

удк 519.681.1

А. М. Миронов, Д. Ю. Жуков

Методы анализа программных систем

Аннотация. В статье вводится новая математическая модель программных систем, и рассматривается проблема формального анализа их корректности. Предлагаются методы редукции анализируемых систем. Изложенные подходы иллюстрируются примером формального анализа протокола передачи сообщений через ненадёжную среду.

1. Введение

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

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

В том случае, когда число состояний исследуемой программной системы является слишком большим и формальная верификация их путём 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 бесконечным 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, [4–6]). Другие подходы основаны на понятии бимоделирования (см., например, [7]). В настоящей работе мы объединяем методы верификации

систем с бесконечным числом состояний, основанные на понятии би-моделирования, 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 $Var(P)$ её *переменных*, причём для разных программ их множества переменных не пересекаются. Каждой переменной x сопоставлено множество её возможных значений $\mathcal{D}_x \subseteq \mathcal{D}$, которые может принимать переменная x . Если X — список переменных вида

$$(x_1, \dots, x_n),$$

то символ \mathcal{D}_X обозначает декартово произведение

$$\mathcal{D}_{x_1} \times \dots \times \mathcal{D}_{x_n}.$$

Кроме того, считается заданным некоторое множество функциональных символов

$$+, \quad -, \quad \mathbf{head}, \quad \mathbf{tail}, \quad \dots$$

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

$$(1) \quad (e_1, \dots, e_n)$$

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

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

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:

$$(2) \quad Op(n) = \left(\begin{array}{c} \text{start} \\ Init(P) \end{array} \right)$$

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

присваивание:

$$(3) \quad Op(n) = \left(x := e \right)$$

где

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

проверка условия:

$$(4) \quad Op(n) = \left(b \right)$$

где b — некоторая формула.

посылка сообщения:

$$(5) \quad Op(n) = \left(P' \leftarrow E \right)$$

где

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

получение сообщения:

$$(6) \quad Op(n) = \left(X \leftarrow P' \right)$$

где

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

выбор:

$$(7) \quad Op(n) = \left(\quad \right)$$

(пустая метка).

остановка:

$$(8) \quad Op(n) = \left(\text{halt} \right)$$

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 рёбер. Из вершин с меткой (8) не выходит ни одного ребра.

2.2. Функционирование программы. Как всегда, *функционирование* программы P заключается в обходе вершин блок-схемы с выполнением операторов, сопоставленных проходимым вершинам. После выполнения оператора, соответствующего текущей вершине, происходит переход по выходящему из неё ребру к следующей вершине. На каждом шаге t функционирования ($t = 0, 1, \dots$) 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 в x значение выражения e .
- 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 путём
 - 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 путём
 - 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 рёбер, которые из неё 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 ненадёжную среду. Эта система будет обозначаться символом UBP (Unbounded Buffer Protocol).

3.1. Описание системы UBP . Задача UBP заключается в том чтобы

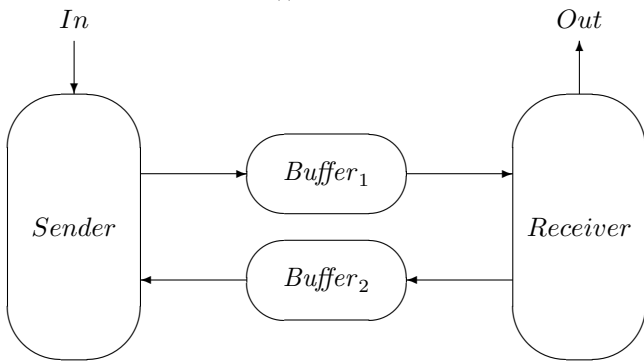
- получать сообщения от программы In , являющейся внешней по отношению к системе UBP , и
- доставлять их в том же порядке и без искажений программе Out (тоже являющейся внешней по отношению к системе UBP).

Предполагается, что в процессе доставки передаваемые сообщения проходят через среду (мы будем считать её 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 четырёх программ:

$Sender$, $Receiver$, $Buffer_1$ и $Buffer_2$

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

Рис. 1. *UBP*

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

3.2. Программа *Sender*. Программа *Sender* показана на Рис. 2. Эта программа имеет массив s , в который она записывает сообщения, поступающие к ней от программы *In*. Получив очередное сообщение, *Sender*

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

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

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

$$(\$, k)$$

где $\$$ — специальный символ, который не может быть искажён 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,

