Российская академия наук
Сибирское отделение
Институт систем информатики
им. А. П. Ершова

Т. Г. Чурина

СПОСОБ ПОСТРОЕНИЯ РАСКРАШЕННЫХ СЕТЕЙ ПЕТРИ, МОДЕЛИРУЮЩИХ SDL-СИСТЕМЫ

Препринт
56

Новосибирск 1998

Настоящая работа посвящена исследованию проблемы автоматического построения сетевых моделей SDL-спецификаций распределенных систем. Язык спецификаций и описаний SDL принят в качестве международного стандарта. Рассматриваются SDL-системы с таймерами, средством сохранения сигнала и приоритетами, позволяющие адекватно представлять значительный класс коммуникационных протоколов. В качестве моделей выбраны раскрашенные сети Петри, предложенные Йенсеном, которые расширяются посредством семантик времени в смысле Мерлина и приоритетов. В работе представлен метод трансляции SDL-систем в данную сетевую модель.

Siberian Division of the Russian Academy of Sciences
A. P. Ershov Institute of Informatics Systems

T. G. Churina

COLOURED PETRI NETS APPROACH TO THE VALIDATION OF SDL-SPECIFICATIONS

Preprint
56

Novosibirsk 1998

This work is devoted to research dealing with automated constructing net models of SDL specifications. In the paper is considered SDL specifications with timers and priorities. The specifications allows to represent a considerable class of communication protocols. Coloured Petri nets extended by priorities and Merlin's time concepts are used as the net models. This work describes a method of translating the SDL specification into the net models.

ВВЕДЕНИЕ

В последние годы прогресс средств передачи и обработки информации привел к стремительной разработке большого количества коммуникационных протоколов. Протокол — это распределенная асинхронная система, обнаружение ошибок в которой является важной проблемой современного программирования. Лаборатория теоретического программирования Института систем информатики СО РАН принимает участие в международном научном проекте ИНТАС—РФФИ № 95-0378, в рамках которого, в частности, исследуются методы анализа и верификации протоколов, записанных на стандартном языке SDL.

Полезные методы трансляции SDL-спецификаций в обобщенные сети Петри, такие как SDL- и PRT(predicat-transition)-сети, описаны в работах [7, 14, 15], в которых используются новые классы сетей Петри высокого уровня и предложены методы построения графа достижимости. В работе [15] представлена техника анализа эффективности для SDL-сетей, однако поскольку графы достижимости весьма громоздки, обычные способы их обработки неэффективны.

В книге Йенсена [12] поставлена проблема автоматического построения сетевых моделей SDL-спецификаций, развития средств их верификации, а также проведения экспериментов по обнаружению семантических ошибок распределенных систем с помощью этих средств. Наш подход состоит в автоматическом переводе SDL-спецификаций в раскрашенные сети Йенсена [10], обогащенные временем. Выбор раскрашенных сетей Петри в качестве модели обусловлен развитым теоретическим аппаратом, положительным опытом использования, а также доступной системой симуляции и анализа Design/CPN [12]. Кроме того, в ИСИ СО РАН реализована экспериментальная система NetCalc, которая позволяет создавать и редактировать модифицированные иерархические раскрашенные сети, обогащенные временным механизмом и приоритетами, а также проводить их симуляцию [6].

Настоящая работа посвящена исследованию данной проблемы. Рассматривается язык SDL [1] без динамических конструкций. Такие спецификации позволяют адекватно представлять значительный класс коммуникационных протоколов. Данная работа состоит из введения, 8 разделов и заключения. Первый раздел содержит описание используемых конструкций языка SDL. Второй посвящен описанию раскрашенных се-

[1]Данная работа частично поддержана грантом ИНТАС—РФФИ № 95-0378

тей Петри, а также их расширения временным механизмом. В третьем разделе изложены основные принципы трансляции SDL-систем и описана трансляция верхнего уровня. В четвертом представлена трансляция блока, состоящего как из подблоков, так и из процессов. Пятый раздел содержит описание трансляции процесса, шестой — перехода. В седьмом разделе содержится описание моделирования операторов языка SDL, процедур, а также обозреваемых и экспортируемых переменных, в восьмом — характеристика полученной сетевой модели и оценка ее размера.

1. КРАТКОЕ ОПИСАНИЕ ЯЗЫКА SDL

SDL (Specification and Description Language) — язык спецификации и описаний — разработан бывшим Международным консультативным комитетом по телеграфии и телефонии (CCITT). Язык предназначен для описания структуры и функционирования систем реального времени, в частности сетей связи. SDL построен на базе модели конечного автомата по объектно-ориентированной схеме.

Самый общий объект, описываемый на SDL, называется *системой*. Все остальные объекты находятся на более низких уровнях иерархии определений. Все, что не вошло в описание системы, называется окружением (внешней средой) системы. Каждая система должна иметь уникальное имя.

Важной особенностью языка SDL является концепция типов данных, в основу которой положена алгебраическая модель. Все виды данных, используемые в конкретной системе, рассматриваются как компоненты единого типа. Элемент типа данных называется значением. Все значения типа данных определяют множество, на котором задаются операторы. Константа (или литерал в SDL) является 0-местным оператором; n-местный оператор (где $n \geq 1$) определяется своей сигнатурой, состоящей из имени оператора, типа результата и типов параметров. Действие оператора задается алгебраическими правилами — аксиомами. Совокупность множества значений типа данных, операторов со своими сигнатурами и аксиом образует алгебраическую систему. В любой SDL-системе существуют предварительно определенные типы (сорта в SDL), такие как целые и вещественные числа, символы, строки. Другие сорта, например различные виды массивов, генерируются по шаблонам с помощью уже определенных сортов, операторов и аксиом.

В SDL определены две синтаксические формы описания систем. Од-

на форма — текстовая, совпадающая с формой описания обычных языков программирования, другая — графическая, в которой система описывается в виде диаграмм, состоящих из графических символов. Текстовая форма описания является более богатой, графическая — более наглядной.

Система состоит из одного или нескольких *блоков*, соединенных между собой и с окружающей средой *каналами*, по которым передаются *сигналы*. Из окружения система получает внешние сигналы и в окружение возвращает ответы, запуск системы возможен только по сигналу извне. Каждый блок и канал в системе имеют уникальное имя.

Каналы бывают одно- или двусторонними. При каждом канале должны быть указаны имена всех сигналов, которые этот канал может передавать. При сигнале могут быть указаны имена сортов, таким образом, сигнал может нести с собой значения указанных сортов. Несколько сигналов могут быть объединены в один список, такому списку присваивается уникальное имя. Один сигнал может входить в разные списки.

Средствами SDL обеспечивается многоуровневое описание системы. По мере "спуска"от уровня к уровню либо детализируются описания уже имеющихся в системе объектов, либо вводятся новые объекты. Блок может быть разбит на более мелкие единицы — подблоки, которые, в свою очередь, сами являются блоками. Подблоки соединяются "новыми"каналами между собой и с рамкой блока. Будем называть "старыми"каналы, входящие или выходящие из блока. В разбиении блока для "старых"каналов должны быть указаны имена "новых"каналов, которые подсоединены к старым. Это подсоединение происходит таким образом, что каждый сигнал, поступивший по "старому"каналу, должен передаваться только по одному "новому"каналу, "новый"канал не может передавать сигнал, который не передавался по "старому"каналу. Аналогичными средствами осуществляется разбиение канала на подканалы и новые блоки.

На самом нижнем уровне иерархии определений блоки содержат *процессы*, являющиеся функциональными компонентами системы и определяющие ее поведение. Внутри блока *маршруты* связывают процессы между собой и с рамкой блокой. При графическом изображении маршруты, которые связывают процессы с рамкой блока, должны либо начинаться от той точки, в которой в блок входит канал, либо оканчиваться в той точке, в которой из блока выходит канал. К одному входному/выходному каналу могут быть присоединены начальные/конечные

точки нескольких маршрутов. Распределение сигналов по маршрутам, присоединенным к некоторому каналу, происходит таким образом, что по каждому маршруту должен передаваться хотя бы один сигнал, поступающий по каналу, а каждый сигнал, поступающий в канал, должен передаваться хотя бы по одному присоединенному маршруту. Кроме того, маршрут не может передавать сигнал, который не передавался по каналу.

Описание процесса состоит из трех частей: заголовка, декларативной части и тела процесса. Заголовок процесса содержит имя, число экземпляров и список формальных параметров с указанием их типов. Формальные параметры — это переменные, используемые в теле процесса. Число экземпляров — пара целых чисел, первое из которых указывает количество создаваемых экземпляров процессов в момент инициации системы, второе задает максимальное число экземпляров процессов, которые могут существовать одновременно в блоке. Так как по одному описанию процесса может быть создано несколько экземпляров, то каждый экземпляр должен получить личный идентификатор. Для этой цели каждый процесс обладает переменной $self$, которой при создании экземпляра процесса присваивается личный идентификатор этого экземпляра.

Каждый экземпляр процесса обладает еще тремя переменными: *parent*, $offspring$ и *sender*. Переменной *parent* порожденного процесса присваивается значение переменной $self$ родительского процесса. Переменной $offspring$ родительского процесса присваивается значение переменной $self$ его последнего отпрыска. Переменной *sender* присваивается личный идентификатор процесса-отправителя. В какой момент это осуществляется будет описано ниже, в п. 5.1.

Декларативная часть процесса содержит описания констант, типов, переменных, экспортируемых переменных, переменных обозревания, входных и выходных сигналов, списков сигналов, таймеров, процедур и макрокоманд.

Тело процесса описывает действия, которые совершает процесс под влиянием входных сигналов. Процесс либо находится в одном из своих состояний, либо совершает переход. При этом каждое состояние должно иметь свое уникальное по отношению к этому процессу имя. Процесс имеет одну стартовую вершину, за которой следует переход. Этот переход совершается не под влиянием входного сигнала, а в результате возникновения процесса. Дальнейшие переходы из состояния в состоя-

ние возможны только под воздействием входных сигналов, исключение составляют переходы, входящие в конструкцию "непрерывный сигнал".

Все сигналы, пришедшие в порт процесса, образуют в нем очередь. Процесс, находясь в одном из своих состояний, обращается к очереди сигналов. Если очередь пуста, то процесс ждет. Если не пуста, то для каждого состояния процесса однозначно указано, как должен реагировать процесс на любой сигнал, который стоит первым в очереди. Возможны три ситуации:

1) явно указано, какой переход должен совершить процесс. Тогда процесс удаляет из очереди сигнал и начинает указанный переход;
2) явно указано, что восприятие сигнала должно быть отложено до того, как процесс войдет в следующее состояние. Тогда процесс оставляет сигнал на своем месте в очереди и переходит к обработке следующего сигнала. Это действие называется сохранением сигнала — *save*;
3) нет никакого указания на то, как должен реагировать процесс на сигнал. В этом случае сигнал удаляется из очереди и процесс переходит к обработке следующего сигнала.

Описание каждого перехода процесса начинается с ключевого слова *state* и завершается одним из следующих ключевых слов: *join*, *stop* либо *endstate* (если последнее опущено, то словом *state*).

При переходе процесс может выполнить любой оператор языка Паскаль, а также такие действия, как принятие решения (*decision*); запрос на создание другого процесса; экспортно-импортную операцию; безусловный переход к другой последовательности действий; установку и сброс таймера.

Последовательность действий, выполняемых процессом во время перехода, может разветвляться. Для этого осуществляется проверка истинности некоторого выражения в конструкции *decision*, которая состоит из вопроса и не менее двух ответов. После каждого ответа указывается последовательность действий, которую должен выполнить процесс в данном случае. Если после нескольких ответов должна быть выполнена одна и та же последовательность действий, то эти ответы объединяют вместе. Также возможен только один ответ *else*, охватывающий все значения выражения (стоящего в вопросе), не учтенные во всех остальных ответах.

Порождение одного процесса другим осуществляется с помощью опе-

рации запроса на создание другого процесса. Процесс может порождать другие процессы только в своем блоке. Иначе процесс должен передать соответствующий сигнал процессу-производителю в другом блоке.

Один экземпляр процесса может передавать различным экземплярам других процессов значение некоторой переменной с помощью экспортно-импортной операции. Для этого при описании переменной он должен указать, что ее значение в дальнейшем будет экспортироваться. Если другой процесс хочет получить экспортированное значение переменной, то он должен в своей декларативной части указать, что намеревается импортировать это значение.

Безусловный переход к другой последовательности действий осуществляется конструкцией *join*, аналогичной оператору *go to* в языке Паскаль. С ее помощью указывается, что следующим должна выполняться последовательность действий этого же самого процесса, перед которым стоит указанная в конструкции *join* метка.

Назначение языка SDL — описание систем реального времени. Комплекс, реализующий систему, должен обладать средствами управления временем, например часами, которые показывают абсолютное время в согласованных единицах времени. В языке предусмотрена стандартная функция *now*, значением которой является значение текущего момента абсолютного времени, и имеется возможность описания таймеров и установки в них любого времени, указанного в принятых единицах. Реализуется эта возможность посредством оператора *set*. Когда абсолютное время в системе станет равным установленному в таймере, в порт процесса будет установлен сигнал от таймера, имя которого совпадает с именем самого таймера. Чтобы процесс мог воспринять сигнал от таймера, этот сигнал должен быть указан в теле процесса в качестве входного. С момента установки таймера и до момента восприятия сигнала от таймера он считается активным. Перевод таймера в неактивное состояние ("сброс таймера") осуществляется оператором *reset*.

Процессы могут содержать описания и вызовы процедур. Один процесс может вызвать процедуру, описанную в другом процессе, но в этом же блоке. При описании процедур явно указывается посредством ключевых слов *in* и *in/out* способ передачи параметров. После слова *in* указываются параметры, которые будут переданы процедуре по значению, после слов *in/out* — по ссылке. Рекурсивные процедуре в языке SDL не допускаются. Процедуры также не могут содержать экспортируемых и обозреваемых переменных. Одну и ту же процедуру могут одновре-

менно вызывать несколько процессов. Вызов процедуры выполняется следующим образом. Создается экземпляр-копия вызванной процедуры. В нем каждому формальному параметру, передающемуся по значению, присваивается значение соответствующего фактического параметра. Формальные параметры, передающиеся по ссылке, заменяются на соответствующие фактические параметры. После выполнения процедуры созданный экземпляр прекращает свое существование, результат его работы передается вызывающему процессу в качестве значений, присвоенных фактическим переменным, переданным по ссылке.

