канала окружения наследуется.
Строка 3	Пусть *qtail* обозначает квалификатор в заканчивающем идентификаторе (*subid*) и пусть *nmtail* обозначает имя конечного идентификатора.
Строка 4	Имя подструктуры не должно быть квалифицировано (так как оно не является удаленным описанием), и идентификатор, заканчивающий определение, должен быть эквивалентен идентификатору, начинающему определение, если он определен
Строка 7	Заканчивающее имя должно быть эквивалентно имени подструктуры канала
Строка 10	Пусть *level* обозначает *Qual* блока, где определяется канал
Строка 11	Создать индивидуальное новое имя для блока, который необходимо построить
Строка 12	Построить вход *Dict* для блока, который необходимо построить
Строки 13-14	Создать индивидуальные новые имена для двух каналов, которые необходимо построить
Строки 15-16	Построить пути для первого из двух новых каналов
Строки 20-21	Построить пути для второго из двух новых каналов
Строки 25-26	Построить два новых канала AS_0
Строка 29	Заменить связи конечных точек канала (*Connect*$_0$*s*) связями подструктуры блока (*Connect*$_0$*s*) в списке определения подструктуры канала.
Строка 31	Построить новый блок. Он содержит пустой список определений и подструктуру блока, содержащую модифицированный список определений
Строка 34	Пусть *di* обозначает *Dict*, содержащий дескриптор для вложенного канала и дескриптор для подструктуры канала.
Строка 36	Преобразовать два построенных определения каналов и определение блока в AS_1
Строка 37	Возвратить эти определения AS_1 и *Dict*, который включает в себя дескрипторы для трех определений, дескрипторы для содержащихся определений, дескриптор для вложенного канала и дескриптор для подструктуры канала.

replace-connects(*decllist*, *newc*1, *endpoint*1, *newc*2; *endpoint*2)(*dict*) ≜ (3.4.2.3)

```
1  (let conset = {d ∈ elems decllist | is-Connect0(d)} in
2   if card conset = 2 then
3     (let {mk-Connect0(blkid1, clist1), mk-Connect0(blkid2, clist2)} = conset in
4      let bqual1 = get-visible-qual(blkid1, BLOCK)(dict),
5          bqual2 = get-visible-qual(blkid2, BLOCK)(dict) in
6      if {bqual1, bqual2} = {endpoint1, endpoint2} then
7        (let (c1,c2) = if bqual1 = endpoint1 then (newc1, newc2) else (newc2, newc1) in
8         let decllist' = ⟨ decllist[i] | i ∈ ind decllist ∧ ¬ is-Connect0(decllist[i])⟩ in
9         decllist' ⌢
10        ⟨mk-Connect0(mk-Id0(⟨⟩,c1), clist1),
11         mk-Connect0(mk-Id0(⟨⟩,c2), clist2)⟩)
12     else
13       exit("§3.2.3: Идентификаторы блоков в соединениях не обозначают конечные точки канала"))
14  else
15   exit("§3.2.3: Должно быть ровно два соединения конечных точек канала"))
```

запись: $Decl_0^*\ Channame_0\ Endpoint\ Channame_0\ Endpoint \rightarrow Dict \rightarrow Decl_0^+$

Цель Изменить список определений подструктуры канала в список определений подструктуры блока так, чтобы список содержал соединения каналов вместо соединений конечных точек каналов.

Параметры

decllist	Список определений, который следует заменить
*newc*1,*newc*2	Имена двух искусственных каналов, которые заменяют канал, содержащий подструктуру канала.
*endpoint*1, *endpoint*2	Конечные точки каналов, содержащие каналыную подструктуру. *newc*1 подсоединяется к *endpoint*1, *newc*2 подсоединяется к *newc*2.

Результат Новый список определений, который должен быть вложен в искусственную подструктуру блока.

Алгоритм

Строка 1	Извлечь связи окончаний каналов из списка определений.
Строки 2 и 13	Должно быть ровно два таких соединения конечных точек каналов.
Строка 3	Произвести декомпозицию множества, содержащего два соединения конечных точек каналов.
Строки 4-5	Построить *Qual* идентификаторов блоков, упоминаемых в соединениях двух конечных точек каналов.
Строка 6	Эти два идентификатора блока должны быть такими же идентификаторами блоков, как упомянутые в определении каналов.
Строка 7	Пусть *c*1 обозначает имя канала, которому соответствует *clist1*, и *clist2* - имя канала, которому соответствует *clist2*
Строка 8	Построить список определений, куда были перемещены соединения двух конечных точек каналов, и
Строки 9-11	Возвратить этот список, объединенный с двумя искусственными соединениями каналов

3.4.3 Определения процессов

transform-processdef(mk-*Prdef*$_0$(*pid*,*ins*,*pl*,*sigl*,*decll*,*body*,*tid*)(*dict*) ≜ (3.4.3.1)

```
1  (let mk-Id0(q, name) = pid in
2   let mk-Instances0(ini, maxi) = if ins = nil then mk-Instances0(nil, nil) else ins in
3   let ini' = if ini = nil then 1 else eval-simple-expr(ini, "INTEGER")(dict),
4       max' = if maxi = nil then infinite else eval-simple-expr(maxi, "INTEGER")(dict) in
5  (q ≠ ⟨⟩
6      → exit ("§2.4.1: Определяющие имена могут быть квалифицированы только в удаленных определениях"),
7  tid ∉ {pid, nil}
8      → exit ("§2.2.2: Завершающее имя в определении процесса отличается от определяющего имени"),
9  ini' > max'
10     → exit ("§2.2.3: Начальный номер больше максимального номера экземпляра"),
11 ini' < 0
12     → exit ("§2.2.3: Начальный номер экземпляров меньше нуля"),
13 max' = 0
14     → exit ("§2.2.3: Максимальный номер экземпляров равен нулю"),
15 T   → (let d = [ IMPLIED        ↦{},
16                  OUTSIGNALS     ↦{},
17                  SCOPEUNIT      ↦dict(SCOPEUNIT) ⌢ ⟨(PROCESS, name)⟩,
18                  DATATYPEDEF    ↦initialdatadef(dict)] in
19         let (as1 list, pd, dict') = transform-processparm(pl)(dict + d),
20           sigset = transform-validinputset(sigl)(dict + d),
21           (as1 declset, ddict) = transform-decllist(decll)(dict + d) in
22         if dom dict' ∩ dom ddict ≠ {} then
23           exit ("§2.2.2: Имена формальных параметров должны отличаться от имен переменных")
24           else
25           (let (servicedeclset1, as1 body, bdict, connectmap) =
26                transform-process-body(body)(dict + d),
27                outsigs = bdict(OUTSIGNALS),
28                number = mk-Number-of-intances1(ini', max') in
29            if (∃ q1, q2 ∈ outsigs ∪ sigset)(len q1 > len q2 ∧ ⟨q1[i] | 1 ≤ i ≤ len q2⟩ = q2) then
30              exit ("§3.3: Процесс использует сигналы на различных уровнях обработки того же самого сигнала")
31            else
32              (let (as1 declset', vdict) = make-implicit-vardecl(bdict),
33                   as1 totset = as1 declset ∪ as1 declset' ∪ servicedeclset1,
34                   as1 tree =
35                   make-as1tree(PROCESS,name,as1totset,number,as1list,as1body)(ddict) in
36               let delem = [d(SCOPEUNIT) ↦ mk-ProcessD(pd,sigset,outsigs,connectmap)] +
37                   dict' + ddict + vdict in
38               ({as1 tree}, delem))))))
```

запись : *Prdef*$_0$→*Dict*→*Process-definition*$_1$-set *Dict*

Цель Преобразовать определение процесса в AS$_1$

Параметры Определение процесса AS$_0$, содержащее

pid	Идентификатор процесса
ins	Конструкция "числа экземпляра"
pl	Список формальных параметров
sigl	Совокупность допустимых входных сигналов
decll	Содержащийся список определений
body	Тело состояния процесса
tid	Завершающий идентификатор

Результат *См. transform-decllist*

Алгоритм

Строка 1 Пусть *name* обозначает имя процесса

Строка 2	Если конструкция "числа экземпляров" пропущена, то она является такой же, как пропущенное начальное число экземпляров и пропущенное максимальное число экземпляров
Строки 3-4	Если начальное число экземпляров не указывается, то оно равно 1. Если максимальное число экземпляров не указывается, тогда оно ничем не ограничено. Предопределенная константа *infinite* (бесконечный) обозначает "неограниченное" число (см. также F.1 раздел 5.8). Если одно из выражений определяется, то оно вычисляется и оно должно быть предопределенного сорта данных INTEGER, т.е. целочисленный *Qual* (*iqual*) дан как параметр *eval-simple-expr*, как единственный допустимый сорт из множества сортов.
Строка 5	Имя (идентификатор) процесса не должно быть квалифицировано (так как оно не является удаленным описанием)
Строка 7	Идентификатор, заканчивающий определение процесса (*tid*), должен быть эквивалентен имени (идентификатору) процесса, если оно определено
Строка 9	Максимальное число экземпляров больше или равно начальному числу экземпляров.
Строка 11	Начальное число экземпляров не должно быть меньше нуля.
Строка 13	Максимальное число экземпляров не должно быть равно нулю.
Строки 15-16	Инициализировать вход IMPLIED для того, чтобы не содержались неявные переменные, и инициализировать OUTSIGNALS для того, чтобы не содержались выходные сигналы. Эти два множества заполняются при преобразовании тела процесса.
Строки 17-18	Модернизировать SCOPEUNIT, чтобы указать уровень процесса и построить начальный *Data-type-definition*$_1$ процесса.
Строка 19	Преобразовать формальные параметры процесса.
Строка 20	Извлечь полную совокупность допустимых входных сигналов.
Строка 21	Преобразовать содержащиеся определения
Строка 22	Имена формальных параметров должны быть отличны от переменных и имен синонимов, определенных в процессе (т.е. они должны иметь различные *Quals*).
Строка 25	Преобразовать тело процесса и возвратить список определений AS_1 из содержащихся сервисов (если они есть), граф процесса AS_1, *Dict*, где используются только вход IMPLIED и вход OUTSIGNALS, и связи между маршрутами сигналов и маршрутами сигналов сервиса (*connectmap* пуст, если в процессе отсутствуют сервисы).
Строки 29-30	Не должно существовать двух выходных сигналов, для которых первый определяется во втором. То же самое относится к входным сигналам
Строка 32	Построить неявные определения переменных AS_1.
Строки 33-38	Возвратить определение процесса AS_1 (as_1*tree*), использование *Dict*, содержащее дескриптор процесса (*delem*), дескрипторы формальных параметров (*dict* ') и дескрипторы объектов, определенных внутри процесса (*ddict*).

transform-validinputset (*sigset*) (*dict*) ≜ (3.4.3.2)

```
1   (let qual = dict(SCOPEUNIT) in
2    let mk-BlockD(,, explicit, ) = dict(get-sur(process-level(qual))) in
3    let routesigset =
4       if ¬ explicit then
5         {}
6         else
7         {squal | (∃mk-SignalrouteD(p1, p2, s1, s2) ∈ rng dict)
8                 ((p1 = qual ∧ squal ∈ s2) ∨ (p2 = qual ∧ squal ∈ s1))} in
9   let squalset =
10      if sigset = nil then
11        {}
12        else
13        (let mk-Inputset0(sigl) = sigset in
14         if sigl = nil then {} else transform-signallist(sigl)(dict)) in
15  let implicit =
16      if is-ServiceD(dict(qual)) then
17        {}
18        else
19        (let (expimpinput,) = import-export-signals(dict) in
20         expimpinput) in
21  let timers = {tqual ∈ dom dict | is-TimerD(dict(tqual)) ∧ is-local(tqual)(dict)} in
22  let locals = {tqual ∈ dom dict | is-SignalD(dict(tqual)) ∧ is-local(tqual)(dict)) in
23  if squalset ∩ timers = {} ∧ locals ⊆ squalset then
24    routesigset ∪ squalset ∪ implicit ∪ timers
25    else
26    exit ("§2.5.2: Допустимый входной сигнал должен содержать все локальные сигналы и не должен содержать таймеры"))
```

запись : $[Signallist_0] \rightarrow Dict \rightarrow Signalqual$-**set**

Цель Извлечь полную совокупность допустимых входных сигналов для процесса или для сервиса

Параметры

sigset Необязательный список идентификаторов сигналов AS_0

Алгоритм

Строка 1 Пусть *qual* обозначает *Qual* сервиса или процесса

Строка 2 *explicit* истинно (true), если явные маршруты сигналов специфицируются для блока окружения.

Строки 3-8 Пусть *routeset* обозначает сигналы, содержащиеся в маршруте сигнала (неявном или явном), ведущие в процесс или сервис. Множество является пустым, если никакие явные маршруты сигнала не специфицируются для блока окружения.

Строки 9-14 Пусть *squalset* обозначает сигналы, содержащиеся в допустимой совокупности входных сигналов, построенной для процесса или сервиса.

Строки 15-19 Если единица подразделения не является сервисом, тогда извлечь *Quals* неявных сигналов, идущих в единицу подразделения.

Строка 21 Пусть *timers* обозначают сигналы таймера, определенные в процессе или сервисе.

Строка 22 Пусть *locals* обозначают сигналы, определенные в процессе или сервисе

Строка 23 Допустимая совокупность входных сигналов процесса или сервиса не должна включать в себя никаких сигналов таймера, она должна содержать все локально определенные сигналы.

Строка 24 Полная допустимая совокупность входных сигналов состоит из сигналов в маршрутах сигналов, сигналов в собранной допустимой совокупности входных сигналов, неявные сигналы и таймеры.

transform-processparm(*pl*)(*dict*) ≜ (3.4.3.3)

```
1   (if pl = ⟨⟩ then
2      (⟨⟩,⟨⟩,[])
3    else
4      (let mk-Parm0(nml,tid) = hd pl in
5       let (as1list, tp', dp') = transform-processparm(tl pl) (dict) in
6       let as0 vardef = mk-Vardefelem0(nml, tid, nil) in
7       let (,d) = transform-vardef(mk-Vardef0(nil,nil, ⟨as0 vardef⟩))(dict) in
8       let tq = get-visible-qual(tid, TYPE)(dict) in
9       let tq' = get-parent(tq)(dict) in
10      let tql = ⟨tq' | 1 ≤ i ≤ len nml⟩ in
11      let as1id = make-as1-identifier (tq')(dict) in
12      let as1 nml = ⟨name-to-name1(nml[i]) | 1 ≤ i ≤ len nml) in
13      if card elems as1 nml ≠ len as1 nml then
14         exit("§2.2.2:Два определения в одной единице подразделения используют одно и то же имя")
15       else
16         (let as1 tree = mk-Process-format-parameter1(as1 nml, as1 id) in
17          (⟨as1 tree⟩ ⌢ as1 list,tql ⌢ tp', d + d')))))
```

запись : $Parm_0^* \rightarrow Dict \rightarrow Process\text{-}formal\text{-}prameter_1^*\ Sortqual^*\ Dict$

Цель Преобразовать список формальных параметров процесса в AS_1

Параметры

pl Список формальных параметров процесса AS_0.

Результат Формальные параметры AS_1, соответствующий список сортов, который используется для проверки фактических параметров при создании вершины запроса, и требование *Dict*, содержащее дескрипторы формальных параметров.

Алгоритм Функция рекурсивно проходит по списку формальных параметров.

Строка 1 Если проход закончен, вернуть пустые списки или пустую таблицу

Строка 2 Пусть *nml* обозначает список имен первого элемента в списке, и пусть *tid* обозначает их сорт

Строка 5 Преобразовать оставшиеся формальные параметры

Строки 6 Построить определение переменной AS_0 для полученных параметров и

Строка 7 Преобразовать это описание переменной

Строка 8 Построить *Qual* сорта переменной

Строка 9 Извлечь "родителя" в случае подтипа

Строка 10 Построить список (*tql*) сорта *Quals*, соответствующего списку переменных.

Строка 11 Построить идентификатор AS_1 (as_1id) частичного родительского типа (*tq'*) для этого сорта списка переменных (*nml*)

Строка 12 преобразовать список имен в список имен AS_1

Строка 13 Имена в списке должны различаться

Строка 16 Построить определение формальных параметров в AS_1 для обработанных параметров (*nml*)

Строка 17 Возвратить определение формальных параметров AS_1 вместе с оставшимися формальными параметрами, возвратить ассоциированный список сорта и возвратить их требования *Dict*

3.4.4 Определения сигналов

transform-signaldef(mk-$Sigdef_0$(*elemlist*)(*dict*) ≜ (3.4.4.1)

```
1   (let mk-Sigelem0(nm, tidl, refinement) = hd elemlist in
2    let dlev = dict + [SCOPEUNIT ↦ dict(SCOPEUNIT) ⇀ ⟨(SIGNAL, nm)⟩] in
3    let (as1 refinement, subsig1, subsig2, d) =
4        if refinement = nil then
5          (nil, {}, {},[])
6         else
7          (let mk-Refinement0(subsiglist) = refinement in
8           let (as1set, s1, s2, d') = transform-refinement(subsiglist)(dlev) in
9           (mk-Signal-refinement1(as1set), s1, s2, d')) in
10   let quall = ⟨get-visible-qual(tidl[i], TYPE)(dict) | 1 ≤ i ≤ len tidl⟩ in
11   let d' = [dict(SCOPEUNIT) ⇀ ⟨(SIGNAL, nm)⟩ ↦ mk-SignalD(quall, subsig1, subsig2)] in
12   let (as1 set', d") =
13       if tl elemlist = ⟨⟩
14         then ({},[])
15         else transform-signaldef(mk-Sigdef0(tl elemlist))(dict) in
16   if dom d' ∩ dom d" ≠ {} then
17       exit ("§2.2.2: Два определения в одной единице подразделения используют одно и то же имя")
18     else
19       (let as1 idlist = ⟨make-as1-identifier(quall[i])(dict) | i ∈ ind quall⟩ in
20        let as1 tree = mk-Signal-definition1(name-to-name1(nm), as1 idlist, as1 refinement) in
21         ({as1 tree} ∪ as1 set', d+ d'+ d")))
```

запись : $Sigdef_0 \rightarrow Dict \rightarrow Signal\text{-}definition_1$-set *Dict*

Цель Преобразовать определение сигналов в AS_1

Параметры Составное определение сигналов AS_0, представляющее список (*elemlist*) элементарных определений сигналов.

Результат См. *transform-decllist*

Алгоритм

Строка 1	Произвести декомпозицию первого определения сигналов в составном определении сигналов AS_0
Строка 2	Модифицировать уровень для того, чтобы указать структуру сигналов.
Строки 3-9	Преобразовать обработанную часть определения сигналов и возвратить содержащиеся определения AS_1 (*as*$_1$ *refinement*), подсигналы, идущие в том же направлении, что и определенный сигнал (*subsig*1), подсигналы, идущие в обратном направлении определенному сигналу (*subsig*2), и требование Dict из новых определений (*d*).
Строка 10	Построить *Quals* сорта из специфицированного списка идентификаторов сорта AS_0.
Строка 11	Построить требование *Dict* к сигналу.
Строки 12-15	Преобразовать остальные определения сигналов в составное определение сигналов.
Строка 16	Никакие два сигнала в составном определении не могут иметь одинаковые имена.
Строка 19	Построить список идентификаторов сорта AS_1.
Строки 20-21	Возвратить множество определений сигналов AS_1 и требования *Dict* из сигналов.

transform-refinement(*decllist*)(*dict*) ≜ (3.4.4.2)

```
1   (if decllist = ⟨⟩ then
2      ({}, {}, {}, [])
3    else
4    (let (as1 declrest, sig1rest, sig2rest, drest) = transform-refinement(tl decllist)(dict) in
5     let mk-Subsignal0(reverse, sigdef) = hd decllist in
6     let (as1decl, d) = transform-signaldef(sigdef)(dict) in
7     if dom d ∩ dom drest ≠ {} then
8       exit ("§2.2.2: Два определения в одной единице подразделения и одном классе объектов определяют одно имя")
9      else
10      (let as1tot = as1declrest ∪ {mk-Subsignal-definition1(reverse, decl) | decl ∈ as1decl} in
11       if reverse = nil then
12        (as1tot, sig1rest ∪ dom d, sig2rest, d + drest)
13        else
14        (as1tot, sig1rest, sig2rest + dom d, d + drest))))
```

запись : *Subsignal*$_0$* → *Dict* → *Subsignal-definition*$_1$-set *Subsignal*-set *Subsignal*-set *Dict*

Цель Преобразовать подсигналы при обработке сигналов в определения подсигналов AS_1.

Параметры

decllist Список определений подсигналов AS_0

Результат Совокупность определений подсигналов AS_1, множество *Quals* сигналов, содержащее сигналы, идущие в том же направлении, что и родительский сигнал, множество *Quals* сигналов, содержащее сигналы, идущие в направлении, обратном родительскому сигналу, (эти два множества функцией *transform-signaldef* помещаются в дескриптор родительского сигнала), и требование *Dict* для подсигналов

Алгоритм

Строка 1 Если просмотр закончен, вернуть без требований.
Строка 4 Преобразовать остальные подсигналы.
Строка 5 Произвести декомпозицию первого подсигнала в списке.
Строка 6 Преобразовать подсигнал так, как будто он является обычным сигналом.
Строка 7 Проверить, чтобы определения подсигналов были уникальными.
Строка 10 Построить совокупность определений подсигналов AS_1 из определений сигналов AS_1 и
Строки 11-14 Возвратить эту совокупность, и, если было специфицировано REVERSE, то добавить *Quals* сигналов (взятые из домена *d*) в первый набор *Signalqual*, иначе добавить их во второй набор *Signalqual*. Также возвратить требования dict для подсигналов (*d*) и требование для остальных подсигналов (*drest*).

3.4.5 Определения процедур

transform-procedureddef(mk-*Procdef*$_0$(*pid*, *pl*, *decll*, *body*, *tid*)) (*dict*) ≜ (3.4.5.1)

```
1   (let mk-Id0(q, name) = pid in
2    if q = ⟨⟩ ∧ tid ∈ {pid, nil} then
3      (let name' = if is-ServiceD(dict(dict(SCOPEUNIT))) then
4                     create-unique-name ()
5                    else
6                     name in
7      let nqual = dict(SCOPEUNIT) ⌢ ⟨(PROCEDURE, name')⟩ in
8      let d = [SCOPEUNIT ↦ dict(SCOPEUNIT) ⌢ ⟨(PROCEDURE, name)⟩,
```

```
9              DATATYPEDEF ↦ initialdatadef(dict)] in
10   let (as₁ list, pd, dict') = transform-procedureparml(pl) (dict + d),
11      (as₁ declset, ddict) = transform-decllist(decll)(dict + d),
12      (mk-Procedure-graph₁ (sta, stateset), bdict) = transform-body(body)(dict + d) in
13   if dom dict' ∩ dom ddict ≠ {} then
14      exit("§2.2.2: Имена формальных параметров должны быть отличны от имен переменных")
15    else
16      if (∃squal ∈ dom dict)(is-ServiceD(dict(squal)) ∧
17                          get-sur(squal) = process-or-service-level(dict(SCOPEUNIT))) ∧
18         stateset ≠ {}
19         then exit("§4.10.2: Процедура в декомпозиции или сервисе содержит значения состояния или выражения импорта")
20         else (let as₁tree =
21                 make-as₁tree(PROCEDURE, name', as₁declset,
22                              as₁list, mk-Procedure-graph₁ (sta, stateset), nil)(ddict) in
23              let delem = [d(SCOPEUNIT) ↦ mk-ProcedureD(pd, ngual)] + dict' + ddict in
24              ({ as₁tree}, delem + bdict)))
25   else
26    (q ≠ ⟨⟩
27      → exit ("§2.4.1: Определяющие имена могут быть только квалифицированы в удаленных определениях"),
28     T→ exit("§2.2.2: Завершающее имя в определении процедуры отличается от определяющего имени")))
```

запись : $Procdef_0 \rightarrow Dict \rightarrow Procedure\text{-}definitions_1\text{-set}\ Dict$

Цель Преобразовать определение процедуры в AS_1

Параметры Определение процедуры, содержащее

pid	Идентификатор процедуры (должен содержать только имя)
pl	Список формальных параметров
decll	Список определений
body	Тело состояния
tid	Идентификатор (имя), завершающее определение

Результат Набор определений AS_1, состоящий из только одного определения и требования *dict* для процедуры и для содержащихся описаний

Алгоритм

Строка 1	Пусть *name* обозначает имя процедуры
Строка 2	Имя процедуры (идентификатор) не должно быть квалифицировано (так как оно не удаленное определение), и имя (идентификатор), заканчивающее определение процедуры должно быть равным имени процедуры, если оно специфицировано.
Строка 3	Создать уникальное имя для определения процедуры, если оно встречается в сервисе.
Строка 7	Создать переменную *Qual*, которая будет использоваться в AS_1
Строки 8-9	Модифицировать SCOPEUNIT, чтобы обозначить единицу подразделения процедуры, и создать начальный $Data\text{-}type\text{-}definition_1$ для процедуры.
Строка 10	Преобразовать формальные параметры процедуры для того, чтобы получить параметры AS_1 (as_1list), их дескрипторы сорта (будут использованы при вызове процедуры) и их требование *Dict*.
Строка 11	Преобразовать содержащиеся определения
Строка 12	Преобразовать тело процедуры и возвратить граф процедуры AS_1 и *Dict*, в котором используются только вход IMPLIED и вход OUTSIGNALS.
Строка 13	Никакой формальный параметр не может иметь то же имя, что и переменная или синоним, определенные в процедуре
Строки 16-19	Если процедура, содержащая состояния или значения импорта, определена в процессе или в процедуре в процессе, тогда процесс не должен содержать сервисы.

Строки 20-24 Возвратить определение процедуры AS_1 (*as_1tree*) и требование DICT, состоящее из дескриптора процедуры, формальных параметров дескрипторов и дескрипторов объектов, определенных в процедуре (*bdict*).

transform-procedureparml(*pl*)(*dict*) ≜ (3.4.5.2)

```
1   (if pl = ⟨⟩ then
2      (⟨⟩,⟨⟩,[])
3     else
4      (let (as1elems, tpelems, delems) =
5          cases hd pl :
6            (mk-Inoutparm0(varlist, tid)
7              →transform-varparm(true, varlist, tid)(dict),
8            mk-Inparm0(varlist, tid)
9              →transform-varparm(false, varlist, tid)(dict)) in
10     let (as1list, tplist, d) = transform-procedureparml(tl pl)(dict) in
11     if dom delems ∩ dom d ≠ {} then
12        exit("§2.2.2: Два определения в одной единице подразделения используют одно и то же имя")
13      else
14        (as1 elems ⌢ as1 list, tpelems ⌢ tplist, delems + d)))
```

запись : *$Procparm_0$* → Dict → $Procedure\text{-}formal\text{-}parameter_1$* FormparmD* Dict*

Цель Преобразовать список формальных параметров процедуры в AS_1

Параметры

pl Список формальных параметров AS_0.

Результат Формальные параметры AS_1, дескрипторы (*FormparmD**), которые помещаются в дескриптор *ProcedureD* и используются для проверки типа фактических параметров и, наконец, требования *Dict* для параметров.

Алгоритм

Строка 1 Если просмотрен весь список, возвратить пустые списки и пустой *Dict*

Строки 4-9 Преобразовать первый параметр, который является либо параметром переменной INOUT (строка 6), либо параметром переменной IN (строка 8).

Строка 10 Преобразовать оставшиеся в списке формальные параметры

Строка 11 Имена, представленные в *pl*, должны быть различны

Строка 14 Возвратить объекты, составленные из первого формального параметра и остальных.

transform-varparm(*isout*, *varlist*, *tid*) (*dict*)' ≜ (3.4.5.3)

```
1   if card elems varlist ≠ len varlist then
2      exit("§2.2.2: Два определения в одной единице подразделения используют одно и то же имя")
3    else
4      (let tqual = get-visible-qual(tid, TYPE)(dict) in
5       let tqual' = get-parent(tqual)(dict) in
6       let as1varlist = ⟨name-to-name1(varlist[i]) | 1 ≤ i ≤ len varlist⟩ in
7       let as1tree = if isout then
8                        mk-Inout-parameter1(as1varlist, make-as1-identifier(tqual)(dict)),
9                       else
10                       mk-In-parameter1(as1varlist, make-as1-identifier(tqual)(dict)),
11        parmdescrlist = if isout then
12                          ⟨mk-InoutDescr(tqual) | 1 ≤ i ≤ len varlist⟩
13                          else
14                          ⟨mk-InDescr(tqual') | 1 ≤ i ≤ len varlist⟩ in
15     let nmlist =
16        ⟨if is-ServiceD(dict(dict(SCOPEUNIT)))
17          then create-unique-name()
18          else varlist[i] | 1 ≤ i ≤ len varlist⟩,
19        newquallist = ⟨dict(SCOPEUNIT) ⌢ ⟨(VALUE, nmlist[i])⟩ | 1 ≤ i < len varlist⟩ in
20     let vquall = ⟨dict(SCOPEUNIT) ⌢ ⟨(VALUE, varlist[i])⟩ | 1 ≤ i ≤ len varlist⟩,
21        delem = [vquall[i] ↦ mk- VarD(tqual, nil, nil, nil, newquallist[i]) | 1 ≤ i ≤ len varlist] in
22     (⟨as1tree⟩, parmdescrlist, delem))
```

запись : *Bool Name*$_0^*$ *Id*$_0$ → *Dict* → *Procedure-formal-parameter*$_1^+$ *FormparmD*$^+$ *Dict*

Цель Преобразовать формальный параметр в AS_1

Параметры

isout	Истина, если формальный параметр является параметром INOUT
varlist	Имена формальных параметров
tid	Их сорт.

Результат См. *transform-procedureparml*

Алгоритм

Строка 1	Имена формальных параметров должны быть различны
Строка 4	Извлечь *Qual* сорта параметров.
Строка 5	Извлечь родительский частичный тип сорта.
Строка 6	Преобразовать список имен в имена AS_1
Строки 7-10	Построить формальный параметр AS_1
Строки 11-14	Построить дескриптор сорта фактических параметров.
Строка 21	Создать требование dict для формального параметра. Содержащиеся дескрипторы есть дескрипторы *VarD*, (подобно обычным переменным), так как по отношению к статическим свойствам формальные параметры переменных действуют как обычные переменные.

3.4.6 Генераторы сорта

transform-sortgenerator (**mk**-*Sortgenerator*$_0$(*nm*, *parml*, *geninstl*, *prop*, *tnm*))(*dict*) ≜ (3.4.6.1)

```
1  (if tnm ∈ {nm,nil} ∧ gen-formparm-unique (parml, {nm}, {}, {}) then
2      (let gqual = dict(SCOPEUNIT) ⁀ ⟨(GENERATOR, nm)⟩ in
3      let prop' = transform-geninst(nm, geninstl, prop)(dict + [gqual ↦ mk-ErrorD()]) in
4      let descr = mk-GeneratorD(parml, prop') in
5      ({}, [gqual ↦ descr])
6    else
7      exit ("§5.4.1.12: Завершающее имя в определении генератора отличается от определяющего имени")
```

запись: *Sortgenerator*$_0$ → *Dict* → *Decl*$_1$-**set** *Dict*

Цель Построить использование *Dict*, соответствующее генератору данных сорта. Никакие объекты AS$_1$ не возвращаются.

Параметры Генератор данных сорта AS$_0$, содержащий

nm	Имя генератора
parml	Формальные параметры
geninstl	Список содержащихся экземпляров генератора
prop	Обычные свойства
tnm	Имя, заканчивающее определение генератора

Результат Для удобства функция возвращает набор определений AS$_1$, хотя список всегда пустой (см. *transform-decllist*). Также возвращается требование *Dict* генератора сорта данных.

Алгоритм

Строка 1	Имя, завершающее определение, должно соответствовать имени генератора, если оно специфицировано, и формальные параметры в параметрах различных классов должны быть уникальными.
Строка 2	Построить *Qual* для генератора.
Строка 3	Преобразовать экземпляры генератора, содержащиеся в определении генератора. *Qual* генератора (*qualg*) обозначается *ErrorD* так, что может быть обнаружена рекурсия.
Строки 4-7	Возвратить требование *Dict*, содержащее формальные параметры генератора (*parml*) и тело генератора (*prop*). Корректность построения применяется к телу после того, как оно заменяется конструкцией экземпляра генератора (не в этом месте).

gen-formparm-unique (*parml*, *sset*, *lset*, *oset*) ≜ (3.4.6.2)

```
1  if parml = ⟨⟩ then
2      true
3    else
4      cases hd parml:
5      (mk-Sortparm0(nmlist)
6        → card elems nmlist = len nmlist ∧ elems nmlist ∩ sset = {} ∧
7          gen-formparm-unique(tl parml, sset ∪ elems nmlist, lset, oset),
8      mk- Termparm0(nmlist),
9      mk-Litparm0(nmlist)
10      → card elems nmlist = len nmlist ∧ elems nmlist ∩ lset = {} ∧
11         gen-formparm-unique(tl parml, sset, lset ∪ elems nmlist, oset),
12     mk-Opparm0(nmlist)
13      → card elems nmlist = len nmlist ∧ elems nmlist ∩ oset = {} ∧
14         gen-formparm-unique(tl parml, sset, lset, oset ∪ elems nmlist))
```

запись: *Genparm*$_0$ *Name*$_0$-**set** *Name*$_0$-**set** *Name*$_0$-**set*** → *Bool*

Цель Проверить, что формальные параметры для генератора уникальны.

Параметры

parml Формальные параметры

sset, lset, oset Набор параметров сорта, параметров литералов или параметров термов и параметров операторов, соответственно рассмотренных до сих пор (функция рекурсивна).

Результат Истина (true), если результат правильный

Алгоритм

Строки 5-7 Для параметра сорта содержащиеся имена должны быть уникальными и ни одно из имен не может встретиться в другом параметре сорта, остальные формальные параметры генератора должны также быть уникальными.

Строки 8-11 Для параметра терма или параметра литерала содержащиеся имена должны быть уникальными, ни одно из имен не может встретиться в другом параметре терма или параметре литерала, остальные формальные параметры генератора также должны быть уникальными.

Строки 12-14 Для параметра оператора содержащиеся имена должны быть уникальными, ни одно из имен не может встретиться в другом параметре оператора, остальные формальные параметры генератора должны также быть уникальными.

3.4.7 Определения сорта

transform-partial-typedef(mk-*Partialtypedef*$_0$(*nm*, *exprop*, *prop*, *vlist*, *tnm*))(*dict*) ≜ (3.4.7.1)

```
1   (if tnm ∈ {nm, nil} then
2      (let tqual = dict(SCOPEUNIT) ⇝ ⟨(TYPE, nm)⟩ in
3      let (implicitdecl, delem) = make-implicit-sigdecls(tqual)(dict) in
4      if (∃opqual ∈ dom dict)((is-OperatorD(dict(opqual)) ∨ is-LiteralD(dict(opqual))) ∧
5                                s-Result(dict(opqual)) = tqual ∧
6                                get-sur(get-sur(opqual)) = dict(SCOPEUNIT)) then
7         if vlist = nil then
8           (let dict' = cases extprop:
9                          (nil
10                          → transform-sortdef(nm, prop, nil)(dict),
11                         mk-Struc0()
12                          → transform-struc(nm, extprop, prop)(dict),
13                         mk-Inherited0(,,)
14                          → transform-inherited(nm, extprop, prop)(dict),
15                         mk-Geninstlist0(instl)
16                          → (let prop' = transform-geninst(nm, instl, prop)(dict) in
17                              transform-sortdef(nm, prop', nil)(dict))) in
18          (implicitdecl, dict'))
19        else
20          (let unm = create-unique-name() in
21          let (,dnew) =
22              transform-partial-typedef(mk-Partialtypedef0(unm, extprop, prop, nil, unm))(dict),
23              id = mk-Id0(dict(SCOPEUNIT), unm) in
24          let (as1syn, dsyn) =
25              transform-syntype(mk-Syntypedef0(nm, id, vlist, nil, tnm))(dict) in
26          (implicitdecl ∪ as1syn, delem + dnew + dsyn))
27      else
28        exit("§5.2.1: Нет операторов, которые возвращают значение этого сорта")
29    else
30     exit("§5.2.1: Завершающее имя в описании частичного типа отлично от определяющего имени")
```

запись: *Partialtypedef*$_0$ → *Dict* → *Decl*$_1$-set *Dict*

Цель Преобразовать определение частичного типа данных в AS_1

Параметры Определение данных сорта, содержащее

nm	Имя сорта
exprop	Расширенные свойства
prop	Свойства (литералы, операторы, равенства и т.д.)
vlist	Список значений в случае подтипа
tnn	Имя, завершающее определение

Результат См. *transform-declist*

Алгоритм

Строка 1	Имя, завершающее определение данных сорта, должно соответствовать имени сорта данных, если оно специфицировано.
Строка 2	Построить *Qual*, представляющий идентификатор сорта.
Строка 3	Построить определения AS_1, которые определяют неявные сигналы, связанные со всеми переменными IMPORT-EXPORT сорта, определенного ранее.
Строка 4	Должен существовать оператор или литерал, который
Строка 5	получает этот сорт, как результирующий сорт, и который
Строка 6	определен в единице подразделения окружающей среды
Строка 7	Если конструкция CONSTANTS пропущена, то это определение вещественного сорта
Строка 8	Построить требование *Dict*, содержащее дескриптор сорта, дескрипторы оператора и литерала и измененные $Data\text{-}type\text{-}definition_1$. Определение сорта AS_0 одно из следующих
Строка 9	Определение (простого) сорта (без расширенных свойств).
Строка 11	Определение структуры или
Строка 13	Сорт, который основывается на наследовании, или
Строка 15	(Последовательность) экземпляр(ов) генератора. В этом случае экземпляры расширяются (строка 15) до того, как они преобразуются обычным способом (строка 17)
Строки 20-26	Если CONSTANTS специфицировано, то это определение подтипа, что означает, что
Строка 20	Генерируется уникальное неявное имя сорта и
Строка 21	$Partialtypedef_0$ без каких-либо CONSTANTS и с этим новым именем преобразованы и
Строка 24	$Syntypedef_0$ с CONSTANTS (*vallist*) и неявным сортом в качестве родителя преобразованы

make-implicit-sigdecls(*tqual*)(*dict*) ≜ (3.4.7.2)

1 (**let** *qset* = {(*tq*,) ∈ **dom** *dict* | *tq* = *tqual*} **in**

2 *transform-decllist*({*make-implicit-decl*(*tqual*, *dict*(*t*)) | *t* ∈ *qset*})(*dict*)

запись: *Sortqual* → *Dict* → $Decl_1$-**set** *Dict*

Цель Построить определения сигналов AS_1, связанные со всеми *NameclosureD*, которые содержат *tqual* сорта (см. определение *Exportmap*).

Алгоритм

Строка 1	Построить множество из таких *NameclosureD*, для которых *Sortqual* равно *tqual*, и
Строка 2	Построить список определений AS_0, построенных присоединением списков определений, связанных с каждым элементом в этом множестве, и преобразовать этот список в AS_1.

make-implicit-decl(*tqual*, **mk**-*SignalnamesD*(, *xqnm*, *xrnm*)) ≜ (3.4.7.3)

```
1   (let as0tid = as0-id(tqual) in
2   mk-Sigdef0(⟨mk-Sigelem0(xqnm, ⟨⟩, nil),
3              mk-Sigelem0(xrnm, ⟨as0tid⟩, nil)⟩))
```

запись: *Sortqual SignalnamesD* → $Sigdef_0$

Цель Создать определения сигнала AS_0, подключенного к EXPORT-IMPORT некоторого имени и некоторого сорта. (*tqual*).

Алгоритм

Строка 1	Построить идентификатор AS_0 сорта, переданного в сигнале xtREPLY
Строки 2-3	Построить определение сигнала, содержащее
Строка 2	Сигнал xtQUERY и
Строка 3	Сигнал xtREPLY.

transform-inherited(*nm*, **mk**-$Inherited_0$(*pid*, *lrenam*, *ops*), *prop*)(*dict*) ≜ (3.4.7.4)

```
1   (let pqual = get-inherited-parent(get-visible-qual(pid, TYPE)(dict))(dict) in
2   if is-recursive-sort(pqual, {})(dict) then
3      exit("§5.4.1.11: Сорт основан на себе самом")
4     else
5      (let pqual' = s-Newqual(dict(pqual)) in
6       let sortdict = transform-sortdef(nm, prop, pqual)(dict) in
7       let typedef = sortdict(DATATYPEDEF) in
8       let mk-Data-type-definition1(typename, union, sorts, sigs, eqs) = typedef in
9       let iqual be s.t. iqual ∈ dom sortdict ∧ is-SortD(sortdict(iqual)) in
10      let mk-SortD(eqs', pq, exp1, nqual) = sortdict(iqual) in
11      let (litmap, litd) = transform-literal-renaming(lrenam, iqual, pqual)(dict) in
12      let concaxioms1 = make-as1-concaxioms(dom litd)(dict) in
13      let (opmap, opd, opset) = transform-operator-renaming(ops, iqual, pqual)(dict) in
14      if dom sortdict ∩ dom litd ∩ dom opd ≠ {} then
15         exit ("§5.2.2: Оператор или литерал определены явно и согласно наследованию")
16        else
17         (let inhax = extract-inherited-axioms(litmap, opmap, nqual, pqual')(dict) in
18          let iid1 = make-as1-identifier(nqual)(dict) in
19          let litsig = {mk-Literal-signature1(nm1, iid1) | mk-Identifier1(, nm1) ∈ rng litmap} in
20          let datatypedef' =
21              mk-Data-type-definition1(typename, union, sorts,
22                                       sigs ∪ litsig ∪ opset, eqs ∪ inhax ∪ concaxioms1) in
23          let sortdescr = mk-SortD(eqs' ∪ inhax ∪ concaxioms1, pq, exp1, nqual) in
24          (sortdict + litd + opd + [iqual          ↦ sortdescr,
25                                    DATATYPEDEF ↦ datatypedef']))))
```

запись: $Name_0$ $Inherited_0$ $Properties_0$ → *Dict* → *Dict*

Цель Преобразовать определение частичного типа, которое основано на наследовании

Параметры

nm	Имя ранее определенного сорта
Inherited	Конструкция наследования AS_0
prop	Свойства AS_0 определены в конструкции ADDING

Результат *Dict*, где вход DATATYPEDEF был изменен, чтобы включить ранее определенный сорт, и где были добавлены дескрипторы для сорта и операторов.

Алгоритм

Строка 1	Извлечь *Qual*, обозначающий идентификатор родительского сорта.
Строки 2-3	Сорт не должен наследовать сам себя.
Строка 5	Извлечь уникальный *Qual* родителя.
Строка 6	Изменить вход DATATYPEDEF определенным ранее сортом и с его ADDING свойствами.
Строка 8	Произвести декомпозицию измененного *Data-type-definition*$_1$.
Строки 9-10	Извлечь (только что присоединенный) *Qual* определенного ранее сорта и произвести его декомпозицию.
Строка 11	Преобразовать переименованную часть литерала в *litmap* отображения и построить дескрипторы для имен литералов, представленных при переименовании литерала.
Строка 12	Построить аксиомы, подразумевающие литералы символьных строк, представленные при переименовании литерала.
Строка 13	Преобразовать часть переименования оператора в таблицу *optmap*, создать дескрипторы для новых имен операторов (*opd*), создать сигнатуры AS$_1$ для операторов (*opset*). Эта функция также работает с операторами, которые неявно наследуются (невидимые операторы)
Строки 14-15	все операторы и литералы, определенные в присоединенной части *sortdict*, и операторы (*opd*), и литералы (*litd*), определенные в части наследования, должны иметь различные сигнатуры.
Строка 17	Извлечь все аксиомы, использующие операторы, которые неявно наследуются.
Строка 18	Построить идентификатор AS$_1$ определенного ранее сорта.
Строка 19	Построить сигнатуры AS$_1$ переименованных литералов.
Строки 20-23	Включить свойства из части наследования в *Data-type-definition* и в дескриптор этого сорта.
Строка 24	Возвратить требование *Dict* из присоединенной части (*sortdict*) для переименованных литералов (*litd*), для переименованных операторов (*opd*), для дескрипторов сорта и измененного *Data-type-definition*$_1$.

is-recursive-sort(*qual*, *qset*)(*dict*) ≜ (3.4.7.5)

```
1   (if qual ∈ qset then
2      true
3    else
4      cases dict(qual):
5       (mk-SortD(,pa,,)
6          → if pa = nil then false else is-recursive-sort(pa, qset ∪ {qual})(dict),
7       mk-SyntypeD(pa,,,)
8          → is-recursive-sort(pa, qset ∪ {qual})(dict)))
```

запись : *Sortqual Sortqual*-**set** → *Dict* → *Bool*

Цель Проверить, что подтип или сорт наследования не базируются на самих себе. Функция рекурсивно проходит через *dict* до тех пор, пока не обнаружится дескриптор частичного типа, который не базируется на наследовании. (Т.е. до тех пор пока применяются ложные условия).

Параметры

qual	Обозначает обрабатываемый сорт
qset	Обозначает множество сортов, на которые уже были ссылки. Когда функция применяется в первый раз, это множество пусто.

Результат Истина, если сорт определен рекурсивно.

Алгоритм

Строка 1	Если на обрабатываемый сорт (*qual*) уже были ссылки, тогда **true** (истина), иначе
Строки 5-6	Если обрабатываемый сорт является частичным типом, неимеющим родителей, тогда этот сорт определен не рекурсивно, в противном случае присоединить частичный тип к набору сорта и продолжить работу с его родителем.
Строка 7	Присоединить обрабатываемый подтип к набору сорта и продолжить работу с его родителем.

transform-literal-renaming(*lrenam*, *qual*, *pqual*)(*dict*) ≜ (3.4.7.6)

```
1   (let plitset = {lq ∈ dom dict | is-LiteralD(dict(lq)) ∧ s-Sortqual(dict(lq)) = pqual} in
2     let defaultmap = [qual ↦ qual | qual ∈ plitset] in
3     let litmap = defaultmap + build-literal-renaming(lrenam, qual, plitset) in
4     ([make-as1-identifier(lq)(dict) ↦ make-as1-identifier(litmap(lq))(dict) | lq ∈ dom litmap],
5      [lq ↦ mk-LiteralD(qual) | lq ∈ rng litmap]))
```

запись: *Literalrenaming*$_0$ *Sortqual Sortqual* → *Dict* →
(*Literal-operator-identifier*$_1$ ⇸ *Literal-operator-identifier*$_1$) *Dict*

Цель Создать таблицу отображения между литералами родителя и новыми литералами, чтобы использовать ее, когда аксиомы наследования будут выделены (в *exstract-inherited-axioms*); собрать дескрипторы *Dict* для новых литералов

Параметры

lrenam	Переименованный литерал AS_0
qual	*Qual* определенного ранее сорта
pqual	*Qual* сорта, на котором базируется наследование.

Результат Созданная таблица и требование *Dict*, содержащие дескрипторы литералов

Алгоритм

Строка 1	Извлечь из *Dict* совокупность литералов, определенных для сорта родителя.
Строка 2	Построить отображение, используемое по умолчанию, т.е. соотношение, которое применяется для литералов, не упомянутых в части переименования.
Строка 3	Создать таблицу переименования литералов (литералы, содержащиеся в ней, являются *Quals*) посредством перезаписи таблицы, используемой по умолчанию.
Строка 4	Возвратить таблицу переименования литералов после того, как проведено конвертирование *Quals* в идентификаторы AS_1, и возвратить
Строка 5	Все дескрипторы литералов (*LiteralDs*), которые содержат *Qual* их сорта

build-literal-renaming(*lrenam, qual, pqual, plitset*) ≜ (3.4.7.7)

```
1   if lrenam = ⟨⟩ then
2     []
3    else
4     (let mk-Literalpair0(nlit, olit) = hd lrenam in
5      let nqual = qual ⌢ ⟨(LITERAL, nlit)⟩,
6          oqual = pqual ⌢ ⟨(LITERAL, olit)⟩ in
7      if oqual ∈ plitset then
8        (let restmap = build-literal-renaming(tl lrenam, qual, pqual, plitset) in
9         if nqual ∈ rng restmap then
10          exit("§5.4.1.11: Литерал определен дважды при переименовании")
11        else
12          if oqual ∈ dom restmap then
13            exit("§5.4.1.11: Литерал переименован дважды")
14          else
15            restmap + [oqual ↦ nqual])
16       else
17        exit("§5.4.1.11: Литерал в переименованном литерале не определен в родительском сорте")
```

запись: *Literalrenaming*$_0$ *Sortqual Sortqual Qualset* -set → (*Qual* ⇸ *Qual*)

Цель Создать из переименованной части AS_1 отображение из *Quals* в *Quals* новых имен литералов.

Параметры

lrenam	Переименованный литерал AS_0
qual	*Qual* ранее определенного сорта
pqual	*Qual* сорта, на котором базируется наследование
plitset	Множество *Quals*, содержащих все литералы, определенные для родительского сорта.

Результат Созданное отображение литералов

Алгоритм

Строка 1	Когда просмотр окончен, тогда возвратить нуль (функция рекурсивна)
Строка 4	Произвести декомпозицию следующей пары имен литералов при переименовании литерала.
Строки 5-6	Построить *Quals* новых имен литералов (*nqual*) и родительских (старых) имен литералов.
Строка 7	Правый литерал в паре *Literalpair*$_0$ должен быть литералом родительского сорта.
Строка 8	Создать отображение для оставшейся части переименования литералов.
Строки 9-12	Если левый литерал (строка 9) или правый литерал (строка 12) находятся в таблице для оставшихся литералов переименования, тогда он упоминается дважды в переименовании литерала
Строка 15	Иначе, возвратить таблицу литералов для оставшихся литералов переименования, где это использование было присоединено

transform-operator-renaming(*oprenam*, *iqual*, *pqual*)(*dict*) ≜ (3.4.7.8)

```
1   (let opset = {qual ∈ dom dict | is-OperatorD(dict(qual)) ∧
2                                   (let mk-OperatorD(argl, res,, ) = dict(qual) in
3                                   pqual ∈ elems argl ∨ pqual = res)} in
4   let explicitset = {qual ∈ opset | is-Explicit(dict(qual))} in
5   let allnameset = {name | (∃q ∈ explicitset)((, (name,, )) = q[len q])} in
6   let oprenam' = if oprenam = ALL then ⟨mk-Operatorpair0(nil,nm) | nm ∈ allnameset⟩ else oprenamin
7   let (opmap, opd) = build-operator-renaming(oprenam', iqual, pqual, explicitset)(dict) in
8   let (opmap', opd') = build-implicit-operators(iqual, pqual, opset \ explicitset)(dict) in
9   let (qmap, opdict) = (opmap + opmap', opd + opd') in
10  let idmap = [make-as1-identifier(qual)(dict) ↦ make-as1-identifier(qmap(qual))(dict) |
11              qual ∈ dom qmap] in
12  let as1typing = {make-as1-typing(op)(dict) | op ∈ dom opdict} in
13  (idmap, opdict, as1typing))
```

запись: (ALL | *Operatorrenaming*$_0$) *Sortqual Sortqual* → *Dict* →
(*Operator-identifier*$_1$ ⇸ *Operator-identifier*$_1$) *Dict Operator-signature*$_1$-**set**

Цель Построить из части оператора переименования AS_1 отображение идентификаторов родительских операторов в идентификаторы операторов ранее определенного сорта. Отображение включает идентификаторы операторов, которые неявно наследуются (оператор неявно унаследован, если он включает родительский сорт в свою сигнатуру)

Параметры

oprnam	Переименование оператора AS_0
iqual	*Qual* определенного ранее сорта
pqual	*Qual* родителя

Результат Построенное отображение операторов

Алгоритм

Строка 1	Создать множество *Quals* операторов из *Dict*, для которого сохраняется, что *Qual* родителя (*pqual*) появился в списке сорта аргументов (*argl*) или является результатом операции.
Строка 4	Создать подмножество *opset*, которое содержит операторы, которые явно определены (или унаследованы).
Строка 5	Создать множество имен операторов для явно определенных операторов.
Строка 6	Если ALL специфицировано, тогда пусть *oprenam'* обозначает *Operatorrenaming*$_0$, который включает все явно заданные операторы, иначе пусть *oprenam'* будет *oprenam*.
Строка 7	Создать отображение *Qual* операторов и дескрипторы *Dict* для явно заданных операторов.
Строка 8	Создать отображение *Qual* операторов и дескрипторы *Dict* для неявно заданных операторов.
Строка 9	Законченое отображение *Qual* операторов есть *qmap* и законченное требование *Dict* есть *opdict*.
Строка 10	Превратить отображение *Qual* операторов в отображение идентификаторов операторов.
Строка 12	Для каждого оператора в требовании *Dict* создать сигнатуру оператора AS_1.
Строка 13	Возвратить отображение идентификаторов оператора, требования *Dict* и сигнатуры AS_1 для операторов.

make-as$_1$-typing(qual)(dict) ≜ (3.4.7.9)

```
1   (let mk-OperatorD(sl, res, qual', ) = dict(qual) in
2    let (, (nm,, )) = qual [len qual'] in
3    let sl1 = ⟨make-as1-identifier(sl[i])(dict) | 1 ≤ i ≤ len sl⟩,
4       res1 = make-as1-identifier(res)(dict) in
5    mk-Operator-signature1 (name-to-name1 (nm), sl1, res1))
```

запись: *OperatorD* → *Dict* → *Operator-signature$_1$*

Цель Из оператора *Qual* создать сигнатуру оператора AS$_1$

Параметры

qual Оператор *Qual*

Результат Созданная сигнатура оператора

Алгоритм

Строка 1 Произвести декомпозицию дескриптора оператора. *qual'* обозначает *Newqual*, содержащий имя, которое будет использоваться в AS$_1$

Строка 2 Извлечь имя оператора при помощи декомпозиции последнего элемента *Qual*. Для оператора последний элемент всегда есть *Operatorqualelem* (см. домен).

Строка 3 Создать список идентификаторов AS$_1$, обозначающий сорты аргументов.

Строка 4 Создать идентификатор AS$_1$, обозначающий результирующий сорт.

Строка 5 Возвратить составную сигнатуру оператора

build-operator-renaming(renlist, qual, pqual, opset)(dict) ≜ (3.4.7.10)

```
1   (if renlist = () then
2      ([], [])
3     else
4      (let mk-Operatorpair0(nop, oop) = hd renlist in
5       let oset = {oqual ∈ opset | (let (, (nm,, )) = oqual[len oqual] in
6                                    get-sur(oqual) = pqual ∧
7                                    nm = oop)} in
8       if oset = {} ∨ (is-Name0 (oop) ∧ (let mk-Name0(, etc) = oop in
9                                          exc ≠ nil)) then
10        if oset = {} then
11          exit("§5.4.1.11: Оператор при переименовании оператора не определен в родительском сорте")
12         else
13          exit("§5.4.1.11: Оператор с восклицательным знаком упоминается при переименовании оператора")
14        else
15         (let (maprest, drest) =
16              build-operator-renaming(tl renlist, qual, pqual, opset \ oset)(dict) in
17          let (map, d) = rename-operator(qual, pqual, oset, nop)(dict) in
18          if dom d ∩ dom drest = {} then
19            (map + maprest, d + drest)
20           else
21            exit("§5.4.1.11: Оператор переименован дважды")
```

запись: *Operatorrenaming$_0$ Sortqual Sortqual Qual*-**set** → *Dict* → (*Qual* ⇸ *Qual*) *Dict*

Цель Посредством переименования операторов построить отображение родительского оператора *Quals* в оператор *Quals* уже определенного сорта. Также построить требование *Dict* для новых операторов.

Параметры

renlist Список переименованных операторов AS$_0$

qual	*Qual* ранее определенного сорта
pqual	*Qual* родительского сорта
opset	Множество, содержащее все *Quals* явно определенных операторов

Результат Построенное отображение *Qual* операторов и требований Dict

Алгоритм

Строка 1	Если просмотр закончен, ничего не возвращать (эта функция является рекурсивной)
Строка 4	Произвести декомпозицию следующей пары переименований в списке переименований
Строка 5	Извлечь множество *Quals* операторов, для которого единица подразделения среды места, где оно определяется, является *Qual* родителя, часть имени соответствует имени оператора родителя, упомянутого при переименовании.
Строки 8-11	В этом множестве должен существовать хотя бы один *Qual*.
Строки 8-13	Имя оператора родителя не должно содержать восклицательный знак.
Строка 15	Построить отображение операторов родителя и требования *Dict* для оставшейся части списка переименования.
Строка 17	Переименовать обрабатываемый оператор. *map* обозначает требования к отображению операторов, *d* обозначает требования к *Dict*.
Строки 18-19	Если *Quals* имени обрабатываемого оператора (могут существовать несколько *Quals* имени оператора поскольку родительское имя оператора, упомянутое при переименовании оператора, может обозначать несколько (дополнительных) операторов) не входят в требования *Dict* к оставшейся части списка переименований, то возвратить требования отображению операторов к обрабатываемому имени оператора (*map*) для оставшейся части списка и возвратить соответствующие требования *Dict*.

rename-operator(*iqual*, *pqual*, *oset*, *nop*)(*dict*) ≜ (3.4.7.11)

```
1   if oset = {} then
2      ([], [])
3    else
4      (let qual ∈ oset in
5       let (, (nm,, )) = qual[len qual] in
6       let mk-OperatorD(arglist, result,, ) = dict(qual) in
7       let arglist' = ⟨if get-parent(arglist[i])(dict) = pqual then iqual else arglist[i] | 1 ≤ i ≤ len arglist⟩ in
8       let parglist = ⟨get-parent(arglist'[i])(dict) | 1 ≤ i ≤ len arglist'⟩ in
9       let result' = if get-parent(result)(dict) = pqual then iqual else result in
10      let presult = get-parent(result')(dict) in
11      let (maprest, drest) = rename-operator(iqual, pqual, oset \ {qual}, nop)(dict) in
12      if result = result' ∧ arglist = arglist' then
13         (maprest, drest)
14       else
15         (let nm' = if nop = nil then nm else nop in
16         let newnm = create-unique-name() in
17         let opqual = iqual ⇝ ⟨(OPERATOR, (nm', parglist, presult))⟩ in
18         let newqual = iqual ⇝ ⟨(OPERATOR, (newnm, parglist, presult))⟩ in
19         (maprest + [iqual ↦ newqual], drest + [opqual ↦ mk-OperatorD(arglist', result', newqual, true)])))
```

запись: *Sortqual Sortqual Qual*-set *Newoperator*$_0$ → *Dict* → (*Qual* ⇸ *Qual*) *Dict*

Цель Построить отображение *Quals* родительских операторов в *Quals* операторов для данного имени оператора. Также построить требования *Dict* для операторов, обозначенных для этого имени

Параметры

iqual	*Qual* сорта наследования
pqual	*Qual* родительского сорта

oset	Множество *Quals* операторов, определенное в родительском сорте и имеющее имя, упомянутое в операторе переименования
nop	Новое имя оператора

Результат Созданное отображение и соответствующие требования Dict

Алгоритм

Строка 1	Если просмотр завершен, тогда возвратить нуль (функция рекурсивна)
Строки 4-5	Взять следующий *Qual* оператора из множества и извлечь имя оператора
Строка 6	Произвести декомпозицию дескриптора оператора для обрабатываемого родительского оператора
Строка 7	Для обрабатываемого оператора заменить все встречающиеся родительские сорта в списке сортов аргумента для обрабатываемого оператора на сорта наследования. Отметим, что функция *get-parent* ничего не должна делать с родителем сорта наследования. *get-parent* заменяет любые подтипы на их (родительские) сорты.
Строка 8	Переместить все подтипы из списка аргументов. Этот список имен используется в *Qual* для оператора.
Строки 9-10	Проделать то же самое для результирующего сорта
Строка 11	Пройти оставшиеся элементы множества *Quals* родительского оператора.
Строки 12-13	Если родительский *Quals* не используется в списке аргументов или в результате оператора, тогда возвратить отображения и требования *Dict* для оставшейся части списка, иначе
Строка 15	Если новое имя оператора не определено в $Operatorpair_0$, тогда оператор наследования имеет то же имя, что и родительский оператор.
Строка 16	Собрать уникальное имя, которое будет использоваться в *Newqual* для оператора (см.описание *OperatorD*)
Строка 17	Собрать *Qual*, который будет использоваться как вход *Dict* для оператора.
Строка 18	Собрать *Newqual* для оператора
Строка 19	Возвратить отображение *Qual* оператора для остальных операторов и для обрабатываемых операторов, возвратить *Dict* для требований.

(3.4.7.12)

build-implicit-operators(*iqual, pqual, opset*)(*dict*)≜

```
1   if opset = {} then
2     ([], [])
3   else
4     (let qual ∈ opset in
5       let (,(,arglist, result)) = qual[len qual] in
6       let arglist' = ⟨if get-parent(arglist[i])(dict) = pqual then iqual else arglist[i] | 1 ≤i≤ len arglist⟩ in
7       let result' = if get-parent(result)(dict) = pqual then iqual else result in
8       let (restmap, drest) = build-implicit-operators(iqual, pqual, opset \ {qual})(dict) in
9       if arglist = arglist' ∧ result = result' then
10        (restmap, drest)
11      else
12        (let newnm = create-unique-name() in
13          let opqual = iqual ⇝ ⟨(OPERATOR, (newnm, arglist ', result ' ))⟩ in
14          (restmap + [qual ↦ oqual], drest + [opqual ↦ mk-OperatorD(arglist',result',opqual,false)])))
```

запись: *Sortqual Sortqual Qual*-set → *Dict* → (*Qual* ⇸ *Qual*) *Dict*

Цель Построить отображение из *Quals* родительского оператора в *Quals* оператора для неявных операторов. Также построить требования *Dict* для операторов.

Параметры

iqual *Qual* для сорта наследования

pqual	*Qual* родительского сорта
oset	Множество *Quals* операторов, использующих родительский сорт

Результат Сформированная таблица отображений и соответствующие требования *Dict*

Алгоритм

Строка 1	Если просмотр закончен, функция ничего не возвращает (функция рекурсивная).
Строка 4	Взять *Qual* оператора в множестве и извлечь список аргументов сорта и результирующий сорт
Строки 6-7	Построить список аргументов сорта, где каждое встретившееся выражение родительского сорта заменено сортом наследования, проделать то же с результирующим сортом
Строка 8	Построить таблицу отображений *Qual* и требования *Dict* для всех оставшихся операторов из *opset*
Строки 9-10	Если *pqual* не используется в сигнатуре, то возвратить построенные объекты.
Строки 12-13	Создать новое уникальное имя для неявного оператора и построить новую таблицу отображений *Qual*, содержащую это имя.
Строка 14	Возвратить отображения *Qual* и требования *Dict* всех оставшихся операторов из *opset* и возвратить требование отображения для этого оператора и его требование *Dict*

extract-inherited-axioms(*litmap*, *opmap*, *qual*, *pqual*)(*dict*) ≜ (3.4.7.13)

1 (**let** *qualset* = *all-visible-sorts*(*dict*) \ {*qual*} **in**

2 **let** id_1 = *make-as*$_1$-*identifier*(*qual*)(*dict*) **in**

3 **let** pid_1 = *make-as*$_1$-*identifier*(*pqual*)(*dict*) **in**

4 **let** *axset* = **union** {s-*Equations*$_1$(*dict*(*qual*)) | *qual* ∈ *qualset*} **in**

5 **let** *axset'* = {*convert-axiom*(*axiom*, *litmap*, *opmap*, id_1, pid_1) *axiom* ∈ *axset*) **in**

6 *axset'* \ *axset*)

запись: (*Literal-operator-identifer*$_1$ ⇸ *Literal-operator-identifier*$_1$)
(*Operator-identifer*$_1$ ⇸ *Operator-identifier*$_1$) *Sortqual Sortqual* →
Dict → *Equations*$_1$

Цель Извлечь все аксиомы наследования, то есть, все аксиомы, которые используют любые операторы или литералы, которые включают в свои сигнатуры родительский сорт

Параметры

litmap	Отображение идентификаторов родительских литералов в идентификаторы литералов, определенные для сорта наследования
opmap	Отображение идентификаторов родительских литералов в идентификаторы литералов, определенные (явно или неявно) для сорта наследования
qual	*Qual* сорта наследования
pqual	*Qual* родительского сорта

Результат Множество равенств, использующих любые родительские операторы/литералы, где идентификаторы родительских операторов/литералов были заменены идентификатором операторов/литералов сорта наследования

Алгоритм

Строка 1	Построить множество *Quals* сортов, которые видимы во вложенных единицах подразделения. Исключить сорт наследования из множества.
Строки 2-3	Построить идентификаторы AS_1 сорта наследования и родительского сорта, соответственно.
Строка 4	Построить множество, которое является совокупностью множеств равенств, определенных в различных (обозреваемых) сортах.

Строка 5	Построить множество, состоящее из равенств, определенных в обозреваемых сортах, и такое, что каждое равенство преобразовано для того, чтобы включать в себя новые операторы/литералы вместо родительских операторов/сортов
Строка 6	Возвратить это множество за исключением тех равенств, которые не были изменены

convert-axiom(*axiom*, *lmap*, *omap*, *id*, *pid*) ≜ (3.4.7.14)

```
1   cases axiom:
2   (mk-Unquantified-equation1(t1, t2)
3       → mk-Unquantified-equation1(convert-term(t1, lmap, omap), convert-term(t2, lmap, omap)),
4    mk-Quantified-equations1(nms, sort, eqs)
5       → mk-Quantified-equations1(nms, if sort = pid then id else sort,
6                                  {convert-axiom(eq, lmap, omap, id, pid) | eq ∈ eqs}),
7    mk-Conditional-equation1(eqs, eq)
8       → mk-Conditional-equation1({convert-axiom(a, lmap, omap, id, pid) | a ∈ eqs},
9                                  convert-axiom(eq, lmap, omap, id, pid)),
10  T → axiom)
```

запись: *Equation*$_1$ (*Literal-operator-identifier*$_1$ ⇸ *Literal-operator-identifier*$_1$) (*Operator-identifier*$_1$ ⇸ *Operator-identifier*$_1$) *Sort-identifier*$_1$ *Sort-identifier*$_1$ → *Equation*$_1$

Цель Модифицировать аксиомы наследования для того, чтобы включить операторы и литералы, определенные с помощью сортов наследования

Параметры

axiom	Аксиома, которую необходимо модифицировать
litmap	Отображение идентификаторов родительских литералов в идентификаторы литералов, определенные для сорта наследования
opmap	Отображение идентификаторов родительских литералов в идентификаторы литералов, определенные (явно или неявно) для сорта наследования
id	Идентификатор AS$_1$ сорта наследования
pid	Идентификатор AS$_1$ родительского сорта

Результат Модифицированная аксиома (равенство)

Алгоритм

Строка 2	Если равенство является простым и неквантифицированным, тогда преобразовать термы с левой (*t1*) и правой (*t2*) стороны.
Строки 4-6	Если равенство является квантифицированным, тогда заменить сорт в квантификации, если он является родительским сортом, и преобразовать равенства, содержащиеся в квантифицированном равенстве.
Строки 7-9	Если равенство является равенством условия, тогда преобразовать ограничения (строка 8) и преобразовать равенства ограничения (строка 9).
Строка 10	Неформализованные текст остается без изменений.

```
convert-term(term, lmap, omap) ≜                                          (3.4.7.15)
 1   if is-Error-term1(term) then
 2     term
 3   else
 4     (let t = cases term:
 5              (mk-Ground-term1(te)     → te,
 6              mk-Composite-term1(te)  → te) in
 7       let t' = cases t :
 8              (mk-Identifier1(,)
 9                → if is-Ground-term1(term) ∧ t ∈ dom lmap then lmap(t) else t,
10              mk-Conditional-term1(t1, t2, t3)
11                → mk-Conditional-term1(convert-term(t1, lmap, omap),
12                                       convert-term(t2, lmap, omap),
13                                       convert-term(t3, lmap, omap)),
14              T → (let (opid, arglist) = t in
15                    let arglist' = ⟨convert-term(arglist[i], lmap, omap) | 1 ≤ i ≤ len arglist⟩ in
16                    if opid ∈ dom omap then
17                      (omap(opid), arglist')
18                    else
19                      (opid, arglist'))) in
20   if is-Ground-term1(term) then
21     mk-Ground-term1(t')
22   else
23     mk-Composite-term1(t'))
```

запись: $Term_1$ ($Literal\text{-}operator\text{-}identifier_1 \twoheadrightarrow Literal\text{-}operator\text{-}identifier_1$)
($Operator\text{-}identifier_1 \twoheadrightarrow Operator\text{-}identifier_1$) → $Term_1$

Цель Модифицировать терм для того, чтобы включить операторы и литералы, определенные с помощью сорта наследования

Параметры

term	Терм, который предстоит модифицировать
litmap	Отображение идентификаторов родительских литералов в идентификаторы литералов, определенные для сорта наследования
opmap	Отображение идентификаторов родительских литералов в идентификаторы литералов, определенные (явно или неявно) для сорта наследования

Результат Модифицированный терм

Алгоритм

Строка 1	Если это терм-ошибка, то ничего не делать
Строки 4-6	Произвести декомпозицию терма. Произвести замену несмотря на то, является ли терм основным или составным термом.
Строка 7	Если терм содержит идентификатор и терм является основным, тогда идентификатор обозначает литерал, и производится его замена, если он присутствует в таблице отображения литералов.
Строки 10-13	Если терм содержит терм условия, тогда преобразовать булевский терм (термы "then" и "else").
Строки 14-19	Если терм является применением оператора, тогда
Строка 14	Произвести декомпозицию применения оператора.
Строка 15	Преобразовать термы аргументов.
Строки 16-19	Если идентификатор оператора присутствует в таблице отображения операторов, тогда взять новый наследуемый идентификатор оператора из таблицы отображений (*omap*), иначе не изменять идентификатор оператора. Снова собрать применение оператора.
Строки 20-23	Снова собрать терм (выполнить в обратном порядке строки 4-6)

transform-geninst(*nm*, *instlist*, *adding*)(*dict*) ≜ (3.4.7.16)

```
1   (if instlist = ⟨⟩ then
2      adding
3    else
4      (let mk-Geninst0(id, parm) = hd instlist in
5       let mk-Properties0(lit, op, ax, ma, init) = transform-geninst(nm, tl instlist, adding)(dict) in
6       let tqual = get-visible-qual(id, GENERATOR)(dict) in
7       let mk-Id0(, genname) = id in
8       let mk-GeneratorD(fparm, prop) = dict(tqual) in
9       if len fparm ≠ len parm then
10        exit("§5.4.1.12.2: Длины списков фактических и формальных параметров должны быть одинаковы")
11       else
12        (let (tm, litm, opm, constm) = collect-genparms(fparm, parm) in
13         let tm' = tm + [genname ↦ mk-Id0(⟨⟩, nm)] in
14         let mk-Properties0(lit', op', ax', ma', init') = prop in
15         let lit" = ⟨insert-genparms(lit'[i])(tm', litm, opm, constm) | 1 ≤ i ≤ len lit'⟩,
16             op" = ⟨insert-genparms(op'[i])(tm', litm, opm, constm) | 1 ≤ i ≤ len op'⟩,
17             ax" = ⟨insert-genparms(ax'[i])(tm', litm, opm, constm) | 1 ≤ i ≤ len ax'⟩,
18             ma" = ⟨insert-genparms(ma'[i])(tm', litm, opm, constm) | 1 ≤ i ≤ len ma'⟩ in
19         if init ≠ nil ∧ init' ≠ nil then
20           exit("§5.5.3.3: Более чем одно присвоение по умолчанию")
21          else
22           (let init" = if init = nil then init' else init in
23            mk-Properties0(lit ⇝ lit", op ⇝ op", ax ⇝ ax", ma ⇝ ma", init')))))
```

запись: $Name_0$ $Geninst_0$ $Properties_0 \rightarrow Dict \rightarrow Properties_0$

Цель: Преобразовать последовательности экземпляров генераторов в AS_0 $Properties_0$.

Параметры

nm — Имя определения сорта AS_0, содержащего экземпляр генератора.

instlist — Список экземпляров генератора AS_0.

adding — Дополнительные свойства, определенные конструкцией ADDING.

Результат — AS_0 $Properties_0$, которые должны быть преобразованы в AS1 в функции *transform-partial-typedef*.

Алгоритм

Строка 1 — Если список экземпляров генератора пуст, то возвратить свойства ADDING (функция рекурсивна).

Строка 4 — Произвести декомпозицию следующего экземпляра генератора из списка.

Строка 5 — Преобразовать оставшиеся экземпляры генератора и возвратить свойства *properties*, куда были включены свойства всех оставшихся экземпляров генератора списка.

Строки 6-7 — Построить *Qual* идентификатора генератора и произвести декомпозицию идентификатора генератора.

Строка 8 — Произвести декомпозицию дескриптора генератора.

Строки 9-10 — Длина списка фактических параметров генератора должна быть равна длине списка формальных параметров генератора.

Строка 12 — Построить 4 таблицы отображения, которые содержат соотношения между фактическими и формальными параметрами. *tm* содержит соотношения между фактическими и формальными параметрами сортов, *litm* -соотношения между фактическими и формальными параметрами литералов, *opm* - соотношения между фактическими и формальными параметрами операторов, *constm* - соотношения между фактическими и формальными параметрами констант.

Строка 13 — Рассмотреть имя генератора в качестве формального параметра и включить его в таблицу отображения сортов.

Строки 15-18	Заменить каждый формальный параметр в теле генератора соответствующим фактическим параметром. т.е. заменить формальные параметры в сигнатуре литералов (строка 15), сигнатуре операторов (строка 16), равенств(строка 17) и разделе отображений равенств (строка 18).
Строка 19	Если экземпляр генератора содержит присвоение по умолчанию, то другой экземпляр генератора или свойства ADDING не могут содержать присвоение по умолчанию.
Строка 23	Собрать эти свойства из свойств этого экземпляра генератора, объединенного со свойствами других экземпляров.

insert-genparms(*node*)(*tmap*, *lmap*, *omap*, *constmap*)≜ (3.4.7.17)

```
1   cases node:
2     (mk-Opspec0(nm, sortl, sort)
3       → mk-Opspec0(insert-parm(nm, omap), ⟨insert-parm(sortl[i], tmap)| 1 ≤ i ≤ len sortl⟩,
4                    insert-parm(sort, tmap)),
5     mk-Equation0(t1, t2)
6       → mk-Equation0(insert-genparms(t1)(tmap, lmap, omap, constmap),
7                      insert-genparms(t2)(tmap, lmap, omap, costmap)),
8     mk-Condequation0(eql, eq)
9       → mk-Condequation0(⟨insert-genparms(eql[i])(tmap,lmap,omap,constmap) | 1 ≤ i ≤ len eql⟩,
10                         insert-genparams(eq)(tmap,lmap,omap,constmap)),
11    mk-Mappingaxiom0(vl, sid, al)
12      → mk-Mappingaxiom0(vl, insert-parm(sid, tmap),
13                         ⟨insert-genparms(al[i])(tmap, lmap, omap, constmap) | 1 ≤ i ≤ len al⟩),
14    mk-Quantequation0(vl, sid, al)
15      → mk-Quantequation0 (vl, insert-parm(sid, tmap),
16                         ⟨insert-genparms(al[i])(tmap, lmap, omap, constmap) | 1 ≤ i ≤ len al⟩),
17    mk-Name0(,)
18      → insert-parm(node, lmap),
19    mk-Id0(,)
20      → insert-parm(node, lmap + constmap),
21    mk-Operatorterm0(x, tli)
22      → mk-Operatorterm0(insert-parm(x, omap),
23                         ⟨insert-genparms(tli[i])(tmap, lmap, omap, constmap) | 1 ≤ i ≤ len tli⟩),
24    mk-Condterm0(e1, e2, e3)
25      → mk-Condterm0(insert-genparms(e1)(tmap, lmap, omap, constmap),
26                     insert-genparms(e2)(tmap, lmap, omap, constmap),
27                     insert-genparms(e3)(tmap, lmap, omap, constmap)),
28    mk-Monadterm0(op, e)
29      → mk-Monadterm0(op, insert-genparms(e)(tmap, lmap, omap, constmap)),
30    mk-Infixterm0(e1, op, e2)
31      → mk-Infixterm0 (insert-genparms(e1) (tmap, lmap, omap, constmap),
32                     op, insert-genparms(e2)(tmap, lmap, omap, constmap)),
33    mk-Spellingterm0(x)
34      → mk-Spellingterm0(insert-parm(x, lmap)),
35    T → node)
```

запись : $(Literal_0 | Op_0 | Axiom_0 | Mappingaxiom_0) \rightarrow (Name_0 \twoheadrightarrow Id_0)\ (Name_0 \twoheadrightarrow Literal_0)$
$(Name_0 \twoheadrightarrow (Name_0 | Quotedop_0))\ (Name_0 \twoheadrightarrow Term_0) \rightarrow (Literal_0 | Op_0 | Axiom_0 | Mappingaxiom_0)$

Цель Заменить в теле генератора формальные параметры генератора фактическими

Параметры

node	Объект, встретившийся в теле генератора. Эта функция рекурсивно проходит по поддеревьям вершины *node*
tmap	Отображение формальных параметров сорта на фактические параметры сорта

lmap	Отображение формальных параметров литералов на фактические параметры литералов
omap	Отображение формальных параметров операторов на фактические параметры операторов
consmap	Отображение формальных параметров констант на фактические параметры констант

Результат Вершина *node*, в которой были заменены все формальные параметры.

