Математическая модель и методы верификации программных систем

А. М. Миронов, Д. Ю. Жуков

В статье вводится новая математическая модель программных систем, и рассматривается проблема их формальной верификации. Предлагаются методы редукции анализируемых систем. Изложенные подходы иллюстрируются примером формальной верификации протокола передачи сообщений через ненадежную среду.

1. Введение

В настоящей статье мы излагаем новую математическую модель программных систем, и основанные на данной модели методы формальной верификации. Данная модель обеспечивает возможность верификации программных систем с большим числом состояний.

Главная проблема при формальной верификации программных систем заключается в экспоненциальном росте числа их состояний в зависимости от размера их описания. Большинство работ в области верификации программных систем связано с построением их полных или частичных диаграмм состояний. Данный подход годится только для программных систем небольшого размера, и неприемлем для верификации сложных программных систем.

В том случае, когда число состояний исследуемой программной системы является слишком большим, и формальная верификация их путем применения таких методов анализа систем с конечным числом состояний, как например model checking (см. [1]), невозможна по причине высокой сложности таких систем, то представляется разумным рассматривать такие системы как системы с потенциально бесконеч-

ным числом состояний, и анализировать их методами, которые не связаны с поиском в конечном множестве состояний.

Существует несколько подходов к верификации систем с бесконечным числом состояний. Некоторые из них основаны на технологии разбиения бесконечного множества состояний исходной системы на конечное множество классов эквивалентности (см. например [4, 5, 6]). Другие подходы основаны на понятии бимоделирования (см. например [7]). В настоящей работе мы объединяем методы верификации систем с бесконечным числом состояний, основанные на понятии бимоделирования, с методами верификации, основанными на построении индуктивных утверждений (см. [2]), а также предлагаем методы редукции для понижения сложности процедуры верификации.

2. Программные системы

Программная система (называемая ниже более коротко **системой**) представляет собой конечную совокупность Σ программ, взаимодействующих друг с другом посредством передачи сообщений. Каждое сообщение представляет собой список значений. Множество всех возможных значений будет обозначаться символом $\mathcal{D}$, а множество всех списков значений — символом $\mathcal{D}^*$.

2.1. Понятие программы

Программа представляется некоторой блок-схемой, каждая вершина n которой помечена некоторым оператором $Op(n)$. Вид этих операторов определяется ниже.

С каждой программой P связано некоторое множество $Var(P)$ ее **переменных**, причем для разных программ их множества переменных не пересекаются. Каждой переменной x сопоставлено множество ее возможных значений $\mathcal{D}_x \subseteq \mathcal{D}$, которые может принимать переменная x. Если X — список переменных вида

$$(x_1, \ldots, x_n),$$

то символ $\mathcal{D}_X$ обозначает декартово произведение

$$D_{x_1} \times \ldots \times D_{x_n}.$$

Кроме того, считается заданным некоторое множество функциональных символов

$$+, \quad -, \quad \textbf{head}, \quad \textbf{tail}, \quad \ldots,$$

при помощи которых можно стандартным образом строить выражения с переменными. Если каждой переменной x, входящей в выражение e, сопоставлено некоторое значение $\xi(x)$, то все выражение e принимает значение $\xi(e)$, определяемое стандартным образом. Если E — список выражений вида

$$(e_1, \ldots, e_n), \tag{1}$$

и каждой переменной x, входящей в какое-либо из выражений списка E, сопоставлено некоторое значение $\xi(x)$, то символ $\xi(E)$ обозначает список соответствующих значений

$$(\xi(e_1), \ldots, \xi(e_n)).$$

Ниже для каждой программы P

- символ $Tm(P)$ будет обозначать совокупность всех выражений с переменными из $Var(P)$
- символ $Fm(P)$ будет обозначать совокупность тех выражений из $Tm(P)$, которые принимают только булевские значения (0 или 1). Выражения из $Fm(P)$ называются **формулами**.

Операторы, входящие в блок-схему, могут иметь следующий вид.

начало:

$$Op(n) = \left(\begin{array}{c} \textbf{start} \\ Init(P) \end{array} \right), \tag{2}$$

где $Init(P)$ — формула, называемая **начальным условием** программы P.

присваивание:

$$Op(n) = \left(\; x := e \; \right), \tag{3}$$

где

- x — некоторая переменная, и
- e — некоторое выражение.

проверка условия:

$$Op(n) = \quad \left(\; b \; \right), \tag{4}$$

где b — некоторая формула.

посылка сообщения:

$$Op(n) = \quad \left(\; P' \leftarrow E \; \right), \tag{5}$$

где

- P' — имя программы, которой посылается сообщение, и
- E — список выражений, значения которых посылаются программе P'.

получение сообщения:

$$Op(n) = \quad \left(\; X \leftarrow P' \; \right), \tag{6}$$

где

- P' — имя программы, от которой приходит сообщение, и
- X — список различных переменных, в которые записываются полученные значения.

выбор:

$$Op(n) = \quad \left(\qquad \right) \tag{7}$$

(пустая метка).

остановка:

$$Op(n) = \quad \left(\; \textbf{halt} \; \right). \tag{8}$$

Из вершин с меткой вида (2), (3), (5), (6) выходит только одно ребро. Из вершин с меткой вида (4) выходят два ребра: одно имеет метку «+», другое — метку «−». Из вершин с меткой (7) может выходить произвольное количество ребер. Из вершин с меткой (8) не выходит ни одного ребра.

2.2. Функционирование программы

Функционирование программы P происходит обычным образом, и заключается в обходе вершин блок-схемы, с выполнением операторов, сопоставленных проходимым вершинам. После выполнения

оператора, соответствующего текущей вершине, происходит переход по выходящему из нее ребру к следующей вершине. На каждом шаге t функционирования ($t = 0, 1, \ldots$) каждая переменная x программы содержит некоторое значение $\xi_t(x)$. Значения переменных в начальный момент должны удовлетворять начальному условию.

Операторы выполняются следующим образом.

- Оператор (3) заносит значение выражения e в переменную x.
- Оператор (4) вычисляет значение формулы b, и
  - если оно равно 1, то происходит переход к следующей вершине по ребру с меткой «+»,
  - иначе - по ребру с меткой «−».
- Оператор (5) выполняется путем
  - ожидания момента, когда программа P' будет готова принять сообщение от P,
  - как только наступает такой момент, программе P' посылается список значений выражений, входящих в E.
- Оператор (6) выполняется путем
  - ожидания момента, когда от программы P' поступит некоторое сообщение,
  - как только наступает такой момент, значения из поступившего сообщения записываются в переменные из списка X.
- Если текущая вершина помечена оператором (7), то
  - из ребер, которые из нее выходят, выбирается ребро, ведущее в вершину, помеченную таким оператором, который возможно выполнить, и
  - происходит переход в эту вершину.
- Оператор (8) завершает выполнение всей программы.

3. Пример системы

В этом пункте мы рассмотрим пример системы, представляющей собой реализацию протокола передачи данных через ненадежную среду.

Эта система будет обозначаться символом *UBP* (Unbounded Buffer Protocol).

3.1. Описание системы *UBP*

Задача *UBP* заключается в том, чтобы получать сообщения от программы *In*, являющейся внешней по отношению к системе *UBP*, и доставлять их в том же порядке и без искажений программе *Out* (тоже являющейся внешней по отношению к системе *UBP*). Предполагается, что в процессе доставки передаваемые сообщения проходят через среду (мы будем считать ее частью системы *UBP*), в которой может произойти частичное искажение передаваемых сообщений. Система *UBP* должна уметь распознавать искажения и перепосылать те сообщения, которые были искажены.

UBP состоит из четырех программ:

$$Sender, \quad Receiver, \quad Buffer_1 \quad \text{и} \quad Buffer_2,$$

взаимодействующих друг с другом так, как показано на рис. 1. Программы $Buffer_1$ и $Buffer_2$ являются моделью среды, через которую передаются сообщения.

Для программ, входящих в *UBP*, мы будем использовать наряду с их полными именами также сокращенные имена: S, R, B_1 и B_2 соответственно.

3.2. Программа *Sender*

Программа *Sender* изображена на рис. 2.

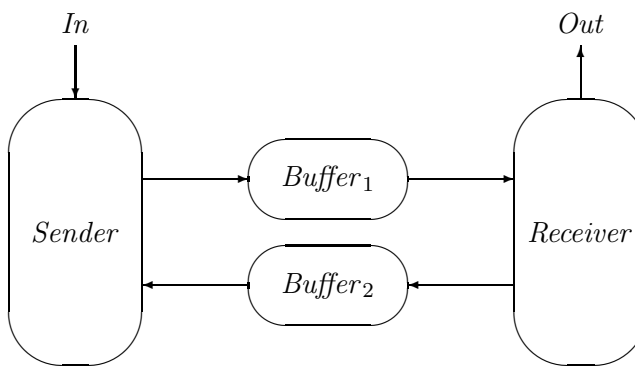

Рис. 1. *UBP*.

Sender имеет массив s, в который он записывает сообщения, поступающие к нему от программы *In*. Получив очередное сообщение g, программа *Sender*

- сопоставляет ему порядковый номер in $(= 1, 2, \ldots)$,
- записывает в $s[in]$ полученное сообщение,
- посылает программе $Buffer_1$ пакет, представляющий собой пару (g, in).

