Межфакультетский курс «Формальная семантика и верификация программного обеспечения» Лекция 1 Верификация программ. Метод Флойда

А.М.Миронов

1 Проблема верификации программ

Проблема верификации (т.е. доказательства правильности) программ занимает центральное положение в теории и практике разработки программного обеспечения. Под правильностью программ понимается их соответствие различным условиям корректности, безопасности, устойчивости в случае непредусмотренного поведения окружения, эффективности использования ресурсов времени и памяти, оптимальности реализованных в программе алгоритмов, и т.п.

Как правило, для обоснования правильности программы её тестируют, т.е анализируют её поведение на некоторых входных данных. Однако тестирование обладает очевидным недостатком: если его возможно провести не для всех допустимых входных данных, а только лишь для их небольшой части (что имеет место почти всегда), то оно не может служить гарантированным обоснованием того, что тестируемая программа обладает проверяемыми свойствами. Как отметил один из основоположников программирования Э. В. Дейкстра, «тестирование может лишь помочь выявить некоторые ошибки, но отнюдь не доказать их отсутствие».

Ошибки в программах могут быть весьма тонкими, но во многих программах наличие даже незначительных ошибок категорически недопустимо. Например, наличие ошибок в таких программах, как

- программы управления атомными электростанциями,
- программы, управляющие работой медицинских устройств,

- программы в бортовых системах управления самолетов и космических аппаратов,

- программы в системах управления секретными базами данных, системах электронной коммерции и т.п.

может привести к существенному ущербу для экономики и жизни людей.

Приведем один пример, иллюстрирующий наличие ошибок даже в очень простых программах, правильность которых на первый взгляд не вызывает никакого сомнения. Рассмотрим программу P, задача которой заключается в зачислении денег на счет клиента банка. Количество денег на счету этого клиента хранится в базе данных банка в переменной x. Когда P выполняет действия по зачислению суммы s на этот счет, она выполняет следующие действия:

- копирует в свою внутреннюю память значение переменной x,

- вычисляет новое значение, которое должна иметь переменная x, оно равно сумме текущего значения x и зачисляемой суммы s, и

- заносит в переменную x это новое значение.

Даже если программа P выполняет все свои действия правильно, это не гарантирует корректности обслуживания клиента в том случае, когда состояние его счета может изменяться несколькими такими программами. Рассмотрим ситуацию, когда состояние счета клиента изменяют две программы P_1 и P_2 описанного выше типа. Возможен следующий вариант совместного выполнения этих программ:

- сначала программа P_1 выполняет свое первое действие, оно начинается в момент времени t_1, а заключительное действие P_1 (обновление значения x) происходит в момент времени t_2, и

- в момент времени, лежащий в интервале между t_1 и t_2, программа P_2 начинает свою операцию зачисления денег на счет клиента, причем выполнение первого действия программы P_2 (копирование значения переменной x) производится до выполнения заключительного действия программы P_1.

После завершения работы обоих программ те деньги, которые зачислила на счет клиента одна из программ P_1, P_2, просто пропадут.

Ошибку описанного выше типа можно не обнаружить путем тестирования, т.к. операция зачисления денег на счет клиента выполняется

практически мгновенно, и поэтому среди тестов, которыми можно анализировать программы подобного типа, с большой вероятностью могут отсутствовать такие тесты, в которых две различные программы, обслуживающие счета клиентов, почти одновременно обращаются к одному и тому же ресурсу памяти.

Если же в результате какого-либо тестирования указанная выше ошибка обнаруживается и для её исправления конструируются специальные программные механизмы (семафоры и т.п.) с целью задания правильной дисциплины обращения программ к одному и тому же ресурсу памяти, то встает вопрос о том, насколько эти механизмы соответствуют своему предназначению (в частности, защищены ли семафоры от непредусмотренного и неавторизованного изменения их значений). Это тоже может анализироваться путем тестирования, и опять может получиться так, что среди тестов, которыми анализируется поведение указанных выше специальных программных механизмов, с большой вероятностью будут отсутствовать такие тесты, в которых проявляется некорректное поведение этих механизмов. Например, две программы P_1 и P_2, работающие с совместно используемым ресурсом R, могут использовать в качестве семафора общую переменную b, принимающую 2 значения: 1 (доступ к R открыт) и 0 (доступ к R закрыт). Если какая-либо из этих программ, например P_1, хочет работать с R, она проверяет значение b:

- если $b = 1$, то P_1 изменяет значение b на 0 (запрещая тем самым доступ к R программе P_2 на время своей работы с R) и работает с R, после чего изменяет значение b обратно на 1,

- а если $b = 0$, то P_1 ждет, пока значение b не станет равным 1.

Однако если программа P_2 тоже хочет работать с R, и она проверила значение b после того момента когда его проверила P_1, но до того момента как P_1 изменила значение b с 1 на 0, то может возникнуть нарушение дисциплины доступа к R.

Гарантированное обоснование правильности программ может быть получено только при помощи альтернативного подхода, принципиально отличного от тестирования. Данный подход называется **верификацией**. В самом общем виде верификация программы может пониматься как построение математического доказательства утверждения о том, что верифицируемая программа соответствует своему предназначению. Предназначение программы может быть выражено, например, путем описания функции, которую должна вычислять эта программа, или правил взаимодействия этой программы с другими программами, т.е. реакции,

которую эта программа должна обеспечивать в ответ на получение сигналов или сообщений от других программ.

Формальное описание предназначения программы (или некоторых свойств, которыми она должна обладать) в виде математического утверждения называется **спецификацией** этой программы. Спецификация может представлять собой формальное описание самых разнообразных свойств программы, например:

- её входные и выходные данные находятся в заданном соотношении,
- программа всегда завершает свою работу,
- во время работы программы не происходит сбоев и ненормальных ситуаций (например, деления на 0, извлечения квадратного корня из отрицательного числа, выхода индекса за границы массива, неавторизованных утечек информации и т.п.),
- программа решает свою задачу за установленное время и использует не более установленного объема памяти.

Для верификации программы P необходимо определить

- математический смысл всех конструкций, используемых в P, называемый **формальной семантикой** (или просто **семантикой**) этих конструкций, и
- спецификацию $Spec$ этой программы, выражающую то свойство программы P, которое необходимо верифицировать,

после чего можно ставить вопрос о верификации P относительно $Spec$, т.е. о построении математического доказательства утверждения о том, что P удовлетворяет $Spec$.

2 Необходимые математические понятия

2.1 Термы и связанные с ними понятия

Мы будем предполагать, что заданы следующие множества.

- Множество $Types$, элементы которого называются **типами**. Мы будем понимать типы так же, как понимаются типы данных в языках программирования. Каждому типу $\tau \in Types$ сопоставлено множество D_τ **значений** типа τ, называемое **доменом** типа τ.

  Символ $\mathcal{D}$ обозначает множество всех значений всех типов.

- Множество Var, элементы которого называются **переменными**. Каждой переменной $x \in Var$ сопоставлен тип $\tau(x) \in Types$. Каждая переменная $x \in Var$ может принимать **значения** в домене $D_{\tau(x)}$, т.е. в различные моменты времени переменная x может быть связана с различными элементами домена $D_{\tau(x)}$.

- Множество Con, элементы которого называются **константами**.

  Каждой константе $c \in Con$ сопоставлены тип $\tau(c) \in Types$ и значение из $D_{\tau(c)}$, обозначаемое тем же символом c и называемое **интерпретацией** константы c. Будем считать, что $\forall\, \tau \in Types$ любой элемент $d \in D_\tau$ является константой типа τ, которой соответствует сам элемент d.

- Множество Fun, элементы которого называются **функциональными символами (ФС)**. Каждому ФС $f \in Fun$ сопоставлены

  – **функциональный тип (ФТ)** $\tau(f)$, который представляет собой запись вида
  $$(\tau_1, \ldots, \tau_n) \to \tau, \tag{1}$$
  где $\tau_1, \ldots, \tau_n, \tau \in Types$, и

  – частичная функция вида $D_{\tau_1} \times \ldots \times D_{\tau_n} \to D_\tau$, где $\tau(f)$ имеет вид (1), данная функция обозначается тем же символом f и называется **интерпретацией** ФС f.

  (Напомним, что функция $f : A \to B$ называется **частичной**, если $\forall\, a \in A$ значение $f(a)$ м.б. не определено.)

Функциональной переменной называется переменная, тип которой является функциональным типом. Множество всех функциональных переменных обозначается записью $FVar$.

Термы строятся из переменных, констант и ФС. Множество всех термов обозначается символом Tm. Каждый терм e имеет тип $\tau(e) \in Types$, определяемый структурой терма e.

Правила построения термов имеют следующий вид:

- каждая переменная и константа является термом того типа, который сопоставлен этой переменной или константе, и

- если $e_1, \ldots, e_n \in Tm$, $f \in Fun \cup FVar$, и $\tau(f)$ имеет вид (1), где $\tau_1 = \tau(e_1)$, …, $\tau_n = \tau(e_n)$, то запись $f(e_1, \ldots, e_n)$ – терм типа τ.

Терм $e \in Tm$ называется **подтермом** терма $e' \in Tm$, если либо $e = e'$, либо $e' = f(e_1, \ldots, e_n)$, где $f \in Fun \cup FVar$, и $\exists\, i \in \{1, \ldots, n\}$: e – подтерм

терма e_i. Запись $e \subseteq e'$, где $e, e' \in Tm$, означает, что e – подтерм e'. Запись $e \subset e'$, где $e, e' \in Tm$, означает, что $e \subseteq e'$ и $e \neq e'$.

Индукцией по структуре терма $e \in Tm$ нетрудно доказать, что

если e_1 и e_2 – различные подтермы терма e, то либо $e_1 \subset e_2$, либо $e_2 \subset e_1$, либо e_1 и e_2 не имеют общих компонентов. (2)

Запись $x \in e$, где $x \in Var, e \in Tm$, означает, что x входит в e.

Будем использовать следующие обозначения и соглашения:

- $\forall\, e \in Tm\ Var(e)$ обозначает множество $\{x \in Var \mid x \in e\}$,
- $\forall\, e_1, \ldots, e_n \in Tm\ Var(e_1, \ldots, e_n) = Var(e_1) \cup \ldots \cup Var(e_n)$,
- $\forall\, X \subseteq Var\ Tm(X)$ обозначает множество $\{e \in Tm \mid Var(e) \subseteq X\}$,
- $FVar(e)$ обозначает множество $Var(e) \cap FVar$,
- $\forall\, \tau \in Types$ запись Tm_τ обозначает множество $\{e \in Tm \mid \tau(e) = \tau\}$, $\forall\, E \subseteq Tm$ запись E_τ обозначает множество $E \cap Tm_\tau$,
- в терме вида $f(e)$ скобки слева и справа от e могут опускаться, если $\tau(f)$ имеет вид $(\tau) \to \tau'$,
- для каждой рассматриваемой функции $f : E \to E'$, где $E, E' \subseteq Tm$, будем предполагать, что $\forall\, e \in E\ \ \tau(e) = \tau(f(e))$.

Терм $e \in Tm$ **замкнут**, если $Var(e) = \emptyset$. Каждому замкнутому терму e соответствует объект $eval(e)$, называемый **значением** данного терма, и либо является элементом $D_{\tau(e)}$, либо не определён. Если e – константа, то $eval(e)$ – интерпретация этой константы, и если e имеет вид $f(e_1, \ldots, e_n)$, то $eval(e)$ определён, только если определено значение функции f на кортеже $(eval(e_1), \ldots, eval(e_n))$, и в этом случае $eval(e)$ равен этому значению. Для каждого замкнутого терма e будем обозначать объект $eval(e)$ той же записью, что и сам терм (т.е. e).

2.2 Примеры типов и функциональных символов

2.2.1 Арифметические типы и функциональные символы

Con содержит типы **N** и **I**, значениями которых являются натуральные $(0, 1, \ldots)$ и целые числа соответственно.

Fun содержит ФС $+, -, \cdot, div, mod$ типа $(\mathbf{I}, \mathbf{I}) \to \mathbf{I}$, и

- функции $+$, $-$, и $\cdot$ представляют собой соответствующие арифметические операции,

- div – частичная функция (определённая, только когда второй аргумент отличен от 0), mod – частичная функция (определённая, только когда второй аргумент больше 0), div и mod вычисляют частное и остаток соответственно от деления первого аргумента на второй.

Термы $+(e_1, e_2)$, $-(e_1, e_2)$, $\cdot(e_1, e_2)$, $div(e_1, e_2)$, $mod(e_1, e_2)$ будут записываться в виде $e_1 + e_2$, $e_1 - e_2$, $e_1 e_2$, e_1/e_2 и $e_1 \% e_2$ соответственно.

2.2.2 Логические типы и функциональные символы

$Types$ содержит тип $\mathbf{B}$, и $D_{\mathbf{B}} = \{0, 1\}$. Термы типа $\mathbf{B}$ называются **формулами**. Множество всех формул обозначается Fm. $\forall X \subseteq Var$ запись $Fm(X)$ обозначает множество $Tm(X) \cap Fm$. При построении формул могут использоваться обычные булевы ФС ($\neg$, $\wedge$, $\vee$, $\to$ и т.д.), которым соответствуют функции отрицания, конъюнкции, дизъюнкции и т.д. Символ 1 обозначает тождественно истинную формулу, а символ 0 – тождественно ложную формулу. Формулы вида $\wedge(e_1, e_2)$, $\vee(e_1, e_2)$ и т.п. мы будем записывать в более привычном виде $e_1 \wedge e_2$, $e_1 \vee e_2$ и т.д. Формулы вида $e_1 \wedge \ldots \wedge e_n$ и $e_1 \vee \ldots \vee e_n$ могут также записываться в виде $\left\{ \begin{smallmatrix} e_1 \\ \vdots \\ e_n \end{smallmatrix} \right\}$ и $\left[\begin{smallmatrix} e_1 \\ \vdots \\ e_n \end{smallmatrix} \right]$ соответственно, а также в виде $\bigwedge_{i \in \{1,\ldots,n\}} e_i$ и $\bigvee_{i \in \{1,\ldots,n\}} e_i$ соответственно, и также в виде $\{e_1, \ldots, e_n\}$ и $[e_1, \ldots, e_n]$ соответственно. Формулы вида $\neg e$ могут обозначаться $\bar{e}$.

Разные ФС могут иметь одинаковое обозначение. Ниже мы приводим примеры таких ФС. Для каждого типа τ

- Fun содержит ФС eq типа $(\tau, \tau) \to \mathbf{B}$, которому соответствует функция, отображающая пару $(d_1, d_2) \in D_\tau \times D_\tau$ в элемент 1, если $d_1 = d_2$, и 0, если $d_1 \neq d_2$;

- если на D_τ задано отношение частичного порядка, то Fun содержит ФС $<$, $\leq$, $>$, $\geq$ типа $(\tau, \tau) \to \mathbf{B}$, каждому из этих ФС соответствует функция, отображающая каждую пару $(d_1, d_2) \in D_\tau \times D_\tau$ в элемент

  - 1, если $d_1 < d_2$, $d_1 \leq d_2$, $d_1 > d_2$, $d_1 \geq d_2$ соответственно, и
  - 0, если соответствующее соотношение неверно;

- Fun содержит ФС if_then_else типа $(\mathbf{B}, \tau, \tau) \to \tau$, соответствующая функция отображает тройку вида $(1, d_1, d_2)$ в d_1 и тройку вида $(0, d_1, d_2)$ в d_2.

Термы $eq(e_1, e_2)$, $<(e_1, e_2)$ и т.д. будут записываться в виде $e_1 = e_2$, $e_1 < e_2$ и т.д. соответственно. Термы $if\_then\_else(e, e_1, e_2)$ будут записываться в виде **if** e **then** e_1 **else** e_2, или $e?e_1 : e_2$.

2.2.3 Строковые типы и функциональные символы

$Types$ содержит типы **L** и **S**, значения которых называются **символами** и **символьными строками** (или просто **строками**) соответственно. Каждая строка представляет собой последовательность символов $a_1 \dots a_n$, где $n \geq 0$. При $n = 0$ эта последовательность пустая и обозначается ε.

Fun содержит

- ФС *head* и *tail* типа $\mathbf{S} \to \mathbf{L}$ и $\mathbf{S} \to \mathbf{S}$ соответственно, которым соответствуют частичные функции, определенные только для непустых строк, данные функции отображают строку $a_1 \dots a_n$ в символ a_1 и строку $a_2 \dots a_n$ соответственно (называемые **головой** и **хвостом** строки $a_1 \dots a_n$ соответственно);
- ФС *conc* типа $(\mathbf{L}, \mathbf{S}) \to \mathbf{S}$, которому соответствует функция, отображающая пару $(a, a_1 \dots a_n)$ в строку $aa_1 \dots a_n$.

Термы $conc(e, e')$, $head(e)$, $tail(e)$ будем записывать в сокращенном виде ee', e_h и e_t соответственно.

2.2.4 Кортежные типы и функциональные символы

Для каждого списка типов $\tau_1, \dots, \tau_n$ (некоторые компоненты этого списка могут совпадать)

- $Types$ содержит тип, обозначаемый записью $(\tau_1, \dots, \tau_n)$, и

$$D_{(\tau_1,\dots,\tau_n)} = D_{\tau_1} \times \dots \times D_{\tau_n},$$

- Fun содержит ФС *tuple* типа $(\tau_1, \dots, \tau_n) \to (\tau_1, \dots, \tau_n)$, ему соответствует тождественная функция, термы вида $tuple(e_1, \dots, e_n)$ будут обозначаться записью $(e_1, \dots, e_n)$.

2.3 Подстановки

Подстановкой называется функция $\theta : Var \to Tm$. Будем говорить, что подстановка θ заменяет переменную $x \in Var$ на терм $\theta(x)$.

Будем использовать следующие обозначения:

- множество всех подстановок обозначается символом Θ;
- $\forall\, \theta \in \Theta$ запись $Var(\theta)$ обозначает множество $\{x \in Var \mid \theta(x) \neq x\}$;
- $\forall\, X \subseteq Var \quad \Theta(X) = \{\theta \in \Theta \mid Var(\theta) \subseteq X\}$;
- подстановка $\theta \in \Theta$ может обозначаться записями

$$x \mapsto \theta(x) \quad \text{или} \quad (\theta(x_1)/x_1, \ldots, \theta(x_n)/x_n), \tag{3}$$

  вторая запись в (3) используется, когда $Var(\theta) = \{x_1, \ldots, x_n\}$;
- $\forall\, \theta \in \Theta,\ \forall\, e \in Tm$ запись e^θ обозначает терм, получаемый из e заменой $\forall\, x \in Var(e)$ каждого вхождения x в e на терм $\theta(x)$;
- $\forall\, \theta, \theta' \in \Theta$ запись $\theta\theta'$ обозначает подстановку $x \mapsto (x^\theta)^{\theta'}$.

Подстановка θ **замкнута**, если $\forall\, x \in Var(\theta)\ Var(x^\theta) = \emptyset$. Множество всех замкнутых подстановок из $\Theta(X)$ обозначается $X^\bullet$.

Пусть заданы терм $e \in Tm$ и список $\vec{x} = (x_1, \ldots, x_n)$ различных переменных, причём $Var(e) \subseteq \{x_1, \ldots, x_n\}$. Будем использовать обозначения:

- $D_{\tau(\vec{x})}$ обозначает множество $D_{\tau(x_1)} \times \ldots \times D_{\tau(x_n)}$;
- $e(\vec{x})$ обозначает функцию вида $D_{\tau(\vec{x})} \to D_{\tau(e)}$, такую, что

$$\forall\, \vec{d} = (d_1, \ldots, d_n) \in D_{\tau(\vec{x})} \quad e(\vec{x}) : \vec{d} \mapsto e^{(d_1/x_1, \ldots, d_n/x_n)}. \tag{4}$$

2.4 Массивы

В программах могут использоваться структуры данных, называемые **массивами**. Массивы обозначаются записями вида $a_{m..n}$, где a – имя массива, m и n – термы типа **I**, обозначающие нижнюю и верхнюю границы массива соответственно. Массив $a_{m..n}$ может обозначаться более коротко путем указания лишь его имени, без указания нижней и верхней границ. Если значения m, n нижней и верхней границ массива a таковы, что $m \leq n$, то массив a непуст, в противном случае он является пустым. Для каждого массива $a_{m..n}$ и каждого $i \in \{m, \ldots, n\}$ определен объект, обозначаемый записью a_i и называемый **компонентой** массива a с индексом i. Компоненты массива a рассматриваются как переменные одинакового типа. Если a и b – массивы вида $a_{m..n}$ и $b_{m..n}$, то $b = perm(a)$ означает, что b – перестановка a, т.е. существует биекция f на $\{m, \ldots, n\}$, такая, что $\forall\, i \in \{m, \ldots, n\}\ \ b_i = a_{f(i)}$.

Пусть задан массив $a_{m..n}$ и $i, j, i', j' \in \{m, \ldots, n\}$. Будем обозначать записями $ord(a_{i..j})$, $a_{i..j} \leq a_{i'..j'}$, $a_{i..j} \leq a_{i'}$ формулы соответственно:

$$\bigwedge_{i \leq k < j}(a_k \leq a_{k+1}), \quad \bigwedge_{\substack{i \leq k \leq j \\ i' \leq k' \leq j'}} (a_k \leq a_{k'}), \quad \bigwedge_{i \leq k \leq j}(a_k \leq a_{i'}).$$

(Напомним, что если множество конъюнктивных членов пусто, то их конъюнкция равна 1.)

2.5 Истинностные значения утверждений

Будем использовать следующее обозначение: если A – произвольное утверждение (выраженное на естественном или формальном языке), то запись $[\![A]\!]$ обозначает значение 1 если A истинно и 0 если A ложно.

3 Программы, представленные в виде блок-схем

В этой части рассматривается задача верификации последовательных и распределенных программ, представленных в виде блок-схем, в операторной форме и в виде процессов. В качестве метода их верификации используется один из наиболее широко распространённых методов верификации программ, известный под названием **метод инвариантов**.

Одним из языков описания последовательных нерекурсивных программ является язык блок-схем. На данном языке программа представляется в графовой форме. Графовая форма представления программ в последнее время завоевывает все бóльшую популярность по причине того, что такая форма представления программ облегчает их понимание и упрощает их анализ.

Блок-схема (БС) представляет собой конечный ориентированный граф, каждая вершина которого имеет один из следующих видов:

- **начальная** вершина, она обозначается символом $\odot$, в каждой БС имеется только одна начальная вершина, из неё выходит одно ребро, и в неё не входит ни одного ребра;

- **присваивание**, вершина такого вида обозначается записью вида

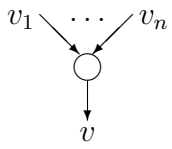
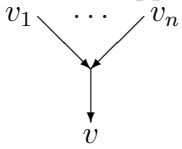
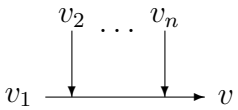
  $$\boxed{x := e},$$

  где $x \in Var$, $e \in Tm$, $\tau(x) = \tau(e)$, из каждого присваивания выходит только одно ребро;

- **условный переход**, вершина такого вида обозначается записью вида $\textcircled{\beta}$, где $\beta \in Fm$, из каждого условного перехода выходит два ребра с метками 1 и 0 соответственно;

- **пустой оператор**, такая вершина обозначается символом $\bigcirc$, из неё выходит только одно ребро;

- **терминальная** вершина, она обозначается символом $\bigotimes$, из каждой терминальной вершины не выходит ни одного ребра.

Пустой оператор в БС, как правило, не изображают, т.е. фрагмент БС вида  может изображаться в виде  , или  , или другим аналогичным образом.

В БС могут использоваться следующие обозначения:

- последовательность вершин вида $\boxed{x_1 := e_1} \to \cdots \to \boxed{x_n := e_n}$, где в каждую вершину (кроме, возможно, первой) входит только одно ребро, может обозначаться прямоугольником $\boxed{\begin{array}{c} x_1 := e_1 \\ \cdots \\ x_n := e_n \end{array}}$;

- запись
$$\boxed{(x_1, \ldots, x_n) := (e_1, \ldots, e_n)} \tag{5}$$
обозначает последовательность присваиваний вида
$$\boxed{y_1 := e_1} \to \cdots \to \boxed{y_n := e_n} \to \boxed{x_1 := y_1} \to \cdots \to \boxed{x_n := y_n}\,, \tag{6}$$
где переменные $y_1, \ldots, y_n$ – новые, т.е. если какая-либо БС содержит фрагмент вида (5), то он рассматривается как последовательность (6), где переменные $y_1, \ldots, y_n$ присутствуют только в присваиваниях из (6) и не присутствуют в этой БС вне (6);

- запись вида $\boxed{x \rightleftharpoons y}$, где $x, y \in Var$, имеет тот же смысл, что и запись $\boxed{(x, y) := (y, x)}$.

С каждой БС P связана некоторая формула $Pre(P)$, называемая **предусловием** БС P.

Если P – БС, то $V(P)$ обозначает совокупность всех вершин P и $Var(P)$ обозначает множество всех переменных, входящих в P.

4 Выполнение блок-схемы

Выполнение БС P – это обход её вершин, начиная с начальной вершины (т.е. последовательность переходов по рёбрам от одной вершины БС P к другой), с выполнением действий, сопоставленных проходимым вершинам. После выполнения действия, соответствующего текущей вершине, происходит переход по выходящему из неё ребру к следующей вершине. На каждом шаге $i \geq 0$ выполнения БС P определена подстановка $\theta_i \in Var(P)^\bullet$. Подстановка θ_0 должна удовлетворять предусловию. Действия в вершинах выполняются в зависимости от вида вершин:

- $\odot$ или $\bigcirc$: никаких действий не происходит;
- $\boxed{x := e}$: x становится связанной со значением терма e;
- $\left(\beta\right)$: для перехода к следующей вершине выбирается ребро с меткой, равной значению формулы β;
- $\otimes$: выполнение БС завершается.

Формальное описание выполнения БС P имеет следующий вид: это последовательность шагов с номерами $0, 1, \ldots$, с каждым шагом $i \geq 0$ выполнения P связаны вершина $v_i \in V(P)$ и подстановка $\theta_i \in Var(P)^\bullet$ (называемые **текущей вершиной** и **текущей подстановкой** на шаге i), причем $v_0 = \odot$, $Pre(P)^{\theta_0} = 1$ и для каждого $i \geq 0$ выполнены следующие условия:

- если v_i – начальная вершина или пустой оператор, то v_{i+1} – конец ребра, выходящего из v_i; $\theta_{i+1} = \theta_i$,
- если $v_i = \boxed{x := e}$, то v_{i+1} – конец ребра, выходящего из v_i, и

$$\theta_{i+1}(x) \stackrel{\text{def}}{=} e^{\theta_i}, \quad \forall\, y \in Var(P) \setminus \{x\} \quad \theta_{i+1}(y) \stackrel{\text{def}}{=} \theta_i(y)$$

(данное действие заключается в обновлении значения переменной x: после исполнения этого действия значение x становится равным

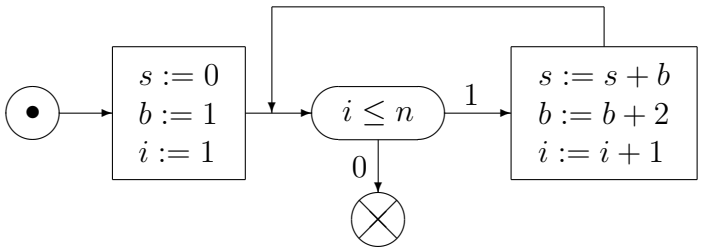
значению терма e на текущих значениях переменных БС P, значения остальных переменных не изменяются);

- если $v_i =$ $\boxed{\beta}$, то
  - если $\beta^{\theta_i} = 1$ или 0, то v_{i+1} – конец ребра, выходящего из v_i, и имеющего метку 1 или 0 соответственно, и $\theta_{i+1} = \theta_i$;
  - если β^{θ_i} не определено, то выполнение БС завершается;
- если $v_i = \bigotimes$, то выполнение БС завершается.

Выполнение действий, соответствующих фрагменту $\boxed{x \rightleftharpoons y}$, можно рассматривать как перемену местами содержимого переменных x и y.

5 Примеры блок-схем

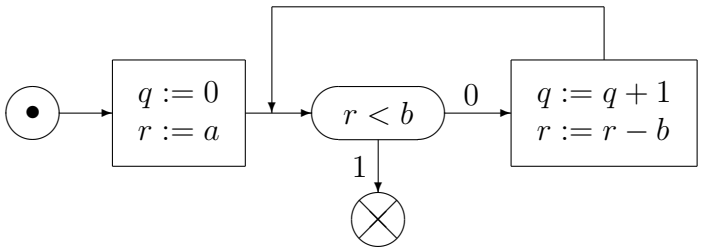
1. БС, вычисляющая сумму $1+3+5\ldots+(2n-1)$, где $n \geq 1$ – входное значение, результат должен быть занесён в переменную s.



(7)

Все переменные в данной БС имеют тип **I**.

2. БС, вычисляющая частное и остаток от целочисленного деления a на b, где a на b – положительные целые числа. В результате выполнения данной программы в переменные q и r должны быть занесены частное и остаток соответственно от деления a на b, т.е. должны быть выполнены соотношения $a = bq + r$ и $0 \leq r < b$.

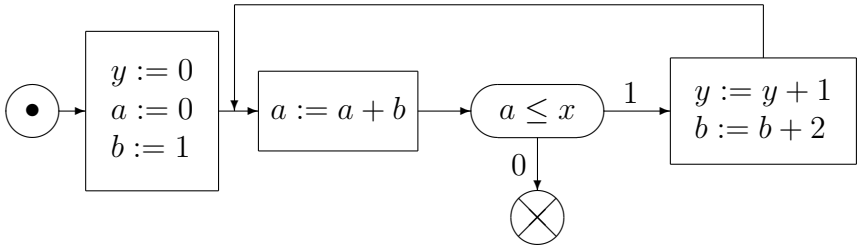


(8)

Все переменные в данной БС имеют тип **I**.

Алгоритм, реализованный в данной БС, заключается в вычитании b из a до тех пор, пока результат не станет меньше b. Число этих вычитаний равно частному, а результат вычитаний – остатку от целочисленного деления a на b.

3. БС, вычисляющая целую часть числа $\sqrt{x}$, где x – неотрицательное целое число. Результат должен быть занесён в переменную y.

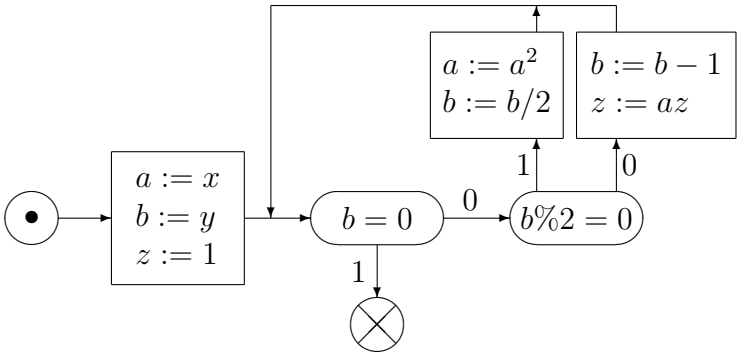


(9)

Все переменные в данной БС имеют тип **I**.

Алгоритм, реализованный в данной БС, заключается в использовании тождества $1+3+5+\ldots+(2n-1) = n^2$. Переменная b принимает последовательно значения $1, 3, 5, \ldots$, эти значения суммируются, и результат заносится в переменную a, а число слагаемых в этой сумме заносится в переменную y. Когда значение a станет превосходить x, из приведенного выше тождества следует, что y содержит требуемое значение $\lfloor\sqrt{x}\,\rfloor$.

4. БС, реализующая быстрый алгоритм возведения в степень. В результате работы данной БС вычисляется значение x^y, где $y \geq 0$. Результат должен быть занесен в переменную z.



(10)

Все переменные в данной БС имеют тип **I**. Исходное значение y предполагается неотрицательным. Вычисление x^y производится по следующему принципу:

- если $y = 0$, то $x^y = 1$,
- если y – положительное четное, то $x^y = (x^2)^{y/2}$, и
- если y – нечетное, то $x^y = x^{y-1}x$.

Данный принцип позволяет понизить число умножений при вычислении x^y с $y - 1$ до $O(\log_2 y)$.