Алгоритм

Строки 2-4	Если объект является сигнатурой оператора, рассмотреть имя оператора (*nm*), список аргументов сорта (*sortl*) и сорт результата (*sort*).
Строки 5-7	Если объект является равенством, рассмотреть термы *t1* и *t2*.
Строки 8-10	Если объект является равенством условия, рассмотреть ограничения (*t1*) и ограничения равенства (*t2*).
Строки 11-13	Если объект является равенством отображения, рассмотреть идентификатора сорта (*sid*) и содержащиеся равенства (*al*).
Строки 14-16	Если объект является квантифицированным равенством, рассмотреть идентификатор сорта (*sid*) и содержащиеся равенства (*al*).
Строка 17	Если вершина есть $Literal_0$ $Name_0$, рассмотреть его, используя отображение литералов.
Строка 19	Если вершина является идентификатором (встретившимся в равенстве) рассмотреть ее, используя отображение литералов и отображение констант (формальные параметры литералов могут быть также использованы в равенствах).
Строки 21-23	Если объект является применением оператора, рассмотреть идентификатор оператора (*x*) и термы аргументов (*tli*).
Строки 24-27	Если объект является термом условия, рассмотреть булевский терм (*e1*) и термы "then" (*e2*) и "else" (*e3*)
Строки 28-32	Если объект является одноместным или двуместным термом, рассмотреть содержащийся(еся) терм(ы).
Строки 33	Если объект является термом проверки, рассмотреть содержащийся идентификатор.
Строка 35	Другие виды вершин *node* не могут или не должны содержать формальные параметры

insert-parm(*x*, *map*) ≜ (3.4.7.18)

```
1   cases x :
2    (mk-Id0(q, nm)
3      → if nm ∈ dom map then
4          if q = ⟨⟩ then
5            cases map(nm):
6             (mk-Name0(,)
7               → mk-Id0(⟨⟩, map(nm)),
8             mk-Quotedop0()
9               → mk-Qualop0(⟨⟩, map(nm)),
10            mk-Nmclass0()
11              → exit("§5.4.1.12: Класс имен не может быть использован в равенствах" ),
12            mk-String0()
13              → mk-Stringterm0(⟨⟩, map(nm)),
14            T→ map(nm))
15         else
16           exit("§5.4.1.12: Формальное имя генератора квалифицировано")
17       else
18         x,
19   mk-Name0(,)
20    → if x ∈ dom map then
21         cases map(x):
22          (mk-Name0(,),
23           mk-Nmclass0(),
24           mk-String0()
25             → map(x),
26           mk-Quotedop0()
27             → map(x),
28           T→ exit("§5.4.1.12: Параметр константы генератора используется в сигнатуре литерала"))
29       else
30         x,
31   T → x)
```

запись: $(Id_0 | Name_0 | Quotedop_0)(Name_0 \twoheadrightarrow (Term_0 | Literal_0 | Quotedop_0)) \rightarrow$
$(Term_0 | Literal_0 | Quotedop_0)$

Цель Проверить, являются ли идентификатор или имя формальными параметрами генератора, и, если это так, заменить их соответствующими фактическими параметрами

Параметры

x Идентификатор или имя, который нужно проверить. Для упрощения функции *insert-genparms* пусть *x* всегда будет $Quotedop_0$ (в этом случае *x* возвращается без изменений).

map Отображение, содержащее связь между формальными и фактическими параметрами для некоторого класса параметров, зависящего от контекста, в котором применяется *insert-parm*. Класс является либо типом формальных параметров, либо композицией формальных параметров литералов и констант.

Результат Соответствующий фактический параметр, если *x* является формальным параметром, в противном случае - *x*

Алгоритм

Строка 2 Если обрабатываемый объект является идентификатором и имя содержится в отображении (*map*), тогда идентификатор обозначает формальный параметр, в этом случае он не должен быть квалифицирован (строки 4 и 6).

Строка 6 Если фактический параметр является именем (литерала или оператора), то возвратить его, вложенным в идентификатор (потому что он использован в применяемом контексте).

Строка 8	Если фактический параметр является оператором в кавычках, то возвратить его вложенным в идентификатор оператора ($Qualop_0$)
Строка 10	Класс имен не может быть использован в применяемом контексте.
Строка 12	Если фактический параметр является символьной строкой, то возвратить его, вложенным в идентификатор литерала строки ($Stringterm_0$).
Строка 14	Иначе фактический параметр является термом, в этом случае возвращается терм.
Строка 18	Если идентификатор не является формальным параметром, то идентификатор возвращается без изменений.
Строка 19	Если рассматриваемый объект является именем и содержится в таблице отображений, тогда он является формальным параметром, используемым в контексте определения.
Строки 22-24	Если фактический параметр является именем или классом имен или символьной строкой, то он возвращается.
Строка 26	Если фактический параметр является инфиксной операцией в кавычках, то возвращается содержащаяся инфиксная операция.
Строка 28	В других случаях фактический параметр обозначает терм, который является недопустимым параметром в данном контексте определения.
Строка 30	Если имя не является формальным параметром, то возвратить его без изменений

```
collect-genparms(fparm, parm) ≜                                          (3.4.7.19)
 1   (if fparm = ⟨⟩ then
 2      ([], [], [], [])
 3     else
 4      (let (trest, litrest, oprest, constrest) = collect-genparms(t1 fparm, t1 parm) in
 5       let exclamation =
 6           if is-Id0(hd parm) then
 7             (let mk-Name0(, exc) = s-Name0(hd parm) in
 8              exc ≠ nil)
 9            else
10             false in
11         cases hd fparm:
12         (mk-Sortparm0(nm)
13           → if is-Id0 (hd parm) ∧ ¬ exclamation then
14                (trest + [nm ↦ hd parm], litrest, omprest, constrest)
15              else
16                exit("§5.4.1.12.2: Фактический параметр генератора TYPE должен быть идентификатором"),
17          mk-Opparm0(nm)
18           → if (is-Id0(hd parm) ∧ s-Qualifier0(hd parm) = ⟨⟩) then
19                (trest, litrest, oprest + [nm ↦ s-Name0(hd parm)], constrest)
20               else
21                if is-Quotedop0(hd parm) then
22                  (trest, litrest, oprest + [nm ↦ hd parm], constrest)
23                 else
24                  exit("§5.4.1.12.2: Фактический параметр генератора OPERATOR должен быть сигнатурой оператора"),
25          mk-Litparm0(nm)
26           → cases hd parm:
27               (mk-String0(str)
28                 → (trest, litrest + [nm ↦ str], oprest, constrest),
29               mk-Id0(q, nam)
30                 → if q = ⟨⟩ ∧ ¬ exclamation then
31                      (trest, litrest + [nm ↦ nam], oprest, constrest)
32                    else
33                      exit("§5.4.1.12.2: Фактический параметр генератора LITERAL должен быть сигнатурой литерала"),
34                mk-Nmclass0()
35                 → (trest, litrest + [nm ↦ hd parm], oprest, constrest),
36                T→ exit("§5.4.1.12.2: Фактический параметр генератора LITERAL должен быть сигнатурой литерала")),
37          mk-Tempar0(nm)
38           → if is-Nmclass0(hd parm) ∨ is-Quotedop0(hd parm) ∨ exclamation then
39                exit("§5.4.1.12.2: Фактический параметр генератора CONSTANT должен быть термом")
40               else
41                (trest, litrest, oprest, constrest + [nm ↦ hd parm]))))
```

запись: $Genparm_0^*\ Genactparm_0 \rightarrow (Name_0 \twoheadrightarrow Id_0)\ (Name_0 \twoheadrightarrow Literal_0)$
$(Name_0 \twoheadrightarrow (Name_0 \mid Quotedop_0))\ (Name_0 \twoheadrightarrow Term_0)$

Цель

Построить 4 таблицы отображения, которые устанавливают соотношение между 4 видами формальных параметров и соответствующих фактических параметров для экземпляров генератора. Перед применением данной функции проверяется, чтобы два списка имели одинаковую длину.

Параметры

fparm	Список формальных параметров
parm	Список фактических параметров

Алгоритм

Строка 1	Когда список формальных параметров просмотрен, возвратить пустые отображения (функция рекурсивна).

Строка 4	Построить отображения для остальных формальных параметров (всех, кроме первого).
Строки 5-10	Пусть *exclamination* будет равно **true**, если фактический параметр является Id_0, а имя $Name_0$ включает в себя "!".
Строки 12-16	Если формальный параметр является параметром сорта, тогда фактический параметр должен быть идентификатором без " ! ". Ассоциация между формальным именем и этим идентификатором добавляется в таблицу отображений сорта.
Строки 17-24	Если формальный параметр является параметром оператора, то фактический параметр должен быть неквалифицированным идентификатором (т.е. именем), либо инфиксным оператором в кавычках.
Строка 19 и строка 22	Ассоциация между формальным именем и фактическим именем относительно фактической инфиксной операции включается в таблицу отображений операторов.
Строка 25	Если формальный параметр является параметром литерала, то фактический параметр должен быть неквалифицированным идентификатором, либо неквалифицированной строкой, либо классом имен.
Строка 27 и строка 34	Ассоциация между формальным именем и фактическим именем или строкой или классом имен включается в таблицу отображений. литералов.
Строки 37-41	Если формальный параметр является параметром константы, тогда фактический параметр должен быть термом (т.е. не классом имен или инфиксной операцией в кавычках). Если терм является Id_0, он не может включать в себя " ! ".
Строка 41	Ассоциация между формальным именем и термом включается в таблицу отображения констант.

transform-syntype(**mk**-*Syntypedef*$_0$(*nm*, *pid*, *initial*, *vallist*, *tnm*))(*dict*)≜ (3.4.7.20)

```
1   (let tq = get-visible-qual(pid, TYPE)(dict) in
2   if is-recursive-sort(tq, {})(dict) then
3     exit("§5.4.1.9: Подтип основан на самом себе")
4    else
5     if tnm ∉ {nm, nil} then
6       exit("§5.4.1.9: Завершающее имя в определении подтипа отличается от имени определения")
7      else
8       (let tqual = get-parent(tq)(dict) in
9        let initial1 = cases dict(tq):
10                      (mk-SortD(,, init, )
11                        → init,
12                       mk-SyntypeD(,, init, )
13                        → init) in
14       let expr1 = if initial = nil then
15                      initial1
16                     else
17                       (let (as1tree,, ) =
18                           transform-expr(initial, CONSTANT, {tqual})(dict) in
19                        as1tree) in
20       let (,as1valset) = transform-valueset({tqual}, vallist)(dict) in
21       let as1id = make-as1-identifier(tqual)(dict) in
22       let nm' = if is-ServiceD(dict(dict(SCOPEUNIT))) then
23                   create-unique-name()
24                  else
25                   nm in
26       let as1dcl = mk-Syn-type-definition1(name-to-name1(nm'), as1id, as1valset) in
27       let synqual = dict(SCOPEUNIT) ⇀ ((TYPE, nm')) in
28       let d = [dict(SCOPEUNIT) ⇀ ((TYPE, nm)) ↦ mk-SyntypeD(tq, synqual, expr1, as1valset)] in
29       ({as1dcl}, d)))
```

запись: *Syntypedef*$_0$ → *Dict* → *Syn-type-definition*$_1$-**set** *Dict*

Цель Преобразовать выражение подтипа в AS_1

Параметры Определение подтипа, содержащее:

nm Имя подтипа.

pid Идентификатор родительского сорта.

initial Начальные значения для переменных подтипа.

vallist Множество допустимых значений для переменных сорта.

trm Имя, завершающее определение.

Результат См. *transform-decllist*

Алгоритм

Строка 1 Извлечь *Qual* для определенного родительского сорта.

Строка 2 Родительский сорт не должен базироваться на данном подтипе.

Строка 5 Завершающее имя должно соответствовать определенному имени подтипа.

Строка 8 Извлечь частичный тип определенного родительского сорта.

Строка 9 Пусть $initial_1$ обозначает начальное выражение *pid*.

Строки 14-19 Построить AS_1 - версию начального выражения. Если им пренебрегаем, значит, используется начальное выражение *pid*.

Строка 20 Преобразовать множество значений.

Строка 22 Если определение подтипа встречается в сервисе, тогда создать новое имя для этого подтипа

Строка 27 Построить *Qual*, который будет использоваться при извлечении идентификатора AS_1 подтипа (там, где используется подтип).

Строки 28-29 Возвратить определение подтипа AS_1 (as_1dcl) и требование *Dict*, содержащее дескриптор подтипа (*d*).

transform-valueset(*tqualset*, *valset*)(*dict*) ≜ (3.4.7.21)

```
1   (if valset = ⟨⟩ then
2      (let boolq = get-predef-sort("BOOLEAN")(dict) in
3       let orq = boolq ⌢ ⟨(OPERATOR, (OR, ⟨boolq, boolq⟩, boolq))⟩ in
4      (tqualset, mk-Range-condition1(make-as1-identifier(orq)(dict), {})))
5    else
6      (let mk-Condition0(cr, expr) = hd valset in
7      let (tset, as1range) = transform-valuerange(tqualset, cr, expr)(dict) in
8      let (trest, mk-Range-condition1(orld, condset)) = transform-valueset(tset, tl valset)(dict) in
9      (trest, mk-Range-condition1(orld, {as1range} ∪ condset))))
```

запись: *Sortqual*-set $Condition_0^* \rightarrow Dict \rightarrow$ *Sortqual*-set $Range\text{-}condition_1$

Цель Преобразовать множество значений в AS_1.

Параметры

tqualset Множество возможных (допустимых) сортов,

valset Список диапазонов значений. Когда множество значений встречается в определении подтипа, *tqualset* содержит только родительский сорт. Когда множество значений встречается в ответе решения, *tqualset* содержит несколько подмножеств (также зависящих от других ответов) возможных сортов для выражения запроса.

Результат Множество возможных (допустимых) сортов после обработки множества значений, т.е. результирующее множество является подмножеством *tqualset*. Также возвращается множество значений AS_1 ($Range\text{-}Expression_1{}^*$).

Алгоритм

Строки 1-4	Когда просмотрены все элементы множества значений, возвращается множество допустимых сортов и пустое множество *Range-condition*$_1$, содержащее AS$_1$-идентификатор логического оператора OR.
Строка 6	Взять первый элемент множества (списка), который состоит из необязательных операторов или необязательных выражений (*cr*), после которых следует выражение (*expr*).
Строки 7-9	Преобразовать диапазон значений и результирующее (возможно ограниченное) подмножество сорта (*tset*) во время преобразования оставшихся значений подмножества.

transform-valuerange(*tqualset, cr, expr*)(*dict*) ≜ (3.4.7.22)

```
1   (let bq = get-predef-sort("BOOLEAN")(dict) in
2    let cr' = if cr = nil then EQ else cr in
3    let isrel = cr' ∈ {NE, EQ, GT, LT, LE, GE} in
4    let cr" = if isrel then cr' else LE in
5    let tqualset' =
6        if isrel then
7          tqualset
8         else
9          (let (,tqset, ) = transform-expr(cr, CONSTANT, {bq})(dict) in
10          tqset ∩ tqualset) in
11   let opqset =
12      {opqual ∈ all-visible-operators(⟨⟩, cr", dict(SCOPEUNIT))(dict) |
13       is-OperatorD(dict(opqual)) ∧
14       (let (, (,argl, res)) = opqual[len opqual] in
15        len argl = 2 ∧
16        res = bq ∧ argl[1] = argl [2] ∧ elems argl ⊂ tqualset')} in
17   if opqset = {} then
18      exit("§5.4.1.9.1: Оператор упорядочивания не определен для сорта диапазона условий")
19    else
20     (let tset = tqualset' ∩ {sq ∈ dom dict | is-SortD(dict(sq)) ∧
21                                  (∃q ∈ opqset)((OPERATOR,(cr",⟨sq, sq⟩,bq))=q[len q])}in
22      if card tset ≥ 1 then
23        (tset, nil)
24       else
25        (let (as1tree,, ) = transform-expr(expr, CONSTANT, tset)(dict) in
26         let sort ∈ tset in
27         let opqual = sort ⇝ ⟨(OPERATOR, (cr", ⟨sort, sort⟩, bq))⟩ in
28         if isrel then
29           (tset, mk-Open-range1(make-as1-identifier(opqual)(dict), as1tree))
30          else
31           (let (as1tree',, ) = transform-expr(cr, CONSTANT, tset)(dict) in
32            let r1 = mk-Open-range1(make-as1-identifier(opqual)(dict), as1tree'),
33                r2 = mk-Open-range1(make-as1-identifier(opqual)(dict), as1tree) in
34            let andq = bq ⇝ ⟨(OPERATOR, (AND, ⟨sort, sort⟩, {bq}))⟩ in
35            (tset, mk-Closed-range1(make-as1-identifier(andq)(dict), r1, r2))))))
```

запись: *Sortqual*-set [*Expr*$_0$| *Relop*$_0$] *Expr*$_0$→*Dict*→ *Sortqual*-set [*Condition-item*$_1$]

Цель Преобразовать *Condition*$_0$AS$_0$ в *Condition-item*$_1$AS$_1$

Параметры

tqualset	Множество допустимых сортов перед преобразованием диапазонов значений
cr	Оператор отношения или первое выражение, встретившиеся в диапазоне значений
expr	(Второе) выражение, встретившееся в диапазоне значений

Результат Множество допустимых сортов после преобразования диапазонов значений и построенный *Condition*$_1$ AS$_1$. Подмножество сортов используется в преобразовании операций решений, где данная функция применяется дважды. Во-первых, с целью получения сорта, и во-вторых, с целью преобразования диапазонов значений

Алгоритм

Строка 1	Построить *Qual* булевского сорта.
Строка 2	Если ни оператор отношений, ни логические выражения не определены в диапазоне значений, то применяется оператор равенства.
Строка 4	Если *cr* является выражением, то оператор " меньше или равно " LE используется, как оператор отношения.
Строки 5-10	Если *cr* является выражением, то множество допустимых сортов (*tqualset*) ограничено допустимыми сортами для данного выражения.
Строки 11-16	Построить *Qual* множества допустимых операторов отношения, так что длина списка аргументов была равна 2; результирующий сорт являлся булевским, два аргумента были одного сорта, причем этот сорт являлся бы одним из допустимых сортов в контексте, в котором используется диапазон значений. (Если во множестве больше, чем один элемент, то имя оператора (*cr*) перегружено.)
Строки 17-18	Если нет таких операторов, то произошла ошибка.
Строки 20-21	Построить новое множество допустимых сортов, ограничивая старое множество допустимых сортов (*tqualset'*) так, чтобы оно содержало только те сорты, которые могут быть использованы в операторах в *opqset*.
Строки 22-23	Если в ограниченном множестве допустимых сортов больше, чем один элемент, тогда диапазон значений (все еще) неоднозначен, диапазон значений не возвращается (возвращается nil).
Строка 25	Вычислить выражение опять. В данном случае *tset* содержит ровно один сорт, и, следовательно, возвращается выражение AS$_1$ (если оно корректно).
Строка 26	Обозначить этот сорт из *tset* как *sort*.
Строка 27	Построить *Qual* используемого оператора.
Строки 28-29	Если *cr* обозначает оператор отношения, тогда возвратить множество, содержащее сорт, и *Condition-item*$_1$, который является *Open-range*$_1$.
Строка 31	Если *cr* является выражением, то преобразовать выражение (повторно).
Строки 32-33	Построить два *Open-range*$_1$, которые вместе с логическим оператором AND образуют *Close-range*$_1$.
Строка 34	Построить *qual* оператора AND и
Строка 35	Возвратить результирующее множество сортов (содержащее один элемент); *Close-range*$_1$, который содержит идентификатор AS$_1$ оператора AND ; и два других *Close-range*$_1$.

transform-synonymdef(**mk**-*Synonymdef*$_0$(*nm*, *tid*, *expr*))(*dict*) ≜ (3.4.7.23)

```
1   (let squal = dict(SCOPEUNIT) ⇝ ((VALUE, nm)) in
2    let sortset = if tid = nil then
3                     all-visible-sorts(dict)
4                    else
5                     {get-parent(get-visible-qual(tid, TYPE)(dict))(dict)} in
6   let (,tpset, ) = transform-expr(expr, CONSTANT, sortset)(dict + [squal↦ mk-ErrorD()]) in
7   if card tpset = 1 then
8     (let tp ∈t Qual be s.t. tp ∈ tpset in
9      ([squal↦ mk-SynD(tp, expr)]))
10   else
11    exit("§5.4.1.13: Сорт синонима не может быть однозначно определен"))
```

запись: *Synonymdef*$_0$ → *Dict* → *Dict*

Цель	Преобразовать определение синонима в AS_1
Параметры	AS_0 определение синонима, содержащее
nm	Имя синонима
tid	Необязательный идентификатор сорта
expr	Пассивное выражение
Результат	См. *transform-decllist*

Алгоритм

Строка 1	Построить квалификатор *Qual*, обозначающий синоним.
Строки 2 - 5	Сорт синонима должен быть найден в множестве сортов, которые видимы для определения синонима, если идентификатор сорта (*tid*) отсутствует, в противном случае сортом синонима является *tid*.
Строка 6	Преобразовать выражение, содержащееся в определении синонима, и вернуть подмножество сортов (*tpset*), которые согласовывают оба сорта синонима. Результирующее AS_1 выражение не используется. Так как идентификатор синонима не должен использоваться в *expr*, *squal* обозначает *ErrorD* во время вычисления выражения.
Строка 7	Допускается только один сорт для согласования сорта синонима и сорта выражения.
Строки 8 - 9	Построить требование *Dict*, содержащее дескриптор синонима.

3.4.7.1 Определения частичного типа

transform-sortdef(*nm*, *prop*, *parent*)(*dict*) ≜ (3.4.7.1.1)

```
1   (let nm' = if is-ServiceD(dict(dict(SCOPEUNIT))) then create-unique-name() else nm in
2    let newqual = dict(SCOPEUNIT) ⇝ ⟨(TYPE, nm')⟩ in
3    let descr = dict(SCOPEUNIT) ⇝ ⟨(TYPE, nm)⟩ in
4    let dict' = dict + [SCOPEUNIT ↦ descr] in
5    let prop' = transform-nameclass(prop) in
6    let prop'' = add-ordering(prop', descr)(dict) in
7    let mk-Properties0(lit, oplist, axioms, mapping, term) = prop'' in
8    let (eqop, eqax) = add-equality(descr)(dict) in
9    let (expr1,, ) = transform-expr(term, CONSTANT, {descr})(dict) in
10   if card elems lit ≠ len lit then
11      exit("§5.2.2: В определении частичного типа литерал определен дважды")
12     else
13      (let litd = [descr ⇝ ⟨(LITERAL, nm)⟩ ↦ mk-LiteralD(descr) | nm ∈ elems lit],
14           (as1op, opd) = transform-typing(oplist ⇝ eqop)(dict'),
15           as1ax = transform-axioms(axioms ⇝ eqax, AXIOMS)(dict'),
16           as1mapping = transform-axioms(mapping, MAPPING)(dict') in
17       let sortid = make-as1-identifier(newqual)(dict) in
18       let as1litset = {mk-Literal-signature1(nm, sortid) | nm ∈ card lit} in
19       let concaxioms1 = make-as1-concaxioms(dom litd)(dict) in
20       let typedef = dict(DATATYPEDEF) in
21       let mk-Data-type-definition1(typename, union, sorts, sigs, eqs ∪ concaxiom') = typedef in
22       let datatypedef' = mk-Data-type-definition1(typename, union, sorts ∪ {name-to-name(nm')},
23                                                   sigs ∪ as1op ∪ as1litset,
24                                                   eqs ∪ as1ax ∪ as1mapping ∪ concaxioms1) in
25       let sortdescr = mk-SortD(as1ax ∪ as1mapping, parent, expr1, newqual) in
26       (litd + opd + [descr          ↦ sortdescr,
27                      DATATYPEDEF    ↦ datatypedef'])))
```

запись: $Name_0\ Properties_0\ [Sortqual] \rightarrow Dict \rightarrow Dict$

Цель

Преобразовать свойства $Properties_0$ определения сорта и добавить результирующие AS_1 свойства к определению типа данных *Data-type-definition*$_1$, содержащемуся в записи *DATATYPEDEF* в *Dict*, и также добавить дескрипторы для литералов, операторов и сорта к *Dict*.

Параметры

nm — AS_0 имя сорта, подлежащее определению

prop — AS_0 свойства, которые должны быть преобразованы

parent — Квалификатор *Qual* родительского сорта в случае, когда свойства порождены сортом наследования, т.е. родитель *parent* обозначает идентификатор сорта, от которого этот сорт наследуется. Он помещается в дескриптор для дескриптора сорта *SortD* (см. определение *SortD*).

Результат

Dict, модифицированный AS_1 свойствами и модифицированный различными дескрипторами

Алгоритм

Строки 1-2 — Создать *Qual* для использования при создании имени AS_1. Если сорт определен в сервисе, новое уникальное имя используется в *Qual*

Строки 3-4 — Обновить SCOPEUNIT, указывающий на то, что свойства вычислены в контексте сорта *nm*, и обновить *Dict*, чтобы включить информацию о новой единице подразделения.

Строка 5 — Преобразовать свойства так, чтобы все классы имен в определении литералов были расширены до последовательности литералов.

Строка 6 — Преобразовать свойства так, чтобы любое упорядочение ORDERING в сигнатурах операторов было заменено сигнатурами оператора и равенствами, отражающими свойства упорядочения.

Строка 7 — Произвести декомпозицию $Properties_0$.

Строка 8 — Построить AS_0 операторы и равенства, отражающие свойства равенств.

Строка 9 — Преобразовать выражение по умолчанию (DEFAULT) в определении сорта в AS_1.

Строка 10 — Количество различных имен литералов в сигнатуре литералов должны быть равно длине списка сигнатур литералов.

Строка 13 — Построить дескрипторы для всех литералов в сигнатуре литералов.

Строка 14 — Построить AS_1 сигнатуры оператора (as_1op) и дескрипторы оператора для операторов в сигнатурах оператора и для операторов равенства.

Строка 15 — Построить равенства AS_1 из равенств AS_0 (аксиом), которые являются равенствами, определенными в свойствах, объединенных с уравнениями равенства.

Строка 17 — Построить равенства, соответствующие части MAP свойств AS_0.

Строка 18 — Построить сигнатуры литералов AS_0.

Строка 19 — Построить равенства AS_1, взятые из любого определенного литерала символьной строки (неявные равенства конкатенации).

Строка 21 — Извлечь и произвести декомпозицию текущего *Data-type-definition*$_1$.

Строки 22-23 — Построить новое *Data-type-definition*, которое является старым определением, обновленным, чтобы включить созданные свойства AS_1.

Строка 25 — Построить дескриптор для сорта.

Строка 26 — Вернуть *Dict*, содержащий дескрипторы для литералов, дескрипторы для операторов, дескриптор для сорта и модифицированный *Data-type-definition*$_1$

transform-nameclass(**mk-***Properties*$_0$(*litlist*, *o*, *a*, *m*, *t*)) ≜ (3.4.7.1.2)

```
1   if (∃lit ∈ elems litlist)(is-Nmclass0(lit)) then
2     (let i ∈ ind litlist be s.t. is-Nmclass0(litlist[i]) in
3      let mk-Nmclass0(regexpr) = litlist[i] in
4      let stringset = {str ∈ Char* | is-in-regular-expr(str, regexpr)} in
5      let nameset = form-names-and-strings(stringset) in
6      let namelist be s.t. card elems namelist = card nameset ∧
7         len namelist = card nameset ∧
8         (∀i1, i2 ∈ ind namelist)
9           (namelist[i1] < namelist[i2] ⊃ i1 < i2) in
10     let litlist' = ⟨litlist[n] | 1 ≤ n < i⟩ ⌢ namelist ⌢ ⟨litlist[n] | i < n ≤ len litlist⟩ in
11     transform-nameclass(mk-Properties0(litlist', o, a, m, t)))
12   else
13   mk-Properties0(litlist, o, a, m, t)
```

Запись: *Properties*$_0$ → *Properties*$_0$

Цель	Преобразовать некоторые AS$_0$ свойства так, чтобы любые содержащиеся классы имен были расширены до последовательности имен литералов.
Параметры	AS$_0$ свойства, содержащие
litlist	Список сигнатуры литералов
o	Список сигнатуры операторов
a	Равенства
m	Отображающие равенства
t	Терм, отображающий значение по умолчанию (DEFAULT)
Результат	Модифицированные AS$_0$ свойства

Алгоритм

Строка 1	Если (все еще) существует класс имен в сигнатуре литералов, то
Строки 2-3	Пусть *i* обозначает индекс элемента в списке сигнатуры литералов, который является классом имен и производит декомпозицию класса имен (строка 3)
Строка 4	Построить множество (Meta-IV) символьных строк, которые удовлетворяют регулярному выражению, обозначенному *regexpr*.
Строка 5	Из этого множества символьных строк построить множество имен (*nameset*), состоящее из AS$_0$ имен и символьных строк.
Строки 6-9	Преобразовать это множество в список, упорядоченный в алфавитном порядке.
Строка 10	Заменить класс имен в сигнатуре литералов созданным списком и
Строка 11	Заменить все следующие классы имен.
Строка 13	Если не осталось ни одного класса имен, вернуть свойства

form-names-and-strings(*stringset*) ≜ (3.4.7.1.3)

```
1   if stringset = {} then
2     {}
3   else
4     (let string ∈ stringset in
5      if len string ≥ 2 ∧ string[1] = "'" ∧ string[len string] = "'" then
6        {mk-String(⟨string[i] | 2 ≤ i ≤ len string - 1⟩)} ∪
7        form-names-and-strings(stringset \ {string})
8      else
9        (let string' = check-name-syntax(string) in
10         {mk-Name0(string', nil)} ∪
11         form-names-and-strings(stringset \ {string})))
```

запись: *Char**-set → (*String*$_0$ | *Name*$_0$)-set

Цель Из множества символьных строк META-IV сформировать множество AS_0 имен и символьных строк.

Алгоритм

Строка 1	Когда просмотр закончен, ничего не возвращать (функция рекурсивна).
Строка 4	Выбрать элемент из множества и обозначить его как *string*.
Строка 5	*string* обозначает символьную строку AS_0, если первый и следующий символы являются одинарными кавычками.
Строки 6-7	Возвратить символьную строку AS_0 вместе с остатком имен и строкой.
Строки 9-11	Если элемент не является символьной строкой, то он обозначает имя AS_0.
Строка 9	Удалить/заменить пробелы в соответствии с семантикой подчеркнутых символов, записать имя заглавными символами и проверить орфографию по лексическим правилам, описанным в Z.100.
Строки 10-11	Возвратить символьную строку AS_0 вместе с остатком имен и символьных строк.

is-in-regular-expr(string, regexpr) ≜ (3.4.7.1.4)

```
1   ( cases regexpr:
2     (mk-Orregexp0(reg, partreg)
3       → is-in-regular-expr(string, reg) ∧
4         is-in-regular-expr(string, partreg),
5     mk-Andregexp0(reg, partreg)
6       → (∃(str1, str2))(str1 ⌢ str2 = string ∧
7                         is-in-regular-expr(str1, reg) ∧
8                         is-in-regular-expr(str2, partreg)),
9     mk-Rngregexp0(str1, str2, exptp)
10      → is-in-regular-range(string, str1, str2, exptp),
11    mk-Singregexp0(str1, exptp)
12      → is-in-regular-single(string, str1, exptp),
13    mk-Parenregexp0(regexp, exptp)
14      → is-in-regular-par(string, regexp, exptp)))
```

запись: $Char^*\ Regularexp_0 \rightarrow Bool$

Цель Проверить, включена ли символьная строка META-IV в регулярное выражение

Параметры

string	META-IV символьная строка
regexpr	Регулярное выражение

Результат True в случае успеха.

Алгоритм

Строки 2-4	Если регулярное выражение состоит из двух OR регулярных выражений, то результат есть true, если строка определена в одном из них.
Строки 5-8	Если регулярное выражение состоит из двух конкатенированных регулярных выражений, то результат есть true, если существуют две строки, такие, что при конкатенации они образуют *string* и такие, что первая подстрока определена в первом регулярном выражении, а вторая - во втором регулярном выражении.
Строка 9	См. *is-in-regular-single*.
Строка 11	См. *is-in-regular-single*.
Строка 13	См. *is-in-regular-par*

is-in-regular-range (*string*, mk-*String*$_0$(*str*1), mk-*String*$_0$(*str*2), *exptp*) ≜ (3.4.7.1.5)

```
1   (if len str1 ≠ 1 ∨ len str2 ≠ 1 then
2      exit("§5.4.1.14: Длина символьной строки в регулярном интервале не равна единице")
3     else
4      (if len string = 0 ∧ exptp = MULT then
5         true
6        else
7         (len string ≤ 1 ∧
8          (∀ch ∈ elems string)
9           ((hd str1 ≤ ch ∧
10            ch ≤ hd str2 ∧
11            ( cases exptp:
12              (mk-Name0(numstr, )
13                → if elems numstr ⊂ { "0", "1", "2", "3", "4", "5", "6", "7", "8", "9" } then
14                      (let i = form-integer(numstr) in
15                       len string = i)
16                    else
17                      exit("§5.4.1.14: Имя повторения в классе имен должно обозначать натуральное число" ),
18              T → true)))))))
```

запись: *Char* String String* [*Regexpexp*$_0$] → *Bool*

Цель Проверить, определена ли строка в регулярном выражении, которое является диапазоном символов, возможно повторенных несколько раз.

Параметры

string	Строка, которая будет проверяться
str1,str2	Последовательности символов, обозначающие верхнюю и нижнюю границы
regexpr	Необязательный коэффициент повторения

Результат True в случае успеха

Алгоритм

Строки 1-2	Длина верхней и нижней границы должны быть равны единице.
Строка 4	Пустая строка тоже может присутствовать в диапазоне символов, если задан коэффициент повторения и он может быть применен к пустым строкам.
Строки 7-10	В противном случае длина должна быть больше или равна единице, и для всех символов (*ch*) строки код представления должен быть больше или равен нижней границе диапазона (строка 8) и или меньше или равен верхней границе диапазона.
Строки 11-17	Если коэффициент повторения есть имя, то он всегда должен обозначать простое литеральное имя (должен состоять из цифр) и обозначаемое им значение должно быть равно длине строки.

form-integer (*str*) ≜ (3.4.7.1.6)

```
1   (let ch = str[len str] in
2    let n = cases ch:
3            ( "0" → 0
4              "1" → 1
5              "2" → 2
6              "3" → 3,
7              "4" → 4,
8              "5" → 5,
9              "6" → 6,
10             "7" → 7,
11             "8" → 8,
```

```
12          "9" → 9) in
13    if tl str = ⟨⟩ then n else form-integer(⟨str[i] | 1 ≤ i ≤ len str⟩) * 10 + n)
```

запись: $Char^* \rightarrow Intg$

Цель Преобразовать последовательность символов (проверка) в значение *Intg* META-IV

Алгоритм

Строка 1	Извлечь последний символ из списка символов.
Строки 2-12	Преобразовать символ в значение *Intg* META-IV.
Строка 13	Если этот символ - единственный в списке, тогда возвратить значение, иначе умножить значения, полученные из остатка списка, и прибавить значение, полученное из символа.

is-in-regular-single(*string*, mk-$String_0$(*str*), *exptp*) ≜ (3.4.7.1.7)

```
1   ( cases exptp:
2     (MULT
3       → (∃n)(conc ⟨str | 0 ≤ i ≤ n⟩ = string),
4      PLUS
5       → (∃n)(conc ⟨str | 1 ≤ i ≤ n⟩ = string),
6      mk-Name0(numstr,)
7       → if elems numstr ⊂ { "0", "1", "2", "3", "4", "5", "6", "7", "8", "9" } then
8           (let n = form-integer(numstr) in
9            conc ⟨str | 1 ≤ i ≤ len n⟩ = string)
10         else
11          exit("§5.4.1.14: Имя повторения в классе имен должно обозначать натуральное число" ),
12     nil
13      → str = string))
```

запись: $Char^*\ String_0\ [Regexpexp_0] \rightarrow Bool$

Цель Проверить, может ли быть сформирована некоторая строка повторением другой строки

Параметры

string	Формируемая строка
str	Повторяемая строка
exptp	Коэффициент повторения

Результат True в случае успеха

Алгоритм

Строка 2	Если коэффициент повторения есть звездочка, то должно существовать значение *n* такое, что повторение (конкатенация) строки *str n* раз образует *string*.
Строка 4	Если коэффициент повторения есть плюс, то должно существовать такое ненулевое значение *n*, что повторение (конкатенация) строки *str n* раз образует *string*.
Строка 6	Если коэффициент повторения есть имя, то оно должно обозначать натуральное число. Результат есть true, если *string* может быть сформирована повторением строки *str n* раз.
Строка 12	Если коэффициент повторения опущен, то строки должны быть равны

is-in-regular-par(string, regexp, exptp) ≜ (3.4.7.1.8)

```
1   cases exptp:
2    (MULT
3      → (∃(str, n,))(conc ⟨str | 0 ≤ i ≤ n⟩ = string ∧
4                    is-in-regular-expr(str, regenxp)),
5    PLUS
6      → (∃(str, n))(conc ⟨str | 1 ≤ i ≤ n⟩ = string ∧
7                    is-in-regular- expr(str, regexp)),
8    mk-Name0(numstr, )
9      → if elems numstr ⊂ {"0", "1", "2", "3", "4", "5", "6", "7", "8", "9"} then
10         (let n = form-integer(numstr) in
11          (∃str)(conc ⟨str | 1 ≤ i ≤ n⟩ = string ∧
12                is-in-regular-expr(str, regexp)))
13       else
14         exit("§5.4.1.14: Имя повторения в классе имен должно обозначать натуральное число"),
15   nil
16     → is-in-regular-expr(string, regexp))
```

запись: *Char* Regularexp* [*Regexpexp*$_0$] → *Bool*

Цель Проверить, может ли строка быть сформирована повторением другой строки, определенной с помощью регулярного выражения

Параметры

string	Формируемая строка
regexp	Повторяемое регулярное выражение
exptp	Необязательный коэффициент повторения

Результат True в случае успеха

Алгоритм

Строки 2-4	Если коэффициент повторения есть звездочка, то должны существовать подстрока *str*, определенная с помощью регулярного выражения *regexp* и число *n*, такое, что повторение строки *str n* раз образует *string*
Строки 5-7	Если коэффициент повторения есть плюс, то должны существовать подстрока *str*, определенная с помощью регулярного выражения *regexp* и такое ненулевое значение *n*, что повторение строки *str n* раз образует *string*.
Строки 8-14	Если коэффициент повторения есть имя, то оно должно обозначать натуральное число. Результат есть true, если существует подстрока *str*, определенная с помощью регулярного выражения так, что строка *string* может быть сформирована повторением строки *str n* раз.
Строка 15	Если коэффициент повторения опущен, то результат есть true, если существует строка *str*, определенная с помощью регулярного выражения

transform-axioms(axioms, context)(dict) ≜ (3.4.7.1.9)

```
1   (if axioms = ⟨⟩ then {} else (transform-axiom(hd axioms, context)(dict) ∪
2                                 transform-axioms(tl axioms, context)(dict)))
```

запись: *Axiom*$_0$* *Context* → *Dict* → *Equations*$_1$

Цель Преобразовать список равенств (аксиом) AS$_0$ в равенства AS$_1$

Параметры

axioms	Преобразуемые аксиомы

context — Контекст, из которого преобразуются аксиомы. Контекст обозначает либо обычные аксиомы (AXIOMS), либо аксиомы отображения (MAPPING)

Результат — Порожденные равенства AS_1

Алгоритм

Строка 1 — Когда просмотр закончен, возвратить пустое множество.

Строки 1-2 — Возвратить множество равенств, состоящее из множества соответствующего первому равенству (строка 1), дополненному оставшимися равенствами (строка 2)

transform-axiom(axiom, context)(dict) ≜ (3.4.7.10)

```
1   ( cases axiom :
2     (mk-Equation0(,)
3       → {transform-equation(axiom, context )(dict)},
4     mk-Condequation0(eql, eq)
5       → (let eql ' = ⟨insert-equals-true(eql[i])(dict) | i ≤ i ≤ len eql⟩,
6             eq' = insert-equals-true(eq)(dict) in
7           {transform-condequation(mk-Condequation0(eql', eq' ), context)(dict)}),
8     mk-Quantequation0(,,)
9       → {transform-quantequation(axiom, context)(dict)},
10    mk-Mappingaxiom0(,,)
11      → transform-mapping(axiom)(dict),
12    T→ if is-Stringterm0(axiom) ∧ s-Qualifier0(axiom) = ⟨⟩ then
13          {mk-Informal-text1(axiom)}
14        else
15          (let axiom' = insert-equals-true(axiom)(dict) in
16           {transform-equation(axiom', context)(dict)})))
```

запись : $Axiom_0$ *Context* → *Dict* → $Equations_1$

Цель — Преобразовать аксиому AS_0 в совокупность равенств AS_1. За исключением случая аксиомы отображения, совокупность состоит из только одного элемента

Параметры

axiom — Аксиома, подлежащая преобразованию

context — Контекст, в котором производится преобразование (см. *transform-axioms*)

Результат — Порожденные равенства AS_1

Строка 2 — Если аксиома является простым равенством, то преобразовать это равенство.

Строки 4-7 — Если аксиома является условным равенством, то заменить булевские элементы в ограничении (*eql*) и ограничиваемом равенстве (*eq*) простым равенством, преобразовать результирующее условное равенство.

Строки 8-10 — Если аксиома является квантифицированными равенствами (строка 8) или равенством отображения (строка 10), то преобразовать ее.

Строки 12-13 — Иначе аксиома является термом, который обозначает неформальный текст, терм является неквалифицированной символьной строкой.

Строки 15-16 — Если аксиома не обозначает неформальный текст, то она является булевской аксиомой и "equals true" (равно истине) вставляется перед преобразованием

insert-equals-true(*axiom*)(*dict*) ≜ (3.4.7.1.11)

```
1   if is-Equation0(axiom) then
2      axiom
3   else
4      (let truenm = mk-Name0("TRUE", nil) in
5       let trueid = mk-Id0((hd dict(SCOPEUNIT)), truenm) in
6       mk-Equation0(axiom, trueid))
```

запись : $Unquantequation_0 \rightarrow Dict \rightarrow Equation_0$

Цель Если аксиома является булевским термом, то построить равенство AS_0, которое определяет терм, который является истинным

Параметры

axiom Проверяемая аксиома

Результат Возможно модифицированная аксиома

Алгоритм

Строка 1 Если аксиома является равенством, то все в порядке, иначе

Строка 4 Она является термом, тогда построить имя булевского сорта.

Строка 5 Построить идентификатор булевского сорта. Квалификатор является первым элементом в *Qual* SCOPEUNIT, которое обозначает текущую единицу подразделения. Первый элемент обозначает системный уровень.

Строка 6 Построить равенство из булевского терма и идентификатора литерала истина

transform-equation(mk-$Equation_0$(*term*1, *term*2), *context*)(*dict*) ≜ (3.4.7.1.12)

```
1   (let (as1term1, as1term2, dict') = transform-lhs-rhs(term1, term2, context)(dict) in
2    build-quant-equation(mk-Unquantified-equation1(as1term1, as1term2))(dict'))
```

запись : $Equation_0\ Context \rightarrow Dict \rightarrow Equations_1$

Цель Преобразовать простое неквантифицированное равенство AS_0 в равенство AS_1

Параметры

term1, term2 Два терма в преобразуемом равенстве

context См. *transform-axioms*

Результат

Строка 1 Преобразовать два терма. Новые дескрипторы *ValueidD* в *dict'* (см. определение *Dict*) содержат информацию о всех значениях идентификаторов, используемых в этих двух термах.

Строка 2 Построить неквантифицированное равенство из двух термов AS_1 и ограничить это равенство посредством некоторой квантификации (если существуют какие-либо дескрипторы *ValueidD* в *dict'*)

transform-lhs-rhs(*term*1, *term*2, *context*)(*dict*) ≜ (3.4.7.1.13)

```
1   (let totset = all-visible-sorts(dict) in
2    let (, tset1, ) =
3       if is-Errorterm0(term1)
4         then (..., totset,...)
5         else transform-term(term1, context, totset)(dict),
6       (,tset2, ) =
7       if is-Errorterm0(term2)
8         then (..., totset,...)
9         else transform-term(term2, context, totset)(dict) in
10   let tqual = if card (tset1 ∩ tset2) = 1 then
11                 (let tq ∈ (tset1 ∩ tset2) in
12                   tq)
13                else
14                   exit("§4.2.3: Два терма в равенстве должны быть одного сорта") in
15   let (as1term1,, d) =
16       if is-Errorterm0(term1)
17         then (mk-Error-term1(), ..., dict)
18         else transform-term(term1, context, {tqual})(dict) in
19   let (as1term2,, d') =
20       if is-Errorterm0(term2)
21         then (mk-Error-term1(),..., d)
22         else transform-term(term2, context, {tqual})(d) in
23   (as1term1, as1term2, d'))
```

запись : $Terms_0\ Terms_0\ Context \rightarrow Dict \rightarrow Term_1\ Term_1\ Dict$

Цель Преобразовать два терма AS_0 из равенства в два терма AS_1

Параметры

term1, term2 Два терма, которые должны быть преобразованы

context См. *transform-axioms*

Результат Два построенных терма AS_1 и *Dict*, включающий дескрипторы *ValueidD* значений идентификаторов в этих двух термах

Строка 1 Извлечь множество *Qual*, содержащее все обозреваемые сорта

Строки 2-9 Преобразовать эти два терма, чтобы определить множество допустимых сортов термов. *tset1* обозначает допустимые сорты для *term1*, *tset2* обозначает допустимые сорты для *term2*. Если один из термов является термом-ошибкой, то все обозреваемые сорты являются допустимыми для этого терма.

Строки 10-14 Определить сорт равенства. Должен существовать ровно один такой сорт, т.е. два множества должны иметь один общий сорт.

Строки 15-23 Если один из термов является термом-ошибкой, то возвратить терм-ошибку AS_1 для этого терма, иначе преобразовать терм (строка 18 и строка 22), допустимый сорт дан в качестве аргумента (*tqual*). *Dict*, возвращенный из *transform-term* для первого терма (строка 18), используется, когда второй терм преобразуется (строка 22) таким образом, что дескрипторы *ValueidD* не создаются дважды.

build-quant-equation(*equation*$_1$)(*dict*) ≜ (3.4.7.1.14)

```
1   if (∃q∈ dom dict)(is-ValueidD(dict(q)) ∧ ¬ s-Explicit(dict(q))) then
2     (let q∈ dom dict be s.t. is-ValueidD(dict(q)) ∧ s-Mapvalue(dict(q)) = nil ∧ ¬ s-Explicit(dict(q)) in
3       let mk-ValueidD(, tqs, ) = dict(q) in
4       let tq∈ tqs in
5       let (,nm) = q[len q] in
6       let equation' = build-quant-equation(equation1)(dict\{q}) in
7       let as1tid = make-as1-identifier(tq)(dict) in
8       mk-Quantified-equations1({name-to-name1(nm)}, as1tid, {equation'}))
9     else
10      equation1
```

запись : *Equation*$_1$ → *Dict* → *Equation*$_1$

Цель Для каждого имени в *dict* заключить равенство AS$_1$ при помощи квантификации

Параметры

equation$_1$ Равенство AS$_1$, которое должно быть квантифицировано. Т.к. функция является рекурсивной, то *equation*$_1$ может быть уже квантифицированным равенством

Алгоритм

Строка 1 Если в *Dict* существует дескриптор *ValueId* (дескриптор идентификатора значения) и он не вводится явной квантификацией, то

Строка 2 Обозначить *Qual* идентификатора значения через *q*.

Строка 3 Произвести декомпозицию дескриптора идентификатора значения. Множество сортов *tqs* на этом шаге содержит только один сорт.

Строка 4 Обозначить сорт идентификатора значения через *tq*.

Строка 5 Извлечь имя идентификатора значения.

Строка 6 Построить новое равенство при помощи ограничения старого равенства посредством квантификации, соответствующего оставшейся части идентификаторов значений.

Строка 7 Из *Qual* идентификатора значения построить идентификатор AS1.

Строка 8 Возвратить равенство, квантифицированное именем AS$_1$ и содержащее новое равенство

transform-quantequation(mk-*Quantequation*$_0$(*vl, tid, axl*), *context*)(*dict*) ≜ (3.4.7.1.15)

```
1   (let tqual = get-visible-qual(tid, TYPE)(dict) in
2    let qualset = {dict(SCOPEUNIT) ⇀ ⟨(VALUE, vl[i])⟩ | 1 ≤ i ≤ len vl} in
3    let dict' = dict + [q ↦ mk-ValueidD(nil, {tqual}, true) |
4                        q∈ qualset] in
5    let as1ax = transform-axioms(axl, context)(dict') in
6    let as1tid = make-as1-identifier(tqual)(dict) in
7    let as1vl = {name-to-name1(vl[i]) | 1 ≤ i ≤ len vl} in
8    mk-Quantified-equations1(as1vl, as1tid, as1az))
```

запись : *Quantequation*$_0$ *Context* → *Dict* → *Quantified-equations*$_1$

Цель Преобразовать квантифицированное равенство в AS$_1$

Параметры

vl, tid, axl Список значений имен, их сорт и список аксиом, содержащиеся в квантифицированном равенстве

context См. *transform-axioms*

Результат Квантифицированное равенство AS$_1$

Алгоритм

Строка 1	Построить *Qual* сорта (*tid*)
Строка 2	Построить *Quals* идентификаторов значений, соответствующих значениям имен (*vl*).
Строки 3-4	Модифицировать *Dict*, чтобы включить дескрипторы идентификаторов значений, **nil** указывает на отсутствие идентификаторов значений литералов (отображений), {*tqual*} указывает, что единственным допустимым сортом для идентификаторов значений является *tqual* (при неявной квантификации), это множество может содержать несколько элементов, *true* указывает, что идентификаторы значений показаны при явной квантификации.
Строка 5	Преобразовать аксиомы квантифицированного равенства в AS_1.
Строка 6	Для *Qual* сорта построить идентификатор AS_1.
Строка 7	Построить совокупность имен AS_1, содержащих имена значений.
Строка 8	Возвратить построенное квантифицированное равенство AS_1

transform-condequation(**mk**-*Condequation*$_0$(*eql*, *eq*), *context*)(*dict*) ≜ (3.4.7.1.16)

```
1   (let (eql1, dict ') = transform-restrictions(eql, context)(dict) in
2    let mk-Equation0(term1, term2) = insert-equals-true(eq)(dict) in
3    let (as1term1, as1term2, dict ") = transform-lhs-rhs(term1, term2, context)(dict ') in
4    let eq1 = mk-Unquantified-equation1(as1term1, as1term2) in
5    build-quant-equation(mk-Conditional- equation1(eql1, eq1))(dict "))
```

запись : *Condequation*$_0$ *Content* → *Dict* → *Equation*$_1$

Цель Преобразовать равенство условия в AS_1

Параметры

eql, eq	Ограничение и равенства ограничения в равенстве условия

Результат Построенное равенство условия AS_1

Алгоритм

Строка 1	Преобразовать ограничения.
Строка 2	Произвести декомпозицию равенства AS_0.
Строка 3	Преобразовать два терма из равенства ограничения AS_0. Заметьте, что *dict* ' используется в преобразовании равенств ограничения таким образом, что идентификаторы неявных значений в обоих частях являются одними и теми же.
Строка 4	Построить равенство AS_1.
Строка 5	Произвести вложение построенного равенства посредством квантификации (если *dict* "содержит только дескрипторы *ValueidD*).

transform-restrictions(*eql*, *context*)(*dict*) ≜ (3.4.7.1.17)

```
1   (if eql = ⟨⟩ then
2      ({}, dict)
3    else
4      (let mk-Equation0(term1, term2) = insert-equals-true(hd eql)(dict) in
5       if is-Errorterm0(term1) ∨ is-Errorterm0(term2) then
6          exit("§5.4.1.7: Терм-ошибка является частью ограничений" )
7       else
8          (let (as1term1, as1term2, dict ') = transform-lhs-rhs(term1, term2, context)(dict) in
9           let (eqlrest, dictrest) = transform-restrictions(tl eql, context)(dict ') in
10          (eqlrest ∪ {mk-Unquantified-equation1(as1term1, as1term2)}, dictrest))))
```

запись : *Unquantequation*$_0$* *Context* → *Dict* → *Equation*$_1$-**set** *Dict*

Цель Преобразовать список ограничений в множество ограничений AS_1 посредством рекурсивного прохода через этот список и сбора информации в *Dict* идентификаторов значений. Эта функция подобна *transform-equation*, но применяется к списку равенств.

transform-mapping(mk-*Mappingaxiom*$_0$(*vl*, *tid*, *axl*))(*dict*) ≜ (3.4.7.1.18)

```
1  (let tqual = get-visible-qual(tid, TYPE)(dict) in
2  let qualset = {dict(SCOPEUNIT) ⇀ ⟨(VALUE, vl[i])⟩ | 1 ≤ i ≤ len vl} in
3  let litset = {qual ∈ dom dict | is-LiteralD(dict(qual)) ∧ s-Sortqual(dict(qual)) = tqual} in
4  transform-mappingaxioms(qualset, tqual, litset, axl)(dict))
```

запись : *Mappingaxiom*$_0$ → *Dict* → *Equations*$_1$

Цель Преобразовать аксиому отображения в множество равенств AS_1

Параметры

vl	Список имен значений в аксиоме отображений
tid	Сорт имен значений
axl	Аксиомы, содержащиеся в аксиоме отображения

Результат Преобразованное равенство AS_1

Алгоритм

Строка 1	Извлечь *Qual* сорта.
Строка 2	Построить множество *Quals* для имен значений.
Строка 3	Извлечь множество *Quals* литералов, определенных для сорта *tqual*.
Строка 4	Выполнить преобразование

transform-mappingaxioms(*qualset*, *tqual*, *litset*, *axl*)(*dict*) ≜ (3.4.7.1.19)

```
1  if qualset = {} then
2    {}
3  else
4    (let qual ∈ qualset in
5     union {transform-axioms(axl, MAPPING)(dict + [qual ↦ mk-ValueidD(litqual, {tqual}, true)]) |
6            litqual ∈ litset} ∪
7     transform-mappingaxioms(qualset \ {qual}, tqual, litset, axl)(dict))
```

запись : *Qual*-set *Sortqual Qual*-set *Axiom*$_0$* → *Dict* → *Equations*$_1$

Цель Для данного множества идентификаторов значений их сорт и литералы, определенные для этого сорта, расширить множество равенств посредством замещения всех идентификаторов значений в аксиомах на литерал

Параметры

qualset	Множество *Quals*, представляющее идентификаторы значений
tqual	*Qual*, представляющий идентификатор их сорта
litset	Множество *Quals*, представляющее литералы, определенные для *tqual*
axl	Расширяемые аксиомы

Результат Множество равенств AS_1, в которых были заменены значения идентификаторов, соответствующие *qualset*

Алгоритм

Строка 1	Если просмотр закончен, вернуть пустое множество равенств.
Строка 4	Взять следующий идентификатор значения в множестве.

Строки 5-6 Построить множество равенств, которое составляет объединение всех этих множеств равенств, в которых идентификатор значений заменяется конкретным литералом из *litset*.

Строка 7 Возвратить построенное множество равенств, унифицированное равенствами, построенными при замене оставшихся идентификаторов значений

add-equality(*tqual*)(*dict*) ≜ (3.4.7.1.20)

```
1   (let bqual = get-predef-sort("BOOLEAN")(dict) in
2    let bid = as0-id(bqual) in
3    let tid = as0-id(tqual) in
4    let opl = ⟨mk-Opspec0(EQ, ⟨tid, tid⟩, bid), mk-Opspec0(NE, ⟨tid, tid⟩, bid)⟩ in
5    let anm = create-unique-name(),
6        bnm = create-unique-name(),
7        cnm = create-unique-name(),
8        trueid = mk-Id0(bqual, mk-Name0("TRUE", nil)) in
9    let aid = mk-Id0(⟨⟩, anm),
10       bid = mk-Id0(⟨⟩, bnm),
11       cid = mk-Id0(⟨⟩, cnm) in
12   let eq1 = mk-Equation0(mk-Infixterm0(aid, EQ, aid), trueid),
13       eq2 = mk-Equation0(mk-Infixterm0(aid, EQ, bid), mk-Infixterm0(bid, EQ, aid)),
14       eq3 = mk-Equation0(mk-Infixterm0(mk-Infixterm0(mk-Infixterm0(aid, EQ, bid), AND,
15                                                    mk-Infixterm0(bid, EQ, cid)),
16                                        IMPLY, mk-Infixterm0(aid, EQ, cid)), trueid),
17       eq4 = mk-Equation0(mk-Infixterm0(aid, NE, bid),
18                          mk-Monadterm0(NOT, mk-Infixterm0(aid, EQ, bid))),
19       eq5 = mk-Condequation0(⟨mk-Equation0(mk-Infixterm0(aid, EQ, bid), trueid)⟩,
20                              mk-Equation0(aid, bid)) in
21   (opl, ⟨mk-Quantequation0(⟨anm, bnm, cnm⟩, tid, ⟨eq1, eq2, eq3, eq4, eq5⟩)⟩))
```

запись: *Sortial* → *Dict* → $Opspec_0^+ Axiom_0^+$

Цель Для данного сорта построить равенство свойств AS_0, как определено в § 5.4.1.4 в Z.100

Параметр

tqual *Qual* сорта

Результат Список определений операторов AS_0 и список аксиом AS_0

Алгоритм

Строки 1-3 Построить (квалифицированные) идентификаторы AS_0 булевского сорта *tqual*.

Строка 4 Построить определения:

"=" : tqual, tqual -> boolean
"/=" : tqual, tqual -> boolean

Строки 5-7 Создать несколько уникальных имен, которые будут использованы при квантификации.

Строка 8 Построить квалифицированный идентификатор AS_0 булевского литерала TRUE.

Строки 9-11 Построить идентификаторы AS_0 для имен значений.

Строки 12-20 Построить пять аксиом (*eq1* - *eq5*):

```
aid = aid                    == true;
aid = bid                    == bid = aid;
(aid = bid) and (bid = cid)=>
                  aid=cid    == true;
aid /= bid                   == not(aid = bid);
aid = bid == true ==> aid    == bid;
```

Строка 21 Возвратить построенные определения операторов и аксиом

add-ordering(**mk**-*Properties*$_0$(*lit, oplist, axioms, map, init*), *tqual*)(*dict*) ≜ (3.4.7.1.21)

```
1   (if (∃op ∈ elems oplist)(is-Ordering0(op)) then
2     (let oplist ' = ⟨oplist[i] | 1 ≤ i ≤ len oplist ∧ is-Opspec0(oplist[i])⟩ ⟨⟩
3      if len oplist = len oplist ' - 1 then
4        (let ordaxioms = as0-ordering-axioms(tqual)(dict),
5             order = if len lit ≤ 1 then ⟨⟩ else as0-order(hd lit, tl lit, tqual),
6             ordtyping = as0-ordering-typing(tqual)(dict) in
7         mk-Properties0(lit, ordtyping ⇀ oplist ', ordaxioms ⇀ axioms ⇀ order, map, init))
8        else
9         exit("§2.2.2: Операторы упорядочивания многократно определены"))
10  else
11    mk-Properties0(lit, oplist, axioms, map, init))
```

запись : $Properties_0\ Sortqual \rightarrow Dict \rightarrow Properties_0$

Цель Расширить директиву упорядочивания в некоторых свойствах в определения операторов и аксиом, как определено в § 5.4.1.8 в Z.100

Параметры Свойства состоят из

lit	Определения литералов
oplist	Определения операторов
axioms	Аксиомы
map	Аксиомы отображения
init	Значение по умолчанию
tqual	Обозначает сорт, определяющий свойства

Результат Измененные свойства

Строка 1	Если в определениях операторов существует директива упорядочивания, то
Строка 2	Построить список определений операторов, в котором была перемещена директива упорядочивания.
Строка 3	В списке операторов должна быть только одна такая директива.
Строка 4	Построить AS_0 свойства операторов упорядочивания.
Строка 5	Если есть литералы, определенные для этого сорта, то построить аксиомы, связанные с упорядочиванием литералов.
Строка 6	Построить определения операторов упорядочивания.
Строка 7	Возвратить измененные свойства

as$_0$*-ordering-axioms*(*squal*)(*dict*) ≜ (3.4.7.1.22)

```
1   (let bqual = get-predef-sort("BOOLEAN")(dict) in
2   let falseid = mk-Id0(bqual, mk-Name0("FALSE", nil)) in
3   let sid = as0-id(squal) in
4   let anm = create-unique-name(),
5       bnm = create-unique-name(),
6       cnm = create-unique-name() in
7   let aid = mk-Id0(⟨⟩, anm),
8       bid = mk-Id0(⟨⟩, bnm),
9       cid = mk-Id0(⟨⟩, cnm) in
10  let axioml =
11      ⟨mk-Equation0(mk-Infixterm0(aid, LT, aid), falseid),
12       mk-Equation0(mk-Infixterm0(aid, LT, bid), mk-Infixterm0(bid, GT, aid)),
13       mk-Equation0(mk-Infixterm0(aid, LE, bid), mk-Infixterm0(mk-Infixterm0(aid, LT, bid)),
14                                                  OR,
15                                                  mk-Infixterm0(aid, EQ, bid))),
16       mk-Equation0(mk-Infixterm0(aid, GE, bid), mk-Infixterm0(mk-Infixterm0(aid, LE, bid),
17                                                  OR,
18                                                  mk-Infixterm0(aid, EQ, bid))),
19       mk-Infixterm0(mk-Infixterm0(aid, LT, bid),
20                     IMPLY,
21                     mk-Monadexpr0(NOT, mk-Infixterm0(aid, LT, bid))),
22       mk-Infixterm0(mk-Infixterm0(mk-Infixterm0(aid, LT, bid),
23                                 AND, mk-Infixterm0(bid, LT, cid)),
24                     IMPLY,
25                     mk-Infixterm0(aid, LT, cid))) in
26  ⟨mk-Quantequation0(⟨anm, bnm, cnm⟩, sid, axioml)⟩)
```

запись : *Sortqual* → *Dict* → *Axiom*$_0$

Цель Построить аксиому AS$_0$, отражающую свойства операторов упорядочивания, т.е.

```
FOR ALL aid, bid, cid IN sid
(aid < aid   == False;
 aid < bid   == bid > aid;
 aid <= bid == (aid < bid) OR (aid = bid);
 aid >= bid == (aid > bid) OR (aid = bid);
 aid < bid   => not(bid < aid);
 aid < bid   and bid < cid => aid < cid);
```

Алгоритм

Строка 11	Построить 1-ое равенство.
Строка 12	Построить 2-ое равенство.
Строка 13	Построить 3-е равенство.
Строка 16	Построить 4-ое равенство (т.е. булевскую аксиому).
Строка 19	Построить 5-ое равенство (т.е. булевскую аксиому).
Строка 22	Построить и возвратить квантифицированное равенство, содержащее пять равенств.

as_0-ordering-typing(tqual)(dict) ≜ (3.4.7.1.23)

```
1   (let tid = as0-id(tqual),
2        bid = as0-id(get-predef-sort("BOOLEAN")(dict)) in
3    ⟨mk-Opspec0(LT, ⟨tid, tid⟩, bid),
4     mk-Opspec0(GT, ⟨tid, tid⟩, bid),
5     mk-Opspec0(LE, ⟨tid, tid⟩, bid),
6     mk-Opspec0(GE, ⟨tid, tid⟩, bid)⟩)
```

запись : $Name_0 \rightarrow Dict \rightarrow Opspec_0^*$

Цель Построить AS_0 определения операторов упорядочивания, т.е.

```
"<"  :  tid, tid -> bid;
">"  :  tid, tid -> bid;
"<="  :  tid, tid -> bid;
">="  :  tid, tid -> bid;
```

as_0-order(lit, litlist, squal) ≜ (3.4.7.1.24)

```
1   (if litlist = ⟨⟩ then
2      ⟨⟩
3    else
4      (if is-Nmclass0(lit) then
5         as0-order(hd litlist, tl litlist, squal)
6       else
7         if is-Nmclass0(hd litlist) then
8           as0-order(lit, tl litlist, squal)
9         else
10          (let id1 = if is-Name0(lit) then
11                         mk-Id0(squal, lit)
12                       else
13                         mk-Stringterm0(squal, lit),
14               id2 = if is-Name0(hd litlist) then
15                         mk-Id0(squal, hd litlist)
16                       else
17                         mk-Stringterm0(squal, hd litlist) in
18          ⟨mk-Infixterm0(id1, LT, id2)⟩ ⌢
19          as0-order(hd litlist, tl litlist, squal))))
```

запись : $Literal_0\ Literal_0^*\ Sortqual \rightarrow Axiom_0^*$

Цель Построить аксиомы, отражающие упорядочивание литералов сорта

Параметры

lit	Первый литерал в списке литералов
litlist	Оставшиеся литералы в списке литералов (функция рекурсивна)
squal	*Qual* сорта, определяющего литералы

Результат Список аксиом в виде:

id1 < id2

Алгоритм

Строка 1	Когда просмотр окончен, ничего не возвращать
Строки 4-5	Если обрабатываемый литерал является классом имен, тогда его пропустить.
Строки 7-8	Если следующий литерал является классом имен, тогда его пропустить.
Строки 10-14	Построить AS_0 идентификатор обрабатываемого литерала и следующего литерала в списке. Заметьте, что полный квалификатор используется для того, чтобы избежать неоднозначности.

Строки 18-19 Возвратить булевскую аксиому для этого литерала, соединенную с булевскими аксиомами для остальных литералов

transform-typing(*oplist*)(*dict*) ≜ (3.4.7.1.25)

```
1   (if oplist = ⟨⟩ then
2      ({},[])
3    else
4      (let mk-Opspec0(nm, opsortl, ressort) = hd oplist in
5      let (oprest, drest) = transform-typing(tl oplist)(dict) in
6      let tqual = get-visible-qual(ressort, TYPE)(dict) in
7      let pqual = get-parent(tqual)(dict) in
8      let as1id = make-as1-identifier(tqual)(dict) in
9      let arglist = ⟨get-visible-qual(opsortl[i], TYPE)(dict) | 1 ≤ i ≤ len opsortl⟩ in
10     let parglist = ⟨get-parent(arglist[i])(dict) | 1 ≤ i ≤ len arglist⟩ in
11     let opparml = ⟨make-as1-identifier(arglist[i])(dict) | 1 ≤ i ≤ len arglist⟩ in
12     let opqual = dict(SCOPEUNIT) ⇀ ⟨(OPERATOR, (nm, parglist, pqual))⟩ in
13     let newnm = create-unique-name() in
14     let newqual = dict(SCOPEUNIT) ⇀ ⟨(OPERATOR, (newnm, parglist, pqual))⟩ in
15     let descr = mk-OperatorD(arglist, tqual, newqual, true) in
16     let as1tree = mk-Operator-signature1(name-to-name1(newnm), opparml, as1id) in
17     if opqual ∈ dom drest then
18        exit("§2.2.2 : Многократное появление определения оператора")
19       else
20        ({as1tree} ∪ oprest, drest + [opqual ↦ descr])))
```

запись : $Opspec_0^* \rightarrow Dict \rightarrow Operator\text{-}signature_1\text{-}\mathbf{set}\ Dict$

Цель Преобразовать список определений операторов в AS_1 и модифицировать *Dict*, чтобы включить в него дескрипторы для операторов

Результат Определение операторов в AS_1 и измененный *dict*

Параметры

oplist Список определений операторов AS_0

Алгоритм

Строка 1 Когда просмотр закончен, возвратить пустые объекты (функция рекурсивна).

Строка 4 Произвести декомпозицию первого определения оператора в списке.

Строка 5 Преобразовать остальные определения операторов.

Строки 6-7 Извлечь *Qual* результирующего сорта и привлечь родительские сорты (*parglist*) в случае подтипа.

Строка 8 Построить идентификатор AS_1 результирующего сорта.

Строки 9-10 Извлечь *Qual* сортов аргументов и привлечь родительские сорты (*parglist*) в случае любого подтипа.

Строка 11 Построить список идентификаторов AS_1 сортов аргументов.

Строка 12 Построить *Quals* оператора. Заметьте, что *Operatorqualelem* (см. определение домена) содержит родительские сорты (не подтипы), такие, что уникальность операторов может быть проверена проверкой *Quals* оператора.

Строка 13 Построить уникальное имя оператора. (Уникальные имена генерируются для всех операторов так, что не возникает перегрузки в AS_1).

Строка 14 Построить *Qual* оператора, который используется для преобразования в AS_1 применяемых результатов оператора.

Строка 15 Для оператора построить дескриптор *Dict*. В этом случае аргументы и результат могут быть подтипами.

Строка 16 Построить AS_1 сигнатуру оператора.

Строка 17	Если *Qual* оператора появляется в требовании *Dict* для оставшихся операторов, тогда два оператора с одним именем, с одними сортами аргументов и одним результатом определяются в одном определении сорта.
Строка 20	Возвратить сигнатуру оператора, унифицированную сигнатурами оставшихся операторов, и возвратить требования *Dict* для этого оператора и оставшихся операторов

3.4.7.2 Определения структуры

transform-struc(*nm*, mk-$Struc_0$(*fl*), mk-$Properties_0$(*l*, *o*, *a*, *ann*, *init*,))(*dict*) ≜ (3.4.7.2.1)

1 (let *sid* = mk-Id_0(*dict*(SCOPEUNIT), *nm*) in
2 let (*o'*, *a'*,) = *transform-fields* (*sid*, *fl*, {}, ⟨⟩) in
3 let *prop'* = mk-$Properties_0$(*l*, *o* ⇝ *o'*, *a* ⇝ *a'*, *am*, *init*) in
4 *transform-sortdef*(*nm*, *prop'*, nil)(*dict*))

запись : $Name_0\ Struc_0\ Properties_0 \rightarrow Dict \rightarrow Dict$

Цель Преобразовать определение сорта STRUCT в AS_1

Параметры

nm	Имя сорта, определенного ранее
fl	Список полей
l, o, a, am, init	Литералы, операторы, аксиомы, аксиомы отображения и значения по умолчанию, специфицированные в конструкции ADDING

Результат *Dict*, модифицированный, чтобы включить дескрипторы для литералов и операторов, и чтобы включить порожденные объекты AS_1 в его вход DATATYPEDEF

Алгоритм

Строка 1	Построить идентификатор AS_0 сорта, определенного ранее.
Строка 2	Сгенерировать AS_0 определения операторов и аксиомы, подразумеваемые определением структуры.
Строка 3	Построить свойства AS_0, содержащие подразумеваемые и дополнительные свойства.
Строка 4	Преобразовать эти свойства обычным способом

transform-fields (*tid*, *flist*, *fset*, *sidlist*) ≜ (3.4.7.2.2)

```
1   (if flist = ⟨⟩ then
2      (⟨mk-Opspec0(mk-Name0("MAKE", EXCLAMATION), sidlist, tid)⟩, ⟨⟩, len sidlist)
3    else
4      (let mk-Fieldspec0(fnml, fid) = hd flist in
5       let nm = hd fnml in
6       if nm ∈ fset then
7          exit("§5.4.1.10: У двух полей структуры одинаковые имена")
8       else
9          (let flist ' = if tl fnml = ⟨⟩ then
10                            tl flist
11                         else
12                            ⟨mk-Fieldspec0(tl fnml, fid)⟩ ⌢ tl flist in
13         let (as0oprest, as0axrest, argnum) =
14             transform-fields(tid, flist ', fset ∪ {nm}, sidlist ⌢ fid) in
15         let (as0op, as0ax) = transform-field (tid, fid, nm, argnum, len sidlist + 1) in
16         (as0op ⌢ as0oprest, as0ax ⌢ as0axrest, argnum))))
```

запись : $Sortid_0\ Fieldspec_0{}^*\ Fieldname_0\text{-}\mathbf{set}\ Sortid_0{}^* \rightarrow Op_0{}^*\ Axiom_0{}^*\ N_1$

Цель Преобразовать список полей из определения структуры AS_0 в список определений операторов и список аксиом

Параметры

tid	Идентификатор AS_0 сорта, определяющий структуру
flist	Список полей
fset	Множество уже используемых имен полей (функция рекурсивна)
sidlist	Текущий список сортов для полей. Этот список используется для построения сигнатуры оператора MAKE!