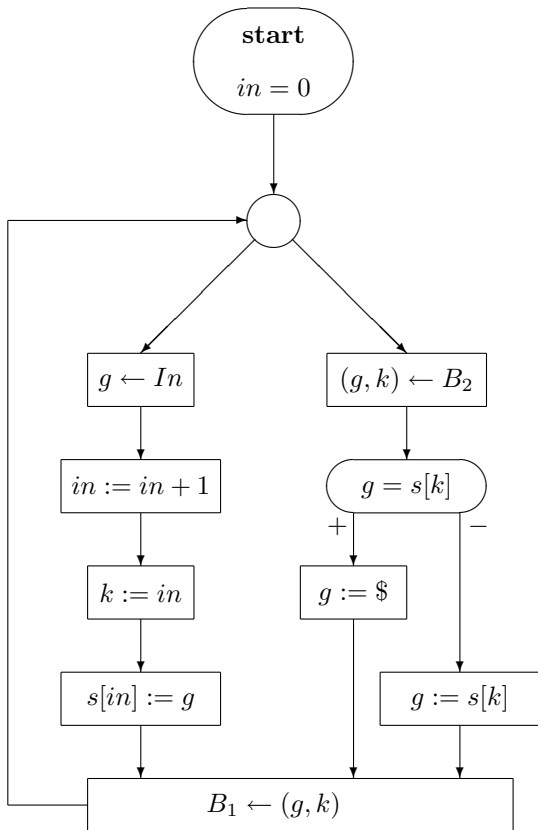


Рис. 2. *Sender*

было искажено в процессе передачи, и в этом случае *Sender* посылает программе *Buffer₁* пакет $(s[k], k)$ ещё раз.

3.3. Программа *Receiver*. Программа *Receiver* изображена на Рис. 3. *Receiver* имеет массив r , в который она записывает сообщения, поступающие к ней от программы *Buffer₁*:

- если *Receiver* получила пакет (m, l) , где m не совпадает с «\$» (это означает, что ещё пока нет уверенности в том, что сообщение с номером l дошло без искажений), то *Receiver*

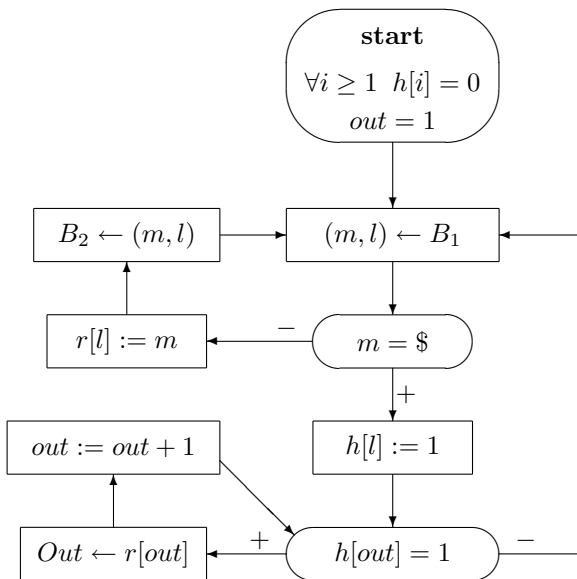


Рис. 3. Receiver

- 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 , и для каждого $i \geq 1$ значение $h[i]$ равно 1 в том случае, когда 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),

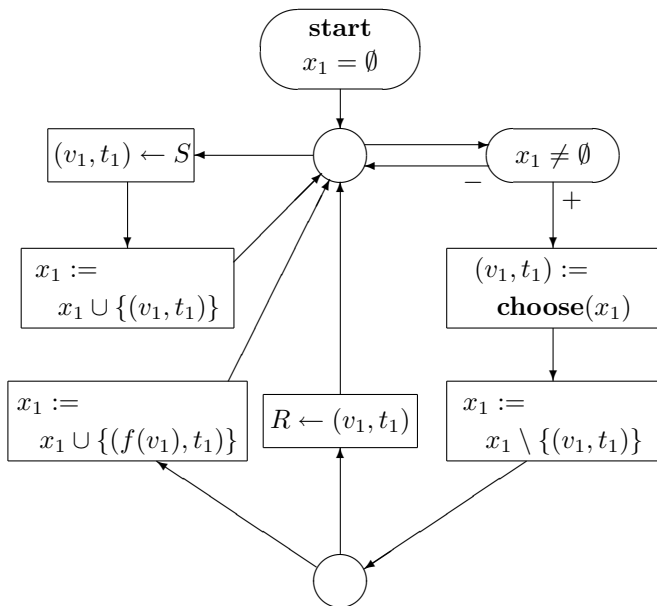


Рис. 4. $Buffer_1$

- 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*

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

такая, что $f(\$) = \$$, и

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

$Buffer_1$ имеет переменную x_1 , в которой содержатся пакеты, полученные от программы *Sender* и пока ещё не посланные программе *Receiver* (т.е. 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 $Receiver_a$, а посылает их $Sender_y$. Её переменные мы будем обозначать символами x_2, v_2, t_2 .

4. Спецификация и верификация систем

4.1. Спецификация систем. Формальное описание некоторых свойств поведения системы называют *спецификацией системы*.

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 $Spec$, изображённой на Рис. 5.

$Spec$ представляет собой неограниченный FIFO буфер, который

- не искажает полученные сообщения и
- выдаёт 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 ещё 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 ,
- знакосочетание $\mathbf{tail}(q)$ обозначает список, получаемый из q путём удаления его первого элемента.