Язык SDL позволяет использовать макросредства. Механизм макросредств представлен конструкциями определения и вызова макрокоманды. Определение макрокоманды состоит из заголовка и тела макрокоманды. Заголовок имеет имя и список формальных параметров, которые при подстановке будут заменены текстом действительных параметров. Тело макрокоманды задается последовательностью предложений языка SDL. Макрокоманда — эта сокращенное обозначение отрезка текста, который не может функционировать самостоятельно, а должен быть вставлен в тело вызывающего процесса. При вызове создается копия тела макрокоманды, в которой все формальные параметры заменяются на соответствующие лексические единицы, указанные в вызове макрокоманды. Из описания системы удаляется вызов макрокоманды, на его место вставляется указанная копия тела макрокоманды. Определения макрокоманд помещаются в декларативной части описания системы, блока или процесса. Определение макрокоманды не может быть вложенным в определение другой макрокоманды.

Ограничения на SDL. Для того чтобы модели были максимально простыми и удобными для анализа и верификации, было решено на этом этапе отказаться от динамических конструкций. Наши интересы связаны в большей степени с моделированием временного поведения систем. Поэтому трансляция будет осуществляться только для тех систем, в которых не происходит динамического создания и уничтожения экземпляров процессов.

В настоящий момент разработана процедура трансляции только для операторов отправления сигналов вида

$$Output \ < \text{имя сигнала} >< \text{параметры} > \ to \ < \text{личный идентификатор} >,$$

и

$$Output \ < \text{имя сигнала} >< \text{параметры} > .$$

Первая конструкция означает, что сигнал, возможно с параметрами,

посылается процессу с номером, указанным после служебного слова *to*. Вторая — всем процессам, связанным с процессом-отправителем маршрутами, которые могут передавать этот сигнал. Таким образом, в этой версии не использована конструкция передачи вида

$$Output \ < \text{имя сигнала} >< \text{параметры} > \ via \ < \text{имя маршрута} > ...$$
$$< \text{имя маршрута} >,$$

которая передает сигнал всем экземплярам тех процессов, с которыми процесс-отправитель связан поименованными в конструкции (после служебного слова *via*) маршрутами.

2. РАСКРАШЕННЫЕ СЕТИ ПЕТРИ, ОБОГАЩЕННЫЕ ВРЕМЕННЫМ МЕХАНИЗМОМ

2.1. Раскрашенные сети Петри

Неиерархические раскрашенные сети Петри [10] являются расширением базовой модели сетей Петри. Раскрашенная сеть Петри (РСП) состоит из трех частей: структуры сети, деклараций и пометки сети.

Структура сети представляет собой направленный двудольный граф с двумя типами вершин — *местами* и *переходами*, соединенными дугами таким образом, что каждая дуга соединяет вершины различных типов. Места имеют *разметку*. Ненулевая разметка места определяется помещением в место одной или нескольких *фишек*. Места и находящиеся в них фишки представляют состояние системы, моделируемой сетью Петри, тогда как переходы — изменение состояния системы. Места, из которых в переход ведут дуги, называются *входными* местами для данного перехода. Места, в которые ведут дуги от данного перехода, называются его *выходными* местами.

Декларации состоят из описания множеств цветов и объявления переменных, каждая из которых принимает значения из некоторого множества цветов. Декларации также могут содержать определение операций и функций. Кроме того, определение множества цветов зачастую неявно вводит набор операций и функций, которые могут быть применены к элементам этого множества.

Пометка сети приписывается месту, переходу либо дуге. В неиерархической сети каждое место имеет три разных типа пометок: имя места,

множество цветов и инициализирующее выражение. Имя не имеет формального значения и служит для идентификации. Множество цветов определяет тип фишек, которые могут находиться в месте, т. е. любая фишка, находящаяся в месте, должна иметь цвет, который является элементом данного множества цветов. Инициализирующее выражение определяет мультимножество над соответствующим множеством цветов, а также то, какие фишки находятся в месте в начальный момент. Фишки, находящиеся в месте в начальный момент, называются *начальной разметкой* данного места.

Переходы имеют два типа пометок: имена и спусковые функции. Спусковая функция перехода является логическим выражением, которое должно быть выполнено до того, как переход сможет сработать.

Дуги имеют один тип пометок: выражения. Выражения на дугах могут содержать переменные, константы, функции и операции, определенные в декларациях. Когда все переменные, входящие в выражение на дуге, связаны, т. е. получили значения из соответствующих множеств цветов, выражение должно определять цвет (или мультимножество цветов) из множества цветов, приписанного месту, с которым связана дуга. Все вхождения одной и той же переменной должны замещаться одним и тем же цветом.

Пример раскрашенной сети взят из работы [6] и приведен на рис. 1. Данная сеть состоит из перехода *trans* и четырех мест: *State*, *counter*, i и j. Все места за исключением i являются как входными, так и выходными местами перехода *trans*. Место i является только входным местом. Декларации сети ограничены пунктирной линией и содержат определения трех множеств цветов и декларации переменных, входящих в выражения на дугах. Рядом с каждым местом приведены имя места, ассоциированное с ним множество цветов и начальная разметка места. Над переходом указаны его имя и спусковая функция.

Переход является активным компонентом раскрашенных сетей. Переходы срабатывают, изменяя при этом разметку своих входных и выходных мест. Чтобы переход мог сработать, все переменные, входящие в спусковую функцию перехода, и выражения на связанных с ним дугах должны получить значения. Выбор для каждой переменной конкретного цвета, при котором спусковая функция перехода имеет истинное значение, называется *связыванием*.

При выбранном связывании выражения на входных дугах перехода определяют, сколько и каких фишек должно содержаться во входных

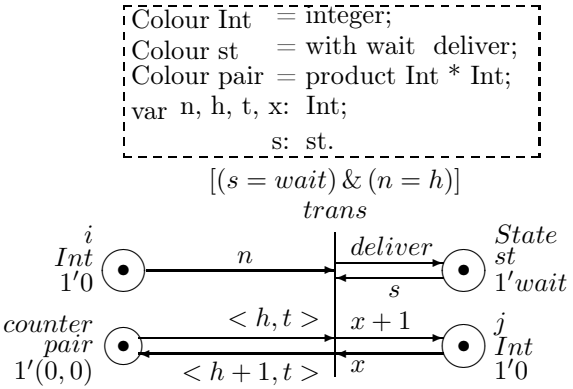

Рис. 1. Пример раскрашенной сети

местах перехода, чтобы переход мог сработать при выбранных значениях переменных. Выражения на выходных дугах определяют, сколько и каких фишек будет помещено в выходные места перехода, когда он сработает. Если входные места перехода содержат необходимый набор фишек, переход возможен при выбранном связывании. Возможный переход может сработать. Срабатывание перехода изымает фишки из входных его мест и добавляет в выходные места. Количество и цвет изымаемых/добавляемых фишек определяются выражениями на соответствующих дугах.

В сети на рис. 1 переход *trans* может сработать при начальной разметке. Все входные места перехода содержат по фишке. Значения $s = wait$, $n = 0$, $h = 0$, $t = 0$, $x = 0$ определяют связывание. После срабатывания перехода *trans* место i станет пустым, место *State* будет содержать фишку со значением *deliver*, место *counter* — фишку со значением $\langle 1, 0 \rangle$, а место j — фишку со значением 1.

2.2. Расширение раскрашенных сетей временным механизмом

Для моделирования оператора *set* необходимо явным образом ввести время в модель сетей Петри. Известен ряд временных расширений ординарных и раскрашенных сетей Петри. В работе [11] временной механизм для раскрашенных сетей Петри реализован с помощью введения

глобальных часов и временных штампов, которые несут фишки. Временной штамп фишки устанавливается создающим ее переходом и определяет время, когда фишка станет доступной для переходов. Для наших целей удобнее и проще определить собственное временное расширение раскрашенных сетей, соединив модели раскрашенных и временных сетей (далее ВСП), предложенных Мерлином [9], точно так же, как это было сделано в работе [6].

Во ВСП временной механизм связан с переходами: каждому переходу сопоставлена пара неотрицательных чисел d_{min} и d_{max}. Переход, каждое из входных мест которого содержит хотя бы одну фишку, называется *возможным*. В отличие от ординарных сетей Петри не всякий возможный переход имеет право сработать.

Если τ — момент времени, в который переход стал возможен, то его срабатывание произойдет в некоторый момент времени из интервала $[\tau + d_{min},\ \tau + d_{max}]$, если условие его возможности не будет нарушено до наступления времени $\tau + d_{max}$ в результате срабатывания другого перехода. Если произошло нарушение условия возможности перехода, то переход реинициируется, и время задержки его будет заново отсчитываться, начиная с момента восстановления условия его возможности. Возможный переход, имеющий право сработать, т. е. переход, который ожидает не менее d_{min} единиц времени с момента, когда он стал возможен, называется *реализуемым*, а интервал $[d_{min},\ d_{max}]$ — *интервалом срабатывания* перехода.

Если переход не сработал в течение интервала $[\tau,\ \tau + d_{max})$, то срабатывание форсируется и произойдет в момент времени $\tau + d_{max}$. Срабатывание перехода происходит мгновенно.

Таким образом, значение d_{min} определяет минимальное время, в течение которого переход должен оставаться возможным, прежде чем он сможет сработать, а значение d_{max} определяет максимальное время, в течение которого переход может оставаться возможным и не сработать.

Обычно рассматриваются ВСП, в которых не допускается кратной возможности переходов, т. е. в любой момент времени ни один переход в сети не имеет такого числа фишек в своих входных местах, которого было бы достаточно для двух или более одновременных срабатываний. Такие сети носят название *T-безопасных* ВСП. Из алгоритма, описанного в следующем пункте, видно, что в результате трансляции SDL-спецификаций получаются сети, переходы которых не имеют возможности кратного срабатывания.

Соединение раскрашенных сетей с описанным временным механизмом происходит естественным образом. У переходов сети появляется новый тип пометки — интервал срабатывания. Те же правила, что и раньше, определяют, может ли переход сработать: если во входных местах перехода содержится набор фишек, необходимый для некоторого связывания, то переход может сработать. Само срабатывание происходит с задержкой, значение которой выбирается из интервала срабатывания.

2.3. Раскрашенные сети с приоритетами

Для моделирования конструкции непрерывного сигнала, в которой используются приоритеты, естественно использовать известную модель приоритетных сетей Петри [3] точно так же, как это было сделано в работе [6]. Сети с приоритетами — ординарные сети, в которых каждому переходу сопоставлен элемент некоторого множества PR, называемого *множеством приоритетов*. Его элементы частично упорядочены некоторым отношением $\leq$ (меньше или равно). Правило срабатывания модифицируется следующим образом: возможный переход может сработать, если его приоритет не меньше приоритета любого другого возможного перехода. Отсутствие приоритета соответствует наименьшему приоритету.

Дополнение раскрашенных сетей приоритетами происходит по той же схеме, что и дополнение временными характеристиками. У переходов раскрашенной сети появляется новый тип пометки — приоритет. Возможность срабатывания перехода определяется так же, как в раскрашенных сетях. Из нескольких возможных переходов срабатывает любой, приоритет которого не меньше, чем у остальных возможных переходов.

3. ОСНОВНЫЕ ПРИНЦИПЫ ТРАНСЛЯЦИИ ВЕРХНЕГО УРОВНЯ

Изложим основные принципы трансляции SDL-системы в РСП. Сеть, моделирующая систему, строится с помощью поэтапного уточнения. На первом этапе строится сеть, которая соответствует основной структуре системы и содержит по одному переходу для каждого блока. Второй этап — трансляция блока. На последующих этапах происходит трансляция процессов и их переходов. Трансляция наиболее важных конструк-

ций приводится с некоторым неформальным обоснованием.

Для представления предопределенных сортов в SDL декларации сети изначально содержат соответствующие множества цветов:

$Colour\ Int = integer$;

$Colour\ Bool = boolean$;

$Colour\ Char = char$.

Корректность такого представления обеспечена тем, что, по определению раскрашенных сетей, множества цветов эквивалентны типам в языках программирования.

Считаем также, что в каждой сети определено множество цветов, состоящее из одного элемента, который не несет информации:

$$Colour\ E = with\ e.$$

Фишка со значением e называется *ординарной*, или *бесцветной*. Бесцветные фишки используются в сетях, моделирующих SDL-системы в служебных целях.

3.1. Моделирование FIFO-очереди

Каждый канал/маршрут имеет ассоциированную с ним FIFO-очередь, в которой сохраняются сигналы, полученные блоком/процессом через данный канал/маршрут. При трансляции SDL-системы очереди, ассоциированные с каналами/маршрутами, естественно представлять местами, а сигналы, сохраняемые в этих очередях, — фишками. Кроме того, каждая такая фишка должна содержать информацию о том, какой процесс ее отправил и какому процессу она предназначается.

Однако в раскрашенных сетях все фишки, находящиеся в некотором месте, равноправны и извлекаются из места в произвольном порядке. Поэтому для моделирования FIFO-очереди мы используем нумерацию фишек в соответствующем месте (рис. 2) точно так же, как это было сделано в работе [5].

Первое поле любой фишки в месте *Queue* содержит номер, который определяет ее положение в очереди, следующие поля — собственно элемент очереди, значение предпоследнего поля фишки соответствует личному идентификатору процесса-получателя, а последнего — процесса-отправителя. Если сигнал предназначается всем процессам, которые принимают данный сигнал, значение предпоследнего поля равно E.

При этом с местом *Queue*, представляющим очередь, связывается

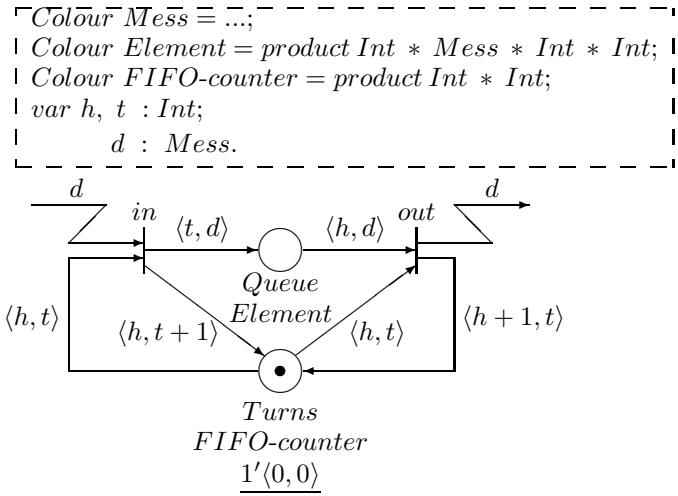

Рис. 2. Моделирование FIFO-очереди

дополнительное место $Turns$, которое служит для моделирования дисциплины FIFO-очереди.