Результат AS_0 определения операторов, аксиомы введенные списком полей и полное число полей. Это число только для внутреннего использования функции

Алгоритм

Строки 1-2	Когда просмотр закончен, то *sidlist* полностью построен и может быть построена сигнатура оператора MAKE!. Длина *sidlist* тоже возвращается, так что аксиомы, включающие в себя оператор MAKE!, построенные на других уровнях рекурсии, могут быть корректно созданы.
Строка 4	Произвести декомпозицию первого поля спецификации.
Строка 5	Пусть *nm* обозначает первое имя в списке имен полей спецификации первого поля.
Строка 6	Если имя уже было использовано, то это - ошибка.
Строка 9	Построить новый список полей, из которого исключено обрабатываемое имя.
Строка 13	Преобразовать оставшиеся поля.
Строка 15	Для обрабатываемого поля построить определения операторов и аксиомы
Строка 16	Возвратить свойства для обрабатываемого поля, объединенные со свойствами оставшихся полей

transform-field (*tid*, *fid*, *fnm*, *totnum*, *index*) ≜ (3.4.7.2.3)

```
1   (let (modop, modax) = as0-modify-properties(tid, fid, fnm, totnum, index),
2        (extop, extax) = as0-extract-properties(tid, fid, fnm, totnum, index) in
3   (⟨modop ⌢ extop⟩, ⟨modax ⌢ extax⟩))
```

запись : $Sortid_0\ Sortid_0\ Fieldname_0\ N_1\ N_1 \rightarrow Op_0{}^*\ Axiom_0{}^*$

Цель Построить подразумеваемые определения операторов и аксиомы, связанные с именем поля

Параметры

tid	AS_0 идентификатор сорта, определяющего структуру
fid	AS_0 идентификатора сорта полей
fnm	Имя поля
totnum	Число полей для сорта *tid*
index	Позиция поля в списке аргументов оператора MAKE!, связанного с *tid*

Результат Определения операторов и аксиомы, связанные с полем *fnm*

Алгоритм

Строка 1	Построить определение оператора и аксиому для оператора модификации, связанного с полем.
Строка 2	Построить определение оператора и аксиому для оператора извлечения, связанного с полем.
Строка 3	Возвратить два определения операторов и две аксиомы

as_0-*modify-properties*(*tid*, *fid*, mk-$Name_0$(*fst*,), *totlistlength*, *index*) ≜ (3.4.7.2.4)

```
1   (let mnm = mk-Name0(fst ⌢ "MODIFY", EXCLAMATION) in
2   let mk-Id0(qualifier, tnm) = tid in
3   let tqual = qualifier ⌢ ⟨(TYPE, tnm)⟩ in
4   let opspec = mk-Opspec0(mnm, ⟨tid, fid⟩, tid) in
5   let fvarid = mk-Id0(⟨⟩, create-unique-name()) in
6   let makearglist = ⟨mk-Id0(⟨⟩, create-unique-name()) | 1 ≤ i ≤ totlistlength⟩ in
7   let makearglist' = ⟨makearglist[i] | 1 ≤ i ≤ index⟩ ⌢
8       ⟨fvarid⟩ ⌢
9       ⟨makearglist[i] | index < i ≤ totlistlength⟩ in
10  let makenm = mk-Name0("MAKE", EXCLAMATION) in
11  let makeop = mk-Operatorterm0(mk-Id0(tqual, makenm), makearglist),
12      makeop' = mk-Operatorterm0(mk-Id0(tqual, makenm), makearglist') in
13  let lterm = mk-Operatorterm0(mk-Id0(tqual, mnm), ⟨makeop, fvarid⟩) in
14  let ax = mk-Equation0(lterm, makeop') in
15  (opspec, ax))
```

запись : $Sortid_0\, Sortid_0\, Fieldname_0\, N_1\, N_1 \rightarrow Op_0\, Axiom_0$

Цель Построить свойства AS_0, чтобы изменить оператор, связанный с полем структуры

Параметры

tid	Идентификатор сорта, определяющего структуру
fid	Идентификатора сорта полей
fst	Проверка имени поля
totlistlength	Число полей, определенных для сорта *tid*
index	Позиция поля в списке аргументов оператора MAKE!, связанного с *tid*

Результат Подразумеваемое определение оператора и аксиома, как определено в §5.4.1.10 Z.100

Алгоритм

Строка 1	Построить имя оператора модификации посредством конкатенации проверки имени поля при помощи строки "MODIFY".

Строки 2-3	Построить квалификатор (*tqual*), обозначающий сорт *tid*. Оператор MAKE! квалифицируется при помощи *tqual*, чтобы избежать неоднозначности.
Строка 4	Построить определение оператора для оператора модификации поля.
Строки 5-6	Построить отличный идентификатор значения (*fvarid*) и создать список отличных идентификаторов значений, которые будут использоваться в качестве аргумента (левого) оператора MAKE! (см. пример в §5.4.1.10 Z.100).
Строки 6-9	Изменить список аргументов посредством замещения идентификатора значения, стоящего на месте *index*, на *fvarid*. Этот список аргументов используется правой частью аксиомы.
Строка 10	Построить $Name_0$ оператора MAKE!.
Строка 11	Сформировать приложение оператора AS_0 для левой части оператора MAKE!.
Строка 12	Сформировать приложение оператора AS_0 для правой части терма.
Строка 13	Построить терм левой части.
Строки 14-15	Возвратить определение оператора и построенную аксиому

as_0-*extract-properties*(*tid*, *fid*, **mk**-$Name_0$(*fst*,), *totlistlength*, *index*) ≜ (3.4.7.2.5)

1 (**let** *enm* = **mk**-$Name_0$(*fst* ⌢ "EXTRACT", EXCLAMATION) **in**
2 **let** **mk**-Id_0(*qualifier*, *tnm*) = *tid* **in**
3 **let** *tqual* = *qualifier* ⌢ ⟨(TYPE, *tnm*)⟩ **in**
4 **let** *opspec* = **mk**-$Opspec_0$(*enm*, ⟨*tid*⟩, *fid*) **in**
5 **let** *makearglist* = ⟨**mk**-Id_0(⟨⟩, *create-unique-name*()) | $1 \le i \le$ *totlistlength*⟩ **in**
6 **let** *makenm* = **mk**-$Name_0$("MAKE", EXCLAMATION) **in**
7 **let** *makeop* = **mk**-$Operatorterm_0$(**mk**-Id_0(*tqual*, *makenm*), *makearglist*) **in**
8 **let** *lterm* = **mk**-$Operatorterm_0$(**mk**-Id_0(*tqual*, *enm*), ⟨*makeop*⟩) **in**
9 **let** *ax* = **mk**-$Equation_0$(*lterm*, *makearglist*[*index*]) **in**
10 (*opspec*, *ax*))

запись : $Sortid_0 Sortid_0 Fieldname_0 N_1 N_1 \rightarrow Op_0 Axiom_0$

Цель Построить свойства AS_0 для оператора извлечения, связанного с полем структуры

Параметры

tid	Идентификатор сорта, определяющего структуру
fid	Идентификатора сорта полей
fst	Проверка имени поля
totlistlength	Число полей, определенных для сорта *tid*
index	Позиция поля в списке аргументов оператора MAKE!, связанного с *tid*

Результат Подразумеваемое определение оператора и аксиома, как определено в §5.4.1.10 Z.100

Алгоритм

Строка 1	Построить имя оператора извлечения посредством конкатенации проверки имени поля при помощи строки "EXTRACT".
Строки 2-3	Построить квалификатор (*tqual*), обозначающий сорт *tid*.
Строка 4	Построить определение оператора для оператора извлечения поля.
Строки 5-6	Построить отличный идентификатор значения (*fvarid*) и создать список отличных идентификаторов значений, которые будут использоваться в качестве аргумента (левого) оператора MAKE! (см. пример в §5.4.1.10 Z.100).
Строки 6-7	Построить приложение оператора AS_0 для левой части оператора MAKE!.
Строка 8	Построить терм левой части.
Строки 10-11	Возвратить определение оператора и построенную аксиому.

3.4.8 Определения таймеров

transform-timerdef(mk-*Timerdef*$_0$(*vlist*))(*dict*) ≜ (3.4.8.1)

```
1   if (∃squal ∈ dom dict)(is-ServiceD(dict(squal)) ∧ get-sur(squal) = dict(SCOPEUNIT)) then
2     exit("§4.10.2: В процессе содержатся определения сервиса и таймеров")
3    else
4     (let mk-Timerelem0(tnm, sortlist) = hd vlist in
5      let tqual = dict(SCOPEUNIT) ⌢ ⟨(SIGNAL, tnm)⟩ in
6      let unm = if is-ServiceD(dict(dict(SCOPEUNIT))) then create-unique-name() else tnm in
7      let nqual = dict(SCOPEUNIT) ⌢ ⟨(SIGNAL, unm)⟩ in
8      let squallist = ⟨get-visible-qual(sortlist[i], TYPE)(dict) | i ∈ ind sortlist⟩ in
9      let as1idlist = ⟨make-as1-identifier(squallist[i])(dict) | i ∈ ind squallist⟩ in
10     let as1tree = mk-Timer-definition1(name-to-name1(unm), as1idlist),
11        delem = [tqual ↦ mk-TimerD(squallist, nqual) | 1 ≤ i ≤ len squallist] in
12     if tl vlist = ⟨⟩ then
13       (⟨as1tree⟩, delem)
14      else
15       (let (as1rest, drest) = transform-timerdef(mk-Timerdef0(tl vlist))(dict) in
16        ({as1tree} ∪ as1rest, delem + drest)))
```

запись : *Timerdef*$_0^+$ → *Dict* → *Timer-definition*$_1$-set *Dict*

Цель Преобразовать в AS_1 определение таймера

Параметры

vlist Список определений (элементарных) таймеров, содержащихся в определении (составного) таймера

Результат Множество определений таймеров AS_1 и требования *Dict*, содержащие дескрипторы таймеров

Алгоритм

Строки 1-2 Процесс не должен содержать никаких сервисов.

Строка 4 Произвести декомпозицию следующего определения элементарного таймера (функция рекурсивна).

Строка 5 Построить *Qual* для таймера.

Строки 6-7 Построить уникальное имя и уникальный *Qual*, который используется для извлечения идентификатора AS_1 (см. определение *Newqual*)

Стоки 8-9 Построить список сортов *Quals*, связанных с таймером, и построить соответствующий список идентификаторов сорта AS_1.

Строка 10 Построить определение таймера AS_1 для обрабатываемого таймера

Строка 11 Построить требование *Dict* для обрабатываемого таймера

Строка 12 Если это элементарное определение таймера является последним элементом в списке, то возвратить определение AS_1 и требование *Dict*, иначе

Строки 15-16 Объединить их с определениями и требованием *Dict*, соответствующим оставшимся определениям элементарного таймера

3.4.9 Определения переменных

transform-vardef(mk-*Vardef*$_0$(*revatt*, *expatt*, *vlist*))(*dict*) ≜ (3.4.9.1)

```
1   (let mk-Vardefelem0(vl, tp, expr) = hd vlist in
2    let tqual = get-visible-qual(tp, TYPE)(dict) in
3    let pq = get-parent(tqual)(dict) in
4    let expr' = if expr = nil then
5                cases dict(tqual):
6                 (mk-SortD(,, expr1, )    → expr1,
```

```
7                 mk-SyntypeD(,, expr1, ) → expr1)
8               else
9                (let (expr1,, ) = transform-expr(expr, CONSTANT, {pq})(dict) in
10                expr1) in
11   let vnum = len vl in
12   let scope = dict(SCOPEUNIT) in
13   let nmlist = ⟨if is-ServiceD(dict(scope)) then create-unique-name() else vl[i] | 1 ≤ i ≤ vnum⟩ in
14   let quallist = ⟨scope ⇝ ⟨(VALUE, vl[i])⟩ | 1 ≤ i ≤ vnum⟩,
15       newquallist = ⟨scope ⇝ ⟨(VALUE, nmlist[i])⟩ | 1 ≤ i ≤ vnum⟩ in
16   let (decll, dexport) = if expatt = nil then
17                            ({},[])
18                           else
19                            build-exported-vardef(vl, tqual, expr)(dict) in
20   let (as1set, delem) =
21       (let as1tp = make-as1-identifier(tqual)(dict) in
22        let as1dl = {mk-Variable-definition1(name-to-name1(nmlist[i]), as1tp) revatt) | i ∈ ind vnum},
23            d = [quallist[i] ↦ mk-VarD(tqual, revatt, expatt, expr', newquallist[i]) | 1 ≤ i ≤ vnum] in
24        (as1dl, d)) in
25   if ¬is-ProcessD(dict(scope)) ∧ (revatt ≠ nil ∨ expatt ≠ nil) then
26      exit("§2.4.5: Переменная в процедуре (или сервисе) не может быть EXPORTED или REVEALED")
27     else
28      if card elems vl ≠ len vl then
29        exit("§2.2.2: Два определения в одной единице подразделения используют одно имя" )
30       else
31        if tl vlist = ⟨⟩ then
32          (as1set ∪ decll, delem + dexport)
33         else
34          (let (as1rest, drest) =
35               transform-vardef(mk-Vardef0(revatt, expatt, tl vlist))(dict) in
36           if dom delem ∩ dom drest ≠ {} then
37             exit("§2.2.2: Два определения в одной единице подразделения используют одно имя" )
38            else
39             (as1set ∪ decll ∪ as1rest, delem + drest + dexport)))
```

запись : $Vardef_0 \rightarrow Dict \rightarrow Variable\text{-}definition_1$ -set *Dict*

Цель Преобразовать в AS_1 определение переменной

Параметры Определение переменной, содержащее:

revatt Необязательный атрибут REVEAL

exatt Необязательный атрибут EXPORT

vlist Список (элементарных) определений переменных

Результат Множество определений AS_1 и требование *Dict*, содержащее дескрипторы определенных переменных

Алгоритм

Строка 1 Произвести декомпозицию первого определения элементарной переменной,(функция рекурсивна).

Строка 2 Извлечь *Qual* их сорта.

Строка 3 Извлечь *Qual* ее родителя (в случае подтипа).

Строки 4-10 Если начальное выражение не специфицированно в определениях переменных, то извлечь AS_1 выражение по умолчанию для ее сорта (которое может и отсутствовать nil), иначе преобразовать специфицированное выражение.

Строка 11 Пусть *vnum* обозначает количество имен переменных, представленное в определении элементарной переменной.

Строка 12 Пусть scope обозначает *Qual* единицы подразделения, определяющей переменные.

Строка 13 Построить новые уникальные имена для переменных, если они определены в сервисе.

Строка 14	Построить *Quals* переменных, которые должны использоваться в качестве входов *Dict* для переменных.
Строка 15	Построить *Quals*, которые должны быть помещены в компоненту *Newqual* дескрипторов переменных (они отличаются от *Quals*, построенных в строке 14, если в сервисе).
Строки 16-19	Если переменные имеют атрибут EXPORT, то построить определения AS_1 (*decll*) для неявных переменных.
Строка 20	Построить определения переменных AS_1, требования *Dict* для переменных при помощи:
Строка 20	Построения идентификатора AS_1 для их сорта.
Строка 21	Для каждой переменной, которая строит определение переменной в AS_1, содержащее имя (*nmlist*[*i*]), ее сорт (as_1*tp*) и, возможно, атрибут раскрытия (*revatt*).
Строка 23	Для каждой переменной построить дескриптор *Dict*, содержащий *Qual* ее сорта (*tqual*), возможно атрибут раскрытия (revatt), возможно атрибут экспорта (*expatt*), возможно начальное выражение AS_1 (*expr'*) и (возможно новый) *Qual*, который будет использоваться в случае применения переменной.
Строки 25-26	До тех пор пока переменные определяются на уровне процесса, они не могут быть EXPORTED или REVEALED.
Строка 28	Имена в *vl* должны быть уникальными.
Строка 31	Если в определении составной переменной больше нет определений элементарных переменных, то возвратить построенные объекты, иначе
Строки 34-39	Объединить их с определениями AS_1 и требованиями *Dict* для оставшихся определений элементарных переменных при условии, что имена в определении переменной являются уникальными

build-exported-vardef(*varlist, tqual, expr*)(*dict*) ≜ (3.4.9.2)

```
1   if varlist = ⟨⟩ then
2     ({}, [])
3    else
4     (let nm = hd varlist in
5      let (vdrest, drest) = build-exported-vardef(tl varlist, tqual, expr)(dict) in
6      let id0 = as0-id(tqual) in
7      let mk-SignalnamesD(impnm,, ) = exportmap((tqual, nm)) in
8      let vardef0 = mk-Vardef0(nil, nil, ⟨mk-Vardefelem0(⟨impnm⟩, id0, expr)⟩) in
9      let (vardef1, dict') = transform-vardef(vardef0)(dict) in
10     (vdrest ∪ vardef1, drest + dict'))
```

запись : $Name_0^*$ *Qual* [$Expr_0$] → *Dict* → *Variable-definition*$_1$-set *Dict*

Цель Построить определения переменных AS_1 для объектов, подразумеваемых экспортируемыми переменными

Параметры

varlist	Список переменных, которые экспортируются
tqual	*Qual* сорта переменных
expr	Выражение, обозначающее начальное значение переменных

Результат Определения AS_1

Строка 1	Когда просмотр закончен, ничего не возвращать (функция рекурсивна).
Строка 4	Пусть nm обозначает первое (следующее) имя в списке переменных.
Строка 5	Построить определения AS_1 для оставшейся части списка переменных.
Строка 6	(Повторно) построить идентификатор AS_0 сорта переменных.

Строка 7	Произвести декомпозицию дескриптора экспорта (*SignalnamesD*), чтобы получить доступ к имени неявной переменной.
Строка 8	Построить определение переменной в AS_0 для этого неявного имени.
Строка 9	Преобразовать определение неявной переменной в AS_1.
Строка 10	Возвратить это определение AS_1, объединенное с определениями оставшихся экспортируемых переменных. Дескриптор *VarD* неявной переменной (содержащийся в *dict* ') включается в результирующий *Dict*.

3.4.10 Определения обозревания

transform-viewdef(mk-$Viewdef_0$(*vars*))(*dict*) ≜ (3.4.10.1)

```
1   (let mk-Viewdefelem0(varlist, tp) = hd vars in
2    if card elems varlist ≠ len varlist then
3      exit("§2.2.2: Два определения в одной единице подразделения используют одно имя")
4    else
5      (let tqual = get-visible-qual(tp, TYPE)(dict) in
6       let tqual ' = get-parent(tqual)(dict) in
7       let (as1set, d) = transform-view(varlist, tqual ')(dict) in
8       if tl vars = ⟨⟩ then
9         (as1set, d)
10      else
11        (let (as1rest, drest) = transform-viewdef(mk- Viewdef0(tl vars))(dict) in
12          if dom d ∩ dom drest ≠ {} then
13            exit("§2.2.2: Два определения в одной единице подразделения используют одно имя")
14          else
15            (as1set ∪ as1rest, d + drest))))
```

запись : $Viewdef_0 \rightarrow Dict \rightarrow View\text{-}definition_1$-set *Dict*

Цель Преобразовать определение обозревания в AS_1

Параметры

vars Список (элементарных) определений, содержащихся в (составном) определении обозревания

Результат Множество AS_1 определений обозреваний и требование *Dict*, содержащее дескрипторы для переменных обозревания

Алгоритм

Строка 1	Произвести декомпозицию первого (следующего) определения элементарного обозревания (функция рекурсивна).
Строка 2	Проверить, что имена в списке переменных являются уникальными.
Строка 5	Извлечь *Qual* их сорта.
Строка 6	Извлечь *Qual* родительского сорта в случае подтипа.
Строка 7	Преобразовать первое определение элементарного обозревания.
Строка 8	Если больше нет определений элементарных обозреваний, то возвратить определение AS_1 и требование *Dict* для этих переменных обозревания, иначе
Строки 11-15	Объединить их с определениями и требованиями для оставшихся определений элементарных обозреваний при условии, что имена в определении обозревания считаются уникальными.

transform-view(*varlist, tqual*)(*dict*) ≜ (3.4.10.2)

```
1  (if varlist = ◊ then
2     ({}, [])
3   else
4     (let mk-Id0(q, nm) = hd varlist in
5      let as1tid = make-as1-identifier(tqual)(dict),
6          (as1rest, drest) = transform-view(tl varlist, tqual)(dict) in
7      let qual = (let bqual = get-sur(dict (SCOPEUNIT)) in
8                  if (∃! vqual ∈ dom dict)
9                    ((get-sur(get-sur(vqual)) = bqual) ∧
10                    (∃q')(q' ⇝ q ⇝ ⟨(VALUE, nm)⟩ = vqual) ∧
11                     is-VarD(dict(equal)) ∧
12                     (let mk-VarD(tq, attr,,, ) = dict(vqual) in
13                      tq = tqual ∧ attr = REVEALED)) then
14                   (let vqual be s.t. (get-sur(get-sur(vqual)) = bqual ∧
15                              is-VarD(dict(vqual)) ∧
16                              (∃q')(q' ⇝ q ⇝ ⟨(VALUE, nm)⟩ = vqual) ∧
17                              (let mk-VarD(tq, attr,,, ) = dict(vqual) in
18                               tq = tqual ∧ attr = REVEALED)) in
19                    vqual)
20                 else
21                  exit("§2.6.1.2: Отсутствуют уникальные соответствующие переменные этого сорта с атрибутом раскрытия")) in
22     let as1id = make-as1-identifier(qual)(dict) in
23     let as1tree = mk-View-definition1(as1id, as1tid) in
24     ({as1tree} ∪ as1rest, drest + [dict(SCOPEUNIT) ⇝ ⟨(VIEW, qual)⟩ ↦ mk-ViewD(qual)]))))
```

запись: Id_0^* *Qual* → *Dict* → *View-definition*$_1$-**set** *Dict*

Цель Преобразовать список переменных обозрения в определения обозрения AS_1

Параметры

varlist	Переменные обозрения
tqual	*Qual* сорта переменных

Результат Определения AS_1

Алгоритм

Строка 1	Когда просмотр закончен, ничего не возвращать (функция рекурсивна).
Строка 4	Произвести декомпозицию первого идентификатора переменных.
Строка 5	Преобразовать сорт переменных в идентификатор AS_1.
Строка 6	Преобразовать оставшиеся переменные обозрения.
Строки 7 -21	Обозначить *Qual* соответствующей переменной с атрибутом раскрытия через *qual*.
Строка 7	Построить *Qual* блока окружения.
Строка 8	Должен присутствовать уникальный идентификатор переменных (т.е. *Qual* переменных), который определяется в процессе этого блока окружения,
Строка 10	который содержит квалификатор и имя, специфицированные в определении обозрения,
Строка 11	который является переменной,
Строка 13	которая относится к тому же сорту и обладает атрибутом раскрытия.
Строки 14 - 19	Если существует такой идентификатор, то пусть *vqual* обозначает его *Qual*.
Строка 22 - 23	Построить определение обозрения AS_1 и
Строка 24	Объединить с определениями, содержащими оставшиеся переменные обозрения (*as*$_1$*rest*). Также объединить дескриптора обозрения (*ViewD*) с дескрипторами оставшихся определений обозрения

3.4.11 Определения импорта

transform-importdef (mk-*Importdef*$_0$ (*importelemlist*))(*dict*) ≜ (3.4.11.1)

```
1  (let mk-Importelem0(varl, tid) = hd importelemlist in
2   let tqual = get-visible-qual(tid, TYPE)(dict) in
3   let pqual = get-parent (tqual)(dict) in
4   let scope = dict (SCOPEUNIT) in
5   let delem = [scope ⇀ ⟨(IMPORT, (varl [i], pqual ))⟩ ↦ mk-ImportD(tqual) | 1 ≤ i ≤ len varl] in
6   if tl importelemlist = ⟨⟩ then
7      ({}, delem)
8    else
9      (let drest = transform-importdef(tl importelemlist)(dict) in
10      if dom delem ∩ dom drest ≠{} then
11         exit("§2.2.2: Два определения в одной единице подразделения используют одно имя")
12       else
13         ({}, delem + drest)))
```

запись: *Importdef*$_0$ → *Dict* → *Decl*$_1$ -set *Dict*

Цель Построить требование *Dict* для импортируемого объекта.

Параметры Определение импорта, содержащее:

importelemlist Список (элементов) определений импорта

Результат Требования *Dict* для переменных импорта и пустое множество (см. *transform-signallistdef*).

Строка 1 Произвести декомпозицию первого элементарного определения импорта (функция рекурсивна).

Строка 2 Извлечь *Qual* сорта, содержащихся переменных (*varl*).

Строка 3 Извлечь родительский сорт в случае подтипа.

Строки 4 - 5 Построить требования *Dict* для содержащихся переменных. Заметьте, что *pqual* обычно используется в *Qual* для того, чтобы достигнуть уникальность пары имя импорта-сорт, а *tqual* используется в дескрипторе, т.к. неявные переменные, связанные с выражениями импорта, могут быть взяты из подтипов.

Строки 6 -9 И объединить их с требованиями для оставшихся определений элементов обозревания (если есть).

Строка 10 Пары имя импорта и сорт должны быть уникальными

3.4.12 Определения маршрута сигнала

transform-signalroutedef (mk-*Sigroutedef*$_0$(*nm, path, opath*))(*dict*) ≜ (3.4.12.1)

```
 1  (let mk-Sigroutepath0(endpoint1, endpoint2, siglist) = path in
 2   let sigroutequal = dict(SCOPEUNIT) ⇝ ⟨(SIGNALROUTE, nm)⟩ in
 3   let sigset = transform-signallist(siglist)(dict) in
 4   let p1 = get-visible-qual(endpoint1, PROCESS)(dict),
 5       p2 = get-visible-qual(endpoint2, PROCESS)(dict) in
 6   let osigset =
 7       if opath = nil then
 8           {}
 9           else
10           (let mk-Sigroutepath0(orig', dest', osiglist) = opath in
11            let p1' = get-visible-qual(orig', PROCESS)(dict),
12                p2' = get-visible-qual(dest', PROCESS)(dict) in
13            if  p1 = p2' ∧ p2 = p1' then
14                transform-signallist(osiglist)(dict)
15             else
16                exit ("§2.5.2:Второй путь должен обозначать обратное направление первого пути")
17   if p1 ≠ p2 ∧ is-local (p1)(dict) ∧ is-local (p2)(dict) then
18     (let as1 p1 = if p1 = ENV then ENVIRONMENT else make-as1-identifier(p1)(dict),
19          as1 p2 = if p2 = ENV then ENVIRONMENT else make-as1-identifier(p2)(dict),
20          as1 sigidset = make-as1 idset (sigset)(dict),
21          as1 sigidset' = make-as1 idset (sigset)(dict) in
22      let as1 path1 = mk-Signal-route-path1(as1, p1, as1p2, as1sigidset),
23          as1 path2 = if as1 sigidset' = {} then
24                          nil
25                        else
26                          mk-Signal-route-path1(as1p2, as1p1, as1sigidset'),
27          descr = mk-SignalrouteD(p1, p2, sigset, osigset) in
28      let as1tree = mk-Signal-route-definition1(name-to-name1 (nm), as1path1, as1path2) in
29      ({as1tree}, [sigroutequal ↦ descr]))
30    else
31     if p1 = p2 then
32       exit("§2.5.2:Конечныеточкимаршрутасигналанеявляютсяразличными")
33     else
34       exit("§2.5.2:Конечные точки маршрута сигнала не определены локально")
```

запись: *Sigroutedef*$_0$ → *Dict* → *Signal-route-definition*$_1$ -**set** *Dict*

Цель Преобразовать определение маршрута сигнала в AS_1

Параметры Определение маршрута сигнала в AS_0, содержащее:

nm	Имя маршрута сигнала
path	Первый путь
opath	Второй необязательный путь

Результат Множество, содержащее определение маршрута сигнала в AS_1, и требование *Dict*, содержащее дескриптор маршрута сигнала

Алгоритм

Строка 1	Произвести декомпозицию первого пути.
Строка 2	Построить *Qual*, обозначающий маршрут сигнала.
Строка 3	Преобразовать список сигналов по первому пути в множество *Quals* сигналов.
Строки 4-5	Извлечь *Qual* двух конечных точек первого пути (*get-visible-qual* может обрабатывать случай, когда конечная точка является ENV).
Строки 6-16	Пусть *osigset* обозначает множество *Qual* сигналов для противоположного направления (второй путь).

Строка 6	Если второй путь не указан, то он является пустым множеством, иначе
Строка 10	Произвести декомпозицию второго пути.
Строки 11-12	Извлечь *Qual* двух конечных точек второго пути.
Строка 13	Первая конечная точка первого пути должна соответствовать второй конечной точке второго пути, и вторая конечная точка первого пути должна соответствовать первой конечной точке второго пути
Строка 14	Пусть *osigset* обозначает множество *Quals* сигналов для второго пути.
Строка 17	Конечные точки маршрута сигнала должны быть различными, конечные точки должны быть определены локально.
Строки 18-19	Построить определение AS_1 (или ENVIRONMENT) этих двух конечных точек.
Строки 20-21	Построить два множества AS_1 сигналов из сигналов, передаваемых в двух направлениях.
Строки 22-23	Построить два пути AS_1 направлений.
Строка 27	Построить дескриптор маршрута сигнала.
Строка 28	Построить определение AS_1 маршрута сигнала.
Строка 29	Возвратить определение AS_1 и требование *Dict*, содержащее построенный дескриптор.
Строки 31-34	Для ясности разделить условие ошибки на два выхода exit.

3.4.13 Определения списка сигналов Signallist

transform-signallistdef(mk-*Signallistdef*$_0$(*nm, signals*))(*dict*) ≜ (3.4.13.1)

```
1  (let siglistqual = dict(SCOPEUNIT) ⇀ ⟨(SIGNALLIST, nm)⟩ in
2   let () =
3       transform-signallist(signals)(dict + [siglistqual ↦ mk-ErrorD()]) in
4  ({}, [siglistqual ↦ mk-SignallistD(signals)]))
```

запись: *Signallistdef*$_0$ → *Dict* → *Decl*$_1$-set *Dict*

Цель Построить требование *Dict* для определения списка сигналов

Параметры Определение списка сигналов в AS_0, содержащее:

nm	Имя списка сигналов
signals	Связанный список сигналов

Результат Помимо требования *Dict*, возвращается пустое множество определений AS_1 (так что функции "преобразования определений" также могут обрабатываться)

Алгоритм

Строка 1	Построить *Qual* списка сигналов.
Строка 3	Преобразовать список сигналов в множество *Qual* сигналов. Однако, результат не используется, так как функция применяется только для проверки рекурсивного определения.
Строка 4	Возвратить требование *Dict*, содержащее дескриптор списка сигналов

transform-signallist(*sigl*)(*dict*) ≜ (3.4.13.2)

```
 1  (if sigl = ◊ then
 2    {}
 3   else
 4    (let sigrest = if tl sigl = ◊ then {} else transform-signallist (tl sigl)(dict) in
 5     let sigsetelem =
 6        cases hd sigl:
 7         (mk-Id0(,)
 8            → {signal-qual(hd sigl)(dict)},
 9         mk-Signallistid0(id)
10            → (let qual = get-visible-qual(id, SIGNALLIST)(dict) in
11                let mk-SignallistD(siglist) = dict (qual) in
12                transform-signallist (siglist)(dict + [qual ↦ mk-ErrorD()]))) in
13    if sigsetelem ∩ sigrest ≠ {} then
14      exit("§2.5.5: Идентификаторы сигналов в списке сигналов не различаются")
15     else
16      sigsetelem ∪ sigrest))
```

запись: *Signallist*$_0$ → *Dict* → *Signalqual*-**set**

Цель Преобразовать список сигналов в множество *Quals*

Параметры

sigl Список сигналов

Алгоритм

Строка 4	Преобразовать все элементы списка, кроме первого (функция рекурсивна).
Строки 5-12	Пусть *sigsetelem* обозначает *Quals* первого элемента в списке.
Строка 7	Если первый элемент списка является идентификатором, то он обозначает сигнал и *sigsetelem* обозначает множество, содержащее *Qual* этого сигнала.
Строка 9	Если первый элемент списка является списком сигналов, то пусть *qual* обозначает *Qual* списка сигналов.
Строка 11	Пусть *siglist* обозначает список сигналов AS$_0$, связанный с идентификатором списка сигналов.
Строка 12	Преобразовать этот список сигналов, но объявить идентификатор этого списка сигналов недействительным, обозначив его как *ErrorD* в *Dict*.
Строка 13	Если любой *Qual* сигнала первого элемента также появляется в другом элементе, то это означает ошибку, иначе
Строка 16	Возвращается объединение множества первого элемента и множества оставшихся элементов списка

3.4.14 Состояния соединения

transform-block-connect(*decllist, sigrouteset, connectmap*)(*dict*) ≜" (3.4.14.1)

```
 1  (if decllist = ◯ then
 2     (let bqual = dict(SCOPEUNIT) in
 3      let outerchanset = {mk-ChannelD(endp1, endp2,,, newc} ∈ rng dict |
 4                          bqual ∈ {endp1, endp2} ∧ newc = nil} in
 5      if outerchanset ≠ dom connectmap then
 6        exit("§2.5.3 : К блоку не подсоединены каналы")
 7       else
 8        if sigrouteset = union rng connectmap then
 9          ({}, connectmap)
10         else
11          exit("§2.5.3 : Маршрут сигнала появляется не только в одном соединении" ))
12    else
13     (cases hd decllist:
14       (mk-Sigroutedef0(nm, mk-Sigroutepath0(end1, end2, ), )
15         → (let qset = if end1 = ENV ∨ end2 = ENV
16                        then {dict (SCOPEUNIT) ⇀ ⟨(SIGNALROUTE, nm)⟩}
17                        else {} in
18             let newset = sigrouteset ∪ qset in
19             transform-block-connect(tl decllist, newset, connectmap)(dict)),
20       mk-Connect0(, )
21         → (let (channel, routeset, connect) = block-connect (hd decllist)(dict) in
22             if channel ∈ dom connectmap then
23               exit("§2.5.3: Соединение канал-маршрут сигнала встретилось дважды")
24              else
25               if union rng connectmap ∩ routeset ≠ {} then
26                  exit("§2.5.3: Маршрут сигнала упоминается более чем в одном соединении")
27                 else
28                  (let nconnectmap = connectmap + [channel ↦ routeset] in
29                   let (restas1, restmap) =
30                       transform-block- connect (tl decllist, sigrouteset, nconnectmap)(dict) in
31                  ({connect} ∪ restas1, restmap))),
32       T → transform-block-connect(tl decllist, sigrouteset, connectmap)(dict))))
```

запись: *Decl*$_0$* *Qual*-set *BlockconnectionD* → *Dict* → *Channel-to-route-connection*$_1$ -set *BlockconnectionD*

Цель Преобразование определений соединений блока в множество AS_1 соединений канал - маршрут

Параметры

decllist	Список определений блока в AS_1
sigrouteset	Множество *Quals*, обозначающее маршруты сигналов, которые уже были обработаны (рекурсивно). Первоначально это множество пусто.
connectmap	Отображение *Quals*, обозначающее каналы, которые уже встречались в определении соединения, в *Quals* маршрута сигнала, к которому подсоединены каналы. Первоначально это отображение пусто.

Результат Построенные определения AS_1 соединений

Алгоритм

Строка 2	Пусть *bqual* обозначает блок
Строка 3	Пусть *outerchanset* обозначает каналы, имеющие блок в качестве одной из конечных точек. Нельзя включать каналы, которые были заменены на два новых канала (т.е. *newc* = nil).
Строки 5-6	Множество каналов, имеющих блок в качестве одной из конечных точек, должно соответствовать множеству каналов, задействованных в соединении в блоке.

Строка 8-11	Возврат, если множество *Quals* маршрута сигнала, полученное из определений маршрутов сигналов в блоке, совпадает с множеством, построенным при слиянии множеств, содержащихся в *connectmap*, т.е. все маршруты сигналов, задействованные в соединении, должны быть определены в блоке, а среда должна являться для них одной из конечных точек, и наоборот.
Строки 14-19	Если первое (следующее) определение в списке является определением маршрута сигнала, то включить *Qual* сигнала (содержащийся в *qset*) в *sigrouteset*, если одна из конечных точек есть ENV.
Строка 20	Если следующее определение есть соединение блока, то
Строка 21	Извлечь *Qual* канала, *Quals* маршрута сигнала и определение соединения AS_1 из соединения блока.
Строка 22	Если канал уже использовался в соединении блока, то это ошибка.
Строка 25	Каждый маршрут сигнала должен быть задействован только в одном соединении канал-маршрут сигнала.
Строка 28	Обновить схему соединений для включения обрабатываемой информации о соединениях.
Строка 32	Если следующее определение отличается от предыдущего, то перейти к остальным определениям.

block-connect(**mk-***Connect*$_0$(*chid, routeidlist*))(*dict*) ≜ (3.4.14.2)

```
1  (let cqual = get-visible-qual(chid, CHANNEL)(dict) in
2  let mk-ChannelD(, endpoint2, sigset1, sigset2, newc) = dict (cqual) in
3  let bqual = dict(SCOPEUNIT) in
4  let connectset = {get-visible-qual (routeidlist [i], SIGNALROUTE)(dict) | i ∈ ind routeidlist} in
5  let (insigset, outsigset) = if endpoint2 = bqual then (sigset1, sigset2) else (sigset2, sigset1) in
6  let cas1 id = make-as1-identifier(cqual)(dict),
7      sas1 set = {make-as1-identifier(cqual)(dict),
8  let cqual' = if news = nil then cqual else convert-channel-qual (news, bqual) in
9  if card connectset = len routeidlist ∧
10    is-wf-connectsignalroutes(connectset, insigset, outsigset, {}, {})(dict) then
11     (cqual', connectset, mk-Channel-to-route-connection1(cas1 id, sas1 set))
12    else
13     exit("§2.5.3:Соединение канал-маршрут сигнала не корректно"))
```

запись: *Connect*$_0$ → *Dict* → *Channelqual Qual*-**set** *Channel-to-route-connection*$_1$

Цель Извлечь *Qual* канала, множество *Quals* путей сигнала и определение соединения AS_1 из соединения каналов AS_0.

Параметры

chid	Идентификатор канала AS_0 в соединении
routeidlist	Идентификатор списка маршрутов сигнала AS_0

Результат *Qual* канала в ограничивающем блоке множество *Qual*, содержащее маршруты сигналов, присоединенные к этому каналу, и соединение AS_1.

Алгоритм

Строка 1	Извлечь *Qual* канала
Строка 2	Произвести декомпозицию дескриптора канала.
Строка 3	Обозначить *Qual* блока, содержащего определения соединений через *bqual*.
Строка 4	Создать множество *Quals*, обозначающее идентификаторы маршрута сигнала путей, содержащиеся в соединении канал-маршрут.
Строка 5	Обозначить через *insigset* входные сигналы, а через *outsigset* - выходные сигналы.

Строки 6-7	Построить идентификатор канала AS_1 и множество идентификаторов AS_1, содержащее маршруты сигналов.
Строка 8	Если канал имеет подструктуру, то использовать *Qual* другого подходящего канала.
Строка 9	Идентификатор маршрута сигнала не должен быть дважды задействован в определении соединения и
Строка 10	Сигналы в маршрутах сигналов должны быть задействованы в канале.
Строка 11	Возвратить *Qual* канала, множество *Quals* маршрутов сигналов и соединение AS_1 канал-маршрут

is-wf-connectsignalroutes (*connectset, insigset, outsigset, res*1, *res*2)(*dict*) ≜ (3.4.14.3)

```
 1  if connectset = {} then
 2    insigset = res1 ∧ outsigset = res2
 3  else
 4   ( let squal ∈ connectset in
 5     let mk-SignalrouteD (endpoint1, endpoint2, sset1, sset2) = dict (squal) in
 6     if ENV ∈ {endpoint1, endpoint2} then
 7       false
 8     else
 9       (let (inset, outset) = if endpoint1 = ENV then (sset1, sset2) else (sset2, sset1) in
10        is-wf-connectsignalroutes (connectset \ {squal}, insigset, outsigset,
11                                   inset ∪ res1, outset ∪ res2)(dict)))
```

запись: *Qual*-set *Signalqual*-set *Signalqual*-set *Signalqual*-set *Signalqual*-set → *Dict* → *Bool*

Цель Убедиться, что маршруты сигналов в соединении канал-маршрут имеют надлежащий интерфейс.

Параметры

connectset	(Оставшееся) множество *Quals* маршрутов сигналов (функция рекурсивна)
insigset	Совокупность *Quals* сигналов, подающихся в блок через канал, специфицированный в соединении канал - маршрут
outsigset	Совокупность *Quals* сигналов, выходящих из блока через канал, специфицированный в соединении канал-маршрут
res1,res2	Рекурсивно образованные совокупности входных и выходных сигналов, специфицированные в маршрутах сигналов. В конечном итоге, эти совокупности должны совпадать с соответствующей совокупностью, специфицированной для канала

Результат Собранная совокупность входных (*res1*) и выходных (*res2*) сигналов должна совпадать соответственно с входными и выходными сигналами, специфицированными в канале

Алгоритм

Строка 1	После окончания возвратить "истина", если все сигналы в канале были задействованы в маршруте сигнала.
Строка 4	Пусть *squal* обозначает *Qual* в совокупности маршрутов сигналов.
Строка 5	Произвести декомпозицию дескриптора *Dict* для маршрута сигнала
Строка 6	ENV должна быть одной из конечных точек для маршрута сигнала.
Строка 9	Пусть *inset* обозначает сигналы из дескриптора маршрута сигнала, поступающие в блок, а *outset* - сигналы, выходящие из блока.
Строка 10	Остальные маршруты сигналов в соединении должны иметь надлежащий интерфейс

transform-substructure-connect(*decllist, channelset, connectmap*)(*dict*) ≜ (3.4.14.4)

```
 1  (if decllist = ⟨⟩ then
 2     (let bqual = get-sur(dict(SCOPEUNIT)) in
 3      let outerchanset = {mk-ChannelD(endp2,,, newc) ∈ rng dict |
 4                          bqual ∈ {endp1, endp2} ∧ newc = nil in
 5      if card outerchanset > card dom connectmap then
 6        exit("§3.2.2: Для канала, соединенного с подструктурой блока, нет соединений")
 7       else
 8        if channelset = union rng connectmap then
 9          {}
10         else
11          exit("§3.2.2: Подканалы появляются не только в одном соединении"))
12   else
13    (cases hd decllist :
14     (mk-Chandef0(nm, mk-Chanpath0(orig, dest,),,, )
15      → (let qset = if orig = ENV ∨ dest = ENV
16                    then {dict(SCOPEUNIT) ↝ ((CHANNEL, nm))}
17                    else {} in
18         let newset = channelset ∪ qset in
19         transform-substructure-connect(tl decllist, newset, connectmap)(dict)),
20     mk-Connect0(, )
21      → (let (chan, chanset, connect) = transform-block-substructure-connect (hd decllist)(dict) in
22         if chan ∈ dom connectmap then
23           exit("§3.2.2: Соединение канала встретилось дважды")
24          else
25           if union rng connectmap ∩ chanset ≠ {} then
26             exit("§3.2.2: Подканал задействован более, чем в одном соединении")
27            else
28             (let nconnectmap = connectmap + [chan ↦ chanset] in
29              {connect} ∪
30              transform-substructure-connect (tl decllist, channelset, nconnectmap)(dict))),
31     T → transform-substructure-connect (tl decllist, channelset, connectmap)(dict))))
```

запись: *Decl*$_0$* *Channelqual*-set (*Channelqual* ⇸ *Channelqual*-set) → *Dict* → *Channel-connection*$_1$-set

Цель Создать соединения каналов, относящиеся к подструктуре блока

Параметры

decllist Определение AS$_0$ списка подструктуры блока вместе с соединениями AS$_0$, связывающими неявный входной-выходной канал

channelset, connectmap, В процессе рекурсивного прохода по списку определений *decllist*, имена каналов из определения каналов, поступающих из среды или в нее, собираются или хранятся в *channelset*, информация о соединениях собирается и хранится в *connectmap*. Когда *transform-substructure-connect* используется изначально, (в *transform-blockdef*), эти параметры пусты.

Результат Собранные соединения AS$_1$.

Алгоритм

Строка 2 Пусть *bqual* обозначает блок, окружающий подструктуру блока

Строка 3 Пусть *outerchanset* обозначает каналы, для которых блок является одной из конечных точек, и не имеющие подструктур.

Строки 5-6 Если каналов, для которых блок является одной из конечных точек, больше, чем каналов в соединениях каналов, значит нескольких соединений каналов недостает.

Строки 8-11	Возврат, если все каналы (для которых среда является одной из конечных точек) задействованы в соединении.
Строка 14	Если обрабатываемое определение является определением канала, то
Строки 15-18	Построить множество, состоящее из пустых множеств, если канал не ведет в среду или из нее, иначе - состоящее из *Qual* каналов.
Строка 19	Проход по оставшемуся списку определений с множеством, добавленным к *channelset*
Строка 21	Если полученное определение является определением канала, то преобразовать соединение в AS_1, получая *Qual* (*chan*) канала, множество *Quals* (*chanset*) каналов и соединение AS_1 *connect*.
Строка 22	Канал не должен содержаться более чем в одном соединении каналов.
Строка 25	Подканал должен быть задействован только в одном соединении каналов.
Строки 28-29	Проход по оставшемуся списку определений с информацией о соединениях, добавленной к *connectmap*.

transform-block-substructure-connect (**mk**-*Connect*$_0$(*chid, chidlist*))(*dict*) ≜ (3.4.14.5)

```
 1  (let cqual = get-visible-qual(chid, CHANNEL)(dict) in
 2   let mk-ChannelD(endpoint1, endpoint2, sigset1, sigset2, newc) = dict(cqual) in
 3   let bqual = get-sur(dict(SCOPEUNIT)) in
 4   if bqual ∈ {endpoint1, endpoint2} then
 5      (let connectset = {get-visible-qual(chidlist [i], CHANNEL)(dict) | i ∈ ind chidlist} in
 6       let (insigset, outsigset) = if endpoint2 = bqual then
 7                                   (sigset1, sigset2)
 8                                  else
 9                                   (sigset2, sigset1) in
10       let cas1 id = make-as1-identifier (cqual)(dict),
11           cas1 list = {make-as1-identifier(cq)(dict) | cq ∈ connectset} in
12       let cqual ' = if newc = nil then cqual else convert-channel-qual(newc, bqual) in
13       if card connectset = len chidlist ∧
14         is-wf-connectchannels(connectset, insigset, outsigset, {}, {})(dict) then
15          (cqual ', connectset, mk-Channel-connection1(cas1 id, cas1 list))
16         else
17          exit("§3.2.2: Соединение канала не является корректным"))
18  else
19   exit("§3.2.2: Соединение канала не является корректным"))
```

запись: *Connect*$_0$ → *Dict* → *Channelqual Channelqual*-set *Channel-connection*$_1$

Цель Преобразовать соединение канала в AS_1

Параметры Соединение канала, содержащее:

chid	Идентификатор канала в объемлющей единице подразделения
chidlist	Список каналов, присоединенных к *chid*

Результат *Qual* канала в объемлющей единице окружения, множество *Qual*, содержащее каналы, присоединенные к этому каналу, и соединение каналов AS_1

Алгоритм

Строка 1	Построить *Qual* из *chid*.
Строка 2	Произвести декомпозицию его дескриптора канала.
Строка 3	Пусть *bqual* обозначает *Qual* объемлющего блока.
Строка 4	Блок должен быть одной из конечных точек канала, задействованного в определении соединения.

Строка 5	Построить множество, содержащее *Quals* каналов, присоединенных к *chid*.
Строка 6	Пусть *insigset* обозначает совокупность *Qual* сигналов, поступающих в подструктуру через канал, а *outsigset* - совокупность *Qual* сигналов, выходящих из подструктуры через канал.
Строки 10-11	Построить идентификатор внешнего канала AS_1 и множество идентификаторов каналов AS_1, к которым он присоединен.
Строка 12	Если канал имеет подструктуру, то использовать *Qual* другого подходящего канала.
Строка 13	Канал не должен быть задействован дважды в соединении каналов и
Строка 14	Сигналы в каналах должны иметь надлежащий интерфейс для внешнего канала.
Строка 15	Если все хорошо, возвратить *Qual* внешнего канала, совокупность *Qual* каналов, к которому он подсоединен, и соединение каналов AS_1.

is-wf-connectchannels(connectset, insigset, outsigset, res1, res2)(dict) ≜ (3.4.14.6)

```
1  (if connectset = {} then
2     is-wf-refinement(insigset, outsigset, res1, res2)(dict)
3   else
4    (let cqual ∈ connectset in
5     let mk-ChannelD(endpoint1, endpoint2, sset1, sset2, ) = dict(cqual) in
6     if ENV ∉ {endpoint1, endpoint2} then
7        false
8       else
9        (let (inset, outset) = if endpoint1 = ENV then (sset1, sset2) else (sset2, sset1) in
10        is-wf-connectchannels(connectset \ {cqual}, insigset, outsigset, res1 ∪ inset, res2 ∪ outset)(dict))))
```

запись: *Channelqual*-set *Signalqual*-set *Signalqual*-set *Signalqual*-set *Signalqual*-set → *Dict* → *Bool*

Цель Убедиться, что каналы в соединении каналов имеют надлежащий интерфейс.

Параметры

connectset	(Оставшиеся) множество *Quals* каналов (функция рекурсивна)
insigset	Совокупность *Quals* сигналов, поступающих в подструктуру блока через подканал
outsigset	Совокупность *Quals* сигналов, выходящих из подструктуры блока через подканал
res1,res2	Рекурсивно образованные совокупности входных и выходных сигналов, специфицированные в подканалах.

Результат "Истина" при удачном завершении

Алгоритм

Строка 1	После завершения убедиться, что построенные совокупности сигналов (*res1*, *res2*) являются корректными уточнениями сигналов, специфицированных для выходного канала.
Строка 4	Пусть *squal* обозначает *Qual* в множестве подканалов.
Строка 5	Произвести декомпозицию дескриптора *Dict* для подканала
Строка 6	ENV должна быть одной из конечных точек для подканала.
Строка 9	Пусть *inset* обозначает сигналы из дескриптора подканала, поступающие в подструктуру блока, а *outset* - сигналы, выходящие из подструктуры блока.
Строка 9	Возвратить "истина", если остальные подканалы в соединении имеют надлежащий интерфейс

is-wf-refinement (*parentset*1, *parentset*2, *subset*1, *subset*2)(*dict*) ≜ (3.4.14.7)

```
1   (if (∃q ∈ subset1 ∪ subset2)(get-sur(q) ∈ parentset1) then
2      (let q ∈ subset1 ∪ subset2 be s.t. get-sur(q) ∈ parentset1 in
3       let parent = get-sur(q) in
4       let mk-SignalD(, sub1, sub2) = dict(parent) in
5       if q ∈ subset1 ∧ sub1 ⊆ subset1 ∧ sub2 ⊆ subset2 then
6         is-wf-refinement (parentset1 \ {parent}, parentset2, subset1 \ sub1, subset2 \ sub2)(dict)
7        else
8         false )
9    else
10    (if (∃q ∈ subset2 ∪ subset1)(get-sur(q) ∈ parentset2) then
11       (let q ∈ subset1 ∪ subset2 be s.t. get-sur(q) ∈ parentset2 in
12        let parent = get-sur(q) in
13        let mk-SignalD(, sub1, sub2) = dict(parent) in
14        if q ∈ subset2 ∧ sub1 ⊆ subset2 ∧ sub2 ⊆ subset1 then
15          is-wf-refinement (parentset1, parentset2 \ {parent}, subset1 \ sub2, subset2 \ sub1)(dict)
16         else
17          false)
18         else
19          parentset1 = subset1 ∧ parentset2 = subset2))
```

запись: *Signalqual*-set *Signalqual*-set *Signalqual*-set *Signalqual*-set → *Dict* → *Bool*

Цель Убедиться, что уточнение сигналов на границе подструктуры сделано надлежащим образом

Параметры

parentset1	Совокупность *Quals* входных сигналов, проходящих по внешнему каналу
parentset2	Совокупность *Quals* выходных сигналов, проходящих по внешнему каналу
subset1	Совокупность *Quals* входных сигналов, проходящих через подканалы
subset2	Совокупность *Quals* выходных сигналов, проходящих через подканалы

Результат "Истина" при удачном завершении

Алгоритм

Строка 1	Если существует сигнал, проходящий через подканал, имеющий входной родительский сигнал, проходящий через внешний канал, то производилось уточнение входного сигнала.
Строка 2	Извлечь любой подсигнал.
Строка 3	Извлечь сигнал, порождающий подсигнал.
Строка 4	Произвести декомпозицию дескриптора сигнала для доступа к его подсигналам *sub*1 и *sub*2.
Строка 5	Так как один из подсигналов специфицирован в подканалах, все подсигналы должны быть специфицированы, т.е. подсигналы в двух направлениях (*sub*1, *sub*2) должны быть подмножествами сигналов, специфицированных для подканалов.
Строка 6	Удалить родительский сигнал и его подсигналы из параметров перед дальнейшими уточнениями.
Строки 10-11	Делать то же, что в строках 1-6, там, где обнаружено присутствие выходного родительского сигнала.
Строка 19	Если (больше) нет уточнений, то входные сигналы, проходящие через внешний канал, должны совпадать с входными сигналами, проходящими через подканалы, а выходные сигналы, проходящие через внешний канал должны совпадать с выходными сигналами, проходящими через подканалы

service-connectmap(*decllist, sigrouteset, connectmap*)(*dict*) ≜ (3.4.14.8)

```
1  (if decllist = ⟨⟩ then
2     (let pqual = dict(SCOPEUNIT) in
3      let outersigrouteset = {mk-SignalrouteD(endp1, endp2,, ) ∈ rng dict | pqual ∈ {endp1, endp2}} in
4      if sigrouteset = {} ∧ connectmap = [] then
5        []
6       else
7        if card outersigrouteset > card dom connectmap
8          then exit("§4.10.2: В процессе утеряно соединение маршрута сигнала")
9          else if sigrouteset ≠ union rng connectmap
10               then exit("§4.10.2: Все маршруты сигналов сервиса должны быть задействованы в в соединении" )
11               else connectmap)
12  else
13    (cases hd decllist :
14      (mk-Sigroutedef0(nm, mk-Sigroutepath0(end1, end2, ), )
15        → (let qset = if end1 = ENV ∨ end2 = ENV
16                        then {dict(SCOPEUNIT) ⇀ ⟨(SIGNALROUTE, nm)⟩}
17                        else {} in
18           let newset = sigrouteset ∪ qset in
19           service-connectmap(tl decllist, newset, connectmap)(dict)),
20      mk-Connect0(, )
21        → (let (route, routeset) = service-connect (hd decllist)(dict) in
22           let nconnectmap = connectmap + [route ↦ routeset] in
23           if route ∉ dom connectmap ∧ routeset ∩ union rng connectmap = {} then
24             service-connectmap (tl decllist, sigrouteset, nconnectmap)(dict)
25            else
26             exit("§4.10.2: Неправильное соединение маршрута сигнала сервиса" )),
27      T → service-connectmap (tl decllist, sigrouteset, connectmap)(dict))))
```

запись: *Decompositiondecl*$_0$* *Qual*-set *ProcessconnectionD* → *Dict* → *ProcessconnectionD*

Цель Убедиться, что маршруты сигнала сервиса декомпозиции сервиса надлежащим образом соединены

Параметры

decllist Список определений декомпозиции

sigrouteset *Quals* маршрутов сигналов сервиса, в которых ENV специфицирована как одна из конечных точек. Маршруты сигналов сервиса собираются в процессе рекурсивного прохода списка определений. Когда функция используется изначально (в *transform-decomposition-body*), множество является пустым

connectmap Отображение, задающее связь между соединениями и маршрутами сигналов сервиса, к которым она относится. Это отображение создается в процессе рекурсивного просмотра списка определений.

Результат Ранее задействованный *ProcessconnectionD*. Используется для проверки конструкций VIA в сервисах.

Алгоритм

Строки 1-9 После учета всех соединений и маршрутов сигналов сервиса совокупность маршрутов сигналов сервиса должна совпадать с маршрутами сигналов сервиса, задействованными в соединении маршрута сигналов сервиса (строка 9).

Строка 2 Пусть *pqual* обозначает процесс

Строка 3 Пусть *outersigrouteset* обозначает маршруты сигналов, одной из конечных точек которых является процесс

Строка 4	Возвратить пустое отображение, если для процесса не были специфицированы маршруты сигналов сервиса (и соединений).
Строка 7	Если маршрутов сигналов, одной из конечных точек которых является процесс, больше, чем маршрутов сигналов соединений маршрутов сигналов сервиса, значит несколько соединений маршрутов сигналов сервиса отсутствуют.
Строка 13	Рассмотреть следующее определение.
Строки 14-18	Если следующее определение является определением маршрутом сигналов сервиса и одной из конечных точек является ENV, то пусть *newset* обозначает множество, которое было обновлено, чтобы включить этот маршрут сигналов сервиса.
Строка 19	Перейти к следующему определению.
Строка 20	Если следующее определение является соединением маршрута сигналов сервиса, то
Строка 21	Убедиться, что соединение произведено правильно, и получить *Qual* маршрута сигнала (*route*) и *Quals* маршрутов сигналов сервиса, к которым он присоединен (*routeset*).
Строка 22	Обновить отображение соединений для включения информации об этом соединении
Строка 23	Маршрут сигналов и маршрут сигналов сервиса не могут быть задействованы более чем в одном соединении, а маршруты сигналов в этом соединении не должны иметь другого соединения.
Строка 24	Перейти к остальным определениям.
Строка 27	Перейти к остальным определениям

service-connect(mk-*Connect*$_0$(*sigrouteid, routeidlist*))(*dict*) ≜ (3.4.14.9)

```
1  (let squal = get-visible-qual (sigrouteid, SIGNALROUTE)(dict) in
2   let mk-SignalrouteD(, endpoint2, sigset1, sigset2) = dict(squal) in
3   let pqual = dict(SCOPEUNIT) in
4   let connectset = {get-visible-qual (routeidlist [i], SIGNALROUTE)(dict) | i ∈ ind routeidlist} in
5   let (insigset, outsigset) = if endpoint2 = pqual then (sigset1, sigset2) else (sigset2, sigset1) in
6   if card connectset ≠ len routeidlist then
7       exit ("§4.10.1: Идентификатор маршрутов сигналов сервиса встретился дважды в соединении маршрута сигнала сервиса")
8     else
9       if is-wf-connectsignalroutes (connectset, insigset, outsigset, {}, {})(dict) then
10        (squal, connectset)
11        else
12       exit("§4.10.1: Сигналы в маршрутах сигналов сервиса должны быть такими же как и для подсоединенного маршрута сигналов"))
```

запись: *Connect*$_0$ → *Dict* → *Qual Qual*-set

Цель Убедиться, что соединение маршрутов сигналов сервиса произведено правильно

Параметры Соединение маршрутов сигналов сервиса AS_0, содержащее

sigrouteid	Маршрут сигналов
routeidlist	Маршруты сигналов сервиса

Результат *Qual* маршрута сигналов и множество *Quals* маршрута сигналов сервиса

Алгоритм

Строка 1	Построить *Qual* маршрута сигналов.
Строка 2	Произвести декомпозицию дескриптора маршрута сигналов.
Строка 3	Пусть *pqual* обозначает *Qual* окружающего процесса
Строка 4	Построить *Quals* маршрутов сигналов сервиса.
Строка 5	Пусть *insigset* обозначает сигналы в маршруте сигналов, поступающие в процесс, а *outsigset* - сигналы, выходящие из маршрута сигналов.
Строки 6-7	Маршруты сигналов сервиса в соединении должны быть разделены и

Строка 9 Сигналы в маршруте сигналов должны соответствовать сигналам в маршрутах сигналов сервиса

3.5 Преобразование выражений

Этот раздел содержит функции, которые преобразовывают выражения и термы в выражения и термы. Здесь также решается проблема "понимания по контексту", т.е. иногда функции преобразования выражений вызываются определенным образом для идентификации сорта выражения. Поэтому функции используют также в качестве аргумента множество допустимых сортов и возвращают множество допустимых сортов, которое является всегда непустым подмножеством множества аргументов. Например, в случае преобразования левой части равенства, множество всех видимых сортов изначально дается в качестве аргументов. Результирующее множество сортов этого преобразования должно содержать только один сорт. Левая и правая части затем преобразуются снова, используя в качестве аргумента именно этот сорт.

Кроме того, информация о контексте *Context*, в котором используются выражение/терм, дается как аргумент. Используются четыре различных вида контекстов: CONSTANT, означающий, что аргумент выражения/терма должен быть константой, EXPRESSION, означающий, что аргумент может быть переменным выражением, AXIOMS, означающий, что аргумент должен быть термом, используемым в аксиомах, и MAPPING, означающий, что аргумент должен быть термом, используемым в части отображения аксиом.

В процессе преобразования собирается информация об импортируемых выражениях и идентификаторах значений, встречающихся в выражении/терме. Поэтому возвращается *Dict*, который может содержать дескрипторы идентификаторов значений (*ValueidD*), а также информацию об импортируемых выражениях во входах *Quotdict* IMPLIED и IMPORTLIST (см. определение домена *Quotdict*).

Функции входа - две (идентичные) функции *transform-term* и *transform-expr*

transform-term(*term, context, sortset*)(*dict*) ≜ (3.5.1)

1 *transform-expr*(*term, context, sortset*)(*dict*)

запись: $Term_0$ *Context Sortqual*-set → *Dict* → $Term_1$ *Sortqual*-set *Dict*

Цель Преобразовать терм AS_0 в терм AS_1. Функция идентична *transform-expr*. Она вводится для улучшения читаемости.

transform-expr(*expr, context, sortset*)(*dict*) ≜ (3.5.2)

```
 1  (let (as1tree, sortset', dict') =
 2      (cases expr :
 3        (mk-Scopeexpr0(scope, ex)
 4          → transform-expr (ex, context, sortset)(dict + [SCOPEUNIT ↦ scope]),
 5        mk-Id0(, )
 6          → if context ∈ {AXIOMS, MAPPING} then
 7               transform-axiom-id (expr, sortset)(dict)
 8             else
 9               transform-id (expr, context, sortset)(dict),
10        mk-Stringterm0(, )
11          → transform-stringexpr (expr, sortset)(dict),
12        mk-Condexpr0(e1, e2, e3),
13        mk-Condterm0(e1, e2, e3)
14          → transform-condexpr (mk-Condexpr0(e1, e2, e3), context, sortset)(dict)
15        mk-Operatorapp0(, )
16          → transform-operatorexpr (expr, context, sortset)(dict),
17        mk-Operatorterm0(, )
18          → transform-operatorterm (expr, context, sortset)(dict),
19        mk-Monadexpr0(op, ex),
20        mk-Monadterm0(op, ex)
21          → transform-monadexpr (mk-Monadexpr0(op, ex), context, sortset)(dict),
22        mk-Infixexpr0(e1, op, e2),
23        mk-Infixterm0(e1, op, e2)
24          → transform-infixexpr (mk-Infixexpr0(e1, op, e2), context, sortset)(dict),
25        mk-Errorterm0()
26          → exit("§5.4.1.7: Терм-ошибка используется в составном терме"),
27        mk-Spellingterm0()
28          → transform-spelling (expr, context, sortset)(dict),
29      T → transform-build-in-expression (expr, context, sortset)(dict))) in
30  let as1 tree' = if context ∈ {AXIOMS, MAPPING} ∧ is-Ground-term1(as1 tree) then
31                    mk-Ground-expression1(as1 tree)
32                  else
33                    as1 tree in
34  (as1 tree', sortset, dict'))
```

Запись: (*Expr*0 | *Term*0 | *Scopeexpr*0) *Context Sortqual*-set → *Dict* →
[*Expression*1 | *Term*1] *Sortqual*-set *Dict*

Цель Преобразовать выражение AS_0 в выражение AS_1

Параметры

expr	Выражение или терм AS_0
context	(Синтаксический) контекст, в котором используется *expr* (см. выше).
sortset	Множество допустимых результирующих сортов для выражения/терма.

Результат

- Результирующее выражение или терм AS_1.
- Результирующее множество (возможных) допустимых результирующих сортов выражения. Это множество всегда будет подмножеством *sortset*. Если мощность множества больше единицы, то результирующее выражение/терм AS_1 не используется (= nil) из-за неоднозначности. (Что не обязательно является ошибкой, т.к. *transform-expr* также учитывает "понимание по контексту", см. выше).
- *dict* обновлен с учетом информации о возможных неявных идентификаторах значений, содержащихся в терме, и неявных импортируемых переменных (см. выше).

Строка 3	Если выражение является специальным синтетическим *Scopeexpr*$_0$ (см. определение AS_0), то преобразовать содержащееся выражение по контексту сервиса, обозначенного через *scope*.
Строки 5-9	Если выражение является идентификатором, то преобразовать его как литерал или идентификатор значения, если выражение встречается в аксиомах, иначе, как литерал, синоним или переменную.
Строка 10	Преобразовать символьную строку.
Строки 12-13	Преобразовать условное выражение или условный терм.
Строка 15	Преобразовать применение оператора, используемое в контексте выражения.
Строка 17	Преобразовать применение оператора, используемое в контексте терма.
Строки 19-20	Преобразовать унарное выражение или терм.
Строки 22-23	Преобразовать бинарное выражение или терм.
Строка 25	Среди термов, вообще говоря, не может быть ошибочных термов (или выражений). Отдельные случаи, когда это допустимо, исключаются перед использованием *transform-term*.
Строка 27	Преобразовать терм проверки.
Строка 29	Преобразовать другие (императивные) виды выражений.
Строки 30-33	Если преобразованное выражение/терм - константа, то это *Groundexpression*$_1$.
Строка 34	Возвратить терм/выражение AS_1, ограниченное множество допустимых сортов и *Dict*, возможно, содержащий дескрипторы идентификаторов значений, и информацию о содержащихся импортируемых выражениях

3.5.1 Идентификаторы

transform-axiom-id (*id, sortset*)(*dict*) ≜ (3.5.1.1)

```
 1  if card sortset = 1 then
 2    transform-axiom-id-of-this-sort (id, sortset)(dict)
 3  else
 4   (let mk-Id0(q, nm) = id in
 5    let litset = all-visible-literals (id, sortset)(dict) in
 6    let litsortset = {get-sur(q) | q ∈ litset} in
 7    let qual = dict (SCOPEUNIT) ⇀ ((VALUE, nm)) in
 8    let sorts = if q ≠ ◊ then
 9                  {}
10                 else
11                  if qual ∈ dom dict
12                    then (let mk-ValueidD(, so, ) = dict (qual) in
13                          so)
14                    else sortset in
15    let valset = sortset ∩ sorts in
16    let totset = litsortset ∪ valset in
17    if card totset = 0 then
18      exit("§2.2.2: Для этого идентификатора значений не может быть использован ни один сорт")
19     else
20      if card totset = 1 then
21        transform-axiom-id-of-this-sort (id, totset)(dict)
22      . else
23        if q = ◊ then
24          (let mk-ValueidD(val, so, e) = if qual ∈ dom dict then
25                                           dict (qual)
26                                         else
27                                           mk-ValueidD(nil, sortset, false) in
28          if so ∩ sortset = {} then
29            (nil, totset, dict)
30          else
31            (let dict' = dict + [qual ↦ mk-ValueidD(val, so ∩ sortset, e)] in
32             (nil, totset, dict')))
33        else
34          (nil, totset, dict))
```

запись: Id_0 *Sortqual*-set → *Dict* → [$Term_1$] *Sortqual*-set *Dict*

Цель	Преобразовать терм, являющийся литералом или идентификатором значения, в AS_1.
Параметры	См. *transform-expr* и введение в этот раздел
Результат	См. *transform-expr* и введение в этот раздел

Алгоритм

Строка 1	Если сорт идентификатора определен, то преобразовать идентификатор с учетом его сорта.
Строка 4	Произвести декомпозицию идентификатора.
Строка 5	Построить множество *Qual* всех видимых литералов, имеющих одинаковый идентификатор AS_0 и являющихся допустимым сортом в этом контексте.
Строка 6	Построить множество *Qual*, содержащее сорт литералов.
Строка 7	Построить *Qual* идентификатора значения.
Строки 8-14	Если идентификатор квалифицирован, то он не может быть идентификатором значения (множество возможных сортов пусто), иначе, если существует дескриптор для идентификатора значения в *Dict* (подразумеваемого явной или неявной квантификацией где-либо), то множество возможных сортов находится в дескрипторе (*so*), иначе идентификатор значения может быть любого сорта, допустимого в этом контексте (*sortset*).

Строка 15	Пусть *valset* обозначает множество допустимых сортов в этом контексте, если идентификатор является идентификатором значения.
Строка 16	Пусть *totset* обозначает множество допустимых сортов в этом контексте, если идентификатор обозначает идентификатор значения или литерал.
Строка 17	Должен существовать по крайней мере один возможный сорт.
Строки 20-34	Если существует один и только один возможный сорт, то преобразовать идентификатор с учетом этого сорта, иначе, не возвращать объекты AS_1 (nil) и возвратить полное множество допустимых сортов. Но первоначально создать или модифицировать дескриптор идентификатора значения, если идентификатор может обозначать идентификатор значения (если он не квалифицирован).
Строки 24-27	Создать дескриптор, если дескриптор идентификатора значения уже не существует.
Строки 28-29	Если множество сортов в дескрипторе не имеет элементов, совпадающих с допустимым множеством сортов в этом контексте, то идентификатор не может обозначать идентификатор значения, и поэтому дескрипторы не добавляются в *Dict*.
Строки 31-32	Если дескриптор имеет элементы, совпадающие с допустимым множеством сортов, то ограничить множество сортов в дескрипторе, так чтобы он содержал только совпадающие элементы. Возвратить модифицированный *Dict*

transform-id (*id, context, sortset*)(*dict*) ≜ (3.5.1.2)

```
1   (let squal = get-visible-qual (id, VALUE)(dict) in
2    let sort = get-parent (s-Sortqual(dict(squal)))(dict) in
3    if is-local (squal)(dict) then
4     if (context = EXPRESSION ∨ is-SynD(dict(squal))) ∧ sort ∈ sortset then
5       if is-SynD(dict(squal)) then
6         transform-expr(s-Expr0(dict(squal)), CONSTANT, {sort})(dict)
7       else
8         (make-as1-identifier (squal)(dict), {sort}, dict)
9     else
10      exit("§2.2.2: Локальные синоним или переменная не относятся к правильному сорту")
11   else
12    (let totalset = all-visible-literals(id, sortset)(dict) ∪
13        all-variables-and-synonyms (id, context, sortset)(dict) in
14     let squalset = {q | (∃d ∈ totalset)(q = get-parent (s-Sortqual (dict (d)))(dict))} in
15     if totalset = {} then
16        exit("§2.2.2: Этому сорту контекста не удовлетворяют синонимы, переменные и литералы")
17       else
18        if card squalset = 1 then
19          if card totalset ≠ 1 then
20            exit("§2.2.2: Некоторые переменные и синонимы допустимы для этого контекста")
21           else
22            (let qual ∈ totalset in
23             let squal ∈ squalset in
24             if is-SynD(dict(qual)) then
25              (let deflevel = [SCOPEUNIT ↦ get-sur(qual)] in
26               transform-expr(s-Expr0(dict(qual)), CONSTANT, {squal})(dict + deflevel))
27             else
28              if is-LiteralD(dict(qual)) then
29                (mk- Ground-term1(make-as1-identifier(qual)(dict)), {squal}, dict)
30               else
31                (make-as1-identifier(qual)(dict), {squal}, dict))
32       else
33        (nil, squalset, dict)))
```

запись: Id_0 *Context Sortqual*-set → *Dict* → [$Expression_1$] *Sortqual*-set *Dict*

Цель Преобразовать литерал, переменную или синоним в AS_1

Параметры См. *transform-expr* и введение в этот раздел

Результат См. *transform-expr* и введение в этот раздел

Алгоритм

Строка 1	Во-первых, использовать алгоритм с нормальной видимостью для нахождения переменной или синонима.
Строка 3	Если они определены локально, то
Строка 4	Если контекст является выражением-константой, то это должен быть синоним, а сорт переменной или синонима должен быть правильным сортом
Строка 5	Если идентификатор обозначает синоним, то преобразовать выражение синонима, иначе, возвратить идентификатор переменной AS_1, а также сорт переменной.
Строки 12-26	Если синоним или переменная не заданы локально, то место определения выбирается по контексту.
Строки 12-14	Полное множество возможных *Quals* для видимых литералов, переменных и синонимов обозначить через *totalset*, а их типы - через *squalset*.
Строки 15-16	Должна существовать по крайней мере одна подходящая переменная, синоним или литерал.

Строки 18-19	Если существует один и только один возможный сорт для контекста, то должен также существовать один и только один возможный идентификатор, а если так, то
Строка 22	Пусть *qual* обозначает *Qual* идентификатора.
Строка 23	Пусть *squal* обозначает *Qual* сорта.
Строки 24-26	Если идентификатор обозначает синоним, то преобразовать выражение синонима в том контексте, где синоним определен (т.е *deflevel*).
Строки 28-31	Возвратить идентификатор переменной или синонима AS_1, и, если идентификатор обозначает литерал, то включить его в *Ground-term*$_1$.
Строка 33	Если количество возможных сортов больше одного, то выражение (все еще) не единственно и возвращается **nil**.

all-variables-and-synonyms(**mk-***Id*$_0$(*qu*, *nm*), *context*, *sortset*)(*dict*) ≜ (3.5.1.3)

```
1   (let level = dict(SCOPEUNIT) in
2     {q ∈ dom dict | (is-SynD(dict(q)) ∨ (is- VarD(dict (q)) ∧ context ≠ CONSTANT)) ∧
3                 (∃q')(get-parent(q' ⇝ qu)(dict) ⇝ ⟨(VALUE, nm)⟩ = q ∧
4                       (∃q")(q ⇝ q" = level) ∧
5                       get-parent (s-Sortqual(dict(q)))(dict) ∈ sortset}})
```

запись : *Id*$_0$ *Context Sortqual*-**set** → *Dict* → *Qual*-**set**

Цель Извлечь *Quals* всех синонимов и переменных, которые могут быть использованы при заданном условии

Параметры

Id$_0$	Идентификатор, квалифицирующий (ограничивающий) результирующее множество *Qual*
context	если контекст - CONSTANT, то извлекаются только синонимы
sortset	Множество сортов, которое также уточняет результирующее множество *Qual*

Алгоритм

Строка 1	Пусть *level* обозначает объемлющую единицу подразделения.
Строки 2-5	Возвратить все те *Quals*, которые обозначают синонимы или переменные, включают квалификатор (строка 3), видимы (строка 4) и имеют правильный сорт (строка 5).

transform-axiom-id-of-this-sort(*id*, *totset*)(*dict*) ≜ (3.5.1.4)

```
1   (let mk-Id0(qual, nm) = id in
2    let q = dict (SCOPEUNIT) ⁀ ⟨(VALUE, nm)⟩ in
3    if qual = ⟨⟩ ∧ q ∈ dom dict ∧
4      (let mk-ValueidD(,so,e) = dict(q) in
5       so = totset ∧ e) then
6      (let mk-ValueidD(val,,) = dict(q) in
7       if val = nil then
8          (mk-Composite-term1(make-as1-identifier(q)(dict)), totset, dict)
9        else
10         transform-axiom-id-of-this-sort(as0-id(val), totset)(dict))
11     else
12     (let litset = all-visible-literals(id, totset)(dict) in
13      if litset = {} then
14        if q ≠ ⟨⟩ then,
15          exit("§5.2.3: Идентификатор значения не должен быть квалифицирован")
16        else
17          (let vald = if q ∈ dom dict then
18                        (let mk-ValueidD(, so, ) = dict(q) in
19                         if so ∩ totset = {} then
20                           exit("§5.2.3: Несовместимое использование неявно квантифицированного имени значения")
21                         else
22                           mk-ValueidD(nil, so∩ totset, false))
23                      else
24                      mk-ValueidD(nil, totset, false) in
25          let d = dict + [q ↦ vald] in
26           (mk-Composite-term1(make-as1-identifier(q)(dict)), totset, d))
27        else
28         (let litqual ∈ litset in
29          (mk-Ground-term1(make-as1-identifier(litqual)(dict)), totset, dict))))
```

запись : Id_0 *Sortqual*-**set** → *Dict* → $Term_1$ *Sortqual*-**set** *Dict*

Цель Преобразовать идентификатор конкретного сорта в AS_1

Параметры

id Преобразуемый идентификатор AS_0

totset Множество *Qual* сортов, содержащих только конкретный сорт

Результат См. *transform-expr* и введение в этот раздел

Алгоритм

Строка 1 Произвести декомпозицию идентификатора.

Строка 2 Построить *Qual* идентификатора, предполагая, что это идентификатор значения.