6 Задача верификации блок-схем

Пусть задана БС P, и её спецификация представляет собой пару формул $(Pre, Post)$ с переменными из $Var(P)$, имеющих следующий смысл:

- формула Pre называется **предусловием** и выражает условие, которому должны удовлетворять значения переменных БС P в момент начала её выполнения;
- формула $Post$ называется **постусловием**, и выражает условие, которому должны удовлетворять значения переменных БС P после завершения её выполнения.

Пусть P – БС и $\theta \in Var(P)^\bullet$. Мы будем говорить, что P **выполняется с начальной подстановкой** θ, если в момент начала выполнения P значение каждой переменной $x \in Var(P)$ было равно $\theta(x)$.

Запись $\theta \xrightarrow{P} \otimes$ обозначает утверждение: если P выполняется с начальной подстановкой θ, то выполнение P когда-либо завершится.

Если верно $\theta \xrightarrow{P} \otimes$, то запись $P\theta$ обозначает подстановку из $Var(P)^\bullet$, определяемую следующим образом: пусть P выполняется с начальной подстановкой θ, тогда $\forall\, x \in Var(P)$ значение $x^{P\theta}$ равно тому значению, которое будет иметь переменная x после завершения выполнения P.

Задача **верификации** БС P относительно спецификации $(Pre, Post)$ заключается в доказательстве следующего утверждения:

$$\forall\, \theta \in Var(P)^\bullet, \quad \text{если } \ Pre^\theta = 1, \ \text{ то } \ \theta \xrightarrow{P} \otimes \ \text{ и } \ Post^{P\theta} = 1. \qquad (11)$$

Данное утверждение обозначается записью $Pre \xrightarrow{P} Post$.

7 Метод инвариантов для верификации блок-схем

В этом параграфе излагается метод доказательства утверждений вида $Pre \xrightarrow{P} Post$, называемый **методом инвариантов**. Для его формули-

ровки введём понятия базового множества и базового пути.

8 Базовые множества и базовые пути

Пусть задана БС P. **Путь** в P – это последовательность $\pi = \alpha_1 \dots \alpha_k$ рёбер P, такая, что $\forall\, i = 1, \dots, k-1$ конец ребра α_i совпадает с началом ребра α_{i+1}. Путь $\pi = \alpha_1 \dots \alpha_k$ называется **циклом** в P, если $\alpha_1 = \alpha_k$.

Множество M точек на некоторых ребрах БС P называется **базовым** для P, если каждый цикл в P содержит ребро, на котором имеется точка из M. Если M – базовое множество для P, то путь π в P называется **базовым** относительно M, если он непуст, на первом и последнем ребрах этого пути имеются точки из M, и на других рёбрах этого пути точек из M нет. Множество всех базовых относительно M путей в P обозначается записью $\Pi_{P,M}$.

Докажем, что множество $\Pi_{P,M}$ конечно. Если $\Pi_{P,M}$ бесконечно, то оно содержит пути сколь угодно большой длины. Тогда в $\Pi_{P,M}$ есть путь π вида $\alpha_1 \dots \alpha_i \dots \alpha_j \dots \alpha_k$, где $\alpha_i = \alpha_j$, $1 < i < j < k$. Последовательность $\alpha_i \dots \alpha_j$ является циклом, и, согласно определению базового множества, одно из рёбер этого цикла содержит точку из M, что противоречит предположению о том, что путь π – базовый относительно M.

$\forall\, \pi \in \Pi_{P,M}$ будем обозначать записями $start(\pi)$ и $end(\pi)$ точки из M, которые лежат на первом и последнем ребре π соответственно.

Базовое множество M для БС P называется **полным**, если

- на ребре, выходящем из начальной вершины P, есть точка из M (такая точка называется **начальной**), и
- на каждом ребре, входящем в какую-либо терминальную вершину P, есть точка из M (такие точки называются **терминальными**).

Пусть M – полное базовое множество для БС P. Каждому выполнению $Exec$ БС P соответствует последовательность $\pi = \alpha_1\alpha_2 \dots$ рёбер P, в которой каждое ребро α_i является тем ребром, по которому происходит перемещение на шаге i этого выполнения от текущей вершины v_i к вершине v_{i+1}. Обозначим записью $i_1 i_2 \dots$ последовательность номеров тех рёбер из π, на которых есть базовые точки. Согласно определению понятий полного базового множества и выполнения БС, $i_1 = 1$, и для каждой пары (i_j, i_{j+1}) соседних индексов в $i_1 i_2 \dots$ последовательность $\pi_j = \alpha_{i_j} \dots \alpha_{i_{j+1}}$ является базовым относительно M путём. Если путь π конечен, то его последнее ребро входит в терминальную вершину и, следовательно, содержит точку из M. Таким образом, можно представить π

в виде последовательности $\pi_1\pi_2\ldots$, где π_1, π_2, $\ldots$ – базовые относительно M пути. Каждый член π_j этой последовательности мы будем называть **компонентой** выполнения $Exec$.

$\forall\,\pi \in \Pi_{P,M}, \forall\,\theta,\theta' \in Var(P)^\bullet$ запись $\theta \xrightarrow{\pi} \theta'$ означает, что π – компонента некоторого выполнения БС P, и θ,θ' – текущие подстановки в моменты прохода через точки $start(\pi)$ и $end(\pi)$ соответственно при движении по пути π во время этого выполнения.

9 Описание метода инвариантов

Пусть заданы БС P и её спецификация, выражаемая предусловием Pre и постусловием $Post$. Доказательство утверждения $Pre \xrightarrow{P} Post$ **методом инвариантов** имеет следующий вид.

1. Выбирается полное базовое множество точек M для P, и с каждой точкой $m \in M$ связывается формула $\varphi_m \in Fm$, называемая **инвариантом** в точке m, причем выполнены следующие условия:

   - если m – начальная точка, то $\varphi_m = Pre$,
   - если m – терминальная точка, то $\varphi_m = Post$,
   - для любого пути $\pi \in \Pi_{P,M}$ и любых подстановок $\theta,\theta' \in Var(P)^\bullet$, таких, что $\theta \xrightarrow{\pi} \theta'$, верна импликация

   $$\varphi^{\theta}_{start(\pi)} = 1 \quad \Rightarrow \quad \varphi^{\theta'}_{end(\pi)} = 1. \tag{12}$$

   Ниже, при анализе импликаций вида (12), $\forall\, x \in Var(P)$ будем обозначать значения x^θ и $x^{\theta'}$ записями x и x' соответственно.

2. Свойство завершаемости P $(\forall\,\theta \in Var(P)^\bullet \;\; Pre^\theta = 1 \Rightarrow \theta \xrightarrow{P} \otimes)$ обосновывается путем указания

   - базового множества N для P,
   - частично упорядоченного множества L, являющегося **фундированным**, т.е. такого, что не существует бесконечной строго убывающей последовательности $l_0, l_1, \ldots$ элементов L, и
   - множества термов $\{u_n \in Tm(Var(P)) \mid n \in N\}$,

   таких, что

   - при каждом выполнении P, $\forall\, n \in N$ при каждом проходе через n текущая подстановка θ удовлетворяет условию $u^\theta_n \in L$, и

- для любого пути $\pi \in \Pi_{P,N}$ и любых подстановок $\theta, \theta' \in Var(P)^\bullet$, таких, что $\theta \xrightarrow{\pi} \theta'$, верно неравенство $u^{\theta}_{start(\pi)} > u^{\theta'}_{end(\pi)}$.

Изложенный в этом пункте метод инвариантов открыт Р. Флойдом .

10 Обоснование метода инвариантов

Докажем, что если выполнены условия в пунктах 1 и 2 параграфа 9, то $Pre \xrightarrow{P} Post$, т.е. $\forall\, \theta \in Var(P)^\bullet$ из $Pre^\theta = 1$ следует $\theta \xrightarrow{P} \otimes$ и $Post^{P\theta} = 1$.

Пусть $Exec$ – произвольное выполнение БС P с начальной подстановкой θ, удовлетворящей условию $Pre^\theta = 1$. Этому выполнению соответствует последовательность $\pi = \alpha_1\alpha_2\ldots$ рёбер P, в которой каждое ребро α_i является тем ребром, по которому происходит перемещение на шаге i этого выполнения от текущей вершины v_i к вершине v_{i+1}.

Если бы выполнение $Exec$ было бесконечным, то π тоже была бы бесконечной, и некоторое ребро встречалось бы в ней бесконечно много раз. Пусть $i_1 i_2 \ldots$ – бесконечная последовательность номеров рёбер из π, таких, что $\alpha_{i_1} = \alpha_{i_2} = \ldots$. Подпоследовательности $\pi_{i_j} = \alpha_{i_j} \ldots \alpha_{i_{j+1}}$ $(j \geq 1)$ последовательности π являются циклами, и т.к. N – базовое множество, то $\forall\, j \geq 1 \quad \pi_{i_j}$ содержит ребро, на котором есть точка из N. Таким образом, точки из N присутствуют на бесконечном числе членов последовательности π. Обозначим номера таких членов записями $j_1, j_2, \ldots$, а точки из N на них – записями n_1, n_2, ... Пути $\alpha_{j_k} \ldots \alpha_{j_{k+1}}$ $(k \geq 1)$ являются базовыми относительно N, поэтому, согласно пункту 2 параграфа 9, текущие подстановки $\theta_{j_1}, \theta_{j_2}, \ldots$ удовлетворяют неравенствам

$$u^{\theta_{j_1}}_{n_1} > u^{\theta_{j_2}}_{n_2} > \ldots, \tag{13}$$

т.е. в L есть бесконечная строго убывающая последовательность (13), что противоречит предположению о фундированности L.

Таким образом, выполнение $Exec$ является конечным. В соответствии со сказанным в конце пункта 8 можно представить π в виде конечной последовательности $\pi_1 \ldots \pi_k$ путей из $\Pi_{P,M}$.

Согласно определениям выполнения и полного базового множества, а также условиям в пункте 1 параграфа 9:

- $m_0 \stackrel{\text{def}}{=} start(\pi_1)$ – начальная точка, поэтому $\varphi_{m_0} = Pre$,
- $m_k \stackrel{\text{def}}{=} end(\pi_k)$ – терминальная точка, поэтому $\varphi_{m_k} = Post$, и
- $\forall\, i = 1, \ldots, k-1 \quad m_i \stackrel{\text{def}}{=} end(\pi_{i-1}) = start(\pi_i) \in M$.

Кроме того, $\forall\, i = 1, \ldots, k$ текущие подстановки θ_{i-1} и θ_i вычисления $Exec$ до и после прохождения компоненты π_i последовательности $\pi_1 \ldots \pi_k$ соответственно удовлетворяют условию $\theta_{i-1} \stackrel{\pi_i}{\to} \theta_i$, поэтому, согласно (12),

$$\forall\, i = 1, \ldots, k \quad \varphi_{m_{i-1}}^{\theta_{i-1}} = 1 \;\Rightarrow\; \varphi_{m_i}^{\theta_i} = 1.$$

Суммируя все вышесказанное, получаем цепочку импликаций:

$$(Pre^{\theta} = 1) \Rightarrow (\varphi_{m_0}^{\theta} = 1) \Rightarrow (\varphi_{m_1}^{\theta_1} = 1) \Rightarrow \ldots$$
$$\ldots \Rightarrow (\varphi_{m_k}^{\theta_k} = 1) \Rightarrow (Post^{P\theta} = 1). \blacksquare$$

11 Примеры фундированных множеств

В качестве фундированных множеств, требуемых для обоснования завершаемости, можно брать, например, следующие множества:

- множество натуральных чисел $\mathbf{N} = \{0, 1, \ldots\}$,
- множество L^k кортежей длины k элементов произвольного фундированного множества L, с лексикографическим порядком, который определяется следующим образом: для любой пары кортежей $a = (a_1, \ldots, a_k)$, $b = (b_1, \ldots, b_k)$ из L^k

  $a \le b \Leftrightarrow a = b$ или $\exists\, i \in \{1, \ldots, k\} : a_i < b_i, \forall\, j \in \{1, \ldots, i-1\}\; a_j = b_j$.

  Обоснуем фундированность этого множества. Пусть существует бесконечная строго убывающая последовательность

  $$(a_1^1, \ldots, a_k^1) > (a_1^2, \ldots, a_k^2) > \ldots \tag{14}$$

  элементов множества L^k. Из (14) и из определения лексикографического порядка следует, что $a_1^1 \ge a_1^2 \ge \ldots$. Поскольку L фундировано, то в этой последовательности неравенств не может быть бесконечного количества строгих неравенств, т.е.

  $$\exists\, i_1 :\; a_1^{i_1} = a_1^{i_1+1} = \ldots. \tag{15}$$

  Из последнего соотношения и из (14) следует, что

  $$(a_2^{i_1}, \ldots, a_k^{i_1}) > (a_2^{i_1+1}, \ldots, a_k^{i_1+1}) > \ldots \tag{16}$$

  Применяя изложенные выше рассуждения к (16), получаем, что

  $$\exists\, i_2 \ge i_1 :\; a_2^{i_2} = a_2^{i_2+1} = \ldots, \tag{17}$$

откуда на основании (16) следует, что

$$(a_3^{i_2}, \ldots, a_k^{i_2}) > (a_3^{i_2+1}, \ldots, a_k^{i_2+1}) > \ldots .$$

Продолжая так и дальше, в конце концов получим, что

$$\exists\, i_k \geq i_{k-1} :\ a_k^{i_k} = a_k^{i_k+1} = \ldots . \tag{18}$$

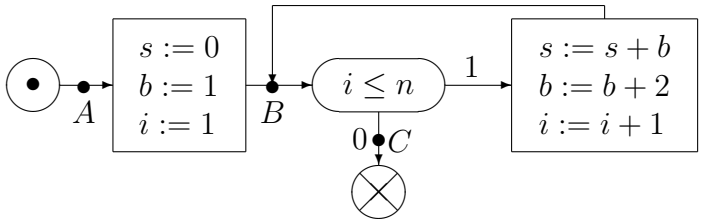
Из (15), (17), (18) следует, что кортежи в (14), начиная с кортежа с номером i_k, совпадают, что противоречит предположению. ■

12 Применение метода инвариантов

В этом пункте мы рассмотрим применение метода инвариантов для верификации БС, изложенных в пункте 5.

12.1 Верификация блок-схемы вычисления суммы

Верифицируем БС (7) относительно предусловия $n \geq 1$ и постусловия $s = n^2$. В качестве множества M точек, необходимого для верификации этой БС методом инвариантов, возьмем множество $\{A, B, C\}$ точек, изображенных ниже:



Инварианты в точках A, B, C имеют следующий вид:

$$\begin{aligned}
&\varphi_A = (n \geq 1) \quad \text{(предусловие)},\\
&\varphi_B = (s = (i-1)^2) \wedge (b = 2i - 1) \wedge (i \leq n + 1),\\
&\varphi_C = (s = n^2) \quad \text{(постусловие)}.
\end{aligned}$$

Для доказательства того, что инвариант φ_B определен правильно, необходимо исследовать все этапы выполнения этой БС, которые начинаются и заканчиваются проходом через точки из M.

1. На первом этапе выполнения этой БС происходит переход от A к B, с выполнением присваиваний в блоке между A и B (т.е $s := 0$,

$b := 1$ и $i := 1$), и (12) имеет вид

$$n \geq 1 \;\Rightarrow\; \left\{ \begin{array}{l} s' = (i'-1)^2 \\ b' = 2i' - 1 \\ i' \leq n' + 1 \end{array} \right\}. \tag{19}$$

Поскольку $s' = 0$, $b' = 1$, $i' = 1$ и $n' = n$, то (19) переписывается в виде

$$n \geq 1 \;\Rightarrow\; \left\{ \begin{array}{l} 0 = 0^2 \\ 1 = 2 \cdot 1 - 1 \\ 1 \leq n + 1 \end{array} \right\}, \tag{20}$$

что, очевидно, верно.

2. Рассмотрим произвольный этап выполнения этой БС, на котором происходит переход от B к B по циклу, здесь

   - выполняются присваивания $s := s + b$, $b := b + 2$ и $i := i + 1$, и
   - верно условие, на основании которого возможно движение по этому циклу ($i \leq n$).

   Импликация (12) с учетом этого условия имеет вид

$$\left\{ \begin{array}{l} s = (i-1)^2 \\ b = 2i - 1 \\ i \leq n + 1 \\ i \leq n \end{array} \right\} \;\Rightarrow\; \left\{ \begin{array}{l} s' = (i'-1)^2 \\ b' = 2i' - 1 \\ i' \leq n' + 1 \end{array} \right\}. \tag{21}$$

   Учитывая соотношения $s' = s + b$, $b' = b + 2$ $i' = i + 1$, $n' = n$, заключаем, что импликация (21) верна.

3. Рассмотрим этап выполнения БС, на котором происходит переход от B к C. Он возможен только при условии $i > n$. Импликация (12) с учетом этого условия имеет вид

$$\left\{ \begin{array}{l} s = (i-1)^2 \\ b = 2i - 1 \\ i \leq n + 1 \\ i > n \end{array} \right\} \;\Rightarrow\; s' = (n')^2. \tag{22}$$

   Учитывая равенства $s' = s$, $n' = n$, а также то, что i имеет тип **I**, и, следовательно, из $i \leq n + 1$ и $i > n$ следует, что $i = n + 1$, заключаем, что импликация (22) верна.

Для доказательства завершаемости положим

$$N \stackrel{\text{def}}{=} \{B\},\ L \stackrel{\text{def}}{=} \mathbf{N},\ u_B \stackrel{\text{def}}{=} n + 1 - i.$$

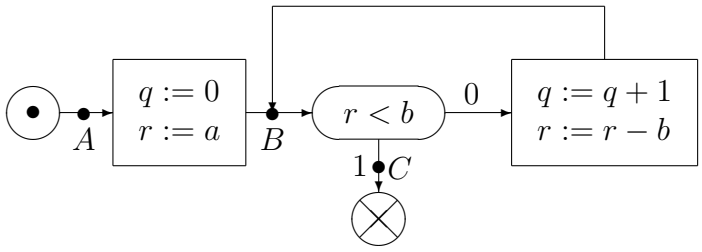
Из доказанного выше соотношения φ_B следует, что при каждом проходе через B текущая подстановка θ удовлетворяет условию $u_B^\theta \in \mathbf{N}$, и при произвольном переходе от B к B по циклу

$$u_B^\theta = (n + 1 - i) > (n + 1 - (i + 1)) = (n' + 1 - i') = u_B^{\theta'},$$

где θ и θ' – текущие подстановки до и после прохода по циклу.

12.2 Верификация блок-схемы деления с остатком

Верифицируем БС (8) относительно предусловия $(a \geq 0) \wedge (b \geq 1)$ и постусловия $(a = bq + r) \wedge (0 \leq r < b)$. Выберем точки A, B, C, как показано на рисунке:



Инвариантами φ_A и φ_C являются предусловие и постусловие соответственно, а инвариант φ_B имеет вид $(a = bq + r) \wedge (r \geq 0)$.

Докажем, что φ_B определен правильно.

1. При первом входе в B φ_B имеет вид $(a = b \cdot 0 + a) \wedge (a \geq 0)$, истинность данного соотношения следует из φ_A.

2. При переходе от B к B по циклу верно дополнительное условие $r \geq b$, с учетом которого импликация (12) имеет вид

$$\left\{ \begin{array}{l} a = bq + r \\ r \geq 0 \\ r \geq b \end{array} \right\} \Rightarrow \left\{ \begin{array}{l} a' = b'q' + r' \\ r' \geq 0 \end{array} \right\}. \qquad (23)$$

   Учитывая соотношения $a' = a$, $b' = b$, $q' = q + 1$, $r' = r - b$, заключаем, что импликация (23) верна.

3. При переходе от B к C верно дополнительное условие $r < b$, что в сочетании с φ_B влечёт φ_C.

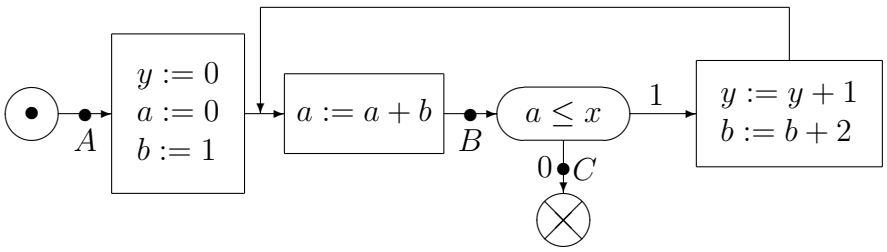
Для доказательства завершаемости положим $N \stackrel{\text{def}}{=} \{B\}$, $L \stackrel{\text{def}}{=} \mathbf{N}$, $u_B \stackrel{\text{def}}{=} r$. Из доказанного выше соотношения φ_B следует, что при каждом проходе через B текущая подстановка θ удовлетворяет условию $u_B^\theta \in \mathbf{N}$, и с учетом дополнительного соотношения $b \geq 1$ (истинность которого в точке B следует из φ_A и из того, что значение переменной b не изменяется в процессе выполнения БС) заключаем, что при произвольном переходе от B к B по циклу

$$u_B^\theta = r > r - b = r' = u_B^{\theta'},$$

где θ и θ' – текущие подстановки до и после прохода по циклу.

12.3 Верификация блок-схемы извлечения корня

Верифицируем БС (9) относительно предусловия $x \geq 0$ и постусловия $y = \lfloor \sqrt{x} \rfloor$. Выберем точки A, B, C, как показано на рисунке:



Определим инварианты в выбранных точках следующим образом:

$$\varphi_A \stackrel{\text{def}}{=} (x \geq 0),\ \varphi_B \stackrel{\text{def}}{=} \left\{ \begin{array}{l} y \geq 0 \\ y^2 \leq x \\ a = (y+1)^2 \\ b = 2y + 1 \end{array} \right\},\ \varphi_C \stackrel{\text{def}}{=} \left\{ \begin{array}{l} y \geq 0 \\ y^2 \leq x < (y+1)^2 \end{array} \right\}$$

(нетрудно видеть, что φ_C эквивалентна постусловию).

Докажем, что φ_B определен правильно.

1. При первом входе в B φ_B имеет вид $\left\{ \begin{array}{l} 0 \geq 0 \\ 0^2 \leq x \\ 1 = (0+1)^2 \\ 1 = 2 \cdot 0 + 1 \end{array} \right\}$, истинность данного соотношения следует из φ_A.

2. При переходе от B к B по циклу верно дополнительное условие

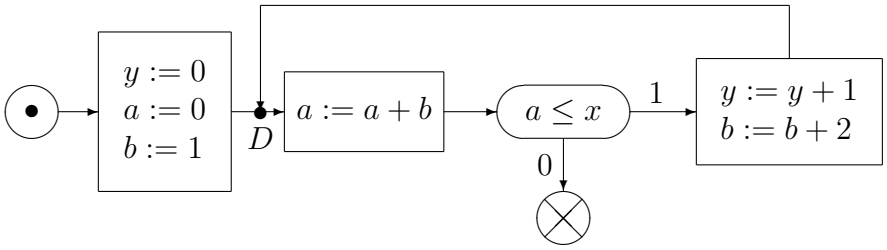
$a \le x$, с учетом которого импликация (12) имеет вид

$$\left\{ \begin{array}{l} y \ge 0 \\ y^2 \le x \\ a = (y+1)^2 \\ b = 2y+1 \\ a \le x \end{array} \right\} \Rightarrow \left\{ \begin{array}{l} y' \ge 0 \\ (y')^2 \le x' \\ a' = (y'+1)^2 \\ b' = 2y'+1 \end{array} \right\}. \qquad (24)$$

Учитывая соотношения $y' = y + 1$, $b' = b + 2$, $a' = a + b + 2$, $x' = x$, нетрудно установить, что импликация (24) верна.

3. При переходе от B к C верно дополнительное условие $x < a$, что в сочетании с φ_B влечёт φ_C.

Докажем завершаемость данной БС. Положим $N \stackrel{\text{def}}{=} \{D\}$, где точка D выбрана так, как показано на рисунке:



и, кроме того, $L \stackrel{\text{def}}{=} \mathbf{N}$, $u_D \stackrel{\text{def}}{=} x - a$. Для обоснования того, что

- при каждом проходе через D текущая подстановка θ удовлетворяет условию $u_D^\theta \in \mathbf{N}$ и
- при переходе от D к D по циклу $u_D^\theta = x - a > x' - a' = u_D^{\theta'}$, где θ и θ' – текущие подстановки до и после прохода по циклу,

определим вспомогательный инвариант в D: $\varphi_D \stackrel{\text{def}}{=} (x \ge a) \wedge (b \ge 1)$.

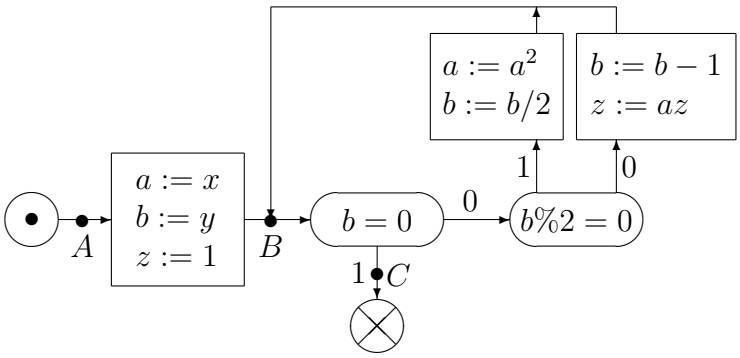
Истинность φ_D при каждом проходе через D следует из того, что

- φ_D верно при первом проходе через D (это следует из φ_A) и
- при произвольном переходе от D к D по циклу верно соотношение, написанное в условном операторе БС ($a \le x$).

Из истинности φ_D при каждом проходе через D следуют вышеупомянутые свойства, связанные с точкой D.

12.4 Верификация блок-схемы возведения в степень

Верифицируем БС (10) относительно предусловия $y \geq 0$ и постусловия $z = x^y$. Выберем точки A, B, C, как показано на рисунке:



Определим инварианты в выбранных точках следующим образом:

$$\varphi_A \stackrel{\text{def}}{=} (y \geq 0),\ \varphi_B \stackrel{\text{def}}{=} \left\{ \begin{array}{l} b \geq 0 \\ y \geq 0 \\ za^b = x^y \end{array} \right\},\ \varphi_C \stackrel{\text{def}}{=} (z = x^y).$$

Нетрудно доказать, что φ_B определен правильно.

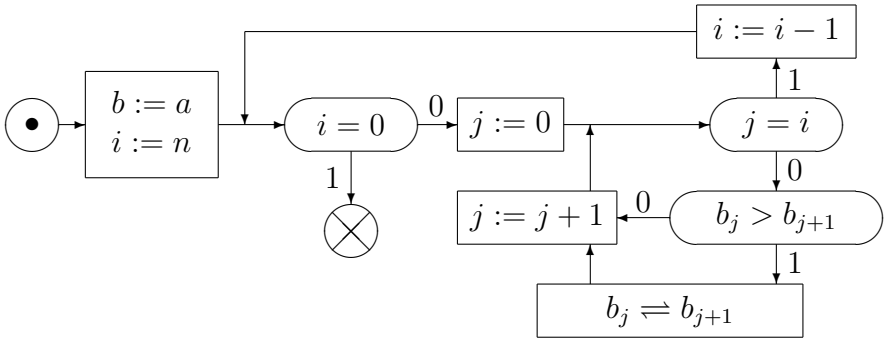
Для доказательства завершаемости данной БС можно взять $N \stackrel{\text{def}}{=} \{B\}$, $L \stackrel{\text{def}}{=} \mathbf{N}$, $u_B \stackrel{\text{def}}{=} b$.

Межфакультетский курс «Формальная семантика и верификация программного обеспечения» Лекция 2 Верификация программ (продолжение). Сортировка пузырьком

А.М.Миронов

1 Описание блок-схемы сортировки пузырьком

Опишем БС, реализующую сортировку массива методом пузырька. Сортируемый массив имеет вид $a_{0..n}$, где $n \geq 0$. Результат должен быть записан в массив $b_{0..n}$. Требуется, чтобы после завершения выполнения этой БС были верны утверждения $b = perm(a)$ и $ord(b)$. (Определения функций $perm$ и ord см. в предыдущей лекци)



(1)

В данном примере и ниже используется следующее обозначение: если a и b – массивы с одинаковыми типами компонентов и одинаковыми нижними и верхними индексами (т.е. имеют вид $a_{m..n}$ и $b_{m..n}$), то $a := b$ обозначает операцию копирования компонентов массива b в соответствующие

компоненты массива a, т.е. последовательность присваиваний $a_m := b_m$, $a_{m+1} := b_{m+1}$, ..., $a_n := b_n$.

Алгоритм, реализованный в БС (1), заключается в выполнении n прогонок по массиву b, каждая из которых заключается в просмотре слева направо участка от 0-й до i-й позиции сортируемого массива и перемене местами его соседних элементов, если они нарушают порядок. Сначала производится прогонка по всему массиву, в результате этой прогонки в последнюю позицию массива помещается его максимальный элемент, в результате следующей прогонки в предпоследнюю позицию помещается следующий по величине элемент и т.д.

2 Верификация блок-схемы сортировки

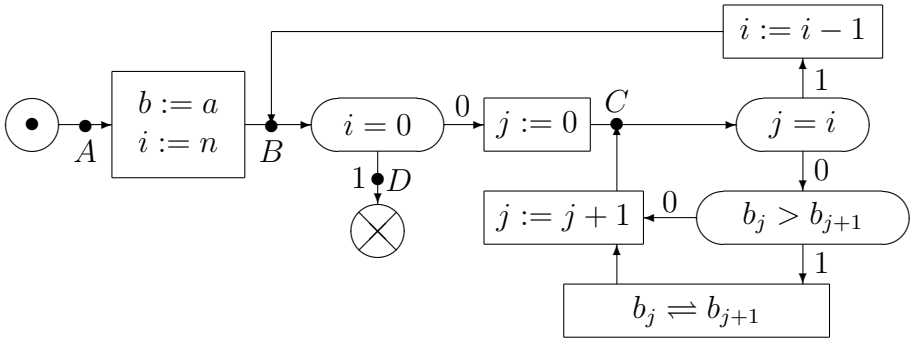
Верифицируем БС (1) относительно предусловия $n \geq 0$ и постусловия $(b = perm(a)) \wedge ord(b)$.

Конъюнктивные члены в постусловии обосновываются раздельно.

Обоснование утверждения $b = perm(a)$ заключается в том, что

- оно является верным после выполнения первого присваивания и
- все остальные действия в этой БС сохраняют его истинность, т.к. единственное действие в БС, которое изменяет массив b, это перестановка его соседних компонентов ($b_j \rightleftharpoons b_{j+1}$), однако эта перестановка не изменяет содержимое массива b.

Для обоснования утверждения $ord(b)$ выберем точки A, B, C, D, как показано на рисунке:



Инвариантами φ_A и φ_D являются формулы ($n \geq 0$) и $ord(b_{0..n})$ соответ-

ственно. Инварианты φ_B и φ_C имеют следующий вид:

$$\varphi_B \stackrel{\text{def}}{=} \left\{ \begin{array}{l} 0 \le i \le n \\ ord(b_{i..n}) \\ b_{0..i} \le b_{i+1..n} \end{array} \right\}, \quad \varphi_C \stackrel{\text{def}}{=} \left\{ \begin{array}{l} 1 \le i \le n \\ 0 \le j \le i \\ ord(b_{i..n}) \\ b_{0..i} \le b_{i+1..n} \\ b_{0..j-1} \le b_j \end{array} \right\},$$

где используются обозначения, введенные в пункте **??**.

Докажем, что инварианты φ_B и φ_C определены правильно.

1. При первом входе в B $i' = n$ и $\varphi_B^{\theta'} = \left\{ \begin{array}{l} 0 \le n \le n \\ ord(b_{n..n}) \\ b_{0..n} \le b_{n+1..n} \end{array} \right\}$. Истинность данного соотношения следует из φ_A.

2. При переходе от B к C верно дополнительное условие $i \neq 0$. Кроме того, меняет свое значение только переменная j ($j' = 0$). С учетом этого импликация (**??**) имеет вид

$$\left\{ \begin{array}{l} 0 \le i \le n \\ ord(b_{i..n}) \\ b_{0..i} \le b_{i+1..n} \\ i \neq 0 \end{array} \right\} \Rightarrow \left\{ \begin{array}{l} 1 \le i \le n \\ 0 \le 0 \le i \\ ord(b_{i..n}) \\ b_{0..i} \le b_{i+1..n} \\ b_{0..0-1} \le b_0 \end{array} \right\}. \qquad (2)$$

   Нетрудно видеть, что импликация (2) верна.