Фишки в месте $Turns$ могут принимать значения из множества цветов $FIFO\text{-}counter$, который определяется следующим образом:

$$Colour\ FIFO\text{-}counter = product\ Int\ *\ Int.$$

При начальной разметке место $Turns$ содержит фишку $\langle 0, 0\rangle$. Значение первого поля фишки в месте $Turns$ означает номер фишки, которая находится в месте $Queue$ и соответствует сообщению в голове очереди. Значение второго поля означает номер, с которым в место $Queue$ будет помещена новая фишка, соответствующая сообщению, которое помещается в хвост очереди.

Место $Turns$ — входное и выходное место для любого перехода сети, который изымает из места-очереди или помещает в него фишку. При срабатывании перехода in, который помещает фишку в место-очередь, из места $Turns$ изымается фишка со значением $\langle h, t\rangle$ и добавляется фишка $\langle h, t + 1\rangle$. В место $Queue$ помещается новая фишка $\langle t, d\rangle$. Из всех фишек, которые содержатся в месте $Queue$, новая фишка имеет максимальный номер t. Значение второго поля фишки в месте $Turns$ увеличилось на единицу и представляет номер, с которым в место $Queue$ будет помещена следующая фишка.

При срабатывании перехода *out* в месте $Turns$ фишка $\langle h, t\rangle$ заменяется фишкой $\langle h + 1, t\rangle$, а из места $Queue$ изымается фишка $\langle h, d\rangle$. Заметим, что фишка, которая изымается из места $Queue$, однозначно определяется значением первого поля фишки в месте $Turns$. Увеличение значения h на единицу определяет фишку, которая будет изъята из места $Queue$ при следующем срабатывании перехода *out*.

3.2. Генерация сети верхнего уровня

На данном этапе порождается сеть, отражающая общую структуру SDL-системы. При порождении сети используются описания всех сигналов и списков сигналов, видимые всем блокам, описания всех каналов, соединяющих между собой блоки и окружающую среду и блоки, а также часть информации из описания процессов. Каждому блоку в сети ставится в соответствие один переход. Количество блоков в системе определяет, сколько переходов будет содержать строящаяся сеть.

Каналы, входящие в блок и выходящие из него, представляются местами, которые связаны дугами с переходом, соответствующим данному блоку. Отображение каналов происходит по следующим правилам:

- канал, через который блок может только получать сигналы (входящий канал) представляется одним местом — входным для соответствующего перехода;
- канал, через который блок может только отправлять сигналы (выходящий канал) представляется одним местом — выходным для соответствующего перехода;
- если через канал происходит двунаправленный обмен сигналами, он представляется в сети двумя местами, одно из которых входное, а второе — выходное для соответствующего перехода.

Описания всех сигналов, которые могут передаваться по каналам в системе, переходят в декларации сети. При этом создаются множества цветов, которые будут использоваться при моделировании взаимодействий между блоками.

Поскольку переходы сети, полученной на первом этапе, заменяются в процессе трансляции подсетями, моделирующими блоки, спусковые функции для них на первом этапе не определяются.

В SDL существует способ передачи между процессами значения некоторой переменной посредством экспортно-импортной операции. Такая операция может производиться между экземплярами процессов, не принадлежащими одному блоку. Процесс, импортирующий значение пере-

менной, получает то значение, которое эта переменная имела тогда, когда процесс-экспортер совершил в последний раз экспортную операцию. На этом этапе каждой экспортируемой переменной ставится в соответствие одно служебное дополнительное место, которое будет входным и выходным для переходов, соответствующих блокам, которые содержат процессы "экспортер" и "импортер". В п. 5.2 будет описано, как это служебное место используется. Инициализирующее выражение для мест, представляющих экспортируемые переменные, определяется в декларативной части процесса.

3.3. Пример

Процесс построения сети проиллюстрируем на модифицированном примере системы S, представленной в [2]. В системе описаны три канала $c1, c2, c3$ и два блока $B1$ и $B2$. По каналу $c1$, соединяющему окружение с блоком $B1$, от блока могут передаваться сигналы $S1, S2$ к окружению, по каналу $c3$, соединяющему блок $B2$ с окружением, блоку может быть передан сигнал $s3(Integer)$, который имеет параметр целого сорта. Между блоками $B1$ и $B2$ имеется двунаправленный канал $c2$. По нему от блока $B1$ к блоку $B2$ может передаваться сигнал $s4$, а в обратном направлении — сигналы $s5, s6$. Блоки $B1$ и $B2$ в системе не описаны. Пример линейного описания системы, совпадающей с системой, изображенной на рис. 3, приведен ниже.

```
system S;
  signal s1,s2,s3(Integer),s4,s5,s6;
  channel C1 from B1 to env with s1,s2;
    endcahnnel C1;
  channel C2 from B1 to B2 with s4;
             from B2 to B1 with s5,s6;
    endcahnnel C2;
  channel C3 from env to B2 with s3;
    endcahnnel C3;
  block B1 referenced;
  block B2 referenced;
endsystem S;
```

Для системы S декларации сети дополняются следующими множествами цветов:

$Colour\ Sig1 = with\ s1 | s2;$

$Colour\ Sig = with\ s3;$

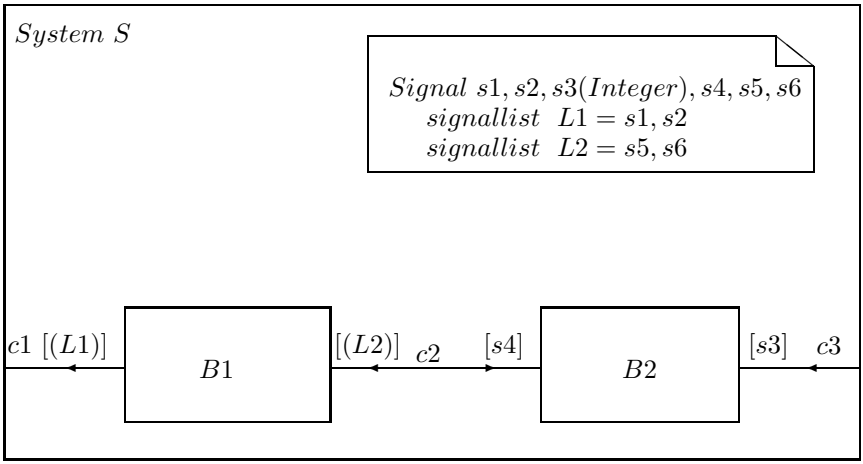

Рис. 3. Описание системы S

$Colour\ Sig3 = product\ Sig\ *\ Int$;
$Colour\ Sig21 = with\ s4$;
$Colour\ Sig22 = with\ s5|\ s6$;
$Colour\ C\_1 = product\ Int\ *\ Sig1\ *\ Int\ *\ Int$;
$Colour\ C\_21 = product\ Int\ *\ Sig21\ *\ Int\ *\ Int$;
$Colour\ C\_22 = product\ Int\ *\ Sig22\ *\ Int\ *\ Int$;
$Colour\ C\_3 = product\ Int\ *\ Sig3\ *\ Int\ *\ Int$.

Первое поле любой фишки, принимающей значение из множества цветов C_1, C_21, C_22, C_3,, полученного при отображении описаний сигналов, содержит натуральное число. Это поле используется в сети при моделировании очереди сообщений, полученных некоторым процессом в дальнейшем построении сети. Последние два поля — личные идентификаторы принимающего и отправляющего процессов. Изначально все места, порожденные по описаниям каналов, имеют нулевую разметку.

В процессе дальнейшего построения сети декларации дополняются переменными, которые будут входить в выражения на дугах сети и спусковые функции переходов, и, возможно, новыми множествами цветов, если блоки и процессы содержат собственные определения сортов, сигналов и списков сигналов.

Сеть для системы S показана на рис. 4 (чтобы не загромождать рисунок, декларации сети опущены). Для большей наглядности в качестве имени перехода используется имя соответствующего блока.

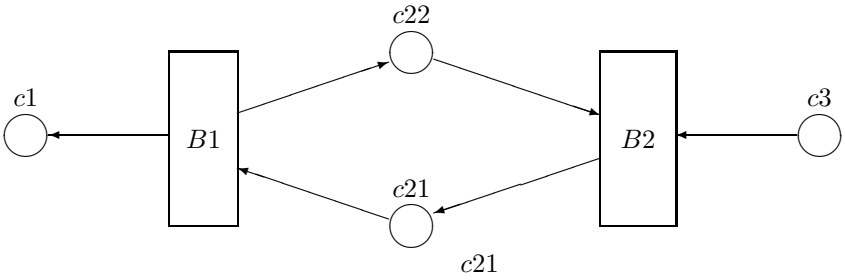

Рис. 4. Сеть для системы S

Таким образом, после завершения первого шага трансляции имеем сеть, которая состоит из переходов (представляющихся "черными ящиками"), соответствующих блокам в системе, и мест, полученных при трансляции каналов и экспортируемых переменных.

4. ТРАНСЛЯЦИЯ БЛОКА

Описание системы — это последовательность диаграмм, где каждая следующая диаграмма осуществляет дальнейшую детализацию системы. Предусмотрены диаграммы подструктуры блоков и каналов. Диаграмма подструктуры блока используется в тех случаях, когда рассматриваемый блок представляет сложный объект и состоит из подблоков и внутренних каналов. Каналы внутри блока назовем “новыми” каналами. Каналы внутри системы, которые соединяют блоки между собой и с окружением, назовем “старыми”. В результате разбиения блоков возникает структура, совершенно аналогичная структуре системы, в которой рамка блока играет роль рамки системы.

Если описание какого-либо блока дано за пределами описания системы (указана ссылка на удаленном описание), то дальнейшего уточнения сети для этого блока не происходит. В результирующей сети данному блоку будет соответствовать один переход, а именно тот, который порожден на предыдущем этапе трансляции.

4.1. Трансляция блока, состоящего из подблоков

Описание разбиения блока на подблоки должно содержать описание новых блоков (или ссылки на их удаленное описание), описание "новых" каналов, соединения "новых" каналов со "старыми" и, возможно, описание новых сигналов, новых списков сигналов, новых типов данных и новых макрокоманд.

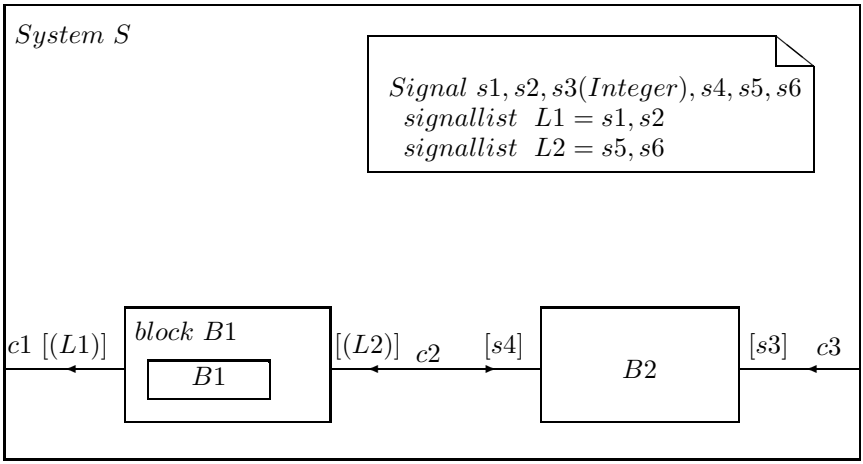

Рис. 5. Уточнение описания системы S

Трансляция блока происходит таким же образом, что и всей системы в целом. На предыдущем этапе каждому блоку поставлен в соответствие один переход. На данном этапе эти переходы заменяются подсетями, которые соответствуют разбиению блока. Каждому новому блоку в подсети соответствует один переход, каждому "новому" каналу одно или два места, в зависимости от того, одно- или двунаправленный канал. Фишки в полученных местах могут принимать значения из множества цветов, которое определяется сигналами, передаваемыми по соответствующим "новым" каналам. Изначально все места, порожденные по описаниям каналов, имеют нулевую разметку. Множество цветов в месте, соответствующем "новому" каналу, присоединенному к "старому", является подмножеством цветов места, соответствующего этому "старому" каналу. Соединение переходов и мест осуществляется дугами, направление которых совпадает с направлением передачи сообщений.

Рассмотрим систему S, представленную на рис. 5, в которой блок

$B1$ имеет подструктуру $B1$. Описание подструктуры $B1$ дано на рис. 6. Каждому подблоку подструктуры $B1$ в сети соответствует переход, имя перехода совпадает с именем подблока.

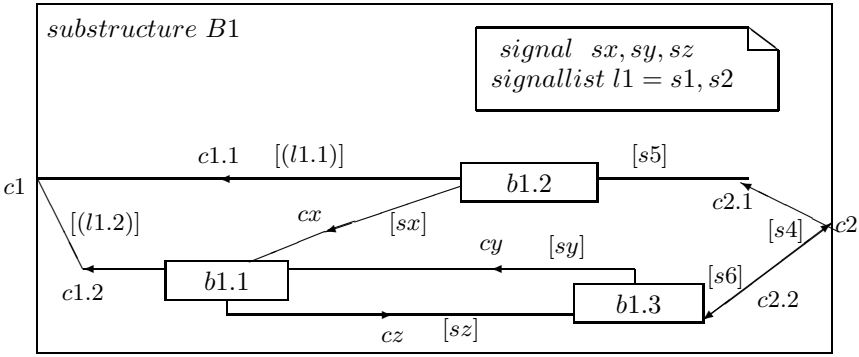

Рис. 6. Описание подструктуры $B1$

На рис. 7 показано, как отображаются в сети "новые" каналы подструктуры $B1$. Место $c1.1$ соответствует "новому" каналу $c1.1$ и является выходным для перехода $B1.2$, место $c1.2$ соответствует "новому" каналу $c1.2$ и является выходным для перехода $B1.1$, место $c2.1$ соответствует "новому" каналу $c2.1$ и является входным для перехода $B1.2$.