Строка 3 Он обозначает явно квантифицированный идентификатор значения, если он не квалифицирован и имеет дескриптор *ValueidD* в *Dict* (строка 3), конкретный сорт совпадает с сортом для идентификатора значения (строка 5), а флаг (*e*), показывающий, что идентификатор явно квантифицированный, имеет значение "истина".

Строки 6-8 Если идентификатор значения не представлен литеральной квантификацией (*val* = nil), то возвратить составной терм, содержащий идентификатор значения AS_1, иначе

Строка 10 Преобразовать идентификатор связанного литерала.

Строка 12 Если идентификатор не является явно квантифицированным идентификатором значения, то извлекаются все возможные *Quals* литералов.

Строки 13-14 Если идентификатор не является литералом, то это -неявно квантифицированный идентификатор значения, и поэтому его не следует квалифицировать.

Строки 17-24	Выделить или создать дескриптор идентификатора значения и обозначить его *vald*.
Строка 18	Если дескриптор уже находится в *Dict* (т.к. где-либо используется), то проведена его декомпозиция и
Строка 19	Должен существовать по крайней мере один допустимый общий сорт для использования в этом или другом случае, и тогда множество допустимых сортов в дескрипторе ограничено *totset* (строка 22).
Строка 24	Иначе, создается новый дескриптор.
Строка 26	Возвратить составной терм AS_1, содержащий идентификатор значения, (единственный) допустимый сорт и обновленный *Dict*.
Строки 28-29	Если существует литеральный *Qual* сорта, то преобразовать его в AS_1

3.5.2 Символьные строки

transform-stringexpr(**mk-***Stringterm*$_0$(*q*, *strnm*), *sortset*)(*dict*) ≜ (3.5.2.1)

```
1   (let lqualset = all-visible-literals(mk-Stringterm0(q, strnm), sortset)(dict) in
2    let litsortset = {get-sur(lit) | lit ∈ lqualset} in
3    if lqualset = {} then
4      exit("§2.2.2: Символьная строка не относится к соответствующему сорту")
5     else
6      if card lqualset = 1 then
7       (let lqual ∈ lqualset in
8        let as1lit = make-as1-identifier(lqual)(dict) in
9         (mk-Ground-term1(as1lit), sortset, dict))
10      else
11       (nil, litsortset, dict))
```

запись : *Stringterm*$_0$ *Sortqual*-**set** → *Dict* → [*Ground-term*$_1$] *Sortqual*-**set** *Dict*

Цель Преобразовать строку символов в AS_1

Параметры См. *transform-expr* и введение в этот раздел

Результат См. *transform-expr* и введение в этот раздел

Алгоритм

Строка 1	Извлечь *Quals* всех литералов, имеющих сорт в *sortset* и которые могут быть обозначены *Stringterm*$_0$
Строка 2	Извлечь *Quals* их сорта
Строки 3-4	Должен существовать по крайней мере один допустимый литерал, соответствующий сорту
Строки 6-9	Если существует один и только один допустимый литерал (и результирующий сорт), то извлечь *Qual* литерала (строка 7), построить литерал AS_1 (строка 8) и возвратить пассивный терм, содержащий этот литерал, множество сортов, содержащее его сорт и неизмененный *Dict*.
Строка 11	Если (по-прежнему) существует более одного допустимого литерала (и сорта), то не возвращать термов AS_1 и возвратить множество возможных сортов и неизмененный *Dict*

3.5.3 Операторы

transform-operatorexpr(**mk-***Operatorapp*$_0$(*expr*, *exprlist*), *context*, *tpset*)(*dict*) ≜ (3.5.3.1)

```
1   (let (qual, nm) = cases expr :
2                     (mk-Id0(q,n)      → (q,n),
3                      mk-Qualop0(q,n)  → (q,n),
```

```
4                    T                    → (⟨⟩, nil)) in
5    let level = dict(SCOPEUNIT) in
6    let opqualset = if nm = nill then {} else all-visible-operatops(qual, nm, level)(dict) in
7    let legalqualset = extract-legal-operators(exprlist, context, opqualset, tpset)(dict) in
8    let op = build-extract-operator(expr, exprlist) in
9    let (as₁tree', tpset ', dict ') =
10       if op = nil then
11         (nil, {}, [])
12       else
13         (trap exit with {nil, {}, []) in
14          transform-operatorexpr (op, context, tpset)(dict)) in
15   let fop = build-field-operator (expr, exprlist) in
16   let (as₁tree", tpset ", dict ") =
17       if fop = nil then
18         (nil, {}, [])
19       else
20         (trap exit with (nil, {}, []) in
21          transform-selectexpr (fop, context, tpset)(dict)) in
22   (card (legalqualset ∪ tpset ' ∪ tpset ") = 0
23       → exit("§2.2.2: Терм оператора не может быть использован в этом контексте" ),
24   card legalqualset = 0 ∧ card tpset ' = 0
25       → (as₁tree", tpset ", dict "),
26   card legalqualset = 0 ∧ card tpset " = 0
27       → (as₁tree', tpset ', dict '),
28   card (tpset ' ∪ tpset ") = 0 ∧ card legalqualset = 1
29       → (let qu ∈ legalqualset in
30          transform-qual-operator(qu, exprlist, context, tpset)(dict)),
31   T → (nil, {get-parent(s-Result(dict(q)))(dict) |
32              q ∈ legalqualset} ∪ tpset ' ∪ tpset ", dict)))
```

запись : *Operatorapp*$_0$ *Context Sortqual*-**set** → *Dict* → *Expression*$_1$ *Sortqual*-**set** *Dict*

Цель Преобразовать выражение AS_1, являющееся применением оператора или извлеченной сокращенной записью, или сокращенной записью выбранного поля в AS_1

Параметры См. *transform-expr* и введение в этот раздел

Результат См. *transform-expr* и введение в этот раздел

Алгоритм

Строки 1-4 Извлечь квалификатор и имя оператора. Если оператор - общее выражение (строка 4), то имя отсутствует, т.к. в этом случае это краткая запись для извлеченного оператора.

Строка 5 Пусть *level* обозначает объемлющую единицу подразделения.

Строка 6 Извлечь *Quals* всех операторов, которые возможны согласно квалификатору и имени.

Строка 7 Ограничить множество *Qual* до нахождения в нем только тех операторов, которые могут быть использованы при учете списка аргументов.

Строки 8-14 Воспользоваться извлеченной сокращенной записью оператора. *op* содержит возможные результирующие сорты для извлеченного применения.

Строки 15-21 Воспользоваться сокращенной записью выбора поля. *fop* содержит возможные результирующие сорты для выбора поля.

Строка 22 Если ни нормальное применение оператора, ни извлеченное применение, ни выбор поля не могут быть использованы в этом контексте, то это ошибка.

Строка 24 Если в этом контексте может быть использован только выбор поля, то выбрать эту сокращенную запись в качестве результата.

Строка 26 Если в этом контексте может быть использована только извлеченное применение, то выбрать эту сокращенную запись в качестве результата.

Строка 28 Если в этом контексте может быть использована только нормальное применение оператора, то извлечь *Qual* оператора и преобразовать применение оператора с учетом этого *Qual*.

Строка 31 Иначе оператор (по-прежнему) неоднозначен, и поэтому не возвращать выражений AS_1, возвратить множество сортов, состоящее из результирующих сортов всех возможных операторов в этом контексте, совместно с возможными сортами для извлеченной сокращенной записи (*tpset* ') и сортами для сокращенной записи выбора поля (*tpset* ").

all-visible-operators(*qual*, *nm*, *level*)(*dict*) ≜ (3.5.3.2)

```
1   {q ∈ dom dict | (∃q')(get-sur(get-sur(q)) ⇝ q' = level) ∧
2                   (∃q")(get-parent (q" ⇝ qual)(dict) ⇝ ⟨(OPERATOR, (nm,, ))⟩ = q)}
```

запись : *Qual* (*Name*$_0$ | *Quotedop*$_0$) *Qual* → *Dict* → *Operatorqual*-**set**

Цель Извлечь *Quals* тех операторов, которые видимы в объемлющей единице подразделения *level*, включают квалификатор *qual* и имеют имя *nm*

Алгоритм Результирующее множество содержит все те *Operatorquals*, которые видимы (т.е. объемлющую единицу подразделения, содержащую определение частичного типа, которое определяет оператор, должно быть частью *level*) и включают квалификатор в *Operatorqual*.

transform-operatorterm(**mk**-*Operatorterm*$_0$(*id*, *exprlist*), *context*, *tpset*)(*dict*) ≜ (3.5.3.3)

```
1   (let (qual, nm) = cases id :
2                    (mk-Id0(q, n)      → (q, n),
3                     mk-Qualop0(q, n)→ (q, n)) in
4   let level = get-sur(dict(SCOPEUNIT)) in
5   let opqualset = all-visible-operators(qual, nm, get-sur(level))(dict) in
6   let legalqualset = extract-legal-operators(exprlist, context, opqualset, tpset)(dict) in
7   if card legalqualset = 0 then
8     exit("§2.2.2: Терм оператора не может быть использован в этом контексте" )
9    else
10    if card legalqualset = 1 then
11      (let qu ∈ legalqualset in
12       transform-qual-operator(qu, exprlist, context, tpset)(dict))
13     else
14      (nil, {get-parent(s-Result (dict (q)))(dict) | q ∈ legalqualset}, dict))
```

запись : *Operatorterm*$_0$ *Context Sortqual*-**set** → *Dict* → *Term*$_1$ *Sortqual*-**set** *Dict*

Цель Преобразовать терм оператора в AS_1

Параметры См. *transform-expr* и введение в этот раздел

Результат См. *transform-expr* и введение в этот раздел

Алгоритм

Строки 1-3 Извлечь квалификатор и имя оператора.

Строка 4 Уровень, на котором оператор должен быть видимым, есть объемлющая единица подразделения, окружающая определение сорта, в котором используется оператор

Строка 5 Извлечь *Quals* всех тех операторов, которые видимы в окружающей единице подразделения и могут иметь *qual* в качестве квалификатора AS_0 и имеющие имя *nm*.

Строка 6 Ограничить множество *Qual* до нахождения в нем только тех операторов, которые могут быть использованы при учете списка аргументов

Строка 7	Должен существовать по крайней мере один возможный терм оператора, соответствующий контексту.
Строки 10-12	Если существует один и только один операторный терм, соответствующий контексту, то преобразовать его в AS_1 с учетом правильного *Qual* (*qu*).
Строка 14	Если операторный терм (по-прежнему) неоднозначен, то не возвращать термов AS_1, а возвратить результирующие сорты возможных операторов

build-extract-operator (*expr*, *exprlist*) ≜ (3.5.3.4)

```
1   if is-Qualop0(expr) then
2     nil
3   else
4     (let nm = mk-Name0("EXTRACT", EXCLAMATION) in
5      mk-Operatorapp0(mk-Id0(⟨⟩, nm), ⟨⟨expr⟩ ⌢ exprlist⟩))
```

запись : $Expr_0\ Exprlist_0 \rightarrow [Operatorapp_0]$

Цель

Создать извлеченое применение оператора из его сокращенной записи.

пример

a(b, c)

(a - формальный параметр *expr*, а список b,c - формальный параметр *exprlist*) преобразовывается в

EXTRACT!(a, b, c)

Если идентификатор оператора - инфиксный оператор, то возвращается **nil**, т.к. в этом случае запись всегда обозначает применение оператора

build-field-operator (*var*, *exprlist*) ≜ (3.5.3.5)

```
1   (let expr = exprlist [len exprlist] in
2    let rest = ⟨exprlist [i] | 1 ≤ i ≤ len exprlist - 1⟩ in
3    if ¬ is-Id0(expr) ∧ s-Qualifier0(expr) = ⟨⟩ then
4      nil
5    else
6      (let exprvar = if len rest = 0 then var else build-field-operator (var, rest) in
7       if exprvar = nil then
8         nil
9       else
10        (let mk-Id0(, nm) = expr in
11         mk-Selectexpr0(exprvar, nm))))
```

запись : $Expr_0\ Exprlist_0 \rightarrow [Selectexpr_0]$

Цель

Создать сокращенною запись извлеченного поля из сокращенной записи с круглыми скобками

пример

a(b, c)

(a - формальный параметр *var*, а список b,c - формальный параметр *exprlist*) преобразуется в

a!b!c

Функция рекурсивна. Это значит, что сначала создается a!b, а затем a!b!c

extract-legal-operators(*exprlist, context, opqualset, sortset*)(*dict*) ≜ (3.5.3.6)

```
1   (if opqualset = {} then
2      {}
3     else
4      (let q ∈ opqualset in
5       let (,ts,) =
6           (trap exit with (nil, {}, []) in
7            transform-qual-operator(q, exprlist, context, sortset)(dict)) in
8       let qs = if ts = {} then {} else {q} in
9       qs ∪ extract-legal-operators(exprlist, context, opqualset \ {q}, sortset)(dict)))
```

запись : *Exprlist*$_0$ *Context Operatorqual*-**set** *Sortqual*-**set** → *Dict* → *Operatorqual*-**set**

Цель Извлечь все *Quals* операторов, которые соответствуют сорту фактических параметров, сорту результата и имеющим конкретное имя

Параметры

exprlist	Фактические параметры
context	Контекст, в котором используется оператор (см. *transform-expr*)
opqualset	Возможные *Quals*, имеющие конкретное имя
sortset	Возможные результирующие сорты

Результат *Quals* оператора, который может быть использован. Это множество является подмножеством *opqualset*

Алгоритм

Строка 1	После завершения возвратить пустое множество (функция рекурсивна).
Строка 4	Протестировать *Qual* оператора при помощи
Строки 5-7	Попытки преобразовать применение оператора с учетом *Qual* оператора. Если это преобразование проходит успешно (если *transform-qual-operator* не отловлен), значит множество результирующих сортов (*ts*) непусто.
Строка 8	Если *Qual* оператора может быть использован, то *qs* обозначает *Qual*.
Строка 9	Объединить *qs* с допустимыми *Quals* оператора из остального *opqualset*

transform-qual-operator (*qual*, *exprlist*, *context*, *sortset*)(*dict*) ≜ (3.5.3.7)

```
1   (let (,(,parm, tqual)) = qual [len qual] in
2   if tqual ∉ sortset then
3     exit("§2.2.2: Результирующий сорт оператора является недопустимым для данного контекста")
4     else
5     if len parm ≠ len exprlist then
6       exit("§5.2: Длина списка параметров не является корректной для оператора")
7     else
8       (let (as1parmlist, d) = transform-actparms (parm, exprlist, context)(dict) in
9        let as1id = make-as1-identifier (qual)(dict) in
10       let const = (∀p ∈ elems as1parmlist)(is-Ground-term1(p)) in
11       let as1tree =
12          if context ∈ {AXIOMS, MAPPING} then
13           if const then
14             mk-Ground-term1((as1id, as1parmlist))
15           else
16             mk-Composite-term1((as1id, as1parmlist))
17          else
18           if const then
19             mk-Ground-term1((as1id, as1parmlist))
20           else
21             mk-Operator-application1(as1id, as1parmlist) in
22       (as1tree, {tqual}, d)))
```

запись : *Operatorqual* (*Exprlist*$_0$| *Term*$_0$*) *Context Sortqual*-set → *Dict* →
(*Term*$_1$| *Expression*$_1$) *Sortqual*-set *Dict*

Цель Преобразовать конкретно определенный оператор в AS_1

Параметры

Qual	*Qual*, который однозначно определяет оператор
exprlist	Список аргументов, являющийся списком термов, если оператор встречается в равенствах. В противном случае это список выражений.
context, sortset	См. *transform-expr*

Результат См. *transform-expr*. Результирующее множество сортов содержит только один сорт

Алгоритм

Строка 1	Извлечь *Operatorqualelem* из *Qual* оператора и идентифицировать результирующий сорт через *tqual*, а сорты аргументов через *parm*
Строки 2-6	Результирующий сорт должен быть одним из допустимых сортов в этом контексте, а длина списка формальных параметров должна совпадать с длиной списка фактических параметров.
Строка 8	Преобразовать фактические параметры, являющиеся либо термами либо выражениями.
Строка 9	Построить идентификатор оператора AS_1
Строка 10	*const* - "истина", если все фактические параметры являются пассивными термами (константы).
Строки 11-18	Если оператор встречается в аксиомах, то сформировать терм AS_1 (строка 13), иначе, сформировать выражение AS_1.
Строка 22	Возвратить сформированный терм/выражение AS_1, результирующий сорт оператора и *Dict*, возможно обновленный в процессе оценки фактических параметров (см. *transform-expr*)

3.5.3.1 Оператор spelling

transform-spelling(mk-*Spellingterm*$_0$(mk-*Id*$_0$(*q*, *nm*)), *context*, *sortset*)(*dict*) ≜ (3.5.3.1.1)

```
1   if context ≠ MAPPING ∨ q ≠ ⟨⟩ then
2     exit("§5.4.1.15: Недопустимое использование оператора SPELLING" )
3    else
4     (let stringsort = get-predef-sort("CHARSTRING")(dict) in
5      let qualset = {q ∈ dom dict | is- ValueidD(dict(q)) ∧ s-Mapvalue(dict (q)) ≠ nil ∧
6                                  (VALUE, nm) = q[len q]} in
7      if card qualset = 1 ∧ stringsort ∈ sortset then
8        (let qual ∈ qualset in
9         let mk-ValueidD(litq,, ) = dict (qual) in
10        let (,mk-Name0(str, )) = litq [len litq] in
11        transform-stringexpr (mk-Stringterm0(⟨⟩, str), {stringsort})(dict))
12      else
13       exit("§5.4.1.15: Недопустимое использование оператора SPELLING" ))
```

запись : *Spellingterm*$_0$ *Context Sortqual*-set → *Dict* → *Term*$_1$ *Sortqual*-set *Dict*

Цель Преобразовать оператор SPELLING в литерал символьной строки AS$_1$, который он обозначает (в функции *transform-mappingaxioms* аксиомы генерируются для каждого возможного литерала квантифицированного имени значения)

Параметры См. *transform-expr* и введение в этот раздел

Результат См. *transform-expr* и введение в этот раздел

Алгоритм

Строка 1 Оператор spelling должен использоваться только в квантификации литералов, а содержащийся идентификатор значения не должен быть квантифицирован

Строка 4 Извлечь *Qual* сорта символьной строки

Строки 5-7 Должен существовать только один *Qual* (т.е. единственный дескриптор) содержащегося имени значения (*nm*), а предопределенный сорт символьной строки должен быть допустим в контексте.

Строки 8-9 Извлечь *Qual* имени значения и произвести декомпозицию его дескриптора. *litq* обозначает *Qual* литерала, представленного именем значения.

Строка 10 Извлечь результат проверки литерала.

Строка 11 Построить терм строки AS$_0$ из проверки литерала и преобразовать терм строки

3.5.3.2 Экспорт, импорт, обозревание, активация

transform-build-in-expression(*expr*, *context*, *sortset*)(*dict*) ≜ (3.5.3.2.1)

```
1   if context ≠ EXPRESSION ∧ ¬ is-Selectexpr0(expr) ∧ ¬ is-Tupleexpr0(expr) then
2     exit("§5.5.4: Императивный оператор не может быть использован в постоянном выражении")
3   else
4     cases expr:
5     (mk-Importexpr0(v, exp)
6       → transform-importexpr(v, exp, sortset)(dict),
7      mk-Viewexpr0(v, exp)
8       → transform-viewexpr(v, exp, sortset)(dict),
9      mk-Nowexpr0()
10      → transform-nowexpr(sortset)(dict),
11     mk-Activeexpr0(,)
12      → transform-activeexpr(expr, sortset)(dict),
13     mk-Selectexpr0(,)
14      → transform-selectexpr(expr, context, sortset)(dict),
15     mk-Tupleexpr0(,)
16      → transform-tupleexpr(expr, context, sortset)(dict),
17     T→ transform-pid-build-in(expr, sortset)(dict))
```

запись : $Expr_0$ *Context Sortqual*-**set** → *Dict* → [$Expression_1$] *Sortqual*-**set** *Dict*

Цель Преобразовать одно из конкретных выражений в AS_1

Параметры См. *transform-expr* и введение в этот раздел

Результат См. *transform-expr* и введение в этот раздел

Алгоритм

Строки 1-2	Только случаи $Selectexpr_0$ или $Tupleexpr_0$ вышеприведенных альтернатив допустимы в постоянных выражениях.
Строка 5	Преобразовать выражение IMPORT.
Строка 7	Преобразовать выражение VIEW.
Строка 9	Преобразовать выражение NOW.
Строка 11	Преобразовать выражение ACTIVE таймера.
Строка 13	Преобразовать выражение выбора поля.
Строка 15	Преобразовать выражение-кортеж.
Строка 17	Иначе, это предопределенное выражение PID

transform-importexpr(mk-Id_0(*qu*, *nm*), *expr*, *sortset*)(*dict*) ≜ (3.5.3.2.2)

```
1   (let level = process-or-service-level(dict(SCOPEUNIT)) in
2    let qualset = {q ∈ dom dict | (∃q', sort)(q' ⌢ qu ⌢ ⟨(IMPORT, (nm, sort))⟩ = q ∧
3                                   (get-sur(q) = level ∨ get-sur(q) = process-level(level)) ∧
4                                   sort ∈ sortset)} in
5    if qualset = {} then
6      exit("§2.2.2: Нет импортируемых переменных, удовлетворяющих контексту")
7     else
8      if card qualset = 1 then
9        (let qual ∈ qualset in
10        let sort = s-Sortqual(dict(qual)) in
11        let importlist = dict(IMPORTLIST) in
12        let imp = dict(IMPLIED) in
13        let nm' = create-unique-name() in
14        let newqual = level ⌢ ⟨(VALUE, nm')⟩,
15            imp' = imp ∪ {(sort, nm')},
16            as1id = make-as1-identifier(newqual)(dict) in
17        let importlist' = importlist ⌢ ⟨(nm', qual, expr)⟩ in
18        (as1id, {get-parent(sort)(dict)}, dict + [IMPORTLIST ↦ importlist',
19                                                  IMPLIED    ↦ imp'])
20       else
21        (let sortset' = {sort | (∃q ∈ qualset)(dict(q) = mk-ImportD(sort))} in
22          (nil, sortset', dict)))
```

запись : Id_0 [$Expr_0$] *Sortqual*-set → *Dict* → [$Expression_1$] *Sortqual*-set *Dict*

Цель Преобразовать импортируемое выражение в его представление AS_1

Параметры

Id	Идентификатор импортируемой переменной
expr	Необязательное выражение PID, обозначающее процесс экспорта
sortset	См. *transform-expr*

Результат См. *transform-expr* и введение в этот раздел. Возвращаемое выражение - идентификатор неявной переменной, заключенный в импортируемом выражении

Алгоритм

Строка 1	Извлечь уровень, на котором задана импортируемая переменная, т.е. процесс или сервис.
Строки 2-4	Извлечь импортируемый *Quals* всех импортируемых переменных, которые могут быть специфицированы с помощью квалификатора (*qu*), имени (*nm*) (строка 2) и сорта *sort* из *sortset*, и которые заданы в объемлющем сервисе или в объемлющем процессе (строка 3).
Строка 5	Должна существовать по крайней мере одна возможная импортируемая переменная.
Строка 9	Если существует уникальный импортируемый *Qual*, то
Строка 9	Обозначить *Qual* этой импортируемой переменной через *qual*.
Строка 10	Обозначить *Qual* ее сорта через *sort*.
Строка 11	Обозначить текущий список импортируемых выражений, используемых где-либо в окружающем выражении, через *importlist*.
Строка 12	Обозначить текущее множество неявных переменных, используемых в теле процесса, через *imp*.
Строка 13-14	Создать уникальное имя переменной AS_0 и построить его *Qual*.
Строка 15	Обновить текущее множество неявных переменных вместе с его сортом (*sort*) и новым именем.
Строка 16	Построить идентификатор AS_1 неявной переменной.

Строка 17	Обновить текущий список импортируемых выражений.
Строка 18	Возвратить идентификатор AS_1 и *Dict*, содержащий обновленный список импорта и множество неявных переменных.
Строка 21	Если импортируемое выражение (все еще) неоднозначно, то возвратить множество возможных сортов

transform-viewexpr(id, expr, sortset)(dict) ≜ (3.5.3.2.3)

```
1   (let mk-Id0(qu, nm) = id in
2   let pqual = process-or-service-level(dict(SCOPEUNIT)) in
3   let qset = {q ∈ dom dict | (let qual = pqual ⌢ ⟨(VIEW, q)⟩ in
4                                qual ∈ dom dict ∧
5                                get-parent(s-Qual(dict(qual)))(dict) ∈ sortset ∧
6                                (∃q')(q' ⌢ qu ⌢ ⟨(VALUE, nm)⟩ = q))} in
7   if card qset ≠ 1 then
8     exit("§2.2.2: Одна и только одна переменная с атрибутом раскрытия должна соответствовать выражению обозревания")
9   else
10    (let q ∈ qset in
11    let inst = get-predef-sort ("PID")(dict) in
12    let as1id = make-as1-identifier(q)(dict),
13        (as1expr,, d) = transform-expr(expr, EXPRESSION, {inst})(dict) in
14    let as1tree = mk-View-expression1(as1id, as1expr) in
15    (as1tree, {s-Sortqual(dict(q))}, d)))
```

запись : $VarId_0\ Expr_0\ Sortqual\text{-set} \rightarrow Dict \rightarrow View\text{-}expression_1\ Sortqual\text{-set}\ Dict$

Цель Преобразовать выражение *View* в AS_1

Параметры

id	Переменная обозревания
expr	Выражение PID, означающее процесс с атрибутом раскрытия
sortset	См. *transform-expr*

Результат См. *transform-expr*

Алгоритм

Строка 1	Произвести декомпозицию идентификатора переменной обозревания.
Строка 2	Извлечь уровень, на котором задана переменная обозревания.
Строки 3-6	Пусть *qset* обозначает множество, которое соответствует выражению обозревания. Множество состоит из тех *Quals* переменных, которые "видимы" в процессе (строки 3-4), имеют соответствующий сорт (строка 5) и включают квалификатор и имя, специфицированные в выражении обозревания (строка 6).
Строка 11	Извлечь *Qual* сорта PID.
Строка 12	Построить идентификатор AS_1 переменной с атрибутом раскрытия (совпадающий с идентификатором переменной обозревания).
Строка 13	Преобразовать выражение PID.
Строка 14	Создать выражение обозревания AS_1.
Строка 15	Возвратить выражение обозревания AS_1, сорт переменной и (возможно) обновленный *Dict*

transform-nowexpr(sortset)(dict) ≜ (3.5.3.2.4)

```
1   (let tqual = get-predef-sort("TIME")(dict) in
2   if tqual ∈ sortset then
3     (mk-Now-expression1(), {tqual}, dict)
4   else
5     exit("§5.5.4.1: Выражение NOW должно использоваться там, где допустимы значения TIME"))
```

запись : $Sortqual\text{-set} \rightarrow Dict \rightarrow Now\text{-}expression_1\ Sortqual\text{-set}\ Dict$

Цель Преобразовать выражение NOW в AS_1

Параметры См. *transform-expr*

Результат См. *transform-expr*

Алгоритм

Строка 1	Извлечь *Qual*, обозначающий сорт TIME.
Строка 2	Если этот сорт допустим в этом месте
Строка 3	То возвратить выражение now AS_1, сорт TIME и неизмененный *Dict*

transform-activeexpr(mk-$Activeexpr_0$(*id*, *actparm*), *sortset*)(*dict*) ≜ (3.5.3.2.5

```
1   (let qual = get-visible-qual(id, SIGNAL)(dict),
2        bqual = get-predef-sort("BOOLEAN")(dict) in
3   let as1id = make-as1-identifier(qual)(dict) in
4   if ¬ is-TimerD(dict(qual)) then
5       exit("§2.8: Идентификатор в выражении активного таймера не обозначает таймер")
6      else
7      (let mk-TimerD(tplist, ) = dict(qual) in
8       if bqual ∈ sortset ∧ len actparm = len tplist then
9         (let (as1actparm, d) = transform-actparms(tplist, actparm, EXPRESSION)(dict) in
10         let as1tree = mk-Timer-active-expression1(as1id, as1actparm) in
11         (as1tree, {bqual}, d))
12       else
13         exit("§2.8: Недопустимое выражение активного таймера")))
```

запись : $Activeexpr_0$ *Sortqual*-set → *Dict* → $Timer\text{-}active\text{-}expression_1$ *Sortqual*-set *Dict*

Цель Преобразовать выражение активного таймера в AS_1

Параметры

id	Идентификатор таймера
actparm	Фактические параметры
sortset	См. *transform-expr*

Результат См. *transform-expr*

Алгоритм

Строки 1-2	Извлечь *Qual* таймера и *Qual* булевского сорта.
Строка 3	Построить идентификатор AS_1 таймера.
Строка 4	Идентификатор AS_0 должен обозначать таймер (а не сигнал).
Строка 7	Пусть *tplist* обозначает список сортов параметров.
Строка 8	Булевский сорт должен быть допустимым результирующим сортом в этом контексте, а длина списка фактических параметров должна совпадать с длиной списка сортов.
Строка 9	Преобразовать фактические параметры в AS_1.
Строки 10-11	Построить выражение активного таймера AS_1 и возвратить его вместе с булевским сортом и (возможно) с обновленным *Dict*

transform-selectexpr(mk-$Selectexpr_0$(*v*, *nm*), *context*, *sortset*)(*dict*) ≜ (3.5.3.2.6)

```
1   (let mk-Name0(str, ) = nm in
2    let opnm = mk-Name0(str ⌢ "EXTRACT", EXCLAMATION) in
3    let opid = mk-Id0(⟨⟩, opnm) in
4    transform-expr(mk-Operatorapp0(opid, ⟨v⟩), context, sortset)(dict))
```

запись : $Selectexpr_0$ *Context Sortqual*-set → *Dict* → $Expression_1$ *Sortqual*-set *Dict*

Цель Преобразовать выражение выбора поля в AS_1

Параметры См. *transform-expr*

Результат См. *transform-expr*

Алгоритм

Строка 1	Пусть *str* обозначает проверку поля.
Строка 2	Создать имя из оператора EXTRACT!, который используется в специфицированном поле.
Строки 3-4	Создать идентификатор оператора, построить применение оператора с учетом выражения в левой части в качестве параметра и преобразовать его обычным образом

transform-tupleexpr(mk-$Tupleexpr_0$(*q*, *exprlist*), *context*, *sortset*)(*dict*) ≜ (3.5.3.2.7)

```
1   (let opid = mk-Id0(q, mk-Name0("MAKE", EXCLAMATION)) in
2    transform-expr(mk-Operatorapp0(opid, exprlist), context, sortset)(dict))
```

запись : $Tupleexpr_0$ *Context Sortqual*-set → *Dict* → $Expression_1$ *Sortqual*-set *Dict*

Цель Преобразовать выражение-кортеж (первичную структуру) в AS_1

Параметры См. *transform-expr*

Результат См. *transform-expr*

Алгоритм

Строка 1	Создать идентификатор оператора MAKE!
Строка 2	Построить применение оператора с учетом списка выражений в качестве списка параметров и преобразовать его обычным образом

transform-pid- build-in(*expr*, *sortset*)(*dict*) ≜ (3.5.3.2.8)

```
1   (let pqual = get-predef-sort("PID")(dict) in
2    if pqual ∈ sortset then
3      ( cases expr :
4         (mk-Parentexpr0()
5           → (mk-Parent-expression1(), {pqual}, dict),
6         mk-Offspringexpr0()
7           → (mk-Offspring-expression1(), {pqual}, dict),
8         mk-Senderexpr0()
9           → (mk-Sender-expression1(), {pqual}, dict),
10        mk-Selfexpr0()
11          → (mk-Self-expression1(), {pqual}, dict)))
12     else
13      exit("§5.5.4.3: Выражение PID используется в неправильном контексте"))
```

запись : $Expr_0$ *Sortqual*-set → *Dict* → *Pid-expression*$_1$ *Sortqual*-set *Dict*

Цель Преобразовать одно из предопределенных выражений PID в AS_1

Параметры См. *transform-expr*

Резаультат См. *transform-expr*

Алгоритм

Строка 1	Извлечь *Qual* сорта PID.
Строка 2	Сорт PID должен быть одним из допустимых сортов в этом месте.
Строки 4-10	Возвратить *Qual* сорта PID (*pqual*), неизмененный *Dict* и
Строка 4	Родительское выражение или
Строка 6	Выражение-потомок или

Строка 8	Выражение передатчика или
Строка 10	Выражение Self

3.5.4 Выражения условия

transform-condexpr(**mk-***Condexpr*$_0$(*bexpr*, *expr*1, *expr*2), *context*, *sortset*)(*dict*) ≜ (3.5.4.1)

```
1   (let boolqual = getpredef-sort("BOOLEAN")(dict) in
2   let (as1,, d) = transform-expr(bexpr, context, {boolqual})(dict) in
3   let (, s', d') = transform-expr(expr1, context, sortset)(d) in
4   let (, s",) = transform-expr(expr2, context, sortset)(d') in
5   if s' ∩ s" = {} then
6      exit("§2.2.2: Выражения условия не соответствуют контексту")
7    else
8      if card (s' ∩ s") = 1 then
9        (let sort ∈ s' ∩ s" in
10        let (as1unique,, dict') = transform-expr(expr1, context, {sort})(dict) in
11        let (as1unique',, dict") = transform-expr(expr2, context, {sort})(dict') in
12        let as1tree =
13           if context ∈ {AXIOMS, MAPPING} then
14              mk-Conditional-term1(as1, as1unique, as1unique')
15              else
16              mk-Conditional-expression1(as1, as1unique, as1unique') in
17        let as1tree' =
18           if is-Ground-term1(as1) ∧ is-Ground-term1(as1unique) ∧ is-Ground-term1(as1unique')
19              then mk-Ground-term1(as1tree)
20              else as1tree in
21        (as1tree', {sort}, dict"))
22      else
23        (nil, s' ∩ s", dict))
```

запись : *Condexpr*$_0$ *Context Sortqual*-**set** → *Dict* → [*Term*$_1$ | *Expression*$_1$] *Sortqual*-**set** *Dict*

Цель Преобразовать выражение или терм условия в AS$_1$

Параметры См. *transform-expr*

Результат См. *transform-expr*

Алгоритм

Строка 1	Извлечь *Qual* булевского сорта.
Строка 2	Преобразовать булевское выражение.
Строка 3	Преобразовать выражение/терм "then".
Строка 4	Преобразовать выражение/терм "else".
Строки 5-6	Должен существовать по крайней мере один результирующий сорт, который может быть использован.
Строка 8	Если имеется ровно один сорт, который может быть использован, тогда
Строка 9	Пусть *sort* обозначает сорт.
Строки 10-11	Вновь преобразовать выражения, на этот раз с уникальным сортом, заданным в качестве аргумента.
Строки 12-16	Образовать выражение AS$_1$ или терм.
Строки 17-20	Если условие, часть "then" и часть "else" являются пассивными термами, тогда результирующее выражение AS$_1$ является пассивным термом.
Строка 20	Возвратить выражение/терм AS$_1$, сорт выражения/терма условия и (возможно) обновленный *Dict*

3.5.5 Инфиксные и префиксные операторы

transform-monadexpr(mk-$Monadexpr_0$(*op*, *expr*), *context*, *sortset*)(*dict*) ≜ (3.5.5.1)

```
1   (let qinfix = mk-Qualop0(⟨⟩, mk-Quotedop0(op)) in
2    let operator =
3       if context ∈ {MAPPING, AXIOMS}
4          then mk-Operatorterm0(qinfix, ⟨expr⟩)
5          else mk-Operatorapp0(qinfix, ⟨expr⟩) in
6    transform-expr(operator, context, sortset)(dict))
```

запись : $Monadexpr_0$ *Context Sortqual*-**set** → *Dict* → [$Expression_1$ | $Term_1$] *Sortqual*-**set** *Dict*

Цель Преобразовать применение префиксного оператора в AS_1

Параметры См. *transform-expr*

Результат См. *transform-expr*

Алгоритм

Строка 1	Из символа оператора построить квалификационный префиксный оператор (квалификатор пуст)
Строки 2-5	Построить терм оператора или применение оператора, в зависимости от того используется ли он в аксиомах или в выражениях
Строка 6	Преобразовать обычным образом построенный терм/выражение

transform-infixexpr(mk-$Infixexpr_0$(*expr*1, *op*, *expr*2), *context*, *sortset*)(*dict*) ≜ (3.5.5.2)

```
1   (let qinfix = mk-Qualop0(⟨⟩, mk-Quotedop0(op)) in
2    let operator =
3       if context ∈ {MAPPING, AXIOMS}
4          then mk-Operatorterm0(qinfix, ⟨expr1, expr2⟩)
5          else mk-Operatorapp0(qinfix, ⟨expr1, expr2⟩) in
6    transform-expr(operator, context, sortset)(dict))
```

запись : $Infixexpr_0$ *Context Sortqual*-**set** → *Dict* → [$Expression_1$ | $Term_1$] *Sortqual*-**set** *Dict*

Цель Преобразовать применение инфиксного оператора в AS_1

Параметры См. *transform-expr*

Результат См. *transform-expr*

Алгоритм

Строка 1	Из символа оператора построить квалификационный инфиксный оператор (квалификатор пуст)
Строки 2-5	Построить терм оператора или применение оператора, в зависимости от того используется ли он в аксиомах или в выражениях
Строка 6	Преобразовать обычным образом построенный терм/выражение

3.6 Обработка обозревания

all-visible-sorts(*dict*) ≜ (3.6.1)

```
1   {qual ∈ dom dict | is-SortD(dict(qual)) ∧
2                  (∃q ∈ Qual)(get-sur(qual) ⇝ q = dict(SCOPEUNIT))}
```

запись : *Dict* → *Sortqual*-**set**

Цель Извлечь из *Dict* множество *Quals* сортов, которые являются видимыми в данном месте, указанном посредством SCOPEUNIT

Алгоритм Возвратить все *Quals*, которые обозначают сорт (строка 1) и для которых существуют частичные *Qual* (*q*) такие, что *Qual* единицы подразделения, которая определяет сорт, при объединении с *q* может образовать *Qual* текущей единицы подразделения

all-visible-literals(*id*, *sortset*)(*dict*) ≜ (3.6.2)

```
1   (let (qual, nm) = cases id:
2                 (mk-Id0(q, n)          ↦ (q, n),
3                  mk-Stringterm0(q, n) ↦ (q, n)) in
4   {q ∈ dom dict | (∃squal ∈ sortset)(squal ⌢ ⟨(LITERAL, nm)⟩ = q) ∧
5                 (∃qu)(get-parent(qu ⌢ qual)(dict) ⌢ ⟨(LITERAL, nm)⟩ = q)})
```

запись : (Id_0 | $Stringterm_0$) *Sortqual*-**set** → *Dict* → *Qual*-**set**

Цель Извлечь множество всех *Quals* литералов, которые видимы в данном месте, которые могут быть обозначены некоторым идентификатором AS_0 и которые относятся к некоторым специфическим сортам

Параметры

id Идентификатор AS_0, который может обозначать литерал

sortset Результирующие *Quals* литералов, все относятся к сортам, содержащимся в *sortset*

Алгоритм

Строки 1-3 Извлечь квалификатор и имя идентификатора (или строку).

Строка 4 Возвратить все *Quals*, которые обозначают литералы, определенные в одном из возможных сортов, и которые включают в себя квалификатор.

all-input-signals(*qual*)(*dict*) ≜ (3.6.3)

```
1   (let qual' = process-or-service-level(qual) in
2     cases dict(qual'):
3      (mk-ServiceD(, , , inp, )  ↦ inp,
4       mk-ProcessD(, inp, , )    ↦ inp))
```

запись : *Qual* → *Dict* → *Signalqual*-**set**

Цель Извлечь *Quals* входных сигналов (включая таймеры)

Параметры

qual *Qual*, обозначающий уровень (единицу подразделения), где требуются входные сигналы

Алгоритм

Строка 1 В случае процедуры модифицировать уровень выбором окружающего процесса или сервиса.

Строка 3 Если единица подразделения является сервисом, то возвратить все входные сигналы.

Строка 4 Если единица подразделения является процессом, то возвратить все входные сигналы

import -export-signals(dict) ≜ (3.6.4)

```
1   (let (emptyq,,) = dict(GLOBALNAMES) in
2   let (exportinput, exportoutput) = extract-exports(dict) in
3   let importset = {(tq, nm) ∈ dom exportmap | (∃q ∈ dom dict)(process-level(q) = dict(SCOPEUNIT) ∧
4                                                  (IMPORT, (nm, tq)) = q[len q])} in
5   let importinput =
6      {xrq | (∃(tq, nm) ∈ importset)
7            ((let mk-SignalnamesD(,, xr') = exportmap((tq, nm)) in
8              get-sur(tq) ⇝ ((SIGNAL, xr')) = xrq))} in
9   let importoutput =
10     {xqq | (∃(tq, nm) ∈ importset)
11           ((let mk-SignalnamesD(, xq, ) = exportmap((tq, nm)) in
12             get-sur(tq) ⇝ ((SIGNAL, xq)) = xqq))} in
13  (exportinput ∪ importinput ∪ {emptyq}, exportoutput ∪ importoutput))
```

запись : *Dict* → *Signalqual*-**set** *Signalqual*-**set**

Цель Извлечь и возвратить *Quals* всех неявных сигналов процесса, связанных с импортом и экспортом и возвратить сигнал *emptyq*.

Алгоритм

Строка 1 Извлечь *Qual* сигнала emptyq, используемого в разрешающем условии/ непрерывном сигнале.

Строка 2 Построить *Quals* всех входных сигналов и *Quals* всех выходных сигналов, используемых, когда процесс экспортирует переменные (т.е. входные сигналы xtQUERY и выходные сигналы xtREPLY).

Строки 3-4 Извлечь *NameclosureD* (см. определение *exportmap*) всех переменных, импортируемых процессом. Множество состоит из всех пар сортов (*tq*) и имен (*nm*), для которых существует переменная импорта, определенная в процессе или содержащемся сервисе *nm*, которые относятся к сорту *tq*.

Строки 5-8 Построить *Quals* всех сигналов xtREPLY, используемых процессом. Множество состоит из всех таких *Quals* xtREPLY, которые могут быть образованы из имен xtREPLY, содержащихся в дескрипторе *SignalnamesD* из *NameclosureD* из *importset*.

Строки 9-12 Проделать те же операции для сигналов xtQUERY, используемых процессом.

Строка 13 Возвратить объединение *Quals* сигналов xtQUERY, сигналов xtREPLY и сигнала emptyq. Сигнал emptyq не возвращается как выходной сигнал, так как выходные сигналы объединяются, чтобы построить неявные маршруты сигналов, и нет маршрутов сигналов, которые содержат сигнал emptyq.

extract-exports(*dict*) ≜ (3.6.5)

```
1   (let vqualset = {qual ∈ dom dict | process-level(qual) = dict(SCOPEUNIT) ∧ is-VarD(dict(qual))} in
2   let closset =
3      {(tq, nm) ∈ dom dict | (∃q ∈ vqualset)
4                              ((let mk-VarD(t,, exp,, ) = dict(q) in
5                               t = tq ∧ (VALUE, nm) = q[len q] ∧
6                               exp = EXPORTED))} in
7   let inputsig =
8      {qual | (∃(tq, nm) ∈ closset)
9              ((let mk-SignalnamesD(, xq, ) = exportmap((tq, nm)) in
10              get-sur(tq) ⇝ ⟨(SIGNAL, xq)⟩ = qual))},
11     outputsig =
12     {qual | (∃(tq, nm) ∈ closset)
13             ((let mk-SignalnamesD(,, xr) = exportmap((tq, nm)) in
14              get-sur(tq) ⇝ ⟨(SIGNAL, xr)⟩ = qual))} in
15  (inputsig, outputsig))
```

запись : *Dict* → *Signalqual*-set *Signalqual*-set

Цель Извлечь и возвратить *Quals* всех сигналов xtQUERY и xtREPLY, используемых процессом экспорта

Алгоритм

Строка 1	Извлечь *Quals* переменных процесса.
Строки 2-6	Из множества переменных построить *NameclosureD*, используемое процессом, т.е. пары сорта *tq* и имени переменной *nm* из *Exportmap*, для которого экспортируется переменная сорта *tq*.
Строки 7-10	Из этого множества *NameclosureD* построить *Quals* сигналов xtQUERY (так же, как это делалось для сигналов xtREPLY в функции *import-export-signals*).
Строки 11-14	Повторить то же самое для сигналов xtREPLY.
Строка 15	Возвратить входные и выходные сигналы

get-visible-qual(*entity*, *entityclass*)(*dict*) ≜ (3.6.6)

```
1   if entity = ENV then
2     ENV
3    else
4     (let (qual, nm) = cases entity:
5                        (mk-Id0(q, n)          ↦ (q, n),
6                        mk-Stringterm0(q, n)   ↦ (q, n),
7                        mk-Qualop0(q, n)       ↦ (q, n)) in
8   let level = dict(SCOPEUNIT) in
9   if (∃q ∈ dom dict)((∃q')(q' ⇝ qual ⇝ ⟨(entityclass, nm)⟩ = q) ∧
10                     (∃q")(get-sur(q) ⇝ q" = level)) then
11    (let q ∈ dom dict be s.t. (∃q')(q' ⇝ qual ⇝ ⟨(entityclass, nm)⟩ = q) ∧
12        (∃q")(get-sur(q) ⇝ q" = level ∧
13              ¬ (∃qu)(get-sur(q) ⇝ qu = level ∧ len qu < len q")) in
14     if is-ErrorD(dict(q)) then
15       exit("§2.2.2: Синоним или генератор или список сигналов определены рекурсивно")
16      else
17       q)
18    else
19      exit("§2.2.2: Идентификатор не видим"))
```

запись : (Id_0 | $Stringterm_0$ | $Qualop_0$ | ENV) *Quot* → *Dict* → (*Qual* | ENV)

Цель Построить *Qual* для идентификатора AS_0, если этот идентификатор видим

Параметры

entity	Идентификатор (или строка или инфиксный оператор), для которого построен *Qual*
entityclass	Класс объектов идентификатора (см. §2.2.2 Z.100). Класс объектов VALUE включает синонимы, переменные и идентификаторы значений. Функция не используется при построении *Quals* литералов и операторов (в соответствии с специальными правилами обозревания).

Алгоритм

Строка 1	Если объект *entity* является ENV, тогда возвратить ENV (см. *transform-channeldef*).
Строки 4-7	Извлечь квалификатор и имя идентификатора (или строки или инфиксного оператора).
Строки 9-10	Если существует такой *Qual* в *Dict*, который включает квалификатор и пару класс объектов и имя в правой части (строка 9), и единица подразделения, обозначенная $qualifier_0$ определена обозреваемой, тогда
Строки 11-13	Пусть *q* обозначает такой *Qual* (строки 11-13) и делает отбор уникальным при помощи выражения, что *q* должно обозначать "ближайший" *Qual* по отношению к текущему уровню (строка 13)
Строки 14-15	Идентификатор не должен быть определен рекурсивно.

signal-qual(*sigid*)(*dict*) ≜ (3.6.7)

```
1   (let qual = (trap exit with nil in
2              {get-visible-qual(sigid, SIGNAL)(dict)}) in
3    if qual ≠ nil then
4      qual
5    else
6     (let mk-Id0(q, nm) = sigid in
7      let qualelem = q ⌢ ⟨(SIGNAL, nm)⟩ in
8      let level = dict(SCOPEUNIT) in
9      let qualset = {qu ∈ dom dict | (∃qua)(qua ⌢ qualelem = qu) ∧
10                                  (∃q', q")(q' ⌢ q" = level ∧ signaldeflevel(qu) = q')} in
11     if card qualset ≠ 1 then
12       exit("§2.2.2: Идентификатор сигнала обозначает более, чем один (под)сигнал")
13     else
14       (let qual ' ∈ qualset in
15        qual ')))
```

запись : $Sigid_0 \rightarrow Dict \rightarrow Signalqual$

Цель Построить и возвратить *Qual* идентификатора сигнала AS_0. Причина, по которой функция *get-visible-qual* не может быть прямо использована для сигналов, заключается в том, что правила видимости для сигналов несколько сложнее из-за понятия подсигналов

Параметры

sigid	Идентификатор сигналов AS_0

Алгоритм

Строки 1-3	Если *sigid* не является подсигналом, тогда *Qual* может быть получен при помощи функции *get-visible-qual*.
Строка 7	Построить частичный *Qual*, который включает в себя *Qualelem*, содержащий имя сигнала
Строка 8	*level* обозначает единицу подразделения, где используется *sigid*.
Строки 9-10	Полное множество возможных *Quals* для идентификатора сигналов является *Quals*, представляющие подсигналы. Множество *Quals* для подсигналов состоит из подсигналов, которые имеют частичные *Quals* в качестве завершающей части (строка 9) и которые содержатся в определении видимого сигнала *q'*.

Строки 11-15 Если существует один и только один возможный *Qual* для *sigId*, то возвратить этот *Qual*

signaldeflevel(qual) ≜ (3.6.8)

```
1   if s-Kind(qual [len qual]) = SIGNAL then
2     signaldeflevel(get-sur(qual))
3   else
4     qual
```

запись : *Signalqual* → *Signalqual*

Цель Из *Qual*, обозначающего единицу подразделения, которая является определением (под) сигнала, построить *Qual* единицы подразделения, где дается определение выходного сигнала

Параметры

Алгоритм

Строки 1-2 Если последний *Qualelem* содержит класс объектов SIGNAL, то *Qual* (все еще) обозначает определение сигнала, и *Qualelem* удаляется перед повторным применением функции

get-visible-variable(id, sortset, export)(dict) ≜ (3.6.9)

```
1   (let qualset = all-variables-and-synonyms(id, EXPRESSION, sortset)(dict) in
2   let qualset '=
3       {qual ∈ qualset | cases dict(qual):
4                         (mk-VarD(, e,,,)
5                           → export = false ∨ e = EXPORT,
6                         mk-SynD(,)
7                           → false)) in
8   if card qualset '= 1 then
9     (let qual ∈ qualset ' in
10     qual)
11   else
12     exit("§2.2.2: Одна или более переменных должны соответствовать контексту"))
```

запись : Id_0 *Sortqual*-**set** *Bool* → *Dict* → *Qual*

Цель Извлечь *Qual* переменной. *get-visible-varible* применяется только там, где не допустимы синонимы (например, присвоение, параметры IN/OUT и т.д.)

Параметры

id	Идентификатор переменной.
sortset	Множество сортов, которые допустимы в данном контексте.
export	Флаг, указывающий на использование переменной в операции экспорта.

Результат *Qual*, который является уникальным для данного контекста.

Алгоритм

Строка 1 Построить множество *Quals*, состоящее из переменных и синонимов, которые соответствуют одному из сортов в *sortset*.

Строки 2-6 Ограничить множество только до переменных и, если *id* используется в операции экспорта, то переменные этого множества должны быть экспортированы.

Строка 1 Множество должно содержать уникальный *Qual*

3.7 Преобразование процедурных графов и графов процессов

В этом разделе производится преобразование тела процесса или процедуры. Результатом являются *Process-graph*$_1$ или *Procedure-graph*$_1$ и *Dict*, который несет информацию о неявных переменных и выходных сигналах, используемых в теле. Если процесс содержит сервисы, то также возвращаются некоторые определения AS$_1$. Функция входа, преобразующая тело процесса, называется *transform-process-body*, а функция входа, которая преобразует тело процедуры, называется *transform-body* (которая также используется *transform-process-body*, если процесс не содержит декомпозицию сервиса). Для упрощения переходы осуществляются только один раз, что означает, что сокращения в переходах (т.е. вершины выбора, следующее состояние с черточкой и импорт/экспорт) расширяются "на лету", во время преобразования переходов.

Преобразование тела состоит из следующих шагов:

1. Вставка терминаторов в ответы решений и вариантов, так, что каждый *Transition*$_0$AS$_0$ содержит *Termstmt*$_0$.

2. Построение *Labeldict*, т.е. набора меток, используемых в теле, и их соответствующих переходов

3. Построение *Statedict*, т.е. список состояний преобразуется в таблицу отображений, которая является более подходящим представлением при преобразовании состояний. В этой конструкции удаляются множественные появления состояний и состояния с звездочкой.

4. Раскрытие вводов со звездочкой и сохранение звездочки. Добавляются также неявные переходы. Так как сигналы, подразумеваемые экспортом, и сигнал пустой очереди разделяются всеми сервисами, то переходы для этих сигналов не добавляются, пока сервисы не будут объединены.

5. Если процесс содержит сервисы, то *Statedict* всех сервисов сливаются ("состояние в разобранном виде") в *statedict*, где каждый вход является именем нового состояния.

6. Раскрытие условия разрешения и непрерывного сигнала. Раскрытие достигается посредством модификации созданного *Statedict*

7. Преобразование *Statedict* и содержащихся переходов в *Process-graph*$_1$ или *Procedure-graph*$_1$. Во время этого преобразования новые входы, подразумеваемые выражениями импорта, добавляются в *Statedict*

transform-process-body(*body*)(*dict*) ≜ (3.7.1)

```
1   if is-Body0(body) then
2     (let (as1body, bdict) = transform-body(body)(dict) in
3      ({}, as1body, bdict, [])
4    else
5     (let mk-Decomposition0(decll) = body in
6      let (as1dclset, sdict) = transform-decllist(decll)(dict) in
7      let connectmap = service-connectmap(decll, {}, [])(dict) in
8      if is-wf-servicesignals(dict) then
9        (let (as1body, dict') = transform-decomposition-body(dict) in
10        (as1dclset, as1body, sdict + dict', connectmap))
11     else
12       exit("§4.10.2: Сигналы сервиса не корректны"))
```

запись : *Processbody*$_0$ → *Dict* → *Decl*$_1$-set *Process-graph*$_1$ *Dict Processconnection*D

Цель Преобразовать *Processbody*$_0$ в его представление AS$_1$

Параметры

body Тело процесса

Результат Список определений AS_1 (список является не пустым, если процесс содержит сервисы), тело процесса AS_1, *Dict*, несущий информацию о неявных переменных, выходные сигналы и т.д., используемые в теле процесса, а также соединения маршрутов сигналов с маршрутами сигналов сервиса для процесса.

Алгоритм

Строки 1-3 Если процесс не содержит сервисы, то преобразовать тело состояния, содержащееся в процессе (как для процедур), иначе

Строка 5 Пусть *decll* обозначает определения сервиса и определения маршрутов сигналов сервиса.

Строка 6 Преобразовать сервисы и маршруты сигналов. Во время этого преобразования сервисы заменяются определениями, которые они содержат. Их тело состояния представляется как *Statedict* в дескрипторах сервиса (содержащихся в *sdict*).

Строка 8 Совокупность допустимых входных сигналов сервисов должна быть расчленена и никакие сигналы в маршрутах сигналов, внешних по отношению к процессу, не могут быть высокоприоритетными сигналами.

Строка 9 Построить тело процесса AS_1 из *Statedict*, содержащегося в дескрипторах сервиса *dict*.

Строка 10 Возвратить определения AS_1 из сервисов, тела процесса и требований *Dict* из определений и неявных переменных выходных сигналов и т.д., используемых в сервисах

tramansform-body(**mk**-$Body_0$(*trans*, *statelist*))(*dict*) ≜ (3.7.2)

```
1   (let trans' = insert-trans-term(trans) in
2   let statelist' = ⟨insert-state-term(statelist[i]) | 1 ≤ i ≤ len statelist⟩ in
3   let labeldict = build-trans-labeldict(trans')([]) in
4   let labeldict' = build-state-labeldict(statelist')(labeldict) in
5   let statedict = build-statedict(statelist', dict(SCOPEUNIT))([]) in
6   let statedict' = remove-asterisk-input-and-save(statedict)(dict) in
7   let statedict'' = remove-cont-enable-from-statelist(dom statedict)(dict, statedict') in
8   let dict' = dict + [LABELDICT ↦ labeldict',
9                       STATEDICT ↦ statedict''] in
10  let (trans1, dict'') = transform-transition(nil, trans', ⟨⟩)(dict') in
11  let (statebody1, dict''') = transform-statelist(⟨⟩)(dict'') in
12  let transition1 = add-varInit(trans1, dict''') in
13  if is-ProcedureD(dict(dict(SCOPEUNIT))) then
14     (mk-Procedure-graph1(mk-Procedure-start-node1(transition1), statebody1), dict''')
15    else
16     (mk-Process-graph1(mk-Process-start-node1(transition1), statebody1), dict'''))
```

запись : $Body_0 \rightarrow Dict \rightarrow (Procedure\text{-}graph_1 \mid Process\text{-}graph_1)\ Dict$

Цель Преобразовать тело состояния или процедуры

Параметры Тело AS_0, содержащее

trans Начальный переход

statelist Список состояний

Результат Граф процедуры или процесса AS_1 и *Dict*, несущий информацию о неявных переменных и выходных сигналах

Алгоритм

Строка 1 Вставить терминаторы в ответы решений и выборов в начальном переходе

Строка 2 Вставить терминаторы в ответы решений и выборов в списке состояний

Строка 3	Собрать информацию о метка (коннекторах), используемых в начальном переходе.
Строка 4	Собрать информацию о метках, используемых в списке состояний. *labeldict '* отражает связь между метками, используемыми в теле процесса или процедуры, и их последующими переходами (см. определение *Labeldict*).
Строка 5	Преобразовать список состояний в отображение имен состояний в дескриптор *StateD* (см. определение *Statedict* в *Quotdict*). Многократное появление состояний и состояний со звездочкой объединяются во время этого преобразования.
Строка 6	Раскрыть вводы со звездочкой, сохранение звездочки и неявные переходы в *Statedict*.
Строка 7	Раскрыть все условия разрешения и непрерывный сигнал в построенном *Statedict*. После этого раскрытия *Statedict* содержит дескрипторы *ContenablestateD* для удаляемых состояний. Новые состояния представляются дескриптором *StateD*.
Строка 8	Включить *LabelDict* и *Statedict* в *Dict* так, что функциям преобразования необходим только один параметр словаря (вместо трех).
Строка 10	Преобразовать начальный переход в AS_1.
Строка 11	Преобразовать список состояний в множество состояний AS_1.
Строка 12	Модифицировать начальный переход, чтобы включить инициацию неявных переменных, используемых в процессе.
Строки 13-16	Возвратить граф процедуры AS_1, если тело состояния принадлежит процедуре, иначе возвратить граф процесса AS_1. В возвращенном *Dict* (*dict '''*) используются только входы OUTSIGNALS и IMPLIED из *Quotdict*

add-varinit(*trans*, *dict*) ≜ (3.7.3)

```
1    (let qualset = {qual ∈ dom dict | is-VarD(dict(qual)) ∧
2                                  get-sur(qual) = dict(SCOPEUNIT)} in
3     add-default-assign(trans, qualset)(dict))
```

запись : $Transition_1$ *Dict* → $Transition_1$

Цель Изменить начальный переход, чтобы включить инициацию переменных.

Параметры

trans	Начальный переход AS_1.
dict	*Dict*, содержащий соответствующие дескрипторы *VarD* и имя AS_0 переменной, связанной с непрерывным сигналом (это имя является одинаковым для всех процессов)

Результат Измененный переход AS_1.

Строки 1-2	Извлечь множество всех *Quals*, которые обозначают локальные переменные.
Строка 3	Модифицировать переход, чтобы включить присвоения по умолчанию локальным переменным.

add-default-assign(trans, descrset)(dict) ≜ (3.7.4)

```
1   if descrset = {} then
2     trans
3   else
4     (let qual ∈ descrset in
5      let trans' = add-default-assign(trans, descrset \ {qual})(dict) in
6      let mk-VarD(,,, expr1, qual) = dict(qual) in
7      if expr1 = nil then
8        trans'
9      else
10       (let as1id = make-as1-identifier(qual)(dict) in
11        let stmt = mk-Task-node1(mk-Assignment-statement1(as1id, expr1)) in
12        let mk-Transition1(actl, term) = trans' in
13        let trans'' = mk-Transition1(stmt ⌢ actl, term) in
14        if is-SyntypeD(dict(qual)) then
15          (let range = s-Range-condition1(dict(qual)) in
16           let mk-Range-condition1(, cset) = range in
17           if cset = {} then
18             trans''
19           else
20             (let des = mk-Decision-node1(expr1,{mk-Decision-answer1(range, trans'')},
21                                          mk-Else-answer1(trans')) in
22              mk-Transition1(⟨⟩, des)))
23         else
24           trans''))
```

запись : *Transition*$_1$ *Qual*-**set** → *Dict* → *Transition*$_1$

Цель Модифицировать начальный переход для включения присвоений по умолчанию для переменных, локальных по отношению к единице подразделения. Так как для неявных переменных, связанных с переменными экспорта, присутствуют также дескрипторы *VarD*, неявные переменные экспорта автоматически инициализируются.

Параметры

trans Переход, который должен быть модифицирован.

descrset *Quals* локальных переменных.

Результат Модифицированный переход.

Строка 1 Когда просмотр закончен, больше не модифицировать переход.

Строка 4 Пусть *qual* обозначает элемент множества *Quals*.

Строка 5 Построить переход, который является *trans*, модифицированным, чтобы включить инициализацию переменных оставшихся *Quals*.

Строка 6 Произвести декомпозицию дескриптора переменных, чтобы обнаружить начальное выражение (*expr*$_1$) и сорт переменной. Отметьте, что *qual* не используется для получения выражения инициализации (это уже сделано).

Строка 7 Если дескриптор не содержит выражения инициализации, то возвратить немодифицированный переход.

Строка 10 Пусть *as*$_1$*id* обозначает идентификатор AS$_1$ переменной.

Строка 11 Пусть *stmt* обозначает работу, которая содержит присвоение по умолчанию.

Строки 12-13 Пусть *trans''* обозначает модифицированный переход.

Строки 14-17 Если сорт переменных является подтипом, тогда может быть построено решение.

Строка 15 Пусть *range* обозначает диапазон условий подтипа в AS$_1$.

Строки 16-17 Если множество *Condition-items* в *range* является пустым, то нет необходимости в построении подтипа, и поэтому возвращается модифицированный переход.

Строки 20-22 Возвратить новый переход, который состоит из пустого списка операций и решения AS_1, способного выполнить присвоение по умолчанию только в случае, когда удовлетворяется условие диапазона.

3.7.1 Вставка терминаторов в ответы

insert-state-term(**mk-***Statebody*$_0$(*idl*, *stlist*, *tnm*)) ≜ (3.7.1.1)

1 (**let** *stlist* ' = ⟨*insert-spec-term*(*stlist*[*i*]) | 1 ≤ *i* ≤ **len** *stlist* ⟩ **in**
2 **mk-***Statebody*$_0$(*idl*, *stlist* ', *tnm*))

запись : *Statebody*$_0$ → *Statebody*$_0$

Цель Вставить терминаторы в ответы в решениях и вариантах, содержащихся в состоянии AS_0

Параметры Состояние, содержащее

idl Список имен состояний для состояния

Результат Модифицированное состояние AS_0

Алгоритм

Строка 1 Вставить терминаторы во все переходы ввода и
Строка 2 Построить состояние AS_0 (повторно).

insert-spec-term(*stspec*) ≜ (3.7.1.2)

1 **cases** *stspec* :
2 (**mk-***Inputspec*$_0$(*vars*, *enab*, *trans*)
3 → **mk-***Inputspec*$_0$(*vars*, *enab*, *insert-trans-term*(*trans*)),
4 **mk-***Contspec*$_0$(*expr*, *prio*, *trans*)
5 → **mk-***Contspec*$_0$(*expr*, *prio*, *insert-trans-term*(*trans*)),
6 **mk-***Priinput*$_0$(*vars*, *trans*)
7 → **mk-***Priinput*$_0$(*vars*, *insert-trans-term*(*trans*)),
8 T → *stspec*)

запись : *Statespec*$_0$ → *Statespec*$_0$

Цель Вставить терминаторы в ответы в решениях и вариантах, которые содержатся в переходе ввода

Параметры

stspec Переход ввода

Результат Модифицированный переход ввода AS_0

Алгоритм

Строка 2 Если переход ввода является вершиной ввода, тогда вставить терминаторы в переход, следующий за вершиной ввода (*trans*)

Строка 4 Если переход ввода является непрерывным сигналом, тогда вставить терминаторы в переход, следующий за непрерывным сигналом (*trans*).

Строка 6 Если переход ввода является приоритетным вводом, тогда вставить терминаторы в переход, следующий за приоритетным вводом (*trans*).

Строка 8 Иначе (в случае вершины сохранения) возвратить переход неизмененным

insert-trans-term(mk-*Transition*$_0$(*actl*, *term*)) ≜ (3.7.1.3)

```
1   (if actl = ⟨⟩ then
2      mk-Transition0(⟨⟩, term)
3    else
4      (let mk-Actstmt0(l, act) = hd actl in
5      if is-Decision0(act) ∨ is-Option0(act) then
6         (let (answerl, els) = cases act:
7                                 (mk-Decision0(, ansl, els)
8                                    → (ansl, els),
9                                  mk-Option0(, ansl, els)
10                                   → (ansl, els)) in
11        let (actl ', answerl ', els ') = insert-answer-term(tl actl, term, answerl, els) in
12        let act ' = cases act:
13                   (mk-Decision0(q,,)
14                     → mk-Actstmt0(l, mk-Decision0(q, answerl ', els ')),
15                   mk-Option0(q,,)
16                     → mk-Actstmt0(l, mk-Option0(q, answerl ', els '))) in
17        let mk-Transition0(actl ", t) = insert-trans-term(mk-Transition0(actl ', term)) in
18        mk-Transition0(⟨act '⟩ ⌢ actl ", t))
19       else
20       if tl actl = ⟨⟩ ∧ term = nil then
21          exit("§2.6.7.1: Терминатор потерян при переходе")
22         else
23         (let mk-Transition0(actl ", t) = '
24              insert-trans-term(mk-Transition0(tl actl, term)) in
25          mk-Transition0(⟨hd actl⟩ ⌢ actl ", t))))
```

запись : *Transition*$_0$ → *Transition*$_0$

Цель Вставить терминаторы в ответы вершин принятия решений и выбора в списке действий

Параметры Переход, содержащий

actl Список действий

term Необязательный терминатор, следующий за списком действий

Результат Модифицированный переход AS$_0$

Алгоритм

Строка 1 Когда просмотр завершен, возвратить переход, содержащий пустой список действий (функция рекурсивна).

Строка 4 Произвести декомпозицию первого предложения действия.

Строка 5 Если действие является решением или выбором, тогда

Строки 6-9 Раскрыть список ответов (*answerl*) и else-часть (*els*) из решения или выбора.

Строка 11 Вставить терминаторы в список ответов и в else-часть. Если терминатор вставлен (соединение), тогда оставшаяся часть списка действий (tl *actl*) модифицирована (*actl '*), чтобы включить метку.

Строки 12-15 Построить модифицированную вершину принятия решения или выбора.

Строка 17 Вставить терминаторы в оставшуюся часть списка действий.

Строка 18 Возвратить переход, содержащий модифицированное решение или выбор (*act '*), остаток списка действий (*actl "*) и терминатор.

Строки 20-25 Если первое действие не является вершиной принятия решения или выбором, тогда

Строки 20-21 Если действие является последним действием и отсутствует терминатор, тогда оно является ошибочным.

Строки 23-25 Вставить терминаторы в оставшуюся часть списка действий и составить модифицированный переход

insert-term(*actl*, *term*, *answerl*, *els*) ≜ (3.7.1.4)

```
    (if answerl = ⟨⟩ then
      if els = nil then
        (actl, ⟨⟩, nil)
      else
      (let mk-Elsepart0(trans) = els in
       let (actl ', trans ') =
          cases trans:
           (nil
             → (let (acl, term') = insert-join(actl, term) in
                 (acl, mk-Transition0(⟨⟩, term'))),
           mk-Transition0(actl ", term")
             → if term" = nil then
                 (let (actl "', term "') = insert-join(actl, term) in
                  (actl "', insert-trans-term(mk-Transition0(actl ", term "'))))
               else
                 (actl, insert-trans-term(trans))) in
       (actl ', ⟨⟩, mk-Elsepart0(trans')))
    else
    (let (actl ', answerrest, els') = insert-answer-term(actl, term, tl answerl, els) in
     let mk-Answer0(vset, trans) = hd answerl in
     let (actlist, trans") =
        cases trans:
         (nil
           → (let (acl, term) = insert-join(actl ', term) in
               (acl, mk-Transition0(⟨⟩, term))),
         mk-Transition0(actl ", term")
           → if term" = nil then
               (let (actl "', term "') = insert-join(actl ', term) in
                (actl "', insert-trans-term(mk-Transition0(actl ", term "'))))
             else
               (actl ', insert-trans-term(trans))) in
     (actlist, ⟨mk-Answer0(vset, trans")⟩ ⌢ answerrest, els')))
```

запись : $Actstmt_0^*$ [$Termstmt_0$] $Answer_0^*$ [$Elsepart_0$] → $Actstmt_0^*$ $Answer_0^*$ [$Elsepart_0$]

Цель Вставить терминаторы в ответы вершин принятия решения или выбора

Параметры

actl	Список действий, который следует за вершиной принятия решения или выбора
term	Терминатор, который следует за вершиной принятия решения или выбора и за списком действий
answerl	Список ответов
els	Необязательная else-часть

Результат

- Список действий, который следует за вершиной принятия решения или выбора. Этот список действий модифицируется, чтобы включить метку, если он не является пустым и если отсутствуют потерянные терминаторы в ответах и в else-части
- Список ответов, содержащих терминаторы
- Else-часть, содержащая терминатор (если else-часть не опущена)

Алгоритм

Строки 1-17	Когда просмотр списка ответов завершен, модифицировать else-часть.
Строка 2	Если else-часть опущена, то возвратить список действий, пустой список ответов и не возвращать else-часть.
Строка 5	Если else-часть специфицирована, то произвести ее декомпозицию.

Строки 6-17	Пусть *actl'* обозначает новый список действий, который следует за вершиной принятия решения или выбора, пусть *trans'* обозначает новый переход в else-части, которая содержит терминатор.
Строка 8	Если переходы не специфицированы, то построить соединение (*term*) и вставить метку в список действий, который следует за вершиной принятия решения или выбора (*acl*). Переход else-части является пустым списком действий и соединением (строка 10).
Строки 11-14	Если переход специфицирован в else-части, но терминатор опущен, тогда вставить метку в список действий (*actl'''*), который следует за решением или выбором, построить соединение (*term'''*), вставить терминаторы в список действий в переходе (строка 14).
Строка 16	Если переход, содержащий терминатор, специфицирован, тогда вставить терминатор в список действий перехода и не производить модификаций списка действий (*actl*), следующих за решением или выбором.
Строка 17	Возвратить модифицированный список действий, пустой список ответов и модифицированную else-часть.
Строка 19	Вставить терминаторы в оставшиеся ответы.
Строка 20	Произвести декомпозицию первого ответа в списке.
Строки 21-31	Повторить то же для первого ответа в списке ответов, что сделано для else-части в строках 6-16.
Строка 32	Возвратить новый список действий, который следует за решением или выбором, первый из модифицированных ответов, связанный с оставшейся частью модифицированных ответов, и модифицированную else-часть (если специфицирована)

insert-join(*actl*, *term*) ≜ (3.7.1.5)

```
1   (if actl = ⟨⟩ then
2     if term = nil then
3       exit("§2.6.7.1: Терминатор потерян при переходе")
4     else
5       (let mk-Termstmt0(,t) = term in
6        (⟨⟩, mk-Termstmt0(nil, t)))
7   else
8    (let mk-Actstmt0(l, act) = hd actl in
9     let l' = if l = nil then create-unique-name() else l in
10    ((mk-Actstmt0(l', act))⌢ tl actl, mk-Termstmt0(nil, mk-Join0(l')))))
```

запись : $Actstmt_0^*\ [Termstmt_0] \rightarrow Actstmt_0^*\ Termstmt_0$

Цель

Из списка действий и последующего терминатора построить соединение к первому действию в списке действий или, если он пуст, к терминатору. Соединение используется как терминатор в ответах, которые не являются заканчивающими решениями (см. *insert-answer-term*)

Параметры

actl	Заданный список действий
term	Терминатор, который следует за списком действий

Результат

Список действий, где первое действие имеет ту же метку, что и построенное соединение, и построенное соединение

Алгоритм

Строка 1	Если список действий пуст, тогда
Строки 2-6	Терминатор должен присутствовать, и, если это так, то
Строки 5-6	Возвратить тот же (но не помеченный) терминатор, ка определенный в переходе, следующим за решением или выбором.
Строка 8	Произвести декомпозицию первого действия в списке действий, следующим за объемлющим решением или выбором.

Строка 9	Извлечь метку из действия и, если оно не содержит меток, то создать ее.
Строка 10	Возвратить новый список действий, который содержит эту метку, и возвратить предложение терминатора, содержащее связь к этой метке

3.7.2 Построение Labeldict

build-state-labeldict(*statebody*)(*labeldict*) ≜ (3.7.2.1)

```
1   (if statebody = ⟨⟩ then
2      labeldict
3    else
4      (let mk-Statebody0(, specl, ) = hd statebody in
5       let specl ' = ⟨specl[i] | 1 ≤ i ≤ len specl ∧ ¬ is-Savespec0(specl[i])⟩ in
6       let labeldict ' = build-spec-labeldict(specl ')(labeldict) in
7       build-state-labeldict(tl statebody)(labeldict ')))
```

запись : $Statebody_0^* \rightarrow Labeldict \rightarrow Labeldict$

Цель Из тела состояний процесса, процедуры или сервиса собрать метки и связанные с ними переходы, чтобы сформировать *Labeldict* (см. определение *Labeldict*)

Параметры

statebody	Список состояний
labeldict	Метки, собранные до сих пор (функция рекурсивна)

Результат Построенный *Labeldict*

Алгоритм

Строка 1	Когда просмотр закончен, возвратить построенный *labeldict*.
Строка 4	Произвести декомпозицию первого состояния в списке
Строка 5	Извлечь все переходы ввода, которые не являются сохранениями.
Строка 6	Обновить *labeldict*, чтобы включить метки в эти переходы ввода.
Строка 7	Собрать метки в оставшихся состояниях

build-spec-labeldict(*speclist*)(*labeldict*) ≜ (3.7.2.2)

```
1   (if speclist = ⟨⟩ then
2      labeldict
3    else
4      (let trans = s-Transition0(hd speclist) in
5       let labeldict ' = build-trans-labeldict(trans)(labeldict) in
6       build-spec-labeldict(tl speclist)(labeldict ')))
```

запись : $Statespec_0^* \rightarrow Labeldict \rightarrow Labeldict$

Цель Из переходов, связанных с состоянием (переходов вводов), собрать метки и связанные с ними переходы, чтобы сформировать *Labeldict* (см. определение *Labeldict*)

Параметры

speclist	Список переходов ввода
labeldict	Метки, собранные до сих пор (функция рекурсивна)

Результат Построенный *Labeldict*

Алгоритм

Строка 1	Когда просмотр закончен, возвратить построенный *Labeldict*
Строки 4-5	Изменить *labeldict*, чтобы включить метки в переход первого перехода ввода.
Строка 6	Собрать метки для оставшихся переходов ввода

build-trans-labeldict(mk-*Transition*$_0$(*actl*, *term*))(*labeldict*) ≜ (3.7.2.3)

```
1   (if actl = ⟨⟩ then
2      (if term = nil then
3         labeldict
4       else
5         (let mk-Termstmt0(l,) = term in
6          if l = nil then
7             labeldict
8           else
9             if l ∈ dom labeldict then
10               exit("§2.6.6: Метки не являются уникальными")
11              else
12               labeldict + [l ↦ mk-Transition0(⟨⟩, term)]))
13    else
14     (let mk-Actstmt0(l, a) = hd actl in
15      let labeldict '=
16          if l = nil then
17             labeldict
18           else
19             if l ∈ dom labeldict then
20               exit("§2.6.6: Метки не являются уникальными")
21              else
22               labeldict + [l ↦ mk-Transition0(actl, term)] in
23      let labeldict "=
24         cases a :
25         (mk-Decision0(, ansl, els)
26            → build-answerlist-labeldict(ansl, els)(labeldict '),
27          mk-Option0(, ansl, els)
28            → build-answerlist-labeldict(ansl, els) (labeldict '),
29          T→ labeldict ') in
30      build-trans-labeldict(mk-Transition0(tl actl, term))(labeldict ")))
```

запись : *Transition*$_0$ → *Labeldict* → *Labeldict*

Цель Из перехода собрать метки и связанные с ними переходы, чтобы сформировать *Labeldict* (см. определение *Labeldict*)

Параметры Переход, содержащий

actl	Список предложений действия
term	Предложение терминатора
labeldict	Метки, собранные до сих пор (функция рекурсивна)

Результат Построенный *Labeldict*

Алгоритм

Строки 1-12	Когда закончен просмотр списка предложений действия, тогда рассмотреть терминатор
Строка 5	Произвести декомпозицию предложения терминатора, чтобы извлечь метку (*l*).
Строка 6	Если метка отсутствует, то возвратить построенный *labeldict*.