4.2. Верификация систем. Проблема *верификации (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

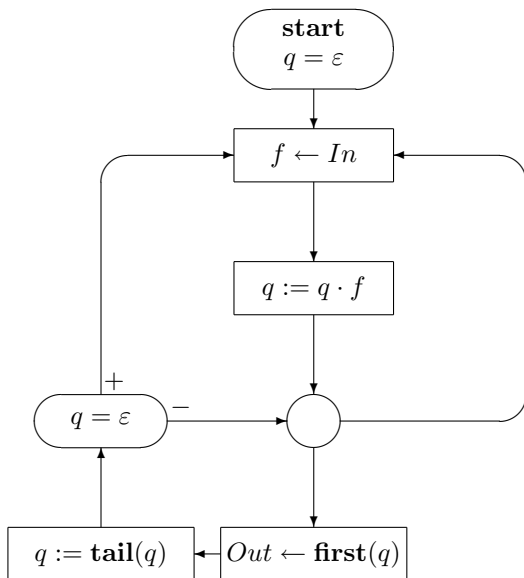


Рис. 5. *Spec*

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

Построению данной модели и разработке методов формального анализа систем в рамках данной модели посвящена оставшаяся часть работы.

5. Формальные модели систем и программ

5.1. Модель системы. Модель системы — список Σ вида

$$(10) \quad \Sigma = (P_1, \dots, P_n)$$

, 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 рёбер — $F(P)$. 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:*

$$(11) \quad \varphi ?,$$

где φ — некоторая формула из $Fm(P)$.

(2) *Посылка или приём сообщения:*

$$(12) \quad P' \leftarrow E \quad (\text{посылка сообщения})$$

или

$$(13) \quad X \leftarrow P' \quad (\text{приём сообщения})$$

где

- P' — некоторая программа,
- E — список выражений из $Tm(P)$,
- X — список различных переменных из $Var(P)$.

(3) *Подстановка*, т. е. отображение вида

$$(14) \quad \theta : Var(P) \rightarrow Tm(P)$$

Мы будем предполагать, что для каждой программы P

- множество её 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,
- 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

$$(15) \quad (e_1/x_1, \dots, e_n/x_n),$$

где

- СПИСОК

$$(16) \quad x_1, \dots, x_n$$

состоит из всех тех переменных из множества

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

для которых $x_i \neq \theta(x_i)$,

- для каждого $i = 1, \dots, n$

$$e_i = \theta(x_i)$$

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

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

Если у действия отсутствует компонента, связанная с приёмом 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) (т. е. 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 рёбер.

На каждом шаге t функционирования ($t = 0, 1, \dots$) каждой переменной x из $\text{Var}(P)$ 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

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

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 $\xi(x)$,
- применением в получившемся выражении операций, соответствующих функциональным символам, входящим в e .

Каждый шаг t функционирования графовой модели P начинается с выбора ребра f с началом в $\xi_t(at_P)$, такого, 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 путём последовательного выполнения всех его компонентов: сначала выполняются первая компонента, потом — вторая (если она есть), затем — третья (если она есть).

Компонента (11) выполняется путём 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 путём

- вычисления значения выражений e_1, \dots, e_n ,
- занесения этих значений в переменные x_1, \dots, x_n соответственно, т. е.

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

5.4. Преобразование программ в их графовые модели.

Алгоритм построения по блок-схеме программы P её 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 рёбер 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 ,

(b) на этом ребре f рисуется метка $\langle f \rangle$, определяемая следующим образом:

- (i) если $Op(n)$ имеет вид (3), то

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

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

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

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

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

- (iv) если $Op(n)$ имеет вид (5) или (6), то $\langle f \rangle$ совпадает с $Op(n)$,

- (v) 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,

(b) каждое из рёбер, выходящее из ν_2 , преобразуем в ребро с той же меткой, выходящее из ν_1 ,

(c) 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.

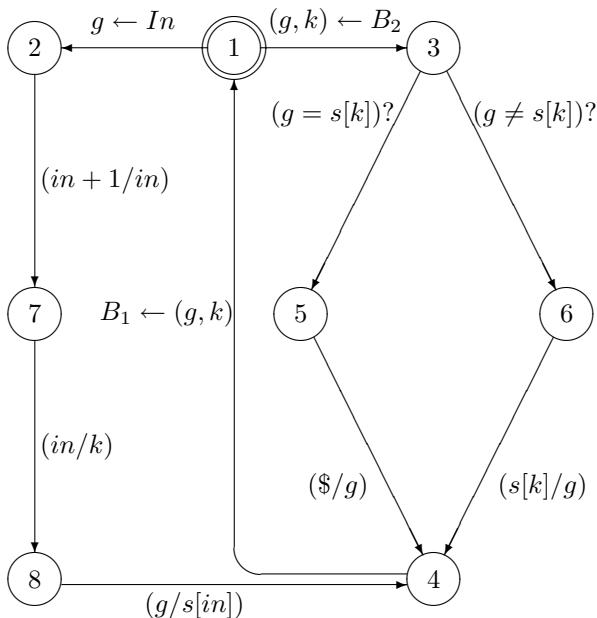


Рис. 6. Графовая модель программы *Sender*

Например, графовые модели программ из системы *UBP*, а также программы *Spec* имеют вид, изображённый на Рис. 6, Рис. 7, Рис. 8 и Рис. 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 её верифицировать.