Также *Sender* может получать пакеты от программы $Buffer_2$, которые являются подтверждениями, посылаемыми ему программой *Receiver*:

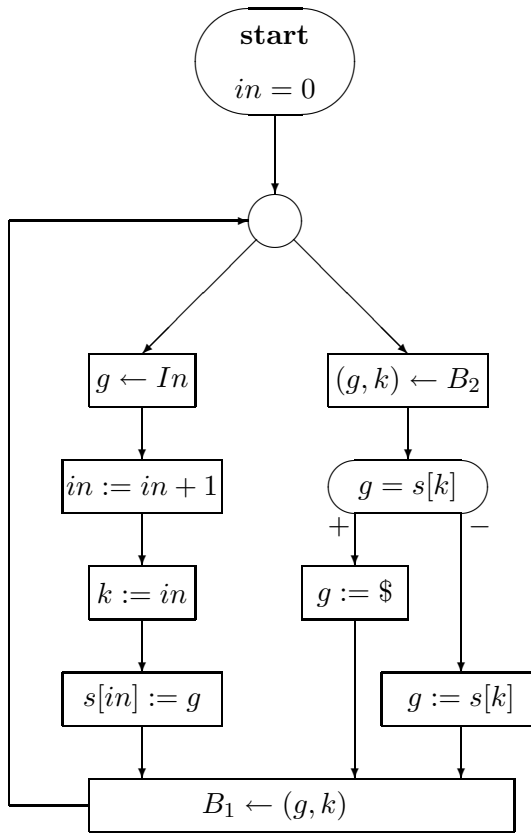

Рис. 2. *Sender*.

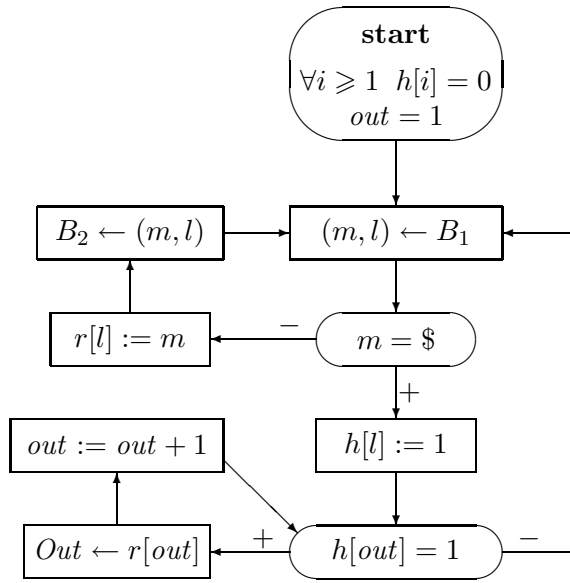

Рис. 3. *Receiver*.

- если полученный пакет имеет вид (g, k), и g совпадает с $s[k]$, то это значит, что *Receiver* получил сообщение с номером k без искажения, и в этом случае *Sender* посылает программе $Buffer_1$ пакет

$$(\$, k),$$

где $ — специальный символ, который не может быть искажен в процессе передачи,

- если же полученный пакет имеет вид (g, k), и g не совпадает с $s[k]$, то это означает, что сообщение с номером k возможно было искажено в процессе передачи, и в этом случае *Sender* посылает программе $Buffer_1$ пакет $(s[k], k)$ еще раз.

3.3. Программа *Receiver*

Программа *Receiver* изображена на рис. 3.

Receiver имеет массив r, в который он записывает сообщения, поступающие к нему от программы $Buffer_1$:

- если *Receiver* получил пакет (m, l), где m не совпадает с «\$» (это означает, что еще пока нет уверенности в том, что сообщение с номером l дошло без искажений), то *Receiver*
  - пересылает этот пакет (в качестве подтверждения) программе $Buffer_2$,
  - записывает значение m в $r[l]$,
- если же полученный пакет имеет вид $(\$, l)$, то это означает, что сообщение с номером l было получено программой *Receiver* без искажения, и его можно переслать программе *Out*.

Receiver имеет булевозначный массив h, где для каждого $i \geqslant 1$ $h[i] = 1$ если i-е сообщение было передано от программы *Sender* программе *Receiver* без искажения.

3.4. Программы $Buffer_1$ и $Buffer_2$

Программа $Buffer_1$ изображена на рис. 4. Данная программа изображает среду, через которую передаются пакеты.

$Buffer_1$ представляет собой неограниченный буфер, который может содержать произвольную совокупность пакетов вида (d, k), где

- d — **тело** пакета (некоторое значение),
- k — **номер** пакета (натуральное число).

Пакеты, содержащиеся в буфере, могут переупорядочиваться. Кроме того, тела этих пакетов могут искажаться, но их номера искажаться не могут. Мы предполагаем, что имеется **искажающая функция**

$$f : \mathcal{D} \to \mathcal{D}$$

такая, что $f(\$) = \$$, и

$$\forall d \in \mathcal{D} \qquad (\ f(d) \neq d \quad \Rightarrow \quad \forall k \geqslant 1 \quad f^k(d) \neq d\ ). \tag{9}$$

$Buffer_1$ имеет переменную x_1, в которой содержатся пакеты, полученные от программы *Sender*, и пока еще не посланные программе

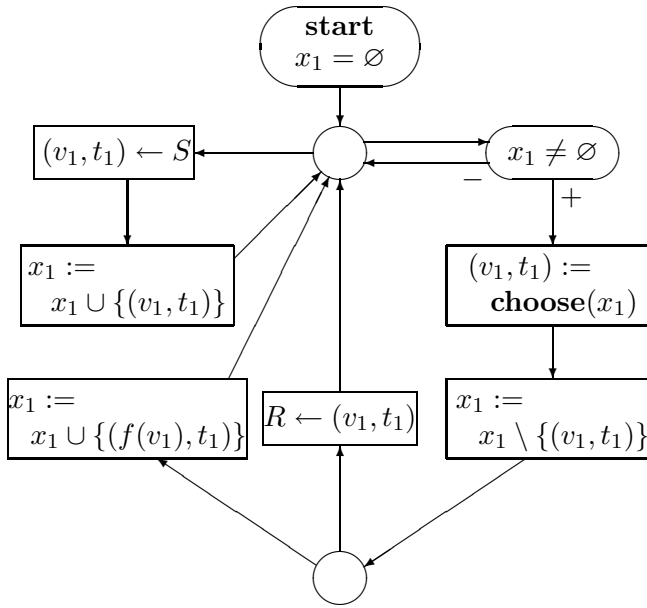

Рис. 4. *Buffer*$_1$.

Receiver (то есть каждое возможное значение переменной x_1 представляет собой *множество* пакетов).

Функция **choose** в программе *Buffer*$_1$ является недетерминированной, и если значение x_1 в текущий момент времени является непустым множеством, то значение **choose**(x_1) есть произвольный пакет из x_1.

Программа *Buffer*$_2$ полностью идентична программе *Buffer*$_1$, за исключением того, что она принимает пакеты от *Receiver*а, а посылает их *Sender*у. Ее переменные мы будем обозначать символами x_2, v_2, t_2.

4. Спецификация и верификация систем

4.1. Спецификация систем

Спецификация системы — это формальное описание некоторых свойств ее поведения.

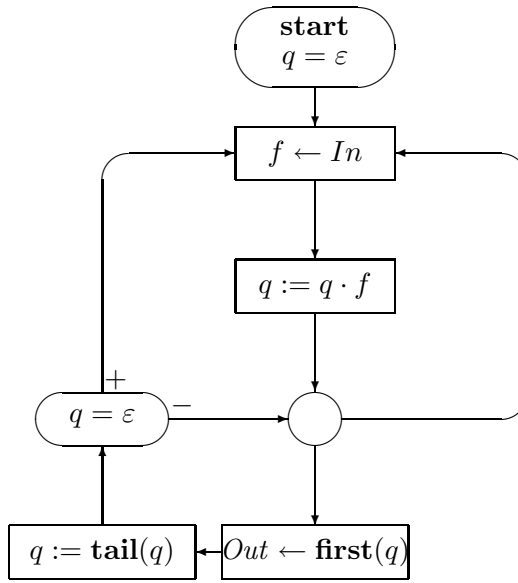

Рис. 5. *Spec*.

Например, одно из свойств системы *UBP* заключается в том, что она работает как очередь типа FIFO («First Input — First Output»), что означает следующее: последовательность сообщений, передаваемых от *UBP* программе *Out*, совпадает с последовательностью сообщений, которую *UBP* получает от программы *In*.

Более формально данное свойство можно выразить следующим образом: внешнее поведение *UBP* эквивалентно поведению программы *Spec*, изображенной на рис. 5.

Spec представляет собой неограниченный FIFO буфер, который

- не искажает полученные сообщения, и
- выдает их программе *Out* в том же порядке, в котором они в него поступают от программы *In*.

Spec имеет переменную q, содержащую список сообщений, которые были получены, но пока еще не выданы.

После получения от программы *In* нового сообщения, *Spec* добавляет его в конец списка q.

В программе *Spec* используются следующие обозначения:

- символ «$\cdot$» обозначает операцию конкатенации,
- символ «ε» обозначает пустой список,
- знакосочетание $\mathbf{first}(q)$ обозначает первый элемент списка q, и
- знакосочетание $\mathbf{tail}(q)$ обозначает список, получаемый из q путем удаления его первого элемента.

4.2. Верификация систем

Проблема **верификации** (то есть **формального анализа**) системы заключается в построении формального доказательства того, что данная система соответствует своей спецификации.

Решение данной проблемы возможно только на основе рассмотрения анализируемых систем как математических объектов. Для этого необходимо

1) построить математическую модель систем,
2) формализовать понятие соответствия системы ее спецификации, и
3) разработать в рамках данной модели формальные процедуры проверки соответствия системы своей спецификации.

Оставшаяся часть работы посвящена

- построению данной модели, и
- разработке методов формального анализа систем в рамках данной модели.

5. Формальные модели систем и программ

5.1. Модель системы

Модель системы представляет собой список Σ вида

$$\Sigma = (P_1, \ldots, P_n), \tag{10}$$

состоящий из графовых моделей программ, входящих в данную систему.