Строка 9	Если метка уже существует в *Labeldict*, то метка была использована дважды.
Строка 12	Возвратить *Labaledict*, который дополнен, чтобы включить метку в предложение терминатора. $Transition_0$, который следует за меткой, имеет пустой список предложений действия и предложение терминатора
Строка 14	Произвести декомпозицию первого предложения действия в списке.
Строки 15-22	Пусть *labeldict'* обозначает *labeldict*, который модифицирован, чтобы включить метку первого предложения действия (если он существует и его метка является уникальной).
Строки 23-29	Пусть *labeldict''* обозначает *labeldict*, который далее изменяется, чтобы включить метки ответов в случае действия принятия решения или выбора.
Строка 30	Собрать метки в оставшихся предложениях действия

build-answerlist-labeldict (*answerl, els*)(*labeldict*) ≜ (3.7.2.4)

```
 1  (if answerl = () then
 2    if els = nil then
 3      labeldict
 4    else
 5      (let mk-Elsepart0(trans) = els in
 6        build-trans-labeldict (trans)(labeldict))
 7  else
 8    (let mk-Answer0(, trans) = hd answerl in
 9      let labeldict ' = build-trans-labeldict (trans)(labeldict) in
10      build-answerlist-labeldict (tl answerl, els)(labeldict '))
```

запись: $Answer_0^*$ [$Elsepart_0$] → *Labeldict* → *Labeldict*

Цель Из ответов в действии принятия решения или выбора собрать метки и связанные с ними переходы, чтобы сформировать *Labeldict* (см. определение *Labeldict*)

Параметры

answerl	Список ответов
els	Необязательная else-часть
labeldict	Метки, собранные до сих пор

Результат Построенный *Labeldict*

Алгоритм

Строки 2-6	Когда просмотрен список ответов, рассмотреть *else*-часть.
Строка 2	Если действие принятия решения или выбора не содержит else-части, тогда возвратить *labeldict*.
Строка 5	Извлечь переходы из *else*-части.
Строка 6	Изменить *labeldict*, чтобы включить метки в переход *else*-части.
Строка 8	Извлечь переход первого ответа в списке.
Строка 9	Изменить *labeldict*, чтобы включить метки перехода.
Строка 10	Собрать метки оставшихся ответов

3.7.3 Построение Statedict

build-statedict (*statelist, scope*)(*statedict*) ≜ (3.7.3.1)

```
 1  (if statelist = ◊ then
 2    statedict
 3   else
 4    (let mk-Statebody0(statenml, stspec, tailname) = hd statelist in
 5     let speclist = ⟨(scope, stspec [i]) | 1 ≤ i ≤ len stspec⟩ in
 6      cases statenml :
 7      (mk-Statenamelist0(nmlist)
 8       → if tailname ≠ nil ∧ elems nmlist ≠ {tailname} then
 9          exit("§2.6.3: Имя, завершающее состояние, должно быть таким же, как и открывающее" )
10         else
11          (let statedict' = insert-state-names (nmlist, speclist)(statedict) in
12           build-statedict (tl statelist, scope)(statedict')),
13       mk-Starredlist0(nmlist)
14       → if tailname ≠ nil then
15            exit("§2.6.3: Имя, завершающее описание, не допустимо в состоянии со звездочкой")
16           else
17            (let totaldict = build-statedict (tl statelist, scope)(statedict) in
18             insert-starred (nmlist, speclist)(totaldict))))
```

запись: *Statebody0** *Qual* → *Statedict* → *Statedict*

Цель Из тела состояния процесса, процедуры или сервиса собрать состояния и связанные с ними переходы, чтобы сформировать *Statedict* (см. определение *Statedict*)

Параметры

statebody	Список состояний
scope	*Qual*, обозначающий единицу подразделения, содержащую состояния. Каждый переход из *Statedict* должен содержать информацию о контексте, в котором разработан этот переход (т.к. *Stadedicts* сервисов объединяются в один *Statedict* перед тем, как они преобразуются в AS_1)
statedict	Состояния, рассмотренные до сих пор (функция рекурсивна)

Результат Построенный *Statedict*

Алгоритм

Строка 1	Когда просмотр закончен, возвратить построенный *Statedict*.
Строка 4	Произвести декомпозицию первого состояния в списке.
Строка 5	Построить список *Specs* для состояния (см. определение *Statedict*), т.е. список пар информации контекста и перехода ввода для состояния.
Строки 7-12	Если список имен состояний определен в состоянии, тогда
Строка 9	Если завершающее имя определено, тогда список имен состояний должен состоять из этого имени.
Строка 11	Для каждого из имен состояний в списке добавить дескриптор *StateD* в *Statedict*.
Строка 12	Построить дескрипторы *StateD* для оставшихся состояний.
Строки 13-18	Если список имен состояний со звездочкой определен, тогда
Строка 14	Завершающее имя не разрешается.
Строка 17	Построить *Statedict* для оставшихся состояний (полный *statedict*) до того, как
Строка 18	Добавляются переходы ввода состояний со звездочкой

insert-state-names (*namelist, stspec*)(*statedict*) ≜ (3.7.3.2)

```
 1  (if namelist = ◊ then
 2    statedict
 3   else
 4    (let nm = hd namelist in
 5     let stspec' =
 6        if nm ∈ dom statedict then
 7          (let mk-StateD(specl, ) = statedict (nm) in
 8           specl)
 9         else
10          ◊ in
11     let statedict' = statedict + [nm ↦ mk-StateD(stspec ⁀ stspec', nil)] in
12     if nm ∈ elems tl namelist then
13        exit("§2.6.3: Имена состояний в состоянии должны быть уникальными")
14       else
15        insert-state-names (tl namelist, stspec)(statedict')))
```

запись: *Statenamelist*$_0$ *Speclist* → *Statedict* → *Statedict*

Цель Изменить *Statedict*, чтобы включить переходы ввода вершины состояния.

Параметры

namelist Список имен состояний, которые специфицированы в вершине состояния

stspec Список *Specs*, который содержит переходы ввода, и который, для каждого состояния в списке, добавляется в *Statedict*

Результат Измененный *Statedict*

Алгоритм

Строка 1 Когда закончен просмотр списка имен состояний, возвратить измененный *Statedict* (функция рекурсивна).

Строка 4 Пусть *nm* обозначает первое имя в списке.

Строки 5-8 Если это имя уже имеет вход в *statedict*, тогда *stspec'* обозначает переходы ввода, связанные с каждым входом, иначе *stspec'* обозначает пустой список переходов ввода.

Строка 11 Пусть *statedict'* обозначает *Statedict*, измененный, чтобы включить, как переходы ввода, которые уже существовали (*stspec'*), так и новые (*stspec*). nil в *StateD* указывает, что состояние не является неявным состоянием, предполагаемым выражением импорта (см. определение *Statedict*).

Строки 12-13 Список имен состояний, специфицированный в состоянии, должен содержать различающиеся имена состояний.

Строка 15 Рассмотреть оставшиеся имена в списке имен состояний

insert-starred (*namelist, stspec*)(*totaldict*) ≜ (3.7.3.3)

```
 1  (if card elems namelist ≠ len namelist ∨
 2    ¬(elems namelist ⊆ dom totaldict) ∨
 3    elems namelist = dom totaldict ∨
 4    totaldict = [] then
 5     exit("§4.4: Недопустимое состояние со звездочкой")
 6    else
 7     (let restset = dom totaldict - elems namelist in
 8      let namelist' = ⟨nm | nm ∈ dom restset⟩ in
 9      insert-state-names (namelist', stspec)(totaldict)))
```

запись: *Statenamelist*$_0$ *Speclist* → *Statedict* → *Statedict*

Цель Изменить *Statedict*, чтобы включить переход ввода из состояния со звездочкой

Параметры

namelist Имена, задействованные в состоянии со звездочкой
stspec *Specs*, содержащие переходы ввода в состоянии со звездочкой.

Результат Измененный *Statedict*

Алгоритм

Строка 1 Имена в списке имен состояний должны быть уникальными, и
Строка 2 Они не должны быть только появлениями имени состояния, и
Строка 3 Они не должны включать всех состояний, и
Строка 4 Должно быть по крайней мере оно состояние, которое не является состоянием со звездочкой.
Строка 7 Извлечь множество состояний, которое покрывается состоянием со звездочкой.
Строка 8 Преобразовать множество в список.
Строка 9 Для образуемого списка состояний добавить *Specs* в *Statedict* (обычным способом)

3.7.4 Раскрытие ввода со звездочкой, сохранения со звездочкой и неявных переходов

В этом разделе звездочки вершин ввода и вершин сохранения удаляются из тела процесса, процедуры или сервиса, осуществляется проверка, что каждое состояние содержит не более одной звездочки. Так как условия разрешения, непрерывные состояния и импортируемые выражения еще не были удалены, то замещение звездочек не включает сигнала пустой очереди и сигналов, подразумеваемых импортом.

remove-asterisk-input-and-save (*statedict*)(*dict*) ≜ (3.7.4.1)

1 [*stnm* ↦ *remove-asterisk-from-state* (*statedict* (*stnm*))(*dict*) | *stnm* ∈ **dom** *statedict*]

запись: *Statedict* → *Dict* → *Statedict*

Цель Удалить ввод со звездочкой и сохранение со звездочкой из тел состояний процесса, процедуры или сервиса

Параметры

statedict *Statedict*, представляющий тела состояний

Результат Модифицированный *Statedict*

Алгоритм Построить *Statedict* (отображение), где удаляется каждое состояние, которое имеет ввод со звездочкой и сохранение со звездочкой

remove-asterisk-from-state(mk-*StateD*(*speclist*,))(*dict*) ≜ (3.7.4.2)

```
1  (let (speclist', saveset, inputset, priinput) =
2      remove-asterisk-from-spec(speclist, {}, {}, {}, false)(dict) in
3   let (inputsignals, prisignals) =
4      (let qual = process-or-service-level (dict (SCOPEUNIT)) in
5       cases dict (qual):
6       (mk-ProcessD(, sig,, )          → (sig, {}),
7        mk-ServiceD(,,, sig, psig)     → (sig, psig))) in
8   if priinput = prisignals ∧ (saveset ∪ inputset) ⊆ inputsignals then
9      (let (expimpinput, ) = import-export-signals (dict) in
10      let implicitinput = inputsignals \ saveset \ priinput \ inputset \ expimpinput in
11      let implicittrans = as0-implicit-transitions (implicitinput)(dict) in
12      mk-StateD(speclist' ⇝ implicittrans, nil))
13     else
14      exit("§2.6.3: Сигналы в вершинах состояний некорректны"))
```

запись: *StateD* → *Dict* → *StateD*

Цель Удалить ввод со звездочкой и сохранение со звездочкой из состояния и включить неявные переходы для неопределенных сигналов в *Speclist*

Параметры Дескриптор *Statedict* (*StateD*), содержащий

speclist *Speclist*, содержащий *Statespec*$_0$s состояния

Результат Модифицированный дескриптор *StateD*

Алгоритм

Строка 1 Построить модифицированный *Speclist* и извлечь сигналы, определенные в вводах, сигналы определенные в сохранении и сигналы, определенные в приоритетных вводах

Строки 3-7 Получить полную совокупность допустимых входных сигналов для процесса или сервиса.

Строка 8 Сигналы в приоритетных вводах состояния должны быть эквивалентны приоритетным сигналам для сервиса, а сигналы, определенные в сохранении и других вводах, должны быть включены в полную совокупность допустимых входных сигналов.

Строка 9 Пусть *expimpinput* обозначает неявные сигналы xtQUERY и xtREPLY, полученные в сокращениях импорта и экспорта, и сигнал *emptyq*.

Строка 10 Сигналы, для которых неявные переходы должны быть созданы, являются сигналами в полной совокупности допустимых входных сигналов минус сигналы, задействованные в сохранении, минус высокоприоритетные сигналы, минус сигналы, задействованные в вводе, минус неявные сигналы.

Строка 11 Создать неявные переходы.

Строка 12 Создать модифицированный дескриптор *StateD*

as$_0$*-implicit-transitions* (*signalqualset*)(*dict*) ≜ (3.7.4.3)

```
1  if signalqualset = {} then
2    ◊
3   else
4    (let signalqual ∈ signalqualset in
5     let inputvars = as0-inputvars ({signalqual})(dict) in
6     let input = mk-Inputspec0(inputvars, nil,
7                          mk-Transition0(◊, mk-Termstmt0(nil, mk-Nextstate0(nil)))) in
8     ((dict(SCOPEUNIT)), input) ⇝ as0-implicit-transitions (signalqualset \ {signalqual})(dict))
```

запись: *Signalqual*-set → *Dict* → *Speclist*

Цель Построить неявные переходы для состояния

Параметры

signalqualset Множество сигналов, для которых должны быть созданы неявные переходы

Результат *Speclist*, содержащий неявные переходы

Алгоритм

Строка 1 Когда просмотр закончен, возвратить *Speclist* ввода.

Строка 4 Пусть *signalqual* обозначает следующий сигнал, подлежащий обработке.

Строка 5 Построить список параметров для сигналов.

Строка 6 Построить переход ввода для сигнала. Переход содержит только черточку следующего состояния.

Строка 8 Построить *Spec* и объединить его с *Speclist* для оставшихся сигналов

remove-asterisk-from-spec (*speclist, sigset, saveset, priset, asterisk*) (*dict*) ≜ (3.7.4.4)

```
(if speclist = ◊ then
    .(◊, sigset, saveset, priset)
    else
     (let (q, spec) = hd speclist in
      cases spec :
      (mk-Savespec0(sigl)
        → if is-Starred0(sigl) then
             if asterisk then
              exit("§4.7: У состояния более одной звездочки для ввода или сохранения")
             else
              (let (specrest, inputtot, savetot, pritot) =
                   remove-asterisk-from-spec(tl speclist, sigset, saveset, priset, true)(dict) in
               let sigq = all-input-signals(dict(SCOPEUNIT))(dict) in
               let (expimpinput, ) = import-export-signals(dict) in
               let sigq' = sigq \ inputtot \ savetot \ pritot \ expimpinput in
               let as0siglist = ⟨as0-id(sig) | sig ∈ sigq'⟩ in
               let spec' = if as0siglist = ◊ then
                           ◊
                          else
                           ⟨(q, mk-Savespec0(as0siglist))⟩ in
              (specrest ⌢ spec', inputtot, sigq' ∪ savetot, pritot))
           else
            (let sigq = transform-signallist (sigl)(dict) in
             let (specrest, inputtot, savetot, pritot) =
                 remove-asterisk-from-spec(tl speclist, sigset, saveset ∪ sigq, priset, asterisk)(dict) in
             if sigq ∩ (sigset ∪ saveset ∪ priset) = {} then
              (specrest ⌢ ⟨hd speclist⟩, inputtot, savetot, pritot)
             else
             exit("§2.6.3: Сигналы состояния не согласуются")),
```

```
30          mk-Prlinput0(inp, ),
31          mk-Inputspec0(inp, enab, trans)
32           → if is-Starred0(inp) then
33               if asterisk then
34                   exit("§4.7: У состояния более одной звездочки для ввода или сохранения")
35                  else
36                   (let (specrest, inputtot, savetot, pritot) =
37                        remove-asterisk-from-spec(tl speclist, sigset, saveset, pritot, true)(dict) in
38                   let sigq = all-input-signals (dict (SCOPEUNIT))(dict) in
39                   let (expimpinput, ) = import-export-signals (dict) in
40                   let sigq' = sigq \ inputtot \ savetot \ pritot \ expimpinput in
41                   let inpvars = as0-inputvars(sigq')(dict) in
42                   let spec' = if inpvars = ◊ then
43                                   ◊
44                                  else
45                                   ⟨⟨q, mk-Inputspec0(inpvars, enab, trans)⟩⟩ in
46                    (specrest ⇝ spec', sigq' ∪ inputtot, savetot, pritot))
47              else
48              (let sigq = {signal-qual (id)(dict) | mk-Inputvars0(id, ) ∈ elems inp} in
49               let (sigset', priset') = if is-Prlinput0(spec) then
50                                         (sigset, priset ∪ sigq)
51                                        else
52                                         (sigset ∪ sigq, priset) in
53               let (specrest, inputtot, savetot, pritot) =
54                   remove-asterisk-from-spec(tl speclist, sigset', saveset, priset', asterisk)(dict) in
55               if sigq ∩ (sigset ∪ saveset ∪ priset) = {} then
56                   (specrest ⇝ (hd speclist), inputtot, savetot, pritot)
57               else
58                exit("§2.6.3: Сигналы состояния не согласуются")),
59          T → remove-asterisk-from-spec (tl speclist, sigset, saveset, priset, asterisk)(dict))))
```

запись: *Statespec0** *Signalqual*-set *Signalqual*-set *Signalqual*-set *Bool* → *Dict* →
*Statespec0** *Signalqual*-set *Signalqual*-set *Signalqual*-set

Цель Удалить ввод со звездочкой и сохранение со звездочкой из состояния. В дополнение к вершинам ввода функция также возвращает множество сигналов, задействованных в вершинах ввода, множество сигналов, задействованных в вершинах сохранения, и множество сигналов, задействованных в вершинах приоритетного ввода

Параметры

speclist	Список оставшихся вводов и сохранений, которые будут обрабатываться (функция рекурсивна)
sigset	*Quals* сигналов, которые уже распознаны в вершинах ввода
saveset	*Quals* сигналов, которые уже распознаны в вершинах сохранения
priset	*Quals* сигналов, которые уже распознаны в вершинах приоритетного ввода
asterisk	Флаг, указывающий, распознаны ли уже сохранение со звездочкой или ввод со звездочкой

Результат Модифицированный список вводов и сохранений, построенный до сих пор, полное множество *Qual* сигналов ввода, полное множество *Qual* сигналов сохранения и множество сигналов с высоким приоритетом, принятых в этом состоянии

Алгоритм

Строка 1	Когда просмотр закончен, то возвратить полное множество *Qual* сигналов ввода, полное множество *Qual* сигналов сохранения и множество сигналов с высоким приоритетом
Строка 4	Произвести декомпозицию первого *Spec* состояния.

Строки 6-27	Если *Statespec*$_0$ в первом *Spec* является вершиной сохранения, тогда
Строка 6	Если звездочка специфицирована в вершине сохранения, тогда
Строка 8	Если сохранение со звездочкой или ввод со звездочкой уже встречались, тогда - ошибка.
Строка 11	Удалить ввод и сохранение со звездочкой из оставшихся вершин ввода и сохранения. true указывает, что сохранение (или ввод) со звездочкой уже встречались. *inputtot* и *savetot* являются явно определенными полными множествами сигналов ввода и сохранения.
Строка 13	Извлечь *Quals* сигналов, которые содержатся в полном допустимом множестве сигналов ввода для сервиса или процесса.
Строки 14-15	Множество сигналов, которые должны быть определены в вершине сохранения, являются сигналами в полном допустимом множестве сигналов ввода (*sigq*) минус сигналы, определенные в вершинах ввода (*inputtot*), минус сигналы, определенные в других вершинах сохранения (*savetot*), минус сигналы с высоким приоритетом, минус сигналы, подразумеваемые при экспорте/импорте.
Строка 16	Преобразовать множество *Quals* сигналов в список AS_0 идентификаторов.
Строка 17	Построить *Spec* нового сохранения при условии существования каких-либо сигналов для сохранения.
Строка 21	Возвратить *Spec*, объединенный с оставшимися *Spec*, полным множеством сигналов ввода, полным множеством сигналов сохранения и множеством сигналов с высоким приоритетом.
Строка 23	Если сохранение содержит список сигналов, тогда преобразовать список сигналов для получения специфицированных сигналов (*sigq*).
Строка 24	Удалить ввод и сохранение со звездочкой из оставшихся вершин ввода и сохранения, где множество сигналов сохранения, полученное до сих пор, изменяется, чтобы включить сигналы этой вершины сохранения.
Строка 26	Сигналы в сохранении должны быть отделены от других сигналов состояния.
Строка 27	Добавить *Spec* к результирующему списку *Specs* (и оставить его неизмененным).
Строки 31-59	Если *Statespec*$_0$ в первом *Spec* является вершиной ввода, тогда
Строка 32	Если звездочка специфицирована в вершине ввода, тогда
Строка 33	Если сохранение со звездочкой или ввод со звездочкой уже встречались, тогда ошибка
Строка 36	Удалить ввод и сохранение со звездочкой из оставшихся вершин ввода и сохранения. true означает, что сохранение (или ввод) со звездочкой уже встречались.
Строка 38	Извлечь *Quals* сигналов, которые содержатся в полном допустимом множестве сигналов ввода для сервиса или процесса.
Строки 39-40	Множество сигналов, которые должны быть определены в вершине сохранения, являются сигналами в полном допустимом множестве сигналов ввода (*sigq*) минус сигналы, определенные в вершинах ввода (*inputtot*), минус сигналы, определенные в других вершинах сохранения (*savetot*), минус сигналы с высоким приоритетом, минус сигналы, подразумеваемые при экспорте/импорте.
Строка 41	Преобразовать множество *Quals* сигналов в список *Inputvars*$_0$, где опущены переменные для приема значений (nil).
Строка 42	Построить новый *Spec* ввода при условии, что не существует каких-либо явно задействованных сигналов.
Строка 46	Возвратить *Spec*, объединенный с оставшимися *Spec*, полным множеством сигналов ввода и полным множеством сигналов сохранения.
Строка 48	Если ввод содержит список *Inputvars*$_0$, тогда извлечь *Quals* сигналов, содержащихся в них
Строка 49	если ввод является приоритетным вводом, тогда добавить сигналы в множество приоритетных вводов (*priset*), иначе добавить в множество нормальных сигналов (*sigset*).
Строка 53	Удалить ввод и сохранение со звездочкой из оставшихся вершин ввода и сохранения

Строка 55	Сигналы в вводе должны быть отделены от других сигналов в состоянии.
Строка 56	Добавить *Spec* к результирующему списку *Specs* (и оставить его неизмененным).
Строка 59	Если $Statespec_0$ является непрерывным сигналом, тогда продолжить со следующим $Statespec_0$

as$_0$*-inputvars* (*starinput*)(*dict*) ≜ (3.7.4.5)

```
1   (if starinput = {} then
2     ◊
3    else
4     (let qual ∈ starinput in
5      let (nm, ) = qual[len qual] in
6      let id = mk-Id0(◊, nm) in
7      let t = cases dict(qual) :
8              (mk-SignalD (tli,,)        → tli,
9               mk- TimerD (tli,)→ tli) in
10     let vnum = len t in
11     let inpu = mk-Inputvars0(id, ⟨nil | 1 ≤ i ≤ vnum⟩) in
12     ⟨inpu⟩ ⌢ as0-inputvars(starinput \ {qual})(dict)))
```

запись: *Signalqual*-set → *Dict* → $Inputvars_0$*

Цель Построить AS_0 списки переменных для сигналов, которые принимаются в вершине ввода со звездочкой или принимаются в неявных переходах.

Параметры

starinput Множество допустимых сигналов ввода для процесса, который не был явно определен ни в вершине ввода, ни в вершине высокоприоритетного ввода, ни в вершине сохранения

Алгоритм

Строка 1	Когда просмотр завершен, тогда ничего не возвращать (функция рекурсивна)
Строка 4	Пусть *qual* обозначает *Qual* сигнала в этом множестве.
Строка 5	Извлечь имя сигнала.
Строка 6	Построить идентификатор сигнала.
Строки 7-9	Пусть *t* обозначает список сорта сигнала или таймера.
Строка 10	Пусть vnum обозначает число значений, передаваемых сигналом.
Строка 11	Построить AS_0 список переменных ввода, где каждая переменная равняется nil (опущена) и
Строка 12	Возвратить его вместе со списками оставшихся сигналов

3.7.5 Распространение непрерывных сигналов и условия разрешения

remove-cont-enable-from-statelist (*stateset*)(*dict, statedict*) ≜ (3.7.5.1)

```
1   (if stateset = {} then
2     statedict
3    else
4     (let state ∈ stateset in
5      let mk-StateD(speclist, ) = statedict(state) in
6      let statedict' = remove-cont-enable-from-state (speclist, state)(dict, statedict) in
7      remove-cont-enable-from-statelist (stateset \ {state})(dict, statedict')))
```

запись: $Statename_0$-set → *Dict Statedict* → *Statedict*

Цель Удалить непрерывный сигнал и условие разрешения из *Statedict* посредством замены соответствующего дескриптора *StateD* дескриптором *ContenablestateD* и посредством добавления новых дескрипторов *StateD* для неявных состояний

Параметры

stateset Имена состояний, которые должны иметь непрерывные сигналы и условия разрешения удаленными. Первоначально это множество включает все состояния (функция является рекурсивной)

Результат Модифицированный *Statedict*

Алгоритм

Строка 1 Когда просмотр закончен, тогда возвратить модифицированный *Statedict*.

Строки 4-5 Пусть *speclist* обозначает *Speclist* одного из оставшихся состояний.

Строка 6 Удалить непрерывные сигналы и условия разрешения в этом состоянии.

Строка 7 Удалить непрерывные сигналы и условия разрешения в оставшихся состояниях

remove-cont-enable-from-state (*speclist, stnm*)(*dict, statedict*) ≜ (3.7.5.2)

```
 1  (let enable ≐ (∃(, sp) ∈ elems speclist)(is-Inputspec0(sp) ∧ s-Enabling0(sp) ≠ nil) in
 2   let cont = (∃(, sp) ∈ elems speclist)(is-Contspec0(sp)) in
 3   if ¬ enable ∧ ¬ cont then
 4    statedict
 5   else
 6    (let (emptyqid, formu1nm, formu2nm) = dict (GLOBALNAMES) in
 7     let formu1 = mk-Id0 (⟨⟩, formu1nm),
 8        formu2 = mk-Id0(⟨⟩, formu2nm) in
 9     let ptrans = pretrans(emptyqid, formu1) in
10     let speclist ' =
11        if cont
12          then expand-continuous (stnm, speclist, emptyqid, formu1, formu2)(dict)
13          else speclist in
14     if ¬ enable then
15       (let newstnm = create-unique-name() in
16        let preterm = mk-Transition0(ptrans, mk-Nextstate0(newstnm)) in
17        statedict + [stnm      ↦ mk-ContenablestateD(preterm),
18                     newstnm   ↦ mk-StateD(speclist', nil)])
19       else
20        (let emptyqinput =
21            if cont then
22              ⟨⟩
23            else
24              (let term = mk-Termstmt0(nil, mk-Nextstate0(stnm)) in
25               ⟨emptyqtrans (term, emptyqid, formu1, formu2)(dict)⟩) in
26        let (mk-Transition0(decision, ), statedict ') =
27            expand-enable(speclist' ⌢ emptyqinput)(statedict) in
28        let preterm = mk-Transition0(ptrans ⌢ ⟨decision⟩, nil) in
29        statedict' + [stnm ↦ mk-ContenablestateD(preterm)])))
```

запись: *Speclist Name*$_0$ → *Dict Statedict* → *Statedict*

Цель Удалить непрерывные сигналы и условия разрешения из состояния посредством модификации *Statedict*

Параметры

speclist *Speclist*, содержащий переходы состояния ввода

stnm Имя состояния

Результат Модифицированный *Statedict*

Алгоритм

Строки 8-10	Если в непрерывном сигнале приоритет пропущен, тогда другие непрерывные сигналы не могут быть определены.
Строки 1-4	Если состояние не содержит ни условий разрешения, ни непрерывных сигналов, тогда *Statedict* оставить неизмененным.
Строка 6	Извлечь AS_0 идентификатор сигнала пустой очереди и AS_0 имена двух переменных, используемых в связи с сигналами пустой очереди.
Строки 7-8	Построить AS_0 идентификаторы двух переменных.
Строка 9	Построить AS_0 список действия, который изменяет "уникальную" переменную (*formu*1) и который выводит сигнал пустой очереди.
Строки 10-13	Раскрыть непрерывные сигналы посредством модифицирования *Speclist* (*speclist*).
Строка 14	Если состояние не содержит условий разрешения, тогда
Строка 15	Построить новое имя для состояния (которое содержит непрерывные сигналы).
Строка 16	Построить переход, который должен быть интерпретирован в каждом следующем состоянии, содержащем данное состояние.
Строка 17	Изменить *Statedict* двумя дескрипторами: Первый используется в преобразовании вершин следующего состояния. Когда вершина следующего состояния содержит имя *stnm*, тогда вершина следующего состояния заменяется на *preterm*. Второй дескриптор используется, когда состояние, содержащее непрерывные сигналы (*stnm*), преобразуется в AS_1.
Строки 20-29	Раскрыть условия разрешения.
Строки 20-25	Построить ввод пустой очереди, пока он еще не включен в *speclist'*, т.е., если состояние не содержит непрерывных сигналов. Когда состояние не содержит непрерывных сигналов, переход, следующий за пустой очередью, является значительно более простым, поскольку в этом случае отсутствуют вершины принятия решения (подразумевается только обратный переход из следующего состояния в текущее).
Строка 26	Раскрыть условие разрешения. Результатом этого раскрытия является действие сложного решения (*decision*), проверяющее на выражениях в условии разрешения, и *Statedict*, который расширяется для включения дескрипторов *StateD* всех вовлеченных неявных состояний.
Строка 28	Построить переход, который должен быть интерпретирован в соответствующих вершинах следующего состояния (как в строках 16-17), и
Строка 29	Заменить дескриптор *StateD* состояния посредством *ContenablestateD* в *StateDict* (как в строках 16-17)

expand-continuous (*statenm, speclist, emptyqid, formu*1, *formu*2)(*dict*) ≜ (3.7.5.:

```
1  if (∃(,stsp) ∈ elems speclist)(is-Contspec0(stsp) ∧ s-Priority0(stsp) = nil) ∧
2    card{(,stsp) ∈ elems speclist | is-Contspec0(stsp)} > 1 then
3     exit ("§4.11: Только один непрерывный сигнал может быть специфицирован, если пропущен приоритет")
4    else
5     (let priomap = collect-priority-info (speclist, len speclist)(dict) in
6       let conttrans = build-cont-trans (speclist, statenm, priomap ) in
7       let contterm = mk-Transition0(conttrans, nil) in
8       let normtrans = ⟨speclist [i] | i ∈ ind speclist ∧ ¬ is-Contspec0(speclist [i])⟩ in
9       let emptyqinput = emptyqtrans(contterm, emptyqid, formu1, formu2)(dict) in
10      normtrans ⌢ ⟨emptyqinput⟩)
```

запись: *Statename*$_0$ *Speclist Sigid*$_0$ *Varid*$_0$ *Varid*$_0$ → *Dict* → *Speclist*

Цель Модифицировать переход ввода для состояния, содержащего непрерывные сигналы.

Параметры

statenm	Имя состояния, содержащего непрерывные сигналы
speclist	Список *Specs*, которые содержат переходы ввода
emptyqid	AS_0 идентификатор сигнала пустой очереди
formu1,formu2	AS_0 идентификаторы переменных, используемых в модели непрерывных сигналов

Результат Модифицированный список *Specs*

Алгоритм

Строка 1	Здесь
Строка 5	Построить отображение, которое содержит отношение между приоритетом сигнала и его позицией (индексом) в *speclist*.
Строки 6-7	Построить действие сложного решения, которое проверяет на различных выражениях в порядке, заданном приоритетами.
Строка 8	Удалить переходы ввода, которые содержат непрерывные сигналы, из *speclist*.
Строка 9	Построить переход ввода пустой очереди, который проверяет имеется ли правильный сигнал пустой очереди (т.е., проверка равны ли *formu1* и *formu2*) и который использует сложное решение (*contterm*) в ветвлении "true".
Строка 10	Добавить переход ввода пустой очереди к переходам ввода, не содержащим непрерывных сигналов

pretrans(emptyqid, formu1id) ≜ (3.7.5.4)

```
1  (let lit1 = mk-Name0("1", nil),
2       self = mk-Selfexpr0() in
3  let infix = mk-Infixexpr0(formu1id, PLUS, lit1) in
4  let update = mk-Assignstmt0(formu1id, infix) in
5  let task = mk-Task0(⟨update⟩),
6     outtoself = mk-Output0(⟨mk-Outputsig0(emptyqid, ⟨formu1id⟩)⟩, self, ◊ ) in
7  ⟨mk-Actstmt0(nil, task), mk-Actstmt0(nil, outtoself)⟩)
```

запись: $Sigid_0$ $Varid_0$ → $Actstmt_0$+

Цель Построить предложения действия:

```
TASK     formulid := formulid + 1;
OUTPUT emptyqid(formulid) TO SELF;
```

Алгоритм

Строка 1	Построить литерал целого числа "1".
Строка 2	Построить PiD выражения self.
Строка 3	Построить formulid + 1.
Строка 4	Построить formulid := formulid + 1.
Строка 5	Построить TASK formulid := formulid + 1.
Строка 6	Построить OUTPUT emptyqid(formulid) TO SELF.
Строка 7	Возвратить оба действия как предложения действия (без меток)

emptyqtrans(*conttrans, emptyqid, formu1id, formu2id*)(*dict*) ≜ (3.7.5.5)

```
1  (let emptyqinput = mk-Inputvars0(emptyqid, ⟨formu2id⟩) in
2   let ans1 = mk-Answer0(⟨mk-Condition0(nil, mk-Name0("TRUE", nil))⟩, conttrans),
3       ans2 = mk-Answer0(⟨mk-Condition0(nil, mk-Name0("FALSE", nil))⟩,
4                         mk-Termstmt0(nil, mk-Nextstate0(nil))) in
5   let decision = mk-Decision0(mk-Infixexpr0(formu1id, EQ, formu2id),
6                               ⟨ans1, ans2⟩, nil) in
7   let emptytrans = mk-Transition0(⟨mk-Actstmt0(nil, decision)⟩, nil) in
8   (dict(SCOPEUNIT), mk-Inputspec0(⟨emptyqinput⟩, nil, emptytrans)))
```

запись: *Transition*$_0$ *Sigid*$_0$ *Varid*$_0$ *Varid*$_0$ → *Dict* → *Spec*

Цель Построить переход ввода

```
INPUT emptyqid(formu2id);
 DECISION (formu1id = formu2id)
  (true)     : contrans;
  (false)    : NEXTSTATE(-);
 ENDDECISION;
```

и возвратить *Spec*, который содержит переход ввода

Параметры

conttrans	Переход который соответствует ветвлению "true". Если состояние не содержит непрерывные сигналы (но содержит условия разрешения), тогда оно содержит следующее состояние исходного состояния, в противном случае оно содержит сложное решение, созданное в *build-cont-trans*. Следующее состояние в ветвлении "false" подразумевает продолжение к неявному состоянию (не исходного состояния) с которым ассоциируется переход ввода.
emptyqid	AS$_0$ идентификатор сигнала пустой очереди
formu1id, formu2id	AS$_0$ идентификаторы переменных, используемых в непрерывном сигнале и условии разрешения

build-cont-trans(*speclist, stnm, priomap*) ≜ (3.7.5.6)

```
1  (let i ∈ dom priomap be s.t. ¬(∃n ∈ dom priomap)(n > i) in
2   let trans = if priomap \ { i } = []
3               then mk-Transition0(⟨⟩, mk- Termstmt0(nil, mk-Nextstate0(stnm)))
4               else build-cont-trans (speclist, stnm, priomap \ { i }) in
5   let index = priomap ( i ) in
6   let (scope, mk-Contspec0(expr,, exittrans)) = speclist [index] in
7   let yesval = mk-Condition0(nil, mk-Id0(⟨⟩, mk-Name0("TRUE", nil))),
8       noval = mk-Condition0(nil, mk-Id0(⟨⟩, mk-Name0("FALSE", nil))) in
9   let yestrans = mk-Answer0(⟨yesval⟩, exittrans),
10      notrans = mk-Answer0(⟨ noval ⟩, trans) in
11  let decision = mk-Decision0(mk-Scopeexpr0(scope, expr), ⟨ yestrans, notrans ⟩, nil) in
12  mk- Transition0(⟨ mk-Actstmt0(nil, decision)⟩, nil))
```

запись: *Speclist Statename*$_0$ (N_1 ⇸ N_1) → *Transition*$_0$

Цель Построить сложное решение, которое осуществляет тестирование и ветвление в непрерывных сигналах в порядке, заданном приоритетами

Параметры

speclist	*Specs*, которые содержат непрерывные сигналы
stnm	Имя (явного) состояния, к которому назначаются непрерывные сигналы

priomap — Отношение между приоритетом непрерывных сигналов и их позициями (индексом) в *speclist*

Результат — Сложное решение предложения действия

Алгоритм

Строка 1 — Пусть *i* обозначает вход в отображении приоритета, который имеет наименьшее значение (т.е., наивысший приоритет среди оставшихся элементов в *priomap*).

Строки 2-4 — Если значение имеет наименьший приоритет (т.е., если оно единственное, оставшиеся в отображении), тогда пусть *trans* обозначает оператор терминатора, содержащий *nextstate* для старого явного состояния (строка 3), иначе пусть *trans* обозначает действие принятия решения для оставшихся приоритетов (строка 4).

Строки 5-6 — Пусть *scope* обозначает *Qual* единицы подразделения, которая определяет состояние (напомним, что *Statedict* сервисов были объединены до того, как данное преобразование имело место, и что вычисление выражений зависит от действительного сервиса), пусть *expr* обозначает выражение в непрерывном сигнале, и пусть *exittrans* обозначает переход в непрерывном сигнале

Строки 7-12 — построить переход, содержащий решение:

```
DECISION (expr):
 (true)    : exittrans;
 (false)   : trans;
ENDDECISION
```

где определено, что *expr* должно быть преобразовано в AS_1 в контексте единицы подразделения, обозначенного scope, который является процедурой, процессом или сервисом

collect-priority-info (*speclist, index*)(*dict*) ≜ (3.7.5.7)

```
 1  (if index = 0 then
 2     []
 3   else
 4    (let emap = collect-priority-info (speclist, index - 1)(dict) in
 5     let (scope, inp) = speclist [index] in
 6     let dict' = [q ↦ dict (q) | q ∈ dom dict ∧ ¬ is-SynD(dict (q))] + [SCOPEUNIT ↦ scope] in
 7      cases inp :
 8       (mk-Contspec0(, prio, )
 9         → (let prio' = if prio = nil then 1 else eval-simple-expr (prio, "INTEGER")(dict') in
10             if prio' ∈ dom emap then
11               exit("§4.11: Несколько непрерывных сигналов с одним приоритетом")
12             else
13               emap + [prio' ↦ index]),
14       T → emap)))
```

запись: *Speclist* N_0 → *Dict* → (N_1 ⇸ N_1)

Цель — Построить отображение, содержащее отношения между приоритетом непрерывного сигнала и значением индекса *Spec* в *Speclist*, содержащем непрерывный сигнал

Параметр

speclist — *Speclist*, который содержит непрерывные сигналы

index — Текущее значение индекса в *speclist*. Первоначально значение индекса равно длине списка. Рекурсия прекращается, когда значение индекса становится равным нулю

Результат — Построенное отображение приоритета

Алгоритм

Строка 1	Когда просмотр закончен, возвратить пустое отображение.
Строка 4	Собрать приоритеты оставшихся непрерывных сигналов.
Строка 5	Извлечь *Quals* единицы подразделения, которая определила непрерывный сигнал и переход ввода.
Строка 6	Отменить *Dict* посредством *Qual* единицы подразделения так, чтобы выражение приоритета могло быть оценено в правильном контексте. Исключить также синонимы из *Dict*, так как недопустимо определять синоним в конструкции приоритета.
Строки 8-13	Если переход ввода является непрерывным сигналом, тогда определить приоритет непрерывного сигнала. Если приоритет пропущен, тогда установить приоритет равным "1", в противном случае оценить простое выражение.
Строки 10-11	Не должно существовать другого непрерывного сигнала с таким же значением.
Строка 13	Добавить приоритет в отображение, содержащее оставшиеся приоритеты.
Строка 14	Если переход ввода не является непрерывным сигналом, тогда возвратить отображение, содержащее оставшиеся непрерывные сигналы

expand-enable (*speclist*)(*statedict*) ≜ (3.5.7.8)

1 (**let** *enabset* = {*speclist* [*i*] | *i* ∈ *indspeclist* ∧ **is**-*Inputspec*$_0$(*speclist* [*i*]) ∧
2 **s**-*Enabling*$_0$(**s**-*Statespec*$_0$(*speclist* [*i*])) ≠ **nil**} **in**
3 **let** *normtrans* = ⟨*speclist* [*i*] | 1 ≤ *i* ≤ **len** *speclist* ∧ *speclist* [*i*] ∉ *enabset*⟩ **in**
4 *build-enable-decision* (*normtrans*, *enabset*)(*statedict*))

запись: *Speclist* → *Statedict* → *Transition*$_0$ *Statedict*

Цель Построить вложенные действия решения, которые моделируют состояние условия разрешения и изменяют *Statedict* посредством подразумеваемых неявных состояний

Параметры

speclist	*Speclist*, содержащий переход ввода состояния

Результат Переход, содержащий порожденное действие решения и измененный *Statedict*

Алгоритм

Строки 1-2	Пусть *enabset* означает множество *Specs*, которое содержит условие разрешения, и
Строка 3	Пусть *normtrans* означает множество *Specs*, которое не содержит условие разрешения.
Строка 4	Построить действие решения и неявные состояния

build-enable-decision (*normtrans, enabset*)(*statedict*) ≜ (3.7.5.9)

```
1   (let falsevalue = mk-Condition0(nil, mk-Id0(⟨⟩, mk-Name0("FALSE", nil))) in
2    let truevalue = mk-Condition0(nil, mk-Id0(⟨⟩, mk-Name0("TRUE", nil))) in
3    let enab ∈ enabset in
4    let (scope, mk-Inputspec0(vl, expr, tr)) = enab in
5    let sigl = ⟨ sigid | (∃varl ∈ elems vl)(s-Sigid0(varl) = sigid ) ⟩ in
6    let newtrans = normtrans ⁀ ⟨(scope, mk-Savespec0(sigl))⟩ in
7    let newtrans' = normtrans ⁀ ⟨(scope, mk-Inputspec0(vl, nil, tr))⟩ in
8    let (trans0, statedict' ) =
9        if enabset \ {enab} = {}
10         then build-implicit-state (newtrans)(statedict)
11         else build-enable-decision(newtrans, enabset \ {enab})(statedict) in
12   let (trans1, statedict'' ) =
13       if enabset \ {enab} = {}
14         then build-implicit-state (newtrans' )(statedict' )
15         else build-enable-decision (newtrans', enabset \ {enab})(statedict' ) in
16   let ans0 = mk-Answer0(falsevalue, trans0) in
17   let ans1 = mk-Answer0(truevalue, trans1) in
18   let decexpr = mk-Scopeexpr0(scope, expr) in
19   let decision = mk-Decision0(decexpr, ⟨ans0, ans1⟩, nil) in
20   let actstmt = mk-Actstmt0(nil, decision) in
21   (mk-Transition0(⟨ actstmt ⟩, nil), statedict'' ))
```

запись: *Speclist Spec*-set → *Statedict* → *Transition*0 *Statedict*

Цель См. *expand-enable*

Параметры

normtrans	*Speclist*, который содержит переходы ввода, которые не содержат условия разрешения.
enabset	Множество *Specs*, содержащее переходы ввода, которые содержат условие разрешения.

Результат См. *expand-enable*

Алгоритм

Строка 1	Построить метку выбора "false"
Строка 2	Построить метку выбора "true"
Строка 3	Извлечь *Specs* из множества *Spec*, представляющего переходы ввода с условием разрешения
Строка 6	Произвести декомпозицию *Spec*
Строка 5	Извлечь идентификаторы сигналов из вершины ввода
Строка 6	Сформировать новый *Speclist*, содержащий старый *Speclist* и *Savespec*0.
Строка 7	Сформировать новый *Speclist*, содержащий старый *Speclist* и *Inputspec*0.
Строки 8-11	Построить дерево решений для случая "false".
Строка 10	Если *enab* является последним условием разрешения, построить решение ведущее к *Nextstates*.
Строка 11	Если *enab* не является последним условием разрешения, построить решение ведущее к решениям.
Строки 12-15	Построить дерево решений для случая "true".
Строка 16	Построить ответ "false".
Строка 17	Построить ответ "true".
Строка 18	Построить выражение принятия решения.
Строка 19	Построить решение.
Строка 20	Построить предложение действия.
Строка 21	Возвратить новый переход и измененный *Statedict*.

build-implicit-state (*alltrans*)(*statedict*) ≜ (3.7.5.10)

```
1  (let stnm = create-unique-name () in
2   let trans = mk-Transition0(⟨⟩, mk- Termstmt0(nil, mk-Nextstate0(stnm))) in
3   (trans, statedict + [stnm ↦ mk-StateD (alltrans, nil)]))
```

запись: *Speclist* → *Statedict* → *Transition*$_0$ *Statedict*

Цель Создать новое неявное состояние с полным *Speclist* и изменить *Statedict*

Параметры

alltrans Полный *Speclist*.

Результат См. *expand-enable*

Алгоритм

Строка 1 Создать имя AS_0 для нового неявного состояния.

Строка 2 Создать переход, содержащий следующее состояние, который является переходом к неявному состоянию.

Строка 3 Возвратить переход и измененный *Statedict*

3.7.6 Преобразование состояний и вершин ввода

transform-statelist(*transformedstates*)(*dict*) ≜ (3.7.6.1)

```
1  (let statedict = dict(STATEDICT) in
2   if dom statedict = transformedstates then
3     ({}, dict)
4   else
5     (let stnm ∈ (dom statedict \ transformedstates) in
6      if is-ContenablestateD (statedict (stnm)) then
7         transform-statelist (transformedstates ∪ {stnm})(dict)
8        else
9         (let mk-StateD (speclist, importinf ) = statedict (stnm) in
10         let (currentstate, impact ) = if importinf = nil then (stnm, ) else importinf in
11         let (saveset1, inputset1, dict') = transform-statespecl(currentstate, speclist )(dict) in
12         let inputset1' =
13             if importinf = nil ∨ impact = nil
14               then inputset1
15               else insert-importact (impact, inputset1) in
16         let statenode = mk-State-node1(name-to-name1(stnm), saveset1, inputset1') in
17         let (bodyrest, dict") = transform-statelist (transformedstates ∪ {stnm})(dict') in
18         ({statenode} ∪ bodyrest, dict")))
```

запись: *Statename*$_0$-**set** → *Dict* → *State-node*$_1$-**set** *Dict*

Цель Преобразовать состояния тела процесса или процедуры в AS_1. Состояния преобразуются с учетом каждого состояния в *Statedict*. Неявные состояния, подразумеваемые в сервисах, непрерывных сигналах и условиях разрешения, являются уже включенными в AS_0 состояния, которые должны быть преобразованы. Во время преобразования новые состояния, подразумеваемые выражениями импорта, могут быть добавлены в *Statedict*

Параметры

transformedstates AS_0 имена состояний, которые уже преобразованы. Когда *transform-statelist* применяется первый раз, множество является пустым.

Результат AS_1 состояния и *Dict*, измененный информацией о неявных переменных (вход IMPLIED), подразумеваемыми выражениями импорта, используемыми в процессе или процедуре, и измененными для включения выходных сигналов (вход *OUTSIGNALS*), используемых в процессе или процедуре.

Алгоритм

Строка 1	Извлечь *Statedict* из *Dict*
Строка 2	Если все преобразованные состояния являются состояниями в *Statedict*, тогда выход.
Строка 5	Пусть *stnm* обозначает состояние, которое еще не было преобразовано.
Строки 6-7	Если состояние содержит непрерывные сигналы или условия разрешения, тогда состояние заменяется некоторым числом неявных состояний, и, следовательно, не преобразовывается.
Строка 9	Произвести декомпозицию дескриптора *Statedict* состояния. Если *importInf* отлично от nil, тогда состояние является состоянием импорта (см. определение *Importstateinf*).
Строка 10	Пусть *currentstate* обозначает имя состояния, которое должно быть вставлено в следующее состояние с черточкой переходов ввода. Если обрабатываемое состояние является неявным, подразумеваемым в выражении импорта, тогда имя состояния находится в дескрипторе *StateD*.
Строка 11	Преобразовать вводы и сохранение состояния.
Строки 12-14	Если состояние является неявным состоянием подразумеваемым в импорте, тогда AS_1 действие, которое использует неявную переменную импорта, должно быть вставлено в преобразованное состояние. Это действие не вставляется во время преобразования импортируемого выражения, потому что во время преобразования выражений осуществляется построение состояний импорта.
Строка 16	Построить вершину состояния AS_1.
Строка 17	Преобразовать оставшиеся состояния.
Строка 18	Объединить построенное состояние с оставшимися состояниями, а также вернуть измененный *Dict*

transform-statespecl (*stnm*, *statespeclist*)(*dict*) ≜ (3.7.6.2)

```
1  (let (saveset, statespeclist') =
2     if (∃i ∈ ind statespeclist )(is-Savespec0(s-Statespec0(statespeclist [i]))) then
3       (let i be s.t. (is-Savespec0(s-Statespec0(statespeclist [i]))) in
4        let mk-Savespec0(sigl) = s-Statespec0(statespeclist [i]) in
5        let sigqualset = transform-signallist (sigl)(dict) in
6       (sigqualset, ⟨ statespeclist [n] | 1 ≤ n ≤ len statespeclist ∧ n ≠ i ⟩))
7     else
8       ({}, statespeclist) in
9  let dict' = dict + [SCOPEUNIT ↦ process-level(dict(SCOPEUNIT))] in
10 let varset = {qual ∈ dom dict | is-VarD(dict (qual)) ∧ process-level (qual) = dict' (SCOPEUNIT)} in
11 let xtQUERYspec0 = as0-xtQUERY-inputs (varset, stnm)(dict) in
12 let (as1nodeset, dict') = transform-input (stnm, statespeclist' ⇀ xtQUERYspec0)(dict) in
13 let saveset1 = mk-Save-signalset1(make-as1idset(saveset)(dict)) in
14 (saveset1, as1nodeset, dict'))
```

запись: *Statename*$_0$ *Speclist* → *Dict* → *Save-signalset*$_1$ *Input-node*$_1$-**set** *Dict*

Цель Преобразовать состояние в AS_1

Параметры

stnm	Имя состояния. Оно вставляется в следующее состояние с черточкой состояния.
statespeclist	*Speclist*, содержащий переходы ввода состояния

Результат AS_1 вершина сохранения и AS_1 вершины ввода

Алгоритм

Строки 1-8	Пусть *saveset* обозначает множество *Signalqual*, содержащее сигналы, которые сохраняются в состоянии, и пусть *statespeclist'* обозначает модифицированный *Speclist*, где сохранение уже удалено (если такое имеется).
Строка 2	Если существует сохранение в *Speclist*, тогда
Строки 3-4	Извлечь сохранение из *Speclist*.
Строка 5	Преобразовать сигналы в сохранении.
Строка 6	Удалить сохранение из *Speclist*
Строки 9-10	Изменить SCOPEUNIT для обозначения окружающего процесса. *dict'* используется только для извлечения локальных переменных процесса (строка 10).
Строка 11	Построить AS_0 переходы ввода для сигналов xtQUERY.
Строка 12	Преобразовать переходы ввода, где вводы xtQUERY уже добавлены.
Строка 13	Построить AS_1 вершину сохранения из множества *Signalquals*.
Строка 14	Возвратить вершину сохранения и вершины ввода

as$_0$*-xtQUERY-inputs* (*exportset, stnm*)(*dict*) ≜ (3.7.6.3)

```
 1  (if exportset = {} then
 2     ◊
 3   else
 4    (let qual ∈ exportset,
 5        term = mk-Nextstate0(stnm),
 6        mk- VarD (tq,, export,, ) = dict (qual) in
 7    if export = nil then
 8      as0-xtQUERY-inputs (exportset \ {qual}, stnm)(dict)
 9     else
10      (let (,nm) = qual [len qual],
11          mk-SignalnamesD (imp, xq, xr) = exportmap((tq, nm)) in
12       let sigelem = ⟨ mk-Outputsig0(mk-Id0(◊, xr), ⟨ mk-Id0(◊, imp)⟩)⟩ in
13       let xroutput = mk-Output0(sigelem, mk-Senderexpr0(), ◊) in
14       let xrtrans = mk-Transition0(⟨ mk-Actstmt0(nil, xroutput)⟩, term) in
15       let xqinput = mk-Inputspec0(⟨ mk-Inputvars0(mk-Id0(◊, xq), ◊ )⟩, nil, xrtrans) in
16  ⟨(dict (SCOPEUNIT), xqinput )⟩ ⇝ as0-xtQUERY-inputs (exportset \ {qual }, stnm)(dict))))
```

запись: *Qual*-set *Name*$_0$ → *Dict* → *Speclist*

Цель Построить AS_0 вершины ввода для всех сигналов процесса, подразумеваемых в экспорте, как определено в §4.13 в Z.100

Параметры

exportset	Множество *Quals* переменных, определенных в экспортирующем процессе
stnm	Имя состояния, к которому должны быть привязаны неявные вводы.

Результат См. определение *Speclist*

Алгоритм

Строка 1	Когда просмотр закончен, ничего не возвращать (функция является рекурсивной).
Строка 4	Пусть *qual* обозначает *Qual* переменной в множестве.
Строка 5	Построить AS_0 вершину следующего состояния, которая следует за неявными вводами.
Строка 6	Произвести декомпозицию дескриптора переменной.
Строка 7	Если он не является экспортируемой переменной, тогда продолжить рассмотрение следующей переменной.
Строка 10	Извлечь имя экспортируемой переменной.

Строка 11	Произвести декомпозицию дескриптора *SignalnamesD* (см. определение *Exportmap*), общего для всех переменных экспорта в системе, имеющих это имя (*nm*) и сорт (*tq*). *imp* обозначает неявную переменную, связанную с переменной экспорта, *xq* обозначает имя сигнала xtQUERY, а *xr* является именем сигнала xtREPLY.
Строка 13	Создать вершину вывода AS_0, которая посылает сигнал xtREPLY.
Строка 14	Создать переход, непосредственно следующий за вершиной ввода.
Строка 15	Создать вершину ввода, который содержит сигнал xtQUERY.
Строка 16	Возвратить *Spec*, содержащий полный переход ввода (*xqinput*), объединенный со списком *Speclist* для оставшихся переменных экспорта

transform-input (*stnm, statespecl*)(*dict*) ≜ (3.7.6.4)

```
 1  (if statespecl = ⟨⟩ then
 2    ({}, dict )
 3   else
 4    (let (scope, node ) = hd statespecl in
 5     let labeldict' =
 6        if is-ServiceD(dict(scope))
 7          then s-Labeldict(dict(scope))
 8          else dict(LABELDICT) in
 9    let dict ' = dict + [SCOPEUNIT ↦ scope,
10                         LABELDICT ↦ labeldict' ] in
11     cases node :
12      (mk-PrInput0(inputvars, trans),
13       mk-Inputspec0(inputvars,, trans)
14        → (let (as1set', dict" ) = transform-input-transition (stnm, inputvars, trans )(dict' ) in
15           let (as1rest, dict''' ) = transform-input (stnm, tl statespecl )(dict" ) in
16          (as1set' ∪ as1rest, dict''' )))))
```

запись: *Statename*$_0$ *Speclist* → *Dict* → *Input-node*$_1$-set *Dict*

Цель Преобразовать переходы ввода состояния и изменить *dict* при помощи информации о неявных переменных и при помощи выходных сигналов.

Параметры

stnm	Имя состояния
statespecl	*Speclist* для состояния.

Результат Множество AS_1 вершин ввода для состояния.

Алгоритм

Строка 1	Кода просмотр закончен, возвратить пустое множество.
Строка 4	Произвести декомпозицию следующего *Spec*, который должен быть рассмотрен.
Строка 5	Извлечь *Labeldict* для единицы подразделения. Если переход ввода исходит из сервиса (до того, как они были объединены), тогда извлечь его из дескриптора сервиса.
Строка 9	Изменить SCOPEUNIT в *Dict* для обозначения единицы подразделения, из которой берет начало переход ввода, и изменить LABELDICT для обозначения *Labeldict* единицы подразделения.
Строки 12-13	Произвести декомпозицию настоящего перехода ввода.
Строка 14	Создать множество AS_1 вершин ввода для всех сигналов в AS_0 вершине ввода.
Строка 15	Преобразовать оставшиеся переходы ввода
Строка 16	Добавить вновь созданные вершины ввода к оставшимся.

transform-input-transition (*stnm, inpvars, trans*)(*dict*) ≜ (3.7.6.5)

```
1  (if inpvars = ◊ then
2    ({}, dict)
3   else
4    (let (vl1, sigq) = transform-inputvars (hd inpvars)(dict) in
5     let as1sigid = make-as1-identifier (sigq)(dict) in
6     let (trans1, dict') = transform-transition(stnm, trans, ◊)(dict) in
7     let inputnode = mk-Input-node1(as1sigid, vl1, trans1) in
8     let (inprest, dict") = transform-input-transition (stnm, tl inpvars, trans )(dict') in
9     ({inputnode } ∪ inprest, dict")))
```

запись: *Statename*$_0$ *Inputvars*$_0^+$ *Transition*$_0$ → *Dict* → *Input-node*$_1$-**set** *Dict*

Цель Преобразовать переход ввода в AS_1

Параметры

stnm	Имя состояния
inpvars	Список пар сигнал - параметры для ввода
trans	Переход, который следует за вводом

Результат Множество AS_1 вершин ввода. Существует столько вершин ввода, сколько существует пар сигнал - параметры ввода в вершинах ввода AS_0.

Алгоритм

Строка 1	Когда просмотрены все пары сигнал-параметры, тогда возвратить пустое множество.
Строка 4	Построить AS_1 список параметров (vl_1) и *Qual* сигнала из первой пары AS_0 сигнал-параметры.
Строка 5	Построить AS_1 идентификатор сигнала.
Строка 6	Преобразовать переход.
Строка 7	Построить AS_1 вершину ввода для сигнала.
Строка 8	Построить AS_1 вершины ввода для для оставшихся пар сигнала.
Строка 9	Возвратить AS_1 вершины ввода, объединенные с оставшимися AS_1 вершинами ввода, и параметры

transform-inputvars (**mk**-*Inputvars*$_0$(*sigid, varl*))(*dict*) ≜ (3.7.6.6)

```
1  (let squal = signal-qual(sigid)(dict) in
2   let tquallist = cases dict (squal ):
3                   (mk-SignalD (tquallist,, ) → tquallist,
4                    mk-TimerD (tquallist, ) → tquallist ) in
5   if (len varl = len tquallist ) then
6     (let vl = transform-inputvarlist (varl, tquallist )(dict ) in
7      (vl, squal ))
8    else
9     exit("§2.6.4: Неправильное число параметров в вершине ввода"))
```

запись: *Inputvars*$_0$ → *Dict* → [*Variable-identifier*$_1$]* *Signalqual*

Цель Построить AS_1 список переменных для вершины ввода и извлечь *Qual* сигнала ввода

Параметры

sigid	AS_0 идентификатор сигнала
varl	AS_0 список переменных

Алгоритм

Строка 1	Пусть *squal* обозначает *Qual* сигнала.
Строки 2-4	Пусть *tquallist* обозначает список сортов для параметров сигнала.
Строка 5	Длина списка сортов должна быть равна длине списка переменных.
Строки 6-7	Преобразовать список переменных в AS_1 и возвратить его вместе с *Qual* сигнала

transform-inputvarlist(*varl, sortlist*)(*dict*) ≜ (3.7.6.7)

```
1  (if varl = ⟨⟩ then
2      ⟨⟩
3     else
4      (let restvl = transform-inputvarlist (tl varl, tl sortlist)(dict) in
5       if hd varl = nil then
6         ⟨ nil ⟩ ⌢ restvl
7       else
8        (let vqual = get-visible-variable (hd varl, {get-parent (hd sortlist)(dict)}, false)(dict) in
9         let mk-VarD (,,,, nqual ) = dict (vqual) in
10        let as1id = make-as1-identifier(nqual )(dict) in
11        ⟨ as1id ⟩ ⌢ restvl)))
```

запись: [$Varld_0$]* *Sortqual** → *Dict* → [*Variable-identifier*$_1$]*

Цель Построить AS_1 список переменных, которые должны быть привязаны к вершине ввода

Параметры

varl	AS_0 список переменных
sortlist	Список сортов для сигнала

Алгоритм

Строка 1	Когда просмотр завершен, возвратить пустой список.
Строка 4	Преобразовать окончание списка.
Строка 5	Если первая переменная пропущена, тогда возвратить объект nil.
Строка 8	Извлечь *Qual* первого идентификатора посредством просмотра контекста
Строка 10	Построить AS_1 идентификатор переменной
Строка 11	Возвратить AS_1 идентификатор переменной, объединенный с идентификаторами переменных для остатка списка

insert-importact (*impact, signalset*$_1$) ≜ (3.7.6.8)

```
1  (let mk-Input-node1(replyid, val, trans ) ∈ signalset1 in
2   let mk-Transition1(nodes, term) = trans in
3   if is-Decision-node1(impact) then
4     {mk-Input-node1(replyid, val, mk-Transition1(nodes, impact))}
5    else
6     {mk-Input-node1(replyid, val, mk-Transition1(nodes ⌢ ⟨ impact ⟩, term ))})
```

запись: (*Graph-node*$_1$ | *Decision-node*$_1$) *Input-node*$_1$-**set** → *Input-node*$_1$-**set**

Цель Вставить AS_1 действие в AS_1 вершину ввода, содержащую сигнал xtREPLY. Это действие построено в связи с преобразованием выражения, которое содержит выражения импорта. Так как выражения импорта раскрываются в состояния AS_0, действие не может быть вставлено, пока состояние еще не преобразовано

Параметры

impact	AS_1 действие, которое должно быть вставлено
signalset$_1$	AS_1 множество вершин ввода, которое содержит только xtREPLY вершины ввода

Результат Модифицированное AS_1 множество вершин ввода, содержащее xtREPLY вершины ввода

Алгоритм

Строка 1	Произвести декомпозицию вершины ввода AS_1.
Строка 2	Произвести декомпозицию содержащегося перехода.
Строка 3	Если вершина должна быть вставлена в решение (в случае, когда вопрос содержит выражение импорта), тогда
Строка 4	Заменить терминатор действием решения, иначе
Строка 6	Присоединить действие к списку действий

3.7.7 Преобразование строк перехода

transform-transition (*tnm*, mk-*Transition*$_0$(*actl*, *term*), *nodes*$_1$)(*dict*) ≜ (3.7.7.1)

```
 1  (if actl = ⟨⟩ then
 2     (let mk-Termstmt0(, t ) = term in
 3      cases t :
 4       (mk-Nextstate0(stnm)
 5         → if tnm = nil ∧ stnm = nil
 6              then exit("§4.9: Следующее состояние с черточкой в начальном переходе")
 7              else (let statedict = dict (STATEDICT) in
 8                    let stnm' = if stnm = nil then tnm else stnm in
 9                    let nextstate =
10                      if tnm = nil ∧ is-ServiceD (dict (dict (SCOPEUNIT))) ∧ stnm' ∉ dom statedict then
11                        (let (,servicelist ) = dict (SERVICES) in
12                         extract-initial-state (servicelist, stnm', ⟨⟩)(dict))
13                        else
14                         if is-ServiceD (dict (dict (SCOPEUNIT))) ∧ stnm' ∉ dom statedict
15                           then extract-servicestate(tnm, stnm' )(dict)
16                           else stnm' in
17                    if nextstate ∈ dom statedict then
18                      if is-ContenablestateD (statedict (nextstate)) then
19                        (let trans = s-Transition0(statedict (nextstate)) in
20                         transform-transition(tnm, trans, nodes1)(dict))
21                      else
22                        (mk-Transition1(nodes1, mk-Nextstate-node1(name-to-name1(nextstate))), dict)
23                    else
24                  exit("§2.6.7.2.1: Имя в следующем состоянии должно обозначать состояние")),
25       mk-Stop0()
26         → (mk-Transition1(nodes1, mk-Stop-node1()), dict ),
27       mk-Return0()
28         → (mk-Transition1(nodes1, mk-Return-node1()), dict ),
29       mk-Join0(l )
30         → (let labeldict = dict (LABELDICT) in
31            if l ∈ dom labeldict then
32              (let trans = labeldict (l ) in
33               transform-transition(tnm, trans, nodes1)(dict ))
34             else
35              exit("§2.6.7.2.2: Метка в перескоке не определена"))))
36  else
37   (let (tran1, dict') = transform-act (tnm, actl, term, nodes1)(dict ) in
38    cases tran1:
39     (mk-Transition1(, )
40       → (tran1, dict'),
41     T → transform-transition (tnm, mk-Transition0(tl actl, term), nodes1 ⌢ tran1)(dict'))))
```

запись: [*Statename*$_0$] *Transition*$_0$ *Graph-node*$_1$* → *Dict* → *Transition*$_1$ *Dict*

Цель Преобразовать переход в AS$_1$

Параметры

tnm	Имя состояния, содержащего переход. Если *tnm* равняется nil, тогда переход является начальным переходом
actl	Список действий перехода
term	Терминатор перехода
nodes$_1$	Вершины (из перехода), которые уже преобразованы

Результат AS$_1$ переход

Алгоритм

Строки 1-35	Когда просмотрен список действий, тогда преобразовать терминатор.
Строка 4	Если терминатор является следующим состоянием, тогда он не должен содержать черточки, если она содержится в начальном переходе.
Строка 7	Пусть *statedict* обозначает *Statedict* процесса.
Строка 8	Пусть *stnm'* обозначает имя в следующем состоянии, если оно определено, в противном случае оно обозначает имя состояния, ведущего к следующему состоянию.
Строки 9-16	Пусть *nextstate* обозначает имя следующего состояния.
Строки 9-12	Если оно является начальным переходом, а единица подразделения является сервисом, и оно не является следующим состоянием, ведущим к состоянию импорта, тогда извлечь имя состояния следующего состояния посредством рассмотрения вершин следующего состояния в начальном терминаторе всех сервисов.
Строка 14	Если единица подразделения является сервисом и имя состояния не может быть найдено в *Statedict* (так как явные состояния в *statedict* уже объединены и каждое новое состояние в *statedict* имеет отличное новое имя), тогда извлечь отличное новое состояние посредством рассмотрения имени состояния старого сервиса и имени состояния нового сервиса.
Строка 17	Имя следующего состояния должно быть определено.
Строки 18-20	Если следующее состояние ведет к состоянию, которое уже заменено некоторым числом состояний условия разрешения, тогда *Statedict* содержит только переход и этот переход заменяет вершину следующего состояния.
Строка 22	Построить AS_1 переход, содержащий преобразованные действия ($nodes_1$) и вершину следующего состояния.
Строка 25	Если терминатор является терминатором *stop*, тогда построить AS_1 переход, содержащий вершину *stop*.
Строка 27	Если терминатор является возвратом, тогда построить AS_1 переход, содержащий вершину возврата.
Строка 29	Если терминатор является соединением, тогда пусть *labeldict* обозначает *Labeldict* для единицы подразделения.
Строка 31	Метка должна быть определена.
Строка 32	Извлечь переход, где метка является определенной.
Строка 33	Заменить перескок этим переходом.
Строка 37	Если в переходе имеются еще действия, тогда преобразовать первое действие в списке действий. Заметьте, что хотя только первое действие преобразуется, полный список действий задается как параметр.
Строка 39	Если результат преобразования первого действия является полным переходом, тогда первое действие содержит выражения импорта и переход возвращается.
Строка 41	В противном случае преобразовать оставшиеся действия

extract-initial-state (*servicelist, nextstate, statelist*)(*dict*) ≜ (3.7.7.2)

```
1  (if servicelist = ⟨⟩ then
2    (let (statetuplemap, ) = dict (SERVICES) in
3     if statelist ∈ dom statetuplemap then
4        statetuplemap (statelist)
5       else
6        exit("§2.6.7.2.1: Имя в следующем состоянии должно обозначать определенное состояние"))
7   else
8    if hd servicelist = dict(SCOPEUNIT) then
9      extract-initial-state (tl servicelist, ..., statelist ∈ (nextstate))(dict )
10     else
11      (let mk-ServiceD(trans,,,, ) = dict (hd servicelist ) in
12       let mk-Transition0(, mk- Termstmt0(, mk-Nextstate0(stnm))) = trans in
13       extract-initial-state (tl servicelist, nextstate, statelist ⁀ ⟨ stnm ⟩)(dict)))
```

запись: *Servicetuple Statename*$_0$ *Statename*$_0^*$ → *Dict* → *Statename*$_0$

Цель Извлечь первое (составное) состояние в случае, когда состояния содержат состояния объединенных сервисов и один из сервисов содержит начальный переход

Параметры

servicelist Список *Quals* сервисов, соответствующих кортежу имени состояния (*statelist*)

nextstate Имя состояния следующего сервиса

statelist Кортеж имени состояния

Результат Выведенное (уникальное) имя состояния

Алгоритм

Строка 1 Когда рассмотрены начальные следующие состояния всех сервисов, тогда

Строка 2 Пусть *statetuplemap* обозначает отношение между кортежами имени состояния и ассоциированными уникальными именами.

Строка 3 Построенный кортеж имени состояния для следующего состояния должен быть включен в *statetuplemap*, в противном случае существует одно из следующих состояний, которое содержит неопределенное имя состояния.

Строка 4 Возвратить уникальное имя состояния, соответствующее следующим состояниям сервиса.

Строки 8-9 Если сервис должен быть рассмотрен в текущем сервисе, тогда добавить имя следующего состояния к кортежу имени состояния до рассмотрения оставшихся сервисов. Имя в следующем состоянии текущего сервиса функцией больше не используется (поэтому "...").

Строка 11 Пусть trans обозначает начальный переход сервиса, который должен быть рассмотрен (он имеет пустой список действий).

Строка 12 Пусть *stnm* обозначает имя состояния в его начальной вершине следующего состояния.

Строка 13 Добавить имя состояния в кортеж имени состояния до рассмотрения оставшихся сервисов

extract-servicestate(*oldstate, stnm*)(*dict*) ≜ (3.7.7.3)

```
1  (let (statetuplemap, servicelist) = dict (SERVICES) in
2   let statelist be s.t. statelist ∈ dom statetuplemap ∧ statetuplemap(statelist) = oldstate in
3   let index be s.t. servicelist [index] = dict (SCOPEUNIT) in
4   let newstatelist = ⟨ statelist [i] | 1 ≤ i < index ⟩ ⌢
5      ⟨ stnm ⟩ ⌢
6      ⟨ statelist [i] | index < i ≤ len statelist ⟩ in
7   if newstatelist ∈ dom statetuplemap then
8     statetuplemap (newstatelist)
9    else
10    exit("§2.6.7.2.1: Имя в следующем состоянии должно обозначать определенное состояние"))
```

запись: *Statenameo Statenameo* → *Dict* → *Statenameo*

Цель Из имени старого (но уникального) состояния и имени нового состояния сервиса получить имя уникального состояния, которое определяется в *Statedict*

Алгоритм

Строка 1 Пусть *statetuplemap* обозначает отношение между кортежами имени состояния и уникальными именами состояния, которые определяются в *Statedict*. Пусть *servicelist* обозначает список *Quals* сервисов, которые соответствовали кортежам имен состояний.

Строка 2 Пусть *statelist* обозначает кортеж имени состояния, который соответствует старому имени состояния.

Строка 3 Пусть *index* обозначает индекс для текущего сервиса.

Строки 4-6 Наложить кортеж имени состояния с новым именем состояния на позицию, указываемую индексом.

Строка 7-8 Если существует уникальное имя состояния для этого кортежа имени состояния, тогда возвратить уникальное имя состояния, иначе имя состояния нового сервиса является неопределенным

3.7.8 Преобразование предложений действия

В этом подразделе предложения действия преобразуются в AS_1. Для каждой функции преобразования задаются имя состояния, содержащего предложение действия, предложения действия, следующие за предложением действия, терминатор, следующий за оставшимися предложениями действия, предложения действия, которые уже преобразованы (см. *transform-act*). Выражения импорта в предложении действия заменяется неявной переменной в AS_1 предложении действия. Если предложение действия содержит какие-либо выражения импорта, тогда полный $Transition_1$, содержащий предварительно преобразованные вершины и следующее состояние для неявного состояния, возвращается вместо него, а *Statedict* изменяется для включения дескриптора *StateD*, который содержит информацию о неявном состоянии, предложения действия, следующие за предложением действия, терминатор, следующий за предложениями действия, имя состояния, следующее за предложениями действия, и преобразованную вершину (см. определения *StateD* и, особенно, *Importstateinf*)

transform-act (*tnm, actl, term,* $nodes_1$)(*dict*) ≜ (3.7.8.1)

```
1  (let mk-Actstmt0(, act ) = hd actl in
2   cases act :
3    (mk-Task0(stmtl )      → transform-stmtlist (tnm, stmtl, tl actl, term, nodes1)(dict),
4     mk-Output0(,, )       → transform-output (tnm, act, tl actl, term, nodes1)(dict),
5     mk-Prioutput0()       → transform-prioutput (tnm, act, tl actl, term, nodes1)(dict),
6     mk-Create0(,)         → transform-create(tnm, act, tl actl, term, nodes1)(dict),
7     mk-Decision0(,,)      → transform-decision (tnm, act, nodes1)(dict),
8     mk-Option0(,, )       → transform-option(tnm, act, nodes1)(dict),
9     mk-Reset0()           → transform-reset (tnm, act, tl actl, term, nodes1)(dict),
10    mk-Set0()             → transform-set (tnm, act, tl actl, term, nodes1)(dict),
11    mk-Export0(idl )      → (⟨ transform-export (idl [i ])(dict) | 1 ≤ i ≤ len idl ⟩, dict),
12    mk-Call0(, )          → transform-call(tnm, act, tl actl, term, nodes1)(dict)))
```

запись: [$Statename_0$] $Actstm_0^+$ [$Termstmt_0$] $Graph\text{-}node_1^*$ → *Dict* → ($Transition_1$ | $Graph\text{-}node_1^+$) *Dict*

Цель Преобразовать предложение действия в AS_1

Параметры

tnm Имя состояния, к которому относится предложение действия. Имя используется для замещения черточки следующего состояния

actl Список предложений действия. Первое предложение действия в этом списке является предложением, которое преобразуется. Оставшиеся используются только, если предложение действия содержит выражения импорта

term Терминатор, который следует за списком предложений действия. Терминатор используется только, если предложение действия содержит выражения импорта

*nodes*1 Предыдущие вершины, которые уже преобразованы.