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

6.1. Композиция действий. Пусть P — некоторая программа, и α_1 и α_2 — пара её действий.

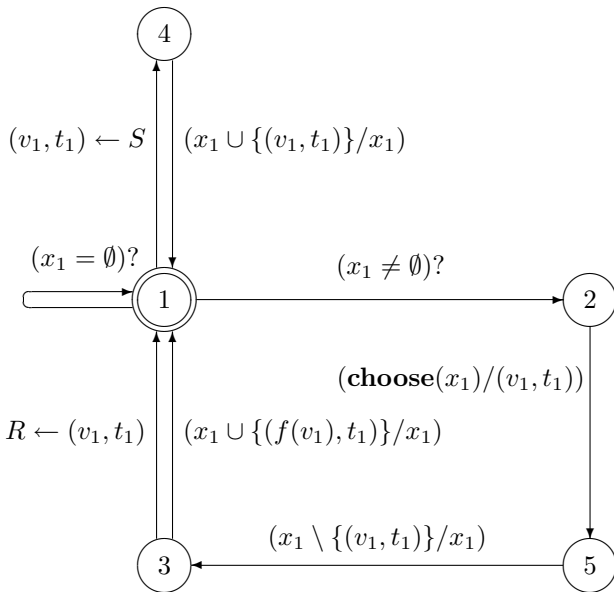


Рис. 8. Графовая модель программы *Buffer*₁

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

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

(2) компонента «подстановка» имеет вид

$$(19) \quad \theta_1\theta_2$$

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

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

(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,

(4) если

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

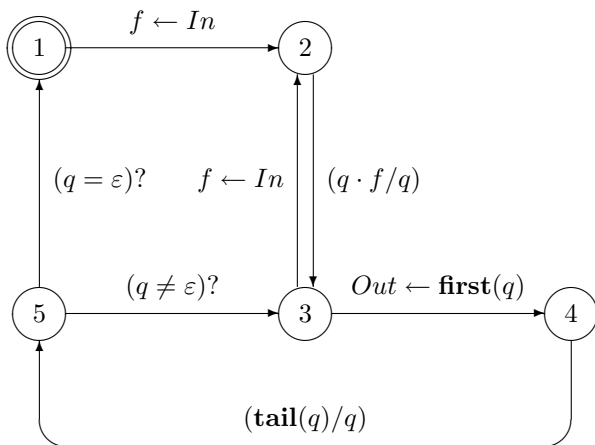


Рис. 9. Графовая модель программы *Spec*

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) данного определения не выполнены, действие (18) не определено.

6.2. Определение редукции программ. Операция редукции программы заключается в удалении из неё одной вершины и преобразовании некоторых рёбер.

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

Для формулировки этих свойств мы введём следующие обозначения.

- Начало и конец ребра f_0 обозначим символами n_0 и n , соответственно.

- Список всех рёбер, выходящих из n , обозначим знаковочетанием

$$(20) \quad f_1, \dots, f_k$$

- Концы рёбер из списка (20) обозначим символами

$$(21) \quad n_1, \dots, n_k$$

- Условия действий $\langle f_0 \rangle, \langle f_1 \rangle, \dots, \langle f_k \rangle$ обозначим символами

$$\varphi_0, \varphi_1, \dots, \varphi_k$$

Необходимые условия для применения операции редукции заключаются в следующем:

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

$$(22) \quad \langle f_0 \rangle \langle f_1 \rangle, \quad \dots, \quad \langle f_0 \rangle \langle f_k \rangle$$

- (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:

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

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

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

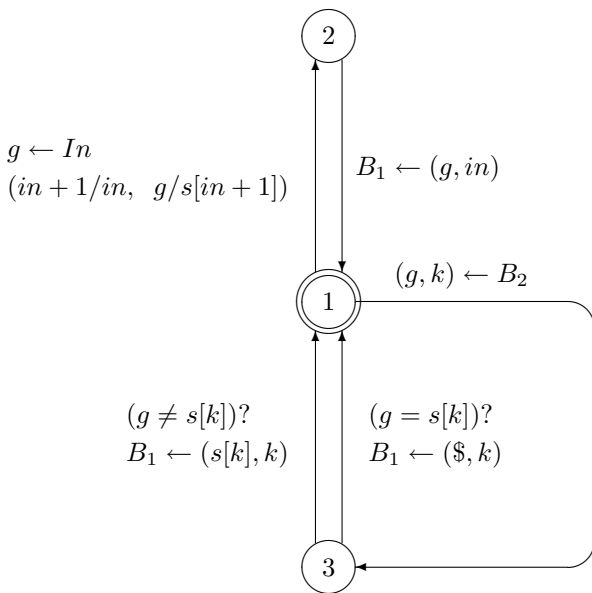


Рис. 10. $Sender'$

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.