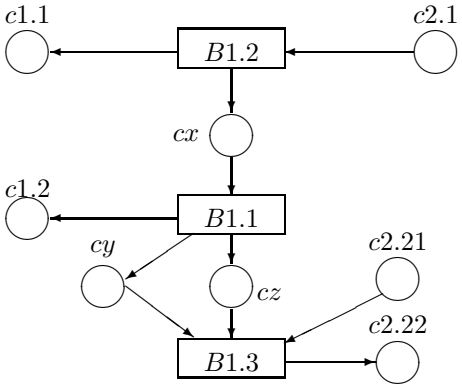

Рис. 7. Сеть для подструктуры $B1$

Каналу $c2.2$ в сети соответствует два места: $c2.21$ — входное и $c2.22$ — выходное для перехода $B1.3$, моделирующего подблок $B1.3$. Новые однонаправленные каналы cx, cy и cz в сети соответственно моделируются

местами с такими же именами.

Каждый сигнал, поступивший по “старому” каналу, может передаваться только по одному “новому” каналу. В строящейся сети места, созданные на предыдущем этапе и соответствующие “старому” каналу, сливаются с местами, соответствующими “новым” каналам, присоединенным к “старому”, следующим образом.

Назовем места, созданные на предыдущем этапе и соответствующие “старому” каналу, “старыми местами”, а места, созданные на этапе трансляции подструктуры и соответствующие “новым” каналам, присоединенным к “старому”, — “новыми местами”. Таким образом, каждому “старому месту” соответствует свой набор “новых мест”. В нашем примере место $c1$ — “старое место”, а $c1.1$ и $c1.2$ являются “новыми местами”.

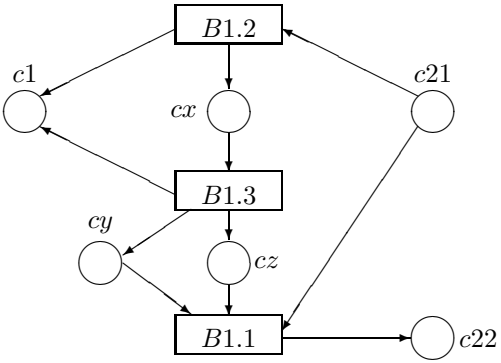

Рис. 8. Результирующая сеть для подструктуры $B1$

Каждое входное “старое место” для перехода, моделирующего блок, сливается с соответствующими “новыми местами”, входными для переходов, моделирующих подблоки. Аналогично каждое выходное “старое место” для перехода, моделирующего блок, сливается с соответствующими “новыми местами”, выходными для переходов, моделирующих в подструктуре подблоки. При этом несколько мест, соответствующих “новым” каналам, могут быть слиты с одним и тем же местом, соответствующим “старому” каналу блока.

На рис. 8 представлена сеть для подструктуры $B1$, в которой места, соответствующие “новым” каналам, уже слиты с местами, соответствующими “старым” каналам. Выходное для перехода $B1$ место $c1$, построенное на предыдущем этапе, слито с местами $c1.1$ (выходным для $B1.2$) и $c1.2$ (выходным для $B1.1$), входное для этого же перехода место $c21$

— с местами $c2.1$ (входным для $B1.2$) и $c2.21$ (входным для $B1.3$), а выходное место $c22$ — с местом $c2.22$ (выходным для $B1.3$).

4.2. Трансляция блока, состоящего из процессов

Каждый блок, не расчлененный на подблоки, должен содержать хотя бы один процесс. Процессы между собой и с рамкой блока соединяются маршрутами. Как было сказано ранее, заголовок процесса содержит имя, число экземпляров и список формальных параметров с указанием их сортов.

Трансляция блока, состоящего из процессов, аналогична трансляции блока любого уровня иерархии. На этом этапе каждому экземпляру процесса соответствует один переход, каждому маршруту одно или два места, в зависимости от того, каков маршрут — одно- или двунаправленный. Изначально все места, порожденные по описаниям маршрутов, имеют нулевую разметку. Соединение переходов и мест осуществляется дугами, направление которых совпадает с направлением передачи сигналов.

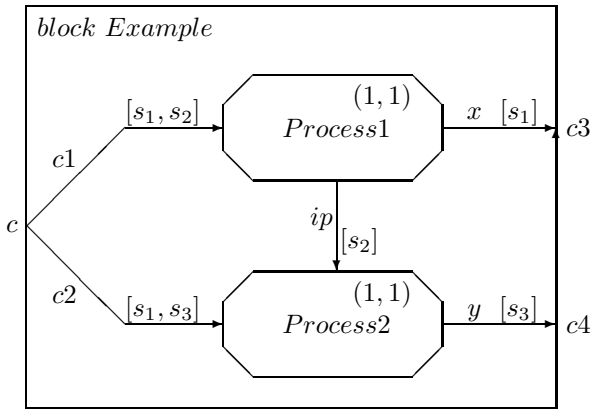

Рис. 9. Описание блока *Example*

К одному входному каналу могут быть подсоединены несколько маршрутов (см. рис. 9). Сигнал, поступающий по такому каналу, может передаваться по нескольким подсоединенным маршрутам. Таким образом, в точке присоединения маршрутов с каналом сигнал может копироваться. В строящейся сети точке присоединения будет соответствовать один

переход, назовем его *copy*. Если канал является односторонним, то место, созданное на предыдущем этапе и моделирующее этот канал, будет входным местом для перехода *copy*, а места, моделирующие подсоединенные маршруты, — выходными. Если канал двусторонний, то это значит, что на предыдущем этапе для него было создано два места, одно входное и одно выходное для перехода, моделирующего блок. Это входное место будет входным местом для построенного перехода *copy*. Спусковая функция и выражения на входных дугах данного перехода определяются сигналами, передаваемыми по маршрутам, которые моделируются соответствующими выходными для перехода *copy* местами.

Каждому формальному параметру ставится в соответствие одно место. Места-формальные параметры являются входными и выходными для соответствующего перехода-процесса. В качестве имени места используется идентификатор соответствующего параметра. Множеством возможных цветов для фишек в месте-формальном параметре становится множество цветов, соответствующее сорту данного параметра.

Один экземпляр процесса может сделать значение некоторой своей переменной доступной другим процессам, указав при описании, что раскрывает (*viewed*) значения этой переменной. Другой процесс, находящийся в этом же блоке, может использовать значение этой переменной, описав ее как обозреваемую (*revealed*). Обозревание совершается мгновенно в том смысле, что процесс, использующий значение этой переменной, получает текущее значение обозреваемой переменной. Каждой обозреваемой переменной соответствует одно место, которое будет входным и выходным местом как для перехода, представляющего процесс, который раскрывает переменную, так и для переходов, представляющих процессы, которые обозревают эту переменную.

Блок может содержать процессы, которые экспортируют/импортируют значения некоторой переменной. На этом этапе каждой экспортируемой переменной ставится в соответствие одно служебное дополнительное место, которое будет входным и выходным для переходов, соответствующих процессам “экспортер” и “импортер”. Использование этого служебного места описывается в п. 5.2.

Проиллюстрируем трансляцию блока *Example*, состоящего из двух процессов *Process*1 и *Process*2. Описание этого блока представлено на рис. 9, а сеть, моделирующая этот блок, приведена на рис. 10. Места $c1$ и $c2$, соответствующие маршрутам в блоке *Station*, являются выходными для перехода *copy*, а место c, моделирующее входной канал для

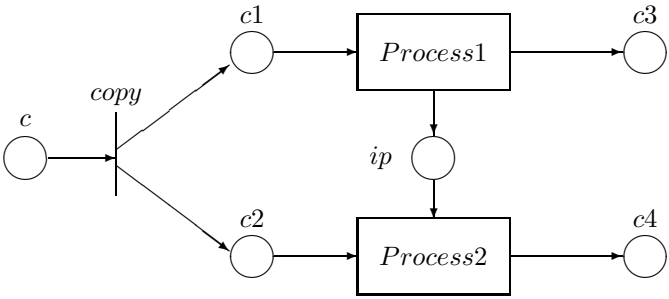

Рис. 10. Описание блока *Station*

этого блока, является входным. Места, представляющие маршруты x и y, слиты соответственно с местами, моделирующими выходные каналы $c3$ и $c4$.

По-прежнему на этом этапе для переходов, соответствующих процессам, не создаются спусковые функции, поскольку эти переходы будут заменены подсетями в процессе дальнейшей трансляции.

5. ТРАНСЛЯЦИЯ ПРОЦЕССА

Как уже было сказано, процесс является основным понятием языка SDL. Более формально можно считать, что процесс из внешней среды воспринимает входные сигналы, обрабатывает их и передает внешней среде выходные сигналы.

На предыдущем этапе каждому экземпляру процесса поставлен в соответствие один переход. При трансляции процесса каждый переход, соответствующий экземпляру процесса, заменяется подсетью, которая будет представлять внутреннюю структуру данного процесса. На этом этапе каждому SDL-переходу в сети будет соответствовать один переход. Для построения всей сети на этом уровне используется часть информации из описания SDL-переходов, но сами переходы на данном этапе по-прежнему представляются “черными ящиками” подобно тому, как это происходило с процессами на предыдущем этапе трансляции.

5.1. Общие принципы трансляции процесса

Все сорта, определения сигналов и списков сигналов, описанные в декларативной части процесса, преобразуются в множества цветов по приведенной выше схеме и заносятся в декларации сети. Множество состояний процесса становится множеством цветов, которое также заносится в декларации сети.

В подсети, соответствующей процессу, для каждой переменной будет создано одно место, для каждого массива — также одно место. Сорт переменной определяет множество цветов соответствующего места. Множество цветов места, соответствующего переменной массива, представляется множеством пар, которые состоят из индекса элемента массива и значения этого элемента. Количество фишек в таком месте совпадает с размерностью массива. Пометка каждого места, соответствующего переменной, состоит из имени места, его цвета и начальной разметки, которая соответствует начальному значению переменной, если оно определено, или нулю, если не определено.

Подсеть, моделирующая процесс, имеет четыре служебных места. Первое место *queue* соответствует порту процесса и содержит очередь фишек, соответствующих сигналам, поступающим по всем маршрутам к данному процессу. В строящейся сети место *queue* сливается с местами, соответствующими входным маршрутам процесса, подобно тому, как объединялись места, соответствующие "старым" и "новым" каналам (см. п. 4.1). В сети также заводится дополнительное место $count\_q$ для моделирования дисциплины FIFO-очереди. Фишки в нем принимают значения из множества цветов $FIFO\text{-}counter$. В начальном состоянии это место содержит фишку $(0, 0)$.

Второе служебное место *State* содержит множество цветов, которое определяется состояниями процесса. Выполнение SDL-перехода — неделимое действие. Так как переход в SDL не всегда представляется одним переходом сети, необходимо гарантировать следующее: пока срабатывают переходы сети, моделирующие один SDL-переход, невозможны переходы сети, моделирующие другие SDL-переходы. Это происходит автоматически при наличии в сети места *State*. Начальная разметка этого места — одна бесцветная фишка.

Третье служебное место $self$ соответствует переменной $self$, начальная разметка — номер, который присваивается процессу после создания всей сети.

Четвертое служебное место *sender* соответствует переменной *sender*.

В момент, когда процесс-получатель воспринимает входной сигнал, его переменной *sender* присваивается личный идентификатор того экземпляра процесса, который послал этот сигнал. Фишка в этом служебном месте соответствует значению переменной *sender*. Начальная разметка этого места — одна фишка значения 0.

Если какой-либо переход процесса воспринимает сигнал, под влиянием которого процесс прекращает функционирование, то подсеть, соответствующая процессу, дополнительно будет иметь пятое служебное место *stop* и служебный переход *stp* для организации остановки процесса. Начальная разметка этого места — одна фишка значения 0.

Если процесс содержит конструкцию непрерывного сигнала, то подсеть имеет дополнительное служебное место, которое будет входным и выходным для всех переходов, моделирующих SDL-переходы в конструкции непрерывного сигнала.

В сети также заводятся дополнительные места для моделирования дисциплины FIFO-очередей — по одному на каждое место, полученное при отображении выходных маршрутов. Фишки в таких местах принимают значения из множества цветов $FIFO\text{-}counter$. В начальном состоянии каждое из этих мест содержит фишку $(0, 0)$.

Кроме того, в сети создается служебный переход *del*, для которого входным местом будет место *queue*. Места $count\_q$, $State$ и $self$ являются входными и выходными для этого перехода. Переход *del* создается по двум причинам. Во-первых, для моделирования потребления сигнала из порта процесса в том случае, если процесс, находясь в определенном состоянии, не воспринимает сигнал, который стоит первым в порту, а во-вторых, для моделирования потребления сигнала из порта процесса в том случае, если сигнал, который стоит первым в порту, предназначен для другого процесса. Спусковая функция перехода *del* создается исходя из состояний, которые имеет процесс, сигналов, которые воспринимаются процессом, и личного идентификатора рассматриваемого процесса.

Каждый процесс имеет одну стартовую вершину. Процесс начинает функционировать с выполнения перехода, следующего за стартовой вершиной процесса. Такой переход будем называть стартовым. Он совершается не под влиянием входного сигнала, а в результате возникновения процесса. В сети ему будет соответствовать переход *start*.

На данном этапе для каждого SDL-перехода процесса, кроме перехода, содержащего конструкцию *save*, в сети создается один переход.

5.2. Соединение мест и переходов

Соединение мест и переходов на этом этапе происходит следующим образом.

– Если некоторая переменная используется в переходе процесса, то соответствующее данной переменной место будет входным и выходным местом для перехода сети, моделирующего этот SDL-переход.

– Если некоторая переменная используется в стартовом переходе процесса, то соответствующее данной переменной место будет входным и выходным местом для перехода *start*.

– Место *queue* будет входным для всех переходов, моделирующих SDL-переходы транслируемого процесса, исключение составляют переходы, которые входят в конструкцию непрерывного сигнала и содержат разрешающее условие с приоритетом. Дополнительное место $count_q$ будет входным и выходным местом для всех переходов, для которых место *queue* будет входным.