5.2. Графовые модели программ

Пусть P — некоторая программа.

Графовая модель программы P представляет собой ориентированный граф с выделенной вершиной $Start(P)$.

Множество вершин данного графа обозначается символом $N(P)$, а множество его ребер — $F(P)$.

Каждое ребро $f \in F(P)$ имеет метку $\langle f \rangle$, которая называется **действием**. Действие может содержать следующие компоненты (каждая из которых может отсутствовать):

1) **Условие:**
$$\varphi\ ?, \tag{11}$$
где φ — некоторая формула из $Fm(P)$.

2) **Посылка или прием сообщения**:
$$P' \ \leftarrow\ E \qquad \text{(посылка сообщения)} \tag{12}$$
или
$$X \ \leftarrow\ P' \qquad \text{(прием сообщения)}, \tag{13}$$
где
- P' — некоторая программа,
- E — список выражений из $Tm(P)$,
- X — список различных переменных из $Var(P)$.

3) **Подстановка**, то есть отображение вида
$$\theta : Var(P) \to Tm(P). \tag{14}$$

Мы будем предполагать, что для каждой программы P

- множество ее переменных $Var(P)$ содержит специальную переменную at_P,
- в каждой графовой модели программы P переменная at_P принимает значения в множестве $N(P)$ вершин данной модели, и
- для каждого ребра $f \in F(P)$ компонента (14) метки $\langle f \rangle$ удовлетворяет следующему условию:
$$\theta(at_P) = n,$$
где вершина n является концом ребра f.

Подстановку (14) мы будем обозначать также записью вида

$$(e_1/x_1, \ldots, e_n/x_n), \tag{15}$$

где

- список

$$x_1, \ldots, x_n \tag{16}$$

  состоит из всех тех переменных из множества

$$Var(P) \setminus \{at_P\}, \tag{17}$$

  для которых $x_i \neq \theta(x_i)$, и
- для каждого $i = 1, \ldots, n$

$$e_i = \theta(x_i).$$

Ниже для каждой подстановки θ вида (14) и каждого выражения $e \in Tm(P)$ мы будем обозначать символом $\theta(e)$ выражение, получаемое заменой каждой переменной x в e на выражение $\theta(x)$. Если E — список выражений вида (1), то символ $\theta(E)$ обозначает список

$$(\theta(e_1), \ldots, \theta(e_n)).$$

Если у действия отсутствует компонента, связанная с приемом или передачей сообщения, то такое действие называется **внутренним**.

Если компоненты «условие» или «подстановка» не указаны, то они по умолчанию имеют следующий вид:

- компонента «условие» есть тавтология (например, формула $1 = 1$),
- компонента «подстановка» действует тождественно на все переменные из (17) (то есть список (16) является пустым).

5.3. Функционирование графовой модели

Понятие **функционирования** графовой модели аналогично понятию функционирования блок-схемы, и заключается в обходе вершин графовой модели (начиная с вершины $Start(P)$), с выполнением действий, являющихся метками проходимых ребер.

На каждом шаге t функционирования ($t = 0, 1, \ldots$) каждой переменной x из $Var(P)$ сопоставлено некоторое значение $\xi_t(x)$. Значения переменных в начальный момент времени должны удовлетворять начальному условию.

На каждом шаге t функционирования графовой модели значение переменной at_P равно той вершине, в которой на данном шаге мы находимся. В частности,

$$\xi_0(at_P) = Start(P).$$

Соответствие ξ_t между переменными и их значениями на шаге t можно рассматривать как отображение

$$\xi_t : Var(P) \to \mathcal{D}.$$

Данное отображение называется **состоянием** программы P в момент времени t. Совокупность всех возможных состояний программы P обозначается символом $S(P)$.

Ниже для каждого состояния ξ и каждого выражения e из $Tm(P)$ знакосочетание $\xi(e)$ обозначает значение, получаемое

- заменой каждой переменной x в e на значение $\xi(x)$, и
- применением в получившемся выражении операций, соответствующих функциональным символам, входящим в e.

Каждый шаг t функционирования графовой модели P начинается с выбора такого ребра f с началом в $\xi_t(at_P)$, что действие $\langle f \rangle$ возможно выполнить. После этого выполняется действие $\langle f \rangle$, и происходит переход в вершину, являющуюся концом ребра f. В этой вершине мы будем находиться в момент $t + 1$.

Действие $\langle f \rangle$ выполняется путем последовательного выполнения всех его компонентов: сначала выполняется первая компонента, потом — другая (если она есть), затем — третья (если она есть).

Компонента (11) выполняется путем вычисления значения формулы φ, и

- если оно равно 0, то выполнение всего действия является невозможным,
- иначе — выполнение действия продолжается.

Выполнение компонентов (12) и (13) происходит аналогично выполнению соответствующих операторов в блок-схемах.

Выполнение компоненты (15) происходит путем

- вычисления значения выражений $e_1, \ldots, e_n$, и
- занесения этих значений в переменные $x_1, \ldots, x_n$ соответственно, то есть

$$\xi_{t+1}(x_1) \stackrel{\text{def}}{=} \xi_t(e_1), \quad \ldots, \quad \xi_{t+1}(x_n) \stackrel{\text{def}}{=} \xi_t(e_n).$$

5.4. Преобразование программ в их графовые модели

Алгоритм построения по блок-схеме программы P ее графовой модели имеет следующий вид.

1) На каждом ребре блок-схемы рисуется точка.
2) Для каждой вершины n блок-схемы, и каждой пары F_1, F_2 ребер блок-схемы, таких что F_1 входит в n, а F_2 - выходит из n
   а) рисуется ребро f, соединяющее точку на F_1 с точкой на F_2,
   б) на этом ребре f рисуется метка $\langle f \rangle$, определяемая следующим образом:
      – если $Op(n)$ имеет вид (3), то

$$\langle f \rangle \stackrel{\text{def}}{=} \left(\ e/x \ \right)$$

      – если $Op(n)$ имеет вид (4), и ребро, выходящее из n, помечено символом «+», то

$$\langle f \rangle \stackrel{\text{def}}{=} \left(\ b \ ? \ \right)$$

– если $Op(n)$ имеет вид (4), и ребро, выходящее из n, помечено символом «$-$», то

$$\langle f \rangle \stackrel{\text{def}}{=} \left(\ \neg b \ ? \ \right)$$

– если $Op(n)$ имеет вид (6) или (5), то $\langle f \rangle$ совпадает с $Op(n)$,
– если $Op(n)$ имеет вид (7), то действие $\langle f \rangle$ пусто (не содержит ни одной компоненты).

3) Если имеется хотя бы одно ребро f с пустым действием $\langle f \rangle$, то
   а) обозначим начало и конец ребра f символами ν_1 и ν_2 соответственно,
   б) каждое из ребер, выходящее из ν_2, преобразуем в ребро с той же меткой, выходящее из ν_1,
   в) удалим ребро f и точку ν_2.

   Данная операция повторяется до тех пор, пока не останется ни одного ребра f с пустым действием.

4) Оставшиеся точки являются вершинами графовой модели.

5) $Start(P)$ есть точка, нарисованная на ребре, выходящем из начальной вершины блок-схемы.

Например, графовые модели программ из системы *UBP*, а также программы *Spec*, имеют вид, изображенный на рис. 6, 7, 8, и 9.

Начиная со следующего пункта, мы будем называть графовые модели программ просто **программами**.

6. Редукция программ

Сложность процедуры верификации систем зависит от сложности их моделей: чем меньше размер модели, тем проще ее верифицировать.

В данном пункте мы излагаем один метод преобразования программ, который позволяет уменьшать их размер.

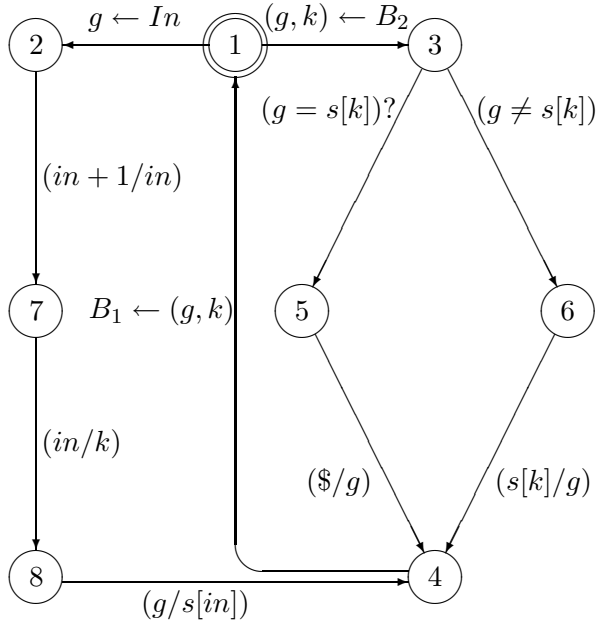

Рис. 6. Графовая модель программы *Sender*.

6.1. Композиция действий

Пусть P — некоторая программа, и α_1 и α_2 — пара ее действий.

В том случае, когда α_1 и α_2 удовлетворяют некоторым условиям, возможно определить действие, называемое **композицией** α_1 и α_2, которое заключается в последовательном выполнении α_1 и α_2.

Одно из этих условий — хотя бы одно из действий α_1, α_2 должно быть внутренним.

Обозначим

- символами φ_1 и φ_2 компоненты «условие» действий α_1 и α_2, и
- символами θ_1 и θ_2 компоненты «подстановка» действий α_1 и α_2.