6.3. Пример редукции программ. В результате применения операции редукции к программам из системы UBP получаются программы

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

изображённые на Рис. 10 – 13.

7. Отношения перехода

7.1. События. *Событием* называется знакосочетание одного из следующих видов:

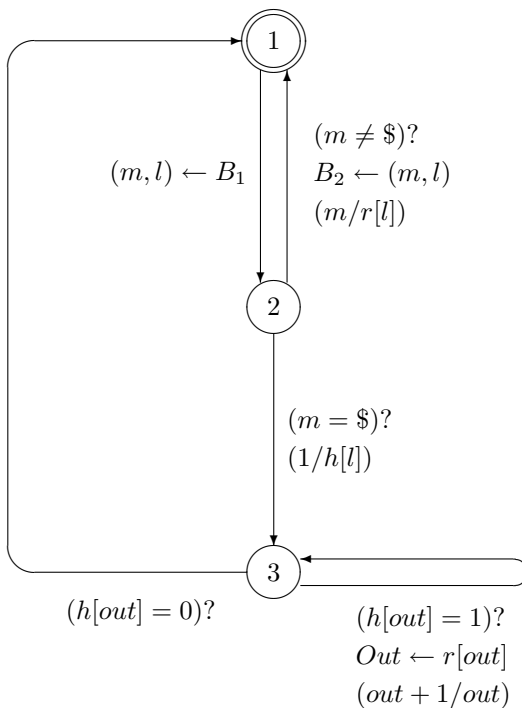


Рис. 11. *Receiver'*

- $P \leftarrow D$ (передача списка значений D программе P),
- $D \leftarrow P$ (приём 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. Для каждой программы P *отношением перехода* на её состояниях называется определяемое ниже подмножество « \rightarrow » вида

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

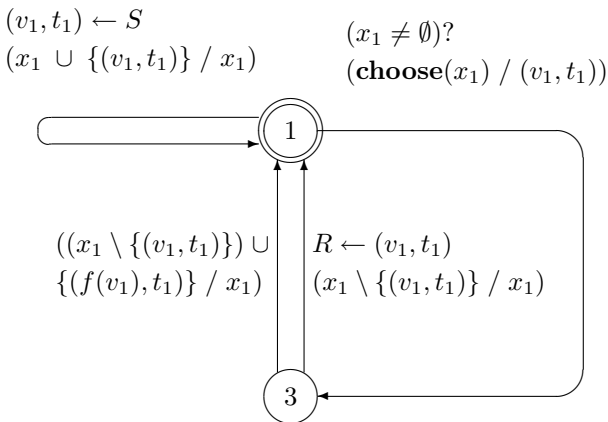


Рис. 12. $Buffer'_1$

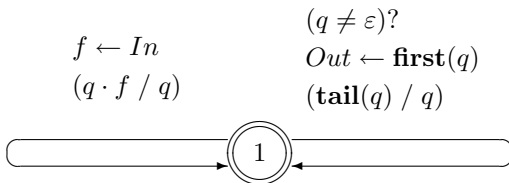


Рис. 13. $Spec'$

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

$$(23) \quad \xi \xrightarrow{\mu} \xi' .$$

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

$$(24) \quad \Xi = (\xi_1, \dots, \xi_n)$$

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

Состояние системы Σ можно рассматривать как функцию вида

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

где

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

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

Состояние (24) называется *начальным*, если

$$\forall i \in \{1, \dots, n\} \quad \xi_i(Init(P_i)) = 1.$$

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

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

Если тройка (Ξ, μ, Ξ') принадлежит данному отношению, то это будет обозначаться знаковосочетанием

$$(25) \quad \Xi \xrightarrow{\mu} \Xi'.$$

Пусть Ξ и Ξ' имеют вид

$$\Xi = (\xi_1, \dots, \xi_n), \quad \Xi' = (\xi'_1, \dots, \xi'_n)$$

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

Внешнее событие:

В данном случае μ имеет вид

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

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

- $$\xi_i \xrightarrow{P_j \leftarrow D} \xi'_i \quad \text{и} \quad \xi_j \xrightarrow{D \leftarrow P_i} \xi'_j$$

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 определённом выше графе есть путь из Ξ_1 в Ξ_2 , каждое ребро которого имеет метку τ .