– Если в переходе процесса используются операторы *set* или *reset*, то в сети место *queue* будет выходным местом для перехода, моделирующего этот SDL-переход.

– Место *sender* будет входным и выходным местом для каждого перехода, моделирующего SDL-переход процесса.

– Если в переходе процесса осуществляется посылка сигнала, то в сети место $self$ будет входным и выходным местом для перехода, который соответствует этому SDL-переходу.

– Если в переходе процесса осуществляется посылка сигнала, то проводится дуга от соответствующего сетевого перехода к месту, моделирующему выходной маршрут процесса. Дополнительное место для организации очереди будет входным и выходным местом для этого перехода.

– Если стартовый переход процесса осуществляет посылку сигнала, то место $self$ будет входным и выходным местом для перехода *start*.

– Если стартовый переход процесса осуществляет посылку сигнала, то проводится дуга от перехода *start* к месту, соответствующему выходному маршруту этого процесса. Сигналы определяют выражение на этой дуге. Дополнительное место для организации очереди — входное и выходное для этого перехода.

– Место *State* — входное и выходное место для каждого перехода в сети (в том числе для переходов *start* и *del*).

– Место *stop* (если оно есть в сети) — входное и выходное место для всех переходов, моделирующих SDL-переходы, в том числе для служеб-

ных переходов *del* и *stp*.

– При наличии в подсети служебного перехода *stp* место *queue* будет входным для него.

Спусковая функция перехода *stp* определяется значением фишки в месте *stop* (подробно смотри ниже в п. 7.5).

Спусковая функция перехода *start* определяется на основании того, что фишка в месте *State* бесцветна. Выражение на выходной дуге перехода *start*, соединяющей этот переход с местом *State*, определяется состоянием, следующим в процессе непосредственно за стартовым переходом.

Процесс, импортирующий значение некоторой переменной, получает то значение, которое эта переменная имела тогда, когда процесс-экспортер совершил в последний раз экспортную операцию. На этом этапе каждой экспортируемой переменной поставлено в соответствие одно место, которое будет входным и выходным для всех переходов, моделирующих SDL-переходы, использующие эту переменную. На предыдущих этапах каждой экспортируемой переменной поставлено в соответствие одно служебное место. Это место будет входным и выходным только для тех переходов, которые соответствуют экспортной и импортной операциям. В процессе функционирования построенной сети переход, моделирующий экспортную операцию, помещает в это дополнительное служебное место фишку, которая совпадает с фишкой в месте, созданном на данном этапе и соответствующим этой экспортируемой переменной.

Трансляция процедур и функций осуществляется на следующем этапе.

5.3. Пример

Проиллюстрируем трансляцию процесса на фрагменте процесса $Initiator$ из протокола $INRES$, описанного в работе [17]. Этот процесс связан двусторонним маршрутом *isap* с рамкой блока. По нему передаются сигналы $ICONreq$ и $IDATreq$ из окружения блока к процессу, а в обратном направлении сигналы $ICONconf$ и $IDISind$. Процесс $Initiator$ также связан с процессом $Coder\_Ini$ двусторонним маршрутом *ipdu*, по которому передаются сигналы CR и DT от процесса $Initiator$ к процессу $Coder\_Ini$, а сигналы CC, DR и AK в обратном направлении.

Совершив стартовый переход, процесс входит в состояние $dis\_connected$. Из этого состояния под воздействием входных сигналов DR или $ICONreq$ возможен один из двух переходов. Часть линейного описания

системы процесса *Initiator* приведена ниже.

```
process $Initiator$;
  DCL counter integer;
  Start;
  NextState dis_connected;
  State dis_connected;
    INPUT DR;     {Первый переход}
    OUTPUT IDISind;
  NextState dis_connected;
    INPUT ICONreq; {Второй переход}
    TASK counter:=1;
    OUTPUT CR;
    set(now+p);
  NextState wait;
  .............
endprocess $Initiator$;
```

Приведенное выше линейное описание эквивалентно графическому на рис. 11. Если первым в очереди находится сигнал $ICONreq$, то совершается первый переход процесса, который воспринимает этот сигнал, переменной *counter* присваивает единицу, посылает в выходной маршрут *idpu* сигнал CR, устанавливает в таймере T время $now + p$ и переходит в состояние *wait*.

Если первым в очереди находится сигнал DR, то совершается второй переход процесса, который воспринимает этот сигнал, посылает в выходной маршрут *isap* сигнал $IDISind$ и переходит в состояние *wait*.

На этом этапе трансляции первому SDL-переходу процесса *Initiator* в сети будет соответствовать переход $trans1$, второму — $trans2$. Декларации сети будут дополнены новым множеством цветов:

$Colour\ DR = with\ DR.$

$Colour\ CR = with\ CR.$

$Colour\ ICONreq = with\ ICONreq.$

$Colour\ IDISind = with\ IDISind.$

$Colour\ dis\_con = with\ dis\_connected.$

$Colour\ wait = with\ wait.$

$Colour\ States\_Ini = E \mid dis\_con \mid wait.$

Применение правил трансляции приводит к сети, изображенной на рис. 12. Чтобы не загромождать рисунок, воспользуемся двунаправленной стрелкой для обозначения того, что место является как входным,

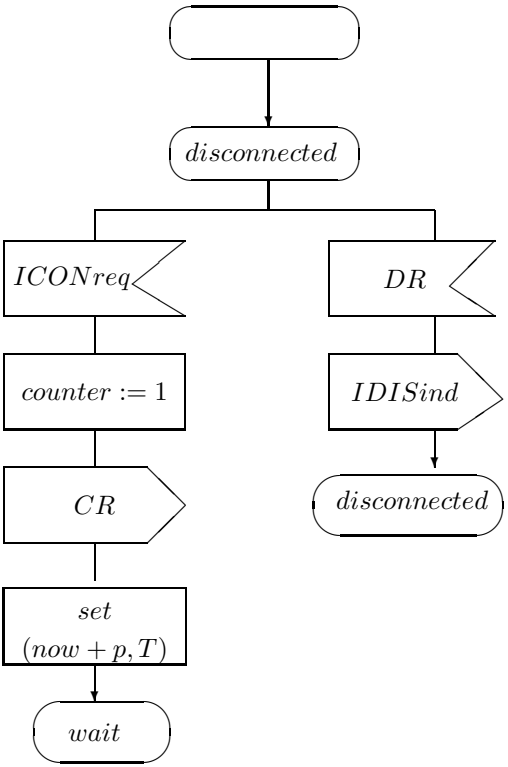

Рис. 11. Фрагмент процесса *Initiator* в *INRES*-протоколе

так и выходным для перехода. Место *counter* соответствует переменной *counter*. Оно является входным и выходным для перехода *trans*2. Сеть содержит четыре служебных места *queue*, *State*, *self*, *sender* и место *count_q* для организации очереди.

Фишка в месте *State* может иметь цвет из множества цветов *States_Ini*, начальная разметка этого места — одна бесцветная фишка. Начальная разметка места *sender* — фишка значения 0, места *self* — фишка целого значения, которое будет присвоено в момент инициации системы.

Каждому двустороннему маршруту на предыдущем шаге в сети поставлено в соответствие два места, одно входное, другое выходное для

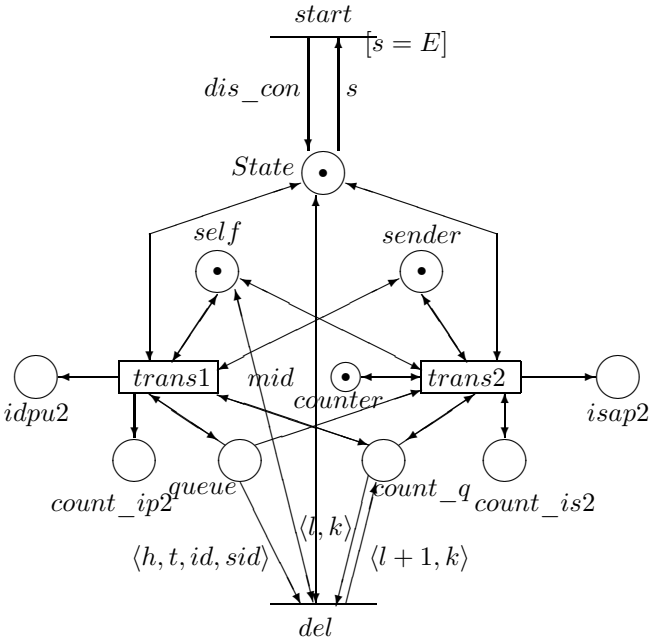

Рис. 12. Сеть для фрагмента процесса *Initiator*

перехода, моделирующего процесс. Эти входные места слиты с местом *queue* в сети на рис. 12. Также сеть содержит выходные места *isap*2, *ipdu*2 и соответственно места *count_is*2, *coun_ip*2 для организации очередей. При начальной разметке все места, соответствующие маршрутам, пусты.

Рассмотрим соединение мест в этой сети с переходом *trans*2. Места *State*, *self* и *sender* являются входными и выходными местами для него. Место *queue* является входным, а *count_q* — входным и выходным. Поскольку второй переход процесса посылает сигнал в канал $ISAP$, то соответствующее место *isap*2 является выходным местом для перехода *trans*2, а место для организации очереди *count_ip*2 — входным и выходным.

Выходная дуга перехода *start* имеет выражение *dis_con*, таким образом, после срабатывания этого перехода в месте *State* появится фиш-

ка значения dis_con.

Спусковая функция перехода del делает возможным его срабатывание, если

- место $State$ содержит фишку цвета dis_con, что соответствует состоянию процесса $dis\_connected$;
- второе поле фишки в месте $queue$, которая соответствует первому сигналу в очереди сигналов, полученных данным процессом, имеет значение, не совпадающее с цветами DR и $ICONreq$, либо если значение третьего поля (id) этой же фишки не равно значению фишки в месте $self$.

Таким образом, переход del моделирует изъятие из порта сигнала, стоящего в очереди первым, который либо не обрабатывается процессом, находящимся в состоянии $dis\_connected$, либо этот сигнал адресован другому процессу.

6. ТРАНСЛЯЦИЯ ПЕРЕХОДА ПРОЦЕССА

Выполнение некоторых SDL-переходов может моделироваться одним переходом сети, что происходит в том случае, если переход не содержит таких действий, как принятие решений ($decision$), переход на метку ($join$), установка (set) и сброс ($reset$) таймера, сохранение сигнала ($save$), разрешающее условие ($provided$) и вызовы процедур. Такой переход процесса представляет собой набор операторов присваивания и, возможно, операторов передачи сигналов. При этом ни один из этих операторов не использует переменной, измененной в этом переходе. Такие переходы будем называть *простыми*, и для них данный этап трансляции станет последним: будут определены спусковые функции и выражения на входных/выходных дугах.

6.1. Трансляция простого перехода

Спусковая функция перехода сети, моделирующего простой переход процесса, определяется состоянием, из которого возможен этот переход; входным сигналом, который указывается в операторе *input* (исключение составляют переходы с разрешающим условием в конструкции непрерывного сигнала); личным идентификатором процесса, который указывается в третьем поле фишки в месте *queue*. В спусковой функции проверяется, что личный идентификатор процесса, которому предназначается сигнал, совпадает с личным идентификатором рассматриваемого процесса. Выражения на дугах определяются состоянием, следующим после ключевого слова *nextstate*, и операторами, составляющими переход процесса.

Заметим, что при принятых ограничениях динамического поведения системы SDL состояние процесса характеризуется собственно его состоянием, содержимым очередей, ассоциированных с маршрутами, и значением всех переменных. При трансляции состояние процесса отображается разметкой моделирующей сети. Построение происходит таким образом, чтобы срабатывание моделирующего перехода было возможно при некоторой разметке тогда и только тогда, когда в соответствующем состоянии процесса может выполниться рассматриваемый SDL-переход. Определение РСП гарантирует, что выполнение простого перехода и срабатывание соответствующего ему перехода сети приводят к эквивалентным изменениям в состоянии процесса и разметке сети соответственно.

Рассмотрим на рис. 11 второй переход , которому на предыдущем этапе поставлен в соответствие переход *trans*2. Последний переход может быть представлен одним переходом сети. Подсеть, которая будет подставлена вместо перехода *trans*2, изображена на рис. 13.

Переход *trans*2 может сработать, если

- место *State* содержит фишку цвета *dis_con*, что соответствует состоянию процесса *dis_connected*;
- второе поле фишки в месте *queue*, которая соответствует первому сигналу в очереди сигналов, полученных процессом, имеет значение, соответствующее сигналу *DR*, а значение ее третьего поля равно значению фишки в месте $self$.

Срабатывание перехода *trans*2 заключается в следующем:

- из места *State* забирается фишка со значением *dis_con* и возвращается в него со значением *wait*;

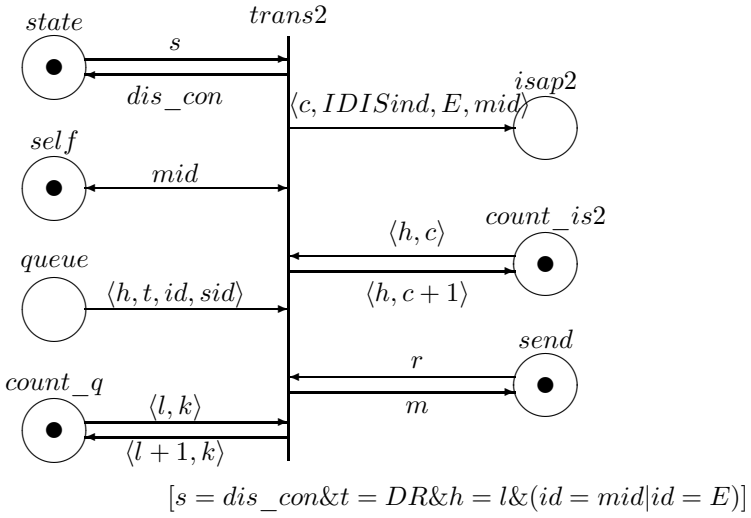

Рис. 13. Переход процесса *Initiator* в *INRES*-протоколе

– из места *queue* забирается фишка (h, t, id, sid) с минимальным значением первого поля; фишка (l, k) забирается из места *count_q* и возвращается в него со значением $(l + 1, k)$;
– в место *isap*2 помещается фишка $(c, IDISind, E, mid)$, где c — значение второго поля фишки в месте *count_is*2; из места *count_is*2 забирается фишка (h, c) и возвращается фишка $(h, c + 1)$.