Композицией α_1 и α_2 называется действие, обозначаемое знакосочетанием

$$\alpha_1\alpha_2 \tag{18}$$

и определяемое следующим образом:

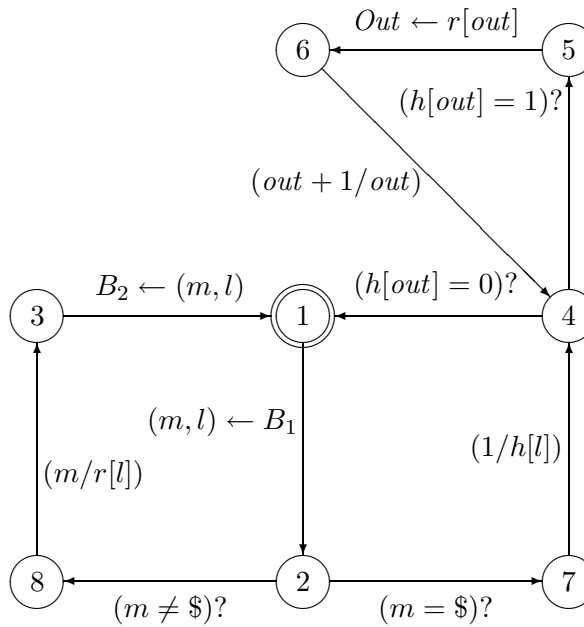

Рис. 7. Графовая модель программы *Receiver*.

1) компонента «условие» действия (18) имеет вид

$$\varphi_1 \ \wedge \ \theta_1(\varphi_2)$$

2) компонента «подстановка» имеет вид

$$\theta_1\theta_2, \tag{19}$$

где знакосочетание (19) обозначает подстановку, называемую **композицией** подстановок θ_1 и θ_2, и определяемую следующим образом:

$$\forall x \in Var(P) \qquad (\theta_1\theta_2)(x) \stackrel{\text{def}}{=} \theta_1(\theta_2(x))$$

3) если

- α_1 содержит компоненту $X \leftarrow P'$, и
- формула $\theta_1(\varphi_2)$ не зависит от переменных из X,

то (18) содержит такую же компоненту,

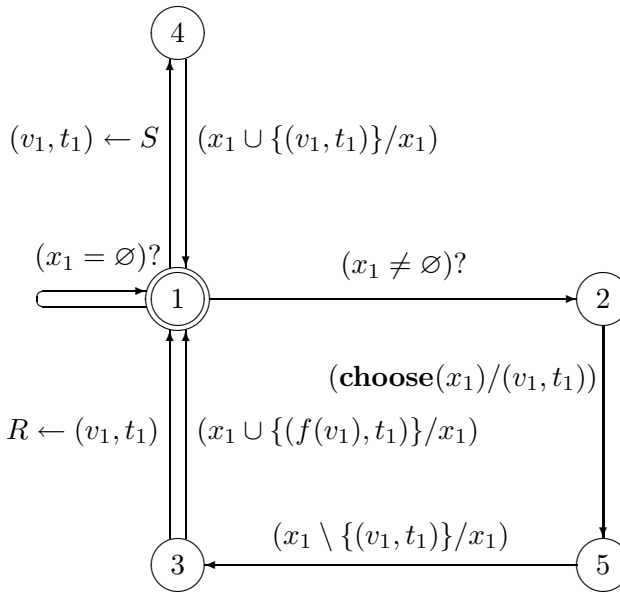

Рис. 8. Графовая модель программы $Buffer_1$.

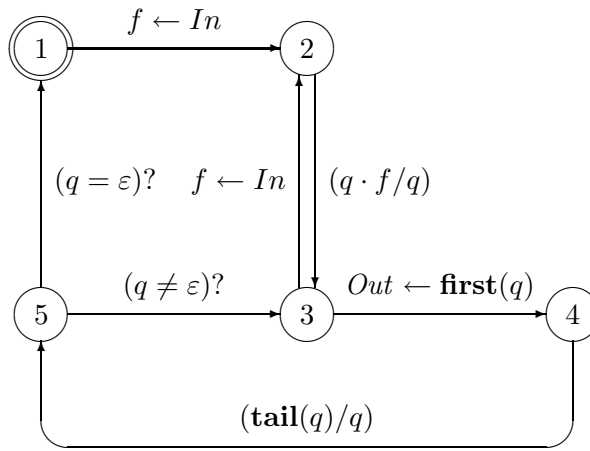

Рис. 9. Графовая модель программы *Spec*.

4) если
   - α_2 содержит компоненту $X \leftarrow P'$, и
   - для всех $y \in Var(P) \setminus X \quad \theta(y)$ не зависит от переменных из X,

   то (18) содержит такую же компоненту,
5) если α_1 содержит компоненту $P' \leftarrow E$, то (18) содержит такую же компоненту,
6) если α_2 содержит компоненту $P' \leftarrow E$, то (18) содержит компоненту
$$P' \leftarrow \theta_1(E).$$

В тех случаях, когда условия в пунктах 3 и 4 не выполнены, действие (18) не определено.

6.2. Определение редукции программ

Операция редукции программы заключается в удалении из нее одной вершины и преобразовании некоторых ребер.

Данную операцию можно применить в том случае, когда в программе P имеется ребро f_0, обладающее перечисленными ниже свойствами.

Для формулировки этих свойств мы введем следующие обозначения.

- Начало и конец ребра f_0 обозначим символами n_0 и n соответственно.
- Список всех ребер, выходящих из n, обозначим знакосочетанием
$$f_1, \ldots, f_k. \tag{20}$$
- Концы ребер из списка (20) обозначим символами
$$n_1, \ldots, n_k. \tag{21}$$
- Условия действий $\langle f_0 \rangle, \langle f_1 \rangle, \ldots, \langle f_k \rangle$ обозначим символами
$$\varphi_0, \varphi_1, \ldots, \varphi_k.$$

Необходимые условия для применения операции редукции заключаются в следующем:

1) $n_0 \neq n$,
2) $n \neq \mathit{Start}(P)$,
3) существуют композиции действий

$$\langle f_0\rangle\langle f_1\rangle, \quad \ldots, \quad \langle f_0\rangle\langle f_k\rangle, \tag{22}$$

4) если действие $\langle f_0\rangle$ — внутреннее, то для каждого ребра f, выходящего из n_0, за исключением f_0, следующая формула является тавтологией:

$$\neg(\varphi \wedge \varphi_0),$$

где φ — условие действия $\langle f\rangle$,

5) если действие $\langle f_0\rangle$ — не внутреннее, то следующие формулы являются тавтологиями:

$$\bigvee_{i=1}^{k} \varphi_i \quad \text{и} \quad \bigwedge_{1\leqslant i<j\leqslant k} \neg(\varphi_i \wedge \varphi_j).$$

Если данные условия выполнены, то к программе P можно применить **операцию редукции**, которая заключается в выполнении следующих действий:

- удаление вершины n и ребер $f_0, f_1, \ldots, f_k$,
- добавление новых ребер из n_0 в вершины из списка (21) с метками из списка (22) соответственно,
- удаление вершин, недостижимых из начальной вершины,
- удаление цикловых ребер с однокомпонентными действиями вида φ ?,
- удаление бесполезных присваиваний (после которых новые значения переменных не используются).

Операция редукции уменьшает число вершин. Может оказаться так, что к редуцированной программе опять можно применить операцию редукции. Применение данной операции несколько раз может существенно уменьшить число вершин анализируемой программы. Например, применение операции редукции к программам из системы *UBP* уменьшает их размер в среднем в 3 раза.

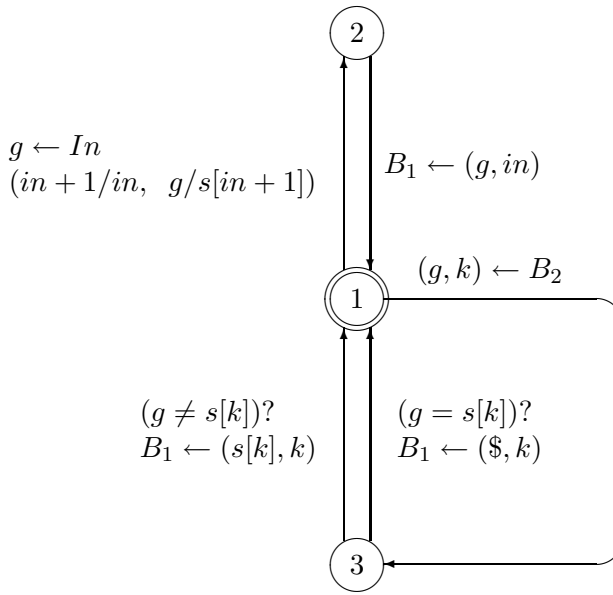

Рис. 10. *Sender′*.

6.3. Пример редукции программ

В результате применения операции редукции к программам из системы *UBP* получаются программы

$$Sender',\ \ Receiver',\ \ Buffer'_i\ \ (i \in \{1,2\}),\ \ Spec',$$

изображенные на рис. 10–13.

7. Отношения перехода

7.1. События

Событием называется знакосочетание одного из следующих видов:

- $P \leftarrow D$ (передача списка значений D программе P),

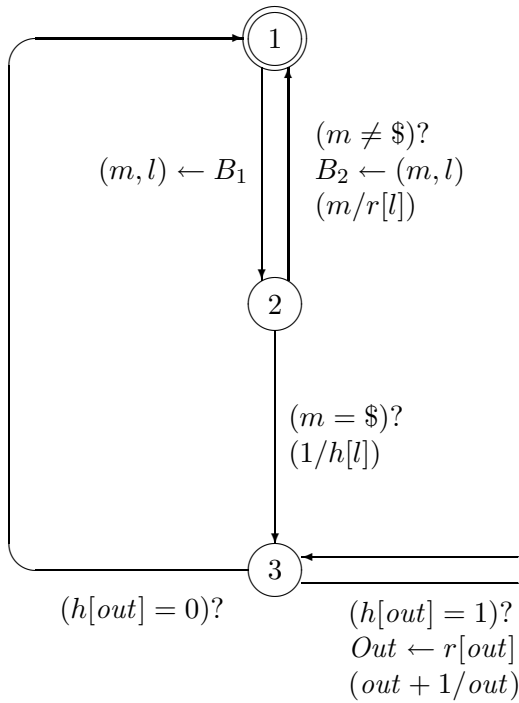

Рис. 11. *Receiver′*.