- знакосочетание

(26)
$$\Xi_1 \xrightarrow{\mu_\tau} \Xi_2$$

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

$$(27) \quad (\Xi_1, \Xi_2) \in R$$

(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 (т.е. 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 8.1. Пусть при помощи редукации из программы P получается программа P' . 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 рёбер, 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 приёмом или передачей сообщения (следовательно, все действия $\langle f_1 \rangle, \dots, \langle f_k \rangle$ — внутренние). В этом случае

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

УТВЕРЖДЕНИЕ 8.2. Пусть Σ и Σ' — системы вида

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

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

9. Верификация системы UBP

9.1. Задача верификации. Задача по верификации системы UBP заключается в том, чтобы доказать эквивалентность

$$(28) \quad (Sender, Receiver, Buffer_1, Buffer_2) \sim Spec.$$

Согласно утверждениям 8.1 и 8.2 из пункта 8.2, соотношение (28) эквивалентно соотношению

$$(29) \quad (Sender', Receiver', Buffer'_1, Buffer'_2) \sim Spec'.$$

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

Для доказательства соотношения (29) необходимо определить би-моделирование между Σ и $Spec'$.

9.2. Неформальное обсуждение функционирования системы Σ . Функционирование системы Σ можно интерпретировать как создание и обработку пакетов.

Для каждого $j \geq 1$ пакет с номером j проходит несколько стадий при своём 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 рёбрам 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

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

Каждый возможный вариант функционирования системы Σ соответствует некоторому пути π

$$(31) \quad \pi = (\Xi_1 \xrightarrow{\mu_1} \Xi_2 \xrightarrow{\mu_2} \dots)$$

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

Так как каждый оператор в программах из системы Σ не уменьшает значения переменных in и out , справедливы неравенства

$$(32) \quad \begin{array}{l} \Xi_1(in) \leq \Xi_2(in) \leq \dots \\ \Xi_1(out) \leq \Xi_2(out) \leq \dots \end{array}$$

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

$$j \leq \Xi_i(in)$$

- 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

$$\Xi_k(out) \leq j$$

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

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

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

- является бесконечной

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

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

9.3. Определение бимоделирования между Σ и $Spec'$. Искомое бимоделирование R мы определяем как бинарное отношение

$$(35) \quad \{(\Xi_1, \Xi_2) \in S(\Sigma) \times S(Spec') \mid \Xi_1(u) = \Xi_2(q)\}$$

где

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

Для доказательства того, что (35) 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 Σ 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

$$(36) \quad \Xi(\varphi) = 1$$

Соотношение (36) мы будем также записывать в виде знаковосочетания

$$\Xi \models \varphi$$

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

- (36) 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 (36) 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 $Var(\Sigma)$:

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

Отметим, что из (9) следует импликация:

$$(37) \quad (d' \in f^*(d)) \wedge (d \in f^*(d')) \quad \Rightarrow \quad d = d'.$$

Ниже для каждого списка e_1, \dots, e_n формул их конъюнкция

$$e_1 \wedge \dots \wedge e_n$$

будет обозначаться также знаковосочетанием

$$\left\{ \begin{array}{c} e_1 \\ \dots \\ e_n \end{array} \right\}.$$

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

- $PASSING(j) \stackrel{\text{def}}{=} \left\{ \begin{array}{l} out \leq j \leq in \\ h[j] = 0 \end{array} \right\}$

(j -е сообщение находится в системе, но пока ещё нет гарантии, что оно было передано программе R успешно).

- $S(j) \stackrel{\text{def}}{=} S_2(j) \vee S_3(j)$, где

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

(либо в программе S создан пакет с номером j , либо в программу S пришёл 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 пришёл пакет с номером j)

- $B_i(j) \stackrel{\text{def}}{=} (x_i(j) \neq \emptyset) \quad (i = 1, 2)$

(в буфере B_i имеется пакет с номером j)

- $\Sigma(j) \stackrel{\text{def}}{=} \bigvee_{P \in \Sigma} P(j)$

(пакет с номером j присутствует в одной из программ из системы Σ)

- $MUTEX(j) \stackrel{\text{def}}{=} \bigwedge_{P \neq P' \in \Sigma} \neg \left\{ \begin{array}{l} P(j) \\ P'(j) \end{array} \right\}$

(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:

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

$$F_2: out \leq in + 1$$

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

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

$$\begin{aligned}
 F_5: (h[j] = 1) &\rightarrow \left\{ \begin{array}{l} r[j] = s[j] \\ j \leq in \end{array} \right\} \\
 F_6: |x_i(j)| \leq 1 &\quad (i = 1, 2) \\
 F_7: (at_S = 2) &\rightarrow \left\{ \begin{array}{l} PASSING(in) \\ g = s[in] \end{array} \right\} \\
 F_8: (at_S = 3) &\rightarrow \left\{ \begin{array}{l} PASSING(k) \\ g \in f^*(r[k]) \\ r[k] \in f^*(s[k]) \end{array} \right\} \\
 F_9: (at_R = 2) &\rightarrow \left\{ \begin{array}{l} PASSING(l) \\ (m = \$) \rightarrow (r[l] = s[l]) \\ (m \neq \$) \rightarrow (m \in f^*(s[l])) \end{array} \right\} \\
 F_{10}: (h[out] = 1) &\rightarrow (at_R = 3)
 \end{aligned}$$

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 (37).