Результат $Graph\text{-}node_1$, если предложение действия не содержит выражений импорта, в противном случае $Transition_1$, который завершается в неявном состоянии импорта. *Dict* изменяется для включения информации об выходных сигналах и неявных переменных

Алгоритм

Строки 1-2 Преобразовать действие (*act*), которое является либо
Строка 3 Работой, либо

Строка 4	выводом, либо
Строка 5	приоритетным выводом, либо
Строка 6	запросом создания, либо
Строка 7	решением, либо
Строка 8	выбором, либо
Строка 9	сбросом таймера, либо
Строка 10	установкой таймера, либо
Строка 11	экспортом, либо
Строка 12	вызовом процедуры
	Заметим, что метка в $Actstmt_0$ больше не используется, так как ассоциированная информация о ней содержится в *Labeldict*

transform-stmtlist (*tnm, stmtl, actl, term,* $nodes_1$)(*dict*) ≜ (3.7.8.2)

```
1   (if stmtl = ⟨⟩ then
2     (nodes1, dict )
3    else
4     (let (as1 task, d) = cases hd stmtl :
5                          (mk-Assignstmt0(, )
6                            → transform-assign(hd stmtl )(dict ),
7                          mk-Text0(text )
8                            → (transform-informal (text ), dict )) in
9      if d (IMPORTLIST) = ⟨⟩ then
10       transform-stmtlist(tnm, tl stmtl, actl, term, nodes1 ⌢ as1 task )(d )
11      else
12       (let rest = if tl stmtl = ⟨⟩ then ⟨⟩ else ⟨mk-Task0(tl stmtl )⟩ in
13        let (outputtrans, d' ) = transform-import (tnm, as1 task, rest ⌢ actl, term)(d ) in
14        transform-transition(tnm, outputtrans, nodes1)(d' ))))
```

запись: [$Statename_0$] ($Assignstmt_0$* | $Text_0$*) $Actstmt_0$* [$Termstsmt_0$] $Graph\text{-}node_1$* → *Dict* → (*Transition*$_1$ | $Graph\text{-}node_1$+) *Dict*

Цель Преобразовать список предложений в AS_1

Параметры

stmtl	Список предложений, которые должны быть преобразованы
tnm,	
actl,	
term,	
$nodes_1$	См. *transform-act* и введение для этого подраздела

Результат См. *transform-act* и введение для этого подраздела

Алгоритм

Строка 1	Когда просмотр закончен, возвратить список AS_1 вершин работ, каждая из которых содержит предложение, и возвратить измененный *Dict*.
Строки 4-7	Пусть as_1 *task* обозначает работу первого предложения в списке.
Строка 5	Если первым предложением является предложение присваивания, тогда преобразовать его.
Строка 7	Если первым предложением является неформальный текст, тогда преобразовать его и оставить *Dict* неизмененным.
Строки 9-10	Если в предложении нет выражений импорта, тогда преобразовать оставшиеся предложения, иначе
Строка 12	Построить работу, которая содержит оставшиеся предложения (если они имеются).
Строка 13	Изменить *Statedict*, чтобы включить информацию о неявном состоянии и построить переход, который содержит предложения действия, которые предварительно уже преобразованы, выход сигнала xtQUERY и следующее состояние неявного состояния

Строка 14 Преобразовать построенный переход

transform-informal (*text*) ≜ (3.7.8.3)

```
1  (let mk-Text0(t ) = text in
2   mk-Task-node1(mk-Informal-text1(t )))
```

запись: $Text_0 \rightarrow Informal\text{-}text_1$

Цель Преобразовать неформальный текст в его представление в AS_1.

transform-assign(mk-*Assignstmt*$_0$(*vid, expr*))(*dict*) ≜ (3.7.8.4)

```
1   (if is-Id0(vid) then
2      (let sortset = all-visible-sorts(dict) in
3       let (,sortset',) = transform-expr(expr, EXPRESSION, sortset )(dict ) in
4       let qual = get-visible-variable(vid, sortset', false)(dict ) in
5       let mk-VarD (q,,,, vqual ) = dict (qual ) in
6       let (as1tree,, d ) = transform-expr(expr, EXPRESSION, {get-parent (q)(dict )})(dict ) in
7       let as1 id = make-as1-identifier(vqual )(dict ) in
8       let as1 assign = mk-Assignment-statement1(as1 id, as1 tree) in
9       (mk-Task-node1(as1assign), d ))
10     else
11      transform-modifyassign(vid, expr)(dict ))
```

запись: $Assignstmt_0 \rightarrow Dict \rightarrow Task\text{-}node_1\ Dict$

Цель Преобразовать предложение присваивания в AS_1.

Параметры Предложение присваивания, содержащее

vid	Переменная, которая должна быть присвоена
expr	Выражение правой стороны

Результат AS_1 предложение присваивания и *Dict*, измененный для включения информации о выражениях импорта в *expr*, и неявные переменные

Алгоритм

Строка 1	Если переменная является идентификатором, тогда
Строки 2-4	Построить ее *Qual* посредством извлечения сортов, которые соответствуют выражению (строка 3), и использования этих сортов при решении правой стороны контекстом (строка 4).
Строка 5	Произвести декомпозицию дескриптора переменной. Пусть *q* обозначает *qual* сорта и пусть *vqual* обозначает его уникальный *Qual* (различные в случае сервисов).
Строка 6	Преобразовать правую сторону выражения, которая должна быть сортом *q*.
Строка 7	Построить AS_1 идентификатор из его уникального *Qual*.
Строка 8	Построить AS_1 предложение присваивания.
Строка 9	Построить и возвратить работу, которая содержит предложение присваивания.
Строка 11	Если переменная не является идентификатором, тогда предложение присваивания является сокращенной нотацией для MODIFY! или предложением модификации поля

transform-modifyassign(vid, expr)(dict) ≜ (3.7.8.5)

```
 1  (let makeexpr(var) =
 2      if is-Indexedvar0(var) then
 3        mk-Operatorapp0(makeexpr(s-Variable0(var)), s-Exprlist0(var))
 4       else
 5        if is-Fieldvar0(var) then
 6          mk-Selectexpr0(makeexpr(s-Expr0(var)), s-Name0(var))
 7         else
 8          var in
 9   cases vid :
10    (mk-Indexedvar0(vid', exprlist )
11     → (let opid = mk-Id0(⟨⟩, mk-Name0("MODIFY", EXCLAMATION)) in
12        let vid" = makeexpr(vid') in
13        let righthandside = mk-Operatorapp0(opid, ⟨ vid" ⟩ ⌢ exprlist ⌢ ⟨ expr ⟩) in
14        let (as1node, dict') =
15            (trap exit with (nil, dict ) in
16             transform-assign(mk-Assignstmt0(vid', righthandside))(dict )) in
17        let (as1 node', dict") =
18            if is-Id0(exprlist [len exprlist ]) ∧
19             s-Qualifier0(exprlist [len exprlist ]) = ⟨⟩ then
20              (let field = s-Name0(exprlist [len exprlist ]) in
21               let vid" = if len exprlist = 1 then
22                            vid'
23                           else
24                            mk-Indexedvar0(vid', ⟨ exprlist [i ] | 1 ≤ i ≤ len exprlist - 1⟩) in
25               let vid''' = mk-Fieldvar0(vid", field ) in
26               trap exit with (nil, dict ) in
27               transform-assign(mk-Assignstmt0( vid''', expr))(dict ))
28            else
29             (nil, dict ) in
30        (as1 node = nil ∧ as1 node' ≠ nil
31          → (as1 node', dict"),
32         as1 node ≠ nil ∧ as1 node' = nil
33          → (as1 node, dict'),
34        T → exit("§5.5.3: Нет или несколько раскрытий измененных присваиваний"))),
35   mk-Fieldvar0(vid', mk-Name0(str, ))
36     → (let opid = mk-Id0(⟨⟩, mk-Name0(str ⌢ "MODIFY", EXCLAMATION)) in
37        let righthandside = mk-Operatorapp0(opid, ⟨ vid', expr ⟩) in
38        transform-assign(mk-Assignstmt0(vid', righthandside))(dict ))))
```

запись: *Variable0 Expr0 → Dict → Task-node1 Dict*

Цель Преобразовать предложение присваивания индексированной переменной или переменной поля в AS_1

Параметры

vid	Переменная с левой стороны
expr	Выражение с правой стороны

Результат Вершина работы, которая содержит AS_1 представление предложения присваивания в сокращенной записи

Алгоритм

Строки 1-8 Построить рекурсивную функцию, которая преобразует индексированную левую сторону в индексированную правую сторону(т.е. применение оператора0), и поле с левой стороны в поле с правой стороны (*Selectexpr0*)

Строки 10-16 Раскрыть сокращенную запись предложение присваивания индексированной переменной. Например,

```
v(e1)(e2, e3) := e4
```

раскрывается в

```
v(e1) := MODIFY! (v(e1), e2, e3, e4),
```

которое через *transform-assign* раскрывается в

```
v := MODIFY! (v, e1, MODIFY! (v(e1), e2, e3, e4))
```

до того, как оно преобразуется в AS_1. Если преобразование ошибочно, ошибка отлавливается, так как это может означать, что присваивание является другим сокращением (построенном в строках 17-29).

Строка 11 Построить идентификатор предложения MODIFY! и

Строка 12 Пусть *vid"* обозначает преобразованную левую сторону.

Строка 13 Построить новую правую сторону.

Строка 16 Преобразовать новое предложение присваивания.

Строки 17-29 Раскрыть круглые скобки предложения присваивания поля. Например,

```
v(e1)(e2, e3) := e4
```

раскрывается в

```
v(e1)(e2)!e3 := e4
```

при условии, что e3 является неквалифицированным идентификатором. Это построенное $Fieldvar_0$ преобразуется в AS_1 и, если преобразование ошибочно, ошибка отлавливается, так как это может означать, что присваивание является другим сокращением (построенным в строках 11-16).

Строки 17-19 Если последнее выражение в списке выражений не является неквалифицированным идентификатором, тогда это сокращение является невозможным.

Строка 20 Извлечь имя поля

Строки 21-24 Если еще существуют выражения в списке выражений, тогда построить другой $Indexvar_0$, иначе вернуть $Variable_0$ (*vid'*).

Строка 25 Построить конструкцию выбора поля.

Строки 26-27 Постараться преобразовать предложение присваивания выбора поля.

Строки 30-34 Должно быть одно и только одно из двух возможных раскрытий, которое является корректным.

Строки 35-38 Раскрыть предложение присваивания выбора поля, например,

```
a ! b ! c := e
```

раскрывается в

```
a ! b := cMODIFY!(a ! b, e),
```

которое через *transform-assign* раскрывается в

```
a := bMODIFY!(a, cMODIFY!(a ! b, e))
```

до того, как оно преобразуется в AS_1.

Строка 36 Построить идентификатор предложения модификации, который модифицирует поле посредством проверки *str*.

Строка 37 Построить новую правую сторону.

Строка 38 Преобразовать новое предложение присваивания

transform-reset (*tnm, resetnode, actl, term, nodes*$_1$)(*dict*) ≜ (3.7.8.6)

```
1  (let mk-Reset0(resetlist ) = resetnode in
2   let (actl1, d ) = transform-reset-elems(tnm, resetlist, actl, term, nodes1)(dict ) in
3   if tl resetlist = ⟨⟩ ∨ is-Transition1(actl1) then
4     (actl1, d )
5   else
6     transform-reset (tnm, mk-Reset0(tl resetlist ), actl, term, nodes1 ⌢ actl1)(d ))
```

запись: [*Statename*$_0$] *Reset*$_0$ *Actstmt*$_0$* [*Termstmt*$_0$] *Graph-node*$_1$* → *Dict* → (*Transition*$_1$ | *Graph-node*$_1$*) *Dict*

Цель Преобразовать действие сброса в AS$_1$

Параметры

tnm,	
actl,	
term,	
nodes$_1$	См. *transform-act*
resetnode	Действие сброса, которое должно быть преобразовано

Результат См. *transform-act*

Алгоритм

Строка 1	Пусть *resetlist* обозначает список таймеров, которые должны быть сброшены.
Строка 2	Преобразовать первый таймер, который должен быть сброшен.
Строка 3	Если этот сброс таймера является последним в списке (функция является рекурсивной) или если переход возвращается (из-за выражений импорта/экспорта в списке выражений), тогда остановить преобразование (если список выражений содержит выражения импорта, тогда оставшиеся таймеры должны быть преобразованы, когда неявное состояние импорта преобразовывается).
Строка 6	Иначе преобразовать оставшиеся сбросы таймеров

transform-reset-elems(*tnm, resetlist, actl, term, nodes*$_1$)(*dict*) ≜ (3.7.8.7)

```
1  (let mk-Resetelem0(timerid, actparm) = hd resetlist in
2   let squal = get-visible-qual (timerid, SIGNAL)(dict ) in
3   if is-SignalD (dict (squal )) then
4     exit("§2.8: Идентификатор в действии сброса не является таймером")
5   else
6     (let mk-TimerD (tplist, ) = dict (squal ) in
7      if len actparm = len tplist then
8        (let (as1 actparm, d ) = transform-actparms(tplist, actparm, EXPRESSION)(dict ) in
9         let as1 id = make-as1-identifier(squal )(dict ) in
10        let as1 tree = mk-Reset-node1(as1 id, as1 actparm) in
11        if d (IMPORTLIST) = ⟨⟩ then
12          (⟨as1 tree ⟩, d )
13        else
14          (let rest = if tl resetlist = ⟨⟩ then ⟨⟩ else ⟨mk-Reset0(tl resetlist )⟩ in
15           let (outputtrans, d') = transform-import (tnm, as1 tree, rest ⌢ actl, term)(d ) in
16           transform-transition(tnm, outputtrans, nodes1)(d')))
17     else
18       exit("§2.8: Длина списка параметров в действии сброса должна соответствовать длине, указанной в определении")))
```

запись: [*Statename*$_0$] *Resetelem*$_0$* *Actstmt*$_0$* [*Termstmt*$_0$] *Graph-node*$_1$* → *Dict* → (*Transition*$_1$ | *Graph-node*$_1$*) *Dict*

Цель Преобразовать сброс таймера в AS$_1$

Параметры См. *transform-reset*

Результат См. *transform-reset*

Алгоритм

Строка 1	Произвести декомпозицию $Resetelem_0$, который собираются преобразовать.
Строка 2	Извлечь *Qual* специфицированного идентификатора.
Строка 3	Идентификатор должен обозначать сигнал таймера.
Строка 6	Пусть *tplist* обозначает сорта списка выражений, определенного для таймера.
Строка 7	Должно быть столько выражений в списке выражений, сколько сортов для таймера.
Строка 8	Преобразовать выражения.
Строка 9	Построить AS_1 идентификатор для таймера.
Строка 10	Построить AS_1 вершину сброса.
Строка 11	Если в списке выражений нет выражений импорта, тогда
Строка 12	Возвратить AS_1 вершину сброса, в противном случае
Строка 14	Построить AS_0 действие сброса для оставшихся сбросов таймера. Это действие должно быть привязано к неявному состоянию импорта.
Строка 15	Изменить *Dict* при помощи состояния импорта и построить AS_0 переход, который ведет к состоянию импорта.
Строка 16	Преобразовать переход

transform-set (*tnm, setnode, actl, term,* $nodes_1$)(*dict*) ≜ (3.7.8.8)

```
1  (let mk-Set0(setlist ) = setnode in
2   let (actl1, d ) = transform-set-elems(tnm, setlist, actl, term, nodes1)(dict ) in
3   if tl setlist = ⟨⟩ ∨ is-Transition1(actl1) then
4     (actl1, d )
5    else
6     transform-set (tnm, mk-Set0(tl setlist ), actl, term, nodes1 ⇝ actl1)(d ))
```

запись: [$Statename_0$] Set_0 $Actstmt_0$* [$Termstmt_0$] $Graph\text{-}node_1$* →
Dict → ($Transition_1$ | $Graph\text{-}node_1$*) *Dict*

Цель Преобразовать действие установки в AS_1

Параметры

tnm,	
actl,	
term,	
$nodes_1$	См. *transform-act*
resetnode	Действие установки, которое должно быть преобразовано

Результат См. *transform-act*

Алгоритм

Строка 1	Пусть *setlist* обозначает список таймеров, которые должны быть установлены.
Строка 2	Преобразовать первый таймер, который должен быть установлен.
Строка 3	Если эта установка таймера является последней в списке (функция является рекурсивной), или если переход возвращается (из-за выражений импорта/экспорта в списке выражений), тогда остановить преобразование (если список выражений содержит выражения импорта, тогда остальные таймеры должен быть преобразованы, когда неявное состояние импорта преобразовывается).
Строка 6	Иначе преобразовать оставшиеся установки таймеров

transform-set- elems(*tnm, setlist, actl, term, nodes*$_1$)(*dict*) ≜ (3.7.8.9)

```
 1  (let mk-Setelem0(timeexpr, timerid, actparm) = hd setlist in
 2   let timesort = get-predef-sort ("TIME")(dict ) in
 3   let (as1 expr, d ) = transform-expr(timeexpr, EXPRESSION, {timesort })(dict ) in
 4   let squal = get-visible-qual (timerid, SIGNAL)(dict ) in
 5   if is-SignalD(dict (squal )) then
 6     exit("§2.8: Идентификатор в действии установки не является таймером")
 7    else
 8     (let mk-TimerD(tplist, ) = dict (squal ) in
 9      if len actparm = len tplist then
10        (let (as1 actparm, d' ) = transform-actparms(tplist, actparm, EXPRESSION)(d ) in
11         let as1 id = make-as1-identifier(squal )(dict ) in
12         let as1 tree = mk-Set-node1(as1 expr, as1 id, as1 actparm) in
13         if d' (IMPORTLIST) = ⟨⟩ then
14           (⟨ as1 tree ⟩, d' )
15          else
16           (let rest = if tl setlist = ⟨⟩ then ⟨⟩ else (mk-Set0(tl setlist )⟩ in
17            let (outputtrans, d" ) =
18                transform-import (tnm, as1 tree, rest → actl, term)(d' ) in
19           transform-transition(tnm, outputtrans, nodes1)(d" )))
20      else
21       exit("§2.8: Длина списка параметров в действии установки должна быть равна длине, указанной в определении")))
```

запись: [*Statename*$_0$] *Setelem*$_0$* *Actstmt*$_0$* [*Termstmt*$_0$] *Graph-node*$_1$* →
Dict → (*Transition*$_1$ | *Graph-node*$_1$*) *Dict*

Цель Преобразовать установку таймера в AS_1

Параметры См. *transform-set*

Результат См. *transform-set*

Алгоритм

Строка 1	Произвести декомпозицию *Setelem*$_0$, который собираются преобразовать.
Строка 2	Пусть *timesort* обозначает *Qual* сорта времени.
Строка 3	Преобразовать определенное выражение времени.
Строка 4	Извлечь *Qual* определенного идентификатора.
Строка 5	Идентификатор должен обозначать сигнал таймера.
Строка 8	Пусть *tplist* обозначает сорта списка выражений, определенного для таймера.
Строка 9	В списке должно быть столько выражений, сколько существует сортов для таймера.
Строка 10	Преобразовать выражения.
Строка 11	Построить AS_1 идентификатор таймера.
Строка 12	Построить AS_1 вершину установки.
Строка 13	Если отсутствуют выражения импорта в списке выражений или в выражении таймера, тогда
Строка 14	Возвратить AS_1 вершину установки, иначе
Строка 16	Построить AS_0 действие установки оставшихся установок таймера. Это действие должно быть связано с неявным состоянием импорта.
Строка 17	Изменить *Dict* при помощи состояния импорта и построить AS_0 переход, который ведет к состоянию импорта.
Строка 19	Преобразовать переход

ort (*id*)(*dict*) ≜ (3.7.8.10)

```
1  (let sortset = all-visible-sorts(dict ) in
2   let qual = get-visible-variable(id, sortset, true)(dict ) in
3   let mk-SignalnamesD(impnm,, ) = exportmap((s-Sortqual (dict (qual )), s-Name0(id ))) in
4   let as1 id = make-as1-identifier(qual )(dict ),
5       as1 id' = make-as1-identifier(impnm)(dict ) in
6  (mk-Task-node1(mk-Assignment-statement1(as1 id', as1 id ))))
```

запись: $Varid_0 \rightarrow Dict \rightarrow Task\text{-}node_1$

Цель Преобразовать действие экспорта в AS_1 предложение присваивания

Параметры

id AS_0 идентификатор экспортируемой переменной

Результат AS_1 работа, содержащая построенное предложение присваивания

Алгоритм

Строки 1-2 Построить *Qual* экспортируемой переменной, используя контекст.

Строка 3 Пусть *impnm* обозначает имя неявной переменной, связанной с экспортируемой переменной

Строки 4-5 Построить AS_1 идентификаторы явной переменной экспорта (as_1 *id*) и неявной переменной экспорта (as_1 *id'*) и

Строка 6 Возвратить работу, которая присваивает явную переменную неявной переменной

3.7.8.1 Преобразование вершин вывода

transform-prioutput (*tnm*, mk-$Prioutput_0$(*outputsigs*), *actl*, *term*, $nodes_1$)(*dict*) ≜ (3.7.8.1.1)

```
1  (let self = mk-Selfexpr0() in
2   let (actl1, d ) = transform-output-elems(tnm, outputsigs, self, (), actl, term, nodes1, true)(dict ) in
3   if tl outputsigs = () ∨ is-Transition1(actl1) then
4     (actl1, d )
5    else
6     transform-prioutput (tnm, mk-Prioutput0(tl outputsigs), actl, term, nodes1 ⌢ actl1)(d ))
```

запись: [$Statename_0$] $Prioutput_0$ $Actstmt_0$* [$Termstmt_0$] $Graph\text{-}node_1$* →
$Dict \rightarrow (Transition_1 \mid Graph\text{-}node_1^*)\ Dict$

Цель Преобразовать приоритетное действие вывода в последовательность AS_1 действий вывода (по одному на каждый задействованный сигнал)

Параметры

tnm,
actl,
term,
$nodes_1$ См. *transform-act*

outputsigs Список выводов, содержащихся в приоритетном действии вывода

Результат См. *transform-act*

Алгоритм

Строки 1-6 Алгоритм следует той же схеме, что и *transform-reset*. Аргумент true для функции *transform-output-elems* указывает, что сигналы являются сигналами с высоким приоритетом

transform-output (*tnm, outnode, actl, term, nodes*$_1$)(*dict*) ≜ (3.7.8.1.2)

1 (**let** **mk**-*Output*$_0$(*outputsigs, expr, via*) = *outnode* **in**
2 **let** (*actl*$_1$, *d*) = *transfosrm-output-elems*(*tnm, outputsigs, expr, via, actl, term, nodes*$_1$, **false**)(*dict*) **in**
3 **if** **tl** *outputsigs* = ⟨⟩ ∨ **is**-*Transition*$_1$(*actl*$_1$) **then**
4 (*actl*$_1$, *d*)
5 **else**
6 *transform-output* (*tnm*, **mk**-*Output*$_0$(**tl** *outputsigs, expr, via*), *actl, term, nodes*$_1$ ⌢ *actl*$_1$)(*d*))

запись: [*Statename*$_0$] *Output*$_0$ *Actstmt*$_0$* [*Termstmt*$_0$] *Graph-node*$_1$* →
Dict → (*Transition*$_1$ | *Graph-node*$_1$*) *Dict*

Цель Преобразовать действие вывода в последовательность AS$_1$ действий вывода (по одному на каждый задействованный сигнал)

Параметры

tnm,
actl,
term,
nodes$_1$ См. *transform-act*
outputsigs Список выводов, содержащихся в действии вывода

Результат См. *transform-act*

Алгоритм

Строки 1-3 Алгоритм следует той же схеме, что и *transform-reset*. Аргумент **false** для функции *transform-output-elems* указывает, что сигналы не являются сигналами с высоким приоритетом

transform-output-elems(*tnm, sigs, expr, via, actl, term, nodes*$_1$, *isprioutput*)(*dict*) ≜ (3.7.8.1.3)

```
 1  (let mk-Outputsig0(sigid, actparm) = hd sigs in
 2   let instsort = get-predef-sort ("PID")(dict ) in
 3   let (as1 expr, d) =
 4       if expr = nil
 5         then (nil, dict)
 6         else transform-expr(expr, EXPRESSION, {instsort })(dict ) in
 7   let squal = signal-qual (sigid )(dict ) in
 8   let qual = process-or-service-level (dict (SCOPEUNIT)) in
 9   let existreceiver = (∃q ∈ dom dict )(is-ServiceD(dict (q)) ∧
10                                       get-sur(q) = get-sur(qual) ∧
11                                       squal ∈ s-Priinputset (dict (q))) in
12   if (is-ProcessD(dict (qual )) ∧ isprioutput) ∨
13    (isprioutput ≠ existreceiver) then
14     exit("§4.10: Недопустимое использование сигнала с высоким приоритетом")
15    else
16     (let qualset = transform-via(squal, via)(dict ) in
17      let as1 pathidset = make-as1 idset (qualset )(dict ) in
18      let dict' = d + [OUTSIGNALS ↦ dict (OUTSIGNALS) ∪ {squal }] in
19      if is-TimerD(dict (squal )) then
20        exit("§2.7.4: Идентификатор в действии вывода должен обозначать сигнал")
21       else
22        (let mk-SignalD(tplist,, ) = dict (squal ) in
23         if len actparm = len tplist then
24           (let (as1 actparm, d') =
25                transform-actparms(tplist, actparm, EXPRESSION)(dict') in
26            let as1 id = make-as1-identifier(squal )(dict ) in
27            let as1 tree = mk-Output-node1(as1 id, as1 actparm, as1 expr, as1 pathidset ) in
28            if d'(IMPORTLIST) = ⟨⟩ then
29              (⟨as1 tree ⟩, d')
30             else
31              (let rest = if tl sigs = ⟨⟩ then
32                          ⟨⟩
33                          else
34                          ⟨mk-Output0(tl sigs, expr, via)⟩ in
35               let (outputtrans, d") =
36                   transform-import (tnm, as1 tree, rest ⇀ actl, term)(d') in
37               transform-transition (tnm, outputtrans, nodes1)( d")))
38         else
39          exit("§2.7.4: Неправильное число аргументов в действии вывода"))))
```

запись: [*Statename*$_0$] *Outputsig*$_0$* [*Expression*$_1$] *Via*$_0$ *Actstmt*$_0$* [*Termstmt*$_0$] *Graph-node*$_1$* *Bool* → *Dict* → (*Transition*$_1$ | *Graph-node*$_1$*) *Dict*

Цель Преобразовать вывод или вывод с приоритетом в AS$_1$

Параметры

tnm, *actl*, *term*, *nodes*$_1$	См. *transform-act*
sigs	Список выводов, которые содержаться в действии вывода. В этой функции преобразуется только первый элемент этого списка
expr	Выражение назначения. Если вывод является выводом с приоритетом, тогда *expr* является SELF
via	Список идентификаторов каналов или маршрутов сигналов, содержащийся в вершине вывода. Если вывод является выводом с приоритетом, тогда этот список пуст

Результат См. *transform-act*

Алгоритм

Строка 1	Пусть *sigid* обозначает идентификатор сигнала вывода, который должен быть преобразован, и пусть *actparm* обозначает список связанных фактических параметров.
Строка 2	Извлечь *Qual* сорта PiD.
Строки 3-6	Если выражение назначения специфицировано, тогда преобразовать его в AS_1. Оно должно быть сорта PiD (*instsort*).
Строка 7	Пусть *squal* обозначает *Qual* сигнала.
Строка 8	Пусть *qual* обозначает *Qual* окружающего сервиса или процесса (но не процедуры)
Строки 9-11	*existreceiver* принимает значение **true**, если существует сервис (*q*) в том же процессе (строка 10), который может принимать сигнал в вершине ввода с высоким приоритетом.
Строки 12-13	Если вывод содержится в процессе, тогда он не должен быть выводом с высоким приоритетом (строка 12), и, если он является сигналом с высоким приоритетом, тогда должен существовать принимающий сервис, и наоборот.
Строки 16-17	Построить множество VIA AS_1.
Строка 18	Изменить вход OUTSIGNALS *Dict*, чтобы включить *squal* выходного сигнала.
Строка 19	Специфицированный сигнал не должен быть сигналом таймера.
Строка 22	Пусть *tplist* обозначает список сортов, связанный с этим сигналом.
Строка 23	Длина списка фактических параметров должна соответствовать длине списка сортов.
Строка 24	Преобразовать фактические параметры.
Строка 26	Построить AS_1 идентификатор сигнала.
Строка 27	Построить AS_1 вершину вывода.
Строки 28-29	Если в выражении назначения и в фактических параметрах отсутствуют выражения импорта, тогда возвратить вершину вывода, иначе
Строка 31	Построить AS_0 действие вывода для оставшихся выводов. Это действие должно быть связано с неявным состоянием импорта.
Строка 35	Изменить *Dict* посредством состояния импорта и построить переход AS_0, который ведет в состояние импорта.
Строка 37	Преобразовать переход

transform-via(signalqual, pathlist)(dict) ≜ (3.7.8.1.4)

```
1   (let processlevel = process-level (dict (SCOPEUNIT)) in
2    let servicelevel = process-or-service-level (dict (SCOPEUNIT)) in
3    let mk-ProcessD(, sset,, connectmap) = dict (processlevel ) in
4    let explicitsigroutes = s-Explicitroutes(dict (get-sur(processlevel ))) in
5    if pathlist = ⟨⟩ then
6      if ¬explicitsigroutes then
7        {}
8      else
9        if (∃mk-SignalrouteD(endp1, endp2, sigset 1, sigset 2) ∈ dom dict )
10          ((processlevel = endp 1 ∧ signalqual ∈ sigset 1) ∨
11           (processlevel = endp 2 ∧ signalqual ∈ sigset 2)) then
12         {}
13       else
14        if signalqual ∈ sset then
15          {}
16         else
17          exit("§2.7.4: Для сигнала на выходе нет допустимого приемника")
18     else
19     (let id = hd pathlist in
20      let qual = (trap exit with get-visible-qual (id, CHANNEL)(dict ) in
21                  get-visible-qual (id, SIGNALROUTE)(dict )) in
22      let qualrest = if tl pathlist = ⟨⟩ then
23                       {}
24                     else
25                       transform-via(signalqual, tl pathlist )(dict ) in
26      if qual ∈ qualrest then
27        exit("§2.7.4: В множестве VIA дважды определены маршрут сигналов или канал")
28       else
29         cases dict (qual ):
30          (mk-SignalrouteD(endp 1, endp 2, sigset 1, sigset 2)
31            → (if (processlevel = endp 1 ∧ signalqual ∈ sigset 1) ∨
32              (processlevel = endp 2 ∧ signalqual ∈ sigset 2) then
33              if connectmap ≠ [] ∧ is-ServiceD(dict (servicelevel )) ∧
34               ¬(∃q ∉ connectmap(qual ))
35                 ((let mk-SignalrouteD(ep1, ep2, sigs1, sigs2) = dict (q) in
36                  (servicelevel = ep 1 ∧ signglqual ∈ sigs 1) ∨
37                  (servicelevel = ep 2 ∧ signalqual ∈ sigs 2))) then
38               exit("§4.10.2: В VIA нет сигналов сервиса, связанных с маршрутом сигналов")
39              else
40               if is-ProcessD(dict (servicelevel )) ∧ connectmap ≠ []
41                then exit("§2.10.2: Недопустимые внешние сервисы VIA, когда определены маршруты сигналов сервиса")
42                else {qual } ∪ qualrest
43              else
44               exit("§2.7.4: Недопустимое множество VIA")),
```

```
45        mk-ChannelD(,,,,)
46         → if explicitsigroutes then
47             exit("§2.7.4: Если маршруты сигналов специфицированы, тогда канал не может быть задействован в VIA")
48            else
49             (let blockqual = get-sur(processlevel) in
50             let mk-BlockD(, , , cmap) = dict(blockqual) in
51             if qual ∈ dom cmap
52               then exit("§2.7.4: Канал в множестве VIA не подсоединен к блоку")
53              else (let squalset =
54                      {q ∈ cmap(qual) | (let mk-SignalrouteD(endp1, endp2, sigset1, sigset2) =
55                                          dict(q) in
56                                        (processlevel = endp1 ∧ signalqual ∈ sigset1) ∨
57                                        (processlevel = endp2 ∧ signalqual ∈ sigset2))} in
58                  if squalset = {}
59                    then exit("§2.7.4: Сигнал не может быть передан по каналу в множестве VIA")
60                    else squalset ∪ qualrest)))))
```

запись : *Signalqual* Via_0 → *Dict* → *Qual*-set

Цель Преобразовать список каналов или идентификаторы маршрутов сигналов, определенных в конструкции VIA, в множество *Qual* маршрутов сигналов

Параметры

signalqual	*Qual* сигнала, определенного в выводе
pathlist	AS_0 список via

Результат Множество *Qual* маршрутов сигналов

Алгоритм

Строка 1	Пусть *processlevel* обозначает *Qual* окружающего процесса (но не процедуры или сервиса).
Строка 4	Пусть *explicitsigroutes* равно true, если маршруты сигналов определены в блоке окружения.
Строка 5	Если идентификаторы каналов или идентификаторы маршрутов сигналов не определяются, тогда
Строка 6	Если маршруты сигналов не специфицированы для блока окружения, тогда возвратить пустое множество, иначе
Строки 9-11	Если существует маршрут сигналов, который включает в себя специфицированный сигнал и который имеет процесс окружения в качестве одной из конечных точек, тогда возвратить пустое множество, указывающее, что сигнал может передаваться любым путем, иначе
Строка 14	Если явные маршруты сигналов определены и сигнал не содержится ни в одном из них, тогда сигнал должен быть частью совокупности допустимых входных сигналов для окружающего процесса.
Строки 19-59	Если маршруты сигналов или каналы определяются в VIA, тогда
Строка 19	Пусть *id* обозначает идентификатор первого канала или маршрута сигналов в списке.
Строки 20-21	Искать идентификатор в *Dict*, как маршрут сигналов. Если ошибка, тогда он должен обозначать канал, искать идентификатор в *Dict*, как канал.
Строка 22	Преобразовать остальные каналы или маршруты сигналов в списке via.
Строка 26	Если маршрут сигналов или канал также встречаются в *Quals* оставшихся списка via, тогда канал или маршрут сигналов был определен дважды.
Строки 30-34	Если *Qual* обозначает маршрут сигналов, тогда сигнал должен быть включен в маршрут сигналов, окружающий процесс должен быть одной из его конечных точек, и, если маршруты сигналов в сервисе специфицированы (т.е. *connectmap* не является пустым), тогда должен существовать маршрут сигналов в сервисе, подсоединенный к сервису и к маршруту сигналов (строки 33-38).

Строка 40	Если маршруты сигналов в сервисе специфицированы, тогда VIA не может быть использован в процедурах на уровне процесса.
Строка 45	Если *Qual* обозначает канал, тогда
Строка 46-47	Он должен быть, потому что маршруты сигналов не определяются явно
Строка 49	Пусть *blockqual* обозначает *Qual* блока окружения.
Строка 50	Пусть *cmap* обозначает *BlockconnectionD* для блока.
Строки 51-52	Канал в VIA должен быть соединен с блоком.
Строки 53-57	Пусть *squalset* обозначает *Quals* маршрутов сигналов, которые связаны с каналом и для которых сигнал является исходящим.
Строки 58-59	Должен быть по крайней мере один такой маршрут сигналов.
Строка 60	Возвратить выбранные маршруты сигналов, объединенные с маршрутами сигналов для остальных в множестве VIA.

3.7.8.2 Преобразование вершин создания

transform-create(*tnm*, mk-*Create*$_0$(*pid*, *exprlist*), *actl*, *term*, *nodes*$_1$)(*dict*) ≜ (3.7.8.2.1)

```
1   (let pqual = get-visible-qual(pid, PROCESS)(dict) in
2    let pqual = get-sur(pqual) in
3    let as1id = make-as1-identifier(pqual)(dict) in
4    let mk-ProcessD(tplist,,,) = dict(pqual) in
5    if get-sur(process-level(dict(SCOPEUNIT))) ≠ bqual then
6       exit("§2.7.2: Созданный процесс должен быть определен в том же самом блоке")
7     else
8       (if len tplist ≠ len exprlist then
9          exit("§2.7.2: Длина фактических параметров должна соответствовать длине списка формальных параметров")
10       else
11         (let(actparmlist, d) = transform-actparms(tplist, exprlist, EXPRESSION)(dict) in
12          let as1tree = mk-Create-request-node1(as1id, actparmlist) in
13          if d(IMPORTLIST) = ⟨⟩ then
14             (⟨as1tree⟩, d)
15           else
16             (let(outputtrans, d ') = transform-import(tnm, as1tree, actl, term)(d) in
17              transform-transition(tnm, outputtrans, nodes1)(d ')))))
```

запись : [*Statename*$_0$] *Create*$_0$ *Actstmt*$_0$* [*Termstmt*$_0$] *Graph-node*$_1$* → *Dict* → (*Transition*$_1$ | *Graph-node*$_1$$^+$) *Dict*

Цель Преобразовать действие создания запроса в AS$_1$

Параметры

tnm, *actl*, *term*, *nodes*$_1$	См. *transform-act*
pid	Идентификатор процесса, содержащийся в действии создания запроса
exprlist	Список фактических параметров

Результат См. *transform-act*

Алгоритм

Строка 1	Пусть *pqual* обозначает *Qual* идентификатора процесса.
Строка 2	Пусть *bqual* обозначает *Qual* блока окружения.
Строка 3	Построить идентификатор AS$_1$ идентификатора процесса.
Строка 4	Пусть *tplist* обозначает сорты фактических параметров.

Строка 5	Окружения процесса, где помещается вершина создания запроса, должна быть такими же (блок), как окружение идентификатора процесса.
Строка 8	Длина списка сортов должна быть равна длине списка фактических параметров.
Строка 9	Преобразовать фактические параметры.
Строка 12	Построить вершину создания запроса AS_1.
Строки 13-14	Если список фактических параметров не содержит импортируемые выражения, тогда возвратить построенную вершину создания запроса AS_1.
Строка 16	Модифицировать *Dict* посредством состояния импорта и построить переход AS_0, который приведет к состоянию импорта.
Строка 17	Преобразовать переход

transform-actparms(*tplist*,*exprlist*,*context*)(*dict*) ≜ (3.7.8.2.2.)

```
1   (if tplist= ⟨⟩ then
2      (⟨⟩,dict)
3    else
4      (let  (as1expr,,d)=
5           if hd exprlist = nil
6             then (nil, , dict)
7             else transform-expr(hd exprlist, context, (hd tplist))(dict) in
8      let(as1rest, drest) = transform-actparms(tl tplist, tl exprlist, context)(d) in
9      (⟨as1expr⟩ ⌢ as1rest, drest)))
```

запись: $ParameterD^*\ Actparmlist_0\ Context \rightarrow Dict \rightarrow [Term_1\ Expression_1]^*\ Dict$

Цель Преобразовать фактические параметры действия вывода или созданного действия запроса или список аргументов операторов в AS_1.

Параметры

tplist	Сорты фактических параметров
exprlist	Фактические параметры
context	Контекст,в который преобразуются выражения. В случае действия вывода или действия создания запроса,контекст является всегда EXPRESSION.

Результат Список необязательных выражений в случае действия вывода или действия создания запроса список термов или выражений в случае фактических параметров операторов. *Dict* возможно модифицирован, чтобы включить в себя информацию об импортируемых выражениях в список фактических параметров.

Алгоритм

Строки 1-2	Когда просмотр завершен, возвратить пустой список и (неизмененный) *Dict*.
Строки 4-7	Преобразовать первый фактический параметр,если он представлен в списке фактических параметров.
Строка 8	Преобразовать остальные фактические параметры.
Строка 9	Возвратить первый фактический параметр,соединенный с остальными фактическими параметрами.

3.7.8.3 Преобразование вершин вызова

transform-call(*tnm*, mk-*Call*$_0$(*procid*, *actparmlist*), *actl*, *term*, *nodes*$_1$)(*dict*) ≜ (3.7.8.3.1)

```
1   (let pqual = get-visible-qual(procid, PROCEDURE)(dict) in
2    let mk-ProcedureD(formparmlist, nqual) = dict(pqual) in
3    let as1id = make-as1-identifier(nqual)(dict) in
4    if len actparmlist = len formparmlist then
5      (let(as1actparm, d) = transform-actparmlist(formparmlist, actparmlist)(dict) in
6       let as1tree = mk-Call-node1(as1id, as1actparm) in
7       if d(IMPORTLIST) = ⟨⟩ then
8         (⟨as1tree⟩, d)
9        else
10        (let(outputtrans, d ') = transform-import(tnm, as1tree, actl, term)(d) in
11         transform-transition(tnm, outputtrans, nodes1)(d ')))
12     else
13      exit("§2.7.3: Длина списка параметров процедуры должна соответствовать длине списка формальных параметров"))
```

запись : [*Statename*$_0$] *Call*$_0$ *Actstmt*$_0$* [*Termstmt*$_0$] *Graph-node*$_1$* → Dict → (*Transition*$_1$ | *Graph-node*$_1$*) *Dict*

Цель Преобразовать вызов процедуры в AS$_1$

Параметры

procid	Идентификатор процедуры
actparmlist	Список фактических параметров
tnm,	
actl,	
term,	
nodes$_1$	См. *transform-act*

Результат См. *transform-act*

Алгоритм

Строка 1	Построить *Qual* процедуры.
Строка 2	Пусть *formparmlist* обозначает дескрипторы формальных параметров, пусть *nqual* обозначает уникальный *Qual* (см. описание *Newqual*).
Строка 3	Построить идентификатор процедуры AS$_1$.
Строка 4	Длина списка формальных параметров должна быть равна длине списка фактических параметров.
Строка 5	Преобразовать фактические параметры.
Строка 6	Построить вершину вызова AS$_1$.
Строки 7-8	Если в фактических параметрах нет импортируемых выражений, тогда возвратить вершину вызова, иначе
Строка 10	Модифицировать *Statedict*, чтобы включить неявное состояние импорта и возвратить переход AS$_0$, ведущий в неявное состояние.
Строка 11	Преобразовать построенный переход

transform-actparmlist(*formparmlist*, *actparmlist*)(*dict*) ≜ (3.7.8.3.2)

```
1   (if formparmlist = ⟨⟩ then
2      (⟨⟩, dict)
3    else
4      (let (actparmlist, dict ') =
5          if hd actparmlist = nil then
6            (⟨nil⟩, dict)
7          else
8            cases hd formparmlist:
9             (mk-InDescr(tqual)
10              → (let (as1expr,, d) =
11                     transform-expr(hd actparmlist, EXPRESSION,{tqual})(dict) in
12                 (as1expr, d)),
13            mk-InoutDescr()
14              → transform-Inoutactparm(hd formparmlist, hd actparmlist)(dict)) in
15   let(actparmrest,drest) = transform-actparmlist(tl formparmlist, tl actparmlist)(dict ') in
16   (⟨actparm⟩ ⌢ actparmrest, drest)))
```

запись : $FormparmD^*\ Actparmlist_0 \rightarrow Dict \rightarrow [Expression_1]^*\ Dict$

Цель Преобразовать список фактических параметров процедуры в список выражений AS_1

Параметры

formparmlist	Список дескрипторов формальных параметров
actparmlist	Список фактических параметров

Результат Список выражений AS_1 и *Dict*, возможно модифицированный, чтобы включить в себя информацию о содержащихся импортируемых выражениях (см. *transform-act*)

Алгоритм

Строки 1-2	Когда просмотр закончен, возвратить пустой список параметров (функция рекурсивна).
Строки 4-14	Пусть *actparm* обозначает выражение AS_1, обозначающее первый фактический параметр.
Строки 5-6	Если фактический параметр опущен, тогда *actparm* является **nil**, иначе
Строки 9-12	Если параметр является параметром IN, тогда преобразовать выражение, которое является фактическим параметром. Оно должно быть сортом, обозначенным *tqual*.
Строки 13-14	Если параметр является параметром IN/OUT, тогда преобразовать параметр IN/OUT.
Строки 15-15	Преобразовать остальные фактические параметры.
Строка 16	Возвратить первый фактический параметр, соединенный с остальными фактическими параметрами

transform-inoutactparm(**mk-***InoutDescr*(*tqual*), *expr*)(*dict*) ≜ (3.7.8.3.3)

```
1   (if is-Id0(expr) then
2      (let vqual = get-visible-variable(expr, {get-parent(tqual)(dict)}, false)(dict) in
3       let mk-VarD(tq,,,, nqual) = dict(vqual) in
4       let as1id = make-as1-identifier(nqual)(dict) in
5       if tq = tqual then
6          (as1id, dict)
7        else
8          exit("§2.7.3: Один и тот же сорт должен быть специфицирован для формального и фактического параметра IN/OUT"))
9    else
10    exit("§2.1.3: Фактический параметр IN/OUT должен быть идентификатором переменной"))
```

запись : $InoutDescr\ Expr_0 \rightarrow Dict \rightarrow Expression_1\ Dict$

Цель Преобразовать фактический параметр IN/OUT в AS_1

Параметры Дескриптор IN/OUT, содержащий

tqual *Qual* его сорта

expr Фактический параметр

Результат Фактический параметр AS_1

Алгоритм

Строка 1 Фактический параметр должен быть идентификатором.

Строка 2 Построить *Qual* идентификатора.

Строка 3 Пусть *tq* обозначает его сорт или подтип, пусть *nqual* обозначает его *Newqual*.

Строка 4 Построить его идентификатор AS_1

Строки 5-8 Сорт переменной (*tq*) должен соответствовать сорту формального параметра. В этом случае тот же идентификатор ссылки на сорт должен быть специфицирован, т.е. сорт несовместим с подтипом сорта.

3.7.8.4 Преобразование вершин принятия решения

transform-decision(*tnm*, *decision*, $nodes_1$)(*dict*) ≜ (3.7.8.4.1)

```
1   (let mk-Decision0(quest, anslist, elsepart) = decision in
2   let isinformalanswers =
3      (∀mk-Condition0(v1, v2) ∈ elems anslist)(v1 = nil ∧ is-Stringterm0(v2) ∧ s-Qualifier0(v2) = ⟨⟩) in
4   let isinformalquestion = is-Stringterm0(quest) ∧ s-Qualifier0(quest) = ⟨⟩ in
5   let isinformaldecision = isinformalanswers ∧ isinformalquestion in
6   let qualset = all-visible-sorts(dict) in
7   let (,tpset,) = transform-expr(quest, EXPRESSION, qualset)(dict) in
8   let tp = if isinformaldecision then
9              nil
10            else
11            if isinformalquestion then
12               (trap exit with get-answer-sort(anslist, qualset)(dict) in
13               get-answer-sort(anslist, tpset)(dict))
14             else
15               get-answer-sort(anslist, tpset)(dict) in
16  let (as1trans, d ") = transform-answers(tnm, anslist, tp)(dict) in
17  let (els, d ''') =
18      if elsepart = nil then
19        (nil, d ")
20       else
21        (let mk-Elsepart0(trans) = elsepart in
22         let (trans1, dict ') = transform-transition(tnm, trans, ⟨⟩)(d ") in
23         (mk-Else-answer1(trans1), dict ')) in
24  let (as1quest ',, d) =
25     if isinformalquestion then
26        (let mk-String0(str) = s-String0(quest) in
27        if isinformaldecision then
28           (mk-Informal-text1(str), {}, dict)
29          else
30           (trap exit with (mk-Informal-text1(str), {}, dict) in
31           transform-expr(quest, EXPRESSION, {tp})(d ''')))
32       else
33        transform-expr(quest, EXPRESSION, {tp})(d ''') in
34  let decisionnode1 = mk-Decision-node1(as1quest ', as1trans, els) in
35  if d(IMPORTLIST) = ⟨⟩ then
36     (mk-Transition1(nodes1, decisionnode1), d)
37    else
38     (let (outputtrans, d ') = transform-import(tnm, decisionnode1, ⟨⟩, nil)(d) in
39      transform-transition(tnm, outputtrans, nodes1)(d '')))
```

запись : [$Statename_0$] $Decision_0$ $Graph\text{-}node_1{}^* \rightarrow Dict \rightarrow Transition_1$ $Dict$

Цель Преобразовать вершину принятия решения в AS_1'

Параметры

tnm,
decision,
$nodes_1$ — См. *transform-act*. В отличие от других функций преобразования действий, *transform-decision* не трогает список действий, следующий за решением в качестве аргумента, т.к. решение прекращает переход и потому, что все ответы на этом шаге содержат терминаторы

Результат Переход, содержащий $nodes_1$, и либо ведущий в вершину принятия решения, либо ведущий в неявное состояние импорта

Алгоритм

Строка 1	Произвести декомпозицию действия решения.
Строка 2	Пусть *isinformalanswers* будет true, если все ответы состоят из неквалифицированной символьной строки.
Строка 4	Пусть *isinformalquestion* будет true, если выражение в запросе состоит из неквалифицированной символьной строки.
Строка 5	Пусть *isinformaldecision* будет true, если решение состоит исключительно из неформального текста (символьных строк).
Строка 6	Пусть *qualset* обозначает *Quals* всех сортов, которые являются видимыми на этом месте.
Строка 7	Извлечь возможные сорты для запроса решения.
Строки 8-15	Пусть *tp* обозначает сорт ответов. Если он является неформальным решением, тогда *tq* равно nil, в противном случае сорт получается просмотром ответов и посредством этого ограничением множества сортов, доступных для запроса. Более того, если запрос является символьной строкой (строка 11), тогда любой обозреваемый сорт может соответствовать сорту ответов (строка 12), если он отказывает получить сорт ответов из множества сортов, удовлетворяющих запросу.
Строка 16	Преобразовать ответы, дающие их сорт (*tp*, который является nil, если все решение неформально).
Строки 17-23	Преобразовать часть else.
Строка 21	Пусть *trans* обозначает переход в части else.
Строки 22-23	Преобразовать переход и построить часть else AS_1.
Строки 24-33	Преобразовать запрос.
Строка 24	Если запрос является неквалифицированной символьной строкой, тогда он обозначает неформальный текст, если все решение является неформальным (строка 27), или, если преобразование запроса, как литерала символьной строки отказывает дать сорт ответа *tp* (строка 31).
Строка 34	Построить вершину принятия решений AS_1.
Строки 35-36	Если запрос не содержит выражения импорта, тогда возвратить переход, содержащий вершины ограничения и вершину принятия решения, иначе
Строка 38	Модифицировать *Statedict*, чтобы включить неявные состояния импорта и построить переход AS_0, ведущий в вершину неявного импорта.
Строка 39	Преобразовать построенный переход

get-answer-sort(*anslist*, *tpset*)(*dict*) ≜ (3.7.8.4.2)

```
1   (if anslist = ⟨⟩ then
2      if card tpset ≠ 1 then
3        exit("§2.2.2: Не существует подходящий сорт для действия решения")
4       else
5        (let tp ∈ Qual be s.t. tp ∈ tpset in
6         tp)
7      else
8      (let mk-Answer0(vlist,) = hd anslist in
9       let isinformal '=
10          len vlist = 1 ∧
11          (let mk- Condition0(v1, v2) = hd vlist in
12       v1 = nil ∧ is-Stringterm0(v2) ∧ s- Qualifier0(v2) = ⟨⟩) in
13       let(tset,) = if isinformal 'then (tpset, ) else transform-valueset(tpset, vlist)(dict) in
14       get-answer-sort(tl anslist, tset)(dict)))
```

запись : $Answer_0^+$ *Sortqual*-set → *Dict* → [*Sortqual*]

Цель Извлечь сорт ответов решений

Параметры

anslist	Список ответов, подключенных к действию решения (исключая **else** часть).
tpset	Возможные сорты ответов. В течении каждого повторного вызова *get-answer-sort* это множество ограничивается допустимым сортом для обрабатывамого ответа. При первоначальном применении *tpset* обозначает возможные сорты запроса решения и, если запрос решения неформальный, то все обозреваемые сорты.

Результат *Qual* сорта действия решения.

Алгоритм

Строки 1-5	Когда просмотрен список ответов, множество возможных сортов должно содержать только один элемент и этот элемент возвращается.
Строка 8	Произвести декомпозицию первого ответа в списке.
Строки 9-12	Пусть *isinformal'* истинно, если ответы состоят из неквалифицированной символьной строки.
Строка 13	Пусть *tset* обозначает множество сортов, возможных для ответов после рассмотрения первого (следующего) ответа. Если ответом является неквалифицированная символьная строка, то он не рассматривается, когда определяется сорт ответа.
Строка 14	Рассмотреть остальные ответы.

transform-answers(*tnm*, *anslist*, *tp*)(*dict*) ≜ (3.7.8.4.3)

```
1   (if anslist = {} then
2      ({}, dict)
3    else
4      (let mk-Answer0(vlist, trans) = hd anslist in
5       let (,as1tree) =
6           if tp = nil then
7              ({}, nil)
8            else
9              (trap exit with({}, nil) in
10              transform-valueset({tp}, vlist)(dict)) in
11     let (trans1, dict ') = transform-transition(tnm, trans, ⟨⟩)(dict) in
12     let (restrans, dict ") = transform-answers(tnm, tl anslist, tp)(dict ') in
13     if as1tree = nil then
14        (let mk-Condition0(, mk-Stringterm0(, lit)) = hd vlist in
15         ({mk-informal-text1(lit)} ∪ restrans, dict "))
16       else
17         ({mk-Decision-answer1(as1tree, trans1)} ∪ restrans, dict ")))
```

запись : [*Statename*$_0$] *Answer*$_0$* [*Sortqual*] → *Dict* → *Decision-answer*$_1$-**set** *Dict*

Цель Преобразовать ответы решения или выбор в AS_1

Параметры

tnm	См. *transform-act*
anslist	Список ответов, которые должны быть преобразованы
tp	Сорт ответов. Если *tp* равен **nil**, тогда запрос и все ответы состоят из символьных строк (то есть решение неформальное)

Результат Множество ответов AS_1

Алгоритм

Строка 1	Когда просмотрен список ответов, возвратить пустое множество (функция рекурсивна).
Строка 4	Пусть *vlist* и *trans* обозначают условия и переход первого ответа в списке.

Строки 5-10	Попытаться преобразовать условия. Если это не выполняется, тогда условия должны содержать неформальный текст, так как *vlist* уже был проверен на соответствие (в *get-answer-sort*).
Строка 11	Преобразовать переход первого ответа.
Строка 12	Преобразовать остальные ответы.
Строки 13-15	Если преобразование обрабатываемого списка условий не выполняется, тогда список условий должен обозначать неформальный текст.
Строка 15	Преобразовать условие как неформальный текст и возвратить результирующий ответ, соединенный с остальными ответами AS_1.
Строка 17	Иначе возвратить преобразованный ответ, соединенный с остальными ответами AS_1

3.7.8.5 Преобразование вариантов

transform-option(*tnm*, mk-*Option*$_0$(*quest*, *anslist*, *elsepart*), *nodes*$_1$)(*dict*) ≜ (3.7.8.5.1)

1 (**let** *qualset* = *all-visible-sorts*(*dict*) **in**
2 **let** (, *tpset*,) = *transform-expr*(*quest*, CONSTANT, *qualset*)(*dict*) **in**
3 **let** *tp* = *get-answer-sort*(*anslist*, *tpset*)(*dict*) **in**
4 **let** *trans* = *eval-option-trans*(*tp*, *quest*, *anslist*, *elsepart*)(*dict*) **in**
5 *transform-transition*(*tnm*, *trans*, *nodes*$_1$)(*dict*))

запись : [*Statename*$_0$] *Option*$_0$ *Graph-node*$_1$* → *Dict* → *Transition*$_1$ *Dict*

Цель Выполнить действие выбора и возвратить результирующий переход

Параметры

tnm, *nodes*$_1$	См. *transform-act*

Алгоритм

Строка 1	Пусть *qualset* обозначает *Quals* всех сортов, которые обозреваемы в этом месте.
Строка 2	Преобразовать запрос выбора. *tpset* обозначает сорты, которые допустимы для запроса.
Строка 3	Пусть *tp* обозначает сорт выбора. Он выбирается при рассмотрении ответов (так же, как делается в *transform-decision*).
Строка 4	Выбрать переход, который соответствует запросу.
Строка 5	Преобразовать выбранный переход

eval-option-trans(*tp*, *question*, *anslist*, *elsepart*)(*dict*) ≜ (3.7.8.5.2)

```
1   (if anslist = ⟨⟩ then
2     if elsepart = nil then
3       exit("§4.3.4: Ни один ответ выбора не соответствует запросу выбора")
4     else
5       s-Transition0(elsepart)
6   else
7   (let mk-Answer0(vlist, trans) = hd anslist in
8    if match-option-answer(tp, vlist, question)(dict) then
9      if (∃ans ∈ elems tl anslist)(match-option-answer(tp, s-Conditionlist0(ans), question)(dict)) then
10       exit("§4.3.4: Более чем один ответ выбора соответствует запросу")
11     else
12       trans
13   else
14     eval-option-trans(tp, question, tl anslist, elsepart)(dict)))
```

запись : *Sortqual* $Expr_0$ $Answer_0$* [$Elsepart_0$] → *Dict* → $Transition_0$

Цель Выбрать ответ из действия выбора, который соответствут запросу выбора

Параметры

tp	*Qual* сорта запросов
question	Необязательный запрос
anslist	Список ответов выбора
elsepart	Else часть выбора

Результат Переход, содержащийся в удовлетворяющем ответе

Алгоритм

Строки 1-5	Если никакой из ответов не удовлетворяет запросу, тогда else часть должна быть специфицирована, и, если это так, возвратить переход, содержащийся в else части.
Строка 7	Произвести декомпозицию первого (следующего) ответа в списке.
Строка 8	Если он удовлетворяет запросу, тогда
Строки 9-12	Должен быть только такой ответ, и, если это так, возвращается переход, содержащийся в ответе.
Строка 14	Иначе продолжить поиск удовлетворяющего ответа

match-option-answer(*tp*, *vlist*, *question*)(*dict*) ≜ (3.7.8.5.3)

```
1   (let as0operator = build-answer-operator(tp, vlist, question) in
2    eval-simple-expr(as0operator, "BOOLEAN")(dict))
```

запись : *Sortqual* $Conditionlist_0$ $Expr_0$ → *Dict* → *Bool*

Цель Проверка, что список условий выбора удовлетворяет запросу выбора

Параметры

tp	Сорт запроса и условия
vlist	Список условий
question	Запрос выбора

Результат True, если список условий удовлетворяет запросу

Алгоритм

Строка 1	Из запроса и списка условий построить программу булевской функции, которая выдает **true**, если список условий удовлетворяет запросу.
Строка 2	Выполнить программу

build-answer-operator(*tp*, *vlist*, *question*) ≜ (3.7.8.5.4)

```
1   (let mk-Condition0(op, const) = hd vlist in
2    let orop = mk-Qualop0(⟨⟩, mk-Quotedop0(OR)) in
3    let op' = if op = nil then EQ else op in
4    let as0op = if op' ∈ {NE, EQ, GL, LT, LE, GE} then
5                 mk-Operatorapp0(mk-Qualop0(tp, mk-Quotedop0(op')), ⟨question, const⟩)
6                else
7                 (let andop = mk-Qualop0(⟨⟩, mk-Quotedop0(AND)) in
8                  let lesseqop = mk-Qualop0(tp, mk-Quotedop0(LE)) in
9                  mk-Operatorapp0(andop, ⟨mk-Operatorapp0(lesseqop, ⟨op', question⟩),
10                                      mk-Operatorapp0(lesseqop, ⟨question, const⟩)⟩)⟩) in
11  if tl vlist = ⟨⟩
12    then as0op
13    else mk-Operatorapp0(orop, ⟨as0op,build-answer-operator(tp, tl vlist, question)⟩))
```

запись: *Sortqual Conditionlist*$_0$ *Expr*$_0$ → *Operatorapp*$_0$

Цель Построить булевскую функцию, которая выдает булевский литерал **true**, если ответ выбора удовлетворяет запросу выбора

Параметры

tp	Сорт запроса
vlist	Список условий, который содержится в ответе
question	Запрос выбора

Результат Построенная программа AS_0

Алгоритм

Строка 1	Произвести декомпозицию первого условия в списке условий.
Строка 2	Построить префиксное имя функции "OR".
Строка 3	Если нет функций или выражений, определенных как первый элемент в условии, тогда это то же, что и функция равенства (EQ).
Строка 4	Если первый элемент в *Condition*$_0$ является оператором отношения, тогда построить оператор tp op'(question, const), где tp является квалификатором оператора, op' является оператором отношения, а (question, const) - списком аргументов.
Строки 7-10	Если первый элемент в *Condition*$_0$ является выражением, тогда условие отражает диапазон, а построенный оператор является тогда "AND" (tp "<=" (op', question), tp "<=" (question, const)) где tp - квалификатор, op' - нижняя граница и const - верхняя граница.
Строка 11	Если это было последнее условие в списке, тогда возвратить построенную программу, иначе
Строка 13	Построить программу, которая производит "OR" результата из предыдущей пограммы с результатом программы, построенной из остального списка условий

3.8 Основные функции создания в AS_1

make-as$_1$*-identifier*(*q*)(*dict*) ≜ (3.8.1)

```
1   (let qual = make-new-qual(q)(dict) in
2    let nameq = get-sur(qual) in
3    let (, nm) = qual[len qual] in
4    let as1nameq = make-as1-qual(nameq) in
5    mk-Identifier1(as1nameq, name-to-name1(nm)))
```

запись : *Qual* → *Dict* → *Identifier*$_1$

Цель Из *Qual*(*q*) построить и возвратить соответствующий идентификатор AS_1

Алгоритм

Строка 1 Извлечь *Newqual* (см. определение *Newqual*) из дескриптора *Dict*, то есть вставить уникальные имена в *Qual*.

Строка 2 Пусть *nameq* обозначает его квалификатор (то есть *Qual* единицы подразделения окружения).

Строка 3 Извлечь именную часть объекта.

Строка 4 Преобразовать *Qual* в квалификатор AS_1.

Строка 5 Построить идентификатор AS_1, содержащий представление AS_1 имени, строки или инфиксной операции

make-as$_1$*-qual*(*qual*) ≜ (3.8.2)

```
1   if tl qual = ⟨⟩ then
2     ⟨⟩
3   else
4    (let(q, nm) = hd qual in
5     let as1nm = name-to-name1(nm) in
6     let as1q = cases q :
7                 (SYSTEM
8                   → mk-System-qualifier1(as1nm),
9                 BLOCK
10                  → mk-Block-qualifier1(as1nm),
11                PROCESS
12                  → mk-Process-qualifier1(as1nm),
13                PROCEDURE
14                  → mk-Procedure-qualifier1(as1nm),
15                TYPE
16                  → mk-Sort-qualifier1(as1nm),
17                SIGNAL
18                  → mk-Signal-qualifier1(as1nm),
19                SUBSTRUCTURE
20                  → mk-Block-substructure-qualifier1(as1nm)) in
21    let q' = make-as1-qual(tl qual) in
22    (as1 q ⌢ q'))
```

запись : *Qual* → *Qualifier*$_1$

Цель Построить и возвратить квалификатор AS_1 из *Qual* (*qual*)

Алгоритм

Строка 1 Когда просмотр закончен, возвратить пустой квалификатор (функция рекурсивна).

Строка 4 Пусть *q* и *nm* обозначают класс объекта и имя первого элемента в qual.

Строка 5	Пусть as_1nm обозначает представление имени в AS_1.
Строка 7	Если класс объектов - система, тогда построить элемент квалификатора системы AS_1.
Строка 9	Если класс объектов - блок, тогда построить элемент квалификатора блока AS_1.
Строка 11	Если класс объектов - процесс, тогда построить элемент квалификатора процесса AS_1.
Строка 13	Если класс объектов - процедура, тогда построить элемент квалификатора процедуры AS_1.
Строка 15	Если класс объектов - сорт, тогда построить элемент квалификатора сорта AS_1.
Строка 17	Если класс объектов - сигнал, тогда построить элемент квалификатора сигнала AS_1.
Строка 19	Если класс объектов - подструктура блока, тогда построить квалификатор подструктуры блока. Два других класса объектов сервисов и подструктуры канала удаляются из *Qual* в функцию *make-new-qual*.
Строки 21-22	Преобразовать остальные *Qual* в AS_1 и соединить его с этим элементом

make-new-qual(*qual*)(*dict*) ≜ (3.8.3)

```
1   (let level = dict(SCOPEUNIT) in
2    let qual ' = cases dict(qual) :
3                 (mk-TimerD(, q)
4                   → q,
5                 mk-ProcedureD(, q)
6                   → q,
7                 mk-SortD(,,, q)
8                   → q,
9                 mk-SyntypeD(, q,,)
10                  → q,
11                mk-VarD(,,,, q)
12                  → q,
13                mk-OperatorD(,, q,)
14                  → q,
15                mk-ChannelD(,,,, sub)
16                  → if sub = nil then qual else convert-channel-qual(sub, level),
17                mk-ChannelsubD(bqual)
18                  → bqual ⌢ ⟨qual[len qual]⟩,
19                mk-ServiceD(,,,,)
20                  → get-sur(qual),
21                T→ qual) in
22  let qualelem = qual '[len qual '] in
23  if tl qual ' = ⟨⟩ then
24    ⟨qualelem⟩
25   else
26    (let qualrest =
27         if is-OperatorD(dict(qual)) ∨ is-LiteralD(dict(qual))
28           then get-sur(get-sur(qual '))
29           else get-sur(qual ') in
30     make-new-qual(qualrest)(dict) ⌢ ⟨qualelem⟩))
```

запись : *Qual* → *Dict* → *Qual*

Цель Вставить уникальные имена в *Qual* объекта. Уникальные имена необходимы из-за сервисов и из-за перегрузки операции

Результат Новый уникальный *qual*

Алгоритм

Строка 1	Пусть *level* обозначает *Qual* единицы подразделения, определяющей объект
Строки 3-21	В дескрипторе объекта находится новый *Qual*. Только объекты, которые могут быть определены в сервисах, каналах и подструктуре канала, переименованы (каналы переименовываются, если они содержат подструктуру (*sub* ≠ nil) и подструктуры канала переименовываются в подструктуру блока). Для *Qual* сервиса возвращается *Qual* объемлющего процесса.
Строки 22-29	Взять *qual* единицы подразделения (*qualrest*), который определил объект, и переименовать этот *Qual*. Если старый *Qual* обозначает операцию или литерал, тогда в новом *Qual* нельзя использовать квалификатора элемента TYPE.
Строка 30	Возвратить переименованную единицу подразделения, соединенную с парой класс объекта и имя для объекта

convert-channel-qual(*sub*, *level*) ≜ (3.8.4)

```
1   (let (b1, c1, b2, c2) = sub in
2    if (∃q)(level = b1 ⇝ q) then
3      get-sur(b1) ⇝ ⟨(CHANNEL, c1)⟩
4    else
5      get-sur(b2) ⇝ ⟨(CHANNEL, c2)⟩)
```

запись : *Newchannels Qual* → *Qual*

Цель Каналы, которые содержат подструктуру, представляются двумя новыми каналами, ведущими в подструктуру, которая представляется подструктурой блока. Когда имя канала используется (в идентификаторе в множестве VIA), оно заменяется именем одного из новых каналов в зависимости от того, где канал используется. *convert-channel-qual* возвращает *Qual* такого нового канала. Заметьте, что не разрешается ссылаться на старое имя канала внутри подструктуры канала

Параметры

sub	Информация о двух новых каналах. Эта информация извлекается из дескриптора канала
level	Единица подразделения, где используется канал

Результат *Qual* нового канала

Алгоритм

Строка 1	Пусть *b*1 обозначает одну из конечных точек блока, *c*1 - новый канал, подключенный к этой конечной точке, *b*2 - другая конечная точка, c2 - новый канал, подсоединенный к этой точке.
Строки 2-5	Если *level* указывает единицу подразделения, которая содержится в *b*1, тогда возвратить *Qual* канала, подсоединенного к этому блоку, иначе возвратить *Qual* другой конечной точки

make-as$_1$*idset*(*idset*)(*dict*) ≜ (3.8.5)

```
1   {make-as1-identifier(id)(dict) | id ∈ idset}
```

запись : *Qual*-set → *Dict* → *Identifier*$_1$-set

Цель Из множества *Quals* построить соответствующее множество идентификаторов AS_1

make-as$_1$tree(node, name, as$_1$set, parm1, parm2, parm3)(dict) ≜ (3.8.6)

```
1   (let as1name = name-to-name1(name),
2        data = dict(DATATYPEDEF) in
3   if sorts-have-values(data)(dict) then
4     (let blocks = {elem | elem ∈ as1set ∧ is-Block-definition1(elem)},
5          channels = {elem | elem ∈ as1set ∧ is-Channel-definition1(elem)},
6          signals = {elem | elem ∈ as1set ∧ is-Signal-definition1(elem)},
7          sigroutes = {elem | elem ∈ as1set ∧ is-Signal-route-definition1(elem)},
8          syntypes = {elem | elem ∈ as1set ∧ is-Syn-type-definition1(elem)},
9          procedures = {elem | elem ∈ as1set ∧ is-Procedure-definition1(elem)},
10         processs = {elem | elem ∈ as1set ∧ is-Process-definition1(elem)},
11         views = {elem | elem ∈ as1set ∧ is-View-definition1(elem)},
12         timers = {elem | elem ∈ as1set ∧ is-Timer-definition1(elem)},
13         vars = {elem | elem ∈ as1set ∧ is-Variable-definition1(elem)} in
14      cases node:
15       (SYSTEM
16          → mk-System-definition1(as1name, blocks, channels, signals, data, syntypes),
17        BLOCK
18          → mk-Block-definition1(as1name, processs, signals, parm1, sigroutes, data, syntypes, parm2),
19        SUBSTRUCTURE
20          → mk-Block-substructure-definition1(as1name, blocks, parm1, channels, signals, data, syntypes),
21        PROCESS
22          → mk-Process-definition1(as1name, parm1, parm2, procedures, signals, data,
23                                   syntypes, vars, views, timers, parm3),
24        PROCEDURE
25          → mk-Procedure-definition1(as1name, parm1, procedures, data, syntypes, vars, parm2)))
26   else
27    exit("§5.2.1: Сорт в определении типа данных не имеет значений")
```

запись : *Quot* [*Name*$_0$] *Decl*$_1$-set [*Channel-to-route-connection*$_1$ | *Channel-connection*$_1$ | *Number-of-instances*$_1$ | *Procedure-formal-parameter*$_1$*] [*Process-formal-parameter*$_1$* | *Procedure-graph*$_1$ | *Block-substructure-definition*$_1$] [*Process-graph*$_1$] → *Dict* → *Decl*$_1$

Цель Построить представление AS$_1$ единицы подразделения, которая может быть системой, блоком, подструктурой блока, процессом или процедурой.

Параметры

node	Вид единицы подразделения, которая должна быть построена
name	Имя единицы подразделения
parm 1, parm2, parm3	Некоторые параметры, которые различаются в зависимости от вида единицы подразделения. Например, если единица подразделения - процесс, тогда *parm 1* является *Number-of-instances*$_1$.

Результат Определение AS$_1$ системы, блока, подструктуры блока, процесса или процедуры

Алгоритм

Строка 1	Извлечь представление имени в AS$_1$.
Строка 2	Извлечь *Data-type-definition*$_1$ единицы подразделения.
Строка 3	Каждый сорт в *Data-type-definition*$_1$ должен иметь по крайней мере одно значение.
Строки 4-13	Разбиение множества определений на множество определений блоков.
Строка 5	Множество определений каналов.
Строка 6	Множество определений сигналов.
Строка 7	Множество определений маршрутов сигналов.
Строка 8	Множество определений подтипов.

Строка 9	Множество определений процедур.
Строка 10	Множество определений процессов.
Строка 11	Множество определений обозреваний.
Строка 12	Множество определений таймеров.
Строка 13	Множество определений переменных.
Строки 15-16	Построить определение системы.
Строки 17-18	Построить определение блока.
Строки 19-20	Построить определение подструктуры блока.
Строки 21-23	Построить определение процесса.
Строки 24-25	Построить определение процедуры.

sorts-have-values(**mk**-*Data-type-definition*$_1$(,, *snmset*, *signatureset*,))(*dict*) ≜ (3.8.7)

```
1   let level = dict(SCOPEUNIT) in
2   (∀nm ∈ snmset)
3    ((∃sign ∈ signatureset)
4     ((let mk-Identifier1(qual, signm) = s-Result1(sign) in
5       level = qual ∧ nm = signm))))
```

запись : *Data-type-definition*$_1$ → *Dict* → *Bool*

Цель Проверить, что определение типа данных для единицы подразделения имеет значение для каждого сорта, то есть что существует по крайней мере один литерал сорта, либо одна операция, которые возвращают значение сорта.

Параметры Определение типа данных, содержащее

snmset	Множество сортов, определяемых определением типа данных
signatureset	Множество литералов и сигнатур операторов для определения типа данных

Результат True в случае успешного завершения

Алгоритм

Строка 2	Для всех имен сортов в множестве сортов, необходимо учитывать, что в множестве сигнатур существует такая сигнатура, что
Строки 4-5	Результирующий сорт определяется локально и его имя является таким же, как имя сорта (*nm*)

make-as$_1$*-conaxioms*(*qualset*)(*dict*) ≜ (3.8.8)

```
1   (if qualset = {} then
2     {}
3    else
4     (let qual ∈ qualset in
5      let(, nm) = qual[len qual] in
6      let sort = get-sur(qual) in
7      if is-Name0(nm) then
8        make-as1-conaxioms(qualset \ {qual}(dict)
9       else
10       (let mk-String0(str) = nm in
11        if len str ≤ 1 then
12          make-as1-conaxioms(qualset \ {qual})(dict)
13         else
14          (let ch = str[len str] in
15           let rest = mk-String0(⟨str[i] 1 ≤ i ≤ len str-1⟩) in
16           if sort ⇝ ⟨(LITERAL,rest)⟩ ∉ dom dict then
17             make-as1-conaxioms(qualset \ {qual})(dict)
18            else
19             (let concop = mk-Qualop0(sort, mk-Quotedop0(CONC)) in
20              let strexpr1 = mk-Stringterm0(⟨⟩ ,rest),
21                  strexpr2 = mk-Stringterm0(⟨⟩, mk-String0(⟨ch⟩)),
22                  strexpr3 = mk-Stringterm0(⟨⟩, nm) in
23              let equation =
24                  mk-Equation0(mk-Operatorterm0(concop, ⟨strexpr1, strexpr2⟩), strexpr3) in
25              let equation1 =
26                  (trap exit with {} in
27                    {transform-axiom(equation, AXIOMS)(dict)}) in
28              equation1 ∪ make-as1-conaxioms(qualset \ {qual}(dict))))))
```

запись : *Qual*-**set** → *Dict* → *Equations*$_1$

Цель Построить представление AS$_1$ неявных аксиом конкатенации, подключенных к символьным строковым литералам, например, 'abc' имеют неявные аксиомы:

'ab' // 'c' == 'abc';
'a' // 'b' == 'ab';

обеспечивающие, что равенства статически корректны.

Параметры Множество *Quals* литералов, определенных в определении частичного типа данных.