- $D \leftarrow P$ (прием списка значений D от программы P),
- τ (внутреннее событие),

где P — некоторая программа, D — некоторый список значений из $\mathcal{D}^*$, и τ — некоторый специальных символ.

Множество всех событий обозначается символом $\mathcal{M}$.

7.2. Отношение перехода на состояниях программы

Пусть P — некоторая программа.

В данном пункте мы определяем **отношение перехода** «$\rightarrow$» вида

$$\rightarrow \quad \subseteq \quad S(P) \times \mathcal{M} \times S(P).$$

Если тройка (ξ, μ, ξ') принадлежит «$\rightarrow$», то данный факт будет обозначаться знакосочетанием

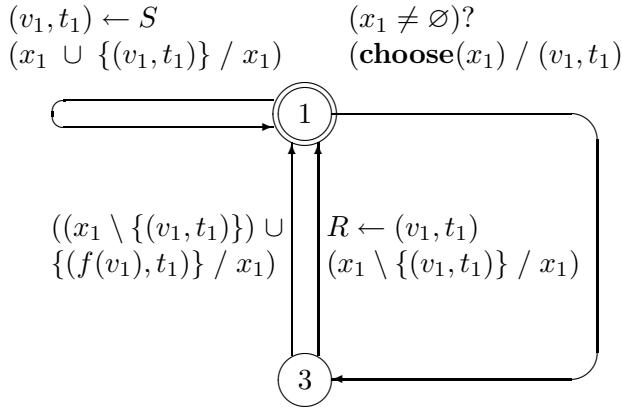

Рис. 12. *Buffer*$'_1$.

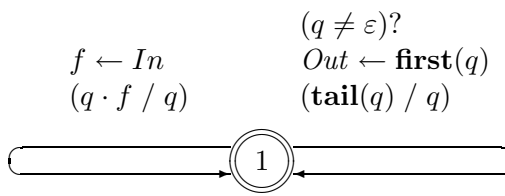

Рис. 13. *Spec*$'$.

$$\xi \xrightarrow{\ \mu\ } \xi'. \tag{23}$$

Соотношение (23) имеет место в том и только в том случае, когда существует ребро f из $F(P)$, обладающее перечисляемыми ниже свойствами. В формулировке данных свойств символы φ и θ обозначают компоненты «условие» и «подстановка» действия $\langle f \rangle$.

1) Вершины $\xi(at_P)$ и $\xi'(at_P)$ являются, соответственно, началом и концом ребра f.
2) $\xi(\varphi) = 1$.
3) Если $\langle f \rangle$ — внутреннее действие, то

$$\mu = \tau \quad \text{и} \quad \xi' = \xi\theta,$$

где символ $\xi\theta$ обозначает состояние, сопоставляющее переменной $x \in Var(P)$ значение $\xi(\theta(x))$.

4) Если $\langle f \rangle$ содержит компоненту $P' \leftarrow E$, то

$$\mu = P' \leftarrow \xi(E) \quad \text{и} \quad \xi' = \xi\theta.$$

5) Если $\langle f \rangle$ содержит компоненту $X \leftarrow P'$, то

$$\mu = D \leftarrow P' \quad \text{и} \quad \xi' = \xi\ (D/X)\ \theta$$

для некоторого $D \in \mathcal{D}_X$.

7.3. Отношение перехода на состояниях системы

Состоянием системы Σ вида (10) называется список Ξ вида

$$\Xi = (\xi_1, \ldots, \xi_n) \tag{24}$$

такой, что для каждого $i \in \{1, \ldots, n\}$ $\quad \xi_i \in S(P_i)$.

Состояние системы Σ можно рассматривать как функцию вида

$$\Xi : Var(\Sigma) \to \mathcal{D},$$

где

$$Var(\Sigma) \stackrel{\text{def}}{=} Var(P_1)\ \cup\ \ldots\ \cup\ Var(P_n).$$

Множество всевозможных состояний системы Σ обозначается символом $S(\Sigma)$.

Состояние (24) называется **начальным**, если для каждого $i \in \{1, \ldots, n\}$ имеет место равенство

$$\xi_i(Init(P_i)) = 1.$$

В данном пункте мы определяем **отношение перехода** «$\rightarrow$» вида

$$\rightarrow \quad \subseteq \quad S(\Sigma) \times \mathcal{M} \times S(\Sigma).$$

Если тройка (Ξ, μ, Ξ') принадлежит данному отношению, то этот факт будет обозначаться знакосочетанием

$$\Xi \xrightarrow{\ \mu\ } \Xi'. \tag{25}$$

Пусть Ξ и Ξ' имеют вид

$$\Xi = (\xi_1, \ldots, \xi_n), \qquad \Xi' = (\xi'_1, \ldots, \xi'_n).$$

Соотношение (25) имеет место в одном из следующих случаев.

Внешнее событие:

В данном случае μ имеет вид

$$D \leftarrow P \quad \text{или} \quad P \leftarrow D \qquad (\text{где } P \notin \Sigma),$$

и для некоторого индекса i

- $\xi_i \xrightarrow{\mu} \xi'_i$, и
- $\xi_j = \xi'_j$ для всех $j \neq i$.

Внутреннее событие:

В данном случае $\mu = \tau$, и

1) либо для некоторого индекса i
   - $\xi_i \xrightarrow{\tau} \xi'_i$, и
   - $\xi_j = \xi'_j$ для всех $j \neq i$,
2) либо для некоторой пары i, j различных индексов
   - $\xi_i \xrightarrow{P_j \leftarrow D} \xi'_i$ и $\xi_j \xrightarrow{D \leftarrow P_i} \xi'_j$ для некоторого $D \in \mathcal{D}^*$,
   - $\xi_k = \xi'_k$ для всех $k \neq i, j$.

8. Эквивалентность систем

Понятие эквивалентности систем позволяет дать точное определение соответствия системы своей спецификации. Мы будем считать, что система Σ соответствует спецификации *Spec*, если имеется эквивалентность между системой Σ и системой (или программой), представляющей спецификацию *Spec*.

8.1. Понятие эквивалентности систем

Пусть Σ — некоторая система.

На множестве $S(\Sigma)$ можно ввести структуру ориентированного размеченного графа: в том случае, когда имеет место соотношение

(25), мы будем считать, что данный граф содержит ребро из Ξ в Ξ' с меткой μ.

Ниже мы будем использовать следующие обозначения: для всех $\Xi_1, \Xi_2 \in S(\Sigma)$

- знакосочетание
$$\Xi_1 \xrightarrow{\tau^*} \Xi_2$$
означает, что либо $\Xi_1 = \Xi_2$, либо в определенном выше графе есть путь из Ξ_1 в Ξ_2, каждое ребро которого имеет метку τ.
- знакосочетание
$$\Xi_1 \xrightarrow{\mu_\tau} \Xi_2 \tag{26}$$
означает, что
  - или $\mu = \tau$ и $\Xi_1 = \Xi_2$,
  - или имеют место соотношения
$$\Xi_1 \xrightarrow{\tau^*} \Xi_1' \xrightarrow{\mu} \Xi_2' \xrightarrow{\tau^*} \Xi_2,$$
где Ξ_1', Ξ_2' — некоторые состояния из $S(\Sigma)$.

Системы Σ_1 и Σ_2 называются **эквивалентными** (данный факт обозначается знакосочетанием $\Sigma_1 \sim \Sigma_2$), если существует бинарное отношение
$$R \subseteq S(\Sigma_1) \times S(\Sigma_2)$$
(называемое **бимоделированием**), такое, что

1) для каждого начального состояния Ξ_1 системы Σ_1 существует начальное состояние Ξ_2 системы Σ_2, такое, что
$$(\Xi_1, \Xi_2) \in R \tag{27}$$
2) для каждого начального состояния Ξ_2 системы Σ_2 существует начальное состояние Ξ_1 системы Σ_1, такое, что имеет место (27)
3) для
   - каждой пары $(\Xi_1, \Xi_2) \in R$, и
   - каждого состояния $\Xi_1' \in S(\Sigma_1)$, такого, что
$$\Xi_1 \xrightarrow{\mu} \Xi_1'$$
для некоторого $\mu \in \mathcal{M}$

существует состояние $\Xi_2' \in S(\Sigma_2)$, такое, что

$$(\Xi_1', \Xi_2') \in R \quad \text{и} \quad \Xi_2 \xrightarrow{\mu_\tau} \Xi_2'$$

4) для
- каждой пары $(\Xi_1, \Xi_2) \in R$, и
- каждого состояния $\Xi_2' \in S(\Sigma_2)$, такого, что

$$\Xi_2 \xrightarrow{\mu} \Xi_2'$$

для некоторого $\mu \in \mathcal{M}$
существует состояние $\Xi_1' \in S(\Sigma_1)$, такое, что

$$(\Xi_1', \Xi_2') \in R \quad \text{и} \quad \Xi_1 \xrightarrow{\mu_\tau} \Xi_1'.$$

Аналогичным образом формулируется понятие эквивалентности программ.

Нетрудно доказать, что «$\sim$» действительно является отношением эквивалентности (то есть оно рефлексивно, симметрично и транзитивно).

8.2. Свойства отношения эквивалентности систем

Ниже мы приводим без доказательства два утверждения, относящиеся к понятию эквивалентности систем. Данные утверждения будут использоваться в пункте 9 для верификации системы *UBP*.