3. При переходе от C к B верно дополнительное условие $j = i$. Кроме того, меняет свое значение только i. С учетом этого (**??**) имеет вид

$$\left\{ \begin{array}{l} 1 \le i \le n \\ 0 \le j \le i \\ ord(b_{i..n}) \\ b_{0..i} \le b_{i+1..n} \\ b_{0..j-1} \le b_j \\ j = i \end{array} \right\} \Rightarrow \left\{ \begin{array}{l} 0 \le i' \le n \\ ord(b_{i'..n}) \\ b_{0..i'} \le b_{i'+1..n} \end{array} \right\}.$$

   Поскольку $i' = i - 1$, то последнюю импликацию можно переписать в виде

$$\left\{ \begin{array}{l} 1 \le i \le n \\ ord(b_{i..n}) \\ b_{0..i} \le b_{i+1..n} \\ b_{0..i-1} \le b_i \end{array} \right\} \Rightarrow \left\{ \begin{array}{l} 0 \le i - 1 \le n \\ ord(b_{i-1..n}) \\ b_{0..i-1} \le b_{i..n} \end{array} \right\}. \qquad (3)$$

   Нетрудно видеть, что импликация (3) верна.

4. При переходе от C к C по короткому циклу верны дополнительные условия $j \neq i$ и $b_j \leq b_{j+1}$. Кроме того, меняет свое значение только переменная j ($j' = j + 1$). С учетом этого (**??**) имеет вид

$$\left\{\begin{array}{l} 1 \leq i \leq n \\ 0 \leq j < i \\ ord(b_{i..n}) \\ b_{0..i} \leq b_{i+1..n} \\ b_{0..j-1} \leq b_j \\ b_j \leq b_{j+1} \end{array}\right\} \Rightarrow \left\{\begin{array}{l} 1 \leq i \leq n \\ 0 \leq j+1 \leq i \\ ord(b_{i..n}) \\ b_{0..i} \leq b_{i+1..n} \\ b_{0..j} \leq b_{j+1} \end{array}\right\}. \tag{4}$$

Нетрудно видеть, что импликация (4) верна.

5. При переходе от C к C по длинному циклу верны дополнительные условия $j \neq i$ и $b_j > b_{j+1}$. Кроме того, меняют свое значение переменная j и массив b ($j' = j + 1$, $b'_j = b_{j+1}$, $b'_{j+1} = b_j$, $\forall\, k \in \{0, \ldots, n\} \setminus \{j, j+1\} \quad b'_k = b_k$). С учетом этого (**??**) имеет вид

$$\left\{\begin{array}{l} 1 \leq i \leq n \\ 0 \leq j < i \\ ord(b_{i..n}) \\ b_{0..i} \leq b_{i+1..n} \\ b_{0..j-1} \leq b_j \\ b_{j+1} < b_j \end{array}\right\} \Rightarrow \left\{\begin{array}{l} 1 \leq i \leq n \\ 0 \leq j+1 \leq i \\ ord(b'_{i..n}) \\ b'_{0..i} \leq b'_{i+1..n} \\ b'_{0..j} \leq b'_{j+1} \end{array}\right\}. \tag{5}$$

Отдельно рассмотрим случаи $j + 1 < i$ и $j + 1 = i$.

- Если $j + 1 < i$, то

$$\left\{\begin{array}{l} b'_0 = b_0, \ldots, b'_{j-1} = b_{j-1}, \\ b'_j = b_{j+1}, b'_{j+1} = b_j, \\ b'_{j+2} = b_{j+2}, \ldots, b'_i = b_i, \ldots, b'_n = b_n, \end{array}\right.$$

  откуда нетрудно обосновать (5).

- Если $j + 1 = i$, то (5) следует из импликации

$$\left\{\begin{array}{l} 1 \leq i \leq n \\ ord(b_{i..n}) \\ b_{0..i} \leq b_{i+1..n} \\ b_{0..i-2} \leq b_{i-1} \\ b_i < b_{i-1} \end{array}\right\} \Rightarrow \left\{\begin{array}{l} ord(b'_{i..n}) \\ b'_{0..i} \leq b'_{i+1..n} \\ b'_{0..i-1} \leq b'_i \end{array}\right\} \tag{6}$$

при условии $\left\{\begin{array}{l} b'_0 = b_0, \ldots, b'_{i-2} = b_{i-2}, \\ b'_{i-1} = b_i, b'_i = b_{i-1}, \\ b'_{i+1} = b_{i+1}, \ldots, b'_n = b_n. \end{array}\right.$

6. При переходе от B к D верно дополнительное условие $i = 0$, что в сочетании с φ_B влечёт φ_D.

Докажем завершаемость БС (1). Положим $N \stackrel{\text{def}}{=} \{C\}$, $L \stackrel{\text{def}}{=} \mathbf{N}^2$ (лексикографический порядок), $u_C \stackrel{\text{def}}{=} (i, i-j)$. Из φ_C следует, что при проходе через C текущая подстановка θ удовлетворяет условию $u_C^\theta \in \mathbf{N}^2$ и при переходе от C к C по верхнему циклу $i' = i - 1$, поэтому

$$u_C^\theta = (i, i-j) > (i-1, \ldots) = u_C^{\theta'},$$

а при переходе от C к C по нижним циклам $i' = i$, $j' = j + 1$, поэтому

$$u_C^\theta = (i, i-j) > (i, i-(j+1)) = (i', i'-j') = u_C^{\theta'},$$

где θ и θ' – текущие подстановки до и после прохода по циклам.

Межфакультетский курс «Формальная семантика и верификация программного обеспечения» Лекция 3 Функциональные программы

А.М.Миронов

1 Парадигма функционального программирования

Функциональное программирование представляет собой одну из парадигм (т.е. совокупность понятий и методов) высокоуровневого программирования. Главной особенностью функционального программирования является представление программы не в виде последовательности действий, преобразующих входные данные в выходные, а в виде определений функций, значения которых должны вычисляться этими программами.

Основным достоинством парадигмы функционального программирования является то, что она позволяет описывать функциональную зависимость результата выполнения программы от её входных данных наиболее простым и естественным образом. Как писал один из классиков функционального программирования Лоуренс Паульсон:

> "Функциональное программирование ставит своей целью придать каждой программе простое математическое толкование, которое должно быть независимо от деталей исполнения."

Использование инструментов функционального программирования позволяет разрабатывать в короткие сроки компактные и легко понимаемые программы, в которых отсутствуют излишние детали, связанные с реализацией вспомогательных операций. На протяжении всей истории своего развития функциональное программирование зарекомендо-

вало себя как один из наиболее эффективных и надёжных инструментов программирования. Применение функционального программирования достигло наибольших успехов в задачах разработки систем искусственного интеллекта и обработки сложноструктурированных данных.

Метод функционального программирования заключается в построении описания вычислимых функций в виде систем функциональных уравнений, решениями которых должны быть описываемые функции. Этот метод альтернативен по отношению к рассмотренному в предыдущей части методу **императивного программирования**, при котором вычислимая функция определяется путём указания действий, преобразующих входные данные в выходные.

Функциональное программирование можно также использовать для

- описания **спецификаций**, т.е. свойств разрабатываемых программ, к числу которых относятся, например, свойства корректности, оптимальности, безопасности, устойчивости и т.д., а также
- быстрого создания прототипов требуемых функций, которые можно затем реализовывать более эффективно на языках программирования низкого уровня.

Во втором случае описание функции в виде функциональной программы рассматривается не как описание реального процесса вычисления её значений, а как эталон, предназначенный для контроля правильности реализации этой функции на языке программирования низкого уровня.

2 Математическое напоминание

2.1 Термы и связанные с ними понятия

Мы будем предполагать, что заданы следующие множества.

- Множество $Types$, элементы которого называются **типами**. Мы будем понимать типы так же, как понимаются типы данных в языках программирования. Каждому типу $\tau \in Types$ сопоставлено множество D_τ **значений** типа τ, называемое **доменом** типа τ.

  Символ $\mathcal{D}$ обозначает множество всех значений всех типов.

- Множество Var, элементы которого называются **переменными**. Каждой переменной $x \in Var$ сопоставлен тип $\tau(x) \in Types$. Каждая переменная $x \in Var$ может принимать **значения** в домене

$D_{\tau(x)}$, т.е. в различные моменты времени переменная x может быть связана с различными элементами домена $D_{\tau(x)}$.

- Множество Con, элементы которого называются **константами**.

  Каждой константе $c \in Con$ сопоставлены тип $\tau(c) \in Types$ и значение из $D_{\tau(c)}$, обозначаемое тем же символом c и называемое **интерпретацией** константы c. Будем считать, что $\forall\, \tau \in Types$ любой элемент $d \in D_\tau$ является константой типа τ, которой соответствует сам элемент d.

- Множество Fun, элементы которого называются **функциональными символами (ФС)**. Каждому ФС $f \in Fun$ сопоставлены

  – **функциональный тип (ФТ)** $\tau(f)$, который представляет собой запись вида

  $$(\tau_1, \ldots, \tau_n) \to \tau, \tag{1}$$

  где $\tau_1, \ldots, \tau_n, \tau \in Types$, и

  – частичная функция вида $D_{\tau_1} \times \ldots \times D_{\tau_n} \to D_\tau$, где $\tau(f)$ имеет вид (1), данная функция обозначается тем же символом f и называется **интерпретацией** ФС f.

  (Напомним, что функция $f : A \to B$ называется **частичной**, если $\forall\, a \in A$ значение $f(a)$ м.б. не определено.)

Функциональной переменной называется переменная, тип которой является функциональным типом. Множество всех функциональных переменных обозначается записью $FVar$.

Термы строятся из переменных, констант и ФС. Множество всех термов обозначается символом Tm. Каждый терм e имеет тип $\tau(e) \in Types$, определяемый структурой терма e.

Правила построения термов имеют следующий вид:

- каждая переменная и константа является термом того типа, который сопоставлен этой переменной или константе, и

- если $e_1, \ldots, e_n \in Tm$, $f \in Fun \cup FVar$, и $\tau(f)$ имеет вид (1), где $\tau_1 = \tau(e_1)$, …, $\tau_n = \tau(e_n)$, то запись $f(e_1, \ldots, e_n)$ – терм типа τ.

Терм $e \in Tm$ называется **подтермом** терма $e' \in Tm$, если либо $e = e'$, либо $e' = f(e_1, \ldots, e_n)$, где $f \in Fun \cup FVar$, и $\exists\, i \in \{1, \ldots, n\}$: e – подтерм терма e_i. Запись $e \subseteq e'$, где $e, e' \in Tm$, означает, что e – подтерм e'. Запись $e \subset e'$, где $e, e' \in Tm$, означает, что $e \subseteq e'$ и $e \neq e'$.

Индукцией по структуре терма $e \in Tm$ нетрудно доказать, что

если e_1 и e_2 – различные подтермы терма e, то либо $e_1 \subset e_2$, либо $e_2 \subset e_1$, либо e_1 и e_2 не имеют общих компонентов. (2)

Запись $x \in e$, где $x \in Var, e \in Tm$, означает, что x входит в e. Будем использовать следующие обозначения и соглашения:

- $\forall\, e \in Tm\ Var(e)$ обозначает множество $\{x \in Var \mid x \in e\}$,
- $\forall\, e_1, \ldots, e_n \in Tm\ Var(e_1, \ldots, e_n) = Var(e_1) \cup \ldots \cup Var(e_n)$,
- $\forall\, X \subseteq Var\ Tm(X)$ обозначает множество $\{e \in Tm \mid Var(e) \subseteq X\}$,
- $FVar(e)$ обозначает множество $Var(e) \cap FVar$,
- $\forall\, \tau \in Types$ запись Tm_τ обозначает множество $\{e \in Tm \mid \tau(e) = \tau\}$, $\forall\, E \subseteq Tm$ запись E_τ обозначает множество $E \cap Tm_\tau$,
- в терме вида $f(e)$ скобки слева и справа от e могут опускаться, если $\tau(f)$ имеет вид $(\tau) \to \tau'$,
- для каждой рассматриваемой функции $f : E \to E'$, где $E, E' \subseteq Tm$, будем предполагать, что $\forall\, e \in E\ \ \tau(e) = \tau(f(e))$.

Терм $e \in Tm$ **замкнут**, если $Var(e) = \emptyset$. Каждому замкнутому терму e соответствует объект $eval(e)$, называемый **значением** данного терма, и либо является элементом $D_{\tau(e)}$, либо не определён. Если e – константа, то $eval(e)$ – интерпретация этой константы, и если e имеет вид $f(e_1, \ldots, e_n)$, то $eval(e)$ определён, только если определено значение функции f на кортеже $(eval(e_1), \ldots, eval(e_n))$, и в этом случае $eval(e)$ равен этому значению. Для каждого замкнутого терма e будем обозначать объект $eval(e)$ той же записью, что и сам терм (т.е. e).

2.2 Примеры типов и функциональных символов

2.2.1 Арифметические типы и функциональные символы

Con содержит типы **N** и **I**, значениями которых являются натуральные $(0, 1, \ldots)$ и целые числа соответственно.

Fun содержит ФС $+, -, \cdot, div, mod$ типа $(\mathbf{I}, \mathbf{I}) \to \mathbf{I}$, и

- функции $+$, $-$, и $\cdot$ представляют собой соответствующие арифметические операции,

- *div* – частичная функция (определённая, только когда второй аргумент отличен от 0), *mod* – частичная функция (определённая, только когда второй аргумент больше 0), *div* и *mod* вычисляют частное и остаток соответственно от деления первого аргумента на второй.

Термы $+(e_1, e_2)$, $-(e_1, e_2)$, $\cdot(e_1, e_2)$, $div(e_1, e_2)$, $mod(e_1, e_2)$ будут записываться в виде $e_1 + e_2$, $e_1 - e_2$, $e_1 e_2$, e_1/e_2 и $e_1 \% e_2$ соответственно.

2.2.2 Логические типы и функциональные символы

Types содержит тип **B**, и $D_{\mathbf{B}} = \{0, 1\}$. Термы типа **B** называются **формулами**. Множество всех формул обозначается Fm. $\forall X \subseteq Var$ запись $Fm(X)$ обозначает множество $Tm(X) \cap Fm$. При построении формул могут использоваться обычные булевы ФС ($\neg$, $\wedge$, $\vee$, $\rightarrow$ и т.д.), которым соответствуют функции отрицания, конъюнкции, дизъюнкции и т.д. Символ 1 обозначает тождественно истинную формулу, а символ 0 – тождественно ложную формулу. Формулы вида $\wedge(e_1, e_2)$, $\vee(e_1, e_2)$ и т.п. мы будем записывать в более привычном виде $e_1 \wedge e_2$, $e_1 \vee e_2$ и т.д. Формулы вида $e_1 \wedge \ldots \wedge e_n$ и $e_1 \vee \ldots \vee e_n$ могут также записываться в виде $\left\{ \begin{array}{c} e_1 \\ \ldots \\ e_n \end{array} \right\}$ и $\left[\begin{array}{c} e_1 \\ \ldots \\ e_n \end{array} \right]$ соответственно, а также в виде $\bigwedge_{i \in \{1,\ldots,n\}} e_i$ и $\bigvee_{i \in \{1,\ldots,n\}} e_i$ соответственно, и также в виде $\{e_1, \ldots, e_n\}$ и $[e_1, \ldots, e_n]$ соответственно. Формулы вида $\neg e$ могут обозначаться $\bar{e}$.

Разные ФС могут иметь одинаковое обозначение. Ниже мы приводим примеры таких ФС. Для каждого типа τ

- *Fun* содержит ФС *eq* типа $(\tau, \tau) \rightarrow \mathbf{B}$, которому соответствует функция, отображающая пару $(d_1, d_2) \in D_\tau \times D_\tau$ в элемент 1, если $d_1 = d_2$, и 0, если $d_1 \neq d_2$;

- если на D_τ задано отношение частичного порядка, то *Fun* содержит ФС $<$, $\leq$, $>$, $\geq$ типа $(\tau, \tau) \rightarrow \mathbf{B}$, каждому из этих ФС соответствует функция, отображающая каждую пару $(d_1, d_2) \in D_\tau \times D_\tau$ в элемент
  - 1, если $d_1 < d_2$, $d_1 \leq d_2$, $d_1 > d_2$, $d_1 \geq d_2$ соответственно, и
  - 0, если соответствующее соотношение неверно;

- *Fun* содержит ФС *if_then_else* типа $(\mathbf{B}, \tau, \tau) \rightarrow \tau$, соответствующая функция отображает тройку вида $(1, d_1, d_2)$ в d_1 и тройку вида $(0, d_1, d_2)$ в d_2.

Термы $eq(e_1, e_2)$, $<(e_1, e_2)$ и т.д. будут записываться в виде $e_1 = e_2$, $e_1 < e_2$ и т.д. соответственно. Термы $if\_then\_else(e, e_1, e_2)$ будут записываться в виде **if** e **then** e_1 **else** e_2, или $e?e_1 : e_2$.

2.2.3 Строковые типы и функциональные символы

Types содержит типы **L** и **S**, значения которых называются **символами** и **символьными строками** (или просто **строками**) соответственно. Каждая строка представляет собой последовательность символов $a_1 \dots a_n$, где $n \geq 0$. При $n = 0$ эта последовательность пустая и обозначается ε.

Fun содержит

- ФС *head* и *tail* типа $\mathbf{S} \to \mathbf{L}$ и $\mathbf{S} \to \mathbf{S}$ соответственно, которым соответствуют частичные функции, определенные только для непустых строк, данные функции отображают строку $a_1 \dots a_n$ в символ a_1 и строку $a_2 \dots a_n$ соответственно (называемые **головой** и **хвостом** строки $a_1 \dots a_n$ соответственно);
- ФС *conc* типа $(\mathbf{L}, \mathbf{S}) \to \mathbf{S}$, которому соответствует функция, отображающая пару $(a, a_1 \dots a_n)$ в строку $aa_1 \dots a_n$.

Термы $conc(e, e')$, $head(e)$, $tail(e)$ будем записывать в сокращенном виде ee', e_h и e_t соответственно.

2.2.4 Кортежные типы и функциональные символы

Для каждого списка типов $\tau_1, \dots, \tau_n$ (некоторые компоненты этого списка могут совпадать)

- *Types* содержит тип, обозначаемый записью $(\tau_1, \dots, \tau_n)$, и

$$D_{(\tau_1,\dots,\tau_n)} = D_{\tau_1} \times \dots \times D_{\tau_n},$$

- *Fun* содержит ФС *tuple* типа $(\tau_1, \dots, \tau_n) \to (\tau_1, \dots, \tau_n)$, ему соответствует тождественная функция, термы вида $tuple(e_1, \dots, e_n)$ будут обозначаться записью $(e_1, \dots, e_n)$.

3 Примеры функциональных программ

Для неформальной иллюстрации понятия функциональной программы мы рассмотрим некоторые примеры функциональных программ, описывающих функции на символьных строках (которые мы будем называть просто строками).

Функциональная программа (ФП) представляет собой систему функциональных уравнений. Функция, которую определяет ФП, является первой компонентой решения этой системы функциональных уравнений.

3.1 Конкатенация строк

Приводимая ниже ФП определяет функцию на строках

$$append : D_{\mathbf{S}} \times D_{\mathbf{S}} \to D_{\mathbf{S}},$$

которая преобразует пару строк (u, v) в их **конкатенацию**, т.е.

- в строку $a_1 \dots a_n\, b_1 \dots b_m$, если u и v имеют вид соответственно

$$a_1 \dots a_n \quad \text{и} \quad b_1 \dots b_m \quad (n, m > 0),$$

- в строку v, если $u = \varepsilon$, и в строку u, если $v = \varepsilon$.

ФП, описывающая функцию *append*, имеет вид

$$\varphi(u, v) = \quad (u = \varepsilon)?v : u_h\varphi(u_t, v). \tag{3}$$

ФП (3) состоит из одного функционального уравнения, неизвестная величина в котором – функциональная переменная φ. Левая часть уравнения (3) состоит из функциональной переменной φ, соответствующей описываемой функции, и списка формальных параметров этой функции (u и v). Правая часть уравнения (3) представляет собой выражение, описывающее связь значения описываемой функции на паре аргументов (u, v) с её значениями на других аргументах. Уравнение (3) можно рассматривать как рекурсивный алгоритм вычисления функции *append*.

Нетрудно доказать, что функция *append* является единственным решением функционального уравнения (3).

Ниже будем обозначать терм $append(u, v)$ записью $u * v$ (или uv).

3.2 Инвертирование строки

Другим примером является ФП, описывающая функцию

$$reverse : D_{\mathbf{S}} \to D_{\mathbf{S}},$$

которая преобразует каждую строку u в строку, получаемую из u её записью в обратном порядке, т.е. $reverse(a_1 \dots a_n) = a_n \dots a_1$.

ФП, описывающая функцию *reverse*, имеет вид

$$\varphi(u) = (u = \varepsilon)?\varepsilon : \varphi(u_t) * (u_h\varepsilon).$$

В этой ФП используется функция *append*, которую мы описали в виде ФП в предыдущем параграфе.

3.3 Поиск подстроки

Третьим примером является ФП, описывающая функцию

$$find : D_{\mathbf{S}} \times D_{\mathbf{S}} \to D_{\mathbf{B}}.$$

Значением данной функции на паре строк u, v является 1, если u является подстрокой строки v, т.е. если v является значением выражения $x * (u * y)$ для некоторых x, y, и 0 иначе.

ФП, описывающая функцию $find$, имеет вид

$$\begin{cases} \varphi_1(u,v) = \varphi_2(u,v)?1 : \Big((v = \varepsilon)?0 : \varphi_1(u, v_t)\Big) \\ \varphi_2(u,v) = (u = \varepsilon)?1 : \Big((v = \varepsilon)?0 : ((u_h = v_h)?\varphi_2(u_t, v_t) : 0)\Big) \end{cases}.$$

Данная ФП представляет собой систему из двух функциональных уравнений. Эта система имеет единственное решение, представляющее собой пару функций

$$(find,\ \mathit{prefix})$$

($find$ соответствует φ_1, а *prefix* – φ_2), где функция $find$ является той функцией, для описания которой предназначена данная ФП, и функция *prefix* является вспомогательной функцией, она имеет вид

$$\mathit{prefix} : D_{\mathbf{S}} \times D_{\mathbf{S}} \to D_{\mathbf{B}};$$

её значением на паре строк u, v является 1, если u – префикс v, т.е. v имеет вид $u * x$ для некоторой x, и 0 иначе.

4 Метод структурной индукции

4.1 Описание метода

Метод **структурной индукции (СИ)** для верификации ФП может использоваться в том случае, когда

- доказываемое утверждение о ФП имеет вид

$$\forall\, \vec{x}\, e, \tag{4}$$

  где $\tau(e) = \mathbf{B}$ и $\vec{x} = Var(e)$, утверждение (4) считается верным, если для каждого $\vec{d} \in D_{\tau(\vec{x})}$ значение терма $e(\vec{d})$ равно 1, и

- существует фундированное частично упорядоченное множество (ЧУМ) P, такое, что каждому элементу $\vec{d} \in D_{\tau(\vec{x})}$ может быть сопоставлен некоторый элемент $c(\vec{d}) \in P$.

Напомним, что ЧУМ называется **фундированным (ФЧУМ)**, если в нём нет бесконечно убывающих цепей, т.е. подмножеств вида $\{p_0, p_1, \ldots\}$, где $p_0 > p_1 > \ldots$.

Приведем два примера ФЧУМ.

- Множество $D_{\mathbf{N}}$ натуральных чисел.
- Множество P^n списков длины n, состоящих из элементов ФЧУМ P, с **лексикографическим** отношением порядка: неравенство

$$(p_1, \ldots, p_n) < (p'_1, \ldots, p'_n)$$

верно, если $\exists\, i \in \{1, \ldots, n\}$: $p_i < p'_i$, $\forall\, j \in \{1, \ldots, i-1\}$ $p_j = p'_j$.

Метод СИ для обоснования утверждения

$$\forall\, \vec{d} \in D_{\tau(\vec{x})} \quad e(\vec{d}) = 1$$

в случае, когда определена функция c вида

$$c : D_{\tau(\vec{x})} \to P, \tag{5}$$

где P – ФЧУМ, заключается в доказательстве того, что для произвольного $\vec{d} \in D_{\tau(\vec{x})}$ верно утверждение

$$(\forall\, \vec{d'} \in D_{\tau(\vec{x})} \quad c(\vec{d'}) < c(\vec{d}) \;\Rightarrow\; e(\vec{d'}) = 1) \quad \Rightarrow \quad e(\vec{d}) = 1. \tag{6}$$

Данный метод обосновывается следующим образом: если для некоторого $\vec{d} \in D_{\tau(\vec{x})}$ значение $e(\vec{d})$ было бы равно 0, то $\exists\, \vec{d_0} \in D_{\tau(\vec{x})}$:

- $e(\vec{d_0}) = 0$ и
- $\forall\, \vec{d'} \in D_{\tau(\vec{x})} \quad c(\vec{d'}) < c(\vec{d_0}) \;\Rightarrow\; e(\vec{d'}) = 1$

(нетрудно доказать, что если такого $\vec{d_0}$ не существует, то в P можно построить бесконечно убывающую цепь). Эта ситуация противоречит утверждению (6) для $\vec{d} = \vec{d_0}$.

4.2 Примеры верификации функциональных программ методом структурной индукции

В излагаемом ниже примере $\tau(\vec{x}) = \mathbf{S}$, $P = D_{\mathbf{N}}$ и функция c типа $\mathbf{S} \to \mathbf{N}$ сопоставляет каждой строке x её длину (т.е. количество символов в ней). В этих примерах для любых $x, y \in D_{\mathbf{S}}$ запись $x < y$ означает, что длина x меньше длины y.

4.2.1 Верификация функциональной программы сортировки

Требуется доказать, что $\forall\, x \in D_{\mathbf{S}}$

$$\mathbf{ord}(\mathbf{sort}(x)) = 1, \tag{7}$$

где

- **sort** – функция сортировки строк, определяемая ФП:

$$\begin{cases} \mathbf{sort}(x) = (x = \varepsilon)?\ \varepsilon : \mathbf{insert}(x_h, \mathbf{sort}(x_t)) \\ \mathbf{insert}(a, y) = (y = \varepsilon) \quad ?\ a\varepsilon \\ \qquad\qquad\qquad\qquad\quad : (a \le y_h) \quad ?\ ay : y_h\, \mathbf{insert}(a, y_t) \end{cases}$$

- **ord** – функция проверки упорядоченности строки:

$$\begin{cases} \mathbf{ord}(x) = (x = \varepsilon) \quad ?\ 1 \\ \qquad\qquad\quad : (x_t = \varepsilon) \quad ?\ 1 \\ \qquad\qquad\qquad\qquad : (x_h \le (x_t)_h) \quad ?\ \mathbf{ord}(x_t) : 0 \end{cases}$$

  Ниже мы будем обозначать терм вида $\mathbf{insert}(a, y)$ записью $a \to y$.
  Если $x = \varepsilon$, то $\mathbf{sort}(x) = \varepsilon$ и

$$\mathbf{ord}(\mathbf{sort}(x)) = \mathbf{ord}(\varepsilon) = 1.$$

  Если $x \neq \varepsilon$, то доказываемое равенство (7) переписывается в виде

$$\mathbf{ord}(x_h \to \mathbf{sort}(x_t)) = 1. \tag{8}$$

По индуктивному предположению, верно равенство

$$\mathbf{ord}(\mathbf{sort}(x_t)) = 1,$$

из которого следует (8) по нижеследующей лемме.

Лемма

Имеет место импликация

$$\mathbf{ord}(y) = 1 \quad \Rightarrow \quad \mathbf{ord}(a \to y) = 1. \tag{9}$$

Доказательство.
Доказываем лемму индукцией по y.
Если $y = \varepsilon$, то правая часть в (9) имеет вид

$$\mathbf{ord}(a\varepsilon) = 1,$$

что верно по определению **ord**.

Пусть $y \neq \varepsilon$ и для каждого $z < y$ верна импликация

$$\mathbf{ord}(z) = 1 \quad \Rightarrow \quad \mathbf{ord}(a \to z) = 1. \tag{10}$$

Обозначим $c \stackrel{\text{def}}{=} y_h$, $d \stackrel{\text{def}}{=} y_t$.
(9) имеет вид

$$\mathbf{ord}(cd) = 1 \quad \Rightarrow \quad \mathbf{ord}(a \to cd) = 1. \tag{11}$$

Для доказательства импликации (11) нужно доказать, что при условии $\mathbf{ord}(cd) = 1$ верны импликации

(a) $a \leq c \quad \Rightarrow \quad \mathbf{ord}(a(cd)) = 1,$

(b) $c < a \quad \Rightarrow \quad \mathbf{ord}(c(a \to d)) = 1.$

(a) верно потому, что из $a \leq c$ следует

$$\mathbf{ord}(a(cd)) = \mathbf{ord}(cd) = 1.$$

Докажем (b).

- $d = \varepsilon$. В этом случае правая часть в (b) имеет вид

$$\mathbf{ord}(c(a\varepsilon)) = 1. \tag{12}$$

  (12) следует из $c < a$.

- $d \neq \varepsilon$. Обозначим $p \stackrel{\text{def}}{=} d_h$, $q \stackrel{\text{def}}{=} d_t$.

  В этом случае надо доказать, что при $c < a$

$$\mathbf{ord}(c(a \to pq)) = 1. \tag{13}$$

1. Если $a \leq p$, то (13) имеет вид

$$\mathbf{ord}(c(a(pq))) = 1. \tag{14}$$

Поскольку $c < a \leq p$, то (14) следует из равенств

$$\mathbf{ord}(c(a(pq))) = \mathbf{ord}(a(pq)) = \mathbf{ord}(pq) =$$
$$= \mathbf{ord}(c(pq)) = \mathbf{ord}(cd) = 1.$$

2. Если $p < a$, то (13) имеет вид

$$\mathbf{ord}(cp(a \to q)) = 1. \tag{15}$$

Поскольку по предположению

$$\mathbf{ord}(cd) = \mathbf{ord}(cpq) = 1,$$

то $c \leq p$, и поэтому (15) можно переписать в виде

$$\mathbf{ord}(p(a \to q)) = 1. \tag{16}$$

При $p < a$

$$a \to d = a \to pq = p(a \to q),$$

поэтому (16) можно переписать в виде

$$\mathbf{ord}(a \to d) = 1. \tag{17}$$

(17) следует по индуктивному предположению для леммы (т.е. из импликации (10), в которой $z \stackrel{\text{def}}{=} d$) из равенства

$$\mathbf{ord}(d) = 1,$$

которое обосновывается цепочкой равенств

$$1 = \mathbf{ord}(cd) = \mathbf{ord}(cpq) = \quad (\text{т.к. } c \leq p)$$
$$= \mathbf{ord}(pq) = \mathbf{ord}(d).$$

Межфакультетский курс «Формальная семантика и верификация программного обеспечения» Лекция 4 Параллельные и распределенные программы

А.М.Миронов

1 Понятие распределенной программы

1.1 Действия

Будем предполагать, что задано множество $Channels$, элементы которого называются **каналами**. $\forall\, c \in Channels$ множество Var содержит переменную x_c, значениями которой являются списки, интерпретируемые как **очереди сообщений** в канале c. Для каждого такого непустого списка $\vec{d} = d_1 \dots d_n$ записи $head(\vec{d})$ и $tail(\vec{d})$ обозначают значение d_1 и список $d_2 \dots d_n$ соответственно. Пустой список обозначается символом ε.

Будем называть **действием** последовательность $A = a_1 \dots a_n$, где $n \geq 0$ (т.е. данная последовательность м.б. пустой), каждая компонента a_i которой называется **элементарным действием (ЭД)** и имеет один из следующих видов:

(1) $e := e'$, где $e, e' \in Tm$, $\tau(e) = \tau(e')$,
(2) $\{\beta\}$, где $\beta \in Fm$,
(3) $c?e$ или $c!e$, где $c \in Channels, e \in Tm$,
(4) $?e$ или $!e$, где $e \in Tm$.