Результат Построенные равенства AS$_1$

Алгоритм

Строка 1	Когда просмотр закончен, возвратить пустое множество равенств (функция рекурсивна).
Строка 4	Рассмотреть следующий литерал.
Строка 5	Пусть *nm* обозначает имя литерала.
Строка 6	Пусть *sort* обозначает его сорт.
Строки 7-8	Если литерал представлен обычным именем, тогда продолжить с остальными литералами, иначе
Строка 10	Пусть *str* обозначает символы в строке.
Строки 11-12	Если число символов не больше единицы, тогда продолжить с остальными литералами.
Строка 14	Пусть *ch* обозначает последний символ в строке.
Строка 15	Пусть *rest* обозначает строку, содержащую все, кроме последнего символа.

Строки 16-17	Если строка, в которой был удален последний символ, не определена для сорта, тогда продолжить с остальными литералами.
Строка 19	Построить операцию '//'.
Строка 20	Построить строковый терм, содержащий осталыные символы и не содержащий квалификатор (*strexpr1*).
Строка 21	Построить строковый терм, содержащий последний символ и не содержащий квалификатор (*strexpr2*).
Строка 22	Построить строковый терм, содержащий строковый литерал (*strexpr3*).
Строка 23	Построить равенство : strexpr1 // strexpr2 == strexpr3
Строки 25-27	Попытаться преобразовать равенство в AS_1. Если это не выполняется, тогда равенства не является корректным и, следовательно, не является подразумеваемым.
Строка 28	Объединить равенство с равенствами, подразумеваемыми из остальных литералов.

as_0-*id*(*qual*) ≜ (3.8.9)

```
1   if qual = ENV then
2     ENV
3   else
4     (let (, nm) = qual[len qual] in
5      let qualifier = get-sur(qual) in
6      mk-Id0(qualifier, nm))
```

запись : (*Qual* | ENV) → (Id_0 | ENV)

Цель Восстановить идентификатор AS_0 из *Qual*. Эта функция используется при раскрытии сокращенных записей.

Параметры

qual *Qual*, который должен быть преобразован в идентификатор AS_0. Так как функция также применяется к конечным точкам каналов и маршрутов сигналов, то функция может иметь ENV в качестве аргумента.

Результат Построенный идентификатор AS_0

Алгоритм

Строка 1	Если он является ENV, тогда не изменять его; иначе
Строка 4	Извлечь часть имени.
Строка 5	Извлечь квалификатор, который является *Qual* единицы подразделения окружения.
Строка 6	Возвратить составленный идентификатор

make-implicit-vardecl(*dict*) ≜ (3.8.10)

```
1   (let(, f1nm, f2nm) = dict(GLOBALNAMES) in
2   let as0id = as0-id(get-predef-sort("INTEGER")(dict)) in
3   let init = mk-Id0(⟨⟩, mk-Name0("0", nil)) in
4   let formudecl = mk-Vardef0(nil, nil, ⟨mk-Vardefelem0(⟨f1nm, f2nm⟩, as0id, init)⟩) in
5   let closureset = dict(IMPLIED) in
6   let as0defs = formudecl ⌢ ⟨make-implicit-import-vardef(closure) | closure ∈ closureset⟩ in
7   transform-vardef(as0defs)(dict))
```

запись : *Dict* → *Variable-definition*$_1$-**set** *Dict*

Цель Построить определения переменных AS_1 для двух неявных переменных, используемых в разрешающем условии и в непрерывном сигнале, и построить определения переменных AS_1 для неявных переменных импорта в процессе

Алгоритм

Строка 1	Пусть *f1nm* и *f2nm* обозначают AS_0 имена двух неявных переменных, используемых в условии разрешения и непрерывном сигнале.
Строка 2	Пусть as_0id обозначает AS_0 идентификатор сорта INTEGER.
Строка 3	Построить выражение инициации. Переменные инициализируются в нуль.
Строка 4	Построить определения переменных AS_0 для двух переменных.
Строка 5	Извлечь множество пар имен переменных и сорта переменных, содержащих неявные переменные, используемые в преобразовании выражений импорта.
Строка 6	Пусть as_0defs обозначает все неявные определения.
Строка 7	Преобразовать определения в AS_1

make-implicit-import-vardef((*t*,*nm*)) ≜ (3.8.11)

1 (let $as_0tid = as_0\text{-}id(t)$ in

2 mk-$Vardef_0$(nil, nil, ⟨mk-$Vardefelem_0$(⟨*nm*⟩, as_0tid, nil)⟩))

запись : *NameclosureD* → $Vardef_0$

Цель	Построить определение переменной AS_0 для неявной переменной импорта
Параметры	Пара сорта переменной *t* и имени переменной (*nm*).

Строка 1	Построить идентификатор AS_0 сорта переменной.
Строка 2	Построить определение переменной AS_0. Переменная не инициализируется, так как ее первое использование всегда в вершине ввода

transform-import(*tnm*, *originalnode*$_1$, *actl*, *term*)(*dict*) ≜ (3.8.12)

```
1   (let importlist = dict(IMPORTLIST) in
2   let (impnm, qual, expr) = hd importlist,
3       statenm = create-unique-name() in
4   let(, (nm, )) = qual[len qual] in
5   let mk-ImportD(tq) = dict(qual) in
6   let mk-SignalnamesD(, xq, xr) = exportmap((tq, nm)) in
7   let lev = dict(SCOPEINUT) in
8   let osig = mk-Outputsig0(mk-Id0(⟨⟩, xq), ⟨⟩) in
9   let outputnode = mk-Actstmt0(nil, mk-Output0(⟨osig⟩, expr, ⟨⟩)) in
10  let sigset = all-input-signals(lev)(dict) \ {lev ↝ ⟨(SIGNAL, xr)⟩} in
11  let savenode = mk-Savespec0(⟨as0-id(sig) | sig ∈ sigset⟩),
12     statedict = dict(STATEDICT) in
13  let termstmt ' = Termstmt0(nil, mk-Nextstate0(mk-Id0(⟨⟩, statenm))) in
14  let outputtrans = mk-Transition0(outputnode, termstmt ') in
15  let impdict = [IMPLIED      ↦ (dict(IMPLIED) ∪ {(tg, impnm)}),
16                 IMPORTLIST   ↦ tl importlist] in
17  if tl importlist = ⟨⟩ then
18     (let inputtrans = mk-Transition0(actl, term) in
19      let inpvars = ⟨mk-Inputvars0(mk-Id0(⟨⟩, xr), ⟨mk-Id0(⟨⟩, impnm)⟩)⟩ in
20      let inputnode = mk-Inputspec0(inpvars, nil, inputtrans) in
21      let stated = mk-StateD(⟨(lev, inputnode), (lev, savenode)⟩, (tnm, originalnode1)) in
22      let statedict ' = statedict + [statenm ↦ stated] in
23      let dict ' = dict + [STATEDICT ↦ statedict '] in
24      (outputtrans, dict ' + impdict))
25   else
26     (let (termstmt ", dict ") = transform-import(tnm, originalnode1, actl, term)(dict + impdict) in
27      let inpvars = ⟨mk-Inputvars0(mk-Id0(⟨⟩, xr), ⟨mk-Id0(⟨⟩, impnm)⟩)⟩ in
28      let inputnode = mk-Inputspec0(inpvars, nil, termstmt ") in
29      let statedict ' =
30         statedict + [statenm ↦ mk-StateD(⟨(lev, inputnode), (lev, savenode)⟩, (tnm, nil))] in
31      let dict "' = dict " + [STATEDICT ↦ statedict '] in
32      (outputtrans, dict "')))
```

запись : $Statename_0\,(Graph\text{-}node_1 \mid Decision\text{-}node_1)\,Actstmt_0^{+}\,[Termstmt_0] \rightarrow Dict \rightarrow Transition_0\,Dict$

Цель Модифицировать *Statedict* в *Dict*, чтобы включить состояния импорта, подразумевасмые в выражении, и построить переход, ведущий в неявное состояние

Параметры

tnm Имя состояния, которому принадлежит действие, содержащее выражение. Имя хранится в дескрипторе *StateD* состояния импорта, такое, что в течении преобразования перехода, следующего за состоянием импорта, следующие состояние с черточкой может быть заменено

originalnode$_1$ Версия AS_1 действия, которое содержит выражение импорта. Эта вершина также хранится в состоянии импорта (два объекта формируют *Importstateinf*, определенный в *Quotdict*), так как она должна быть включена в переход AS_1, следующий за состоянием импорта

actl, term Действия AS_0 и терминатор, который сопровождает действие, содержащее выражение(я) импорта. Переход, который следует за вводом xtREPLY, содержит эти два объекта. Когда состояние импорта преобразуется, *originalnode*$_1$ добавляется в переход (см. *insert-importact*). Если действие содержит более одного выражения импорта, тогда переход, следующий за вводом xtREPLY для других состояний импорта, состоит из вывода сигнала xtQUERY и следующего состояния в следующем состоянии импорта.

Результат Переход AS_0, содержащий выходной сигнал xtQUERY, связанный с первым состоянием импорта, и *Dict*, измененный, чтобы включить дескрипторы для неявных состояний импорта

Алгоритм

Строка 1 Извлечь информацию о выражениях импорта, которые встречаются в выражении.

Строка 2 Извлечь информацию о первом выражении импорта. *impnm* обозначает имя неявной переменной, заменяющей выражение импорта, *qual* обозначает *Qual* переменной импорта, используемой в выражении импорта, *expr* обозначает выражение *pid* AS_0 из выражения импорта.

Строка 3 Построить имя для неявного состояния.

Строка 4 Извлечь имя переменной импорта.

Строка 5 Извлечь *Qual* результирующего сорта выражения импорта.

Строка 6 Произвести декомпозицию *SignalnamesD*, связанного с парой результирующий сорт - имя переменной импорта. *xq* - имя сигнала сигнала xtQUERY, *xr* - имя сигнала xtREPLY.

Строка 7 Пусть *lev* обозначает текущую единицу подразделения, которая является процессом, сервисом или процедурой.

Строки 8-9 Построить действие выхода, содержащее сигнал xtQUERY и выражение PiD.

Строка 10 Пусть *sigset* обозначает *Quals* сигналов, которые должны быть сохранены в неявном состоянии.

Строка 11 Построить AS_0 вершину сохранения.

Строки 13-14 Построить переход, содержащий выход сигнала xtQUERY, который завершается в неявном состоянии.

Строка 15 Добавить новую переменную импорта к множеству неявных переменных, содержащихся на входе IMPLIED в *Dict*, и удалить информацию о выражении импорта из списка, содержащегося на входе IMPORTLIST.

Строка 17 Если выражение импорта было последним (или единственным) в выражении, тогда

Строка 18 Построить переход, содержащий действия и терминатор, который сопровождает действие, содержащее выражение.

Строка 20 Построить ввод сигнала xtREPLY.

Строки 22-23 Добавить неявное состояние в *Statedict*. *lev* указывает, что вершина ввода и вершина сохранения будут преобразованы в том же контексте, что и та, для которой встретилось одно из выражений импорта (напомним, что состояния преобразуются после того, как были объединены сервисы).

Строка 24 Возвратить переход, содержащий выход, и возвратить измененный *Dict*.

Строка 26 Если есть еще выражения импорта, тогда создать переход выхода, ведущий к следующему состоянию импорта, и изменить *Dict*, чтобы включить оставшиеся состояния и новую неявную переменную.

Строка 28 Создать ввод, содержащий переход, ведущий к следующему состоянию импорта.

Строка 29-31 Изменить *Statedict*, чтобы включить новое неявное состояние. Дескриптор *Importstateinf* в дескрипторе *StateD* не должен содержать действие AS_1 ($originalnode_1$), так как это состояние не является последним состоянием, неявно заданным выражениями импорта в действии.

Строка 32 Возвратить переход, содержащий выход, и возвратить измененный *Dict*

3.9 Расширение сервисов

build-service-descriptor(mk-*Servicedef*$_0$(*id*, *sigset*, *dcll*, *body*, *tid*))(*dict*) ≜ (3.9.1)

```
1   (let mk-Id0(q, snm) = id in
2    let squal = dict(SCOPEUNIT) ⌢ ⟨(SERVICE, snm)⟩ in
3    let dict ' = dict + [SCOPEUNIT ↦ squal] in
4    if q ≠ ⟨⟩ then
5      exit("§2.4.1: Определяющие имена могут только быть квалифицированы в удаленных определениях")
6     else
7      if tid ∈ {id, nil} then
8        (let sigqualset = transform-validinputset(sigset)(dict ') in
9         let(as1dclset, dict ") = transform-decllist(dcll)(dict ') in
10        let(starttrans, labeldict, statedict, priinput) =
11            build-service-statedict(body)(dict) in
12        (as1dclset, dict "+[squal↦ mk-ServiceD(starttrans, labeldict, statedict, sigqualset, priinput)]))
13      else
14        exit("§2.2.2: Завершающее имя в определении сервиса отлично от определяющего имени")
```

запись : *Servisedef*$_0$ → *Dict* → *Decl*$_1$-set *Dict*

Цель Преобразовать определения, содержащиеся в определении сервиса, в AS$_1$ и создать дескриптор сервиса, который содержит граф сервиса в виде *Statedict* и *Labeldict*

Параметры Определение сервиса, содержащее

id	Идентификатор сервиса
sigset	Множество допустимых входных сигналов сервиса
dcll	Локальные определения
body	Граф сервиса
tid	Идентификатор, завершающий определение сервиса

Результат Определения AS$_1$ локальных определений и *Dict*, содержащий дескриптор сервиса и дескрипторы локальных определений

Алгоритм

Строки 2-3	Изменить SCOPEUNIT в *Dict*, чтобы обозначить сервис
Строки 4-7	Идентификатор сервиса не должен быть квалифицирован (строка 4), и, если конечный идентификатор специфицирован, тогда он должен быть тем же, что и идентификатор сервиса (строка 7)
Строка 8	Сигналы, которые могут быть получены на входе или приоритетном входе сервиса, есть сигналы, специфицированные в совокупности допустимых входных сигналов или специфицированные в маршруте сигналов сервиса, объединенных с таймерами, определенными в сервисе.
Строка 9	Преобразовать локальные определения.
Строка 10	Извлечь начальный переход (*starttrans*), построить *Statedict* (*statedict*) и *Labeldict* (*labeldict*) из графа сервиса и извлечь сигналы, которые имеют более высокий приоритет.
Строка 12	Возвратить версию AS$_1$ локальных определений и *Dict*, измененный, чтобы включить их дескрипторы и дескриптор сервиса .

build-service-statedict(**mk**-*Body*$_0$(*trans, statelist*))(*dict*) ≜ (3.9.2)

```
1   (let trans' = insert-trans-term(trans) in
2    let statelist ' = ⟨insert-state-term(statelist[i]) | 1 ≤ i ≤ len statelist⟩ in
3    let labeldict = build-trans-labeldict(trans ')([]) in
4    let labeldict ' = build-state-labeldict(statelist ')(labeldict) in
5    let statedict = build-statedict(statelist ', dict(SCOPEUNIT))([]) in
6    let statedict ' = remove-asterisk-input-and-save(statedict)(dict) in
7    let priinput = extract-priinput(statedict ')(dict) in
8    (trans', labeldict ', statedict ', priinput))
```

запись : *Body*$_0$ → *Dict* → *Transition*$_0$ *Labeldict Signalqual*-**set**

Цель Создать *Statedict* и *Labeldict* для сервиса

Параметры Тело сервиса, содержащее

trans Начальный переход
statelist Состояния сервиса

Результат Начальный переход, где были вставлены терминаторы в ответы решения, построенный *Labeldict*, построенный *Statedict* и *Quals* сигналов с высоким приоритетом.

Алгоритм

Строка 1 Включить терминаторы в начальный переход.
Строка 2 Включить терминаторы в состояния сервиса.
Строка 3 Построить *Labeldict* для начального перехода.
Строка 4 Расширить *Labeldict* для того, чтобы включить все тело сервиса.
Строка 5 Построить *Statedict*.
Строка 6 Удалить вводы с звездочкой и сохранение с звездочкой из состояний в *Statedict*.
Строка 7 Извлечь *Quals* сигналов с высоким приоритетом.
Строка 8 Возвратить начальный переход, *Labeldict*, *Statedict* и сигналы с высоким приоритетом.

extract-priinput(*statedict*)(*dict*) ≜ (3.9.3)

```
1   (if statedict = [] then
2      {}
3    else
4      (let stnm ∈ dom statedict in
5       let mk-StateD(speclist,) = statedict(stnm) in
6       let inputset = {inp | (, inp) ∈ elems speclist ∧ is-Priinput0(inp)} in
7       let inpvars = {inpv | (∃mk-Priinput0(inpl, ) ∈ inputset)(inpv ∈ elems inpl )} in
8       let priset = {signal-qual(id)(dict) |
9                     (∃mk-Inputvars0(id ', ) ∈ inpvars)(id ' = id)} in
10      priset ∪ extract-priinput(statedict \ {stnm})(dict)))
```

запись : *Statedict* → *Dict* → *Signalqual*-**set**

Цель Извлечь сигналы с высоким приоритетом из состояний сервиса путем инспектирования входных вершин

Параметры *Statedict*, содержащий состояния сервиса

Результат Сигналы сервиса с высоким приоритетом

Алгоритм

Строка 1 Когда закончен просмотр *statedict*, возвратить пустое множество (функция рекурсивна).

Строка 4	Взять некоторое состояние в *Statedict*.
Строка 5	Извлечь входные данные состояния.
Строка 6	Извлечь приоритетные входные вершины из состояния.
Строки 7-8	Извлечь сигналы с высоким приоритетом из входов с приоритетом.
Строка 10	Возвратить сигналы с высоким приоритетом вместе с сигналами с высоким приоритетом из оставшихся состояний

transform-decomposition-body(dict) ≜ (3.9.4)

```
1   (let servicelist = ⟨squal ∈ dom dict | is-local(squal)(dict) ∧
2                                  is-ServiceD(dict(squal))⟩ in
3   if servicelist = ⟨⟩ then
4     exit("§4.10: Декомпозиция должна содержать по крайней мере одно определение сервиса")
5    else
6     (let statetuplemap = merge-states(servicelist, ⟨⟩, ⟨⟩, 1)(dict) in
7      let statetuplenamemap = [statetuple ↦ create-unique-name() |
8                               statetuple ∈ dom statetuplemap] in
9      let statedict = [statetuplenamemap (tuple) ↦ (statetuplemap(tuple), nil, nil) |
10                     tuple ∈ dom statetuplemap] in
11     let statedict ' = double-states(statedict, servicelist)(dict) in
12     let statedict " = remove-cont-enable-from-statelist(dom statedict ')(dict, statedict') in
13     let dict ' = dict + [SERVICES   ↦ (statetuplenamemap, servicelist),
14                          STATEDICT  ↦ statedict "] in
15     let (firststate, firstservice) = extract-firststate-or-service (servicelist, ⟨⟩, nil)(dict) in
16     if firstservice = nil then
17       (let term1 = mk-Nextstate-node1(name-to-name1(firststate)) in
18        let trans1 = add-service-varinit(mk-Transition1(⟨⟩, term1), elems servicelist)(dict) in
19        let (statebody1, dict "") = transform-statelist({})(dict ') in
20        (mk-Process-graph1(mk-Process-start-node1(trans1), statebody1), dict ""))
21      else
22       (let mk-ServiceD(trans, labeldict ,,, ) = dict(firstservice) in
23        let dict " = dict ' + [LABELDICT  ↦ labeldict,
24                               SCOPEUNIT ↦ firstservice] in
25        let (trans1, dict "') = transform-transition(firststate, trans, ⟨⟩)(dict ") in
26        let trans1 ' = add-service-varinit(trans1, elems servicelist)(dict) in
27        let (statebody1, dict "") = transform-statelist({})(dict "') in
28        (mk-Process-graph1(mk-Process-start-node1(trans1 '), statebody1), dict ""))))
```

запись : *Dict* → *Process-graph*$_1$ *Dict*

Цель Преобразовать декомпозицию сервиса в граф AS$_1$ процесса

Параметры *Dict*, который содержит дескрипторы сервисов

Результат Граф AS$_1$ процесса и *Dict*, который содержит информацию о выходных сигналах, использованных в сервисе, и о неявных переменных (также, как для *transform-body*)

Алгоритм

Строки 1-2	Создать список *Quals* сервиса. Этот список является существенным в преобразовании сервиса. Позиции в списке соответствуют позициям в кортежах имен состояний, как также упомянуто в модели преобразования в § 4.10.2 Z.100.
Строка 3	Должен существовать по крайней мере один сервис в декомпозиции
Строка 6	Объединить состояния сервиса. Результатом является специальный *Statedict*, в котором входами являются кортежи имен состояний вместо имен состояний.
Строки 7-8	Создать отображение, которое определяет уникальное имя состояния для каждого кортежа имени состояния в специальном *Statedict*.

Строка 9	Создать обычный *Statedict*, где кортежи имен заменены соответствующими уникальными именами.
Строка 11	Создать новый *Statedict*, где каждое состояние разделено на два состояния: Состояние, принимающее приоритетные входные сигналы, и состояние, принимающее все другие входные сигналы. Второму из этих состояний дано уникальное имя.
Строка 12	Удалить условия разрешения и непрерывные сигналы из *Statedict* так же, как это делается для графа нормального процесса.
Строка 13	Составить список сервисов и отношений между кортежами имен состояний и уникальными именами состояний, доступных для функций преобразования состояний посредством включения информации в *Dict*. Также включить *Statedict* в *Dict*, как это делается для графа нормального процесса.
Строка 15	Извлечь *Quals* сервиса, который содержит начальные действия или извлечь первое состояние в случае, когда нет сервиса, который содержит начальный переход, т.е. либо *firststate* есть **nil** или *firstservice* есть **nil**.
Строка 16	Если нет сервиса, который содержит начальный переход, тогда
Строка 17	Построить следующее состояние AS_1 к первому состоянию.
Строка 18	Создать список действий AS_1, который инициализирует переменные сервиса и который прекращается следующим состоянием после первого состояния.
Строка 19	Преобразовать состояния.
Строка 20	Возвратить граф процесса, где начальный переход содержит инициализацию переменных и построенную вершину следующего состояния.
Строка 22	Если есть сервис, содержащий начальный переход, тогда извлечь его (*trans*). Также извлечь *Labeldict*, который будет использован в преобразовании начального перехода.
Строка 23	Сделать *Labeldict* и контекст, в котором начальный переход преобразован, доступными для функции преобразования путем включения их в *Dict*.
Строка 25	Преобразовать начальный переход.
Строка 26	Добавить список действий AS_1, который инициализирует переменные сервиса, к начальному переходу.
Строка 27	Преобразовать состояния.
Строка 28	Возвратить граф процесса

double-states(*statedict*, *servicedict*)(*dict*) ≜ (3.9.5)

```
1   (if statedict = {} then
2     {}
3    else
4     (let stnm ∈ dom statedict in
5      let mk-StateD(speclist, ) = statedict(stnm) in
6      let (priinput, norminput) be s.t. priinput ∪ norminput = elems speclist ∧
7          (∀(, inp) ∈ priinput)(is-Priinput0(inp)) ∧
8          (∀(, inp) ∈ norminput)(¬ is-Priinput0(inp)) in
9      if priinput = {} then
10       [stnm ↦ statedict(stnm)] + double-states(statedict \ {stnm}, servicelist)(dict)
11      else
12       (let saveset =
13           {(servicelist[i], mk-Savespec0(siglist)) | i ∈ ind servicelist ∧
14                                                 (let sigset =
15                                                     all-input-signals(servicelist[i])(dict) in
16                                                 siglist = ⟨as0-id(sig) | sig ∈ sigset ⟩)} in
17       let qual ∈ elems servicelist in
18       let newstnm = create-unique-name() in
19       let continput = (qual, mk-Contspec0(mk-Id0(⟨⟩, mk-Name0("TRUE", nil)), nil,
20                                   mk-Termstnmt0(nil, mk-Nextstate0(newstnm)))) in
21       let firstspec = ⟨spec | spec ∈ (priinput ∪ saveset)⟩ ⌢ ⟨(continput, nil)⟩,
22           secondspec = ⟨spec | spec ∈ norminput ⟩ in
23       let firststate = [stnm ↦ mk-StateD(firstspec, nil)],
24           secondstate = [newstnm ↦ mk-StateD(secondspec, (stnm, nil))] in
25       firststate + secondstate + double-states(statedict \ {stnm}, servicelist)(dict))))
```

запись : *Statedict Servicequal*$^{+}$ → *Dict* → *Statedict*

Цель Заменить каждое состояние в *Statedict* двумя состояниями, где первое состояние имеет сигналы с приоритетом в качестве входных, а второе состояние имеет другие сигналы в качестве входных.

Параметры

statedict (Остаток) старый *Statedict* (функция рекурсивна)

servicelist Список *Quals* сервисов (см. *transform-decomposition*)

Результат Расширенный *Statedict*

Алгоритм

Строка 1 Когда просмотр закончен, возвращается пустое отображение.

Строка 4 Взять следующее состояние для обработки.

Строка 5 Пусть *speclist* обозначает список входов.

Строки 6-8 Разделить входы на входы с приоритетом и другие вводы.

Строки 9-10 Если нет входов с приоритетом, тогда не изменять состояние.

Строки 12-16 Построить *Specs*, каждый из которых содержит сохранение всех нормальных сигналов. Для каждого сервиса существует один *Spec*.

Строка 17 Взять некоторый произвольный *Qual* сервиса из списка сервиса. Этот сервис используется в созданном *Spec* (строка 19), который содержит непрерывный сигнал во втором состоянии, то есть созданный непрерывный сигнал преобразуется в контексте произвольного сервиса, потому что непрерывный сигнал не использует идентификаторы, которые контекстно зависимы.

Строка 18 Создать уникальное имя для второго состояния.

Строка 19 Построить непрерывный сигнал, который ведет во второе состояние. Условие в непрерывном сигнале есть "true". Заметьте, что непрерывный сигнал не был явно задействован в модели сервисов

в Z.100. Однако, SAMETOKEN (см.§ 4.11 в Z.100) соответствует неявной переменной (названной "n" в § 4.12) и XCONT соответствует неявному сигналу emptyq.

Строка 21 — Создать *Speclist* для первого состояния. Он включает непрерывный сигнал.

Строка 22 — Создать *Statedict* для второго состояния.

Строка 23 — Создать требование *Statedict* для первого состояния.

Строка 24 — Создать требование *Statedict* для второго состояния.

Строка 25 — Продолжить для того, чтобы удвоить остальные состояния.

extract-firststate-or-service(*servicelist*, *statelist*, *firstservice*)(*dict*) ≜ (3.9.6)

```
1   (if servicelist = ⟨⟩ then
2      if firstservice ≠ nil then
3        (nil, firstservice)
4       else
5        (let(statetuplenamemap, ) = dict(SERVICES) in
6         (statetuplenamemap(statelist), firstservice))
7      else
8       (let mk-ServiceD(mk-Transition0(actl, term),, stdict,,) = dict(hd servicelist) in
9        if actl = ⟨⟩ ∧ is-Nextstate0(s-Terminator0(term)) then
10         (let mk-Termstmt0(, mk-Nextstate0(stnm)) = term in
11          if stnm ∈ dom stdict then
12            extract-firststate-or-service(tl servicelist, statelist ⌢ ⟨stnm⟩, firstservice)(dict)
13           else
14            exit("§2.6.7.2.1: Имя в следующем состоянии должно обозначать определенное состояние"))
15        else
16         if firstservice ≠ nil then
17           exit("§4.10.2 : Более чем один сервис содержит строку начального перехода")
18          else
19           extract-firststate-or-service(tl servicelist, nil, hd servicelist)(dict)))
```

запись : *Servicequal** [*Statename*$_0$*] [*Servicequal*] → *Dict* → [*Statename*$_0$] [*Servicequal*]

Цель — Извлечь либо начальное состояние составных состояний в *Statedict*, либо сервис, который содержит начальный переход.

Параметры

servicelist — Список сервисов (см.*transform-decomposition*)

statelist — Кортеж имен состояний, который построен при рекурсивном вызове функции. Когда все сервисы в *servicelist* были рассмотрены, кортеж имен определяет уникальное имя состояния, которое должно быть возвращено (через отображение кортежа состояний, см.*transform-decomposition*).

firstservice — Если сервис, содержащий начальный переход, был найден, тогда *firstservice* содержит *Quals* сервиса. Как только такой сервис найден, рекурсия может остановиться и возвращается сервис, но ради проверки, что есть только один такой сервис, сервис берется как параметр для следующего рекурсивного обращения

Результат — Если существует сервис, который содержит начальный переход, тогда *Quals* этого сервиса (*Servicequal*), иначе имя уникального начального состояния

Алгоритм

Строки 1-2 — Когда рассмотрены все сервисы, тогда, если сервис, содержащий начальный переход, был найден, тогда возвратить *Qual* этого сервиса (*firstservice*), иначе

Строки 5-6 — начальное состояние есть состояние, которое соответствует начальным следующим состояниям в сервисах (*statelist* содержит эти начальные состояния сервисов).

Строка 8	Произвести декомпозицию дескриптора следующего сервиса, который должен быть рассмотрен.
Строка 9	Если начало перехода содержит только следующее состояние, тогда
Строка 10	Пусть *stnm* обозначает имя состояния в следующем состоянии.
Строка 11	Если состояние определено в сервисе (т.е., если оно может быть найдено в *Statedict*, локальном по отношению к сервису), тогда
Строка 12	Продолжить, принимая во внимание следующий сервис. Присоединить имя состояния в список имен состояний так, что окончательно уникальное состояние может быть найдено, если ни один из сервисов не содержат строку начального перехода (строка 6).
Строки 16-19	Если сервис, содержащий строку начального перехода, еще не найден, тогда рассмотреть следующий сервис. Как только сервис, содержащий начальный переход, найден, список имен состояний не используется (т.е. это nil)

add-service-varinit(*trans*, *squalset*)(*dict*) ≜ (3.9.7)

```
1    (if squalset = {} then
2       add-varinit(trans, dict)
3     else
4      (let qual ∈ squalset in
5       let dict' = dict + [SCOPEUNIT ↦ qual] in
6       let trans' = add-varinit(trans, dict') in
7       add-service-varinit(trans', squalset \ {qual})(dict)))
```

запись : $Transition_1\ Servicequal\text{-}set \rightarrow Dict \rightarrow Transition_1$

Цель Присоединить AS_1 инициализацию переменных сервиса к начальному переходу.

Параметры

trans	Начальный переход
squalset	*Quals* сервисов декомпозиции сервиса

Результат Измененный переход

Алгоритм

Строка 1	Когда просмотрены все сервисы, тогда присоединить инициализацию переменных, определенных на уровне процесса, к переходу.
Строка 4	Пусть *qual* обозначает один из (оставшихся) сервисов.
Строка 5	Пусть SCOPEUNIT в *Dict* обозначает этот сервис.
Строка 6	Пусть *trans'* обозначает переход, изменяющий инициализацию переменных для этого сервиса. Функция *add-varinit* также используется, когда построение начального перехода присоединяется к процессу или процедуре.
Строка 7	Добавить инициализации для оставшихся сервисов

merge-states(*servicelist*, *statenmtuple*, *statelist*, *index*)(*dict*) ≜ (3.9.8)

```
1    (if index > len servicelist then
2       [statenmtuple ↦ mk-StateD(statelist, nil)]
3     else
4      (let mk-ServiceD(,, statedict,, ) = dict(servicelist[index]) in
5       merge {merge-states(servicelist,
6                           statenmtuple ⁀ ⟨statenm⟩,
7                           statelist ⁀ s-Speclist(statedict(statenm)),
8                           index + 1)(dict) | statenm ∈ dom statedict}))
```

запись : $Servicequal^{+}\ Statename_0^{*}\ Speclist\ N_1 \rightarrow Dict \rightarrow (Statename_0^{+} \rightarrow StateD)$

Цель Объединить ("STATE EXPLODE") *Statedicts* сервисов в *Statedict* как в отображение, где входы есть кортежи имен состояний (тот же вид кортежей имен состояний, что определен в части модели §4.10.2 Z.100). Функция *transform-decomposition-body*, где эта функция применяется, представляет уникальное имя состояния для каждого из кортежей и посредством этого превращает отображение в обычный *Statedict*

Параметры

servicelist Список *Servicequals* сервисов. Позиции сервисов в этом списке соответствуют позициям состояний в кортежах имен состояний

statenmtuple, statelist Кортеж имен состояний (*statetuple*) и входы (*statelist*) для результирующего состояния создаются рекурсивно. Когда *merge-states* применяется в первый раз, они пусты.

index Индекс для *servicelist*, который указывает, какой сервис был обработан на этом уровне рекурсии.

Алгоритм

Строка 1 Если имеется построенный (полный) кортеж имен состояния (*statenmtuple*), тогда возвратить требование для конкретного кортежа.

Строка 4 Пусть *statedict* обозначает *Statedict* сервиса, соответствующего индексу.

Строки 5-9 Для сервиса составить отображение, которое включает все кортежи состояний, которые имеют *statenmtuple* в качестве первых элементов (составленные из сервисов, которые уже обработаны) и которые имеют состояние (*statenm*) из сервиса в качестве следующего элемента. Требования различных состояний сервиса объединяются в одно отображение посредством операции **merge**

is-wf-servicesignals(*dict*) ≜ (3.9.9)

```
1   (let processsignals =
2       {squal | (∃mk-SignalrouteD(p1, p2, s1, s2) ∈ rng dict)
3              ((p1 = dict(SCOPEUNIT) ∧ squal ∈ s2) ∨ (p2 = dict(SCOPEUNIT) ∧ squal ∈ s1))} in
4   (∃d1, d2 ∈ dom dict)(is-ServiceD(dict(d1)) ∧ is-ServiceD(dict(d2)) ∧
5                 get-sur(d1) = dict(SCOPEUNIT) ∧ get-sur(d2) = dict(SCOPEUNIT) ∧
6                 s-Validinputset(dict(d1)) ∩ s-Validinputset(dict(d2)) ≠ {} ⊃
7                 (d1 = d2 ∧ s-Priinputset(dict(d1)) ∩ processsignals = {})))
```

запись : *Dict* → *Bool*

Цель Проверить, что входные сигналы сервисов не пересекаются и что нет внешних приоритетных сигналов к объемлющему процессу.

Результат True, если сигналы корректны

Алгоритм

Строки 1-3 Пусть *processsiggnals* обозначает сигналы, содержащиеся в маршруте сигнала (неявном или явном).

Строка 4 Для каждых двух *Quals d1* и *d2* объектов, определенных в процессе, должно сохраняться то, что, если они обозначают сервисы (строка 4), определены в окружающем процессе (строка 5) и имеют общие входные сигналы (строка 6), тогда они должны обозначать тот же сервис, и этот сервис не должен иметь сигналы с высоким приоритетом, ведущие к среде окружающего процесса (строка 7).

transform-servicesigroutedef(mk-*Sigroutedef*$_0$(*nm*, *path*, *opath*))(*dict*) ≜ (3.9.10)

```
1   (let mk-Sigroutepath0(endpoint1, endpoint2, siglist) = path in
2   let sigroutequal = dict(SCOPEUNIT) ⇀ ⟨(SIGNALROUTE, nm)⟩ in
3   let sigset = transform-signallist(siglist)(dict) in
4   let s1 = get-visible-qual(endpoint1, SERVICE)(dict),
5       s2 = get-visible-qual(endpoint2, SERVICE)(dict) in
6   let osigset =
7       if opath = nil then
8         {}
9        else
10        (let mk-Sigroutepath0(orig', dest', osiglist) = opath in
11         let s1' = get-visible-qual(orig', SERVICE)(dict),
12             s2' = get-visible-qual(dest', SERVICE)(dict) in
13         if s1 = s2' ∧ s2 = s1' then
14           transform-signallist(osiglist)(dict)
15          else
16           exit ("§4.10.1:Конечные точки в направлениях двунаправленного маршрута сигналов должны совпадать")) in
17  if s1 ≠ s2 then
18    (let descr = mk-SignalrouteD(s1, s2, sigset, osigset) in
19     ({}, [sigroutequal ↦ descr]))
20   else
21    exit("§4.10.1: Конечные точки маршрута сигналов сервиса должны быть различны"))
```

запись : *Sigroutedef*$_0$ → *Dict* → *Decl*$_1$-**set** *Dict*

Цель Создать дескриптор *Dict* маршрута сигналов сервиса

Параметры Маршрут сигналов сервиса содержит

nm	Имя маршрута сигнала
path	Первый маршрут
opath	Второй необязательный маршрут

Результат Пустой набор определений AS_1 (потому что все функции, которые преобразуют определения AS_1, возвращают определения AS_1) и требование *Dict*, содержащее дескриптор

Алгоритм

Строка 1	Произвести декомпозицию первого маршрута в маршруте сигналов сервиса.
Строка 2	Построить *Qual*, который обозначает маршрут сигналов сервиса.
Строка 3	Построить *Quals* для сигналов в первом маршруте.
Строки 4-5	Построить *Qual* конечных точек.
Строки 6-16	Если второй маршрут опущен, тогда *osigset* пустой, иначе *osigset* обозначает сигналы во втором маршруте (противоположные направления, как *sigset*).
Строка 10	Произвести декомпозицию второго маршрута.
Строки 11-12	Построить *Qual* конечных точек во втором маршруте
Строки 12-13	Первая конечная точка первого маршрута должна быть равна второй конечной точке второго маршрута, вторая конечная точка первого маршрута должна быть равна первой конечной точке второго маршрута
Строка 17	Один и тот же сервис не должен быть специфицирован в обоих конечных точках.
Строка 18	Построить дескриптор маршрута сигналов сервиса

3.10 Создание неявных каналов и маршрутов сигналов

Этот раздел содержит функции, которые используются для создания блока :

- Неявные каналы, которые соединены с блоком, если он содержит экспортирующий процесс (для импортирующих процессов есть другая конечная точка блока, содержащая экспортирующие процессы, которые создают каналы)

- Неявные подканалы, которые соединены с блоком, если он содержит импортирующие или экспортирующие процессы

- Определение соединений, если блок содержит подструктуру, которая импортирует или экспортирует в его окружающую среду

- Маршруты неявных сигналов и их ассоциированные определения соединения задаются в import/export и задаются совокупностями допустимых входных сигналов в случае, когда нет маршрутов сигналов, специфицированных в AS_0

- Дескрипторы *ExpimpchanD* должны содержатся в дескрипторе *BlockD*. Эти дескрипторы содержат информацию о том, как блок должен будет взаимодействовать, так чтобы неявные подканалы могли быть созданы для другой конечной точки блока

Функция входа, которая применяется в *transform-block*, есть *implicit-channels-and-signalroutes*.

implicit-channels-and-signalroutes(*decllist*)(*dict*) ≜ (3.10.1)

```
1   (let (exportset, importset) = imp-exp-in-processes(dict) in
2   let sigroutes = implicit-signalroutes(decllist)(dict),
3       blockimport = all-importeurs(dict),
4       blockexpoteurs = all-exporteurs(dict),
5       (fatherimpm, fatherexpm) = father-block-import-export(dict) in
6   let exporttoenv = {closeure | closure ∈ (exportset ∩ dom fatherexpm)},
7       importfromblock = {closure | closure ∈ (importset ∩ dom blockexporteurs)},
8       importfromenv = {closure | closure ∈ (importset ∩ dom fatherimpm)} in
9   let importblockmap = [closure ↦ blockexporteurs(closure) | closure ∈ importfromblock],
10      importenvmap = [cl ↦ ⟨(ENV, create-unique-name()) | 1 ≤ i ≤ len fatherimpm⟩ |
11                      cl ∈ importfromenv],
12      exportblockmap = [cl ↦ ⟨(bqual, create-unique-name()) | cl ∈ (exportset ∩ blockimport(bqual))⟩ |
13                        bqual ∈ dom blockimport],
14      exportenvmap = [cl ↦ ⟨(ENV, create-unique-name()) | 1 ≤ i ≤ len fatherexpm⟩ |
15                      cl ∈ exporttoenv] in
16  let importenvchannels = as0-channels(importenvmap, false)(dict),
17      exportblockchannels = as0-channels(exportblockmap, true)(dict),
18      exportenvchannels = as0-channels(exportenvmap, true)(dict) in
19  let connects = implicit-connects(importblockmap, exportblockmap)(dict) in
20  let decll = importenvchannels ⌢ exportblockchannels ⌢ exportenvchannels in
21  (decll, sigroutes, connects, importblockmap, exportblockmap))
```

запись : $Decl_0^* \rightarrow Dict \rightarrow Decl_0^*\ Decl_0^*\ Decl_0^*\ ExpimpchanD\ ExpimpchanD$

Цель Создать неявные каналы, определения соединения и маршруты сигналов для блока

Параметры

decllist Список определений блока. Используется для проверки наличия явных маршрутов сигналов в блоке

Результат Каналы, которые должны быть определены на том же уровне, что и блок, маршруты сигналов и соединения маршрутов сигналов, которые должны быть определены в блоке, соединения подструктуры блока, которые должны быть определены в подструктуре (если есть), отношение между именами неявных каналов и передаваемыми импортируемыми сигналами (имя канала выведено из *ExpimpchanD*, определенного для экспортирующего блока) и отношения между именами неявных каналов и экспортируемыми сигналами

Строка 1	Пусть *exportset* обозначает множество пар *Qual* сорта и экспортируемых имен (*NameclosureDs*) для процессов в блоке и пусть *importset* обозначает *NameclosureDs*, соответствующий тому, что экспортируется в блок.
Строка 2	Пусть *sigroutes* обозначает маршруты неявных сигналов, созданных для блока.
Строка 3	Пусть *blockimport* обозначает отношение между *Quals* блока и *NameclosureDs* для импортируемых в блок имен.
Строка 4	Пусть *blockexporteurs* обозначает *ExpimpchanD*, содержащий экспортируемые имена во всех других блоках и их именах.
Строка 5	Пусть *fatherimpm* обозначает дескриптор *ExpimpchanD* импорта для блока, который содержит этот блок, и пусть *fatherexpm* обозначает дескриптор *ExpimpchanD* экспорта для блока, который содержит этот блок
Строка 6	*NameclosureDs* для имен, которые экспортируются из этого блока в среду единицы подразделения, есть пересечение между *NameclosureDs* для имен, экспортируемых из блока (*exportset*), и *NameclosureDs* для имен, также экспортируемых в среду блока
Строка 7	*NameclosureDs* для имен, которые импортируются в этот блок из другого блока, есть пересечение между *NameclosureDs* для имен, импортируемых из этого блока (*importset*), и *NameclosureDs* для имен, экспортируемых в другой блок
Строка 8	*NameclosureDs* для имен, которые импортируются в этот блок из среды единицы подразделения, есть пересечение между *NameclosureDs* для имен, импортируемых из этого блока (*importset*), и *NameclosureDs* для имен, экспортируемых в окружающий блок.
Строка 9	Построить дескриптор *ExpimpchanD*, который содержит информацию о именах каналов, передающих имена сигналов импорта. Имена каналов получаются из дескриптора, содержащегося в *BlockD* для экспортирующего блока.
Строки 10-11	Построить дескриптор *ExpimpchanD*, который содержит информацию о именах подканалов, передающих имена сигналов импорта в среду блока. В этом случае имена каналов созданы, так как другая конечная точка есть ENV.
Строка 12-12	Построить дескриптор *ExpimpchanD*, который содержит информацию о именах каналов, передающих имена сигналов экспорта другим блокам (в той же единице подразделения).
Строки 14-15	Построить дескриптор *ExpimpchanD*, который содержит информацию о именах подканалов, передающих имена сигналов экспорта среде единицы подразделения.
Строка 16	Построить подканалы AS_0 для импортируемых имен в блок.
Строка 17	Построить каналы AS_0 для имен, которые экспортируются в другой блок
Строка 18	Построить каналы AS_0 для имен, которые экспортируются в окружающую единицу подразделения.
Строка 19	Построить определения соединения для всех неявных подканалов в подструктуре блока.
Строка 20	Пусть *decll* обозначает определения, которые должны быть определены на том же уровне, где определен блок, т.е. все построенные каналы.
Строка 21	Возвратить определения каналов, определения маршрута сигналов и два отображения *ExpimpchanD*

3.10.1 Создание неявных каналов

as_0-channels(map, isexport)(dict) ≜ (3.10.1.1)

```
1   (if map = [] then
2      ⟨⟩
3    else
4      (let clos ∈ dom map in
5      ⟨as0-channeldefs(map(clos), isexport, dict(clos))(dict)⟩⌢
6      as0-channels(map \ {clos}, isexport)(dict)))
```

запись : *ExpimpchanD Bool → Dict → $Chandef_0^*$*

Цель Построить неявные каналы, связанные либо с импортом из среды блока, либо с экспортом в среду блока, или экспортом из блока в другой блок

Параметры

map *ExpimpchanD*, содержащий информацию об имени канала и имя одной из конечных точек. Вторая конечная точка есть текущий блок, т.е. построенный из *dict*(SCOPEUNIT)

isexport True, если каналы для имен экспорта в текущем блоке должны быть созданы

Результат Построенный список определений канала AS_0

Алгоритм

Строка 1 Когда просмотрены все импортируемые или экспортируемые имена, тогда возвратить пустой список.

Строка 4 Взять следующий *NameclosureD* для импортируемого или экспортируемого имени.

Строка 5 Построить определение канала для импортируемого или экспортируемого имени.

Строка 6 Построить определения канала для остальных *NameclosureDs*

as_0-channeldefs(list, isexport, signalnames)(dict) ≜ (3.10.1.2)

```
1   (if list = ⟨⟩ then
2      ⟨⟩
3    else
4      (let level = dict(SCOPEUNIT)
5          blkid = as0-id(level)
6           mk-SignalnamesD(, xq, xr) = signalnames in
7      let signallist1 = ⟨mk-Id0(⟨⟩, xr)⟩,
8          signallist2 = ⟨mk-Id0(⟨⟩, xq)⟩,
9          (bqual, chan) = hd list in
10     let blkid ' = as0-id(bqual) in
11     let (fromblock, toblock) = if isexport then (blkid, blkid ') else (blkid ', blkid) in
12     let direction1 = mk-Chanpath0(fromblock, toblock, signallist1),
13         direction2 = mk-Chanpath0(fromblock, toblock, signallist2) in
14     ⟨mk-Chandef0(chan, direction1, direction2, nil, chan)⟩⌢
15     as0-channeldefs(tl list, isexport, signalnames)(dict)))
```

запись : *(Otherend $Channame_0$)* Bool SignalnamesD → Dict → $Chandef_0^*$*

Цель Для данного *SignalnamesD*, двух конечных точек канала и имен каналов построены определения каналов, передающих сигнал xtREPLY и сигнал xtQUERY

Параметры

list	Список пар, каждая из которых состоит из конечной точки, отличной от текущей, и имени канала. Список пар определяет каналы, которые должны быть созданы
isexport	True, если текущий блок есть экспортирующий блок
sugnalnames	Дескриптор *SignalnamesD*, содержащий сигналы xtREPLY и xtQUERY, соответствующие некоторому *NameclosureD*

Результат Список определений каналов AS_0

Алгоритм

Строка 1	Когда просмотр завершен, возвратить пустой список (функция рекурсивна).
Строки 4-14	Построить определение канала для следующего элемента в списке.
Строка 4	Пусть *level* обозначает *Qual* текущего блока.
Строка 5	Пусть *blkid* обозначает идентификатор текущего блока.
Строка 6	Пусть *xq* и *xr* обозначают сигнал xtQUERY и сигнал xtREPLY, которые должны быть переданы каналом.
Строки 7-8	Построить два списка сигналов, которые содержат эти сигналы.
Строка 9	Пусть *bqual* обозначает *Qual* блока другого конца и пусть *chan* обозначает имя канала.
Строка 10	Построить идентификатор другого конца.
Строка 11-11	Если текущий блок является экспортером, тогда *fromblock* обозначает этот блок, иначе *fromblock* обозначает другую конечную точку (в случае подканала).
Строки 12-13	Построить пути для двух списков сигналов.
Строка 14	Построить определение канала.
Строка 15	Построить определения канала для остальных в списке (*list*)

father-block-import-export(*dict*) ≜ (3.10.1.3)

```
1   (let level = dict(SCOPEUNIT) in
2    if len level > 2 then
3      (let fatherlevel = get-sur(get-sur(level)) in
4       let mk-BlockD(exp, imp,, ) = dict(fatherlevel) in
5       (exp, imp))
6     else
7      ([], []))
```

запись : *Dict* → *ExpimpchanD ExpimpchanD*

Цель Извлечь два *ExpimpchanDs* для блока, который содержит текущий блок

Результат *ExpimpchanD* для каналов экспорта и *ExpimpchanD* для каналов импорта

Алгоритм

Строка 1	Пусть *level* обозначает *Qual* текущего блока.
Строка 2	Если блок содержится в подструктуре, тогда
Строка 3	Пусть *fatherlevel* обозначает *Qual* блока, содержащего подструктуру.
Строки 4-5	Возвратить *ExpimpchanDs* для этого блока.
Строка 8	Если текущий блок определен на системном уровне, тогда отображения пусты, так как не возможно осуществлять импорт/экспорт из/в среду системы

imp-exp-in-processes(dict) ≜ (3.10.1.4)

```
1   (let level = dict(SCOPEUNIT) in
2   let pquals = {qual | qual ∈ dom dict ∧
3                        is-ProcessD(dict(qual)) ∧
4                        get-sur(qual) = level} in
5   if pquals = {} then
6     (let subblockquals = {qual | qual ∈ dom dict ∧
7                                  is-BlockD(dict(qual)) ∧
8                                  get-sur(get-sur(qual)) = level} in
9     let eiset = {imp-exp-in-processes(dict + [SCOPEUNIT ↦ qual]) | qual ∈ subblockquals} in
10     (union {imp | (imp, ) ∈ eiset}, union {exp | (, exp) ∈ eiset}))
11    else
12     (let eset = {vqual | vqual ∈ dom dict ∧
13                        is-VarD(dict(vqual)) ∧
14                        get-sur(vqual) ∈ pquals},
15          iset = {vqual | vqual ∈ dom dict ∧ is-Import(dict(vqual)) ∧
16                        get-sur(vqual) ∈ pquals} in
17      let expset =
18         {closure | (∃q ∈ eset)
19                  ((let mk-VarD(t,, exp,, ) = dict(q),
20                        (,vnm) = q[len q] in
21                     exp = EXPORTED ∧ closure = (t, vnm)))},
22          impset =
23          {closure | (∃q ∈ iset)
24                  ((let mk-ImportD(t) = dict(q),
25                        (,vnm) = q[len q] in
26                     closure = (t, vnm)))} in
27      (expset, impset)))
```

запись : *Dict* → *NameclosureD*-**set** *NameclosureD*-**set**

Цель Извлечь пары сорт-имена переменных (*NameclosureDs*) для переменных экспорта и для переменных импорта, определенных в процессах в блоке или, если блок не содержит процессы, тогда для процессов в содержащейся подструктуре.

Результат *NameclosureDs* для экспортируемых переменных и *NameclosureDs* для импортируемых переменных

Алгоритм

Строка 1	Пусть *level* обозначает *Qual* текущего блока.
Строки 2-4	Пусть *pquals* обозначает *Quals* процессов, определенных в блоке.
Строка 5	Если в блоке нет процессов, тогда
Строки 6-8	Пусть *subblockquals* обозначает *Quals* блоков, содержащихся в подструктуре.
Строка 9	Извлечь множество пар *NameclosureD* для экспортируемых переменных и для импортируемых переменных для подблоков.
Строка 10	Присоединить все экспортируемые *NameclosureDs* к этому множеству и присоединить все импортируемые *NameclosureDs* к этому множеству.
Строки 12-14	Если блок содержит процессы, тогда пусть *eset* обозначает множество *Quals* переменных для переменных, определенных в процессах.
Строки 15-16	Пусть *iset* обозначает множество *Quals* переменных для переменных импорта, определенных в процессах.
Строки 17-21	Пусть *expset* обозначает *NameclosureDs* для экспортируемых переменных, т.е. множество пар сорта (*t*) и имени экспортируемой (*exp* = EXPORTED) переменной (*vnm*).
Строки 22-26	Пусть *impset* обозначает N*ameclosureDs* для импортируемых переменных.

Строка 27 Возвратить два множества

all-importeurs(*dict*) ≜ (3.10.1.5)

```
1   (let level = dict(SCOPEUNIT) in
2   let sur = get-sur(level) in
3   let blockset = {qual | qual ∈ dom dict ∧
4                          is-BlockD(dict(qual)) ∧
5                          get-sur(qual) = sur} \ {level} in
6   let expimpmap = [q ↦ imp-exp-in-processes(dict + [SCOPEUNIT ↦ q]) | q ∈ blockset] in
7   [q ↦ closset | q ∈ dom expimpmap ∧ (, closset) = expimpmap(q)])
```

запись : *Dict* → (*Blockqual* ⇸ *NameclosureD*-set)

Цель Извлечь информацию о отношении между *NameclosureDs* для импорта и импортирующего блока. Текущий блок исключается в отношении (отображение)

Алгоритм

Строка 1	Пусть *level* обозначает текущий блок.
Строка 2	Пусть *sur* обозначает *Qual* подструктуры, окружающей блок.
Строки 3-5	Построить множество, состоящее из всех блоков в подструктуре, исключая текущий блок.
Строка 6	Создать отображение из импортирующих блоков в пары множеств *NameclosureDs* для блока.
Строка 7	Возвратить отображение из импортирующих блоков в *NameclosureD* для импортируемых переменных в блоках

all-exporteurs(*dict*) ≜ (3.10.1.6)

```
1   (let level = dict(SCOPEUNIT) in
2   let sur = get-sur(level) in
3   let blockset = {qual | qual ∈ dom dict ∧
4                          is-BlockD(dict(qual)) ∧
5                          get-sur(qual) = sur} \ {level} in
6   get-exporteurs(blockset)(dict))
```

запись : *Dict* → *ExpimpchanD*

Цель Построить (составной) *ExpimpchanD*, который включает экспортируемые переменные во все другие блоки (исключая текущий блок), которые осуществляют экспорт.

Алгоритм

Строка 1	Пусть *level* обозначает *Qual* текущего блока.
Строка 2	Пусть *sur* обозначает *Qual* подструктуры, окружающей блок.
Строки 3-5	Пусть *blockset* обозначает множество *Quals* для всех других блоков в подструктуре.
Строка 6	Рекурсивно пройти по всему множеству *Quals*, пока возможно присоединение *ExpimpchanDs* для блоков

all-exporteurs(*blockset*)(*dict*) ≜ (3.10.1.7)

```
1   (if blockqset = {} then
2      []
3    else
4      (let q ∈ blockqset in
5      let blkqual = dict(SCOPEUNIT) in
6      let mk-BlockD(expm,,, ) = dict(q) in
7      let expm' = [closure ↦ tup | closure ∈ dom expm ∧
8                                  (let tup' = expm(closure) in
9                                  if (∃i ∈ ind tup')(s-Otherend(tup[i]) = blkqual) then
10                                    (let i ∈ ind tup' be s.t. s-Otherend(tup[i]) = blkqual in
11                                     tup = ⟨tup[i]⟩)
12                                  else
13                                  false)] in
14     let rest = get-exporteurs(blockqset \ {q})(dict) in
15     [clos ↦ (rest + expm')(clos) | clos ∈ (dom rest - dom expm')] +
16     [clos ↦ newlist | clos ∈ (dom rest ∩ dom expm') ∧ newlist = rest(clos) ⌢ expm'(clos)]))
```

запись : *Blockqual*-**set** → *Dict* → *ExpimpchanD*

Цель Построить составной *ExpimpchanD*, который включает информацию об экспорте во всех блоках.

Параметры

blockqset Множество (оставшихся) блоков для обработки (функция рекурсивна)

Результат Построенный составной *ExpimpchanD*

Алгоритм

Строка 1 Когда обрабатываются все блоки, возвратить пустое отображение.

Строка 4 Пусть *q* обозначает *Qual* следующего блока в множестве.

Строка 5 Пусть *blkqual* обозначает *Qual* текущего блока.

Строка 6 Пусть *expm* обозначает *ExpimpchanD*, соответствующей экспорту из блока.

Строки 7-13 Ограничить отображение *ExpimpchanD* таким образом, чтобы оно включало только те элементы, где другая конечная точка есть текущий блок.

Строка 14 Построить составной *ExpimpchanD* для остальных блоков.

Строки 15-16 Возвратить *ExpimpchanD*, где списки в двух отображениях (*expm'* и *rest*) конкатенированы. Строка 15 определяет неизмененные входы, строка 16 определяет входы, которые созданы конкатенацией списков для двух входов из отображений

3.10.2 Создание неявных маршрутов сигналов

implicit-signalroutes(*decl*)(*dict*) ≜ (3.10.2.1)

```
1   (let existexplicit = (∃d ∈ decl)(is-Sigroutedef0(d)) in
2   if ¬ existexplicit ∧ (∃mk-Prdef0(, inst,,,,, ) ∈ elems decl)(inst = nil) then
3      exit("§2.5.2: Совокупность допустимых входных сигналов должна быть специфицирована, когда не специфицированы маршруты сигналов")
4     else
5      (let bqual = dict(SCOPEUNIT) in
6       let cqualset =
7          {cqual ∈ dom dict | is-ChannelD(dict(cqual)) ∧
8                              (let mk-ChannelD(endp1, endp2,,, ) = dict(cqual) in
9                               bqual ∈ {endp1, endp2})} in
10      let pathset = {p ∈ dom dict | is-ProcessD(dict(p)) ∧ is-local(p)(dict)} in
11      let pathpairset ∈ (Processqual-set)-set be s.t. (∀e ∈ pathpairset)(card e = 2) ∧
12          union pathpairset = pathset ∧
13          (∀p1, p2 ∈ pathset)(p1 ≠ p2 ⊃ {p1, p2} ∈ pathpairset) in
14      make-as0-local-signalroutes(pathpairset, existexplicit)(dict) ⌢
15      make-as0-env-signalroutes(cqualset, existexplicit)(dict)))
```

запись: $Decl_0^* \rightarrow Dict \rightarrow Decl_0^*$

Цель Построить маршруты неявных сигналов AS_0 для блока

Алгоритм

Строка 1	Пусть *existexplicit* будет **true**, если определены маршруты сигналов в блоке.
Строки 2-3	Если нет маршрутов явных сигналов, определенных для блока, тогда все содержащиеся описания процессов должны содержать совокупность допустимых входных сигналов.
Строка 5	Извлечь *Qual* блока.
Строки 6-9	Извлечь дескрипторы для тех каналов, которые имеют этот блок, как одну из конечных точек.
Строка 10	Построить множество, содержащее *Qual* всех процессов в блоке.
Строки 11-13	Построить множество из множеств *Processqual*, где каждое включенное множество содержит два элемента (строка 11), множества содержат только те *Quals* процессов, которые определены в блоке (строка 12), множество содержит все возможные сочетания множеств (строка 13).
Строка 14	Построить маршруты сигналов между процессами в блоке и
Строка 15	Построить маршруты сигналов, соединяющие с средой блока

make-as$_0$-local-signalroutes(*pathset*, *existexplicit*)(*dict*) ≜ (3.10.2.2)

```
1   (if pathset = {} then
2      ⟨⟩
3    else
4      (let {p1, p2} ∈ pathset in
5      let mk-ProcessD(, insig1, outsig1, ) = dict(p1),
6         mk-ProcessD(, insig2, outsig2, ) = dict(p2) in
7      let (insig1', outsig1') = if existexplicit then
8                                  import-export-signals(dict)
9                                 else
10                                 (insig1, outsig1),
11        (insig2', outsig2') = if existexplicit then
12                                  import-export-signals(dict)
13                                 else
14                                  (insig2, outsig2) in
15     let as0p1id = as0-id(p1),
16        as0p2id = as0-id(p2) in
17     let (com1sig, com2sig) = (insig1' ∩ outsig2', insig2' ∩ outsig1') in
18     let as0sigset1 = {as0-id(id) | id ∈ com1sig},
19        as0sigset2 = {as0-id(id) | id ∈ com2sig} in
20     let com1 = if com1sig = {}
21                 then nil
22                 else mk-Sigroutepath0(as0p1id, as0p2id, as0sigset1),
23        com2 = if com2sig = {}
24                 then nil
25                 else mk-Sigroutepath0(as0p2id, as0p1id, as0sigset2) in
26     let (com', com") = if com1 = nil then (com2, com1) else (com1, com2) in
27     let route0 =
28        if com' = nil ∧ com" = nil
29          then ⟨⟩
30         else (let nm0 = create-unique-name() in
31               ⟨mk-Sigroutepath0(nm0, com', com")⟩) in
32     route0 ⌢ make-as0-local-signalroutes(pathset \ {{p1, p2}}, existexplicit)(dict)))
```

запись : (*Processqual*-set)-set *Bool* → *Dict* → *Sigroutedef*$_0$*

Цель Построить маршруты сигналов AS$_0$, которые ведут между процессами в блоке

Параметры

pathset	Множество пар (множества, содержащие два элемента) процесса
existexplicit	True, если есть определенные маршруты сигналов для блока в AS$_0$

Результат Список маршрутов сигналов AS$_0$

Алгоритм

Строка 1	Когда просмотр закончен, возвратить пустой список (функция рекурсивна).
Строка 4	Пусть *p1* и *p2* обозначают комбинацию процессов, которые подлежат обработке.
Строки 5-6	Произвести декомпозицию дескрипторов процессов.
Строки 7-11	Если маршруты сигналов специфицированы для процесса в AS$_0$, тогда маршруты сигналов построены только для сигналов импорта/экспорта.
Строки 15-16	Составить идентификаторы AS$_0$ для процессов.
Строка 17	Пусть *com1sig* обозначает сигналы, которые идут от *p2* к *p1*, и *com2sig* обозначает сигналы, которые идут от *p1* к *p2*.
Строка 18	Построить множество идентификаторов AS$_0$, соответствующих *com1sig*.
Строка 19	Построить множество идентификаторов AS$_0$, соответствующих *com2sig*.
Строки 20-25	Пусть *com1* и *com2* обозначают пути соединений AS$_0$ от *p2* к *p1* и, соответственно, от *p1* к *p2* (nil, если путь не содержит сигналы).

Строки 26-31	Пусть $route_0$ обозначает маршрут сигналов, который содержит сигналы между двумя процессами.
Строка 32	Объединить определение маршрута сигналов с оставшимися определениями для пар процессов

make-as₀-env-signalroutes (*cqualset, existexplicit*)(*dict*) ≜ (3.10.2.3)

```
 1  (if cqualset = {} then
 2     ◊
 3   else
 4    (let bqual = dict (SCOPEUNIT) in
 5     let cqual ∈ cqualset in
 6     let pqualset = {pqual | pqual ∈ dom dict ∧ is-local(pqual)(dict)} in
 7     let sigroutes = make-as0-channel-signalroutes (cqual, pqualset, existexplicit)(dict) in
 8     if sigroutes = {} then
 9        make-as0-env-signalroutes (cqualset \ {cqual }, existexplicit)(dict)
10       else
11        (let as0 chanid = as0-id(cqual),
12             routenameset = {nm | (∃mk-Sigroutedef0(nm',, ) ∈ sigroutes)(nm' = nm)} in
13         let as0 list = ⟨as0-id(bqual ⇝ ⟨(SIGNALROUTE, nm)⟩) | nm ∈ routenameset⟩ in
14         ⟨sigroute | sigroute ∈ sigroutes⟩ ⇝
15         ⟨mk-Connect0(as0chanid, as0list)⟩ ⇝
16         make-as0-env-signalroutes (cqualset \ {cqual }, existexplicit)(dict))))
```

запись: *Channelqual*-set *Bool* → *Dict* → *Decl**

Цель Построить маршруты неявных сигналов AS_0, которые связаны с средой блока

Параметры

cqualset	Множество каналов, с которыми связан блок
existexplicit	True, если маршруты сигналов определены в конкретном синтаксисе для блока

Результат Построенный список маршрутов сигналов AS_0 и соответствующие определения соединения

Алгоритм

Строка 1	Когда просмотр завершен, возвратить пустой список (функция рекурсивна).
Строка 4	Пусть *bqual* обозначает *Qual* блока.
Строка 5	Взять *Qual* следующего канала для обработки.
Строка 6	Пусть *pqualset* обозначает *Quals* процессов в блоке.
Строка 7	Создать маршруты сигналов, связанных с каналом.
Строка 8	Если нет маршрутов сигналов, связанных с каналом (это ошибка, но она проверялась, когда маршруты сигналов преобразовались в AS_1) или маршруты сигналов для канала уже существуют, тогда продолжить рассматривать остальные каналы.
Строка 11	Создать идентификатор канала AS_0.
Строка 12	Извлечь имена построенных маршрутов сигналов.
Строка 13	Заменить имена идентификаторами.
Строка 14	Возвратить маршруты сигналов и
Строка 15	Их ассоциированное определение соединения и
Строка 16	Маршруты сигналов и соединения, соответствующие остальным каналам

make-as₀-channel-signalroutes(cqual, pqualset, existexplicit)(dict) ≜ (3.10.2.4)

```
1  (if pqualset = {} then
2      ◊
3     else
4      (let mk-ChannelD(end1,, sigset1, sigset2, ) = dict(cqual) in
5       let pqual ∈ pqualset in
6       let mk-ProcessD(, insig, outsig, ) = dict(pqual) in
7       let (insig', outsig') = if existexplicit then
8                                 import-export-signals (dict)
9                                else
10                                (insig, outsig) in
11      let bqual = dict(SCOPEUNIT) in
12      let (fromprocesssignals, toprocesssignals) = if bqual = end1 then
13                                                   (sigset1∩ outsig', sigset2 ∩ insig')
14                                                  else
15                                                   (sigset2 ∩ outsig', sigset1 ∩ insig') in
16      if fromprocesssignals ∪ toprocesssignals = {} then
17        make-as₀-channel-signalroutes(cqual, pqualset \ {pqual}, existexplicit)(dict)
18       else
19        (let routenm = create-unique-name() in
20         let as₀pid = as₀-id(pqual) in
21         let as₀fromsignals = {as₀-id(id) | id ∈ fromprocesssignals },
22             as₀tosignals = {as₀-id(id) | id ∈ toprocesssignals } in
23         let path1 = if fromprocesssignals = {}
24                     then nil
25                     else mk-Sigroutepath₀(as₀ pid, ENV, as₀fromsignals ),
26             path2 = if toprocesssignals = {}
27                     then nil
28                     else mk-Sigroutepath₀(ENV, as₀ pid, as₀ tosignals) in
29         let (path', path'') if path1 = nil then (path2, path1) else (path1, path2) in
30         ⟨mk-Sigroutedef₀(routenm, path', path'')⟩ ⇝
31  make-as₀-channel-signalroutes (cqual, pqualset \ {pqual}, existexplicit)(dict))))
```

запись: *Channelqual Processqual*-**set** *Bool* → *Dict* → *Sigroutedef₀**

Цель Построить маршруты неявных сигналов, соответствующих данному каналу

Параметры

cqual	*Qual* канала
pqualset	*Quals* процессов в блоке
existexplicit	**True**, если маршруты сигналов определены в блоке

Результат Построенный список маршрутов сигналов AS_0

Алгоритм

Строка 1	Когда просмотр завершен, возвратить пустой список (функция рекурсивна).
Строка 4	Пусть *end1* обозначает конечную точку канала, из которой сигналы *sigset* могут быть переданы и из которой сигналы *sigset2* могут быть приняты .
Строка 5	Пусть *pqual* обозначает следующий процесс, который будет рассмотрен.
Строка 6	Пусть *insig* обозначает входные сигналы процесса и *outsig* обозначает выходные сигналы процесса.
Строка 7	Если маршруты сигналов специфицированы в конкретном синтаксисе, тогда маршруты неявных сигналов есть только те, которые предполагаются из импорта/экспорта.
Строка 11	Пусть *bqual* обозначает *Qual* окружающего блока.
Строки 12-15	Пусть *fromprocesssignals* обозначает сигналы, которые переданы из процесса по этому каналу, и *toprocesssignals* обозначает сигналы, которые приняты по этому каналу.

Строки 16-17	Если нет сигналов, которые переданы или приняты по этому каналу, тогда продолжить рассматривать остальные процессы.
Строка 19	Создать неявное имя для маршрута сигналов, который должен быть построен.
Строка 20	Создать идентификатор AS_0 процесса.
Строка 21	Заменить выходные сигналы на идентификаторы сигналов AS_0.
Строка 22	Заменить входные сигналы на идентификаторы сигналов AS_0.
Строки 23-25	Если есть сигналы, переданные из процесса по этому каналу, тогда построить соответствующий $Sigroutepath_0$.
Строки 26-28	Если есть сигналы, принятые процессом по этому каналу, тогда построить соответствующий $Sigroutepath_0$.
Строка 29	Пусть *path'* обозначает путь, который содержит сигналы (первый путь), и пусть *path''* обозначает второй необязательный путь.
Строка 30	Построить и возвратить маршрут сигналов, связанный с каналом, и построить маршруты сигналов для остальных в процессе, связанных с каналом

3.10.3 Создание неявных операторов соединения

implicit-connects(*exp*, *imp*)(*dict*) ≜ (3.10.3.1)

```
1   (let level = dict (SCOPEUNIT) in
2    let expset =
3       {expmap | (∃ qual ∈ dom dict)
4                 (is-BlockD(dict(qual)) ∧
5                 get-sur(qual) = level ∧
6                 dict(qual) = mk-BlockD(expmap,,, ))} in
7    let impset =
8       {impmap | (∃ qual ∈ dom dict)
9       (is-BlockD(dict(qual)) ∧
10       get-sur(qual) = level ∧
11       dict(qual) = mk-BlockD(, impmap,, ))} in
12   let outconnect =
13      if exp = [] then
14         ◊
15        else
16         (let closure ∈ dom exp in
17          gen-connect (expset, closure, exp (closure), exp \ {closure})),
18      inconnect =
19      if imp = [] then
20         ◊
21        else
22         (let closure ∈ dom imp in
23          gen-connect(impset, closure, imp(closure), imp \ {closure})) in
24   outconnect ⌢ inconnect)
```

запись: *ExpimpchanD ExpimpchanD* → *Dict* → $Connect_0^*$

Цель Построить соединения каналов AS_0 для неявных каналов, предполагаемых импортом/экспортом

Параметры

exp	Отображение экспорта *ExpimpchanD* для блока, содержащего блоки, с которыми связаны неявные подканалы
imp	Отображение импорта *ExpimpchanD* для блока, содержащего блоки, с которыми связаны неявные подканалы

Результат Список соединений каналов AS_0

Алгоритм

Строка 1	Пусть *level* обозначает *Qual* текущей подструктуры блока.
Строки 2-6	Построить множество отображений экспорта *ExpimpchanD*. Для каждого подблока в подструктуре существует одно отображение.
Строки 7-11	Построить множество отображений импорта *ExpimpchanD*. Для каждого подблока в подструктуре существует одно отображение.
Строки 12-17	Построить соединения каналов для каналов, предполагаемых в экспорте в подструктуру. *closure* обозначает первый обрабатываемый *NameclosureD*.
Строки 18-23	Построить соединения каналов для каналов, предполагаемых в импорте в подструктуру. *closure* обозначает первый обрабатываемый *ExpimpclosureD*.

gen-connect (*mapset, closure, list, map*) ≜ (3.10.3.2)

```
 1  (if list = ◊ then
 2     if map = [] then
 3        ◊
 4       else
 5        (let closure' ∈ dom map in
 6         gen-connect(mapset, closure', map(closure'), map \ {closure'}))
 7   else
 8    (let (mapset', chanlist) = gen-connect-elem(mapset, closure, ◊) in
 9     let (, channame) = hd list in
10     let connect = mk-Connect0(channame, chanlist ) in
11     ⟨connect⟩ ⇝ gen-connect (mapset ', closure, tl list, map)))
```

запись: *ExpimpchanD*-set *NameclosureD* (*Otherend Channame*$_0$)* *ExpimpchanD* → *Connect*$_0^+$

Цель Построить соединения каналов, предполагаемые импортом или экспортом переменной в/из блок подструктуры

Параметры

expimpset	Множество *ExpimpchanDs* экспорта или *ExpimpchanDs* импорта для блоков в подструктуре блока, т.е. информация о экспорте или импорте в подблоках
closure	*NameclosureD*, которому должен быть обработан
list	Содержит список неявных каналов, соединенных с подструктурой, которая передает неявные сигналы, соответствующие *NameclosureD* (*closure*)
map	*ExpimpchanD* для остальных в *nameclosureDs*, которые должны быть обработаны для подструктуры

Результат Список соединений каналов AS_0

Алгоритм

Строка 1	Когда просмотрены все каналы, соединенные с подструктурой блока, которая соответствует *closure*, тогда
Строка 2	Если нет больше *NameclosureDs*, которые должны быть обработаны, тогда возврат, иначе
Строка 5	Пусть *closure* обозначает следующий *NameclosureD*, который должен быть обработан.
Строка 6	Создать соединения каналов для неявных подканалов, соответствующих *closure*, и создать соединения каналов для остальных *NameclosureDs*.
Строка 8	Изменить множество отображений *ExpimpchanD* (*mapset'*) для подблоков таких, что входы, соответствующие *closure*, которые ведут к среде подструктуры, исключены. Также построить список идентификаторов подканалов AS_0, который будет использован в соединениях каналов.

Строка 9	Пусть *channame* обозначает имя неявного канала, соединенного с подструктурой.
Строка 10	Создать соединение каналов.
Строка 11	Объединить соединение каналов с соединениями каналов для остальных неявных подканалов

gen-connect-elem(mapset, closure, chanlist) ≜ (3.10.3.3)

```
1  (if mapset = {} then
2    ({}, chanlist)
3   else
4    (let map ∈ mapset in
5     let (mapset', chanlist') = gen-connect-elem(mapset \ {map}, closure, chanlist) in
6     let clist = map(closure) in
7     if (∃ index ∈ ind clist)(s-Otherend(clist [index]) = ENV) then
8      (let index be s.t. s-Otherend(clist[index] = ENV in
9       let clist' = ⟨clist[i] | 1 ≤ i ≤ len clist ∧ i ≠ index⟩ in
10      let map' = map + [closure ↦ clist'] in
11      let (, chan) = clist[index] in
12      (mapset' ∪ {map'}, chanlist' ⌢ ⟨mk-Id0 ⟨⟩, chan⟩)))
13    else
14     (mapset', chanlist')))
```

запись: *ExpimpchanD*-set *NameclosureD* Id_0* → *ExpimpchanD*-set Id_0*

Цель Извлечь идентификаторы каналов, которые должны быть использованы в соединении каналов, которые подразумеваются некоторой переменной импорта или экспорта. Идентификаторы каналов находятся в дескрипторах *ExpimpchanD* различных подблоков. После извлечения идентификатора канала информация удаляется из дескриптора *ExpimpchanD*. Функция также возвращает модифицированные дескрипторы *ExpimpchanD*

Параметры

mapset	Множество отображений *ExpimpchanD*. Для каждого подблока существует одно отображение
closure	*NameclosureD*, к которому присоединены неявные подканалы
chanlist	Список идентификаторов каналов, которые созданы рекурсивно

Результат Модифицированное множество отображений *ExpimpchanD* и построенный список идентификаторов подканалов

Алгоритм

Строка 1	Когда просмотрены все дескрипторы *ExpimpchanD*, тогда возвратить построенный список идентификаторов каналов.
Строка 4	Пусть *map* обозначает дескриптор следующего *ExpimpchanD*, который должен быть рассмотрен.
Строка 5	Изменить множество отображений *ExpimpchanD*, соответствующих остальным подблокам, и построить соответствующие подканалы, ведущие к остальным подблокам
Строка 6	*clist* содержит информацию обо всех подканалах, соединенных с подблоком, передающим неявные сигналы, соответствующие *NameclosureD* (*closure*)
Строка 7	Если один из подканалов ведет к среде подструктуры, тогда
Строка 8	Пусть *index* будет индексом элемента, содержащего имя канала, ведущего к среде.
Строка 9	Исключить элемент из списка.
Строка 10	Модифицировать отображение *ExpimpchanD* новым списком.
Строка 11	Пусть *chan* обозначает имя подканала.