Утверждение 1. *Пусть программа P' получается из программы P при помощи редукции. Тогда $P \sim P'$.*

Для доказательства данного утверждения нужно построить бимоделирование R между P и P'.

Пусть P' получается из P преобразованиями, описанными в пункте 6.2. Мы будем использовать те обозначения для удаляемых ребер, которые приведены в этом пункте.

Рассмотрим два случая.

1) $\langle f_0 \rangle$ — внутреннее действие. В этом случае

$$R \stackrel{\text{def}}{=} \{(\xi, \xi) \mid \xi \in S(P)\} \cup \{(\xi, \xi') \mid f_0 : \xi \xrightarrow{\tau} \xi'\}$$

2) $\langle f_0 \rangle$ содержит компоненту, связанную с приемом или передачей сообщения (и, следовательно, все действия $\langle f_1 \rangle, \ldots, \langle f_k \rangle$ — внутренние). В этом случае

$$R \stackrel{\text{def}}{=} \{(\xi, \xi) \mid \xi \in S(P)\} \cup \{(\xi, \xi') \mid f_i : \xi \stackrel{\tau}{\to} \xi',\ i = 1, \ldots, k\}.$$

Утверждение 2. *Пусть* Σ *и* Σ' *— системы вида*

$$\Sigma = (P_1, \ldots, P_n), \quad \Sigma' = (P'_1, \ldots, P'_n)$$

и для каждого $i = 1, \ldots, n \quad P_i \sim P'_i$. *Тогда* $\Sigma \sim \Sigma'$.

9. Верификация системы *UBP*

9.1. Задача верификации

Задача верификации системы *UBP* заключается в том, чтобы доказать эквивалентность

$$(Sender, Receiver, Buffer_1, Buffer_2) \sim Spec. \tag{28}$$

Согласно утверждениям 1 и 2 из пункта 8.2, соотношение (28) эквивалентно соотношению

$$(Sender', Receiver', Buffer'_1, Buffer'_2) \sim Spec'. \tag{29}$$

Ниже мы будем обозначать символом Σ систему в левой части соотношения (29), а входящие в нее программы — символами S, R, B_1 и B_2 соответственно.

Для доказательства соотношения (29) необходимо определить бимоделирование между Σ и $Spec'$.

9.2. Неформальное обсуждение функционирования системы Σ

Функционирование системы Σ можно интерпретировать как создание и обработку пакетов.

Для каждого $j \geqslant 1$ пакет с номером j проходит несколько стадий при своем движении в системе Σ:

1) создание j-го пакета (в программе S, в момент поступления в систему j-го сообщения от программы In),
2) пересылка j-го пакета от одной программы системы Σ к другой программе системы Σ по ребрам на рис. 1,
3) модификация j-го пакета (в программах B_1, B_2 или S),
4) уничтожение j-го пакета (в программе R, в момент выдачи из системы j-го сообщения программе Out).

Отметим, что для каждого состояния Ξ системы Σ имеет место эквиваленция

$$\left(\begin{array}{l}\text{в состоянии } \Xi \\ \text{в системе имеется} \\ \text{пакет с номером } j\end{array}\right) \quad \Leftrightarrow \quad \Xi \models out \leqslant j \leqslant in. \tag{30}$$

Каждый возможный вариант функционирования системы Σ соответствует некоторому пути π

$$\pi = \qquad (\Xi_1 \xrightarrow{\mu_1} \Xi_2 \xrightarrow{\mu_2} \ldots) \tag{31}$$

в ориентированном графе с множеством вершин $S(\Sigma)$, который был определен в пункте 8.1.

Поскольку каждый оператор в программах из системы Σ не уменьшает значения переменных in и out, то, следовательно, справедливы неравенства

$$\begin{array}{l}\Xi_1(in) \leqslant \Xi_2(in) \leqslant \ldots \\ \Xi_1(out) \leqslant \Xi_2(out) \leqslant \ldots\end{array} \tag{32}$$

Следовательно,

- номер первого состояния в последовательности (31), в котором j-й пакет присутствует в какой-либо из программ системы Σ, является наименьшим из i, удовлетворяющих неравенству

$$j \leqslant \Xi_i(in);$$

- номер последнего состояния в последовательности (31), в котором j-й пакет присутствует в какой-либо из программ системы Σ, является наибольшим из k, удовлетворяющих неравенству

$$\Xi_k(out) \leqslant j.$$

Таким образом, те состояния из последовательности (31), в которых j-й пакет присутствует в какой-либо из программ системы Σ, образуют связную подпоследовательность $\pi(j)$, которая

- является пустой, в том случае, когда при данном варианте функционирования j-е сообщение вообще не поступало в систему,
- имеет вид

$$\pi(j) = (\Xi_i \xrightarrow{\mu_i} \Xi_{i+1} \xrightarrow{\mu_{i+1}} \ldots \xrightarrow{\mu_{k-1}} \Xi_k) \tag{33}$$

в том случае, когда j-е сообщение поступило в систему в момент времени i и выдается из системы программе Out в момент времени k,

- является бесконечной

$$\pi(j) = (\Xi_i \xrightarrow{\mu_i} \Xi_{i+1} \xrightarrow{\mu_{i+1}} \ldots) \tag{34}$$

в том случае, когда j-е сообщение поступило в систему в момент времени i, но никогда не выдается программе Out.

9.3. Определение бимоделирования между Σ и $Spec'$

Искомое бимоделирование R мы определяем как совокупность пар (Ξ_1, Ξ_2) из $S(\Sigma) \times S(Spec')$, удовлетворяющих условию

$$\Xi_1(u) = \Xi_2(q),$$

где

- $u \stackrel{\text{def}}{=} (s[out], \ldots, s[in])$
(список сообщений, которые поступили в систему, но еще пока не выданы)
- q — переменная программы $Spec'$.

Для доказательства того, что данное отношение действительно является бимоделированием, будут использоваться некоторые инварианты системы Σ, которые определяются в пункте 9.4.

9.4. Инварианты системы Σ

9.4.1. Понятие инварианта

Инвариантом системы называется произвольная формула с переменными из множества $Var(\Sigma)$, которая является истинной в каждом состоянии данной системы, достижимом из начального состояния. Формула φ называется **истинной** в состоянии Ξ, если

$$\Xi(\varphi) = 1. \tag{35}$$

Соотношение (35) мы будем также записывать в виде знакосочетания

$$\Xi \models \varphi.$$

Доказательство того, что некоторая формула φ является инвариантом системы, обычно состоит из доказательства следующих двух утверждений:

- (35) верно для каждого начального состояния Ξ,
- для каждой пары Ξ, Ξ' состояний, таких, что

  $$\Xi \xrightarrow{\ \mu\ } \Xi' \quad \text{для некоторого } \mu \in \mathcal{M}$$

  из (35) следует соотношение $\Xi'(\varphi) = 1$.

9.4.2. Вспомогательные выражения и формулы

В рассуждениях о системе Σ будут использоваться следующие выражения с переменными из множества $Var(\Sigma)$:

- $x_i(j) \stackrel{\text{def}}{=} x_i \cap \{(d, j) \mid d \in \mathcal{D}\} \qquad (j \geqslant 1,\ i = 1, 2)$,
  то есть $x_i(j)$ есть множество пакетов с номером j в буфере B_i,
- $f^*(d) \stackrel{\text{def}}{=} \{f^k(d) \mid k \geqslant 0\} \qquad (d \in \mathcal{D})$,
  то есть $f^*(d)$ есть множество, состоящее из d и всех возможных искажений значения d.

Отметим, что из (9) следует импликация:

$$(d' \in f^*(d)) \ \wedge \ (d \in f^*(d')) \qquad \Rightarrow \qquad d = d'. \tag{36}$$

Ниже для каждого списка $e_1, \ldots, e_n$ формул их конъюнкция $e_1 \wedge \ldots \wedge e_n$ будет обозначаться также знакосочетанием

$$\left\{ \begin{array}{l} e_1 \\ \ldots \\ e_n \end{array} \right\}.$$

Определим вспомогательные формулы, которые будут использоваться при определении инвариантов системы Σ.

- $PASSING(j) \stackrel{\text{def}}{=} \left\{ \begin{array}{l} out \leqslant j \leqslant in \\ h[j] = 0 \end{array} \right\}$
  (j-е сообщение находится в системе, но пока еще нет гарантии, что оно было передано программе R успешно).
- $S(j) \stackrel{\text{def}}{=} S_2(j) \vee S_3(j)$, где

  $$S_2(j) \stackrel{\text{def}}{=} \left\{ \begin{array}{l} at_S = 2 \\ in = j \end{array} \right\}, \qquad S_3(j) \stackrel{\text{def}}{=} \left\{ \begin{array}{l} at_S = 3 \\ k = j \end{array} \right\}$$

  (либо в программе S создан пакет с номером j, либо в программу S пришел от B_2 пакет с номером j)
- $R(j) \stackrel{\text{def}}{=} \left\{ \begin{array}{l} at_R = 2 \\ l = j \end{array} \right\}$
  (в программу R пришел пакет с номером j)
- $B_i(j) \stackrel{\text{def}}{=} (x_i(j) \neq \varnothing) \qquad (i = 1, 2)$
  (в буфере B_i имеется пакет с номером j)
- $\Sigma(j) \stackrel{\text{def}}{=} \bigvee\limits_{P \in \Sigma} P(j)$
  (пакет с номером j присутствует в одной из программ из системы Σ)
- $MUTEX(j) \stackrel{\text{def}}{=} \bigwedge\limits_{P \neq P' \in \Sigma} \neg \left\{ \begin{array}{l} P(j) \\ P'(j) \end{array} \right\}$
  (пакет с номером j не может присутствовать одновременно в двух разных программах из системы Σ).