9.4.4. *Интерпретация инвариантов.* Инвариант F_1 можно интерпретировать как утверждение о том, что для каждого $j \geq 1$ 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

$$(38) \quad S(j), R(j), B_1(j), B_2(j)$$

Мы будем обозначать данные классы теми же символами, что и формулы из списка (38), которые в них истинны.

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

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

$$S_2(j) \tag{A_1(j)}$$

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

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

соответственно

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

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

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

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

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

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

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

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

соответственно

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

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

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

соответственно

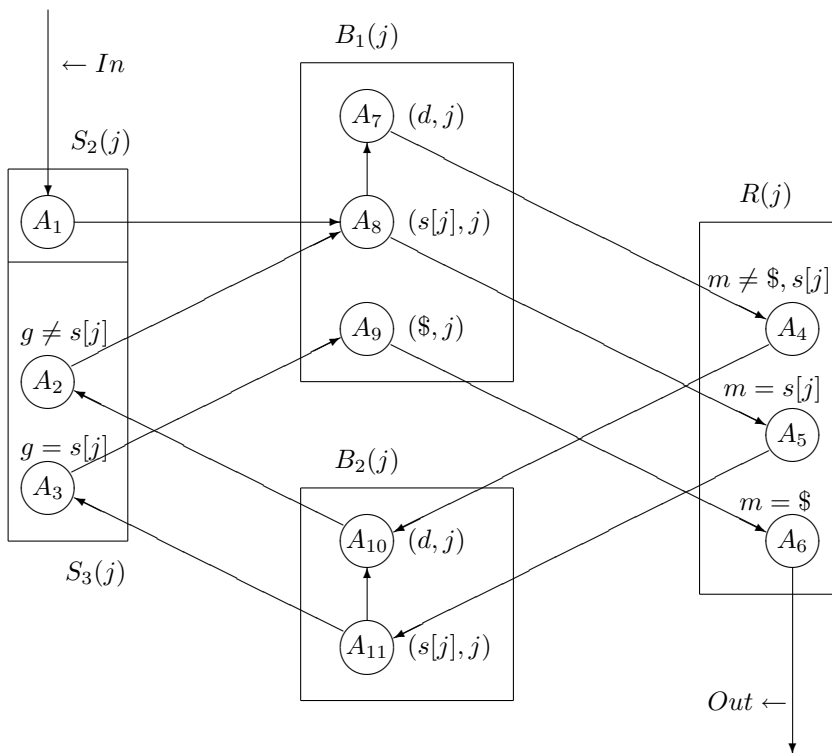


Рис. 14. Диаграмма состояний j -го пакета

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

Состояния $A_1(j)$ – $A_{11}(j)$ могут быть представлены в виде диаграммы, изображённой на Рис. 14. Стрелки на этой диаграмме изображают изменение состояния и места j -го пакета в системе Σ . Состояние j -го 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 класс, в котором находится Ξ_i , соединён стрелкой с классом, в котором находится Ξ_{i+1} .

Ниже мы будем пользоваться следующей леммой.

ЛЕММА 9.1. 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

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

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 :

$$(39) \quad \mu = \tau, \quad \mu = (d \leftarrow In), \quad \mu = (Out \leftarrow d)$$

где символ d обозначает список, состоящий из одного значения, которое мы будем обозначать тем же символом d .

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

$$\Xi_1(at_P) = i \quad \text{и} \quad \Xi'_1(at_P) = j$$

$$(1) \quad \mu = \tau.$$

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 $S : 1 \rightarrow 2$. В этом случае

$$\Xi'_1(in) = \Xi_1(in) + 1, \quad \Xi'_1(s[in]) = d$$

Таким образом,

$$\begin{aligned} \Xi'_1(u) &= \Xi'_1(s[out], \dots, s[in]) = \\ &(\Xi'_1(s[out]), \dots, \Xi'_1(s[in - 1]), \Xi'_1(s[in])) = \\ &(\Xi_1(s[out]), \dots, \Xi_1(s[in]), d) = \Xi_1(u) \cdot d \end{aligned}$$

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 происходит по левому ребру на Рис. 13.

$$(3) \mu = (Out \leftarrow d).$$

Это возможно только в случае $R : 3 \rightarrow 3$. В этом случае

$$(40) \quad \Xi_1 \models (h[out] = 1)$$

$$(41) \quad \Xi_1(r[out]) = d$$

$$(42) \quad \Xi'_1(out) = \Xi_1(out) + 1$$

Из (40) на основании F_5 следует соотношение

$$(43) \quad \Xi_1 \models \left\{ \begin{array}{l} r[out] = s[out] \\ out \leq in \end{array} \right\}$$

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

$$\begin{aligned} \Xi_1(s[out], s[out + 1], \dots, s[in]) &= \\ &= \Xi_2(\mathbf{first}(q)) \cdot \Xi_2(\mathbf{tail}(q)) \end{aligned}$$