Строка 12	Возвратить модифицированное множество отображений *ExpimpchanD* и список идентификаторов каналов.
Строка 14	Если нет подканалов, ведущих в среду подструктуры, тогда возвратить отображение *ExpimpchanD* и неизмененный список идентификаторов каналов

3.11 Сервисные функции

get-predef-sort(*quot*)(*dict*) ≜ (3.11.1)

```
1  (let nm = mk-Name0(quot, nil),
2      level = dict(SCOPEUNIT) in
3   let sys = level[1] in
4   sys ↝ ((TYPE, nm)))
```

запись: *Char** → *Dict* → *Sortqual*

Цель Извлечь *Qual* предопределенного сорта

Параметры

quot	Последовательность символов, представляющая *spelling* имени сорта

Алгоритм

Строка 1	Построить имя AS_0 из *spelling*.
Строки 2-3	Пусть *sys* обозначает *Qual* системного уровня.
Строка 4	Возвратить *Qual* для предопределенного сорта (определенного на системном уровне)

get-parent(*qual*)(*dict*) ≜ (3.11.2)

```
1  (if qual ∈ dom dict then
2      (if is-SortD(dict(qual)) then
3          qual
4         else
5          (let mk-SyntypeD(q,,, ) = dict(qual) in
6           get-parent(q)(dict \ {qual})))
7   else
8    exit("§5.4.1.9: Подтип определен в собственных термах"))
```

запись: *Sortqual* → *Dict* → *Sortqual*

Цель Из *Qual*, обозначающего либо сорт, либо подтип, извлечь родительский сорт (в случае подтипа)

Алгоритм

Строка 1	Если *Qual* находится в *Dict*, тогда
Строки 2-3	Если *Qual* обозначает сорт, тогда возвратить *Qual*, иначе
Строки 5-6	Если *Qual* обозначает подтип, тогда извлечь родительский сорт, но исключить *Qual* подтипа из *Dict* так, что он мог бы быть обнаружен, если подтип определен рекурсивно

is-local(*qual*)(*dict*) ≜ (3.11.3)

```
1  qual = ENV ∨
2  dict (SCOPEUNIT) = get-sur(qual)
```

запись: (*Qual* | ENV) → *Dict* → *Bool*

Цель Возвратить **true**, если *Qual* локально определен в единице подразделения

Алгоритм

Строка 1 Если *qual* есть ENV (используемое в связи с каналами и маршрутами сигналов) или

Строка 2 Они определены в той же единице подразделения, тогда возвратить **true**

process-level (*qual*) ≜ (3.11.4)

```
1  (let (q, ) = qual [len qual] in
2   if q ∈ {PROCEDURE, SERVICE} then
3     process-level (get-sur(qual))
4   else
5     qual)
```

запись: *Qual* → *Qual*

Цель Из *Qual*, обозначающего процесс, процедуру или сервис выбрать *Qual* окружающего процесса

Алгоритм

Строки 1-5 Если *Qual* обозначает процедуру или сервис, тогда применить функцию над *Qual* окружающей единицы подразделения

process-or-service-level (*qual*) ≜ (3.11.5)

```
1  (let (q, ) = qual [len qual] in
2   if q = PROCEDURE then
3      process-or-service-level (get-sur(qual))
4   else
5     qual)
```

запись: *Qual* → *Qual*

Цель Из *Qual*, обозначающего процесс, процедуру или сервис, выбрать *Qual* окружающего процесса или сервиса

Алгоритм

Строки 1-5 Если *Qual* обозначает процедуру, тогда применить функцию к *Qual* окружающей единицы подразделения

get-sur(*qual*) ≜ (3.11.6)

```
1  ⟨qual [i] | 1 ≤ i ≤ len qual - 1 ⟩
```

запись: *Qual* → *Qual*

Цель Из *Qual* возвратить *Qual* окружающей единицы подразделения посредством перемещения последнего элемента

get-inherited-parent(qual)(dict) ≜ (3.11.7)

```
1  (if qual ∈ dom dict then
2     cases dict(qual) :
3      (mk-SortD(, pqual,, )
4        → if pqual = nil then qual else get-inherited-parent(pqual)(dict \ {qual}),
5      mk-SyntypeD(pqual,,, )
6        → get-inherited-parent (pqual)(dict \ {qual}))
7  , else
8    exit("§5.4.1.9: Подтип определен в собственных термах"))
```

запись: *Sortqual* → *Dict* → *Sortqual*

Цель Из *Qual*, обозначающего сорт, либо подтип, извлечь родительский сорт для данного сорта наследования.

Алгоритм

Строка 1 Если *Qual* есть в *Dict*, то

Строка 3 Если *Qual* обозначает ненаследуемый сорт, то возвратить *Qual*, иначе, если *Qual* обозначает наследуемый сорт, то извлечь родительский сорт, но исключить *Qual* сорта из *Dict* таким образом, чтобы его можно было определить в случае, если сорт наследует себя сам.

Строка 5 Если *Qual* обозначает подтип, то извлечь родительский сорт, но исключить *Qual* подтипа из *Dict* таким образом, чтобы его можно было определить в случае, если он наследует сам себя.

3.12 Генерация вспомогательной информации

Этот раздел содержит функции, которые генерируют вспомогательную информацию *Auxiliary information*, которая используется нижележащей системой. Входной функцией является *generate-auxiliary-information*

generate-auxiliary-information(extparms)(dict) ≜ (3.12.1)

```
1  (let (starttime, timeunit) = derive-time-inf(extparms)(dict) in
2   let tick = make-tick-function (timeunit)(dict) in
3   let terminf = make-term-inf (dict) in
4   let expired = make-expired-function(dict) in
5   let delayf = derive-delay-inf(extparms)(dict) in
6   ((tick, starttime), terminf, expired, delayf))
```

запись: *External-Information* → *Dict* → *Auxiliary-Information*

Цель Выдать вспомогательную информацию, которая используется нижележащей системой (приложение F.3)

Параметры

extparms Внешняя информация, из которой выделены требуемые объекты

Результат *Auxiliary-Information*

Алгоритм

Строка 1 Выдать литерал TIME, обозначающий время запуска, и литерал DURATION, обозначающий единицу исчисления времени

Строка 2 Построить функцию, которая применяется для вычисления текущего времени (т.е. она прибавляет *timeunit* к текущему времени)

Строка 3 Построить идентификатор для сорта PID и для трех литералов: NULL, TRUE, FALSE

Строка 4	Построить функцию, которая применяется для проверки истечения выбранного таймера (т.е. сравнением двух значений TIME)
Строка 5	Извлечь функцию, которая применяется в модели неопределенной задержки в каналах
Строка 6	Построить и возвратить *Auxiliary-Information*

make-tick-function(*timeunit*)(*dict*) ≜ (3.12.2)

```
1  (let durationq = get-predef-sort ("DURATION")(dict),
2      timeq = get-predef-sort ("TIME")(dict) in
3  let mk-OperatorD(,, nq, ) = dict(timeq ↝ ⟨(OPERATOR, (PLUS, ⟨(timeq, durationq), timeq)))⟩ in
4  let as1 id = make-as1-identifier(nq)(dict) in
5  let f (value) = mk-Ground-term1((as1 id, ⟨value, timeunit⟩)) in
6  f)
```

запись: *Literal-operator-identifier*$_1$ → *Dict* → (*Ground-term*$_1$ → *Ground-term*$_1$)

Цель Построить функцию, которая обновляет время с помощью данной единицы измерения времени. Функция применяется в абстрактном автомате SDL. Применяется текущее значение NOW и возвращается новое значение

Параметры

timeunit	Единица измерения времени, представленная в виде AS_1 литерала.

Алгоритм

Строка 1	Построить *Qual* сорта DURATION.
Строка 2	Построить *Qual* сорта TIME
Строка 3	Произвести декомпозицию дескриптора оператора "+", определенного для сорта TIME. *Qual* оператора сформирован посредством конкатенации *Qual* сорта TIME и элемента, содержащего класс объекта (OPERATOR), имя оператора (PLUS), сорты аргумента (*timeq* и *durationq*) и результирующий сорт (*timeq*). Ссылка на определение *Operatorqualelem*.
Строка 4	Построить AS_1 идентификатор оператора.
Строки 5-6	Построить и возвратить функцию Meta-IV, которая прибавляет единицу времени к значению, когда задано значение SDL.

make-term-inf (*dict*) ≜ (3.12.3)

```
1  (let bqual = get-predef-sort ("BOOLEAN")(dict) in
2  let pidqual = get-predef-sort ("PID")(dict) in
3  let pid1 = make-as1-identifier(pidqual)(dict),
4      null1 = make-as1-identifier(pidqual ↝ ⟨(LITERAL, mk-Name0("NULL", nil))⟩)(dict),
5      true1 = make-as1-identifier(bqual ↝ ⟨(LITERAL, mk-Name0("TRUE", nil))⟩)(dict),
6      false1 = make-as1-identifier(bqual ↝ ⟨(LITERAL, mk-Name0("FALSE", nil))⟩)(dict) in
7  (pid1, null1, true1, false1))
```

запись: *Dict* → *Term-Information*

Цель Построить *Term-Information* для использования в динамических семантиках (см. определение домена *Term-Information*)

Результат *Term-Information*, состоящий из идентификаторов сорта PID, литералов NULL, TRUE, FALSE.

make-expired-function(dict) ≜ (3.12.4)

1 (let *bqual* = *get-predef-sort* ("BOOLEAN")(*dict*) in
2 let *geop* = mk-*Qualop*$_0$(*bqual*, mk-*Quotedop*$_0$(GE)) in
3 let *expiredf*(*currenttime*, *expiredtime*) =
4 *eval-simple-expr* (mk-*Operatorapp*$_0$(*geop*, (*currenttime*, *expiredtime*)), "BOOLEAN")(*dict*) in
5 *expiredf*)

запись: *Dict* → *Is-expiredF*

Цель Построить функцию, которая проверяет, превышает ли данное время значение времени окончания, т.е. построить функцию, которая выдает значение Meta-IV **true**, если

```
currenttime >= expiredtime,
```

где `currenttime` и `expiredtime` - формальные параметры.

Функция применяется в динамических семантиках

Алгоритм

Строка 1 Построить *Qual* сорта BOOLEAN
Строка 2 Построить AS_0 идентификатор оператора " >= ".
Строки 3-5 Построить и возвратить функцию *Is-Expired* (*expiredf*).

3.13 Отображение глобальных констант

Ниже определяются два глобальных отображения *name-to-name*$_1$ и *exportmap*.

3.13.1 Соотношение между именами AS_0 и AS_1

Имена в AS_1 не имеют такого представления, как в AS_0, так как проверка имен не имеет значения для интерпретации. Таким образом, имена AS_0 (*Names*$_0$), символьные строки (*String*$_0$), и выражения типа "оператор" (*Quotedop*$_0$) преобразуются в имена AS_1 согласно значениям глобальных отображений (*name-to-name*$_1$).

let *name-to-name*$_1$ be s.t. ($\forall nm_1 \in$ rng *name-to-name*$_1$)(is-*Name*$_1$(nm_1)) $\wedge$
($\forall$($nm1$, $nm2$) $\in$ (*Name*$_0$ | *String*$_0$ | *Quotedop*$_0$))
(({$nm1$, $nm2$} $\subset$ dom *name-to-name*$_1$) $\wedge$
($nm1 \neq nm2 \supset$ *name-to-name*$_1$($nm1$) $\neq$ *name-to-name*$_1$($nm2$)) in

Цель Построить отображение, которое конвертирует AS_0 имена, строки и операторы в кавычках в AS_1 имена.

Алгоритм Построить отображение такое, что все возможные AS_0 имена, строки и операторы в кавычках существуют в отображении, и каждые два элемента отображения преобразуются в различные AS_1 имена.

3.13.2 Соотношение между именами импорта/экспорта и именами неявных сигналов

В SDL несколько неявных сигналов связаны с каждой переменной экспорта. Две переменные экспорта подключают одинаковые неявные сигналы, если они имеют одинаковые имена (*spelling*) и являются переменными одного сорта данных. С целью управления этими "общими" сигналами используется специальное отображение констант *exportmap*, элементы которой в качестве дескриптора подключаемого сигнала состоят из пары: сорт и имя переменной экспорта.

let *exportmap* = [*closure* ↦ mk-*SignalnamesD*(*create-unique-name*(),
create-unique -name(),
create-unique -name() | *closure* ∈ *NameclosureD*] in

Домен *exportmap* определен следующим образом:

1	*Exportmap*	=	*NameclosureD* ⇸ *SignalnamesD*
2	*NameclosureD*	=	*Sortgual Name*$_0$
3	*SignalnamesD*	::	*Impliednm Xquerynm Xreplynm*
4	*Impliednm*	=	*Name*$_0$
5	*Xquerynm*	=	*Name*$_0$
6	*Xreplynm*	=	*Name*$_0$

Дескриптор сигнала (*SignalnamesD*) состоит из:

Impliednm — Неявная переменная, присоединяемая к каждой переменной импорта.

Xquerynm — Сигнал xtQUERY, посылаемый импортером.

Xreplnm — Сигнал xtREPLY, посылаемый экспортером и несущий значение сорта *Sortqual*.

3.14 Неформальные функции

apply-generic-parameters(*genericsystem, extparameters*) ≜ (3.14.1)

1 /* *Преобразовать общие определения системы в конкретные системные определения* */

запись: *Sys*$_0$ *External-Information* → *Sys*$_0$

SDL не определяет каким образом применяются общие параметры. Неформальная функция требует определение системы AS$_0$ и внешнюю информацию, которая включает некоторые фактические общие параметры, и возвращает определение системы AS$_0$, не содержащее общих параметров, т.е. определение системы, не содержащее внешних синонимов и неформального текста в ответах необязательных действий

derive-time-inf (*extparms*)(*dict*) ≜ (3.14.2)

1 /* *Выдать стартовое время и единицу времени из внешней информации* */

запись: *External-Information* → *Dict* → *Literal-operator-identifier*$_1$ *Literal-operator-identifier*$_1$

Неформальная функция возвращает *Literal-operator-identifier*$_1$ сорта TIME, обозначающий стартовое время системы, и *Literal-operator-identifier*$_1$ сорта DURATION, обозначающий значение увеличения абсолютного времени (оба значения применяются в модели для выражения NOW, см.Приложение F.3)

derive-delay-inf(*extparms*)(*dict*) (3.14.3)

1 /* *Выдать функцию задержки из внешней информации* */

запись: *External* → *Information* → *Dict* → *DelayF*

Эта неформальная функция выдает из внешней информации императивную функцию (*DelayF*), применяемую в приложении F.3 для моделирования недетерминированной задержки в каналах.

eval-expr(*expr, sort*)(*dict*) ≜ (3.14.4)

1 /* *Вычислить значение* */

запись: *Ground-expression*$_1$ *Char*$^+$ → *Dict* → (*Intg* | *Bool*)

Цель Вычислить простое выражение AS_1

Параметры

expr Простое выражение AS_1
sort Написание сорта выражения

Результат META-IV присваивается тип Intg, если сорт SDL есть "Integer", META-IV присваивается тип Bool, если сорт SDL есть "Boolean".

Алгоритм Для экономии места эта функция не включена в формальное определение. См. §5.6 Z.100 для формальных спецификаций предопределенных сортов и Приложение F.3 для формального определения модели данных SDL.

select-consistent-subset (*systemdefinition, genericparameters*) ≜ (3.14.5)

1 /* *По значению AS_1 и фактическому подмножеству параметров выделить совместимое подмножество* */

запись: *Systemdefinition*$_1$ *External-Information* → *Block-identifier*$_1$-**set**

Эта неформальная функция возвращает множество идентификаторов блоков, обозначающие совместимые подмножества. Эта функция применяется в Приложении F.3 для выделения блоков, которые должны быть интерпретированы. В Приложении F.3 проверяется, что совместимые подмножества согласуются со свойствами, определенными в §3.2.1 в Z.100.

create-unique-name() ≜ (3.14.6)

1 /* *Создать новый* **mk**-*Name*$_0$ (*Token*, **nil**) */

запись: () → *Name*$_0$

Эта неформальная функция возвращает AS_0 *Name*$_0$, которое ранее не было использовано. Заметьте, что эта функция по своей природе является императивной. Подразумевается, что глобальные переменные, запоминающие последовательность используемых имен, используются функцией

check-name-syntax (*string*) ≜ (3.14.7)

1 /* *Заменить символьную строку заглавными буквами* */

запись: *Char** → *Char**

Во-первых, любая последовательность подчеркнутых символов, сопровождаемых пробелами, удаляется, затем любая последовательность пробелов заменяется символом подчеркивания, затем все символы заменяются на заглавные, и, наконец, проверяется соответствие полученного написания правилам написания имен, определенным в §2.2.1 в Z.100.

4 Уточнения к Z.100

После последнего издания текста рекомендаций Z.100 SG X (Март 1988) в тексте рекомендаций было найдено несколько небольших несоответствий и неясностей. С целью дополнения формального описания было принято несколько уточнений в следующих ситуациях:

1. §2.7.4 в Z.100 предложение:
   "Тем не менее, порядок установлен, если два сигнала переданы по идентичному пути, соединяющему *Original-process* и *Destination-process*."
   интерпретируется как
   "Тем не менее, порядок установлен, если два сигнала переданы по идентичным каналам, соединяющим *Original-process* и *Destination-process*, или если они определены в одном блоке."

2. §2.7.5 в Z.100 предложение:
"*Decision-answers* должны быть взаимно исключающими."
интерпретируется как
"*Decision-answers* должны быть взаимно исключающими и должны быть одного сорта. Если *Decision-question* содержит *Expression*(выражение), то оно должно быть того же сорта, что и *Decision-answers*."

3. §2.7 в Z.100 с целью избежать циклических определений добавляется новое предложение перед *Model*:
"Контекст решения (т.е. сорт) определен независимо от <ответов>, которые есть <символьная строка>."

4. §3.3 в Z.100 предложение:
"Если согласованое частичное подмножество содержит определение блока, связанное с подсигналами, тогда блок, с которым оно связывается, должен использовать те же сигналы."
интерпретируется как
"При выборе согласованного подмножества множество сигналов в маршрутах сигналов, связанное с конечной точкой канала, не должно содержать родительских сигналов для содержащихся подсигналов, и, если только другая конечная точка не есть ENVIRONMENT системы, то множество сигналов для первой конечной точки должно соответствовать множеству сигналов маршрутов сигналов для другой конечной точки.

5. §4.3.3 в Z.100 предложение:
"<Булевское простое выражение> не должно зависеть от какого-либо определения внутри <определения отбора>."
модифицируется следующим образом (для обеспечения единственности выбора и единственности построения синонимов):
"Синоним, использующийся непосредственно или косвенно (через другие синонимы) в <булевском простом выражении> в <определении отбора> должно удовлетворять следующим требованиям:

- Он должен соответствовать правилам видимости, описанным в §2.2.2, даже, если определение синонима не является выбранным.
- Сорт <определения синонима> должен квалифицироваться на системном уровне, если существует видимое переопределение этого сорта, даже если переопределение является локальным или выбранным.
- Должно существовать порядок выбора, который позволяет вычислять все <простые выражения>.

6. §4.10.2 в Z.100 предложение:
"Сигнал является сигналом с высоким приоритетом тогда и только тогда, когда он задействован в <приоритетном вводе> <определения сервиса>."
интерпретируется как
"Сигнал является сигналом с высоким приоритетом в процессе тогда и только тогда, когда он задействован в <приоритетном вводе> <определения сервиса> в процессе."

7. §4.10.2 в Z.100 предложение:
"<Определение процедуры> не должно иметь таких <состояний>, когда объемлющее <определение процесса> содержит <определение сервиса>."
интерпретируется как
"<Определение процедуры> не должно иметь таких <состояний>, когда объемлющее <определение процесса> содержит <определение сервиса>." <Определение процедуры>, обозреваемое более, чем в одном сервисе не должно содержать конструкцию VIA."

8. §5.6.7.1 в Z.100 равенства :

r > 1 == True => a * r > a == True;
r > 1 == True => a / r < a == True;

должны быть

r > 1 == True => a * r > = a == True;
r > 1 == True => a / r <= a == True;

для описания случая, когда a равно нулю.

Указатель доменов

Act$_0$ 7, **8**

Activeexpr$_0$ **18**, 149, 152

Actparmlist$_0$ **8**, 213, 215

Actstmt$_0$ **7**, 166, 167, 168, 170, 183, 184, 187, 190, 198, 199, 203, 204, 205, 206, 207, 208, 212, 214, 231

Andregexp$_0$ **16**, 97

Answer$_0$ **9**, 11, 167, 171, 184, 187, 218, 219, 221

Assignment-statement$_1$ **Z.100**, 164, 200, 206

Assignstmt$_0$ **8**, **18**, 183, 199, 200, 201

Auxiliary-Information **30**, 33, 258

Axiom$_0$ 12, **13**, 17, 85, 100, 101, 106, 107, 109, 110, 113, 114, 115

BlockD 21, 22, **23**, 52, 54, 63, 211, 245, 246, 247, 248, 253

Block-definition$_1$ **Z.100**, 30, 226

Block-identifier$_1$ **Z.100**, 262

Block-qualifier$_1$ **Z.100**, 223

Block-substsucture-definition$_1$ **Z.100**, 226

Block-substructure-qualifier$_1$ **Z.100**, 223

BlockconnectionD **23**, 125

Blockdecl$_0$ **3**

Blockdef$_0$ **3**, 10, 11, 33, 35, 37, 38, 41, 48, 52, 54, 58

Blockid$_0$ **3**, 5, 37

Blockname$_0$ **3**

Blockqual 23, **24**, 247, 248

Blockref$_0$ **3**, 10, 11, 37

BlocksubD 22, **23**, 54

Blocksub$_0$ 3, **10**

Blocksubdecl$_0$ **10**

Blocksubdef$_0$ **10**, 38, 39, 43, 44, 48, 54, 58

Blocksubid$_0$ **10**

Blocksubname$_0$ **10**

Blocksubref$_0$ **10**, 38

Body$_0$ 4, 5, **7**, 11, 39, 48, 161, 162, 234

Bool 22, **23**, 27, 30, 39, 41, 44, 47, 69, 70, 74, 97, 98, 99, 100, 127, 130, 131, 160, 178, 208, 221, 227, 240, 244, 250 251 252 256 261

Call-node$_1$ **Z.100**, 214

Call$_0$ **8**, 198, 214

Chandef$_0$ 3, **5**, 10, 11, 38, 41, 48, 52, 57, 58, 128, 244

Channame$_0$ **5**, 23, 24, 60, 244, 254

ChannelD 21, 22, **24**, 57, 58, 125, 126, 128, 129, 130, 211, 224, 249, 252

Channel-connection$_1$ **Z.100**, 128, 129, 226

Channel-definition$_1$ **Z.100**, 30, 57, 226

Channel-path$_1$ **Z.100**, 57

Channel-to-route-connection$_1$ **Z.100**, 125, 126, 226

Channelqual **23**, 126, 128, 129, 130, 251, 252

ChannelsubD 22, **23**, 59, 224

Chanpath$_0$ **5**, 57, 58, 128, 244

Chansub$_0$ 5, **10**

Chansubdecl$_0$ **10**

Chansubdef$_0$ 3, **10**, 35, 38, 39, 48, 58, 59

Chansubid$_0$ **10**, 38

Chansubname$_0$ **10**

Chansubref$_0$ **10**, 38

Char **2**, 3, 44, 96, 97, 98, 99, 100, 256, 261 , 262

Closed-range$_1$ **Z.100**, 92

Composite-term$_1$ **Z.100**, 83, 141, 147

Condequation$_0$ 13, **14**, 85, 101, 105, 107

Condexpr$_0$ **17**, 135, 154

Condition-item$_0$ **Z.100**, 92

Condition$_0$ **14**, 91, 184, 187, 217, 218, 219, 222

Conditional-equation$_1$ **Z.100**, 82, 105

Conditional-expression$_1$ **Z.100**, 154

Conditional-term$_1$ **Z.100**, 83, 154

Conditionlist$_0$ 9, 12, **14**, 221, 222

Condterm$_0$ **13**, 85, 135

Connect$_0$ 3, **6**, 10, 11, 41, 48, 60, 125, 126, 128, 129, 132, 133, 251, 253, 254

Connectpoint$_0$ **6**, 48

ContenablestateD **28**, 181, 188, 195

Context **30**, 100, 101, 102, 103, 104, 105, 134, 135, 139, 140, 143, 144, 146, 147, 148, 149, 152, 153, 154, 155, 213

Contspec$_0$ 7, **12**, 165, 181, 182, 184, 185, 237

Create-request-node$_1$ **Z.100**, 212

Create$_0$ **8**, 198, 212

Data-type-definition$_1$ **Z.100**, 28, 30, 33, 56, 73, 94, 227

Datadef$_0$ 3, **4**, 5, 10, 11, 17, 32, 33

Decision-answer$_1$ **Z.100**, 164, 219

Decision-node$_1$ **Z.100**, 28, 164, 193, 217, 231

Decision$_0$ 8, **9**, 166, 170, 184, 187, 198, 217

Decl$_0$ **11**, 35, 39, 41, 43, 44, 45, 46, 48, 49, 50, 51, 52, 60, 125, 128, 242, 249, 251

Decl$_1$ **30**, 51, 52, 54, 57, 59, 70, 71, 72, 121, 123, 161, 226, 233, 241

Decomposition$_0$ 4, **11**, 38, 39, 43, 44, 48, 161

Decompositiondecl$_0$ **11**, 132

DelayF **30**, 261

Descr **21**

Descriptordict 20, **21**

Dest$_0$ **5**

Destination$_0$ **5**

Destinationprocess **25**

Dict **20**, 22, 33, 43, 44, 45, 46, 47, 48, 51, 65, 66, 67, 68, 09, 70, 71, 72, 73, 74, 75, 77, 78, 79, 80, 81, 84, 90, 91, 92, 93, 94, 100, 101, 102, 103, 104, 105, 100, 107, 108, 109, 110, 111, 112, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135, 137, 139, 140, 141, 142, 143, 144, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164, 174, 175, 178, 180, 181, 182, 184, 185, 188, 189, 190, 191, 192, 193, 195, 196, 197, 198, 199, 200, 201, 203, 204, 205, 206, 207, 208, 211, 212, 213, 214, 215, 216, 217, 218, 219, 220, 221, 223, 224, 225, 226, 227, 228, 229, 231, 233, 234, 235, 237, 238, 239, 240, 241, 242, 244, 245, 246, 247, 248, 249, 250, 251, 252, 253, 256, 258, 259, 260, 261

Else-answer$_1$ **Z.100**, 164, 217

Elsepart$_0$ **9**, 11, 167, 171, 217, 221

Emptyqid **28**

Enabling$_0$ **7**, 12, 181, 186

Endpoint **24**, 59, 60

Entity **21**

Equation$_0$ **13**, 85, 101, 102, 105, 107, 109, 114, 115, 228

Equation$_1$ **Z.100**, 82, 104, 105

Equations$_1$ **Z.100**, 26, 81, 100, 101, 102, 106, 228

ErrorD 22, **28**, 43, 46, 70, 93, 123, 124, 158

Error-term$_1$ **Z.100**, 83, 103

Errorterm$_0$ 13, **14**, 103, 105, 135

ExpimpchanD **23**, 242, 244, 245, 247, 248, 253 254, 255

Explicit **27**, 77, 104

Explicitroutes **23**, 210

Export$_0$ 8, **12**, 198

Exportchannels **23**

Exportmap, **261**

Expr$_0$ 4, **8**, 9, 11, 12, 14, 17, 18, 26, 28, 44, 47, 92, 118, 135, 139, 145, 149, 150, 151, 153, 201, 216, 221, 222

Expression$_1$ **Z.100**, 26, 135, 139, 143, 147, 149, 150, 152, 153, 154, 155, 208, 213, 215, 216

Exprlist$_0$ **18**, 145, 146, 147, 201

External-Information **30**, 32, 33, 258, 261, 262

Extproperties$_0$ 12, **14**

Fieldname$_0$ **15**, 113, 114, 115

Fieldspec$_0$ **15**, 113

Fieldvar$_0$ **18**, 201

FormparmD **25**, 68, 69, 215

Formuniquenm **28**

Genactparm$_0$ **16**, 89

GeneratorD **21**, 22, **26**, 70, 84

Generatorid$_0$ **16**

Generatorname$_0$ **15**

Geninst$_0$ **16**, 84

Geninstlist$_0$ 14, **15**, **16**, **71**

Genparm$_0$ **15**, 26, 70, 89

Globalnames **28**, 35

Graph-node$_1$ **Z.100**, 28, 193, 195, 198, 199, 203, 204, 205, 206, 207, 208, 212, 214, 217, 220, 231

Ground-expression$_1$ **Z.100**, 135, 261

Ground-term$_1$ **Z.100**, 30, 83, 135, 139, 141, 142, 147, 154, 259

Id$_0$ **2**, 3, 4, 5, 6, 8, 9, 10, 11, 13, 15, 16, 17, 35, 37, 38, 39, 48, 54, 58, 60, 61, 66, 69, 71, 84, 85, 87, 89, 102, 107, 109, 110, 112, 114, 115, 120, 124, 135, 137, 139, 140, 141, 142, 144, 145, 148, 150, 151, 152, 153, 156, 158, 159, 160, 180, 181, 184, 187, 190, 200, 201, 216, 239, 231, 233, 237, 244, 255

Identifier$_1$ **Z.100**, 73, 83, 223, 225, 227

Impliednm, **261**

Impoperator$_0$ **17**, 18

ImportD 21, 22, **27**, 121, 150, 231, 246

Importchannels **23**

Importdef$_0$ 4, 11, **12**, 41, 52, 121

Importelem$_0$ **12**, 121

Importexpr$_0$ **12**, 18, 149

Importqualelem **21**

Importstateinf **28**

InDescr **25**, 69, 215

In-parameter$_1$ **Z.100**, 69

Indexedvar$_0$ **18**, 201

Infixexpr$_0$ **17**, 135, 155, 183, 184

Infixop$_0$ **14**, 17, 18

Infixterm$_0$ 13, **14**, 85, 107, 109, 110, 135

Informal-text$_1$ **Z.100**, 101, 200, 217, 219

Inherited$_0$ 14, **15**, 71, 73

Initial$_0$ **4**

Initialvalue$_0$ 12, 14, 16, **18**

InoutDescr **25**, 69, 215, 216

Inout-parameter$_1$ **Z.100**, 69

Inoutparm$_0$ **5**, 68

Inparm$_0$ **5**, 68

Input-node$_1$ **Z.100**, 189, 191, 192, 193

Inputset$_0$ **4**, 11, 63

Inputspec$_0$ **7**, 165, 175, 178, 181, 184, 186, 187, 190, 191, 231

Inputvars$_0$ **7**, 11, 178, 180, 184, 190, 192, 231, 234

Instances$_0$ **4**, 41, 61
Intg 44, 99, 261
Is-expiredF **30**, 260

Join$_0$ **8**, 168, 195

Kind **21**, 160

Label$_0$ **7**, 8, 28
Labeldict 23, **28**, 169, 170, 171, 191, 234
LiteralD 21, 22, **27**, 71, 75, 94, 106, 139, 224
Literal-operator-identifier$_1$ **Z.100**, 30, 75, 81, 82, 83, 259, 261
Literalsignature$_1$ **Z.100**, 73, 94
Literal$_0$ **12**, 85, 87, 89, 110
Literalaxiom$_0$ **17**
Literalpair$_0$ **15**, 76
Literalrenaming$_0$ **15**, 75, 76
Litparm$_0$ **15**, 70, 89

Mappingaxiom$_0$ **12**, 17, 85, 101, 106
Mapvalue **27**, 104, 148
Maximum$_0$ **4**
Monadexpr$_0$ **17**, 109, 135, 155
Monadterm$_0$ 13, **14**, 85, 107, 135

Name$_0$ **2**, 3, 4, 5, 6, 7, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 21, 28, 35, 37, 43, 44, 45, 46, 48, 49, 50, 59, 69, 70, 73, 78, 84, 85, 87, 89, 94, 96, 98, 99, 100, 102, 107, 109, 110, 112, 113, 114, 115, 118, 144, 145, 148, 152, 153, 181, 183, 184, 187, 190, 201, 206, 226, 228, 229, 237, 256, 259, 260, 261, 262
Name$_1$ **Z.100**, 260
NameclosureD, 23, 28, 230, 246, 247, 254, 255, **261**
Newchannels **24**, 225
Newliteral$_0$ **15**
Newoperator$_0$ **15**, 79
Newqual 24, **25**, 26, 27, 73
Nextstate-node$_1$ **Z.100**, 195, 235
Nextstate$_0$ **8**, 175, 181, 184, 188, 190, 195, 196, 231, 237, 238
Nmclass$_0$ 12, **16**, 87, 89, 96, 110
Now-expression$_1$ **Z.100**, 151
Nowexpr$_0$ **18**, 149
Number-of -instances$_1$ **Z.100**, 61, 226
N$_0$ **185**
N$_1$ 113, 114, 115, 184, 185, 239

Offspring-expression$_1$ **Z.100**, 153
Offspringexpr$_0$ **18**, 153
Oldliteral$_0$ **15**
Oldoperator$_0$ **15**
Op$_0$ **12**, 85, 113, 114, 115
Open-range$_1$ **Z.100**, 92
OperatorD 21, 22, **27**, 71, 77, 78, 79, 80, 92, 111, 224, 259
Operotor-application$_1$ **Z.100**, 147
Operator-identifier$_1$ **Z.100**, 77, 81, 82, 83
Operator-signature$_1$ **Z.100**, 77, 78, 111
Operatorapp$_0$ **17**, **18**, 135, 142, 143, 145, 152, 153, 155, 201, 222, 260
Operatorname$_0$ **12**, 15
Operatorpair$_0$ **15**, 77, 78
Operatorqual **30**, 144, 146, 147
Operatorqualelem **21**
Operatorrenaming$_0$ **15**, 77, 78
Operatorterm$_0$ **13**, 85, 114, 115, 135, 144, 155, 228
Opparm$_0$ **15**, 70, 89
Opspec$_0$ **12**, 85, 107, 108, 110, 111, 113, 114, 115
Option$_0$ 8, **11**, 166, 170, 198, 220
Ordering$_0$ 12, **17**, 108
Orig$_0$ **5**
Origin$_0$ **5**
Originprocess **25**
Orregexp$_0$ **16**, 97
Otherend **23**, 244, 248, 254, 255
Output-node$_1$ **Z.100**, 208
Output$_0$ **8**, 183, 190, 198, 207, 208, 231
Outputset **25**
Outputsig$_0$ **8**, 11, 183, 190, 208, 231

ParameterD **25**, 213
Parenregexp$_0$ **16**, 97
Parent-expression$_1$ **Z.100**, 153
Parentexpr$_0$ **18**, 153
Parentid$_0$ 14, **15**
Parentqual **26**
Parm$_0$ **4**, 64
Partialtypedef$_0$ **12**, 17, 50, 52, 71
Partregexp$_0$ **16**
Pid-expression$_1$ **Z.100**, 153
Piexpr$_0$ **8**
Prdecl$_0$ **4**
Prdef$_0$ 3, **4**, 11, 35, 37, 39, 41, 48, 52, 54, 61, 249
Prid$_0$ **4**, 8, 37
Priinput$_0$ 7, **11**, 166, 178, 191, 234, 237
Priinputset **23**, 208, 240
Priority$_0$ **12**, 182
Prioutput$_0$ 8, **11**, 198, 206
Prname$_0$ **4**
Procdecl$_0$ **5**
Procdef$_0$ 3, 4, **5**, 11, 35, 37, 39, 41, 52, 66, 67
ProcedureD 21, 22, **25**, 67, 162, 214, 224
Procedure-definition$_1$ **Z.100**, 30, 67, 226
Procedure-formal-parameter$_1$ **Z.100**, 68, 69, 224
Procedure-graph$_1$ **Z.100**, 67, 162, 226
Procedure-qualifier$_1$ **Z.100**, 223
Procedure-start-node$_1$ Z.100, 162
ProcessD 21, 22, **25**, 61, 117, 156, 175, 208, 210, 212, 246, 249, 250, 252

Process-definition$_1$ **Z.100**, 30, 61, 226
Process-formal-parameter$_1$ **Z.100**, 64, 226
Process-graph$_1$ **Z.100**, 161, 162, 226, 236
Process-qualifier$_1$ **Z.100**, 223
Process-start-node$_1$ **Z.100**, 162, 235
Processbody$_0$ **4**, 161
ProcessconnectionD **25**, 132, 161
Processqual **25**, 30, 249, 250, 252
Procid$_0$ **5**, 8, 37
Procname$_0$ 4, **5**
Procparm$_0$ **5**, 68
Procref$_0$ **4**, 5, 11, 37
Properties$_0$ **12**, 15, 26, 73, 84, 94, 96, 108, 112
Prref$_0$ 3, **4**, 11, 37

Qual **21**, 23, 24, 25, 27, 28, 30, 35, 39, 41, 76, 78, 79, 80, 93, 106, 118, 120, 125, 126, 127, 132, 133, 140, 144, 151, 155, 156, 158, 160, 164, 172, 190, 211, 218, 223, 224, 225, 228, 229, 256, 257
Qualelem **21**
Qualifier$_0$ **2**, 8, 13, 17, 18, 37, 49, 89, 101, 145, 201, 217, 218
Qualifier$_1$ **Z.100**, 223
Qualifierelem$_0$ **2**
Qualop$_0$ 13, **18**, 87, 142, 144, 145, 155, 158, 222, 228, 260
Quantequation$_0$ **13**, 85, 101, 104, 107, 109
Quantified-equations$_0$ **Z.100**, 82, 104
Question$_0$ **9**, 11
Quot 158, 226
Quotdict 20, **28**
Quotedop$_0$ 12, 16, **18**, 21, 85, 87, 89, 144, 155, 222, 228, 260

Range-condition$_1$ **Z.100**, 26, 91, 164
Refdecl$_0$ **3**, 39
Refinement$_0$ 6, **10**, 65
Regexpexp$_0$ **16**,198, 99, 100
Regularexp$_0$ **16**, 97, 100
Relop$_0$ **14**, 92
Remotedef$_0$ **3**
Reset-node$_1$ **Z.100**, 203
Reset$_0$ 8, **9**, 198, 203
Resetelem$_0$ **9**, 203
Result **27**, 71, 143, 144
Result$_1$ **Z.100**, 227
Return-node$_1$ **Z.100**, 195
Return$_0$ **8**, 195
Rngregexp$_0$ **16**, 97

Save-signalset$_1$ **Z.100**, 189
Savespec$_0$ **7**, 169, 177, 187, 189, 231, 237
Scope$_0$ **8**
Scopeexpr$_0$ **8**, 9, 135, 184, 187
Scopeunit$_0$ **2**, 21
Select$_0$ 3, 4, 5, 10, **11**, 39, 44, 49, 50
Selectexpr$_0$ **17**, 145, 149, 152, 201
Self-expession$_1$ **Z.100**, 153
Selfexpr$_0$ **18**, 153, 183, 206
Sender-expression$_1$ **Z.100**, 153
Senderexpr$_0$ **18**, 153, 190
ServiceD 22, **23**, 63, 66, 0, 49, 90, 94, 116, 117, 156, 175, 191, 195, 196, 208, 210, 224, 233, 235, 238, 239, 240
Servicedecl$_0$ **11**
Servicedef$_0$ 3, **11**, 35, 37, 39, 48, 52, 233
Serviceid$_0$ **11**, 37
Servicename$_0$ **11**
Servicequal **28**, 237, 238, 239
Serviceref$_0$ **11**, 37
Servicetuple **28**, 196
Set-node$_1$ **Z.100**, 205
Set$_0$ 8, **9**, 198, 204, 205
Setelem$_0$ **9**, 205
Sigdef$_0$ 3, 4, **6**, 10, 11, 35, 41, 52, 65, 73
Sigelem$_0$ **6**, 35, 65, 73
Sigid$_0$ **6**, 7, 8, 159, 182, 183, 184, 187
SignalD 21, 22, **24**, 63, 65, 131, 180, 192, 203, 205, 208
Signal-definition$_1$ **Z.100**, 30, 65, 226
Signal-qualifier$_1$ **Z.100**, 223
Signal-refinement$_1$ **Z.100**, 65
Signal-route-definition$_1$ **Z.100**, 30, 122, 226
Signal-route-path$_1$ **Z.100**, 122
SignallistD 21, 22, **25**, 123, 124
Signallist$_0$ 4, 5, **6**, 7, 25, 03, 124
Signallistdef$_0$ 3, 4, **6**, 10, 11, 41, 52, 123
Signallistid$_0$ **6**, 124
Signallistname$_0$ **6**
SignalnamesD, 73, 118, 157, 158, 190, 206, 231, 244, **261**
Signalqual 23, **24**, 25, 28, 59, 63, 66, 124, 127, 130, 131, 156, 157, 158, 159, 160, 175, 178, 180, 192, 211, 234
SignalrouteD 21, 22, **25**, 63, 122, 127, 132, 133, 210, 211, 240, 241
Signame$_0$ **6**
Sigroutedef$_0$ 3, **5**, 11, 41, 48, 52, 54, 122, 125, 132, 241, 249, 250, 251, 252
Sigroutename$_0$ **5**
Sigroutepath$_0$ **5**, 122, 125, 132, 241, 250, 252
Singregexp$_0$ **16**, 97
SortD 21, 22, **26**, 46, 73, 74, 90, 92, 94, 116, 155, 224, 256, 258
Sort-identifier$_1$ **Z.100**, 30, 82
Sort-qualifier$_1$ **Z.100**, 223
Sortgenerator$_0$ **15**, 17, 52, 70
Sortid$_0$ 4, 5, **6**, 9, 12, 13, 15, 16, 17, 113, 114, 115
Sortname$_0$ **12**
Sortparm$_0$ **15**, 70, 89
Sortqual 21, **24**, 25, 26, 27, 64, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 91, 92,

94, 106, 107, 108, 109, 110, 134, 135, 137, 139, 140, 141, 142, 143, 144, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 156, 160, 193, 206, 218, 219, 221, 222, 256, 258, 261

Spec **28**, 184, 187

Speclist **28**, 173, 175, 181, 182, 184, 185, 186, 187, 188, 189, 190, 191, 239

Spellingterm$_0$ 13, **17**, 85, 135, 148

Starred$_0$ **7**, 177, 178

Starredlist$_0$ **7**, 172

StateD **28**, 173, 175, 180, 181, 188, 231, 234, 237, 239

State-node$_1$ **Z.100**, 188

Statebody$_0$ **7**, 165, 169, 172

Statedict 23, **28**, 172, 173, 174, 180, 181, 186, 187, 188, 234, 237

Statename$_0$ **7**, 8, 28, 180, 182, 184, 188, 189, 191, 192, 195, 196, 197, 198, 199, 203, 204, 205, 206, 207, 208, 212, 214, 217, 219, 220, 231, 238, 239

Statenamelist$_0$ **7**, 172, 173

Statespec$_0$ **7**, 28, 165, 169, 178, 186, 189

Statetuplemap **28**

Stop-node$_1$ **Z.100**, 195

Stop$_0$ **8**, 195

String$_0$ **3**, 12, 13, 15, 16, 21, 87, 89, 96, 98, 99, 217, 228, 260

Stringterm$_0$ **13**, 17, 87, 101, 110, 135, 142, 148, 156, 168, 217, 218, 219, 228

Struc$_0$ 14, **15**, 71, 112

Subsignal-definition$_1$ **Z.100**, 66

Subsignal$_0$ **10**, 66

SynD 21, 22, **26**, 46, 93, 139, 140, 160, 185

Syn-type-definition$_1$ **Z.100**, 30, 90, 226

Synonymdef$_0$ **16**, 17, 46, 49, 52, 93

Synonymname$_0$ **16**

SyntypeD 21, 22, **26**, 74, 90, 117, 164, 224, 256, 258

Syntypedef$_0$ **14**, 17, 50, 52, 71, 90

Syntypename$_0$ **14**

Sys$_0$ **3**, 32, 33, 261

Sysdecl$_0$ **3**

Sysdef$_0$ **3**, 33

Sysname$_0$ **3**

SystemD 21, 22, **23**, 33

System-definition$_1$ **Z.100**, 33, 226, 262

System-qualifier$_1$ **Z.100**, 223

Tailid$_0$ **3**, 4, 5, 10, 11

Tailname$_0$ **3**, 5, 7, 12, 14, 15

Task-node$_1$ **Z.100**, 164, 200, 201, 206

Task$_0$ **8**, 183, 198, 199

Term-Information **30**, 259

Term$_0$ **13**, 14, 16, 85, 87, 89, 103, 134, 135, 147

Term$_1$ **Z.100**, 83, 103, 134, 135, 137, 141, 144, 147, 148, 154, 155, 213

Terminator$_0$ **8**, 238

Termparm$_0$ **15**, 70, 89

Termstmt$_0$ **7**, **8**, 167, 168, 170, 175, 181, 184, 188, 195, 196, 198, 199, 203, 204, 205, 206, 207, 208, 212, 214, 231, 237, 238

Text$_0$ **3**, 8, 199, 200

Time-information **30**

TimerD 21, 22, **24**, 63, 116, 152, 180, 192, 203, 205, 208, 224

Timer-active-expression$_1$ **Z.100**, 152

Timer-definition$_1$ **Z.100**, 30, 116, 226

Timerdef$_0$ 4, **9**, 11, 41, 52, 116

Timerelem$_0$ **9**, 116

Timerid$_0$ **9**, 18

Timername$_0$ **9**

Transition$_0$ **7**, 9, 11, 12, 23, 28, 166, 167, 169, 170, 175, 181, 182, 184, 186, 187, 188, 190, 192, 195, 106, 221, 231, 234, 238

Transition$_1$ **Z.100**, 163, 164, 193, 195, 198, 199, 203, 204, 205, 206, 207, 208, 212, 214, 217, 220, 235, 239

Tupleexpr$_0$ **17**, 149, 153

Unquantequation$_0$ **13**, 14, 102, 105

Unquantified-equation$_1$ **Z.100**, 82, 102, 105

Validinputset 23, **25**, 240

Value$_0$ 6, **14**

ValueidD 21, 22, **27**, 104, 106, 137, 141, 148

Valuename$_0$ **13**, 17

VarD 21, 22, **26**, 69, 117, 120, 140, 158, 160, 163, 164, 189, 190, 193, 200, 216, 224, 246

Vardef$_0$ 4, 5, **6**, 11, 41, 52, 64, 116, 117, 118, 229, 230

Vardefelem$_0$ **6**, 64, 116, 118, 229, 230

Variable-definition$_0$ **Z.100**, 30, 117, 118, 226, 229

Variable-identifier$_1$ **Z.100**, 192, 193

Variable$_0$ **18**, 201

Varid$_0$ **6**, 7, 12, 18, 151, 182, 183, 184, 193, 206

Varname$_0$ 4, 5, **6**, 12

Via$_0$ **8**, 208, 211

ViewD 21, 22, **27**, 120

View-definition$_1$ **Z.100**, 30, 119, 120, 226

View-expression$_1$ **Z.100**, 151

Viewdef$_0$ 4, **6**, 11, 41, 52, 119

Viewdefelem$_0$ **6**, 119

Viewexpr$_0$ **18**, 149

Viewqualelem, **21**

Xquerynm, **281**

Xreplynm, **261**

Указатель функций

add-default-assign, 163, **164**
add-equality, 94, **107**
add-ordering, 94, **108**
add-service-varinit, 235, **239**
add-varinit, 162, **163**, 239
all-exporteurs, 242, **247**
all-importeurs, 242, **247**
all-input-signals, **156**, 177, 178, 231, 237
all-variables-and-synonyms, 139, **140**, 160
all-visible-literals, 137, 139, 141, 142, **156**
all-visible-operators, 92, 143, **144**
all-visible-sorts, 81, 93, 103, **155**, 200, 206, 217, 220
apply-generic-parameters, 33, **261**
as$_0$-channeldefs, **244**
as$_0$-channels, 242, **244**
as$_0$-extract-properties, 113, **115**
as$_0$-global-entities 33, **35**
as$_0$-id, 58, 73, 107, 109, 110, 118, 141, 177, **229**, 230, 231, 237, 244, 250, 251, 252
as$_0$-implicit-transitions, **175**
as$_0$-inputvars, 175, 178, **180**
as$_0$-modify-properties, 113, **114**
as$_0$-order, 108, **110**
as$_0$-ordering-axioms, 108, **109**
as$_0$-ordering-typing, 108, **110**
as$_0$-xtQUERY-inputs, 189, **190**

block-connect, 125, **126**
build-answer-operator, 221, **222**
build-answerlist-labeldict, 170, **171**
build-cont-trans, 182, **184**
build-enable-decision, 186, **187**
build-exported-vardef, 117, **118**
build-extract-operator, 143, **145**
build-field-operator, 143, **145**
build-implicit-operators, 77, **80**
build-implicit-state, 187, **188**
build-literal-renaming, 75, **76**
build-operator-renaming, 77, **78**
build-quant-equation, 102, **104**, 105
build-service-descriptor, 52, **233**
build-service-statedict, 233, **234**
build-spec-labeldict, **169**
build-state-labeldict, 162, **169**, 234
build-statedict, 162, **172**, 234
build-trans-labeldict, 162, 169, **170**, 171, 234

check-name-syntax, 96, **262**
collect -genparms, 84, **89**
collect -illegal-synonyms, 43, **49**
collect-priority-info, 182, **185**
collect-sorts, 43, **50**
collect-synonyms, 45, **46**
convert-axiom, 81, **82**
convert-channel-qual, 126, 129, 224, **225**
convert-term, 82, **83**
create-unique-name, 33, 35, 56, 58, 66, 69, 71, 79, 80, 90, 94, 107, 109, 111, 114, 115, 116, 117, 150, 168, 181, 188, 231, 235, 237, 242, 250, 252, 261, **262**

definition-of -SDL **32**
derive-delay-inf, 258, **261**
derive-time-inf, 258, **261**
double-states, 235, **237**

emptyqtrans, 181, 182, **184**
eval-expr, 44, **261**
eval-option-trans, 220, **221**
eval-simple-expr, **44**, 61, 185, 221, 260
expand-continuous, 181, **182**
expand-enable, 181, **186**
extract-exports, 157, **158**
extract-firststate-or-service, 235, **238**
extract-inherited-axioms, 73, **81**
extract-initial-state, 195, **196**
extract-legal-operators, 143, 144, **146**
extract-priinput, **234**
extract-servicestate, 195, **197**

father-block-import-export, 242, **245**
form-integer, **98**, 99, 100
form-names-and-strings, **96**

gen-connect, 253, **254**
gen-connect-elem, 254, **255**
gen-formparm-unique, **70**
generate-auxiliary-information, 33, **258**
get-answer-sort, 217, **218**, 220
get-exporteurs, 247, **248**
get-inherited-parent, 73, **258**
get-parent, 46, 64, 69, 79, 80, 90, 93, 111, 116, 119, 121, 139, 140, 143, 144, 150, 151, 151, 156, 193, 200, 216, **256**
get-predef-sort, 44, 47, 91, 92, 107, 109, 110, 148, 151, 152, 153, 154, 205, 208, 229, **256**, 259, 260
get-sur, 63, 67, 71, 78, 116, 120, 128, 129, 131, 137, 139, 142, 144, 150, 155, 157, 168, 160, 163, 208, 210, 211, 212, 223, 224, 225, 228, 229, 240, 245, 246, 247, 253, 256, **257**
get-visible-qual, 40, 57, 60, 64, 65, 69, 73, 84, 90, 93, 104, 106, 111, 116, 119, 121, 122, 124, 126, 129, 133, 139, 152, **158**, 159, 203, 205, 210, 212, 214, 241
get-visible-variable, **160**, 193, 200, 206, 216

imp-exp-in-processes, 242, **246**, 247

implicit-channels-and-signal-routes, 54, **242**
implicit-connects, 242, **253**
implicit-signalroutes, 242, **249**
import-export-signals, 63, **157**, 175, 177, 178, 250, 252
initialdatadef, 54, **56**, 61, 67
insert-answer-term, 166, **167**
insert-equals-true, 101, **102**, 105
insert-genparms, 84, **85**
insert-importact, 188, **193**
insert-join, 167, **168**
insert-parm, 85, **87**
insert-spec-term, **165**
insert-starred, 172, **173**
insert-state-names, 172, **173**
insert-state-term, 162, **165**, 234
insert-trans-term, 162, 165, **166**, 167, 234
is-consistent-Dict **22**
is-in-regular-expr, 96, **97**, 100
is-in-regular-par, 97, **100**
is-in-regular-range, 97, **98**
is-in-regular single, 97, **99**
is-local, 57, 63, 122, 139, 235, 249, 251, **256**
is-recursive-sort , 73, **74**, 90
is-wf-connectchannels, 129, **130**
is-wf-connectsignalroutes, 126, **127**, 133
is-wf-entities, 39, **41**
is-wf-refinerment, 130, **131**
is-wf-servicesignals, 161, **240**
is-wf-simple-expr, 44, **47**
make-as$_0$-channel-signalroutes, 251, **252**
make-as$_0$-env-signalroutes, 249, **251**
make-as$_0$-local-signalroutes, 249, **250**
make-as$_1$-concaxioms, 73, 94, **228**
make-as$_1$-identifier, 56, 57, 64, 65, 69, 73, 75, 77, 78, 81, 90, 91, 92, 94, 104, 111, 116, 117, 120, 122, 126, 129, 139, 141, 142, 147, 150, 151, 152, 164, 192, 193, 200, 203, 205, 206, 208, 212, 214, 216, **223**, 225, 259
make-as$_1$-qual, **223**
make-as$_1$-typing, 77, **78**
make-as$_1$idset, 57, 122, 189, 208, **225**
make-as$_1$tree, 33, 54, 61, 67, **226**
make-expired-function, 258, **260**
make-implicit-decl, 72, **73**
make-implicit-import-vardef, 229, **230**
make-implicit-sigdecls, 71, **72**
make-implicit-vardecl, 61, **229**
make-new-qual, 223, **224**
make-term-inf, 258, **259**
make-tick-function, 258, **259**
match-option-answer, **221**
merge-states, 235, **239**
pretrans, 181, **183**
process-level, 63, 150, 157, 158, 189, 210, 212, **257**
process-or-service-level, 67, 150, 151, 156, 175, 208, 210, **257**

remove-asterisk-from-spec, 175, **177**, 178
remove-asterisk-from-state, 174, **175**
remove-asterisk-input-and-save, 162, **174**, 234
remove-cont-enable-from-state, 180, **181**
remove-cont-enable-from-statelist, 162, **180**, 235
remove-references, 35, **37**, 38, 39
remove-select, 33, **43**, 48
remove-select-in-decllist, 43, **44**
remove-select-in-enclosed-scopeunit, 43, **48**
rename-operator, 78, **79**
repeat-collecting-synonyms, 43, 44, **45**
replace-connects, 58, **60**
replace-references, 33, **35**

select-consistent-subset, 32, **262**
select-remote-number-of-instances, 37, **41**
service-connect, 132, **133**
service-connectmap, **132**, 161
signal-qual, 124, **159**, 178, 192, 208, 234
signaldeflevel, 159, **160**
sorts-have-values , 226, **227**
transform-act, 195, **198**
transform-activeexpr, 149, **152**
transform-actparmlist, 214, **215**
transform-actparms, 147, 152, 203, 205, 208, 212, **213**
transform-answers, 217, **219**
transform-assign, 199, **200**, 201
transform-axioms, 100, **101**, 228
transform-axiom-id, 135, **137**
transform-axiom-id-of-this-sort, 137, **141**
transform-axioms, 94, **100**, 104, 106
transform-block-connect, 54, **125**
transform-block-substructure-connect, 128, **129**
transform-blockdef, 52, **54**
transform-body, 67, 161, **162**
transform-build-in-expression, 135, **149**
transform-call, 198, **214**
transform-channel-sub, 57, **58**
transform-channeldef, 52, **57**
transform-condequation, 101, **105**
transform-condexpr, 135, **154**
transform-create, 198, **212**
transform-decision, 198, **217**
transform-decl, 51, **52**
transform-decllist, 33, **51**, 54, 59, 61, 67, 72, 161, 233
transform-decomposition-body, 161, **235**
transform-equation, 101, **102**

transform-export, 198, **206**
transform-expr, 44, 46, 47, 90, 92, 93, 94, 117, 134, **135**, 139, 151, 152, 153, 154, 155, 200, 205, 208, 213, 215, 217, 220
transform-field, **113**
transform-fields, 112, **113**
transform-geninst, 70, 71, **84**
transform-id, 135, **139**
transform-import, 199, 203, 205, 208, 212, 214, 217, **231**
transform-importdef, 52, **121**
transform-importexpr, 149, **150**
transform-inflxexpr, 135, **155**
transform-informal, 199, **200**
transform-inherited, 71, **73**
transform-inoutactparm, 215, **216**
transform-input, 189, **191**
transform-input-transition, 191, **192**
transform-inputvarlist, 192, **193**
transform-inputvars, **192**
transform-lhs-rhs, 102, **103**, 105
transform-literal-renaming, 73, **75**
transform-mapping, 101, **106**
transform-mappingaxioms, **106**
transform-modifyassign, 200, **201**
transform-monadexpr, 135, **155**
transform-nameclass, 94, **96**
transform-nowexpr, 149, **151**
transform-operator-renaming, 73, **77**
transform-operatorexpr, 135, **142**, 143
transform-operatorterm, 135, **144**
transform-option, 198, **220**
transform-output, 198, **207**
transform-output-elems, 206, 207, **208**
transform-partial-typedef, 52, **71**
transform-pid-build-in, 149, **153**
transform-prioutput, 198, **206**
transform-proceduredef, 52, **66**
transform-procedureparml, 67, **68**
transform-process-body, 61, **161**
transform-processdef, 52, **61**
transform-processparm, 61, **64**
transform-qual-operator, 143, 144, 146, **147**
transform-quantequation, 101, **104**
transform-refinement, 65, **66**
transform-reset, 198, **203**
transform-reset-elems, **203**
transform-restrictions, **105**
transform-selectexpr, 143, 149, **152**
transform-servicesigroutedef, 52, **241**
transform-set, 198, **204**
transform-set-elems, 204, **205**
transform-signaldef, 52, **65**, 66
transform-signallist, 57, 63, 122, 123, **124**, 177, 189, 241
transform-signallistdef, 52, **123**
transform-signalroutedef, 52, **122**
transform-sortdef, 71, 73, **94**, **112**
transform-sortgenerator, 52, **70**
transform-spelling, 135, **148**
transform-statelist, 162, **188**, 235
transform-statespecl, 188, **189**
transform-stmtlist, 198, **199**
transform-stringexpr, 135, **142**, 148
transform-struc, 71, **112**
transform-substructure-connect, 54, **128**
transform-synonymdef, 52, **93**
transform-syntype, 52, 71, **90**
transform-system 32, **33**
transform-term, 103, **134**
transform-timerdef, 52, **116**
transform-transition, 162, 192, **195**, 199, 203, 205, 208, 212, 214, 217, 219, 220, 235
transform-tupleexpr, 149, **153**
transform-typing, 94, **111**
transform-validinputset, 61, **63**, 233
transform-valuerange, 91, **92**
transform-valueset, 90, **91**, 218, 219
transform-vardef, 52, 64, **116**, 117, 118, 229
transform-varparm, 68, **69**
transform-via, 208, **210**
transform-view, 119, **120**
transform-viewdef, 52, **119**
transform-viewexpr, 149, **151**

processor *system* **Annex F.3**, 32

Указатель ключевых слов

ALL 15, 77
AND 14, 92, 107, 109, 222
AXIOMS 30, 94, 135, 147, 154, 155, 228

BIOCK 2, 21, 22, 38, 41, 48, 54, 57, 58, 60, 223, 226

CHANNEL 21, 22, 57, 58, 126, 128, 129, 210, 225
CONC 14, 228
CONSTANT 30, 44, 46, 47, 90, 92, 93, 94, 117, 139, 140, 220

DATATYPEDEF 28, 33, 54, 56, 61, 67, 73, 94, 226
DIV 14

ENV 5, 6, 23, 24, 25, 41, 57, 122, 125, 127, 128, 130, 132, 158, 229, 242, 252, 255, 256
ENVIRONMENT 57, 122
EQ 14, 92, 107, 109, 184, 222
EXCLAMATION 113, 114, 115, 145, 152, 153, 201
EXCLAMATIONMARK 2
EXPORT 160
EXPORTED 6, 26, 158, 246
EXPRESSION 30, 139, 149, 151, 152, 160, 200, 203, 205, 208, 212, 215, 217

GE 14, 92, 109, 110, 222, 260
GENERATOR 21, 22, 70, 84
GLOBALNAMES 28, 33, 157, 181, 229
GT 14, 92, 109, 110, 222

IMPLIED 28, 61, 150, 229, 231
IMPLY 14, 107, 109
IMPORT 21, 22, 121, 150, 157
IMPORTLIST 28, 150, 199, 203, 205, 208, 212, 214, 217, 231
IN 14

LABELDICT 28, 162, 191, 195, 235
LE 14, 92, 109, 110, 222
LEVEL 52
LITERAL 21, 22, 76, 94, 156, 228, 259
LT 14, 92, 109, 110, 222

MAPPING 30, 94, 106, 135, 147, 148, 154, 155
MINUS 14, 17
MOD 14
MULT 14, 16, 98, 99, 100

NE 14, 92, 107, 222
NOT 14, 17, 18, 107, 109

OPERATOR 21, 22, 79, 80, 91, 92, 111, 144, 259
OR 14, 91, 109, 222
OUTSIGNALS 28, 61, 208

PLUS 14, 16, 99, 100, 183, 259
PROCEDURE 2, 22, 39, 41, 66, 67, 214, 223, 226, 257
PROCESS 2, 22, 39, 41, 48, 61, 122, 212, 223, 226

REM 14
REVEALED 6, 26, 120
REVERSE 10

SCOPEUNIT28 33, 43, 46 48, 54, 56, 57, 58, 61, 63, 65, 66, 67, 69, 70, 71, 90, 92, 93, 94, 102, 104, 106, 111, 112, 116, 117, 120, 121, 122, 123, 125, 126, 128, 129, 132, 133, 135, 137, 139, 140, 141, 143, 144, 150, 151, 155, 157, 158, 159, 162, 163, 175, 177, 178, 184, 185, 189, 190, 191, 195, 196, 197, 208, 210, 212, 224, 227, 231, 233, 234, 235, 239, 240, 241, 244, 245, 246, 247, 248, 249, 251, 252, 253, 256
SERVICE 2, 22, 39, 41, 48, 233, 241, 257
SERVICES 28, 195, 196, 197, 235, 238
SIGNAL 2, 21, 22, 65, 116, 152, 157, 158, 159, 160, 203, 205, 223, 231
SIGNALLIST 21, 22, 123, 124
SIGNALROUTE 21, 22, 122, 125, 126, 132, 133, 210, 241, 251
STATEDICT 28, 162, 188, 195, 231, 235
SUBSTRUCTURE 2, 22, 38, 39, 41, 48, 54, 58, 223, 226
SYSTEM 2, 21, 22, 33, 41, 49, 223, 226

TYPE 2, 22, 46, 56, 64, 65, 69, 71, 73, 90, 93, 94, 104, 106, 111, 114, 115, 116, 119, 121, 223, 256

VALUE 21, 22, 43, 46, 69, 93, 104, 106, 117, 120, 137, 139, 140, 141, 148, 150, 151, 158
VIEW 21, 22, 120, 151

XOR 14

Сообщения об ошибках

§2.10.2: Недопустимые внешние сервисы VIA, когда определены маршруты сигналов сервиса 210
§2.2.2: Символьная строка не относится к соответствующему сорту 142
§2.2.2: Выражения условия не соответствуют контексту 154
§2.2.2: Конечное имя в определении блока отличается от определяющего имени 54
§2.2.2: Конечное имя в определении подструктуры блока отличается от определяющего имени 54
§2.2.2: Конечное имя в определении канала отличается от определяющего имени 57
§2.2.2: Завершающее имя в определении подструктуры канала отличается от определяющего имени 58
§2.2.2: Завершающее имя в определении процедуры отличается от определяющего имени 67
§2.2.2: Завершающее имя в определении процесса отличается от определяющего имени 61
§2.2.2: Завершающее имя в определении сервиса отличается от определяющего имени 233
§2.2.2: Завершающее имя в определении системы отличается от определяющего имени 33
§2.2.2: Идентификатор не видим 158
§2.2.2: Локальные синоним или переменная не относятся к правильному сорту 139
§2.2.2: Многократное появление определения оператора 111
§2.2.2: Имена формальных параметров должны быть отличны от от имен переменных 61,67
§2.2.2: Не существует подходящий сорт для действия решения 218
§2.2.2: Нет импортируемых переменных, удовлетворяющих контексту 150
§2.2.2: Для этого идентификатора значений не может быть использован ни один сорт 137
§2.2.2: Некоторые переменные и синонимы допустимы для этого контекста 139
§2.2.2: Идентификатор обозначает более чем один (под) сигнал 159
§2.2.2: Синоним или генератор или список сигналов определены рекурсивно 158
§2.2.2: Терм оператора не может быть использован в этом контексте 143, 144
§2.2.2: Два определения в одной единице подразделения и одном классе объектов определяют одно имя 51, 66
§2.2.2: Два определения в одной единице подразделения используют одно и то же имя 64, 65, 68, 69, 117, 119, 121
§2.2.3: Начальный номер больше максимального номера экземпляра 61
§2.2.3: Начальный номер экземпляров меньше нуля 61
§2.2.3: Максимальный номер экземпляров равен нулю 61
§2.4.1: Определяющие имена могут быть квалифицированы только в удаленных описаниях 54, 58, 61, 67, 233
§2.4.1: Открывающий и заканчивающий идентификаторы в определении различны 37, 38
§2.4.1 Нет удаленных определений, соответствующих ссылке 37, 38
§2.4.1 В системном определении не ссылаются на удаленные определения 35
§2.4.1 Удаленные определения не являются уникальными 35
§2.4.2 Определение системы должно содержать по крайней мере один блок 34
§2.4.3 Блок должен содержать либо один или более процессов, либо описание подструктуры 54
§2.4.4 Удаленное число экземпляров спецификации не соответствует ссылке процесса 41
§2.4.5 Переменная в процедуре (или сервисе) не может быть EXPORTED или REVEALED 117
§2.5.2 Конечные точки маршрута сигнала не являются различными 122
§2.5.2 Конечные точки маршрута сигнала не определены локально 122
§2.5.2 Второй путь должен обозначать обратное направление первого пути 122
§2.5.2 Допустимый входной сигнал должен содержать все локальные сигналы и не должен содержать сервисы 63
§2.5.2 Совокупность допустимых входных сигналов должна быть специфицирована, когда не специфицированы маршруты сигналов 249
§2.5.3 Соединение маршрут-сигнал встретилось дважды 125
§2.5.3 Соединение маршрут-сигнал не корректно 126
§2.5.3 К блоку не подсоединены каналы 125
§2.5.3 Маршрут сигнала появляется не только в одном соединении 125
§2.5.3 Маршрут сигнала упоминается более чем в одном соединении 125
§2.5.5 Идентификаторы сигналов в списке сигналов не различаются 124
§2.5 Окончание канала определяется в другой единице подразделения, отличной от канала 57
§2.5 Окончания канала должны быть различны 57
§2.5 Второй путь в определении канала должен обозначать обратное направление первого пути 57
§2.6.1.2 Отсутствуют уникальные соответствующие переменные этого сорта с атрибутом раскрытия 120

§2.6.3: Имя, завершающее состояние, должно быть таким же, как и открывающее 172
§2.6.3: Имя, завершающее описание, не допустимо в состоянии со звездочкой 172
§2.6.3: Сигналы состояния не согласуются 178, 178
§2.6.3: Имена состояний в состоянии должны быть уникальными 173
§2.6.4: Неправильное число параметров в вершине ввода 192
§2.6.6: Метки не являются уникальными 170
§2.6.7.1: Терминатор потерян при переходе 166, 168
§2.6.7.2.1: Имя в следующем состоянии должно обозначать состояние 195, 196, 197, 238
§2.6.7.2.2: Метка в перескоке не определена 195
§2.7.2: Длина фактических параметров должна соответствовать длине списка формальных параметров 212
§2.7.2 Созданный процесс должен быть определен в том же блоке 212
§2.7.3: Фактический параметр IN/OUT должен быть идентификатором переменной 216
§2.7.3: Длина списка параметров процедуры должна соответствовать длине списка формальных параметров 214
§2.7.3: Один и тот же сорт должен быть специфицирован для фактического и формального параметра IN/OUT 216
§2.7.4: Канал в множестве VIA не подсоединен к блоку 211
§2.7.4: Идентификатор в действии вывода должен обозначать сигнал 208
§2.7.4: Если маршруты сигналов специфицированы, то тогда канал не может быть задействован в VIA 211
§2.7.4: Недопустимое множество VIA 210
§2.7.4: Неправильное число аргументов в действии вывода 208
§2.7.4: Сигнал не может быть передан по каналу в множестве VIA 211
§2.7.4: Для сигнала на выходе нет допустимого приемника 210
§2.7.4: В множестве VIA дважды определены маршрут сигналов или канал 210
§2.8: Идентификатор в действии сброса не является таймером 203
§2.8: Идентификатор в действии установки не является таймером 205
§2.8: Идентификатор в выражении активного таймера не обозначает таймер 152
§2.8: Недопустимое выражение активного таймера 152
§2.8: Длина списка параметров в действии сброса должна соответствовать длине, указанной в определении 203
§2.8: Длина списка параметров в действии установки должна соответствовать длине, указанной в определении 205

§3.2.2: Подструктура блока не содержит определение блока 54
§3.2.2: Соединение канала встретилось дважды 128
§3.2.2: Соединение канала не является корректным 129
§3.2.2: Для канала, соединенного с подструктурой блока, нет соединений 128
§3.2.2: Подканалы появляются не только в одном соединении 128
§3.2.2: Подканал задействован более чем в одном соединении 128
§3.2.3: Идентификаторы блоков в соединениях не обозначают конечные точки канала 60
§3.2.3: Должно быть ровно два соединения конечных точек канала 60
§3.3: Процесс использует сигналы на различных уровнях обработки того же самого сигнала 61

§4.10.1: Идентификатор маршрута сигналов сервиса дважды встретился в соединении маршрута сигнала сервиса 133
§4.10.1: Сигналы в маршрутах сигналов сервиса должны быть теми же как для подсоединенного маршрута сигналов 133
§4.10.1: Конечные точки в направлениях двунаправленного маршрута сигналов должны совпадать 241
§4.10.1: Конечные точки маршрута сигналов сервиса должны быть различны 241
§4.10.2: Все маршруты сигналов сервиса должны быть задействованы в соединении 132
§4.10.2: Неправильное соединение маршрута сигнала сервиса 132
§4.10.2: В процессе утеряно соединение маршрута сигнала 132
§4.10.2: Больше чем один сервис содержит строку начального перехода 238
§4.10.2 В VIA нет сигналов сервиса, связанных с маршрутом сигналов 210
§4.10.2: Процедура в декомпозиции или сервисе содержит значения состояния или выражения импорта 67
§4.10.2: В процессе содержатся определения сервиса и таймеров 116
§4.10.2: Сигналы сервиса не корректны 161
§4.10: Декомпозиция должна содержать по крайней мере одно определение сервиса 235
§4.10: Недопустимое использование сигнала с высоким приоритетом 208
§4.11: Только один непрерывный сигнал может быть специфицирован, если пропущен приоритет 182
§4.11: Несколько непрерывных сигналов с одним приоритетом 185
§4.2.3: Два терма в равенстве должны быть одного сорта 103

§4.3.2: Простое выражение не является предопределенным сортом или содержит неопределенные идентификаторы 44
§4.3.3: Определения в отборе не допустимы в данной единице подразделения 39
§4.3.3: Выбранное определение недопустимо в этой единице подразделения 48
§4.3.4: Более чем один ответ выбора соответствует запросу 221
§4.3.4: Ни один ответ выбора не соответствует запросу 221
§4.4: Недопустимое состояние со звездочкой 173
§4.7: У состояния более одной звездочки для ввода или сохранения 177, 178
§4.9: Следующее состояние с черточкой в начальном переходе 195

§5.2.1: Завершающее имя в описании частичного типа отлично от определяющего имени 71
§5.2.1: Сорт в определении типа данных не имеет значений 226
§5.2.1: Нет операторов, которые возвращают значение этого сорта 71
§5.2.2: В определении частичного типа литерал определен дважды 94
§5.2.2: Оператор или литерал определены явно и согласно наследованию 73
§5.2.3: Несовместимое использование неявно квантифицированного имени значения 141
§5.2.3: Идентификатор значения не должен быть квалифицирован 141
§5.2: Длина списка параметров не является корректной для оператора 147
§5.4.1.10: У двух полей структуры одинаковые имена 113
§5.4.1.11: Литерал определен дважды при переименовании 76
§5.4.1.11: Литерал в переименованном литерале не определен в родительском сорте 76
§5.4.1.11: Литерал переименован дважды 76
§5.4.1.11: Оператор при переименовании оператора не определен в родительском сорте 78
§5.4.1.11: Оператор переименован дважды 78
§5.4.1.11: Оператор с восклицательным знаком упоминается при переименовании оператора 78
§5.4.1.11: Сорт основан на себе самом 73
§5.4.1.12.2: Фактический параметр генератора CONSTANT должен быть термом 89
§5.4.1.12.2: Фактический параметр генератора LITERAL должен быть сигнатурой литерала 89
§5.4.1.12.2: Длины списков фактических и формальных параметров должны быть одинаковы 84
§5.4.1.12.2: Фактический параметр генератора OPERATOR должен быть сигнатурой оператора 89
§5.4.1.12.2: Фактический параметр генератора TYPE должен быть идентификатором 89
§5.4.1.12: Завершающее имя в определении генератора отличается от определяющего имени 70
§5.4.1.12: Параметр константы генератора используется в сигнатуре литерала 87
§5.4.1.12: Формальное имя генератора квалифицировано 87
§5.4.1.12: Класс имен не может быть использован в равенстве 87
§5.4.1.13: Сорт синонима не может быть однозначно определен 93
§5.4.1.14: Длина символьной строки в регулярном интервале не равна единице 98
§5.4.1.14: Имя повторения в классе имен не должно обозначать натуральное число 98, 99, 100
§5.4.1.15: Недопустимое использование оператора SPELLING 148
§5.4.1.7: Терм-ошибка является частью ограничений 105
§5.4.1.7: Терм-ошибка используется в составном терме 135
§5.4.1.9.1: Оператор упорядочивания не определен для сорта диапазона условий 92
§5.4.1.9: Завершающее имя в определении подтипа отличается от имени определения 90
§5.1.4.9: Подтип определен в собственных термах 256, 258
§5.5.3.3: Более чем одно присвоение по умолчанию 84
§5.5.3: Нет или несколько раскрытий измененных присваиваний 201
§5.5.4.1: Выражение NOW должно использоваться там, где допустимы значения TIME 151
§5.5.4.3: Выражение PID используется в неправильном контексте 153
§5.5.4: Императивный оператор не может быть использован в выражении константы 149