9.4.3. Определение инвариантов

Инварианты системы Σ имеют следующий вид:

$$F_1\text{:}\ (out \leqslant j \leqslant in) \ \rightarrow \left\{ \begin{array}{l} \Sigma(j) \\ MUTEX(j) \end{array} \right\}$$

$$F_2\text{:}\ out \leqslant in + 1$$

$$F_3\text{:}\ (d, j) \in x_1 \ \rightarrow \left\{ \begin{array}{l} PASSING(j) \\ (d = \$) \rightarrow (s[j] = r[j]) \\ (d \neq \$) \rightarrow (d \in f^*(s[j])) \end{array} \right\}$$

$$F_4\text{:}\ (d, j) \in x_2 \ \rightarrow \left\{ \begin{array}{l} PASSING(j) \\ d \in f^*(r[j]) \\ r[j] \in f^*(s[j]) \end{array} \right\}$$

$$F_5\text{:}\ (h[j] = 1) \ \rightarrow \left\{ \begin{array}{l} r[j] = s[j] \\ j \leqslant in \end{array} \right\}$$

$$F_6\text{:}\ |x_i(j)| \leqslant 1 \qquad (i = 1, 2)$$

$$F_7\text{:}\ (at_S = 2) \ \rightarrow \left\{ \begin{array}{l} PASSING(in) \\ g = s[in] \end{array} \right\}$$

$$F_8\text{:}\ (at_S = 3) \ \rightarrow \left\{ \begin{array}{l} PASSING(k) \\ g \in f^*(r[k]) \\ r[k] \in f^*(s[k]) \end{array} \right\}$$

$$F_9\text{:}\ (at_R = 2) \ \rightarrow \left\{ \begin{array}{l} PASSING(l) \\ (m = \$) \rightarrow (r[l] = s[l]) \\ (m \neq \$) \rightarrow (m \in f^*(s[l])) \end{array} \right\}$$

$$F_{10}\text{:}\ (h[out] = 1) \ \rightarrow \ (at_R = 3).$$

Доказательство того, что формулы F_1–F_{10} действительно являются инвариантами системы Σ, производится непосредственной проверкой: сначала устанавливается, что формула

$$\bigwedge_{i=1}^{10} F_i$$

истинна во всех начальных состояниях системы Σ, а затем рассматриваются индуктивные переходы. В данном доказательстве используется импликация (36).

9.4.4. Интерпретация инвариантов

Инвариант F_1 можно интерпретировать как утверждение о том, что для каждого $j \geqslant 1$ и для каждого пути (31) совокупность всех состояний, входящих в подпоследовательность $\pi(j)$, можно разбить на 4 непересекающихся класса (некоторые из которых могут быть пустыми), в которых соответственно истинны формулы

$$S(j),\ R(j),\ B_1(j),\ B_2(j). \tag{37}$$

Мы будем обозначать данные классы теми же символами, что и формулы из списка (37), которые в них истинны.

В свою очередь, каждый из перечисленных классов можно поделить на несколько непересекающихся подклассов (некоторые из которых могут быть пустыми), с учетом более детальной информации о состояниях:

1) класс $S(j)$ разбивается на 3 подкласса, в которых истинны формулы

$$S_2(j) \qquad (A_1(j))$$

$$\left\{ \begin{array}{l} S_3(j) \\ g \neq s[j] \end{array} \right\} \qquad (A_2(j))$$

$$\left\{ \begin{array}{l} S_3(j) \\ g = s[j] \end{array} \right\} \qquad (A_3(j))$$

соответственно;

2) класс $R(j)$ разбивается на 3 подкласса, в которых истинны формулы

$$\left\{ \begin{array}{l} R(j) \\ m \neq s[j], \$ \end{array} \right\} \qquad (A_4(j))$$

$$\left\{ \begin{array}{l} R(j) \\ m = s[j] \end{array} \right\} \qquad (A_5(j))$$

$$\left\{ \begin{array}{l} R(j) \\ m = \$ \end{array} \right\} \qquad (A_6(j))$$

соответственно;

3) класс $B_1(j)$ (который, на основании инварианта F_6, или пуст, или состоит из одного элемента) разбивается на 3 подкласса, в которых истинны формулы

$$\left\{ \begin{array}{l} x_1(j) = \{(d, j)\} \\ d \in f^*(s[j]) \setminus \{s[j], \$\} \end{array} \right\} \qquad (A_7(j))$$

$$x_1(j) = \{(s[j], j)\} \qquad (A_8(j))$$

$$x_1(j) = \{(\$, j)\} \qquad (A_9(j))$$

соответственно;

4) класс $B_2(j)$ (который тоже или пуст, или состоит из одного элемента) разбивается на 2 подкласса, в которых истинны формулы

$$\left\{ \begin{array}{l} x_2(j) = \{(d, j)\} \\ d \in f^*(s[j]) \setminus \{s[j]\} \end{array} \right\} \qquad (A_{10}(j))$$

$$x_2(j) = \{(s[j], j)\} \qquad (A_{11}(j))$$

соответственно.

Получившиеся 11 классов состояний из подпоследовательности $\pi(j)$ мы будем обозначать теми же символами, что и определяющие их формулы, то есть $A_1(j)$–$A_{11}(j)$. Данные классы можно интерпретировать как **состояния j-го пакета.**

Состояния $A_1(j)$–$A_{11}(j)$ могут быть представлены в виде диаграммы, изображенной на рис. 14. Стрелки на этой диаграмме изображают изменение состояния и места j-го пакета в системе Σ. Состояние j-пакета может изменяться в соответствии со стрелками на диаграмме (при таких переходах система Σ выполняет только внутренние действия). Для каждой пары Ξ_i, Ξ_{i+1} соседних состояний из $\pi(j)$

- либо Ξ_i и Ξ_{i+1} находятся в одном и том же классе,
- либо классы, в которых находятся Ξ_i и Ξ_{i+1}, соединены стрелкой.

Ниже мы будем пользоваться следующей леммой.

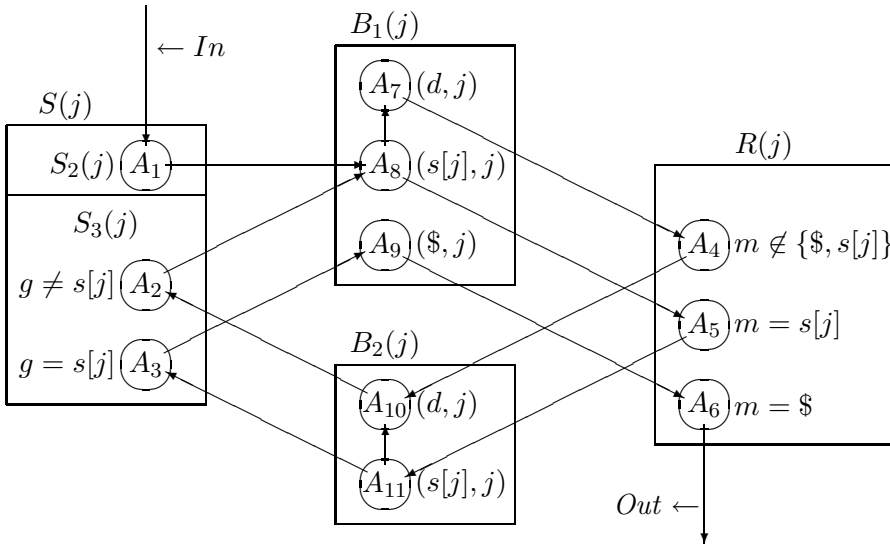

Рис. 14. Диаграмма состояний j-го пакета.

Лемма. *Пусть пара $A_p(j)$, $A_q(j)$ состояний j-го пакета такова, что в диаграмме на рис. 14 существует стрелка из $A_p(j)$ в $A_q(j)$.*

Тогда для каждого состояния Ξ системы Σ, такого, что

$$\Xi \models A_p(j)$$

существует состояние Ξ' системы Σ, такое, что

$$\Xi' \models A_q(j) \quad u \quad \Xi \xrightarrow{\tau} \Xi'.$$

9.5. Проверка условий 1 и 2

Условия 1 и 2 из определения бимоделирования верны по причине того, что

- равенство

  $$\Xi_1(u) = \varepsilon$$

  имеет место для каждого начального состояния Ξ_1 системы Σ, и

- равенство
$$\Xi_2(q) = \varepsilon$$
имеет место для каждого начального состояния Ξ_2 системы $Spec'$.

9.6. Проверка условия 3

Пусть имеются состояния Ξ_1, Ξ_2, Ξ_1' и событие μ, обладающие свойствами, изложенными в формулировке условия 3 в пункте 8.1.

Для определения искомого состояния Ξ_2' мы рассмотрим три возможных вида события μ:

$$\mu = \tau, \quad \mu = (d \leftarrow In), \quad \mu = (Out \leftarrow d). \tag{38}$$

Ниже знакосочетание $P : i \to j$ (где P — произвольная программа из Σ) означает, что

$$\Xi_1(at_P) = i \quad \text{и} \quad \Xi_1'(at_P) = j.$$

1) $\mu = \tau$.
В этом случае $\Xi_1'(u) = \Xi_1(u)$.
Определим $\Xi_2' \stackrel{\text{def}}{=} \Xi_2$.

2) $\mu = \ (d \leftarrow In)$.
Это возможно только в случае $S : 1 \to 2$. В этом случае
$$\Xi_1'(in) = \Xi_1(in) + 1, \qquad \Xi_1'(s[in]) = d.$$
Таким образом,
$$\begin{aligned}\Xi_1'(u) = \Xi_1'(s[out], \ldots, s[in]) = \\ = (\Xi_1'(s[out]), \ldots, \Xi_1'(s[in-1]), \Xi_1'(s[in])) = \\ = (\Xi_1(s[out]), \ldots, \Xi_1(s[in]), d) = \Xi_1(u) \cdot d.\end{aligned}$$
Определим
$$\Xi_2'(q) \stackrel{\text{def}}{=} \Xi_2(q) \cdot d, \quad \Xi_2'(f) \stackrel{\text{def}}{=} d.$$
Переход от Ξ_2 к Ξ_2' осуществляется по левому ребру на рис. 13.