откуда следуют равенства

$$(44) \quad \Xi_1(s[out]) = \Xi_2(\mathbf{first}(q))$$

и

$$(45) \quad \Xi_1(s[out + 1], \dots, s[in]) = \Xi_2(\mathbf{tail}(q))$$

Из (44), (43) и (41) следует равенство

$$(46) \quad \Xi_2(\mathbf{first}(q)) = d$$

Из (42) и (45) следует равенство

$$(47) \quad \Xi'_1(u) = \Xi_2(\mathbf{tail}(q))$$

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 происходит 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. (39)). 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,

$$\Xi'_2(f) = d, \quad \Xi'_2(q) = \Xi_2(q) \cdot d \quad (= \Xi_1(u) \cdot d)$$

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

$$(48) \quad \Xi_1 \xrightarrow{\tau^*} \Xi$$

и

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

Из (48) следует, что $\Xi_1(u) = \Xi(u)$, откуда получаем:

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

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

$$\begin{aligned}
 \Xi'_1(u) &= \Xi'_1(s[out], \dots, s[in]) = \\
 &= (\Xi'_1(s[out]), \dots, \Xi'_1(s[in-1]), \Xi'_1(s[in])) = \\
 &= (\Xi_1(s[out]), \dots, \Xi_1(s[in]), d) = \\
 &= \Xi_1(s[out], \dots, s[in]) \cdot d = \Xi_1(u) \cdot d = \\
 &= \Xi_2(q) \cdot d = \Xi'_2(q)
 \end{aligned}$$

т. е. $(\Xi'_1, \Xi'_2) \in R$.

9.7.2. *Случай $\mu = (Out \leftarrow d)$.* Данный вид μ возможен только в том случае, когда переход от Ξ_2 к Ξ'_2 осуществляется по правому ребру на Рис. 13. Следовательно,

$$\begin{aligned}
 (49) \quad & \Xi_2(q) \neq \varepsilon \\
 & \mathbf{first}(\Xi_2(q)) = d \\
 & \mathbf{tail}(\Xi_2(q)) = \Xi'_2(q)
 \end{aligned}$$

Из (49) следует, что

$$\Xi_1(s[out], \dots, s[in]) = \Xi_1(u) = \Xi_2(q) = d \cdot \Xi'_2(q)$$

Поэтому

$$(50) \quad \Xi_1 \models (out \leq in)$$

$$(51) \quad \Xi_1(s[out]) = d$$

$$(52) \quad \Xi_1(s[out+1], \dots, s[in]) = \Xi'_2(q)$$

Рассмотрим два подслучая.

$$(1) \quad \Xi_1 \models (h[out] = 1).$$

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

$$\begin{aligned} \Xi'_1(u) &= \Xi'_1(s[out], \dots, s[in]) = \\ &= (\Xi'_1(s[out]), \dots, \Xi'_1(s[in])) = \\ &= (\Xi_1(s[out + 1]), \dots, \Xi_1(s[in])) = \Xi'_2(q) \end{aligned}$$

т. е. $(\Xi'_1, \Xi'_2) \in R$.

(2)

$$(53) \quad \Xi_1 \models h[out] = 0$$

Из (50) и из рассуждений в пункте 9.4.4 следует, что существует индекс $i \in \{1, \dots, 11\}$, такой, что

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

Для каждого $j \geq 1$ 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

$$\begin{aligned} \Xi' &\models (out \leq in) \\ \Xi'(s[out]) &= d \\ \Xi'(s[out + 1], \dots, s[in]) &= \Xi'_2(q) \end{aligned}$$

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. ↑1
- [2] Floyd R.W. *Assigning meanings to programs* // Mathematical Aspects of Computer Science // Symp. Appl. Math./ American Mathematical Society ред. Schwartz J. T. Т. 19. — Providence, R.I., 1967, с. 19–32. ↑1
- [3] Milner R. A Calculus of Communicating Systems Lecture Notes in Computer Science. — Т. 92: Springer, 1980. ↑
- [4] Esparza J. *Decidability of model-checking for infinite-state concurrent systems* // Acta Informatica. — Т. 34, 1997, с. 85–107. ↑1
- [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. — Т. 1633: Springer-Verlag, 1999, с. 146–159. ↑
- [6] McMillan K.L. *Verification of Infinite State Systems by Compositional Model Checking* // Conference on Correct Hardware Design and Verification Methods, 1999, с. 219–234. ↑1
- [7] Burkart O., Caucal D., Moller F., Steffen B. *Verification on infinite structures* // Handbook of Process Algebra ред. Bergstra J., Ponse A., Smolka S.: Elsevier Science, 2001, с. 545–623. ↑1

A. M. Mironov, D. Y. Zhukov. *Analysis methods for program systems.* (in russian.)

ABSTRACT. We consider the problem of formal representation and analysis of program systems, and propose the methods of reduction of systems under analysis. The analysis technique is a synthesis of inductive assertion method with bisimulation equivalence method. The analysis technique is represented by a case study of verification of a network protocol with unbounded buffers.