6.2. Трансляция сложного перехода

Переходы, которые в сети невозможно представить одним переходом, будем называть *сложными*. Если переход процесса содержит операторы, использующие измененные в процессе выполнения перехода переменные, конструкции *decision*, *join*, *set*, *reset*, *save*, вызовы процедур, а также разрешающее условие, то процесс трансляции будет продолжен. На последующих этапах трансляции сложный переход разбивается на фрагменты. Фрагмент сложного перехода может быть вызовом процедуры, операторами *set*, *reset*, *save*, *decision*, либо последовательностью операторов, которая может быть смоделирована одним переходом сети.

Каждому фрагменту в сети сопоставляется переход. Данные перехо-

ды последовательно соединяются в том же самом порядке, что и соответствующие фрагменты, с помощью дополнительных служебных мест. Кроме того, создаются два служебных перехода, один из которых становится первым, а второй — последним в цепочке переходов, моделирующих сложный SDL-переход. Первый служебный переход далее будет называться *begin*, а второй — *end*. Они присоединяются к цепочке переходов, моделирующих фрагменты, также с помощью служебных мест. Служебное место может содержать бесцветную фишку. Изначально служебные места не размечены.

После того как переход разбит на фрагменты, трансляция каждого из них осуществляется отдельно. Если некоторая переменная используется во фрагменте сложного SDL-перехода, то соответствующее данной переменной место будет входным и выходным местом для перехода сети, моделирующего этот фрагмент. Если фрагмент содержит оператор *output*, то будет проведена дуга от сетевого перехода, который соответствует данному фрагменту, к месту, которое соответствует маршруту, в который передается сигнал, используемый в операторе *output*. Дополнительное место, которое служит для поддержания дисциплины FIFO-очереди среди сигналов, полученных через этот маршрут, будет входным и выходным для данного перехода. Служебное место *sender* также будет входным и выходным для этого перехода.

Процесс трансляции завершается, когда каждый из полученных фрагментов представляется одним сетевым переходом. Таким образом, выполнение сложного SDL-перехода в сети моделируется последовательным срабатыванием всех моделирующих его переходов, начиная с перехода *begin* и заканчивая переходом *end*.

Место *State* является входным для первого служебного перехода и выходным для второго служебного перехода. Состояние, указанное после служебного слова *State*, преобразуется в спусковую функцию первого служебного перехода. Фишка из места *State* забирается, как только срабатывает первый служебный переход, и возвращается в него после срабатывания второго служебного перехода. Таким образом, в сети не может сработать никакой переход, кроме моделирующих рассматриваемый SDL-переход. Наличие в сети места *State* превращает срабатывание нескольких переходов, соответствующих одному и тому же SDL-переходу, в непрерывное действие. Однако в процессе выполнения этих переходов сети ничто не мешает выполнению таких переходов сети, которые моделируют переходы других процессов. Но это не влияет на

правильность выполнения всей системы, так как выполнение разных процессов независимо.

Из места *queue* будет проведена дуга к первому служебному переходу *begin*. Дополнительное место для поддержания дисциплины FIFO-очереди будет входным и выходным для перехода *begin*. Сигнал, указанный в операторе *input*, преобразуется в спусковую функцию перехода *begin*. Место $self$ будет входным и выходным для первого служебного перехода, а значение его фишки также преобразуется в спусковую функцию перехода *begin*. Если какой-либо переход процесса воспринимает сигнал, под влиянием которого процесс прекращает функционирование, то в сети создано место *stop* — входное и выходное для служебного перехода *begin*, а значение его фишки преобразуется в спусковую функцию перехода *begin*.

7. ТРАНСЛЯЦИЯ ОПЕРАТОРОВ И ПРОЦЕДУР

Рассмотрим трансляцию конструкций *save*, *set*, *reset*, *decision*, разрешающего условия, непрерывного сигнала и процедур.

7.1. Моделирование конструкции *save*

Как было отмечено, сигналы из порта процесса извлекаются согласно дисциплине FIFO-очереди. В любом состоянии процесс должен извлечь из очереди тот сигнал, который прибыл в нее ранее всех остальных. Но иногда требуется, чтобы процесс, находясь в каком-либо состоянии, мог пропустить вперед сигнал, пришедший после стоящего впереди сигнала. В этом случае используется средство *save* — отсрочка восприятия того сигнала, который находится первым в очереди. Восприятие этого сигнала откладывается только до перехода процесса в следующее состояние. Если и в этом состоянии восприятие сигнала должно быть отсрочено, то вновь используется конструкция *save*. Если в некотором состоянии отсрочено восприятие нескольких сигналов, стоящих впереди подряд на первых местах в очереди, то средство сохранения сигнала применяется в отношении каждого из них.

Рассмотрим сеть, представленную на рис. 14, которая моделирует восприятие некоторым процессом сигнала Sig, сохраняя при этом все остальные сигналы в очереди. В данной сети представлены место *queue*, моделирующее порт процесса; дополнительное служебное место для организации очереди $count_q$; место q_{stack}, в которое будут помещаться

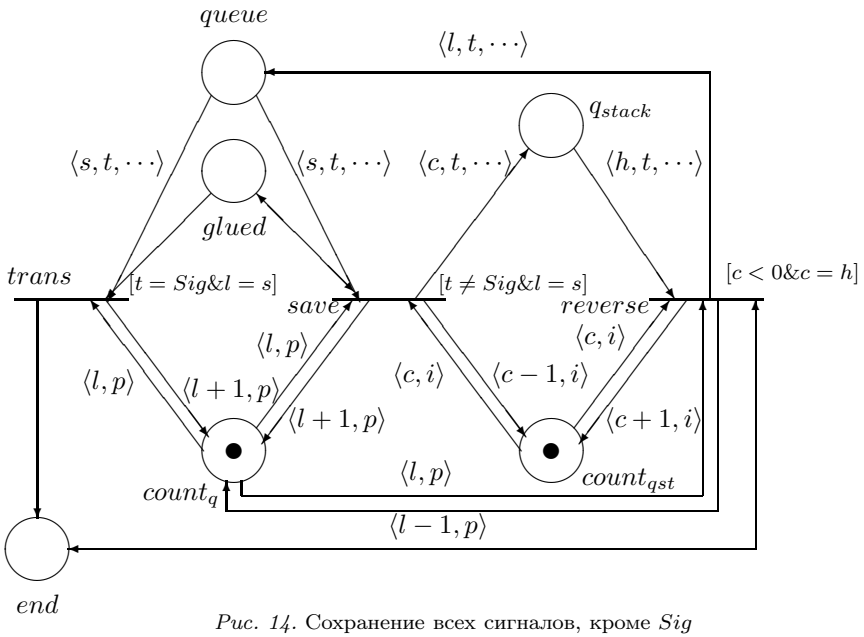

Рис. 14. Сохранение всех сигналов, кроме Sig

фишки, соответствующие отложенным сигналам; дополнительное место $count_{qst}$ для организации дисциплины очереди; служебное место *glued*. Последнее место может иметь бесцветную фишку.

Два служебных перехода *save* и *reverse* созданы для моделирования механизма отсрочки восприятия сигнала. Переход *save* может сработать, если значение второго поля фишки в месте *queue*, соответствующей первому сигналу в очереди сигналов, не совпадает с Sig. Срабатывание перехода *save* заключается в следующем:

– из места *queue* забирается фишка $(s, t, ...)$ (через "..."обозначены два поля, содержащие личные идентификаторы принимающего и отправляющего процессов) с минимальным значением первого поля; фишка (l, p) забирается из места $count_q$ и возвращается в него со значением $(l + 1, p)$;

– в место q_{stack} помещается фишка $(c, t...)$, где c — значение первого поля фишки в месте $count_{qst}$; из места $count_{qst}$ забирается фишка (c, i) и возвращается фишка $(c - 1, i)$;

– из места *glued* забирается бесцветная фишка и она же возвращается.

Переход *save* может срабатывать до тех пор, пока второе поле фиш-

ки, соответствующей первому сигналу в очереди, в месте *queue* не будет иметь значение, равное Sig. Каждый раз в место q_{stack} помещается фишка, значение первого поля которой соответствует минимальному номеру среди всех номеров, являющихся значением первого поля фишек в этом месте. Таким образом, переход *save* моделирует сохранение отложенных сигналов, запоминая их в другой очереди в порядке, обратном к тому, в каком они находились в порту процесса.

Переход *trans* может сработать, если значение второго поля фишки, соответствующей первому сигналу в очереди сигналов, совпадает с Sig. Срабатывание перехода *trans* заключается в том, что из места *queue* забирается фишка $(s, t, ...)$ с минимальным значением первого поля; фишка (l, p) забирается из места $count_q$ и возвращается в него со значением $(l + 1, p)$; из места *glued* забирается бесцветная фишка.

Переход *reverse* может сработать, если в месте *end* имеется фишка, а в месте q_{stack} — хотя бы одна. Срабатывание перехода *reverse* заключается в следующем:

– из места q_{stack} забирается фишка $(h, t, ...)$ с минимальным значением первого поля; фишка (c, i) забирается из места $count_{qst}$ и возвращается в него со значением $(c + 1, i)$;

– в место *queue* помещается фишка $(l, t...)$, где l — значение первого поля фишки в месте $count_q$; из места $count_q$ забирается фишка (l, p) и возвращается в него со значением $(l - 1, p)$;

– из места *end* забирается бесцветная фишка и она же возвращается.

При срабатывании перехода *reverse* в место *queue* помещается фишка, значение первого поля которой соответствует минимальному номеру среди всех номеров, которые являются значением первого поля фишек в этом месте. Таким образом, переход *reverse* моделирует возвращение в очередь процесса отложенных сигналов на то место, на каком они находились в очереди до применения конструкции *save*.

Функционирование сети (см. рис. 14) начнется тогда, когда место *glued* будет содержать фишку и место *queue* будет иметь хотя бы одну фишку. В процессе работы сети переход *save* может срабатывать до тех пор, пока в месте *queue* не окажется фишка, несущая сигнал Sig. После этого станет возможным переход *trans*, а в результате его срабатывания — переход *reverse*. Сеть на рис. 14 назовем *save*.

Таким образом,перебирая сигнал за сигналом в порядке их следо-

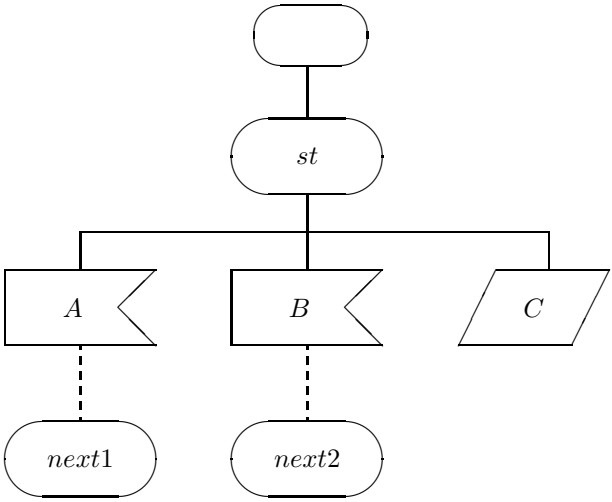

Рис. 15. SDL-фрагмент с конструкцией *save*

вания в очереди, процесс извлекает такой, который указан в качестве входного для того состояния, в котором находится процесс. На рис.15 показан сравнительно простой случай использования средства сохранения сигнала. Представим следующую ситуацию. Пусть в порт процесса прибыл сигнал C, а затем сигнал A. Процесс в такой ситуации оставит сигнал C на месте, а сигнал A будет извлечен из очереди, выполнится SDL-переход, воспринимающий сигнал A, и процесс перейдет в состояние $next1$. На рис. 16 представлена сеть, моделирующая механизм отсрочки восприятия сигнала для фрагмента, изображенного на рис. 15. В этой сети также представлено служебное место $State$ и дополнительный переход $return$, срабатывание которого забирает фишку из места end и возвращает фишку в место $State$. Во фрагменте на рис. 15 первый переход соответствует восприятию сигнала A, второй — B. Будем считать, что эти переходы моделируются подсетями (не одним переходом), обведенными штриховой линией с именами $trans\_1$ и $trans\_2$. После срабатывания служебного перехода end либо в подсети $trans\_1$, либо в $trans\_2$ в месте end появится соответственно фишка значения либо $next1$, либо $next2$.