Компоненты вида (1) и (2) называются **присваиванием** и **условным переходом** соответственно. Компоненты вида (3) называются **вводом из канала** c и **выводом в канал** c сообщения e соответственно. Компоненты вида (4) называются **приемом** и **посылкой** сообщения e

соответственно. Будем предполагать, что среди компонентов A м.б. не более одной компоненты вида (4). Если в A нет компоненты вида (4), то будем называть A **внутренним** действием.

Количество компонентов в A называется **длиной** действия A и обозначается $|A|$. Если $|A| = 0$ (т.е. A пуста), то A обозначается символом ε. Множество всех действий обозначается символом Act. Если $A, A' \in Act$, то AA' обозначает конкатенацию последовательностей A и A'. $\forall\, A \in Act$ запись $Var(A)$ обозначает множество всех переменных, входящих в A.

Для удобства восприятия компоненты действия могут разделяться точкой с запятой, кроме того, они могут записываться не в виде последовательности, а в столбик.

1.2 Процессы

Процессом будем называть граф P со следующими свойствами:

- P имеет выделенные вершины $\odot$ и $\otimes$, называемые **начальной** и **терминальной** вершинами соответственно, из $\otimes$ не выходят рёбра,
- каждому ребру графа P сопоставлена метка $A \in Act$.

Процесс является формальным описанием поведения дискретной динамической системы, работа которой заключается в последовательном выполнении действий, связанных с вводом и выводом сообщений, приёмом и посылкой сообщений, а также с изменением значений переменных.

Каждое ребро процесса P представляется записью $v \xrightarrow{A} v'$, где v и v' – начало и конец этого ребра, A – метка этого ребра.

Для каждого процесса P записи $V(P)$ и $Channels(P)$ обозначают совокупности всех вершин графа P и каналов, входящих в P, соответственно. Запись $Var(P)$ обозначает множество, в которое входят все переменные процесса P, а также не входящие в P переменные x_c для каждого канала $c \in Channels(P)$ и переменная at_P. Будем предполагать, что значениями переменной at_P являются вершины из $V(P)$ и для каждого ребра $v \xrightarrow{A} v'$ процесса P первая компонента действия A имеет вид $\{at_P = v\}$, а последняя – $at_P := v'$, в записи меток рёбер эти условный переход и присваивание будут опускаться.

Если какая-либо вершина v процесса P не является начальной и множества всех ребер в P с концом в v и с началом в v имеют вид

$$\{v_i \xrightarrow{A_i} v \mid i = 1, \ldots, n\} \quad \text{и} \quad \{v \xrightarrow{A'_j} v'_j \mid j = 1, \ldots, n'\}$$

соответственно, где v отличается от всех вершин v_i и v'_j, и либо все действия $A_1, \ldots, A_n$ внутренние, либо все действия $A'_1, \ldots, A'_{n'}$ внутренние, то к данному графу можно применить операцию **редукции**, которая заключается в удалении v и связанных с нею ребер и добавлении ребер вида $v_i \xrightarrow{A_i A'_j} v'_j$, где $A_i A'_j$ – конкатенация последовательностей A_i и A'_j.

1.3 Распределенные программы

Распределенная программа (РП) – это семейство процессов

$$P = \{P_i \mid i \in I\}.$$

Записи $V(P)$, $Channels(P)$ и $Var(P)$ обозначают декартово произведение $\prod_{i \in I} V(P_i)$ и объединения $\bigcup_{i \in I} Channels(P_i)$ и $\bigcup_{i \in I} Var(P_i)$ соответственно.

С каждой РП P связано **предусловие** $Pre(P) \in Fm$. Как правило, $Pre(P)$ является конъюнкцией нескольких формул, среди которых присутствуют формулы $at^{\theta}_{P_i} = \odot$ и $x^{\theta}_c = \varepsilon$, где $i \in I$, $c \in Channels(P)$.

Состояние РП P – это замкнутая подстановка $\theta \in Var(P)^{\bullet}$.

Состояние $\theta \in Var(P)^{\bullet}$ называется **начальным**, если $Pre(P)^{\theta} = 1$, и **терминальным**, если $\forall\, i \in I\ at^{\theta}_{P_i} = \otimes$.

1.4 Каналы в распределенных программах

В описании РП используются **каналы**, предназначенные для передачи сообщений между процессами. Во время своей работы процессы могут передавать сообщения друг другу через каналы следующим образом:

- процесс-отправитель посылает свое сообщение в канал и
- процесс-получатель забирает это сообщение из канала.

В этой главе мы представляем модель поведения канала в виде идеального буфера типа FIFO (First Input – First Output), который в каждый момент содержит очередь сообщений: для каждого $c \in Channels(P)$ и каждого $\theta \in Var(P)^{\bullet}$ значение x^{θ}_c является очередью непрочитанных сообщений в канале c в состоянии θ. При записи в канал записываемое сообщение добавляется в конец очереди x^{θ}_c, а при чтении сообщения из канала берется первый элемент этой очереди. Поступившие в канал сообщения не теряются и сохраняют правильный порядок.

Однако в некоторых случаях для представления поведения канала передачи сообщений необходимо использовать другие модели. Канал может представлять собой сложную коммуникационную среду, в которой

пересылаемые сообщения могут подвергаться самым разным преобразованиям, и модель этого канала должна учитывать все эти свойства. Например, в канале могут происходить

- искажения пересылаемых сообщений, или их потеря,
- переупорядочение сообщений (отправитель может послать сначала сообщение m и затем другое сообщение m', а получатель может получить их в обратном порядке – сначала m', а потом – m, и
- дублирование пересылаемых сообщений, это возможно тогда, когда
  - отправитель посылает получателю сообщение m,
  - посланное сообщение m, двигаясь от отправителя к получателю, застревает в некоторой промежуточной точке (например, сервер, на котором хранилось сообщение m в процессе доставки, оказался надолго отключенным),
  - по истечении некоторого тайм-аута ожидания подтверждения об успешной доставке сообщения m отправитель принимает решение, что сообщение m пропало в канале, и посылает это сообщение повторно,
  - затем исходное сообщение m выходит из той промежуточной точки, в которой оно застряло, после чего оно присутствует в канале уже в двух экземплярах – в виде исходного сообщения и в виде повторно посланного дубликата.

Иногда для анализа количественных свойств анализируемых РП в модели канала важно учитывать различные количественные характеристики этого канала (например, максимальное число сообщений, которые могут одновременно содержаться в этом канале, вероятности потерь сообщений, искажения, задержки и т.п.).

1.5 Переходы в распределенных программах

Элементарный переход (ЭП) в РП P – это утверждение, обозначаемое записью $\theta \xrightarrow{a} \theta'$, где θ, $\theta' \in Var(P)^\bullet$, a – ЭД, входящее в метку некоторого ребра какого-либо процесса из P. С каждым ЭП l связана **реакция** $react(l)$, которая имеет один из следующих видов:

- **прием сообщения** и **посылка сообщения**, обозначаются записями $?d$ и $!d$ соответственно, где d – некоторое значение, такую реакцию имеют ЭП $\theta \xrightarrow{a} \theta'$, в которых a имеет вид $?e$ или $!e$,

- **невидимая реакция**, обозначается символом τ, такую реакцию имеют остальные ЭП.

ЭП $\theta \xrightarrow{a} \theta'$ определяется следующим образом:

- если $a = (e := e')$, то $\exists\, \tilde{\theta} \in \Theta(Var(e)) : e^{\tilde{\theta}} = (e')^{\theta}$ и $\theta' = \tilde{\theta}\theta$,
  поясним смысл перехода вида $\theta \xrightarrow{e:=e'} \theta'$: при его выполнении терм e рассматривается как шаблон, в котором необходимо заменить каждую переменную $x \in Var(e)$ на такое значение $x^{\tilde{\theta}}$, чтобы значение получившегося терма было равно $(e')^{\theta}$, подстановка θ' получается из θ заменой значения каждой переменной $x \in Var(e)$ на $x^{\tilde{\theta}}$,
- если $a = \{\beta\}$, то $\beta^{\theta} = 1$ и $\theta' = \theta$,
- если $a = c?e$, то $x_c^{\theta} \neq \varepsilon$, $\exists\, \tilde{\theta} \in \Theta(Var(e)) : e^{\tilde{\theta}} = head(x_c^{\theta})$, $\theta' = \tilde{\theta}(tail(x_c)/x_c)\theta$,
- если $a = c!e$, то $\theta' = (x_c^{\theta} e^{\theta}/x_c)\theta$,
- если $a =?e$, то $\exists\, \tilde{\theta} \in \Theta(Var(e))$: $react(\theta \xrightarrow{a} \theta') =?e^{\tilde{\theta}}$, $\theta' = \tilde{\theta}\theta$,
- если $a =!e$, то $react(\theta \xrightarrow{a} \theta') =!e^{\theta}$, $\theta' = \theta$.

Переход в РП P – это утверждение, обозначаемое записью $\theta \xrightarrow{A} \theta'$ или $\theta \xrightarrow{A|B} \theta'$, где $\theta, \theta' \in Var(P)^{\bullet}$ (θ называется **началом** данного перехода, а θ' – его **концом**) и A – метка некоторого ребра какого-либо процесса из P, A и B в переходе $\theta \xrightarrow{A|B} \theta'$ – метки некоторых ребер различных процессов из P. С каждым переходом l связана **реакция** $react(l)$, которая либо имеет вид $?d$ или $!d$, где d – некоторое значение (если переход имеет вид $\theta \xrightarrow{A} \theta'$ и A содержит ЭД вида $?e$ или $!e$), либо является невидимой реакцией τ (в остальных случаях).

Переходы $\theta \xrightarrow{A} \theta'$ и $\theta \xrightarrow{A|B} \theta'$ определяются рекурсивно:

- $\theta \xrightarrow{\varepsilon} \theta'$ и $\theta \xrightarrow{\varepsilon|\varepsilon} \theta'$ верно, если и только если $\theta = \theta'$,
- $\theta \xrightarrow{A|B} \theta'$ верно, если и только если верно $\theta \xrightarrow{B|A} \theta'$,
- если $A = aA'$, где a – ЭД, не прием или посылка, $\exists\, \tilde{\theta}$: $\theta \xrightarrow{a} \tilde{\theta}$, то
  - из $\tilde{\theta} \xrightarrow{A'} \theta'$ следует $\theta \xrightarrow{A} \theta'$, $react(\theta \xrightarrow{A} \theta') = react(\tilde{\theta} \xrightarrow{A'} \theta')$,
  - из $\tilde{\theta} \xrightarrow{A'|B} \theta'$ следует $\theta \xrightarrow{A|B} \theta'$,

- если $A = aA'$, где $a = ?e$ или $!e$, $\exists\,\tilde{\theta}$: $\theta \xrightarrow{a} \tilde{\theta}$, то
  - из $\tilde{\theta} \xrightarrow{A'} \theta'$ следует $\theta \xrightarrow{A} \theta'$, $react(\theta \xrightarrow{A} \theta') = react(\theta \xrightarrow{a} \tilde{\theta})$,
  - из $a = ?e$, $B = (!e')B'$, $e^{\tilde{\theta}} = (e')^{\theta}$ и $\tilde{\theta} \xrightarrow{A'|B'} \theta'$ следует $\theta \xrightarrow{A|B} \theta'$.

Переход $\theta \xrightarrow{A} \theta'$ интерпретируется следующим образом: если РП P в некоторый момент находился в состоянии θ и начиная с этого момента некоторый процесс P_i из P последовательно выполнял ЭД, входящие в A, а остальные процессы из P в течение всего этого времени находились в ожидании, то после завершения выполнения последовательности ЭД, входящих в A, новым состоянием РП P м.б. θ'.

Переход $\theta \xrightarrow{A|B} \theta'$ интерпретируется следующим образом: если РП P в некоторый момент находилась в состоянии θ и начиная с этого момента некоторые различные процессы P_i и P_j из P последовательно выполняли ЭД, входящие в A и B соответственно, и одна пара из этих действий представляла собой **синхронное взаимодействие** (когда в некоторый момент один из этих процессов выполнил посылку сообщения, которое было принято в этот же момент другим процессом), а остальные процессы из P в течение всего этого времени находились в ожидании, то после завершения выполнения процессами P_i и P_j всех ЭД из A и B новым состоянием РП P м.б. θ'.

1.6 Выполнение распределенной программы

Выполнение РП представляет собой порождение последовательности состояний $\pi = (\theta_0, \theta_1, \ldots)$ РП P, такой, что θ_0 – начальное состояние и для каждой пары θ, θ' соседних членов этой последовательности

- либо имеется переход $\theta \xrightarrow{A} \theta'$, где A – метка ребра какого-либо процесса P_i из P (будем называть такой переход P_i**-переходом**),
- либо имеется переход $\theta \xrightarrow{A|B} \theta'$, где A и B – метки ребер различных процессов из P.

Выполнение РП P называется **полным**, если соответствующая последовательность состояний является конечной, т.е. имеет вид $\theta_0, \ldots, \theta_n$, причем θ_n – терминальное состояние.

Будем использовать следующее обозначение: если задано выполнение $\pi = (\theta_0, \theta_1, \ldots)$ и θ, θ' – состояния, входящие в π, то запись $\theta <_\pi \theta'$ означает, что $\theta = \theta_i$ и $\theta' = \theta_j$ для некоторых индексов $i < j$.

Приведенное выше описание выполнения РП P не является точным описанием реального выполнения этой РП. Во время реального выполнения P допускается одновременное выполнение некоторых действий, относящихся к разным процессам, входящим в эту РП, и не связанных с синхронным взаимодействием. Будем предполагать, что при реальном выполнении P могут выполняться одновременно лишь такие ЭД, что если в одном из этих ЭД производится изменение значения некоторой переменной, то эта переменная не входит в другие из этих ЭД. Нетрудно видеть, что при таком выполнении значения переменных РП изменятся так же, как если бы данные ЭД исполнялись по очереди. Таким образом, для анализа логических свойств какой-либо РП можно без ограничения общности предполагать, что выполнение этой РП происходит в соответствии с нашей моделью, т.е. одновременно могут выполняться лишь такие ЭД в различных процессах из P, которые являются синхронным взаимодействием.

Межфакультетский курс «Формальная семантика и верификация программного обеспечения» Лекция 5 Неподвижные точки функциональных программ

А.М.Миронов

1 Пополненные домены и функции на них

1.1 Пополненные домены

Для каждого типа $\tau \in Types$ запись $\tilde{D}_\tau$ обозначает множество, состоящее из всех элементов домена D_τ, и ещё одного элемента ω (для обозначения этого элемента используется один и тот же символ ω для всех доменов), называемого **неопределённым значением**. Множество $\tilde{D}_\tau$ называется **пополненным доменом типа** τ. Будем рассматривать $\tilde{D}_\tau$ как **частично упорядоченное множество (ЧУМ)**, отношение порядка на котором определяется следующим образом: $\forall\, d_1, d_2 \in \tilde{D}_\tau$,

$$d_1 \le d_2 \;\;\Leftrightarrow\;\; d_1 = \omega \;\text{ или }\; d_1 = d_2. \tag{1}$$

Для каждого списка $\tilde{D}_1, \ldots, \tilde{D}_n$ пополненных доменов будем рассматривать их декартово произведение $\tilde{D}_1 \times \ldots \times \tilde{D}_n$ тоже как ЧУМ, отношение порядка на котором определяется следующим образом:

$$\forall\, \vec{d}, \vec{d'} \in \tilde{D}_1 \times \ldots \times \tilde{D}_n \quad \vec{d} \le \vec{d'} \;\Leftrightarrow\; d_1 \le d'_1,\; \ldots,\; d_n \le d'_n,$$

где $\vec{d} = (d_1, \ldots, d_n), \vec{d'} = (d'_1, \ldots, d'_n)$.

1.2 Монотонные функции

Функция $f : \tilde{D}_1 \times \ldots \times \tilde{D}_n \to \tilde{D}$ называется **монотонной**, если

$$\forall\, \vec{d}, \vec{d'} \in \tilde{D}_1 \times \ldots \times \tilde{D}_n \text{ из } \vec{d} \leq \vec{d'} \text{ следует } f(\vec{d}) \leq f(\vec{d'}).$$

Примером немонотонной функции является **нестрогое равенство**

$$\equiv\colon \tilde{D} \times \tilde{D} \to \tilde{D}_{\mathbf{B}},$$

где $\equiv (d_1, d_2) = 1$, если $d_1 = d_2$, и 0 иначе. Будем записывать $\equiv (d_1, d_2)$ в виде $d_1 \equiv d_2$. Немонотонность функции $\equiv$ следует из того, что

- $(\omega, \omega) \leq (\omega, d)$, где $d \in D$, но
- $(\omega \equiv \omega) = 1,\ (\omega \equiv d) = 0,\ 1 \not\leq 0$ в смысле отношения порядка на пополненных доменах.

1.3 Естественные продолжения

Напомним, что каждому $f \in Fun$ сопоставлена частичная функция

$$f :\ D_{\tau_1} \times \ldots \times D_{\tau_n} \to D_\tau,\ \text{где } \tau(f) = (\tau_1, \ldots, \tau_n) \to \tau. \tag{2}$$

Ниже в этой части будем считать, что каждый ФС $f \in Fun$ связан не с функцией вида (2), а с обозначаемой тем же символом f тотальной (т.е. всюду определённой) функцией

$$f : \tilde{D}_{\tau_1} \times \ldots \times \tilde{D}_{\tau_n} \to \tilde{D}_\tau, \tag{3}$$

которая определяется следующим образом:

- если $f \neq if\_then\_else$, то данная функция называется **естественным продолжением** исходной частичной функции f и каждому $\vec{d} \in \tilde{D}_{\tau_1} \times \ldots \times \tilde{D}_{\tau_n}$ данная функция сопоставляет
  - значение исходной частичной функции (2) на $\vec{d}$, если все компоненты $\vec{d}$ отличны от ω, и данное значение определено,
  - ω, если либо среди компонентов $\vec{d}$ есть ω, либо значение исходной частичной функции f на $\vec{d}$ не определено;
- если $f = if\_then\_else$, то соответствующие ему функции вида $\tilde{D}_{\mathbf{B}} \times \tilde{D}_\tau \times \tilde{D}_\tau \to \tilde{D}_\tau$ определяются не как естественные продолжения, а следующим образом:

$$(1, d_1, d_2) \mapsto d_1,\ \ (0, d_1, d_2) \mapsto d_2,\ \ (\omega, d_1, d_2) \mapsto \omega. \tag{4}$$

Нетрудно доказать, что все естественные продолжения и функции (4) являются монотонными функциями.

Выше было определено понятие функции, соответствующей терму: если заданы терм $e \in Tm$ и список $\vec{x} = (x_1, \ldots, x_n)$ различных переменных, причём $Var(e) \subseteq \{x_1, \ldots, x_n\}$, то терму e соответствует функция $e(\vec{x})$. Если считать, что ФС, входящим в e, сопоставлены естественные продолжения исходных функций или функции (4), то всему терму e м.б. сопоставлена функция

$$e(\vec{x}) : \tilde{D}_{\tau(x_1)} \times \ldots \times \tilde{D}_{\tau(x_n)} \to \tilde{D}_{\tau(e)}, \tag{5}$$

определяемая так же, как аналогичная функция (**??**). Нетрудно доказать, что функция (5) монотонна.

1.4 Частично упорядоченные множества монотонных функций

Для каждого функционального типа $ft = (\tau_1, \ldots, \tau_n) \to \tau$ запись D_{ft} обозначает множество всех монотонных функций f вида

$$f : \tilde{D}_{\tau_1} \times \ldots \times \tilde{D}_{\tau_n} \to \tilde{D}_{\tau}. \tag{6}$$

Будем рассматривать D_{ft} как ЧУМ, отношение порядка на котором определяется следующим образом: $\forall\, f_1, f_2 \in D_{ft}$ полагаем $f_1 \le f_2$, если

$$\forall\, \vec{d} \in \tilde{D}_{\tau_1} \times \ldots \times \tilde{D}_{\tau_n} \quad f_1(\vec{d}) \le f_2(\vec{d}),$$

где отношение неравенства на $\tilde{D}_\tau$ понимается в смысле определения (1).

Предполагаем, что Fun содержит ФС $\textcircled{\omega}$, которому соответствует функция из D_{ft}, где ft – произвольный ФТ, принимающая на каждом своём аргументе значение ω. Нетрудно видеть, что $\forall\, f \in D_{ft} \quad \textcircled{\omega} \le f$.

Цепь в D_{ft} – это последовательность $\{f_i \mid i \ge 0\}$ элементов D_{ft}, удовлетворяющая условию: $f_0 \le f_1 \le f_2 \le \ldots$. Это условие эквивалентно тому, что $\forall\, \vec{d} \in \tilde{D}_{\tau_1} \times \ldots \times \tilde{D}_{\tau_n}$ верна цепочка неравенств

$$f_0(\vec{d}) \le f_1(\vec{d}) \le f_2(\vec{d}) \le \ldots. \tag{7}$$

Согласно определению порядка на пополненных доменах, (7) эквивалентно условию: либо $\forall\, i \ge 0 \quad f_i(\vec{d}) = \omega$, либо $\exists\, i \ge 0$, $\exists\, d \in D_\tau$:

$$\forall j < i \quad f_j(\vec{d}) = \omega, \quad \forall j \ge i \quad f_j(\vec{d}) = d. \tag{8}$$

Обозначим записью $\sup_{i\geq 0} f_i$ функцию вида (6), определяемую следующим образом: $\forall\, \vec{d} \in \tilde{D}_{\tau_1} \times \ldots \times \tilde{D}_{\tau_n}$

$$\sup_{i\geq 0} f_i(\vec{d}) \stackrel{\text{def}}{=} \begin{cases} \omega & \text{если } \forall\, i \geq 0 \quad f_i(\vec{d}) = \omega, \\ d & \text{если } \exists\, i \geq 0 : \text{ верно (8).} \end{cases} \tag{9}$$

Нетрудно доказать, что функция $f = \sup_{i\geq 0} f_i$ монотонна (т.е. принадлежит D_{ft}) и является наименьшей верхней гранью цепи $\{f_i \mid i \geq 0\}$, т.е. $\forall\, i \geq 0 \quad f_i \leq f$, и $\forall\, f' \in D_{ft}$ если $\forall\, i \geq 0 \quad f_i \leq f'$, то $f \leq f'$.

1.5 Полные частично упорядоченные множества

ЧУМ P называется **полным**, если P содержит наименьший элемент, т.е. такой элемент $\mathbf{0}$, что $\forall\, p \in P\ \mathbf{0} \leq p$, и для каждой цепи $p_0 \leq p_1 \leq p_2 \leq \ldots$ элементов P существует **наименьшая верхняя грань** этой цепи, т.е. такой элемент $\sup_{i\geq 0} p_i \in P$, который обладает следующими свойствами: $\forall\, i \geq 0 \quad p_i \leq \sup_{i\geq 0} p_i$, и если элемент $p' \in P$ таков, что $\forall\, i \geq 0 \quad p_i \leq p'$, то $\sup_{i\geq 0} p_i \leq p'$.

Из рассуждений в конце параграфа 1.4 вытекает, что D_{ft} – полное ЧУМ. Наименьшим элементом D_{ft} является функция $\textcircled{\omega}$.

2 Функциональные программы

2.1 Понятие функциональной программы

Функциональная программа (ФП) – это совокупность Σ формальных равенств вида

$$\begin{cases} \varphi_1(\vec{x}_1) = e_1 \\ \ldots \\ \varphi_n(\vec{x}_n) = e_n \end{cases}, \tag{10}$$

где $\varphi_1, \ldots, \varphi_n$ – различные переменные из $FVar$, и $\forall\, i = 1, \ldots, n$,

- $\varphi_i(\vec{x}_i)$ и e_i – термы одинакового типа,
- $Var(e_i) \subseteq \vec{x}_i$, $FVar(e_i) \subseteq \{\varphi_1, \ldots, \varphi_n\}$.

ФП (10) можно рассматривать как систему функциональных уравнений, решением которой является произвольный список $\vec{f} = (f_1, \ldots, f_n)$ функций, таких, что

$$\forall\, i = 1, \ldots, n \quad f_i = e_i^{(f_1/\varphi_1, \ldots, f_n/\varphi_n)}(\vec{x}_i). \tag{11}$$

Ниже запись $\Sigma : \left\{ \begin{array}{l} \varphi_1(\vec{x}_1) = e_1 \\ \cdots \\ \varphi_n(\vec{x}_n) = e_n \end{array} \right.$ означает, что Σ – это ФП, состоящая из равенств, изображённых справа от фигурной скобки. Если Σ состоит из одного равенства, то фигурная скобка может отсутствовать.

2.2 Функционал, соответствующий функциональной программе

Пусть Σ – ФП вида (10).

Обозначим P_Σ декартово произведение $D_{\tau(\varphi_1)} \times \ldots \times D_{\tau(\varphi_n)}$.

Будем рассматривать P_Σ как ЧУМ, отношение порядка на котором определяется следующим образом: $\forall\, \vec{f}, \vec{f'} \in P_\Sigma$, где

$$\vec{f} = (f_1, \ldots, f_n), \quad \vec{f'} = (f'_1, \ldots, f'_n);$$

будем полагать $\vec{f} \le \vec{f'}$, если $f_1 \le f'_1,\ \ldots,\ f_n \le f'_n$.

Нетрудно видеть, что P_Σ – полное ЧУМ, т.к.

- список $(\textcircled{\omega}, \ldots, \textcircled{\omega}) \in P_\Sigma$ является наименьшим элементом в P_Σ и
- для каждой цепи $\vec{f_0} \le \vec{f_1} \le \vec{f_2} \le \ldots$ элементов P_Σ существует её наименьшая верхняя грань $\sup_{i\ge 0} \vec{f_i}$, которая имеет следующий вид:

$$\sup_{i\ge 0} \vec{f_i} \;=\; (\sup_{i\ge 0} f_{i1}, \ldots, \sup_{i\ge 0} f_{in}),$$

где $\forall\, i \ge 0 \quad \vec{f_i} = (f_{i1}, \ldots, f_{in})$.

Функционал, соответствующий ФП Σ, это отображение F_Σ вида $F_\Sigma : P_\Sigma \to P_\Sigma$, которое сопоставляет каждому списку функций

$$\vec{f} = (f_1, \ldots, f_n) \in P_\Sigma$$

список функций $\vec{f'} = (f'_1, \ldots, f'_n)$, определяемых следующим образом:

$$\forall\, i \in \{1, \ldots, n\} \quad f'_i \stackrel{\text{def}}{=} e_i^{(f_1/\varphi_1, \ldots, f_n/\varphi_n)}(\vec{x}_i).$$

2.3 Непрерывные функционалы на полных частично упорядоченных множествах

Пусть P – полное ЧУМ.

Функционал на P – это отображение F вида $F : P \to P$.

Функционал $F : P \to P$ называется

- **монотонным**, если $\forall\, p_1, p_2 \in P$ из $p_1 \leq p_2$ следует $F(p_1) \leq F(p_2)$;
- **непрерывным**, если он является монотонным и для каждой цепи $p_0 \leq p_1 \leq p_2 \leq \ldots$ элементов P верно равенство

  $$F(\sup_{i\geq 0} p_i) = \sup_{i\geq 0} F(p_i)$$

  (отметим, что правая часть данного равенства имеет смысл, поскольку из свойства монотонности F следует, что последовательность $\{F(p_i) \mid i \geq 0\}$ является цепью).

Непрерывность функционала не является следствием его монотонности. Например, рассмотрим функционал F на D_{ft}, где $ft = (\mathbf{N}) \to \mathbf{N}$:

$$\forall f \in D_{ft} \quad F(f) \stackrel{\text{def}}{=} \begin{cases} f, & \text{если } \forall\, k \geq 0 \quad f(k) = k, \\ \textcircled{\omega} & \text{иначе.} \end{cases}$$

Данный функционал является монотонным, но не непрерывным, т.к. цепь $\{f_i \mid i \geq 0\}$ функций из D_{ft}, где $\forall\, i \geq 0$ функция f_i соответствует терму $(x < i)?\ x : \omega$, обладает следующими свойствами:

- $\forall\, i \geq 0 \quad F(f_i) = \textcircled{\omega}$, поэтому $\sup_{i\geq 0} F(f_i) = \textcircled{\omega}$;
- $\sup_{i\geq 0} f_i = id = F(\sup_{i\geq 0} f_i)$, где id – тождественная функция.

Будем использовать следующее обозначение: если F – функционал на P, то записи F^0, F^1, $\ldots$ обозначают функционалы на P, определяемые следующим образом: $\forall\, p \in P\ F^0(p) \stackrel{\text{def}}{=} p$ и $\forall\, i \geq 0 \quad F^{i+1}(p) \stackrel{\text{def}}{=} F(F^i(p))$.

2.4 Неподвижные точки функционалов на полных частично упорядоченных множествах

Пусть F – функционал на полном ЧУМ P. Элемент $p \in P$ называется

- **неподвижной точкой (НТ)** функционала F, если $F(p) = p$;
- **наименьшей НТ (ННТ)** функционала F, если p – НТ функционала F и для каждой НТ p' функционала F $p \leq p'$.

Отметим, что если ННТ функционала F существует, то она единственна, т.к. если p' и p'' – две ННТ функционала F, то, согласно определению ННТ, должны быть верны неравенства $p' \leq p''$ и $p'' \leq p'$, откуда, в силу свойства антисимметричности отношения частичного порядка, получаем совпадение p' и p''.

Из определений в пунктах 2.1 и 2.2 следует, что для каждой ФП Σ список функций $\vec{f} \in P_\Sigma$ является решением системы Σ тогда и только тогда, когда $\vec{f}$ является НТ функционала F_Σ.

Теорема 1
Если P – полное ЧУМ и $F : P \to P$ – непрерывный функционал, то существует ННТ функционала F.

Доказательство.
Определим последовательность $\{p_i \mid i \geq 0\}$ элементов P следующим образом: $p_0 \stackrel{\text{def}}{=} \mathbf{0}$, где $\mathbf{0}$ – наименьший элемент P, и $\forall\, i \geq 0 \quad p_{i+1} \stackrel{\text{def}}{=} F(p_i)$.

Последовательность $\{p_i \mid i \geq 0\}$ является цепью, это можно доказать по индукции: $p_0 = \mathbf{0} \leq p_1$, и $\forall\, i \geq 0$, из неравенства $p_i \leq p_{i+1}$ и монотонности функционала F следует соотношение

$$p_{i+1} = F(p_i) \leq F(p_{i+1}) = p_{i+2}.$$

Обозначим символом p элемент $\sup_{i\geq 0} p_i$. Имеем:

- p является НТ функционала F, т.к., согласно свойству непрерывности F, верны соотношения

$$F(p) = F(\sup_{i\geq 0} p_i) = \sup_{i\geq 0} F(p_i) = \sup_{i\geq 0} p_{i+1} = p;$$

- p является ННТ функционала F, т.к. если $F(p') = p'$, то

$$\forall\, i \geq 0 \quad p_i \leq p'. \tag{12}$$

  (12) можно доказать по индукции:

  - $p_0 = \mathbf{0} \leq p'$;
  - если $p_i \leq p'$, то из монотонности F следует

$$p_{i+1} = F(p_i) \leq F(p') = p'.$$

  Согласно определению точной верхней грани, из (12) следует искомое неравенство $p = \sup_{i\geq 0} p_i \leq p'$ ■

2.5 Непрерывность функционалов, соответствующих функциональным программам

Теорема 2

Для каждой ФП Σ функционал F_Σ непрерывен.

Доказательство.

Мы рассмотрим лишь случай, когда ФП Σ состоит из одного уравнения (общий случай рассматривается аналогично), т.е. Σ имеет вид

$$\Sigma : \qquad \varphi(\vec{x}) = e. \tag{13}$$

Для каждого подтерма $e' \subseteq e$ верны включения: $FVar(e') \subseteq \{\varphi\}$ и $Var(e') \subseteq \vec{x}$. Поэтому для каждого подтерма $e' \subseteq e$ можно определить функционал $F_{e'}$ на $D_{\tau(\varphi)}$:

$$\forall\, f \in D_{\tau(\varphi)} \;\; F_{e'}(f) \stackrel{\text{def}}{=} (e')^{(f/\varphi)}(\vec{x}).$$

Докажем индукцией по структуре e', что функционал $F_{e'}$ непрерывен. Отметим, что из этого следует утверждение теоремы, т.к. $F_\Sigma = F_e$.

Если e' не содержит φ, то множество значений функционала $F_{e'}$ состоит из одного элемента. Очевидно, что такой функционал непрерывен.