3) $\mu = \ (Out \leftarrow d)$.
Это возможно только в случае $R : 3 \to 3$. В этом случае

$$\Xi_1 \models (h[out] = 1), \tag{39}$$

$$\Xi_1(r[out]) = d, \tag{40}$$

$$\Xi'_1(out) = \Xi_1(out) + 1. \tag{41}$$

Из (39) на основании F_5 следует соотношение

$$\Xi_1 \models \left\{ \begin{array}{l} r[out] = s[out] \\ out \leqslant in \end{array} \right\}. \tag{42}$$

Равенство $\Xi_1(u) = \Xi_2(q)$ можно записать следующим образом:

$$\Xi_1(s[out], s[out+1], \ldots, s[in]) = \Xi_2(\mathbf{first}(q)) \cdot \Xi_2(\mathbf{tail}(q)),$$

откуда следуют равенства

$$\Xi_1(s[out]) = \Xi_2(\mathbf{first}(q)) \tag{43}$$

и

$$\Xi_1(s[out+1], \ldots, s[in]) = \Xi_2(\mathbf{tail}(q)). \tag{44}$$

Из (43), (42) и (40) следует равенство

$$\Xi_2(\mathbf{first}(q)) = d. \tag{45}$$

Из (41) и (44) следует равенство

$$\Xi'_1(u) = \Xi_2(\mathbf{tail}(q)). \tag{46}$$

Определим

$$\Xi'_2(q) \stackrel{\text{def}}{=} \Xi_2(\mathbf{tail}(q)), \quad \Xi'_2(f) \stackrel{\text{def}}{=} \Xi_2(f).$$

Переход от Ξ_2 к Ξ'_2 осуществляется по правому ребру на рис. 13.

9.7. Проверка условия 4

Пусть имеются состояния Ξ_1, Ξ_2, Ξ'_2 и событие μ, обладающие свойствами, изложенными в формулировке условия 4 в пункте 8.1.

Для определения искомого состояния Ξ'_1 мы рассмотрим возможные виды события μ (см. (38)). Заметим, что случай $\mu = \tau$ невозможен ввиду структуры программы *Spec′*.

9.7.1. Случай $\mu = (d \leftarrow In)$

Данный вид μ возможен только в том случае, когда переход от Ξ_2 к Ξ_2' осуществляется по левому ребру на рис. 13. Следовательно,

$$\Xi_2'(f) = d, \qquad \Xi_2'(q) = \Xi_2(q) \cdot d \quad (= \Xi_1(u) \cdot d).$$

Нетрудно доказать, что существует состояние Ξ системы Σ, обладающее следующими свойствами:

$$\Xi_1 \xrightarrow{\tau^*} \Xi \tag{47}$$

и

$$\Xi \models (at_S = 1).$$

Из (47) следует, что $\Xi_1(u) = \Xi(u)$, откуда получаем:

$$\Xi(s[out], \ldots, s[in]) = \Xi_2(q).$$

Определим Ξ_1' следующим образом:

$$\begin{aligned}
&\Xi_1'(at_S) \stackrel{\text{def}}{=} 2,\\
&\Xi_1'(in) \stackrel{\text{def}}{=} \Xi_1(in) + 1,\\
&\Xi_1'(g) \stackrel{\text{def}}{=} d,\\
&\Xi_1'(s[in]) \stackrel{\text{def}}{=} d,
\end{aligned}$$

(значение Ξ_1' на неуказанных переменных совпадает со значением Ξ_1 на этих переменных).

Имеем:

$$\Xi \xrightarrow{\mu} \Xi_1'$$

и

$$\begin{aligned}
\Xi_1'(u) = \Xi_1'(s[out], \ldots, s[in]) &= \\
= (\Xi_1'(s[out]), \ldots, \Xi_1'(s[in-1]), \Xi_1'(s[in])) &= \\
= (\Xi_1(s[out]), \ldots, \Xi_1(s[in]), d) = \Xi_1(s[out], \ldots, s[in]) \cdot d &= \\
= \Xi_1(u) \cdot d = \Xi_2(q) \cdot d = \Xi_2'(q), &
\end{aligned}$$

то есть $(\Xi_1', \Xi_2') \in R$.

9.7.2. Случай $\mu = (\boldsymbol{Out} \leftarrow d)$

Данный вид μ возможен только в том случае, когда переход от Ξ_2 к Ξ_2' осуществляется по правому ребру на рис. 13. Следовательно,

$$\begin{aligned} &\Xi_2(q) \neq \varepsilon, \\ &\mathbf{first}(\Xi_2(q)) = d, \\ &\mathbf{tail}(\Xi_2(q)) = \Xi_2'(q). \end{aligned} \tag{48}$$

Из (48) следует, что

$$\Xi_1(s[out], \ldots, s[in]) = \Xi_1(u) = \Xi_2(q) = d \cdot \Xi_2'(q)$$

Поэтому

$$\Xi_1 \models (out \leqslant in), \tag{49}$$

$$\Xi_1(s[out]) = d, \tag{50}$$

$$\Xi_1(s[out+1], \ldots, s[in]) = \Xi_2'(q). \tag{51}$$

Рассмотрим два подслучая.

1) $\Xi_1 \models (h[out] = 1)$.
Из инвариантов (F_{10}) и (F_5) следует, что в этом случае

$$\Xi_1(at_R) = 3$$

и

$$\Xi_1 \models r[out] = s[out] \quad (= d).$$

Определим Ξ_1' следующим образом:

$$\begin{aligned} &\Xi_1'(at_R) \stackrel{\text{def}}{=} 3, \\ &\Xi_1'(out) \stackrel{\text{def}}{=} \Xi_1(out) + 1, \end{aligned}$$

(значение Ξ_1' на неуказанных переменных совпадает со значением Ξ_1 на этих переменных).
Имеем:

$$\Xi_1 \xrightarrow{\mu} \Xi_1'$$

и

$$\Xi'_1(u) = \Xi'_1(s[out], \ldots, s[in]) = (\Xi'_1(s[out]), \ldots, \Xi'_1(s[in])) = \\ = (\Xi_1(s[out+1]), \ldots, \Xi_1(s[in])) = \Xi'_2(q),$$

то есть $(\Xi'_1, \Xi'_2) \in R$.

2)

$$\Xi_1 \models h[out] = 0. \tag{52}$$

Из (49) и из рассуждений в пункте 9.4.4 следует, что существует индекс $i \in \{1, \ldots, 11\}$, такой, что

$$\Xi_1 \models A_i(out).$$

Для каждого $j \geqslant 1$ из любого состояния j-го пакета существует путь в диаграмме на рис. 14 в состояние $A_6(j)$. В частности, существует путь из $A_i(out)$ в $A_6(out)$. Поэтому из леммы в конце пункта 9.4.4 следует, что существует состояние Ξ системы Σ, такое, что

$$\Xi_1 \xrightarrow{\tau^*} \Xi \quad \text{и} \quad \Xi \models A_6(out).$$

Нетрудно доказать, что существует состояние Ξ' системы Σ со следующими свойствами:

$$\Xi \xrightarrow{\tau^*} \Xi' \quad \text{и} \quad \Xi' \models (h[out] = 1).$$

Очевидно, что имеют место соотношения

$$\Xi' \models (out \leqslant in),$$
$$\Xi'(s[out]) = d,$$
$$\Xi'(s[out+1], \ldots, s[in]) = \Xi'_2(q).$$

Оставшаяся часть доказательства для подслучая 2 повторяет доказательство для подслучая 1, с заменой Ξ_1 на Ξ'.

10. Заключение

В настоящей работе мы описали новую математическую модель программных систем и изложили новый подход к проблеме их верификации. Предложенная техника верификации является синтезом

метода индуктивных утверждений и метода, основанного на понятии бимоделирования. Мы предложили метод редукции программ, который может быть использован для понижения сложности процедуры верификации. Одним из наиболее актуальных направлений дальнейших исследований в рамках данного подхода является разработка методов автоматизированного синтеза инвариантов для верификации программных систем.

Список литературы

[1] Clarke E., Grumberg O., Peled D. Model checking. MIT Press, 2001.

[2] Floyd R. W. Assigning meanings to programs. Proc. Symp. Appl. Math., **19**. In: J. T. Schwartz (ed.) Mathematical Aspects of Computer Science. P. 19–32. American Mathematical Society. Providence, R.I., 1967.

[3] Milner R. A Calculus of Communicating Systems // Lecture Notes in Computer Science. Vol. 92. Springer, 1980.

[4] Esparza J. Decidability of model-checking for infinite-state concurrent systems // Acta Informatica. 34: 85–107. 1997.

[5] Abdulla P. A., Annichini A., Bensalem S., Bouajjani A., Habermehl P., Lakhnech Y. Verification of Infinite-State Systems by Combining Abstraction and Reachability Analysis // Lecture Notes in Computer Science. 1633. P. 146–159. Springer-Verlag, 1999.

[6] McMillan K. L. Verification of Infinite State Systems by Compositional Model Checking // Conference on Correct Hardware Design and Verification Methods. P. 219–234. 1999.

[7] Burkart O., Caucal D., Moller F., Steffen B. Verification on infinite structures. In: J. Bergstra, A. Ponse, S. Smolka (eds.) Handbook of Process Algebra. Chapter 9. P. 545–623. Elsevier Science, 2001.