Рассмотрим случай, когда второе поле фишки в месте $queue$, которая моделирует первый сигнал в очереди сигналов, имеет значение,

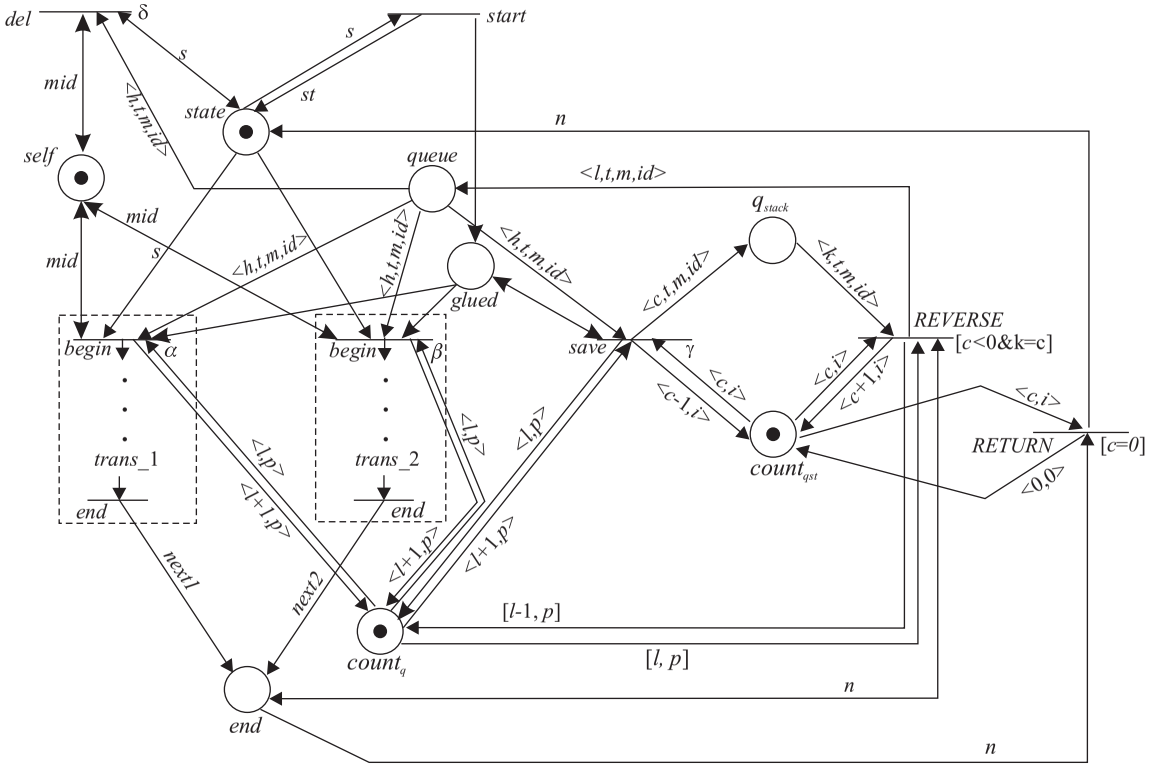

Рис. 16. Сеть для SDL-фрагмента, изображенного на рис. 15. Здесь

α — $[s=st\&t=A\&h=l\&(mid=m|m=E),]$

β — $[s=st\&t=B\&h=l\&(mid=m|m=E)]$,

γ — $[t=C\&h=l\&(mid=m|m=E)]$,

δ — $[s=st\&((t\neq A\&t\neq B\&t\neq C)|mid\neq m)]$.

соответствующее сигналу C, а в месте $State$ — фишка значения st. В этом случае в сети возможен переход $save$, его срабатывание приведет к тому, что из места $queue$ будет изъята, а в место q_{stack} помещена фишка, соответствующая сигналу C, в месте $count_{qst}$ фишка значения (0,0) будет заменена на фишку значения (-1,0). Если в месте $queue$ с минимальным номером находится фишка, снова соответствующая сигналу C, то переход $save$ сработает вновь, новая фишка из места $queue$ добавится с минимальным номером в место q_{stack} (т. е. фишка будет помещена в голову очереди), а в месте $count_{qst}$ появится фишка значения (-2,0).

Допустим, что теперь второе поле фишки в месте $queue$, которая моделирует первый сигнал в очереди сигналов, имеет значение, соответствующее сигналу A. Тогда в сети может сработать служебный переход $begin$ подсети $trans\_1$. Его срабатывание изымет фишку из места $State$. После срабатывания перехода end в этой же подсети фишка значения $next1$ появится в месте end. Становится возможным переход $reverse$, который может сработать столько раз, сколько фишек имеется в месте q_{stack}. При срабатывании он забирает фишку с минимальным номером из места q_{stack} и помещает ее с минимальным же номером в место $queue$.

После того как в месте q_{stack} не останется ни одной фишки, в сети может сработать переход $return$. Его срабатывание изымет фишку из места end и поместит ее в место $State$. Таким образом, в результате функционирования сети, изображенной на рис. 16, в месте $queue$ с минимальными номерами будут находиться фишки, соответствующие отложенным сигналам.

7.2. Моделирование разрешающего условия и непрерывного сигнала

Разрешающее условие — это булевское выражение, помещенное в угловые скобки после символа входа. Переход под воздействием некоторого входного сигнала In совершается, только если значение булевского выражения равно $true$. В противном случае процесс переходит к обработке следующего сигнала в очереди, а предыдущий остается на своем месте. Предположим, что входной сигнал In обрабатывается процессом в некотором состоянии $State$.

Коротко опишем отображение разрешающего условия. SDL-переход с разрешающим условием моделируется сетью, которая аналогична сети на рис. 16. Эта сеть содержит подсети, соответствующие SDL-перехо-

дам, совершающимися в состоянии *State* (на рис. 16 это подсети с именами *trans_1* и *trans_2*), переходы *save*, *reverse*, *return* сети рис. 16 и дополнительный служебный переход. Если последовательность действий, следующая после символа входа In, отображается одним сетевым переходом, то сигнал In и разрешающее условие преобразуются в его спусковую функцию, иначе — в спусковую функцию служебного перехода *begin* подсети, моделирующей данную последовательность действий. Спусковая функция дополнительного служебного перехода также определяется сигналом In и разрешающим условием. Этот переход может сработать, если в месте-очереди находится сигнал, соответствующий In, а значение булевского выражения равно $false$. Его срабатывание (как и перехода *start* на рис. 16) помещает бесцветную фишку в место *glued*, в результате чего может сработать переход *save*, моделирующий сохранение сигнала In в дополнительной очереди. Если SDL-переход, совершающийся в состоянии *State* и воспринимающий следующий сигнал в очереди, моделируется одним переходом в сети, то он может сработать после срабатывания перехода *save*, иначе может сработать служебный переход *begin* подсети, моделирующей данный SDL-переход. Таким образом, в случае ложности разрешающего условия входной сигнал In останется в очереди на прежнем месте, а процесс перейдет к обработке следующего в очереди сигнала.

Конструкция непрерывного сигнала может содержать переход, совершающийся под воздействием некоторого входного сигнала In, и несколько переходов, каждый из которых не содержит входного сигнала, но имеет булевское выражение с приоритетом. Находясь в некотором состоянии, процесс просматривает очередь сигналов. Если она пуста, то проверяются булевские выражения, входящие в эту конструкцию. Если истинно только одно из них, то выполняется последовательность действий, следующая за этим булевским выражением, иначе выполняется последовательность действий, следующая за выражением, имеющим максимальный приоритет. Если в очереди находятся сигналы, отличные от In, то под их воздействием выполнится пустой переход и они будут удалены из очереди.

На предыдущем этапе было создано дополнительное служебное место, являющееся входным и выходным для каждого перехода, моделирующего SDL-переход в конструкции непрерывного сигнала. Начальная разметка этого места — одна бесцветная фишка. Моделирование конструкции непрерывного сигнала отображается приоритетами пере-

ходов сети. Если SDL-переход с булевским выражением простой, то это выражение преобразуется в спусковую функцию соответствующего перехода, а приоритет — в его пометку, дополнительное служебное место будет для него входным и выходным. В противном случае булевское выражение преобразуется в спусковую функцию перехода *begin*, а служебное место будет для него входным. Это место будет выходным для перехода *end* подсети, моделирующей данный SDL-переход. Значение пометки перехода *begin* будет совпадать с приоритетом соответствующего булевского выражения. Аналогично, если SDL-переход с входным сигналом In простой, то приоритет соответствующего перехода устанавливается выше, чем приоритет перехода, моделирующего SDL-переход с булевским выражением, имеющим максимальный приоритет. В противном случае этот приоритет имеет служебный переход *begin* подсети, моделирующей данный SDL-переход. Все остальные переходы не помечаются, что соответствует наименьшему приоритету.

7.3. Моделирование средств управления временем

Моделирование оператора *set***.** Как было сказано в разд. 1, комплекс, реализующий систему SDL, должен обладать часами с абсолютным временем в согласованных единицах. Измерение отрезков времени осуществляется функцией *now*. Оператор $set(N, t)$ устанавливает в таймере t время N в принятых единицах. Новая установка таймера отменяет предыдущую.

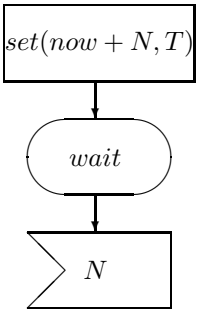

Рис. 17. Фрагмент с оператором *set*

На рис. 17 изображен фрагмент, реализующий получение сигнала от таймера t через N единиц времени. В этом фрагменте как только

с момента установки таймера, т. е. с момента исполнения оператора *set*, пройдет N единиц, в порт процесса поступит сигнал T. Процесс, находясь в состоянии *wait*, воспримет этот сигнал.

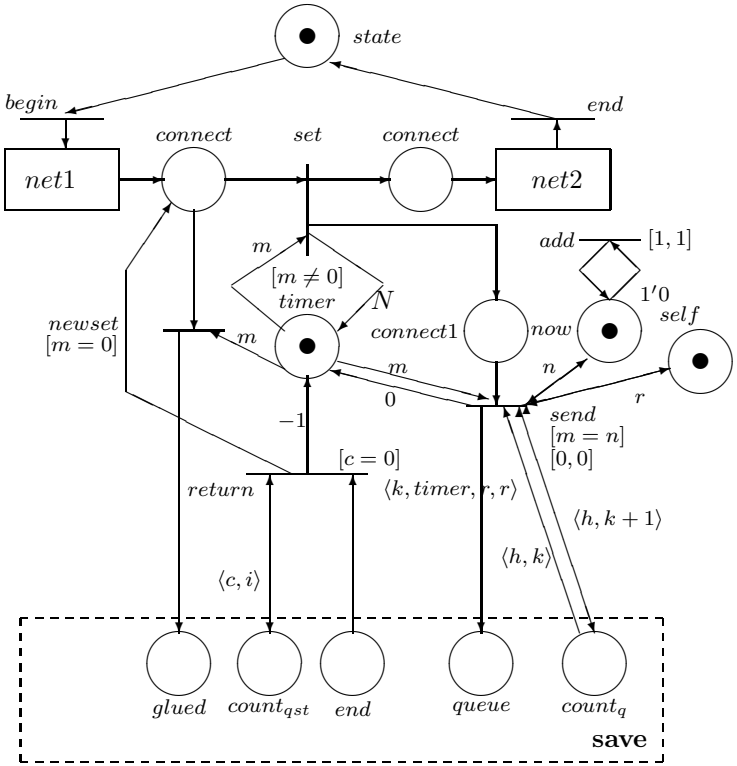

Рис. 18. Сеть для перехода, содержащего оператор *set*

Таймер считается активным с момента его установки и до момента восприятия процессом от него сигнала. Таймер можно перевести в неактивное состояние (говорят “сбросить таймер”) оператором *reset*. С момента инициации системы и до первой установки таймера он также считается неактивным.

Рассмотрим основные моменты моделирования SDL-перехода, содержащего оператор $set(N, timer)$. На рис. 18 представлена сеть, соответствующая SDL-переходу, содержащему оператор *set*. Сеть имеет

служебные переходы *begin* и *end* и подсети *net*1 и *net*2, моделирующие операторы, соответственно находящиеся до и после оператора *set* в SDL-переходе. В обведенном штриховой линией прямоугольнике (рис. 18) находится подсеть *save* (см. рис. 14), в которой спусковая функция перехода *trans* определяется сигналом от таймера. Чтобы не загромождать рисунок, дуги и переходы этой подсети не изображены, а показаны только новые дуги, которые соединяют элементы этой подсети с другими элементами.

В сети имеется место *now*, значение фишки которого определяет текущий момент в системе, начальная разметка — одна фишка значения 0. Переход *add* имеет интервал срабатывания [1,1]. Срабатывая, он забирает из места *now* фишку со значением текущего времени, ожидает одну единицу времени и выдает в место *now* фишку со значением, на единицу больше предыдущей. Подсеть содержит место *timer*, множеством цветов которого служит множество целых чисел. Начальная разметка этого места — одна фишка значения -1. Фишка такого цвета означает, что таймер находится в неактивном состоянии.

Служебное место *connect*1 соединяет переход *set* с переходом *send*, осуществляющим посылку сигнала от таймера в место *queue*, моделирующее очередь процесса. Переход *send* может сработать в том случае, если значение фишки в месте *timer* совпадает со значением фишки в месте *now*. Это соответствует тому, что значение в таймере стало равно текущему времени в системе. Интервал срабатывания перехода *send* равен $[0, 0]$, т. е. переход обязан сработать сразу, как только станет возможным. Все остальные переходы в сети, кроме перехода *add*, имеют интервал срабатывания $[0, \infty]$. Срабатывание перехода *send* заменит фишку в месте *timer* на фишку со значением 0 и добавит фишку $(k, timer, r, r)$ (где r — личный идентификатор данного процесса) в место *queue*. Таким образом, фишка значения 0 в месте *timer* означает, что сигнал от таймера находится в очереди процесса.

При потреблении сигнала от таймера каким-либо SDL-переходом в моделирующей этот SDL-переход подсети существует переход, срабатывание которого забирает фишку, соответствующую сигналу от таймера, из очереди и заменяет фишку в месте *timer* на -1, после чего таймер станет неактивным.

Поскольку новая установка таймера должна отменять предыдущую, в сети должен быть реализован механизм, с помощью которого в месте *timer* изменяется значение, если сигнал от таймера еще не был отправ-

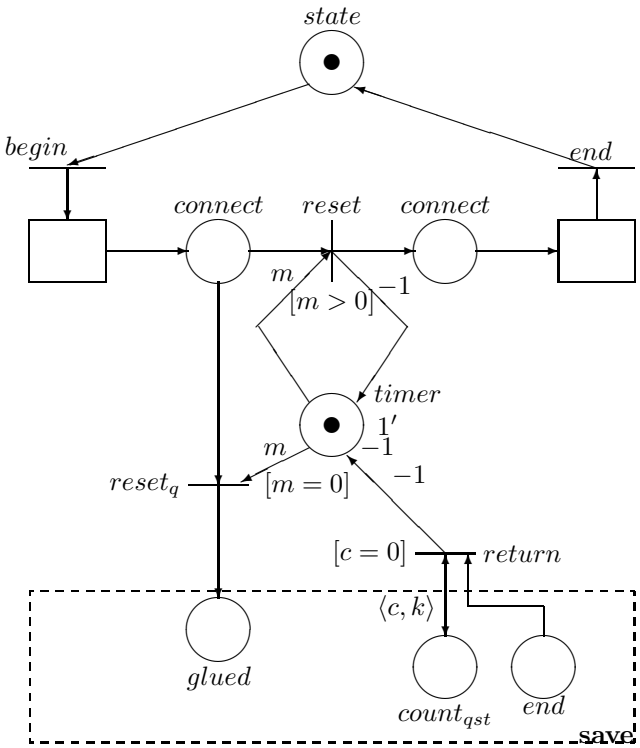

Рис. 19. Сеть для перехода, содержащего оператор *reset*

лен в очередь процесса, иначе предыдущий сигнал от таймера изымается из очереди процесса. Если значение фишки в месте *timer* не равно 0, может сработать переход *set*. В результате его срабатывания фишка в этом месте заменится на новую, соответствующую новому установленному времени. Иначе (т. е. если предыдущий сигнал от таймера находится в очереди процесса) в сети может сработать переход *newset*. При срабатывании перехода *newset* в месте *glued* (см. рис. 14) появляется фишка, что делает возможным срабатывание переходов *save*, *trans* и *reverse* в подсети *save*. Функционирование сети, названной *save*, реализует сохранение всех сигналов, находящихся в очереди, кроме сигнала от таймера. Переход *trans* подсети *save* может сработать, если значение второго поля фишки, соответствущей первому сигналу в очереди, совпадает с сигналом от таймера. После завершения срабатываний пе-

рехода *reverse* в сети может сработать переход *return*. После его срабатывания в месте *timer* появится фишка значения -1, в результате чего становится возможным переход *set*.