Если e' содержит φ, то возможны два случая:

- $e' = g(e_1, \ldots, e_m)$, где $g \in Fun$, и
- $e' = \varphi(e_1, \ldots, e_m)$, где $\varphi \in FVar$.

Разберём лишь первый случай (второй случай разбирается аналогично).

Предположим, что функционалы $F_{e_1}, \ldots, F_{e_m}$ непрерывны. Докажем, что функционал $F_{e'}$, где $e' = g(e_1, \ldots, e_m),\ g \in Fun$, также непрерывен.

Согласно определению понятия функции, соответствующей терму, $\forall\, f \in D_{\tau(\varphi)}$, $\forall\, \vec{d} \in D_{\tau(\vec{x})}$ верны равенства

$$\begin{array}{l} F_{e'}(f)(\vec{d}) = (e')^{(f/\varphi)}(\vec{d}) = g(e_1, \ldots, e_m)^{(f/\varphi)}(\vec{d}) = \\ = g(e_1^{(f/\varphi)}, \ldots, e_m^{(f/\varphi)})(\vec{d}) = g(e_1^{(f/\varphi)}(\vec{d}), \ldots, e_m^{(f/\varphi)}(\vec{d})) = \\ = g(F_{e_1}(f)(\vec{d}), \ldots, F_{e_m}(f)(\vec{d})). \end{array} \tag{14}$$

Поэтому, если функции $f_1, f_2 \in D_{\tau(\varphi)}$ таковы, что $f_1 \leq f_2$, то в силу монотонности функционалов $F_{e_1}, \ldots, F_{e_m}$, $\forall\, \vec{d} \in D_{\tau(\vec{x})}$ верны соотношения

$$F_{e_1}(f_1)(\vec{d}) \leq F_{e_1}(f_2)(\vec{d}), \ldots, F_{e_m}(f_1)(\vec{d}) \leq F_{e_m}(f_2)(\vec{d}),$$

откуда, учитывая (14) и монотонность функции g, получаем:

$$F_{e'}(f_1)(\vec{d}) = g(F_{e_1}(f_1)(\vec{d}), \ldots, F_{e_m}(f_1)(\vec{d})) \leq$$
$$\leq g(F_{e_1}(f_2)(\vec{d}), \ldots, F_{e_m}(f_2)(\vec{d})) = F_{e'}(f_2)(\vec{d}),$$

т.е. функционал $F_{e'}$ является монотонным.

Докажем, что для каждой цепи $\{f_i \mid i \geq 0\}$ в $D_{\tau(\varphi)}$ верно равенство

$$F_{e'}(\sup_{i\geq 0} f_i) = \sup_{i\geq 0} F_{e'}(f_i). \tag{15}$$

$\forall\, i \geq 0$ из $f_i \leq \sup_{i\geq 0} f_i$, и из монотонности $F_{e'}$ следует неравенство

$$F_{e'}(f_i) \leq F_{e'}(\sup_{i\geq 0} f_i),$$

откуда, согласно определению наименьшей верхней грани, следует

$$\sup_{i\geq 0}\ F_{e'}(f_i) \leq F_{e'}(\sup_{i\geq 0} f_i).$$

Для доказательства (15) докажем, что верно обратное неравенство:

$$F_{e'}(\sup_{i\geq 0} f_i) \leq \sup_{i\geq 0}\ F_{e'}(f_i), \tag{16}$$

т.е. $\forall\, \vec{d} \in D_{\tau(\vec{x})}$ верно неравенство

$$F_{e'}(\sup_{i\geq 0} f_i)(\vec{d}) \leq \sup_{i\geq 0}\ F_{e'}(f_i)(\vec{d}). \tag{17}$$

Согласно (14), левую часть (17) можно переписать в виде

$$g(F_{e_1}(\sup_{i\geq 0} f_i)(\vec{d}), \ldots, F_{e_m}(\sup_{i\geq 0} f_i)(\vec{d})). \tag{18}$$

По индуктивному предположению, верны равенства

$$F_{e_1}(\sup_{i\geq 0} f_i) = \sup_{i\geq 0} F_{e_1}(f_i), \ldots, F_{e_m}(\sup_{i\geq 0} f_i) = \sup_{i\geq 0} F_{e_m}(f_i).$$

Поэтому (18) можно переписать в виде

$$g\left(\sup_{i\geq 0}\ F_{e_1}(f_i)(\vec{d}), \ldots, \sup_{i\geq 0}\ F_{e_m}(f_i)(\vec{d})\right). \tag{19}$$

Из (9) следует, что существует номер j, такой, что

$$\sup_{i\geq 0}\ F_{e_1}(f_i)(\vec{d}) = F_{e_1}(f_j)(\vec{d}), \ldots, \sup_{i\geq 0}\ F_{e_m}(f_i)(\vec{d}) = F_{e_m}(f_j)(\vec{d}).$$

Поэтому (19) можно переписать в виде

$$g\left(F_{e_1}(f_j)(\vec{d}), \ldots, F_{e_m}(f_j)(\vec{d})\right). \tag{20}$$

Согласно (14), значение выражения (20) равно $F_{e'}(f_j)(\vec{d})$.

Поэтому доказываемое неравенство (17) можно переписать в виде

$$F_{e'}(f_j)(\vec{d}) \leq \sup_{i\geq 0} F_{e'}(f_i)(\vec{d}).$$

Очевидно, что последнее неравенство верно. ■

Таким образом, согласно теоремам 1 и 2, у функционала F_Σ существует ННТ. Мы будем обозначать эту ННТ символом σ.

Будем говорить, что список функций $\vec{f}$ является НТ (ННТ) ФП Σ, если $\vec{f}$ является НТ (ННТ) функционала F_Σ.

Будем использовать следующие обозначения:

- если ФП Σ имеет вид (10), то компоненты её ННТ σ, а также соответствующие им ФС будут обозначаться записями $\sigma_1, \ldots, \sigma_n$ ($\forall\, i \in \{1, \ldots, n\}$ запись σ_i соответствует i-му уравнению в Σ);
- если ФП обозначается записью Σ^s, где s – некоторый символ, то её ННТ и компоненты этой ННТ (а также соответствующие им ФС) будут обозначаться записями $\sigma^s,\ \sigma^s_1,\ \ldots$;
- если Σ состоит из одного уравнения, то список σ состоит из одной функции, в этом случае символ σ обозначает также функцию, являющуюся единственной компонентой этого списка, и ФС, которому соответствует эта функция.

2.6 Нахождение наименьших неподвижных точек функциональных программ

ННТ ФП Σ можно найти, например, путём

- вычисления списков термов $\vec{e}_i$, соответствуюших $F^i_\Sigma(\mathbf{0})$ $(i = 0, 1, \ldots)$,
- нахождения по $\vec{e}_i$ списка термов $\vec{e}$, соответствущего $\sup_{i\geq 0} F^i_\Sigma(\mathbf{0})$ (который совпадает с σ по теореме 1).

Рассмотрим пример нахождения данным методом ННТ ФП:

$$\Sigma: \quad \left\{ \begin{array}{lll} \varphi(x) & = & (x > 100) \quad ? \, x - 10 \\ & & \qquad\qquad\ : \varphi\varphi(x + 11) \end{array} \right. \tag{21}$$

Нетрудно установить, что для $i = 1, \ldots, 11$

$$\begin{array}{lll} e_i & = & (x > 100) \quad ? \, x - 10 \\ & & \qquad\qquad\ : (x > 101 - i) \quad ? \, 91 : \omega \end{array}$$

а для $i \geq 11$

$$e_i \;=\; (x > 100) \quad ?\; x - 10 \\ \qquad\qquad\quad : (x > 90 - 11 \cdot (i - 11)) \quad ?\; 91 : \omega \tag{22}$$

При фиксированном x и достаточно больших i выражение

$$x > 90 - 11 \cdot (i - 11)$$

в терме (22) будет истинным, поэтому функция σ соответствует терму

$$(x > 100)?\; x - 10 : 91. \quad \blacksquare$$

2.7 Примеры неподвижных точек функциональных программ

1. ФП
$$\begin{cases} \varphi_1(x) = (x = 0)?\; 1 : \varphi_1(x - 1) + \varphi_2(x - 1) \\ \varphi_2(x) = (x = 0)?\; 0 : \varphi_2(x + 1) \end{cases}$$
имеет следующие НТ:
$$\begin{pmatrix} (x = 0 \vee x = 1)?\; 1 : n \cdot (x - 1) + 1 \\ (x = 0)?\; 0 : n \end{pmatrix},$$
где n – любой элемент $\tilde{D}_{\mathbf{N}}$. ННТ этой ФП $= \begin{pmatrix} (x = 0 \vee x = 1)?\; 1 : \omega \\ (x = 0)?\; 0 : \omega \end{pmatrix}$.

2. ФП
$$\begin{cases} \varphi_1(x) = (x > 100)?\; x - 10 : \varphi_1\varphi_2(x + 11) \\ \varphi_2(x) = (x > 100)?\; x - 10 : \varphi_2\varphi_1(x + 11) \end{cases}$$
имеет единственную НТ
$$\begin{pmatrix} (x > 100)?\; x - 10 : 91 \\ (x > 100)?\; x - 10 : 91 \end{pmatrix}.$$

3. ФП
$$\varphi(x, y) \;=\; (x = y) \quad ?\; y + 1 \\ \qquad\qquad\quad : \varphi(x, \varphi(x - 1, y + 1))$$
имеет, например, следующие НТ:

   (a) $(x = y)?\; y + 1 : x + 1$

   (b) $(x \geq y)?\; x + 1 : y - 1$

(c) $(x \geq y \ \wedge \ even(x-y))?\ x+1 : \omega$
(функция *even* принимает значение 1 на чётных числах и 0 на нечётных).

Последняя функция – ННТ этой ФП.

4. У ФП $\varphi(x) \ = \ \varphi(x)$ каждая функция является НТ, её ННТ = $\textcircled{\omega}$.

5. У ФП $\varphi(x) \ = \ \neg\varphi(x)$, где $\tau(\varphi)$ имеет вид $(\tau) \to \mathbf{B}$, есть единственная НТ – $\textcircled{\omega}$.

6. Каждая НТ ФП
$$\varphi(\vec{x}) \ = \ e_1?\ \varphi(\vec{x}) : e_2$$
имеет вид
$$e_1?\ e : e_2,$$
где e – любой терм типа $\tau(e_2)$, такой, что $Var(e) \subseteq \vec{x}$.
ННТ этой ФП = $e_1?\ \omega : e_2$.

7. Каждая НТ ФП
$$\varphi(x) \ = \ (x = 0)?\ 1 : \varphi(x+1)$$
имеет вид
$$(x = 0)?\ 1 : i,$$
где i – произвольный элемент $\tilde{D}_{\mathbf{N}}$.
ННТ этой ФП = $(x = 0)?\ 1 : \omega$.

2.8 Немонотонные функциональные программы

Немонотонные ФП (НФП) отличаются от обычных ФП тем, что НФП могут содержать ФС, которым соответствуют немонотонные функции.

У НФП могут отсутствовать НТ или ННТ, например:

1. НФП
$$\varphi(x) \ = \ (\varphi(x) \equiv 0)?\ 1 : 0$$
не имеет НТ.
Действительно, если бы у этой НФП была НТ f, то
   - из $f(0) = 0$ следовало бы, что $f(0) = 1$, и
   - из $f(0) \neq 0$ следовало бы, что $f(0) = 0$.

2. НФП

$$\varphi(x) \;=\; (\varphi(x) \equiv 0)?\; 0 : 1$$

имеет две НТ (константы 0 и 1), но не имеет ННТ.

Межфакультетский курс «Формальная семантика и верификация программного обеспечения» Лекция 6 Существование наименьшей неподвижной точки. Вычисление значения неподвижной точки на конкретных данных

А.М.Миронов

В этой главе мы рассматриваем задачу вычисления значений ННТ ФП на заданных значениях аргументов.

В целях простоты изложения мы рассматриваем в этой главе лишь ФП с одной функциональной переменной φ. Каждый терм e в этой главе удовлетворяет условию $FVar(e) \subseteq \{\varphi\}$.

1 Постановка задачи

Задача вычисления значений ННТ ФП заключается в построении алгоритма, который по заданной ФП Σ вида

$$\Sigma : \quad \varphi(\vec{x}) = e \tag{1}$$

и заданному списку $\vec{d} \in D_{\tau(\vec{x})}$ должен вычислить значение $\sigma(\vec{d})$.

Отметим, что для решения данной задачи не требуется нахождение терма, которому соответствует функция σ.

2 Метод решения

Один из возможных методов решения данной задачи заключается в том, что по заданным ФП Σ вида (1), и списку $\vec{d} \in D_{\tau(\vec{x})}$ строится последова-

тельность термов

$$C^0_{\Sigma,\vec{d}} \quad C^1_{\Sigma,\vec{d}} \quad C^2_{\Sigma,\vec{d}} \quad \ldots \tag{2}$$

называемая **вычислительной последовательностью**. Каждый её член является в некотором смысле аппроксимацией искомого значения $\sigma(\vec{d})$. Если вычислительная последовательность (2) конечна, то её последний элемент должен быть константой, значение которой равно $\sigma(\vec{d})$.

Последовательность (2) строится следующим образом.

1. Терм $C^0_{\Sigma,\vec{d}}$ имеет вид $\varphi(\vec{d})$.

2. Если терм $C^i_{\Sigma,\vec{d}}$ содержит функциональную переменную φ, то терм $C^{i+1}_{\Sigma,\vec{d}}$ получается из $C^i_{\Sigma,\vec{d}}$

   (a) заменой в нём некоторых подтермов вида

   $$\varphi(e_1, \ldots, e_n) \tag{3}$$

   на термы вида

   $$e^{(e_1/x_1,\ldots,e_n/x_n)}, \tag{4}$$

   где $(x_1, \ldots, x_n)$ – это список $\vec{x}$ в (1), и

   (b) упрощением получившегося терма.

3. Если терм $C^i_{\Sigma,\vec{d}}$ не содержит φ, то он является последним членом последовательности (2).

Ниже мы будем называть

- те подтермы вида (3) терма $C^i_{\Sigma,\vec{d}}$, которые выбираются для замены согласно пункту (2a), **раскрываемыми** подтермами, и
- замену (3) на (4) – **раскрытием** подтерма (3).

Если среди раскрываемых подтермов терма $C^i_{\Sigma,\vec{d}}$ одни подтермы содержатся в других, то раскрытие таких подтермов осуществляется по принципу «от меньших к большим», т.е. каждый раскрываемый подтерм раскрывается только после того, как будут раскрыты все содержащиеся в нём раскрываемые подтермы.

Преобразования термов, указанные в пунктах (2a) и (2b), объясняются более подробно ниже.

3 Вычислительные правила

Правило выбора раскрываемых подтермов терма $C^i_{\Sigma,\vec{d}}$ мы будем называть **вычислительным правилом**. Ниже будут рассматриваться следующие вычислительные правила.

1. **PO** (Parallel Outermost) – раскрываются все самые внешние подтермы вида (3) (т.е. не содержащиеся ни в каком подтерме вида (3)).

2. **LO** (Left Outermost) – раскрывается самый левый из самых внешних подтермов вида (3).

3. **PI** (Parallel Innermost) – раскрываются все самые внутренние подтермы вида (3) (т.е. не содержащие подтермов вида (3)).

4. **LI** (Left Innermost) – раскрывается самый левый из самых внутренних подтермов вида (3).

Мы будем считать, что символ C в (2) обозначает вычислительное правило, используемое при построении этой последовательности. Если это правило имеет специальное обозначение, то мы можем использовать это обозначение вместо символа C (т.е., например, если при построении термов вычислительной последовательности используется правило **PO**, то термы, входящие в данную последовательность, могут обозначаться записью $\mathbf{PO}^i_{\Sigma,\vec{d}}$).

4 Упрощение терма

Упрощение терма, о котором говорится в пункте (2b) описания построения вычислительной последовательности (2), представляет собой последовательность **упрощающих преобразований**.

Для определения понятия упрощающего преобразования введём следующие вспомогательные понятия.

1. Термы r и s называются **эквивалентными**, если

$$\forall f \in D_{\tau(\varphi)} \quad r^{(f/\varphi)}(\vec{x}) = s^{(f/\varphi)}(\vec{x}), \tag{5}$$

   где $\vec{x}$ состоит из переменных, входящих в r и s.

   Если r и s эквивалентны, то будем обозначать это записью $r \equiv s$.

2. Если терм v имеет вид

$$g(v_1, \ldots, v_n) \quad (g \in Fun) \tag{6}$$

то запись $\hat{v}$ обозначает терм

$$g(h_1, \ldots, h_n), \tag{7}$$

где $\forall\, i = 1, \ldots, n$,

$$h_i \stackrel{\text{def}}{=} \begin{cases} v_i, & \text{если } v_i \text{ – константа,} \\ x_i & \text{(переменная того же типа,} \\ & \text{что и } v_i) \text{ иначе,} \end{cases} \tag{8}$$

причём все переменные, входящие в (7), различны.

Мы будем говорить, что терм s получен из терма r **упрощающим преобразованием**, если s является результатом замены в r подтерма v вида (6) на терм v', который равен

- константе d, если $\hat{v} \equiv d$, или
- терму v_i, если h_i – переменная и $\hat{v} \equiv h_i$.

Нетрудно видеть, что при данной замене

- $v' \equiv v$, поэтому $s \equiv r$, и
- длина v' меньше длины v, поэтому длина s меньше длины r.

Терм r называется **неупрощаемым**, если не существует терма, который получается из r упрощающим преобразованием.

Терм s называется **упрощением** терма r, если s – неупрощаемый и либо $s = r$, либо существует последовательность r_1, …, r_n, такая, что

- $r_1 = r$, $r_n = s$, и
- $\forall\, i = 1, \ldots, n-1 \quad r_{i+1}$ получен из r_i упрощающим преобразованием.

Теорема 1
Для любого терма r существует терм s, являющийся его упрощением.

Доказательство.
Определим последовательность термов

$$r_1 \;\; r_2 \;\; \ldots \tag{9}$$

следующим образом: $r_1 \stackrel{\text{def}}{=} r$, и для каждого терма r_i в (9)

- если r_i неупрощаемый, то он – последний элемент в (9),
- иначе определяем r_{i+1} как результат какого-либо упрощающего преобразования терма r_i.

Последовательность (9) не может быть бесконечной, потому что длина каждого её терма r_i, где $i > 1$, меньше длины терма r_{i-1}.

Искомый терм s является последним элементом в (9). ■

Последовательность (9) с заданным первым элементом r может быть построена неоднозначно, т.к. при переходе от r_i к r_{i+1} возможно несколько вариантов выбора упрощающего преобразования.

Тем не менее, как устанавливает теорема 3, последний элемент последовательности (9) однозначно определяется её первым элементом.

Мы будем использовать следующие обозначения.

- Для любых термов r и s мы будем обозначать записью

$$r \to s \tag{10}$$

  тот факт, что $s = r$, или s получен из r упрощающим преобразованием.

- Если r – терм и s – вхождение некоторого подтерма в r, то для каждого терма s', тип которого равен типу терма s, запись $r^{(s'/s)}$ обозначает терм, получаемый из r заменой вхождения s на терм s'.

Теорема 2

Если

$$r \to s_1 \quad \text{и} \quad r \to s_2, \tag{11}$$

то

$$\exists\, s: \quad s_1 \to s \quad \text{и} \quad s_2 \to s \tag{12}$$

(в литературе по теории программирования данное свойство бинарного отношения называют **свойством ромба**, или **свойством Чёрча–Россера**, или **конфлюентностью**).

Доказательство.

Если $s_1 = r$, то $s \stackrel{\text{def}}{=} s_2$, и если $s_2 = r$, то $s \stackrel{\text{def}}{=} s_1$.

Если $s_1 = r^{(u_1'/u_1)}$ и $s_2 = r^{(u_2'/u_2)}$, то возможны следующие случаи.

1. Вхождения u_1 и u_2 в терм r совпадают.

   В этом случае $u_1' = u_2'$, т.к. для $i = 1, 2$ варианты

- $\hat{u}_i \equiv d$, где d – константа, и
- $\hat{u}_i \equiv h_j$, где u_i имеет вид (6), и h_j – переменная, определённая согласно (8),

являются взаимоисключающими, и если имеет место второй вариант, то номер j определён однозначно. Доказательство этого факта опирается на то, что все пополненные домены состоят более чем из одного элемента.

2. $u_1 \subset u_2$.

   Пусть u_2 и $\hat{u}_2$ имеют вид

   $$g(v_1, \ldots, v_n) \quad \text{и} \quad g(h_1, \ldots, h_n) \tag{13}$$

   соответственно.

   Из $u_1 \subset u_2$ следует, что $u_1 \subseteq v_i$, где v_i – один из подтермов терма u_2, упомянутых в (13).

   Обозначим символом y подтерм $u_2^{(u_1'/u_1)}$ терма s_1. Нетрудно видеть, что $s_1 = r^{(y/u_2)}$.

   Термы y и $\hat{y}$ отличаются от термов (13) не более чем в i-й компоненте списка, идущего сразу после ФС g. Мы обозначим эти i-е компоненты для y и $\hat{y}$ записями v_i^y и h_i^y соответственно.

   Согласно определению правил упрощения, возможен один из следующих вариантов:

   (a) $\hat{u}_2 \equiv d$, где d – константа.
   В этом случае $\hat{y} \equiv d$ и $s \stackrel{\text{def}}{=} s_2 = s_1^{(d/y)}$;

   (b) $\hat{u}_2 \equiv h_j$, где h_j – переменная и $j \neq i$.
   В этом случае $\hat{y} \equiv h_j$ и $u_2' = y' = v_j$.
   В качестве s можно взять s_2.
   Поскольку $s_1 = r^{(y/u_2)}$ и $s = r^{(v_j/u_2)}$, то $s = s_1^{(v_j/y)}$;

   (c) $\hat{u}_2 \equiv h_i$, где h_i – переменная.
   В этом случае $u_2' = v_i$ и $s_2 = r^{(v_i/u_2)}$.
   Из $\hat{u}_2 \equiv h_i$ следует, что $\hat{y} \equiv h_i^y$.

      i. Если $h_i^y \in Var$, то в качестве s можно взять $s_1^{(v_i^y/y)}$.
      $s_2 \to s$ следует из равенства $s = s_2^{(u_1'/u_1)}$. По предположению, $u_1 \subseteq v_i$, и после замены u_1 на u_1' подтерм v_i терма s_2 преобразуется в терм v_i^y, в результате чего получается s.

ii. Если $h_i^y = d \in Con$, то $v_i^y = d$ и $\hat{y} \equiv d$.
В данном случае $s \stackrel{\text{def}}{=} s_1^{(d/y)}$.
$s_2 \to s$ следует из равенства $s = s_2^{(d/v_i)}$, которое верно потому, что терм v_i^y равен константе d и равен терму $v_i^{(u_1'/u_1)}$, что может быть только в том случае, когда

$$v_i = u_1, \quad u_1' = d \quad \text{и} \quad \hat{v}_i \equiv d.$$

3. $u_2 \subset u_1$. Данный случай аналогичен предыдущему.

4. Вхождения термов u_1 и u_2 не пересекаются.

В этом случае $u_2 \subseteq s_1,\ u_1 \subseteq s_2$ и $s \stackrel{\text{def}}{=} s_1^{(u_2'/u_2)} = s_2^{(u_1'/u_1)}$. ■

Теорема 3
Если s_1 и s_2 – упрощения терма r, то $s_1 = s_2$.

Доказательство.
Если r неупрощаемый, то $r = s_1 = s_2$.
Если r упрощаемый, то существуют последовательности термов

$$\begin{array}{l} r = s_{11},\ \ s_{12},\ \ \ldots,\ \ s_{1n} = s_1 \\ r = s_{11},\ \ s_{21},\ \ \ldots,\ \ s_{m1} = s_2 \end{array}$$

такие, что имеют место соотношения

$$\begin{array}{ccccccc} s_{11} & \to & s_{12} & \to & \ldots & \to & s_{1n} \\ \downarrow & & & & & & \\ s_{21} & & & & & & \\ \downarrow & & & & & & \\ \ldots & & & & & & \\ \downarrow & & & & & & \\ s_{m1} & & & & & & \end{array} \tag{14}$$

(мы можем рисовать стрелки, изображающие отношение (10), не только горизонтально, но и вертикально).

Диаграмму (14) можно достроить до диаграммы

$$\begin{array}{ccccccc} s_{11} & \to & s_{12} & \to & \ldots & \to & s_{1n} \\ \downarrow & & \downarrow & & & & \downarrow \\ s_{21} & \to & s_{22} & \to & \ldots & \to & s_{2n} \\ \downarrow & & \downarrow & & & & \downarrow \\ \ldots & & \ldots & & & & \ldots \\ \downarrow & & \downarrow & & & & \downarrow \\ s_{m1} & \to & s_{m2} & \to & \ldots & \to & s_{mn} \end{array} \tag{15}$$

где термы s_{22}, ..., s_{mn} определяются индуктивно: если для некоторых i из $\{1, \ldots, m-1\}$ и j из $\{1, \ldots, n-1\}$ уже определены термы s_{ij}, $s_{i,j+1}$, $s_{i+1,j}$, удовлетворяющие условиям

$$\begin{array}{ccc} s_{ij} & \to & s_{i,j+1} \\ \downarrow & & \\ s_{i+1,j} & & \end{array}$$

то терм $s_{i+1,j+1}$ определяется как терм, удовлетворяющий условиям

$$\begin{array}{ccc} & & s_{i,j+1} \\ & & \downarrow \\ s_{i+1,j} & \to & s_{i+1,j+1} \end{array}$$

(существование такого терма гарантируется свойством ромба).

Поскольку термы $s_1 = s_{1n}$ и $s_2 = s_{m1}$ неупрощаемы, то все термы в последней строке и в последнем столбце диаграммы (15) совпадают. В частности,

$$s_1 = s_{1n} = s_{mn} = s_{m1} = s_2. \quad \blacksquare$$

Таким образом, для каждого терма r существует единственный терм, являющийся его упрощением.

5 Функция C_Σ

Пусть заданы вычислительное правило C и ФП Σ вида

$$\Sigma: \quad \varphi(\vec{x}) = e. \tag{16}$$

Мы будем обозначать записью C_Σ функцию из $D_{\tau(\varphi)}$, сопоставляющую каждому $\vec{d} \in D_{\tau(\vec{x})}$ значение $C_\Sigma(\vec{d})$, которое называется **результатом вычисления** ФП Σ на списке $\vec{d}$ и определяется следующим образом: по тройке $(C, \Sigma, \vec{d})$ строится вычислительная последовательность (2). Если она конечна, то $C_\Sigma(\vec{d})$ равно значению последнего терма в этой последовательности (который, как нетрудно видеть, является константой). Если эта последовательность бесконечна, то $C_\Sigma(\vec{d}) \stackrel{\text{def}}{=} \omega$.

Теорема 4
Для любого $\vec{d} \in D_{\tau(\vec{x})}$ верно неравенство $C_\Sigma(\vec{d}) \leq \sigma(\vec{d})$.

Доказательство.
Если последовательность (2) бесконечна, то $C_\Sigma(\vec{d}) = \omega \leq \sigma(\vec{d})$.

Пусть последовательность (2) конечна.

Все термы в последовательности

$$(C^0_{\Sigma,\vec{d}})^{(\sigma/\varphi)} \quad (C^1_{\Sigma,\vec{d}})^{(\sigma/\varphi)} \quad \ldots \tag{17}$$

не содержат переменных, поэтому каждый из этих термов определяет некоторое значение. Докажем, что все эти значения совпадают.

Последовательность (2) является подпоследовательностью последовательности термов, в которой каждый терм, кроме первого, получается из предыдущего раскрытием некоторого подтерма, или упрощающим преобразованием. Поэтому совпадение значений всех термов в (17) является следствием следующих двух утверждений.

1. Если термы s и r не содержат переменных и s получен из r заменой подтерма
$$\varphi(e_1, \ldots, e_n) \tag{18}$$
на терм
$$e^{(e_1/x_1,\ldots,e_n/x_n)}, \tag{19}$$
то значения термов
$$s^{(\sigma/\varphi)} \quad \text{и} \quad r^{(\sigma/\varphi)} \tag{20}$$
совпадают.

2. Если терм s получен из терма r упрощающим преобразованием, то $s \equiv r$ (в частности, в том случае, когда s и r не содержат переменных, значения термов (20) совпадают).

Утверждение 2 верно потому, что при упрощающем преобразовании происходит замена подтерма на терм, эквивалентный заменяемому.

Обоснуем утверждение 1.

Если s получен из r заменой подтерма (18) на терм (19), то $s^{(\sigma/\varphi)}$ получен из $r^{(\sigma/\varphi)}$ заменой подтерма

$$\sigma(e_1^{(\sigma/\varphi)}, \ldots, e_n^{(\sigma/\varphi)}) \tag{21}$$

на терм

$$(e^{(\sigma/\varphi)})^{(e_1^{(\sigma/\varphi)}/x_1,\ldots,e_n^{(\sigma/\varphi)}/x_n)}. \tag{22}$$

Функция σ является НТ функционала, сопоставляющего каждой функции $f \in D_{\tau(\varphi)}$ функцию $e^{(f/\varphi)}(\vec{x})$, т.е. для каждого списка

$$(d_1, \ldots, d_n) \in D_{\tau(\vec{x})}$$

верно равенство

$$\sigma(d_1, \ldots, d_n) = (e^{(\sigma/\varphi)})^{(d_1/x_1, \ldots, d_n/x_n)}. \qquad (23)$$

Полагая в (23)

$$\forall i = 1, \ldots, n \quad d_i \stackrel{\text{def}}{=} e_i^{(\sigma/\varphi)},$$

получаем, что значения термов (21) и (22) совпадают.

Следовательно, терм $s^{(\sigma/\varphi)}$ получается из терма $r^{(\sigma/\varphi)}$ заменой его подтерма на терм, значение которого равно значению заменяемого подтерма. Таким образом, значения термов (20) совпадают, т.е. утверждение 1 обосновано. Из утверждений 1 и 2 следует, что значения всех термов в (17) совпадают.

Поскольку

- значение первого терма в (17) равно $\sigma(\vec{d})$ и
- значение последнего терма в (17) равно $C_\Sigma(\vec{d})$,

то мы заключаем, что если вычислительная последовательность (2) конечна, то верно равенство $C_\Sigma(\vec{d}) = \sigma(\vec{d})$. ■

6 Вспомогательные понятия

В этом параграфе мы определяем понятия и доказываем теоремы, которые будут использоваться в параграфе 7 при доказательстве теоремы 9 о свойстве безопасных вычислительных правил.

6.1 Полные раскрытия

Пусть задана ФП $\Sigma: \quad \varphi(\vec{x}) = e$.

Определим последовательность термов $e^{(0)}, e^{(1)}, \ldots$ (называемую **полным раскрытием** для ФП Σ) следующим образом:

- $e^{(0)} = \varphi(\vec{x})$ и
- $\forall\, i \geq 0$, терм $e^{(i+1)}$ получается из $e^{(i)}$ раскрытием каждого подтерма вида $\varphi(\ldots)$, причём эти раскрытия выполняются по принципу «от меньших подтермов к большим», т.е. каждый подтерм вида $\varphi(\ldots)$ раскрывается после того, как будут раскрыты все содержащиеся в нём подтермы вида $\varphi(\ldots)$.

  (Эти раскрытия в $e^{(i)}$ можно выполнять в порядке «справа налево»: сначала раскрывается самый правый подтерм вида $\varphi(\ldots)$, затем

самый правый из подтермов вида $\varphi(\ldots)$, начало которого расположено левее начала первого раскрытого подтерма, и т.д.)

Теорема 5
Для любого $i \geq 0$ и любой функции $f \in D_{\tau(\varphi)}$

$$(e^{(i)})^{(f/\varphi)}(\vec{x}) = F_\Sigma^i(f). \tag{24}$$

Доказательство.
Докажем (24) индукцией по i.
При $i = 0$ обе части (24) равны функции f.
Пусть (24) верно для некоторого $i \geq 0$ и произвольной функции f.
Из определения терма $e^{(i+1)}$ следует, что терм $(e^{(i+1)})^{(f/\varphi)}$ получается из $(e^{(i)})^{(f/\varphi)}$ заменой каждого подтерма вида

$$f(e_1, \ldots, e_n) \tag{25}$$

на терм

$$(e^{(f/\varphi)})^{(e_1/x_1,\ldots,e_n/x_n)}, \tag{26}$$

причём данные замены выполняются по принципу «от меньших подтермов к большим». Из определения функционала F_Σ следует, что терм (26) эквивалентен терму

$$F_\Sigma(f)(e_1, \ldots, e_n), \tag{27}$$

поэтому терм $(e^{(i+1)})^{(f/\varphi)}$ эквивалентен терму, получаемому из терма $(e^{(i)})^{(f/\varphi)}$ заменой каждого подтерма вида (25) на терм (27), т.е. заменой каждого вхождения f на $F_\Sigma(f)$.

Таким образом, верно соотношение $(e^{(i+1)})^{(f/\varphi)} \equiv (e^{(i)})^{F_\Sigma(f)/\varphi)}$, откуда следует равенство функций

$$(e^{(i+1)})^{(f/\varphi)}(\vec{x}) = (e^{(i)})^{(F_\Sigma(f)/\varphi)}(\vec{x}). \tag{28}$$

Поскольку мы предполагаем, что (24) верно для каждой функции f, то, в частности, правая часть в (28) равна

$$F_\Sigma^i(F_\Sigma(f)) = F_\Sigma^{i+1}(f). \quad \blacksquare$$

6.2 Индексированные термы

Индексированным термом (ИТ) называется терм r, такой, что каждому вхождению функциональной переменной φ в r сопоставлено натуральное число, называемое **глубиной** этого вхождения, причем выполнено условие **монотонности**: если u – подтерм терма r, имеющий вид $\varphi(\ldots)$, то глубина первого вхождения φ в u не превосходит глубины остальных вхождений φ в u.

Для каждого ИТ r и каждого $j \geq 0$

- $in(r, j)$ обозначает число вхождений в r символа φ глубины j и
- $in(r)$ обозначает последовательность $\{in(r, i) \mid i \geq 0\}$.

6.3 Σ-переходы

Пусть задана ФП $\Sigma: \quad \varphi(\vec{x}) = e, \;$ где $\vec{x} = (x_1, \ldots)$.

Σ-переходом называется пара ИТ (r, s), удовлетворяющая одному из следующих условий.

1. $r \to s$, и глубина каждого вхождения φ в s равна глубине соответствующего вхождения φ в r
   (поскольку в данном случае $s = r$ или s получается из r удалением некоторых символов или заменой их на константу, то каждому вхождению φ в s соответствует некоторое вхождение φ в r).

2. $s = r^{(u'/u)}$, где $u = \varphi(e_1, \ldots), \quad u' = e^{(e_1/x_1, \ldots)}$, причем для каждого вхождения φ в s, содержащегося в u',

   - если это вхождение содержится в подтерме e_i, подставленном вместо переменной x_i в e, то глубина этого вхождения на 1 больше глубины соответствующего вхождения φ в u,
   - иначе глубина этого вхождения на 1 больше глубины первого вхождения φ в u,

   и для каждого вхождения φ в s, не содержащегося в u', его глубина равна глубине соответствующего вхождения φ в r.

Если пара (r, s) является Σ-переходом, то этот факт обозначается записью $r \stackrel{\Sigma}{\to} s$. Σ-переход (r, s) называется **упрощением** или **раскрытием**, если s получается из r в результате упрощающего преобразования или раскрытия соответственно.

Для любых ИТ r и s запись $r \stackrel{\Sigma*}{\to} s$ означает, что существует последовательность термов r_0, …, r_n, обладающая свойствами

$$r = r_0 \stackrel{\Sigma}{\to} \ldots \stackrel{\Sigma}{\to} r_n = s. \tag{29}$$

В излагаемых ниже теоремах используется следующее отношение порядка на множестве бесконечных последовательностей натуральных чисел: если $\vec{a} = \{a_i \mid i \geq 0\}$ и $\vec{b} = \{b_i \mid i \geq 0\}$ – две такие последовательности, то

$$\vec{a} \leq \vec{b} \;\Leftrightarrow\; \left\{ \begin{array}{ll} \vec{a} = \vec{b}, & \text{или} \\ \exists\, i \geq 0 & \left\{ \begin{array}{ll} a_i < b_i \\ \forall\, j:\ 0 \leq j < i & a_j = b_j \end{array} \right. \end{array} \right.$$

(такой порядок называется **лексикографическим**).

Теорема 6
Если $r \overset{\Sigma*}{\to} s$, то $in(r) \geq in(s)$.

Доказательство.
Достаточно рассмотреть случай $r \overset{\Sigma}{\to} s$.

1. s получается из r упрощающим преобразованием, т.е. удалением некоторых символов (или заменой их на константу). Поскольку глубина каждого вхождения φ в s совпадает с глубиной соответствующего вхождения φ в r, то

$$\forall\, j \geq 0 \quad in(r, j) \geq in(s, j),$$

   откуда следует $in(r) \geq in(s)$.

2. $s = r^{(u'/u)}$, где $u = \varphi(e_1, \ldots)$, $u' = e^{(e_1/x_1,\ldots)}$.

   Пусть первое вхождение φ в u имеет глубину k.

   В этом случае

   - глубина вхождений φ в u больше или равна k,
   - для каждого $j \geq 0$ число вхождений φ глубины j в s, находящихся вне u', будет то же, что и число вхождений φ глубины j в r, находящихся вне u,
   - в подтерме u' терма s нет вхождений φ глубины k и
   - для каждого $j < k$ в подтерме u терма r вхождения φ глубины j отсутствуют, и в подтерме u' терма s их тоже нет.

   Следовательно, верны соотношения

   - $\forall\, j : 0 \leq j < k \quad in(r, j) = in(s, j)$ и
   - $in(r, k) > in(s, k)$,

   из которых следует $in(r) \geq in(s)$. ■

Пусть заданы ФП $\Sigma :\ \varphi(\vec{x}) = e$, вычислительное правило C и список $\vec{d} \in D_{\tau(\vec{x})}$. Термы вычислительной последовательности $\{C^i_{\Sigma,\vec{d}} \mid i \geq 0\}$ можно рассматривать как ИТ, такие, что каждая пара соседних термов удовлетворяет соотношению

$$C^i_{\Sigma,\vec{d}} \overset{\Sigma*}{\to} C^{i+1}_{\Sigma,\vec{d}}. \tag{30}$$

Будем считать, что глубина вхождения φ в $C^0_{\Sigma,\vec{d}} = \varphi(\vec{d})$ равна 0.

Теорема 7

Если вычислительная последовательность $\{C^i_{\Sigma,\vec{d}} \mid i \geq 0\}$ бесконечна, то $\forall\, M \geq 0 \quad \exists\, N \geq 0$:

$$\forall\, j = 0, \ldots, M \qquad in(C^N_{\Sigma,\vec{d}}, j) = in(C^{N+1}_{\Sigma,\vec{d}}, j) = \ldots \tag{31}$$

Доказательство.

Индукция по M.

1. Если $M = 0$, то $N = 1$.

2. Пусть для некоторого $M \geq 0$ существует номер N, для которого верно (31). Тогда из теоремы 6 и из определения лексикографического порядка следует, что

$$in(C^N_{\Sigma,\vec{d}}, M+1) \geq in(C^{N+1}_{\Sigma,\vec{d}}, M+1) \geq \ldots \tag{32}$$

Цепочка (32) не может бесконечно убывать, т.е. $\exists\, N' \geq N$:

$$in(C^{N'}_{\Sigma,\vec{d}}, M+1) = in(C^{N'+1}_{\Sigma,\vec{d}}, M+1) = \ldots \tag{33}$$

Из (31) и из $N' \geq N$ следует, что для каждого $j = 0, \ldots, M$ цепочка (33), в которой второй аргумент функции in будет заменён на j, также будет верной. Таким образом, для $M+1$ в качестве искомого номера можно взять N'. ■

Теорема 8

Если вычислительная последовательность $\{C^i_{\Sigma,\vec{d}} \mid i \geq 0\}$ бесконечна, то $\forall\, M \geq 0 \quad \exists\, N$: в каждом раскрываемом подтерме терма $C^N_{\Sigma,\vec{d}}$ первое вхождение φ имеет глубину $> M$.

Доказательство.

Выберем N таким, чтобы было верно (31).

Если существует раскрываемый подтерм терма $C^N_{\Sigma,\vec{d}}$, в котором первое вхождение φ имеет глубину $j \leq M$, то $in(C^N_{\Sigma,\vec{d}}, j) > in(C^{N+1}_{\Sigma,\vec{d}}, j)$, что противоречит (31). ■

7 Безопасные вычислительные правила

Пусть заданы ФП $\Sigma:\ \varphi(\vec{x}) = e$ и вычислительное правило C.

Правило C называется **безопасным** для Σ, если $\forall\, \vec{d} \in D_{\tau(\vec{x})}$, если последовательность $\{C^i_{\Sigma,\vec{d}} \mid i \geq 0\}$ бесконечна, то $\forall\, i \geq 0$

$$(C^{i,\omega}_{\Sigma,\vec{d}})^{(\sigma/\varphi)} = \omega, \tag{34}$$

где $C^{i,\omega}_{\Sigma,\vec{d}}$ получается из $C^i_{\Sigma,\vec{d}}$ заменой каждого самого внешнего раскрываемого подтерма (т.е. не содержащегося в другом раскрываемом подтерме) на константу ω.

Теорема 9

Если C безопасно для Σ, то $\forall\, \vec{d} \in D_{\tau(\vec{x})}$

$$C_\Sigma(\vec{d}) = \sigma(\vec{d}). \tag{35}$$

Доказательство.

Как было установлено в теореме 4, равенство (35) может нарушаться только в том случае, когда

$$\omega < \sigma(\vec{d}) \tag{36}$$

и вычислительная последовательность $\{C^i_{\Sigma,\vec{d}} \mid i \geq 0\}$ бесконечна.

Поскольку $\sigma = \sup_{i\geq 0} F^i_\Sigma(\textcircled{\omega})$, то $\exists\, n$:

$$\sigma(\vec{d}) = F^n_\Sigma(\textcircled{\omega})(\vec{d}) = (e^{(n)})^{(\textcircled{\omega}/\varphi)}(\vec{d}), \tag{37}$$

где $e^{(n)}$ – соответствующий терм из полного раскрытия для Σ (второе равенство верно согласно теореме 5).

Из определения полного раскрытия следует, что существуют ИТ r_0, $\ldots$, r_k, такие, что

$$\varphi(\vec{d}) = e^{(0)}(\vec{d}) = r_0 \overset{\Sigma}{\to} \ldots \overset{\Sigma}{\to} r_k = e^{(n)}(\vec{d}), \tag{38}$$

причём для каждого $i = 0, \ldots, k-1$ терм r_{i+1} получается из r_i раскрытием некоторого подтерма.

Из (36), (37) и определения r_k следует, что

$$\omega < r_k^{(\textcircled{\omega}/\varphi)}. \tag{39}$$

Обозначим символом M максимальную глубину вхождения φ в r_k.

По теореме 8, $\exists\, N$: в каждом раскрываемом подтерме терма $C^N_{\Sigma,\vec{d}}$ первое вхождение φ имеет глубину $> M$.

Поскольку для каждого $i = 0, \ldots, N-1$ верно (30), то существует последовательность ИТ s_0, …, s_l, такая, что

$$\varphi(\vec{d}) = s_0 \stackrel{\Sigma}{\to} \ldots \stackrel{\Sigma}{\to} s_l = C^N_{\Sigma,\vec{d}}. \tag{40}$$

Если в (40) есть пара соседних Σ-переходов, первый из которых – упрощение, а следующий за ним – раскрытие, то мы преобразуем (40) в соответствии со следующим правилом. Пусть i – наименьший номер, такой, что (40) содержит пару

$$s_i \stackrel{\Sigma}{\to} s_{i+1} = s_i^{(u'/u)} \stackrel{\Sigma}{\to} s_{i+2} = s_{i+1}^{(v'/v)}, \tag{41}$$

где $u = g(u_1, \ldots)$, $g \in Fun$ и $v = \varphi(v_1, \ldots)$.

1. Если вхождения u' и v в s_{i+1} не пересекаются, или $v \subseteq u'$, то можно считать, что $v \subseteq s_i$.

   Заменим в (40) пару (41) на пару Σ-переходов

   $$s_i \stackrel{\Sigma}{\to} s_i^{(v'/v)} \stackrel{\Sigma}{\to} s_{i+2}.$$

2. Если $u' \subset v = \varphi(v_1, \ldots)$, то $u' \subseteq v_j$ для некоторого j.

   В этом случае $\exists\, w = \varphi(w_1, \ldots) \subseteq s_i : u \subseteq w_j$.

   Заменим в (40) пару (41) на последовательность

   $$s_i \stackrel{\Sigma}{\to} s_{i1} \stackrel{\text{def}}{=} s_i^{(e^{(w_1/x_1,\ldots)}/w)} \stackrel{\Sigma}{\to} s_{i2} \stackrel{\Sigma}{\to} \ldots \stackrel{\Sigma}{\to} s_{ip} = s_{i+2},$$

   где $\forall\, q = 1, \ldots, p-1 \quad s_{i(q+1)}$ получается из s_{iq} заменой u на u' в одном из вхождений w_j.

Обозначим получившуюся последовательность тем же знакосочетанием (40), что и исходную последовательность. Если получившаяся последовательность будет содержать пару соседних Σ-переходов, первый из которых – упрощение, а следующий за ним – раскрытие, то опять преобразуем эту последовательность в соответствии с описанным выше правилом, и т.д. Будем выполнять такие преобразования до тех пор, пока не получится такая последовательность (40), в которой сначала идут раскрытия, а затем упрощения. Нетрудно видеть, что данный процесс завершается после конечного числа шагов.

Пусть s_m – тот член последовательности (40), на котором заканчиваются раскрытия и начинаются упрощения, т.е. все Σ-переходы в подпоследовательности

$$\varphi(\vec{d}) = s_0 \stackrel{\Sigma}{\to} \ldots \stackrel{\Sigma}{\to} s_m \tag{42}$$

являются раскрытиями и все Σ-переходы в подпоследовательности

$$s_m \stackrel{\Sigma}{\to} \ldots \stackrel{\Sigma}{\to} s_l = C^N_{\Sigma,\vec{d}} \tag{43}$$

являются упрощениями.

Для каждого терма s и каждого $i \geq 0$ обозначим $\textcircled{\omega}_i(s)$ терм, получаемый из s заменой каждого вхождения φ глубины $> i$ на $\textcircled{\omega}$.

Нетрудно видеть, что

$$\textcircled{\omega}_M(s_m)^{(\sigma/\varphi)} = \omega \tag{44}$$

(это следует из того, что терм $(C^{N,\omega}_{\Sigma,\vec{d}})^{(\sigma/\varphi)}$, значение которого равно ω, можно получить из левой части (44) упрощениями и заменами некоторых вхождений ω на термы вида $\sigma(\ldots)$).

Если s_m содержит подтерм u вида $\varphi(u_1, \ldots)$, в котором первое вхождение φ имеет глубину $\leq M$, то добавим к (42) Σ-переход

$$s_m \stackrel{\Sigma}{\to} s_m^{(u'/u)}, \quad \text{где} \;\; u' = e^{(u_1/x_1,\ldots)}. \tag{45}$$

Из (44) следует равенство $\textcircled{\omega}_M(s_m^{(u'/u)})^{(\sigma/\varphi)} = \omega$, которое обосновывается соотношениями

$$\textcircled{\omega}_M(u')^{(\sigma/\varphi)} \leq \textcircled{\omega}_{M+1}(u')^{(\sigma/\varphi)} = \textcircled{\omega}_M(u)^{(\sigma/\varphi)}.$$

Обозначим последовательность Σ-переходов, полученную добавлением (45) к (42), той же записью, что и исходную последовательность (42). Будем выполнять описанную выше операцию добавления Σ-переходов к (42) до тех пор, пока последний терм s_m в получившейся последовательности не перестанет содержать вхождения φ глубины $\leq M$. Из вышесказанного следует, что для этого терма верно равенство (44), из которого следует равенство

$$s_m^{(\textcircled{\omega}/\varphi)} = \omega. \tag{46}$$

Преобразуем последовательности (38) и (42) в такие последовательности Σ-переходов, в которых каждая пара соседних Σ-переходов удовлетворяет следующему условию: пусть данная пара имеет вид

$$a \stackrel{\Sigma}{\to} b = a^{(u'/u)} \stackrel{\Sigma}{\to} c = b^{(v'/v)}, \tag{47}$$

тогда номер позиции в терме a, в которой расположен первый символ подтерма u, меньше номера позиции в терме b, в которой расположен первый символ подтерма v.

Предположим, что данное условие не выполнено для некоторой пары соседних переходов (47). Тогда возможен один из двух случаев.

1. Вхождения u' и v в терм b не пересекаются.

   В этом случае заменим (47) на пару Σ-переходов

   $$a \overset{\Sigma}{\to} b' \overset{\text{def}}{=} a^{(v'/v)} \overset{\Sigma}{\to} b'^{(u'/u)} = c.$$

2. $\exists\, w = \varphi(w_1, \ldots) \subseteq a :\ u \subseteq w_j$ для некоторого j, $v = w^{(u'/u)}$.

   В этом случае заменим пару (47) на последовательность Σ-переходов

   $$a \overset{\Sigma}{\to} b_1 \overset{\text{def}}{=} a^{(e^{(w_1/x_1,\ldots)}/w)} \overset{\Sigma}{\to} b_2 \overset{\Sigma}{\to} \ldots \overset{\Sigma}{\to} b_p = c,$$

   где $\forall\, q = 1, \ldots, p-1 \quad b_{q+1}$ получается из b_q заменой u на u' в одном из вхождений w_i.

Нетрудно убедиться, что после конечного числа преобразований данного типа последовательности (38) и (42) преобразуются в такие последовательности Σ-переходов, которые будут удовлетворять изложенному выше условию. Будем обозначать результаты данных преобразований теми же записями, что и исходные последовательности (38) и (42). Таким образом, последовательности (38) и (42) обладают следующими свойствами:

- все Σ-переходы в них являются раскрытиями, причем в каждом из этих Σ-переходов (кроме последних) раскрываемый подтерм расположен левее раскрываемого подтерма в следующем за ним Σ-переходе;
- (38) имеет вид $\varphi(\vec{d}) = r_0 \overset{\Sigma}{\to} \ldots \overset{\Sigma}{\to} r_k$, и все вхождения φ в r_k имеют глубину $\leq M$;
- (42) имеет вид $\varphi(\vec{d}) = s_0 \overset{\Sigma}{\to} \ldots \overset{\Sigma}{\to} s_m$, и все вхождения φ в s_m имеют глубину $> M$.

Определим бинарное отношение R на множестве термов как множество пар вида (r_i, s_j), где $r_i \in (38)$, $s_j \in (42)$, обладающих следующими свойствами:

- r_i можно представить в виде конкатенации

$$a_1\ u_1\ \ldots\ a_n\ u_n\ a, \tag{48}$$

где $n \geq 0$ (т.е. компоненты a_1, …, u_n могут отсутствовать) и

  - $u_1, \ldots, u_n$ – вхождения подтермов, каждый из которых имеет вид $\varphi(\ldots)$,
  - $a_1, \ldots, a_n, a$ – символьные строки,
  - если $i < k$, то раскрываемый подтерм терма r_i содержится в подстроке a;

- s_j можно представить в виде конкатенации

$$a_1\ v_1\ \ldots\ a_n\ v_n\ a, \tag{49}$$

где $n \geq 0$ и

  - $v_1, \ldots, v_n$ – вхождения подтермов,
  - $a_1, \ldots, a_n, a$ – те же символьные строки, что и в (48),
  - если $j < m$, то раскрываемый подтерм терма s_j содержится в подстроке a.

Отметим, что

$$(s_0, r_0) \in R \tag{50}$$

(в данном случае $n = 0$ и $a = \varphi(\vec{d})$).

Если $(r_i, s_j) \in R$ и $i < k$, то $j < m$, т.к. если $j = m$, то терм s_m содержит подтерм вида $\varphi(\ldots)$ терма r_i (в подстроке a) и

- все вхождения φ в этот подтерм имеют глубину $\leq M$, но
- все вхождения φ в s_m имеют глубину $> M$.

Аналогично если $(r_i, s_j) \in R$ и $j < m$, то $i < k$.

Докажем, что если $(r_i, s_j) \in R$ и $i < k$, то $\exists\, i' \geq i$ и $\exists\, j' \geq j$: $(i', j') \neq (i, j)$ и $(r_{i'}, s_{j'}) \in R$. Отметим, что на основании (50) отсюда будет следовать $(r_k, s_m) \in R$.

Обозначим символами α и β номера позиций в подстроке a, в которых расположены первые символы раскрываемых подтермов u и v термов r_i и s_j соответственно.

Рассмотрим три случая.

1. $\alpha = \beta$. В данном случае $u = v$, $i' = i + 1$, $j' = j + 1$.

2. $\alpha < \beta$. Этот случай невозможен, т.к. если s_j в позиции α подстроки a содержит символ φ глубины $\leq M$ и раскрываемый подтерм терма s_j расположен правее этого вхождения φ, то поскольку в каждом из Σ-переходов в (42) (кроме первого) раскрываемый подтерм расположен правее раскрываемого подтерма в предыдущем Σ-переходе, то

   - символ φ глубины $\leq M$ будет входить в s_{j+1}, …, s_m,
   - но все вхождения φ в s_m имеют глубину $> M$.

3. $\alpha > \beta$. Пусть r_i и s_j можно представить в виде конкатенаций (48) и (49) соответственно, компоненты которых обладают указанными выше свойствами. Представим подстроку a в виде конкатенации $b\ v\ c$, где v – раскрываемый подтерм в s_j.

   Нетрудно доказать, что $\exists\, i' \geq i$, $\exists\, j' \geq j$: $(i', j') \neq (i, j)$, и $r_{i'}$ и $s_{j'}$ можно представить в виде конкатенаций

$$\begin{aligned} r_{i'} &= a_1\ u_1\ \ldots\ a_n\ u_n\ b\ w\ c \\ s_{j'} &= a_1\ v_1\ \ldots\ a_n\ v_n\ b\ w'\ c \end{aligned}$$

   соответственно, где w и w' – термы, w имеет вид $\varphi(\ldots)$ и

   - либо $i' = k$ и $j' = m$,
   - либо раскрываемые термы в $r_{i'}$ и $s_{j'}$ содержатся в c.

Таким образом, во всех случаях существует пара $(r_{i'}, s_{j'}) \in R$ с требуемыми свойствами.

Как было отмечено выше, из доказанного следует, что $(r_k, s_m) \in R$, т.е. r_k и s_m можно представить в виде конкатенаций (48) и (49) соответственно, компоненты которых обладают указанными выше свойствами.

Обозначим записями a_1^ω, u_1^ω, …, v_n^ω, a^ω результаты замены в соответствующих компонентах конкатенаций (48) и (49) функциональной переменной φ на ФС $\textcircled{\omega}$. Поскольку термы u_1, …, u_n имеют вид $\varphi(\ldots)$, то $u_1^\omega \equiv \omega$, …, $u_n^\omega \equiv \omega$, откуда следуют соотношения

$$\begin{aligned} r_k^{(\textcircled{\omega}/\varphi)} &= a_1^\omega\ u_1^\omega\ \ldots\ a_n^\omega\ u_n^\omega\ a^\omega \equiv a_1^\omega\ \omega\ \ldots\ a_n^\omega\ \omega\ a^\omega \leq \\ &\leq a_1^\omega\ v_1^\omega\ \ldots\ a_n^\omega\ v_n^\omega\ a^\omega = s_m^{(\textcircled{\omega}/\varphi)}, \end{aligned}$$

которые противоречат соотношениям (39) и (46). ■

8 Свойства правил PO, LO, PI, LI

8.1 Безопасность правила PO

Теорема 10

Вычислительное правило **PO** безопасно для любой ФП.

Доказательство.

Рассмотрим произвольный терм $C^i_{\Sigma,\vec{d}}$ в вычислительной последовательности, которая строится по правилу **PO**.

Если $C^i_{\Sigma,\vec{d}}$ имеет вид $\varphi(\ldots)$, то $(C^{i,\omega}_{\Sigma,\vec{d}})^{(\sigma/\varphi)} = \omega$, т.е. в этом случае условие (34) выполнено.

Рассмотрим случай, когда $C^i_{\Sigma,\vec{d}}$ имеет вид

$$g(\ldots\varphi(\vec{e}_1)\ldots\varphi(\vec{e}_k)\ldots), \tag{51}$$

где $g \in Fun$ и $\varphi(\vec{e}_1)$, …, $\varphi(\vec{e}_k)$ – раскрываемые подтермы.

Условие (34) для терма (51) имеет вид

$$g(\ldots\omega\ldots\omega\ldots) = \omega, \tag{52}$$

где терм в левой части является результатом замены подтермов $\varphi(\vec{e}_i)$ терма (51) на ω.

Докажем соотношение (52). Предположим, что оно неверно, т.е. для некоторой константы $d \neq \omega$ верно соотношение

$$g(\ldots\omega\ldots\omega\ldots) = d.$$

Тогда в силу монотонности функции g для каждого списка b_1, …, b_k констант соответствующих типов верно соотношение

$$g(\ldots b_1\ldots b_k\ldots) = d, \tag{53}$$

откуда следует, что значение терма $C^i_{\Sigma,\vec{d}}$ равно d, поскольку

- $\varphi(\vec{e}_1)$, …, $\varphi(\vec{e}_k)$ – самые внешние подтермы терма $C^i_{\Sigma,\vec{d}}$, имеющие вид $\varphi(\ldots)$, и
- в терме $C^i_{\Sigma,\vec{d}}$ нет переменных.

Согласно нижеследующей лемме, из совпадения значения терма $C^i_{\Sigma,\vec{d}}$ с константой d следует совпадение самого терма $C^i_{\Sigma,\vec{d}}$ с константой d, которое противоречит предположению о том, что $C^i_{\Sigma,\vec{d}}$ имеет вид (51).

Лемма

Для любого неупрощаемого терма v и любой константы $d \neq \omega$ верна импликация

$$v \equiv d \quad \Rightarrow \quad v = d \tag{54}$$

(где равенство понимается как совпадение термов, а не их значений).

Доказательство.

Индукция по длине терма v.

Соотношение $v \equiv d$ не может быть верным, когда

- $v = d'$, где d' – константа и $d' \neq d$,
- $v = x$, где x – переменная, т.к. в этом случае $v^{(d'/x)} = d' \neq d$, если d' – константа и $d' \neq d$,
- v имеет вид $\varphi(\ldots)$, т.к. в этом случае $v^{(\textcircled{\omega}/\varphi)} \equiv \omega \neq d$.

Предположим, что соотношение $v \equiv d$ верно, когда

$$v = g(v_1, \ldots, v_n), \quad \text{где} \quad g \in Fun.$$

Докажем, что в этом случае будет верно совпадение термов

$$\hat{v} = d, \tag{55}$$

которое противоречит предположению о неупрощаемости v.

(55) следует из доказываемого ниже соотношения

$$\begin{gathered} g(d_1, \ldots, d_n) \equiv d, \quad \text{где } \forall\, i = 1, \ldots, n \\ d_i \stackrel{\text{def}}{=} \begin{cases} v_i, & \text{если } v_i \text{ – константа} \\ \omega & \text{иначе.} \end{cases} \end{gathered} \tag{56}$$

Для каждого терма u будем обозначать записью u^ω терм, получаемый из $u^{(\textcircled{\omega}/\varphi)}$ заменой всех входящих в него переменных на константу ω.

Из $v \equiv d$ следует, что

$$d \equiv v^\omega = g(v_1^\omega, \ldots, v_n^\omega). \tag{57}$$

Докажем, что

$$\forall\, i = 1, \ldots, n \quad v_i^\omega \equiv d_i. \tag{58}$$

- Если v_i – константа, то $d_i = v_i = v_i^\omega$.

- Если v_i – не константа, то $d_i = \omega$, и если бы v_i^ω не был эквивалентен ω, т.е. $v_i^\omega \equiv d' \neq \omega$, то было бы верно соотношение

$$v_i \equiv d' \neq \omega. \tag{59}$$

  Поскольку терм v неупрощаемый, то терм v_i тоже неупрощаемый. По индуктивному предположению, верна импликация (54), с заменой v на v_i, т.е. из (59) следует равенство $v_i = d'$, которое противоречит предположению о том, что v_i – не константа.

Соотношение (56) следует из (57) и (58).

Таким образом, соотношение $v \equiv d$ может быть верно только в том случае, когда v является константой d. ■

8.2 Безопасность правила LO

Теорема 11

Если в ФП Σ каждому входящему в неё ФС, кроме *if_then_else*, сопоставлена функция, являющаяся естественным продолжением, то вычислительное правило **LO** является безопасным для Σ.

Доказательство.

Обозначим символом r раскрываемый подтерм в терме $C^i_{\Sigma,\vec{d}}$ вычислительной последовательности, которая строится по правилу **LO**.

Если $r = C^i_{\Sigma,\vec{d}}$, то $(C^{i,\omega}_{\Sigma,\vec{d}})^{(\sigma/\varphi)} = \omega$, т.е. в этом случае условие (34) выполнено.

Пусть $r \neq C^i_{\Sigma,\vec{d}}$. Так как r – самый внешний подтерм вида $\varphi(\ldots)$ в терме $C^i_{\Sigma,\vec{d}}$, то существует последовательность $r_1, \ldots, r_n$ подтермов терма $C^i_{\Sigma,\vec{d}}$, обладающая следующими свойствами:

- $r_1 = r$, $r_n = C^i_{\Sigma,\vec{d}}$;
- $\forall\, j = 1, \ldots, n-1$ терм r_{j+1} имеет вид

$$g(v_1, \ldots, v_m) \quad (g \in Fun), \tag{60}$$

  где для некоторого $k \in \{1, \ldots, m\}$ $\quad v_k = r_j$.

Терм r является подтермом всех термов $r_1, \ldots, r_n$. Обозначим записью r_j^ω (где $j = 1, \ldots, n$) терм, получаемый из терма r_j заменой его подтерма r на константу ω.

Докажем индукцией по j, что $\forall\, j = 1, \ldots, n$

$$r_j^\omega = \omega. \tag{61}$$

Для $j = 1$ соотношение (61), очевидно, верно.
Пусть для некоторого $j \in \{1, \ldots, n-1\}$

- верно (61),
- r_{j+1} имеет вид (60) и
- $\exists\, k \in \{1, \ldots, m\} : v_k = r_j$.

тогда

$$r_{j+1}^\omega = g(v_1, \ldots, v_{k-1}, r_j^\omega, v_{k+1}, \ldots, v_m). \tag{62}$$

Если $g \neq \mathit{if\_then\_else}$, то соотношение

$$r_{j+1}^\omega = \omega$$

следует из (61), (62) и предположения о том, что функция, соответствующая ФС g, является естественным продолжением.

Пусть $g = \mathit{if\_then\_else}$. Докажем, что в этом случае номер k, такой, что $v_k = r_j$, равен 1.

Если $k \neq 1$, то r входит в v_2 или v_3. По предположению, r является самым левым из самых внешних подтермов вида $\varphi(\ldots)$ в терме $C^i_{\Sigma,\vec{d}}$. Поэтому v_1 не содержит вхождений φ. Следовательно, v_1 состоит только из констант и ФС. Поскольку v_1 неупрощаемый, то, значит, он является константой.

Согласно определению функции $\mathit{if\_then\_else}$, если v_1 является константой, то терм

$$r_{j+1} = \mathit{if\_then\_else}(v_1, v_2, v_3)$$

является упрощаемым. Это противоречит тому, что терм r_{j+1} – неупрощаемый, поскольку он является подтермом неупрощаемого терма $C^i_{\Sigma,\vec{d}}$.

Итак, $k = 1$, и $r_{j+1} = \mathit{if\_then\_else}(r_j, v_2, v_3)$, откуда следует, что

$$r_{j+1}^\omega = \mathit{if\_then\_else}(r_j^\omega, v_2, v_3). \tag{63}$$

Учитывая индуктивное предположение $r_j^\omega = \omega$ и определение функции $\mathit{if\_then\_else}$, заключаем, что (63) $= \omega$.

Таким образом, $\forall\, j = 1, \ldots, n$ верно (61).