Моделирование оператора *reset***.** Перевод таймера в неактивное состояние осуществляется оператором *reset*. Оператор *reset* "сбрасывает"таймер в том случае, если сигнал от таймера еще не был послан в очередь процесса, и изымает сигнал из очереди процесса в противном случае. Сеть на рис. 19 моделирует SDL-переход с оператором *reset* и содержит в себе подсеть *save* аналогично сети, моделирующей оператор *set*. Подсеть *save* начинает функционировать после срабатывания перехода $reset_q$, который, в свою очередь, может сработать, если в месте *timer* находится фишка значения 0, — свидетельство того, что сигнал от таймера находится в очереди процесса. После срабатываний перехода *reverse* в подсети *save* может сработать переход *return*. Его срабатывание помещает фишку значения -1 в место *timer*. Переход *reset* может сработать в том случае, если в месте *timer* находится фишка значения строго больше 0. Это свидетельствует о том, что сигнал от таймера еще не был послан в очередь процесса. Срабатывая, переход *reset* изымает из места *timer* фишку и помещает в него фишку значения -1, т. е. таймер приводится в неактивное состояние.

7.4. Представление конструкции *decision*

Рассмотрим конструкцию *decision*, представленную на рис. 20.

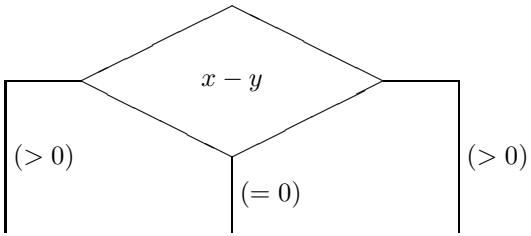

Рис. 20. Оператор *decision*

Как было отмечено ранее, последовательность действий, выполняемых процессом во время перехода, может разветвляться. Каждый ответ в конструкции *decision* будем называть *условием возможности ветви* перехода — последовательности действий, которая указана после ответа.

Если ветвь конструкции *decision* в сети может быть представлена одним переходом, то условие возможности этой ветви преобразуется в спусковую функцию соответствующего перехода. Если же ветвь перехода представляется подсетью, то в этой подсети первый служебный переход *begin* будет иметь спусковую функцию, соответствующую условию возможности этой ветви перехода.

На рис. 21 представлена сеть, соответствующая фрагменту с рис. 20, в котором предполагается, что каждая ветвь в сети моделируется одним переходом.

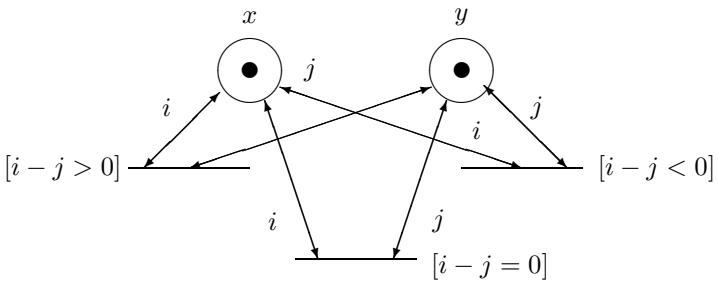

Рис. 21. Фрагмент сети, соответствующий оператору *decision*

7.5. Моделирование "уничтожения"процесса

Процесс перестает существовать, когда при переходе входит в состояние *stop*. Если один процесс хочет "уничтожить"другой, он посылает сигнал — назовем его *kill*, — под влиянием которого процесс переходит в состояние *stop*. Для моделирования "уничтожения"процесса в сети создаются место *stop* и переход *stp*, спусковая функция которого определяется только сигналом *kill*. При срабатывании этого перехода в место *stop* заносится фишка значения 1. При моделировании сложного SDL-перехода спусковая функция первого служебного перехода *begin*, а при моделировании простого SDL-перехода одним переходом сети спусковая функция этого перехода зависят от значения фишки в месте *stop*. Спусковая функция перехода *del* также зависит от значения фишки в этом месте. Таким образом, если это значение равно 1, то никакой переход в сети, моделирующей процесс, сработать не может. Такому процессу могут посылаться сигналы от других процессов, они будут накапливаться в очереди, но потребляться никакими переходами в сети не будут.

7.6. Трансляция процедур

Процедуры транслируются в отдельную сеть таким же способом, что и фрагмент перехода. Каждый переход, который представляет оператор вызова, заменяется сетью, моделирующей соответствующую процедуру. Места, моделирующие переменные, которые участвуют в вызове процедуры, являются входными местами первого перехода сети для данной процедуры. Этот переход вычисляет значения фактических параметров. Трансляция процедур SDL-системы нисколько не отличается от трансляции процедур Estelle-спецификации, описанной в работе [6], где дан следующий иллюстративный пример (см. рис. 22), моделирующий подстановку фактических параметров в момент вызова процедуры: $CALL\ cancel(u + v,\ s)$, где

```
procedure cancel;
FPAR IN b : Integer;
     IN/OUT a : Integer;
DCL  i : Integer;
START;
      {procedure body}
ENDPROCEDURE cancel;
```

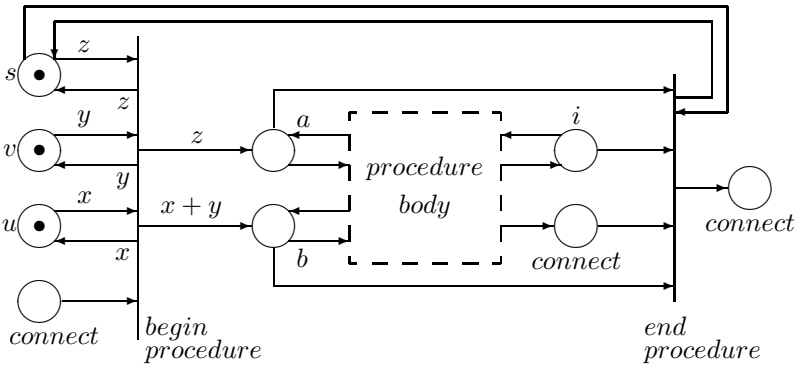

Рис. 22. Подстановка фактических параметров

Аналогичным образом осуществляется трансляция макрокоманд.

8. ВИД И ОЦЕНКА РАЗМЕРА СЕТИ

В ходе работы алгоритма, генерирующего сетевую модель, все переменные (в том числе обозреваемые и импортируемые), кроме массивов, отображаются в места, содержащие единственную фишку. Состояние процесса SDL представляется единственным значением фишки в месте *State*. Дополнительные служебные места $self$, *send*, *stop* имеют одну фишку. Кроме того, в сети существуют служебные места, отвечающие за поток управления в ней, которые в любой момент времени имеют не более одной фишки.

Таким образом, алгоритм генерирует сети, в которых все места, за исключением соответствующих массивам и маршрутам/каналам, могут содержать не более одной фишки.

Кроме того, моделирующая сеть становится "густой", так как имеет много дуг: многие SDL-переходы используют одни и те же переменные, поэтому соответствующие сетевые переходы имеют одни и те же входные и выходные места. Если SDL-переход моделируется одним сетевым переходом, то место *State* будет входным и выходным для таких сетевых переходов. Иначе место *State* будет входным для перехода *begin* в подсетях, моделирующих переход, и выходным для перехода *end*.

Рассмотрим оценку размера сети для процесса, который содержит var переменных, par параметров и который имеет m входных и выходных маршрутов. Кроме того, пусть данный процесс содержит n операторов, среди которых k_1 операторов *decision*, k_2 операторов *set*, k_3 операторов *reset*, k_4 конструкций *save*, D определений процедур и C вызовов процедур. Обозначим

$k = k_1 + k_2 + k_3 + k_4$,

$att = var + 5 + 2 * (m + 1) + par$.

Тогда сеть, моделирующая такой процесс, будет иметь максимально TN переходов и PN мест, где

$TN = (n + 2m + 5 * k) * C * D$,

$PN = n + (att + 7 * k) * C * D$.

Если процесс не содержит процедур, то оценка становится линейной.

9. ЗАКЛЮЧЕНИЕ

В работе описана процедура трансляции статических SDL-спецификаций в расширенные цветные сети Петри. Расширения этих сетей временным механизмом и приоритетами естественно и позволяет развить средства симуляции и анализа. Выбор SDL-спецификаций с использованием конструкции таймера позволяет представить достаточно широкий класс протоколов, а отсутствие динамических конструкций упрощает сетевую модель.

Метод трансляции этих спецификаций в модели приводит к квазибезопасным сетям, т. е. таким, в которых все места, исключая соответствующие массивам, маршрутам и каналам, содержат не более одной фишки, что позволяет существенно повысить эффективность моделирования, так как исключает перебор вариантов связывания переменных. Все конструкции описаны с некоторым неформальным обоснованием, приведены верхние оценки размера моделирующей сети.

Данная работа является частью большого проекта, цель которого — валидация коммуникационных протоколов. Проект содержит в себе две части. Первая предназначена для автоматического построения сетевой модели с использованием описанной в данной работе процедуры трансляции, вторая — система NetCalc — позволяет редактировать и симулировать построенную сетевую модель.

Автор выражает благодарность В. А. Непомнящему за постоянное внимание и поддержку, Е. В. Окунишниковой за совместную плодотворную разработку алгоритма трансляции Estelle-спецификаций в раскрашенные сети Петри, идеи которого использованы в этой работе, Г. И. Алексееву за критику и конструктивные предложения.

СПИСОК ЛИТЕРАТУРЫ

1. **Recommendation** Z.100 CCITT Specification and Description Language (SDL)
2. **Карабегов А.В., Тер-Микаэлян Т.М.** Введение в язык SDL.— М.: Радио и связь, 1993.
3. **Котов В. Е.** Сети Петри. — М.: Наука, 1984.
4. **Окунишникова Е. В., Чурина Т. Г.** Способ построения раскрашенных сетей Петри, моделирующих Estelle-спецификации // Проблемы спецификации и верификации параллельных систем. — Новосибирск, 1995. — С. 95—123.
5. **Churina T. G., Okunishnikova E. V.** Coloured Petri nets approach to the validation of Estelle specifications // Proc. of Workshop on Concurrency, Specification and Programming. — Warsaw, Poland, 1997. — P. 25—36.
6. Верификация Estelle-спецификаций распределенных систем посредством раскра-

шенных сетей Петри /В.А. Непомнящий, А.В. Быстров и др. — Новосибирск, 1998. — 140 с.
7. **Fisher J., Dimitrov E.** Verification of SDL'92 specifications using extended Petri nets // Proc. IFIP 15th Intern. Conf. on Protocol Specification, Testing and Verification. — Warsaw, Poland, 1995. — P. 455—458.
8. **Algayres B. et al.** VESAR: a pragmatic approach to formal specification and verification // Computer Networks and ISDN Systems. — 1993. — Vol. 25, N 7. — P. 779—790.
9. **Berthomieu B., Diaz M.** Modelling and verification of time dependent systems using time Petri nets // IEEE Transact. on Software Eng. — 1991. — Vol. 17, N 3. — P. 259—273.
10. **Jensen K.** Coloured Petri nets: Basic concepts, analysis methods and practical use. — Berlin a. o.: Springer-Verlag, 1996. — Vol. 1. Basic concepts.
11. **Jensen K.** Coloured Petri nets: Basic concepts, analysis methods and practical use. — Berlin a. o.: Springer-Verlag, 1996. — Vol. 2. Analysis methods.
12. **Jensen K.** Coloured Petri nets: Basic concepts, analysis methods and practical use. — Berlin a. o.: Springer-Verlag, 1997. — Vol. 3. Practical use.
13. **Marchena S., Leon G.** Transfomation from LOTOS specs to Galileo nets // Intern. Conf. on Formal Description Techniques I. — Amsterdam: North-Holland, 1989. — P. 217—230.
14. **Kettunen E., Montonen E., Tuuliniemi T.** An interactive PrT-Net tool for verification of SDL-specifications: Techn. Rep. No.3. — Helsinki Univ. of Technology, Digital System Laboratory, 1988.
15. **Bause F. et al.** SDL and Petri net perfomance analysis of communicating systems // Proc. IFIP 15th Intern. Symp. on Protocol Specification, Testing and Verification. — Warsaw, Poland, 1995. — P. 259—272.
16. **van der Aalst W. M. P.** Interval timed coloured Petri nets and their analysis // Lect. Notes Comput. Sci. — 1993. — Vol. 691. — P. 453—472.
17. **Ferenc B., Hogrefe D., Sarma A.** SDL with applikations from protocol specification. — Englewood Cliffs, NJ: Prentice Hall, 1991.

Т. Г. Чурина

СПОСОБ ПОСТРОЕНИЯ РАСКРАШЕННЫХ СЕТЕЙ ПЕТРИ, МОДЕЛИРУЮЩИХ SDL-СИСТЕМЫ

Препринт
56

Рукопись поступила в редакцию 12.10.1998
Рецензент В. В. Окольнишников
Редактор Л. А. Карева

Подписано в печать 05.02.1999
Формат бумаги 60×84 1/16 Объем 3,3 уч.-изд.л., 3,5 п.л.
Тираж 50 экз.

Отпечатано на ризографе “AL Group”, 630090, пр. Акад. Лаврентьева, 3