В частности, $r_n^\omega = \omega$, поэтому $(C^{i,\omega}_{\Sigma,\vec{d}})^{(\sigma/\varphi)} = (r_n^\omega)^{(\sigma/\varphi)} = \omega$. ■

8.3 Пример небезопасности правила LO

В излагаемом ниже примере используется ФС $\cdot$ типа $(\mathbf{I},\mathbf{I}) \to \mathbf{I}$, которому соответствует следующая монотонная функция:

- на $D_{\mathbf{I}} \times D_{\mathbf{I}}$ она совпадает с обычным умножением;
- $0 \cdot \omega = \omega \cdot 0 = 0$;
- $\forall\, d \in D_{\mathbf{I}} \setminus \{0\} \quad d \cdot \omega = \omega \cdot d = \omega$;
- $\omega \cdot \omega = \omega$.

Рассмотрим ФП

$$\Sigma: \quad \begin{array}{ll} \varphi(x) = (x = 0) & ?\ 0 \\ & :\ \varphi(x+1) \cdot \varphi(x-1) \end{array} .$$

σ принимает на всех аргументах значение 0. Однако вычисление $\varphi(1)$ по правилу **LO** будет бесконечным.

8.4 Пример небезопасности правил PI и LI

Пусть ФП Σ имеет вид

$$\Sigma: \quad \begin{array}{ll} \varphi(x,y) = (x = 0) & ?\ 1 \\ & :\ \varphi(x-1, \varphi(x,y)) \end{array}$$

Тогда

- $\sigma = (x \geq 0)?\ 1 : \omega$, но
- $\mathbf{PI}_\Sigma = \mathbf{LI}_\Sigma = (x = 0)?\ 1 : \omega$.

Например, вычисление $\varphi(1,0)$ по правилу **LI** породит бесконечную последовательность

$$\begin{array}{l} \mathbf{LI}^0_\Sigma = \varphi(1,0) \\ \mathbf{LI}^1_\Sigma = \varphi(0, \varphi(1,0)) \\ \mathbf{LI}^2_\Sigma = \varphi(0, \varphi(0, \varphi(1,0))) \\ \ldots \end{array}$$

Межфакультетский курс «Формальная семантика и верификация программного обеспечения» Лекция 7 Верификация распределенных систем. Задача вычисления произведения матриц

А.М.Миронов

1 Спецификация и верификация распределенных программ

В этом параграфе мы предполагаем, что рассматриваются только такие РП, в которых нет ЭД вида $?e$ и $!e$.

Спецификация таких РП м.б. задана в виде постусловия $Post \in Fm$. Задача верификации РП P относительно постусловия $Post$ заключается в обосновании следующих утверждений:

- для каждого полного выполнения $\theta_0, \ldots, \theta_n$ РП P $Post^{\theta_n} = 1$,
- не существует бесконечных выполнений P,
- не существует таких выполнений $\theta_0, \ldots, \theta_n$ РП P, что состояние θ_n является **тупиковым** (т.е. θ_n – не терминальное и не существует переходов РП P с началом в θ_n).

Для формального обоснования изложенных выше утверждений введем следующие обозначения. Пусть действие $A \in Act$ не содержит ЭД вида $?e$ и $!e$, $Var(A) \subseteq X \subseteq Var$ и $\varphi, \psi \in Fm(X)$. Запись

$$\varphi \overset{A}{\to} \psi \tag{1}$$

обозначает следующее утверждение:

$$\forall\, \theta, \theta' \in X^{\bullet}, \text{ если } \theta \overset{A}{\to} \theta', \text{ то из } \varphi^{\theta} = 1 \text{ следует } \psi^{\theta'} = 1.$$

Ниже будет использоваться следующее обозначение: пусть $\varphi \in Fm$, и $e, e' \in Tm$, $\tau(e) = \tau(e')$. Запись $\varphi^{(e'/e)}$ обозначает формулу

$$(e^{(\vec{x}'/\vec{x})} = e') \to \varphi^{(\vec{x}'/\vec{x})}, \tag{2}$$

где

- $\vec{x} = (x_1, \ldots, x_n)$ – список всех переменных из $Var(e)$,
- $\vec{x}' = (x'_1, \ldots, x'_n)$ – список новых различных переменных-дубликатов соответствующих переменных из $\vec{x}$, причём переменные из $\vec{x}'$ не встречаются за пределами формулы (2),
- $(\vec{x}'/\vec{x})$ обозначает подстановку $(x'_1/x_1, \ldots, x'_n/x_n)$.

Утверждение (1) можно определить индукцией по структуре A:

- $\varphi \stackrel{\varepsilon}{\to} \psi$ эквивалентно импликации $\varphi \to \psi$,
- $\varphi \stackrel{A(e:=e')}{\longrightarrow} \psi$ эквивалентно утверждению $\varphi \stackrel{A}{\to} \psi^{(e'/e)}$,
- $\varphi \stackrel{A\{\beta\}}{\longrightarrow} \psi$ эквивалентно утверждению $\varphi \stackrel{A}{\to} (\beta \to \psi)$,
- $\varphi \stackrel{A(c?e)}{\longrightarrow} \psi$ эквивалентно утверждению $\varphi \stackrel{A}{\to} \psi^{(head(x_c)/e, tail(x_c)/x_c)}$,
- $\varphi \stackrel{A(c!e)}{\longrightarrow} \psi$ эквивалентно утверждению $\varphi \stackrel{A}{\to} \psi^{(x_c e/x_c)}$.

Утверждения 1 и 2 в начале этого параграфа м.б. обоснованы, например, путем построения формулы $\varphi \in Fm$ (называемой **инвариантом** РП P), фундированного множества L и терма $u \in Tm$, таких, что доказуемы импликации:

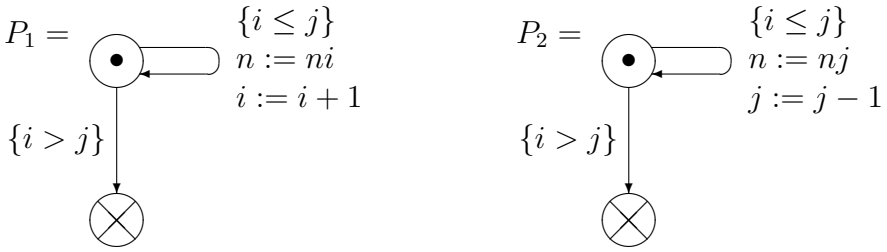
$$\begin{array}{l} Pre \to \varphi, \;\; \varphi \stackrel{A}{\to} \varphi, \;\; \left\{ \begin{array}{l} \varphi \\ \bigwedge_{i \in I} (at_{P_i} = \otimes) \end{array} \right\} \to Post, \\ \varphi \to (u \in L), \;\; \varphi \wedge (u = x) \stackrel{A}{\to} (u < x), \end{array} \tag{3}$$

где A – метка какого-либо ребра произвольного процесса из P, x – новая переменная типа $\tau(u)$, не встречающаяся за пределами последней формулы из (3).

2 Примеры распределенных программ

2.1 Вычисление факториала

РП $P = \{P_1, P_2\}$ предназначена для вычисления $n = k!$, где k – входное значение. Процессы P_1 и P_2 имеют следующий вид:



P_1 и P_2 имеют общие переменные n, i, j, k. В переменной n в конце работы будет содержаться искомое значение $k!$. В начале работы $n = 1$. P_1 домножает n на числа из списка $(1, 2, \ldots, k)$, начиная с его начала и двигаясь по этому списку слева направо, а P_2 домножает n на числа из того же списка $(1, 2, \ldots, k)$, начиная с его конца и двигаясь по этому списку справа налево. Процессы заканчивают свою работу, когда домножаемые числа (i в P_1 и j в P_2) удовлетворяют неравенству $i > j$.

Спецификация: $\left\{ \begin{array}{l} Pre = \ (n = 1) \wedge (i = 1) \wedge (j = k), \\ Post = \ (n = k!). \end{array} \right.$

Для верификации можно использовать следующие инвариант φ, фундированное множество L и функцию u:

$\varphi = \left\{ \begin{array}{l} i \le j + 1, \ \ n = (i-1)!\frac{k!}{j!} \\ (at_{P_1} = \otimes) \vee (at_{P_2} = \otimes) \rightarrow (i > j) \end{array} \right\}$,

$L = \mathbf{N}$, $u = j + 1 - i + [\![at_{P_1} = \odot]\!] + [\![at_{P_2} = \odot]\!]$.

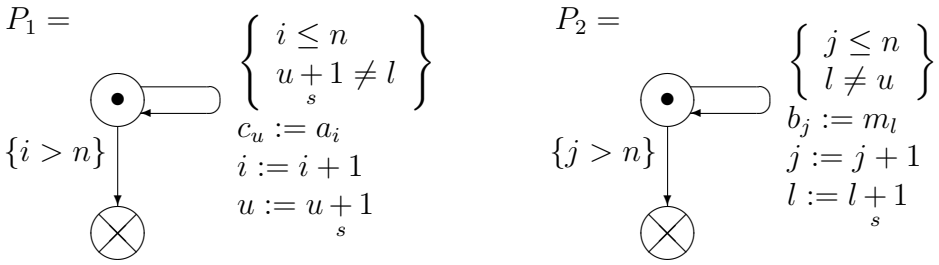
2.2 Передача сообщений через буфер

Рассматриваемая в этом пункте РП P имеет вид $\{P_1, P_2\}$ и предназначена для записи значений массива $a_{1..n}$ в соответствующие компоненты массива $b_{1..n}$. Компоненты массива b недоступны процессу P_1, и компоненты массива a недоступны процессу P_2, т.е. выполнение действий вида $b_i := a_i$ невозможно. Для решения указанной выше задачи используются

- массив $c_{0..s-1}$, доступный процессам P_1 и P_2, он рассматривается как буфер между P_1 и P_2, в нем содержатся значения, которые посланы процессом P_1, но пока не получены процессом P_2, и

- доступные обоим процессам P_1 и P_2 переменные l и u, принимающие значения в множестве $\{0, \ldots, s-1\}$, значения данных переменных являются нижней и верхней границей соответственно области массива c, хранящей отправленные, но пока еще не доставленные значения, операции с l и u выполняются по модулю s, мы будем обозначать их записями $\underset{s}{+}$ и $\underset{s}{-}$.

Процессы P_1 и P_2 имеют следующий вид:



Спецификация: $\left\{ \begin{array}{l} Pre = \ (i = j = 1) \wedge (u = l = 0) \wedge (n \geq 1) \wedge (s \geq 1), \\ Post = \ \bigwedge_{i=1}^{n} (b_i = a_i). \end{array} \right.$

Для верификации можно использовать следующий инвариант φ, фундированное множество L и функцию u:

$$\varphi = \left\{ \begin{array}{l} (i \leq n+1) \wedge (j \leq n+1) \\ a_{1..i-1} = b_{1..j-1} c_{l..u-1} \\ (at_{P_1} = \otimes) \rightarrow (i = n+1) \\ (at_{P_2} = \otimes) \rightarrow (j = n+1) \end{array} \right\},$$

$L = \mathbf{N}$, $u = (n+1-i) + (n+1-j) + [\![at_{P_1} = \odot]\!] + [\![at_{P_2} = \odot]\!]$.

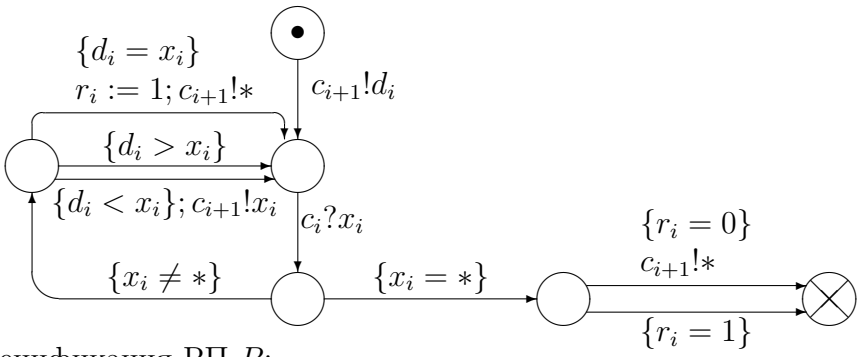
Запись $a_{1..i-1} = b_{1..j-1} c_{l..u-1}$ понимается следующим образом: если $i > 1$, $j > 1$, $l \neq u$, то часть $a_{1..i-1}$ массива a является конкатенацией части $b_{1..j-1}$ массива b и части $c_{l..u-1}$ массива c. При этом если $0 < u < l$, то $c_{l..u-1} \stackrel{\text{def}}{=} c_{l..s-1} c_{0..u-1}$, и в остальных случаях указанная выше запись тоже определяется естественным образом.

2.3 Избрание лидера

В этом пункте рассматривается РП $P = \{P_1, \ldots, P_n\}$, где для каждого $i = 1, \ldots, n$ процесс P_i содержит значение d_i и переменные x_i и r_i, причем значения $d_1, \ldots, d_n$ различны. На взаимодействие между процессами $P_1, \ldots, P_n$ накладывается ограничение: $\forall\, i = 1, \ldots, n$ P_i может получать сообщения только от P_{i-1} и посылать сообщения только P_{i+1}, где

$n+1 \stackrel{\text{def}}{=} 1$ и $1-1 \stackrel{\text{def}}{=} n$. Задачей P является нахождение «лидера» среди своих компонентов, т.е. такого P_i, что $d_i = \max_{j=1,\ldots,n} d_j$. В P используется выделенное значение $*$, отличное от всех d_i.

$\forall\, i = 1, \ldots, n$ процесс P_i имеет следующий вид:



Спецификация РП P:

$$\begin{cases} Pre = (n \geq 2) \wedge \bigwedge_{i=1}^{n} \Big((r_i = 0) \wedge (d_i \neq *) \wedge \big(\bigwedge_{j \neq i} (d_j \neq d_i) \big) \Big), \\ Post = \bigvee_{i=1}^{n} \Big((r_i = 1) \wedge \big(\bigwedge_{j \neq i} (r_j = 0) \wedge (d_j < d_i) \big) \Big). \end{cases}$$

2.4 Параллельная сортировка

Излагаемая в этом пункте РП $P = \{P_0, \ldots, P_n\}$ предназначена для сортировки массива $a_{1..n}$, результат записывается в массив $b_{1..n}$.

Как известно, нижняя оценка временно́й сложности любого алгоритма сортировки массива из n элементов на однопроцессорной машине равна $O(n \log_2 n)$, однако в том случае, когда число доступных процессоров для сортировки массива из n элементов больше n, временна́я сложность решения этой задачи может быть понижена до $O(n)$ за счет возможности параллельной работы процессоров. Ниже излагается РП, которую можно исполнять на $(n+1)$-процессорной машине, получая при этом указанное выше понижение временно́й сложности.

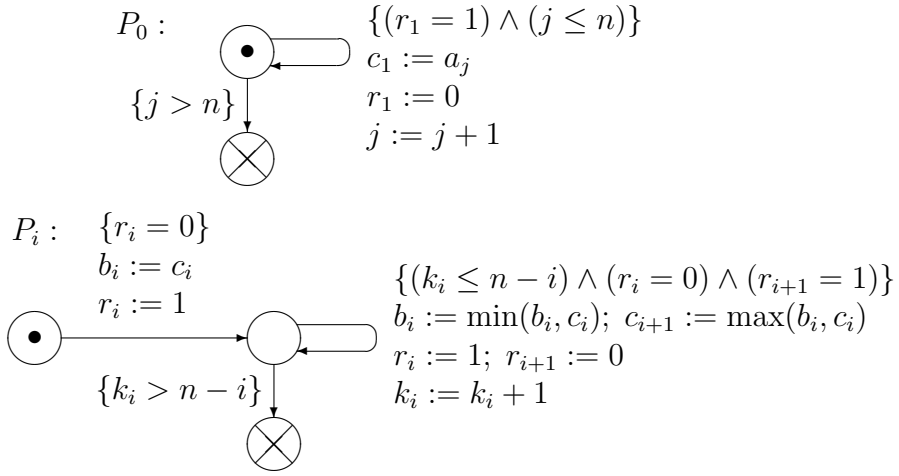
Идея излагаемой ниже РП заключается в следующем. Процесс P_0 получает по очереди элементы массива a, и затем эти элементы передаются процессам $P_1, \ldots, P_n$ по цепочке «слева направо». В каждый момент времени процесс P_i $(i = 1, \ldots, n)$ хранит в b_i некоторый элемент, полученный от предыдущего процесса P_{i-1}, и сравнивает с этим элементом следующий получаемый от P_{i-1} элемент c_i: если $c_i \geq b_i$, то процесс P_i передаёт c_i процессу P_{i+1}, иначе P_i передаёт элемент в b_i процессу P_{i+1} и записывает c_i в b_i.

Алгоритм работы процессов $P_0, \ldots, P_n$ имеет следующий вид. P_0 записывает очередной полученный элемент в переменную c_1, после этого

- устанавливает значение переменной r_1 в 0, что дает возможность процессу P_1 начать обрабатывать элемент c_1, и
- увеличивает на 1 номер j ожидаемого элемента массива a.

Каждый из процессов P_i $(i = 1, \ldots, n)$ получает возможность выполнить очередной шаг, только когда $r_i = 0$ (эту возможность ему предоставляет процесс P_{i-1}). После выполнения процессом P_i очередного шага значение переменной k_i равно числу принятых элементов. После того как P_i передает своему правому соседу элемент c_{i+1}, он открывает ему возможность выполнить очередной шаг действием $r_{i+1} := 0$, а для себя устанавливает r_i в 1, т.е. следующий шаг P_i будет возможен, только когда P_{i-1} передаст P_i следующий элемент в c_i и установит r_i в 0.

Процессы P_0 и P_i $(i = 1, \ldots, n)$ имеют следующий вид:



Спецификация: $\begin{cases} Pre = (n \geq 1) \wedge (j = 1) \wedge \bigwedge_{i=1}^{n}((r_i = 1) \wedge (k_i = 1)), \\ Post = perm(a, b) \wedge ord(b). \end{cases}$

3 Распределенная программа перемножения матриц

В этом пункте излагается пример РП, задача которой – по матрицам A, B вычислить их произведение. Данная РП является математической моделью приведённой в [**?**] MPI-программы перемножения матриц. РП P имеет вид $\{P_0, \ldots, P_n\}$, где $n \geq 1$. Процесс P_0 называется **менеджером**, процессы $P_1, \ldots, P_n$ называются **работниками**.

3.1 Неформальное описание распределенной программы перемножения матриц

Неформально работу излагаемой в этом пункте РП P для перемножения матриц A и B можно описать следующим образом. Каждый работник имеет матрицы A и B. Работа менеджера заключается в назначении задач работникам и приеме результатов от них. Каждая задача работнику заключается в вычислении одной строки матрицы $C = AB$. Менеджер назначает эту задачу путем посылки работнику номера i той строки матрицы A, которую работник должен умножить на B. Когда менеджер получает от работника результат (т.е. сообщение с вычисленной строкой $A_iB = C_i$), он посылает этому работнику новый номер (если еще есть неназначенные строки) или 0 (если неназначенных строк нет).

3.2 Спецификация программы перемножения матриц

Спецификация изложенной выше РП перемножения матриц представляет собой следующее утверждение: после завершения выполнения этой РП верно равенство $C = AB$, которое эквивалентно утверждению

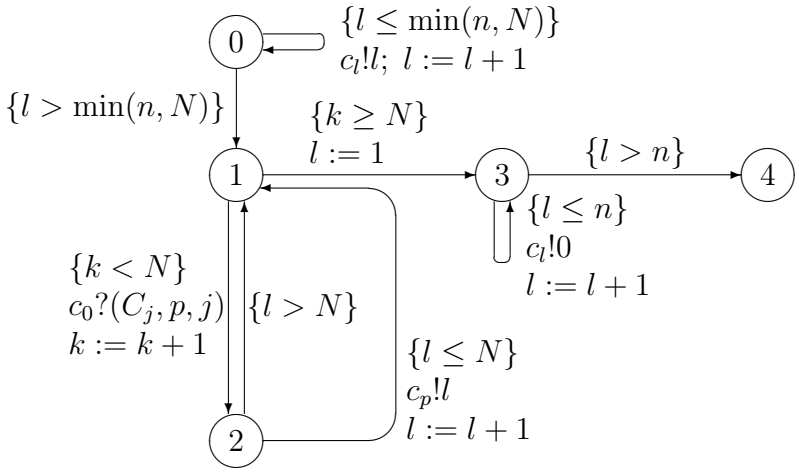
$$\forall\, j = 1, \ldots, N \quad C_j = A_jB. \tag{4}$$

3.3 Определение распределённой программы перемножения матриц

В этом пункте мы определяем РП $P = \{P_0, P_1, \ldots, P_n\}$, перемножения матриц. Входными данными P являются A, B (матрицы-сомножители), N (число строк в A), n (число работников).

3.3.1 Процесс P_0 («менеджер»)

В процессе P_0 строки матрицы-результата C записываются в переменные $C_1, \ldots, C_N$. Процесс P_0 имеет следующий вид:



(5)

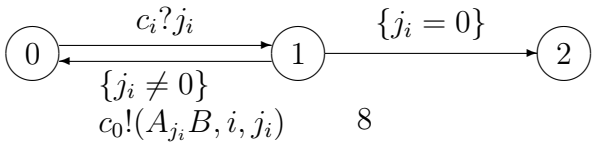
$Pre(P_0) = (N \ge 1) \wedge (l = 1) \wedge (k = 0)$.

Поясним работу процесса P_0:

- переход $0 \to 0$: менеджер P_0 дает первые задания работникам: посылает работникам по очереди номер l строки матрицы A до тех пор, пока либо не закончатся номера строк, либо все имеющиеся работники не получат задачи,
- переход $0 \to 1$: вход в цикл (состоящий из перехода $1 \to 2$ и двух переходов $2 \to 1$) приема результатов от работников и назначения им новых заданий:
  - переход $1 \to 2$: получение от работника результата выполнения полученной им задачи,
  - переходы $2 \to 1$: выдача работнику новой задачи (если такая задача имеется) либо пустой переход (если новых задач нет),
- переход $1 \to 3$: завершение приема результатов от работников, вход в цикл $3 \to 3$ посылки работникам сигнала окончания работы (0),
- переход $3 \to 4$: завершение работы.

3.3.2 Процесс P_i для $i > 0$ («работник»)

Процесс P_i имеет следующий вид:



(6)

Символ i в (6) обозначает константу, равную номеру этого процесса. Поясним работу процесса P_i:

- переход $0 \to 1$: P_i получает от менеджера число j_i, которое интерпретируется либо как номер строки матрицы A, которую нужно умножить на B и послать вычисленное произведение менеджеру (если $j_i \neq 0$), либо как сигнал об окончании работы (если $j_i = 0$),
- переход $1 \to 0$: посылка менеджеру вычисленного произведения,
- переход $1 \to 2$: окончание работы.

3.4 Дополненные процессы

Для облегчения верификации РП P можно добавить

- к $Var(P)$ дополнительные переменные, называемые **вспомогательными переменными** (будем обозначать записью $Var(P)$ исходное множество переменных P вместе с добавленными переменными), и
- к действиям процессов, входящих в P, дополнительные присваивания вида $x := e$, где x – вспомогательная переменная.

Вышеуказанные присваивания не оказывают влияния на выполнение исходной РП P, они предназначены для выражения зависимостей между значениями переменных во время выполнения исходной РП.

Процессы, получаемые путем добавления вышеуказанных присваиваний, будем называть **дополненными** процессами.

Межфакультетский курс «Формальная семантика и верификация программного обеспечения» Лекция 8 Пример верификации распределенной программы (умножение двух матриц)

А.М.Миронов

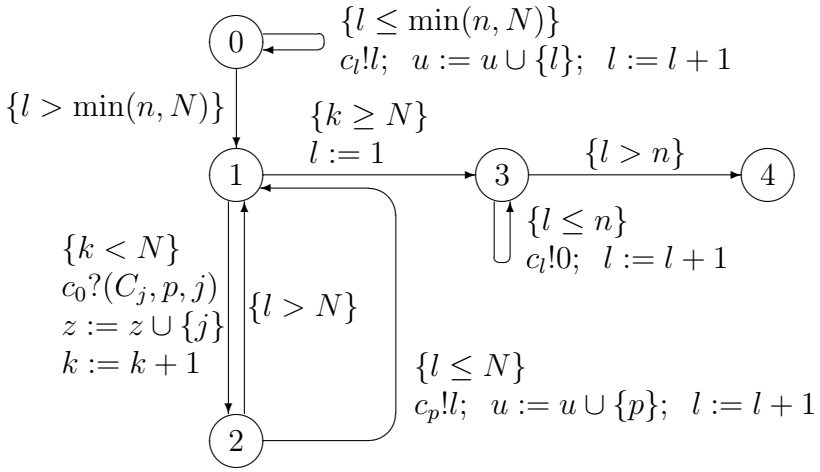
1 Верификация распределенной программы перемножения матриц

Для доказательства утверждения о том, что определенная в предыдущей лекции РП P удовлетворяет спецификации $Post = (\forall\, l = 1, \ldots, N \;\; C_l = A_l B)$, добавим к $Var(P)$ вспомогательные переменные u, y, z и связанные с ними присваивания. Значения вспомогательных переменных будут иметь следующий смысл: $\forall\, \theta \in Var(P)^\bullet$

- $u^\theta \subseteq \{1, \ldots, n\}$, u^θ состоит из номеров каналов из $\{c_1, \ldots, c_n\}$ с непустым содержимым в состоянии θ,
- $y^\theta \subseteq \{1, \ldots, N\}$, y^θ состоит из номеров строк матрицы A, для которых в состоянии θ вычисляется их произведение на B, но результат вычисления пока не помещен в канал c_0,
- $z^\theta \subseteq \{1, \ldots, N\}$, z^θ равно множеству номеров всех строк, которые P_0 записал в C до момента прихода РП P в состояние θ.

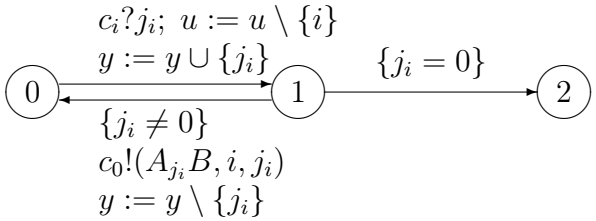
Начальные значения u, y и z равны $\emptyset$.

Дополненный процесс P_0 имеет следующий вид:



(1)

$\forall\, i = 1, \ldots, n$ дополненный процесс P_i имеет следующий вид:



(2)

1.1 Теоремы, обосновывающие корректность P

Будем использовать следующее обозначение: если d – множество кортежей, то d^k обозначает множество k–х компонентов кортежей из d.

Теорема 1

Для каждого выполнения π РП P и каждого состояния $\theta \in \pi$, если $at^{\theta}_{P_0} \notin \{3, 4\}$, то в θ верны утверждения:

1. $x^3_{c_0} \sqcup x_{c_1} \sqcup \ldots \sqcup x_{c_n} \sqcup y \sqcup z = \{1, \ldots, l-1\} \subseteq \{1, \ldots, N\}$,
2. $x^2_{c_0} \sqcup u \subseteq \{1, \ldots, l-1\}$,
3. если $at_{P_i} = 1$, где $i > 0$, то $(j_i \in y) \wedge (i \notin u) \wedge (i \in \{1, \ldots, l-1\})$,
4. $\forall\, i = 1, \ldots, n \quad i \in y \ \Leftrightarrow\ at_{P_i} \ne 0$,
5. если $at_{P_0} = 2$, то $(p \notin u) \wedge (p \in \{1, \ldots, l-1\})$,
6. $\forall\, i = 1, \ldots, n\ \ |x_{c_i}| = 1$, если $i \in u$, и 0 иначе,

7. каждый элемент x_{c_0} имеет вид $(A_l B, v, l)$,

8. $|z| = k$,

9. $\forall\, j \in z \;\; C_j = A_j B$.

Доказательство.

Данные утверждения верны в начальном состоянии РП P.

Пусть эти утверждения верны в θ, где $at^{\theta}_{P_0} \neq 3, 4$, и $\theta \xrightarrow{A} \theta'$ – переход РП P, причем $at^{\theta'}_{P_0} \neq 3, 4$. Для доказательства истинности в θ' утверждений 1–9 в формулировке теоремы достаточно рассмотреть P_0-переходы $0 \to 0$, $1 \to 2$, правый переход $2 \to 1$ и P_i-переходы $(i > 0)$ $0 \to 1$, $1 \to 0$.

Ниже вместо фразы «из истинности в состоянии θ утверждения S в формулировке теоремы» будем говорить кратко «из S».

- При P_0-переходе $0 \to 0$ изменяются значения x_{c_l}, u и l. Из 1 следует, что при добавлении l к x_{c_l} множества в левой части 1 останутся дизъюнктными, и из условия $\{l \leq \min(n, N)\}$ данного перехода следует, что равенство и включение в 1 будут верны в θ'. Свойство 2 верно в θ' потому, что оно верно в θ, $u^{\theta'} = u^{\theta} \cup \{l^{\theta}\}$, значение $x^2_{c_0}$ не изменяется при этом переходе, и $l \notin x^2_{c_0}$ в θ, т.к. 2 верно в θ. Свойство 5 верно в θ' потому, что $at^{\theta'}_{P_0} \neq 2$. Свойство 6 верно в θ' потому, что оно верно в θ, и при переходе от θ к θ' среди переменных $x_{c_1}, \ldots x_{c_n}$ изменяется лишь значение $x_{c_{l^\theta}}$, и поскольку $l^{\theta} \notin u^{\theta}$, то $|x_{c_{l^\theta}}| = 0$, следовательно, в θ' верно $|x_{c_l}| = 1$, и кроме того, $l^{\theta} \in u^{\theta'}$. Свойства 7, 8, 9 верны в θ' потому, что значения переменных, упомянутых в данных свойствах, не изменяются при рассматриваемом переходе.

- При P_0-переходе $1 \to 2$ изменяются значения x_{c_0}, p, j, C_j, z, k.

  Свойство 1 верно в θ' потому, что изменения в множествах в левой части 1 заключаются в перенесении элемента j^{θ} из $(x^{\theta}_{c_0})^3$ в z^{θ}. Свойства 2, 6, 7 верны в θ' потому, что при переходе от θ к θ' значения u, l и $x_{c_1}, \ldots, x_{c_n}$ не изменяются, а значение x_{c_0} уменьшается. Свойство 5 следует из 2. Свойство 8 верно в θ' потому, что при переходе от θ к θ' к z добавляется новый элемент и k увеличивается на 1. 9 верно в θ' потому, что 9 и 7 верны в θ.

- При P_0-переходе $2 \to 1$ изменяются значения переменных x_{c_p}, u и l. Рассуждения для данного перехода почти аналогичны рассуждениям для P_0-перехода $0 \to 0$. Включение в 1 следует из условия $\{l \leq N\}$ в данном переходе. Свойство 2 в θ' следует из свойств 2 и

5 в θ. Свойство 6 в θ' верно потому, что из свойства 5 в θ следует, что $|x_{c_p}| = 0$ в θ, поэтому $|x_{c_p}| = 1$ в θ'. Свойства 7, 8, 9 верны в θ' потому, что они верны в θ, и переменные в этих свойствах не меняют значений при переходе от θ к θ'.

- При P_i-переходах $0 \to 1$ и $1 \to 0$, где $i > 0$, все свойства 1–9 в θ' следуют из соответствующих свойств в θ. Отдельно обоснуем свойство 3 в θ' для P_i-перехода $0 \to 1$. Данный переход возможен только в случае $|x^{\theta}_{c_i}| > 0$, что, согласно свойству 6 в θ, эквивалентно свойству $i \in u$ в θ. Из свойства 2 в θ заключаем, что верно свойство $i \in \{1, \ldots, l-1\}$ в θ. ■

Теорема 2
Любое выполнение РП P завершается после конечного числа шагов.

Доказательство.
Пусть существует бесконечное выполнение π РП P. Докажем, что число P_0-переходов в π конечно. Если бы число таких переходов было бесконечно, то из определения процесса P_0 следует, что

- в π нет состояний θ, таких, что $at^{\theta}_{P_0} = 3$,
- существует состояние $\theta \in \pi$, такое, что $at^{\theta}_{P_0} = 1$,
- начиная с этого состояния число P_0-переходов в π вида $1 \to 2$ бесконечно, что невозможно по причине того, что при каждом таком переходе увеличивается значение переменной k, которое, согласно утверждениям 1 и 8 теоремы 1, не превосходит N.

Пусть θ – такое состояние в π, начиная с которого π не содержит P_0-переходов, и π' – часть π, начинающаяся с θ. Нетрудно видеть, что

$$\begin{array}{l}\exists\, i \in \{1, \ldots, n\}\text{: } \pi' \text{ содержит бесконечно много} \\ P_i\text{-переходов вида } 0 \to 1.\end{array} \qquad (3)$$

Поскольку π' не содержит P_0-переходов, то значение $|x^{\theta}_{c_i}|$ не может увеличиваться, и согласно (3) оно бесконечно уменьшается, что невозможно. ■

Теорема 3
Не существует таких выполнений $\theta_0, \ldots, \theta$ РП P, что θ тупиковое состояние (т.е. θ – не терминальное и в P нет переходов с началом в θ).

Доказательство.

Пусть существует такое выполнение $\pi = (\theta_0, \ldots, \theta)$ РП P, что в P нет переходов с началом в θ. Нетрудно доказать, что $at^{\theta}_{P_0} \in \{1, 4\}$ и $\forall\, i = 1, \ldots, n\ at^{\theta}_{P_i} \in \{0, 2\}$.

1. Пусть $at^{\theta}_{P_0} = 1$. В этом случае во всех состояниях из π верно утверждение $at_{P_0} \notin \{3, 4\}$, поэтому в них верны все утверждения теоремы 1. Из отсутствия переходов с началом в θ следует, что $k^{\theta} < N$ и $x^{\theta}_{c_0} = \emptyset$, поэтому из утверждения 1 теоремы 1 следует, что в θ верно

$$x_{c_1} \sqcup \ldots \sqcup x_{c_n} \sqcup y \sqcup z = \{1, \ldots, l-1\} \subseteq \{1, \ldots, N\}. \tag{4}$$

   Если $\exists\, i > 0 : at^{\theta}_{P_i} = 2$, то из определения процесса P_i следует, что $\exists\, \theta' <_{\pi} \theta : at^{\theta'}_{P_i} = 0$ и $x^{\theta'}_{c_i} = \{0\}$. В θ' верно утверждение 1 теоремы 1, что влечет неверное утверждение $0 \in \{1, \ldots, N\}$. Поэтому

$$\forall\, i > 0 \quad at^{\theta}_{P_i} = 0. \tag{5}$$

   По утверждению 4 теоремы 1, из (5) следует, что $y^{\theta} = \emptyset$. Из (5) и из отсутствия переходов с началом в θ следует, что для каждого $i = 1, \ldots, n\ |x^{\theta}_{c_i}| = 0$, поэтому из (4) и из утверждения 8 теоремы 1 следует, что

$$k^{\theta} = |z^{\theta}| = l^{\theta} - 1 \leq N. \tag{6}$$

   Докажем, что соотношения (6) и $k^{\theta} < N$ несовместимы.

   Пусть θ' – первое состояние в выполнении π, такое, что $at^{\theta'}_{P_0} = 1$. Докажем, что среди P_0-переходов в π между θ' и θ нет среднего P_0-перехода вида $2 \to 1$ (с меткой $\{l > N\}$).

   Предположим, что такой переход есть и он имеет вид $\theta_1 \overset{\{l>N\}}{\longrightarrow} \theta_2$. Из утверждения 1 теоремы 1 для θ_1 следует, что $l^{\theta_1} - 1 \leq N$, что в сочетании с $l^{\theta_1} > N$ дает соотношение $N < l^{\theta_1} \leq N + 1$, из которого следует равенство $l^{\theta_1} - 1 = N$. Поскольку при переходах от θ' к θ значение l не убывает, то $l^{\theta} - 1 \geq N$, что в сочетании с (6) дает равенство $l^{\theta} - 1 = N$. Данное равенство противоречит неравенству $k^{\theta} = l^{\theta} - 1 < N$.

   Таким образом, совокупность P_0-переходов в π между θ' и θ представляет собой последовательность чередований P_0-перехода $1 \to 2$ и правого P_0-перехода $2 \to 1$. При выполнении каждой пары таких переходов значения k и l увеличиваются на 1, поэтому разность $l - k$ при выполнении каждой пары таких переходов остается постоянной. Поскольку $k^{\theta'} = 0 < l^{\theta'} - 1$, т.е. $(l-k)^{\theta'} > 1$, то $(l-k)^{\theta} > 1$, т.е. $k^{\theta} < l^{\theta} - 1$, что противоречит (6).

Таким образом, случай $at^{\theta}_{P_0} = 1$ невозможен.

2. Пусть $at^{\theta}_{P_0} = 4$. Из определения P_0 следует, что $\exists\, \theta' <_\pi \theta$: $at^{\theta'}_{P_0} = 1$ и $k^{\theta'} \geq N$. Поскольку в θ' верны утверждения 1 и 8 теоремы 1, то

$$N \leq k^{\theta'} = |z^{\theta'}| \leq N, \;\; \forall\, i = 0, 1, \ldots, n \;\; x^{\theta'}_{c_i} = \emptyset, \;\; y^{\theta'} = \emptyset,$$

откуда следует $k^{\theta'} = |z^{\theta'}| = N$.

Из истинности в θ' утверждения 4 теоремы 1 следует, что

$$\forall\, i = 1, \ldots, n \;\; at^{\theta'}_{P_i} = 0.$$

Нетрудно видеть, что часть выполнения π от θ' до θ представляет собой комбинацию P_0-переходов $1 \to 3$, $3 \to 3$ и $3 \to 4$, а также P_i-переходов $0 \to 1$ и $1 \to 2$ ($\forall\, i = 1, \ldots, n$). Следовательно, для каждого $i = 1, \ldots, n$ $at^{\theta}_{P_i} = 2$, т.е. θ – терминальное состояние.

Из истинности в θ' утверждения 9 теоремы 1, которое в данном случае имеет вид

$$\forall\, j \in z = \{1, \ldots, N\} \;\; C_j = A_j B, \tag{7}$$

следует, что и в терминальном состоянии θ верно (7). ■

Таким образом, P завершает свою работу после конечного числа шагов, и после завершения выполнения P верно постусловие (7). ■

Межфакультетский курс «Формальная семантика и верификация программного обеспечения» Лекция 9 Теория процессов. Понятие процесса. Операции над процессами. Эквивалентность процессов

А.М.Миронов

В предыдущих частях изучалось понятие процесса. В этой части мы рассматриваем данное понятие на разных уровнях абстракции. Сначала мы определяем понятие процесса в упрощенной формулировке и определяем различные алгебраические операции и эквивалентности на таких процессах. Затем мы обобщаем данное понятие до понятия процесса с передачей сообщений и рассматриваем обобщения операций и эквивалентностей на процессы с передачей сообщений.

Наиболее существенный вклад в теорию процессов внес выдающийся английский математик **Робин Милнер**.

1 Неформальное понятие процесса и примеры процессов

Прежде чем сформулировать формальное понятие процесса, мы опишем данное понятие неформально и рассмотрим простейшие примеры процессов.

1.1 Неформальное понятие процесса

Мы понимаем под процессом модель поведения динамической системы на некотором уровне абстракции.

Процесс можно представлять себе как граф P, компоненты которого имеют следующий смысл.

- Вершины графа P называются **состояниями** и изображают ситуации (или классы ситуаций), в которых может находиться моделируемая система в различные моменты своего функционирования.

  Одно из состояний является выделенным, оно называется **начальным состоянием** процесса P.

- Рёбра графа P имеют метки, изображающие **действия**, которые может исполнять моделируемая система.

- Функционирование процесса P описывается переходами по рёбрам графа P от одного состояния к другому. Функционирование начинается из начального состояния.

  Метка каждого ребра изображает действие процесса, исполняемое при переходе от состояния в начале ребра к состоянию в его конце.

1.2 Пример процесса

В качестве первого примера рассмотрим процесс, представляющий собой простейшую модель поведения некоторого торгового автомата.

Мы будем представлять себе этот автомат как машину, которая имеет монетоприемник, кнопку и лоток для выдачи товара. Когда покупатель хочет приобрести товар, он опускает монету в монетоприемник, нажимает на кнопку, и после этого в лотке появляется товар.

Предположим, что наш автомат торгует шоколадками по цене 1 монета за штуку. Опишем действия такого автомата.

- По инициативе покупателя в автомате могут происходить следующие действия: попадание в щель монеты и нажатие кнопки.

- В ответ автомат может осуществлять реакцию: выдавать в лоток шоколадку.

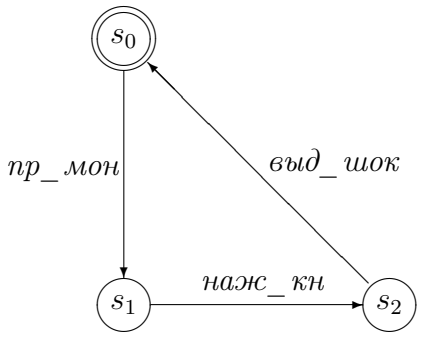
Обозначим действия короткими именами:

- прием монеты мы обозначим записью *пр_мон*,

- нажатие кнопки – записью *наж_кн*, и

- выдачу шоколадки – записью *выд_шок*.

Процесс нашего торгового автомата выглядит следующим образом:



(1)

Диаграмма (1) объясняет, как функционирует торговый автомат:

- вначале автомат находится в состоянии s_0, в этом состоянии он ожидает появления в приемнике монеты (то, что состояние s_0 является начальным, изображается на диаграмме двойным кружочком вокруг идентификатора этого состояния),
- когда монета появляется, автомат переходит в состояние s_1 и ждет нажатия на кнопку,
- после нажатия кнопки автомат переходит в состояние s_2, выдает шоколадку и возвращается в состояние s_0.

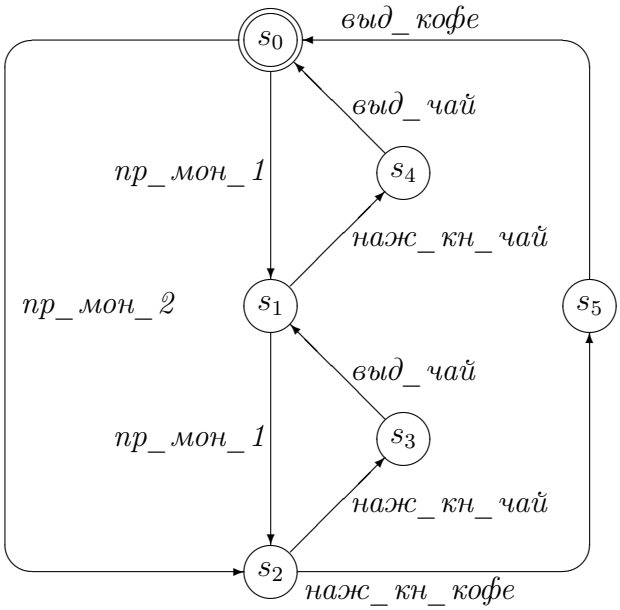
1.3 Другой пример процесса

Рассмотрим более сложный пример торгового автомата, который отличается от предыдущего тем, что продаёт два вида товаров: чай и кофе, причём стоимость чая – 1 рубль, а стоимость кофе – 2 рубля.

Автомат имеет две кнопки: одну – для чая, другую – для кофе.

Покупатель может платить монетами достоинством в 1 рубль и 2 рубля. Данные монеты будут обозначаться знакосочетаниями *мон_1* и *мон_2* соответственно. Если покупатель опустил в монетоприемник монету *мон_1*, он может купить только чай. Если же он опустил монету *мон_2*, он может купить кофе или два чая. Кофе можно купить также, опустив в монетоприёмник две монеты *мон_1*.

Процесс такого торгового автомата выглядит следующим образом:



Для формального определения понятия процесса мы должны уточнить понятие действия. Это уточнение излагается в параграфе 2.

2 Действия

Для задания процесса P, представляющего собой модель поведения некоторой динамической системы, должно быть указано множество $Act(P)$ **действий**, которые может выполнять процесс P. Будем предполагать, что действия всех процессов являются элементами некоторого универсального множества Act всех возможных действий, которые может выполнить какой-либо процесс, т.е. для любого процесса P $Act(P) \subseteq Act$.

Выбор множества $Act(P)$ действий процесса P зависит от целей моделирования. В разных ситуациях для представления модели анализируемой системы в виде некоторого процесса могут выбираться разные множества действий.

Будем предполагать, что Act делится на 3 следующих класса.

1. **Входные действия**, которые изображаются записями вида $?\alpha$. Действие вида $?\alpha$ понимается как прием объекта с именем α.

2. **Выходные действия**, которые изображаются записями вида $!\alpha$. Действие вида $!\alpha$ понимается как посылка объекта с именем α.

3. **Внутреннее** (или **невидимое**) действие, которое обозначается символом τ. Внутренним мы называем такое действие процесса P, которое не связано с его взаимодействием с **окружающей средой** (т.е. с процессами, являющимися внешними по отношению к процессу P и с которыми он может взаимодействовать). Например, внутреннее действие может быть связано со взаимодействием компонентов процесса P.

   В действительности внутренние действия могут быть самыми разнообразными, но для обозначения всех внутренних действий мы будем использовать один и тот же символ τ. Это отражает наше желание не различать все внутренние действия, т.к. они не являются «наблюдаемыми» извне процесса P.

Обозначим знакосочетанием $Names$ совокупность имён объектов, которые могут приниматься или посылаться процессами. Множество $Names$ предполагается бесконечным.

Множество Act представляет собой дизъюнкное объединение

$$Act = \{?\alpha \mid \alpha \in Names\} \sqcup \{!\alpha \mid \alpha \in Names\} \sqcup \{\tau\}.$$

Отметим, что объекты, которые принимаются и посылаются процессами, могут иметь самую различную природу (как материальную, так и нематериальную). Например, ими могут быть материальные ресурсы, люди, деньги, информация, энергия и т.д.

Кроме того, сами понятия приема и посылки могут иметь виртуальный характер, т.е. слова «прием» и «посылка» могут использоваться лишь как метафоры, а в действительности никакого приема или посылки какого-либо реального объекта может и не происходить.

Для каждого имени $\alpha \in Names$ действия $?\alpha$ и $!\alpha$ называются **комплементарными**. Будем использовать следующие обозначения:

- для каждого действия $a \in Act \setminus \{\tau\}$
  - $\bar{a}$ обозначает действие, комплементарное к a: $\overline{?\alpha} = !\alpha$, $\overline{!\alpha} = ?\alpha$,
  - $name(a)$ обозначает имя в a, т.е. $name(?\alpha) = name(!\alpha) = \alpha$,
- $\forall\, L \subseteq Act \setminus \{\tau\}$ $\overline{L} \stackrel{\text{def}}{=} \{\overline{a} \mid a \in L\}$, $names(L) \stackrel{\text{def}}{=} \{name(a) \mid a \in L\}$.

3 Определение понятия процесса

Процессом (над множеством действий Act) называется тройка P вида (S, s^0, R), компоненты которой имеют следующий смысл:

- S – множество, элементы которого называются **состояниями**,
- $s^0 \in S$ – состояние, называемое **начальным состоянием**,
- $R \subseteq S \times Act \times S$, элементы множества R называются **переходами**, если переход из R имеет вид (s_1, a, s_2), то
  - будем говорить, что этот переход является переходом из состояния s_1 в состояние s_2 с выполнением действия a,
  - состояния s_1 и s_2 называются **началом** и **концом** этого перехода, а действие a называется **меткой** этого перехода, и
  - будем обозначать данный переход записью $s_1 \xrightarrow{a} s_2$.

Совокупность всех процессов обозначается записью $Proc$.

Процесс (S, s^0, R) можно представлять себе как граф с множеством вершин S, выделенной вершиной s^0, рёбра которого соответствуют переходам из R: если переход имеет вид $s_1 \xrightarrow{a} s_2$, то ему соответствует ребро с началом s_1, концом s_2 и меткой a.

Функционирование процесса $P = (S, s^0, R)$ заключается в порождении последовательности переходов вида $s^0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} \dots$ и выполнении действий $a_0, a_1, a_2 \dots$, соответствующих этим переходам. Более подробно: на каждом шаге функционирования $i \geq 0$

- процесс находится в некотором состоянии s_i $(s_0 = s^0)$,
- если есть хотя бы один переход из R с началом в s_i, то P
  - недетерминированно выбирает переход с началом в s_i, помеченный таким действием a_i, которое можно выполнить в текущий момент времени (если таких переходов нет, то процесс временно приостанавливает свою работу до того момента, когда появится хотя бы один такой переход),
  - выполняет действие a_i, после чего переходит в состоние s_{i+1}, которое является концом выбранного перехода,
- если в R нет переходов с началом в s_i, то P заканчивает работу.

Ниже для каждого процесса P запись $Act(P)$ будет обозначать множество невнутренних действий процесса P:

$$Act(P) = \{a \in Act \setminus \{\tau\} \mid \exists\, s \xrightarrow{a} s' \in R\}.$$

Процесс (S, s^0, R) называется **конечным**, если S – конечное множество. Конечный процесс можно изображать диаграммой, в которой

- каждому состоянию соответствует кружочек на плоскости, в котором м.б. написан идентификатор (имя этого состояния),
- каждому переходу соответствует стрелка из начала этого перехода в его конец, на которой написана метка этого перехода,
- начальное состояние обозначается двойным кружочком.

Состояние s процесса $P = (S, s^0, R)$ называется

- **достижимым**, если $s = s^0$ или существует последовательность переходов в P, имеющая вид $s^0 \xrightarrow{a_1} \ldots \xrightarrow{a_n} s$,
- **недостижимым**, если оно не является достижимым,
- **терминальным**, если не существует переходов с началом в s.

Будем считать процессы $P_1 = (S_1, s_1^0, R_1)$ и $P_2 = (S_2, s_2^0, R_2)$ равными, если они изоморфны как графы, т.е. существует биективное отображение $f : S_1 \to S_2$, такое, что $f(s_1^0) = s_2^0$, и для каждой пары вершин $s, s' \in S_1$ и каждого $a \in Act$ существует ребро $s \xrightarrow{a} s' \in R_1$ тогда и только тогда, когда существует ребро $f(s) \xrightarrow{a} f(s') \in R_2$.

Кроме того, будем считать процессы $P_1 = (S_1, s_1^0, R_1)$ и $P_2 = (S_2, s_2^0, R_2)$ равными, если P_2 получается из P_1 удалением недостижимых состояний и связанных с ними рёбер.

Процесс (S, s^0, R) называется **детерминированным**, если $\forall\, s \in S$, $\forall\, a \in Act$ существует не более одного $s' \in S$, такого, что $s \xrightarrow{a} s' \in R$.

4 Операции на процессах

В этом пункте мы определим операции на процессах, при помощи которых из одних процессов можно строить другие, более сложные процессы.

4.1 Префиксное действие

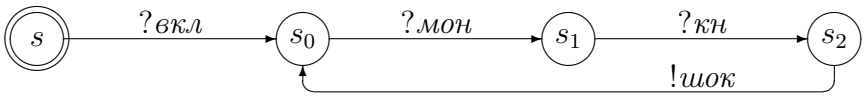
Первая такая операция – префиксное действие.

Пусть заданы процесс $P = (S, s^0, R)$ и действие $a \in Act$. Применением операции **префиксного действия** к паре (a, P) является процесс, обозначаемый записью $a; P$ и имеющий вид $(S \sqcup \{s\}, s, R \sqcup \{s \xrightarrow{a} s^0\})$, т.е.

- множество состояний процесса $a; P$ получается добавлением к S нового состояния s, которое будет начальным в $a; P$, и

- множество переходов процесса $a; P$ получается добавлением к R нового перехода $s \xrightarrow{a} s^0$.

Проиллюстрируем действие данной операции на примере торгового автомата из параграфа 1.2. Обозначим процесс, представляющий поведение этого автомата, записью $P_{\text{та}}$. Расширим множество действий данного автомата новым действием $?вкл$, которое будет означать включение этого автомата в сеть. Процесс $?вкл; P_{\text{та}}$ представляет поведение нового торгового автомата, который в начальном состоянии не может ни принимать монет, ни воспринимать нажатия на кнопку, ни выдавать шоколадок. Единственное, что он может, – это стать включенным, после этого его поведение ничем не будет отличаться от поведения исходного автомата. Графовое представление процесса $?вкл; P_{\text{та}}$ имеет вид

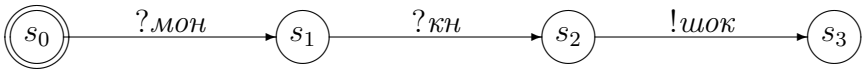


4.2 Пустой процесс

Среди всех процессов существует один наиболее простой. Этот процесс имеет всего одно состояние и не имеет переходов. Для обозначения такого процесса используется константа (т.е. нульарная операция) $\mathbf{0}$.

Возвращаясь к примерам с торговыми автоматами, можно сказать, что процесс $\mathbf{0}$ представляет поведение сломанного автомата, то есть такого автомата, который вообще не может выполнять никаких действий.

Путем применения операций префиксного действия к процессу $\mathbf{0}$ можно определять поведение более сложных автоматов. Рассмотрим, например, процесс $P = ?мон; ?кн; !шок; \mathbf{0}$. Графовое представление этого процесса имеет вид



Этот процесс задает поведение автомата, который обслуживает ровно одного покупателя и после этого ломается.

4.3 Альтернативная композиция

Следующая операция на процессах – альтернативная композиция. Данная операция используется в том случае, когда по паре процессов P_1, P_2 требуется построить процесс P, который будет функционировать либо

как процесс P_1, либо как процесс P_2, причём выбор процесса, в соответствии с которым P будет функционировать, может определяться как самим P, так и окружающей средой, в которой функционирует P.

Например, если P_1 и P_2 имеют вид $P_1 = ?\alpha; P_1'$, $P_2 = ?\beta; P_2'$ и в начальный момент времени окружающая среда может предложить P ввести объект α, но не может предложить P ввести объект β, то P должен выбрать то поведение, которое является единственно возможным в данной ситуации, т.е. работать так же, как процесс P_1.

Отметим, что в данном случае выбирается такой процесс, первое действие в котором может быть выполнено в текущий момент времени. Выбрав P_1 и выполнив действие $?\alpha$, процесс P обязан продолжать работу в соответствии со своим выбором, т.е. в соответствии с процессом P_1'.

Если же в начальный момент времени окружающая среда может предложить P ввести как объект α, так и объект β, то P недетерминированно (т.е. произвольно) выбирает процесс, в соответствии с которым он будет работать, или же этот выбор производится с учётом дополнительных факторов.

Операция альтернативной композиции определяется следующим образом. Пусть процессы P_1, P_2 имеют вид $P_i = (S_i, s_i^0, R_i)$ $(i = 1, 2)$, причём множества состояний S_1 и S_2 не имеют общих элементов (если S_1 и S_2 имеют общие элементы, то для определения процесса $P_1 + P_2$ в качестве второго слагаемого надо взять не сам P_2, а его изоморфную копию, которая дизъюнктна с P_1).

Альтернативная композиция процессов P_1 и P_2 – это процесс

$$P_1 + P_2 = (S_1 \sqcup S_2 \sqcup \{s^0\}, s^0, R),$$

где $R = R_1 \sqcup R_2 \sqcup \{s^0 \xrightarrow{a} s \mid s_i^0 \xrightarrow{a} s \in R_i, i = 1, 2\}$ (т.е. R получается добавлением к $R_1 \sqcup R_2$ переходов из начального состояния, дублирующих переходы из начальных состояний процессов-слагаемых).

Рассмотрим в качестве примера торговый автомат, который продаёт газированную воду, причём

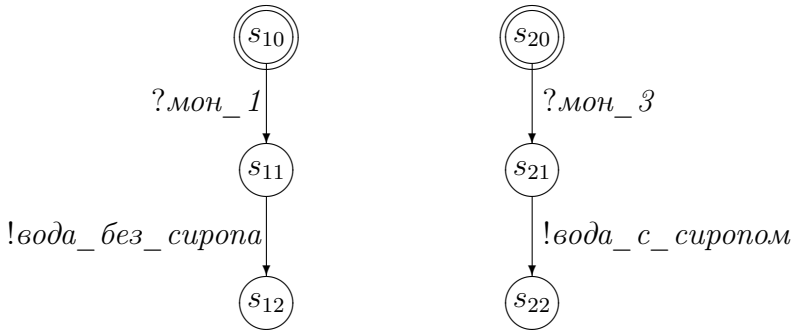
- если покупатель опускает в него монету *мон_ 1* достоинством в 1 копейку, то автомат выдаёт стакан воды без сиропа,
- а если покупатель опускает в него монету *мон_ 3* достоинством в 3 копейки, то автомат выдаёт стакан воды с сиропом

и сразу после продажи одного стакана воды автомат ломается.

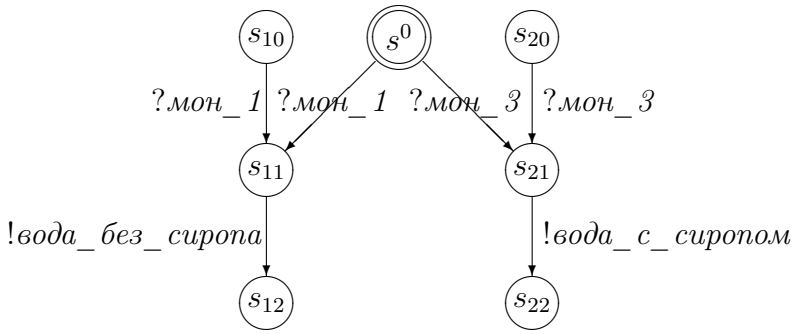
Поведение данного автомата описывается следующим процессом:

$$\begin{array}{rcl} P_{\text{газ\_вода}} & = & ?\textit{мон\_ 1}; !\textit{вода\_ без\_ сиропа}; \mathbf{0} \;\; + \\ & + & ?\textit{мон\_ 3}; !\textit{вода\_ с\_ сиропом}; \mathbf{0} \end{array} \qquad (2)$$

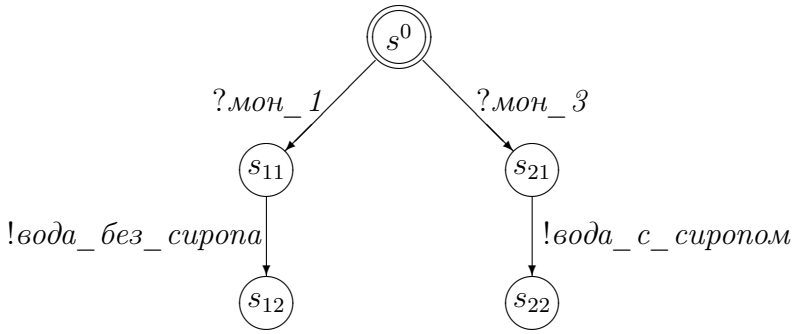
Графовые представления слагаемых в сумме (2) имеют вид



Согласно определению альтернативной композиции, графовое представление процесса (2) получается добавлением к предыдущей диаграмме нового состояния и переходов из этого состояния, дублирующих переходы из начальных состояний слагаемых, в результате чего получается следующая диаграмма:



Поскольку состояния s_{10} и s_{20} недостижимы, то их и связанные с ними переходы можно удалить, в результате чего получится диаграмма



которая и является искомым графовым представлением процесса (2).

Рассмотрим другой пример. Опишем разменный автомат, который может принимать купюры достоинством в 1000 рублей. Автомат должен

выдать либо 2 купюры по 500 рублей, либо 10 купюр по 100 рублей, причем выбор способа размена осуществляется независимо от желания клиента. Сразу после одного сеанса размена автомат ломается.

$$P_{\text{размен}} = ?\mathit{1\_no\_1000}; (!\mathit{2\_no\_500}; \mathbf{0} + !\mathit{10\_no\_100}; \mathbf{0})$$

На этих двух примерах видно, что альтернативная композиция может использоваться для описания как минимум двух принципиально различных ситуаций.

1. Во-первых, она может выражать зависимость поведения системы от поведения её окружения.

   Например, в случае автомата $P_{\text{газ\_вода}}$ реакция автомата определяется действием покупателя, а именно достоинством монеты, которую он ввел в автомат. В данном случае процесс, представляющий поведение моделируемого торгового автомата, является **детерминированным**, то есть его функционирование однозначно определяется входными действиями.

2. Во-вторых, на примере автомата $P_{\text{размен}}$ мы видим, что при одних и тех же входных действиях возможна различная реакция автомата. Это – пример **недетерминизма**, то есть неопределенности поведения системы. Неопределённость в поведении систем может происходить по крайней мере по двум причинам.

   (a) Во-первых, поведение систем может зависеть от **случайных факторов**. Такими факторами могут быть, например, сбои в аппаратуре, коллизии в компьютерной сети, отсутствие купюр необходимого достоинства в банкомате или что-либо еще.

   (b) Во-вторых, модель всегда есть некоторая абстракция или упрощение реальной системы. А при этом некоторые факторы, которые влияют на поведение этой системы, могут быть просто исключены из рассмотрения.

   В частности, на примере процесса $P_{\text{размен}}$ мы видим, что реальная причина выбора варианта поведения автомата может не учитываться в процессе, представляющем модель поведения этого автомата.

4.4 Параллельная композиция

Операция параллельной композиции используется для построения моделей поведения динамических систем, состоящих из взаимодействующих компонентов.

Если система состоит из двух компонентов, поведение которых описывается процессами P_1 и P_2, и функционирование системы заключается в совместном функционировании её компонентов, то поведение этой системы описывается процессом, который называется **параллельной композицией** процессов P_1 и P_2 и определяется следующим образом.

Пусть процессы P_1 и P_2 имеют вид $P_i = (S_i, s_i^0, R_i)$ $(i = 1, 2)$. **Параллельной композицией** процессов P_1 и P_2 называется процесс

$$P_1 \,|\, P_2 = (S_1 \times S_2, (s_1^0, s_2^0), R),$$

где R содержит следующие переходы:

- переход $(s_1, s_2) \xrightarrow{a} (s_1', s_2)$, если $s_1 \xrightarrow{a} s_1' \in R_1$, $s_2 \in S_2$,
- переход $(s_1, s_2) \xrightarrow{a} (s_1, s_2')$, если $s_1 \in S_1$, $s_2 \xrightarrow{a} s_2' \in R_2$,
- переход $(s_1, s_2) \xrightarrow{\tau} (s_1', s_2')$, если $s_1 \xrightarrow{a} s_1' \in R_1$, $s_2 \xrightarrow{\overline{a}} s_2' \in R_2$.

Таким образом, каждое выполнение действия процессом $P_1|P_2$ представляет собой

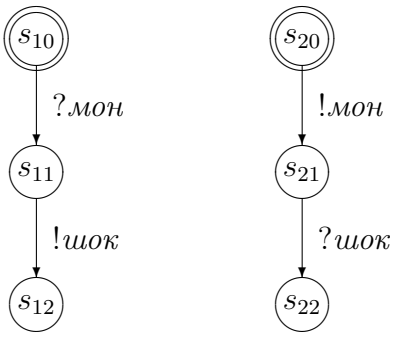
- либо одиночное выполнение действия одним из процессов P_1, P_2, во время выполнения которого другой из этих процессов приостанавливает свою работу,
- либо одновременное совместное выполнение действий обоими процессами P_1, P_2, которое заключается в том, что
  - один из этих процессов (P_i) передаёт другому процессу (P_j) некоторый объект и
  - процесс P_j в тот же самый момент времени принимает от процесса P_i этот объект,

  такой вид совместного выполнения называется **синхронным взаимодействием**.

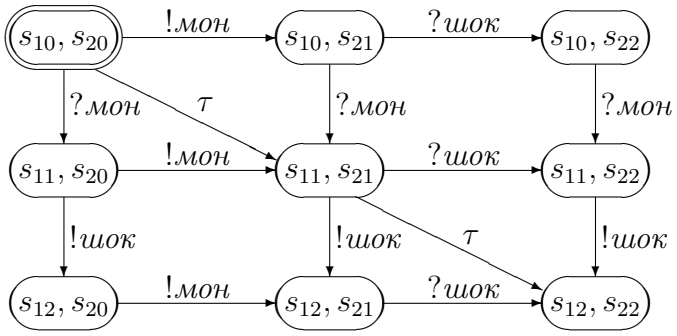
Приведём в качестве примера процесс $P_1 \,|\, P_2$, где процессы P_1 и P_2 представляют поведение торгового автомата и покупателя:

- $P_1 = ?\mathit{мон}; !\mathit{шок}; \mathbf{0}$, т.е. P_1 получает монету, выдаёт шоколадку и после этого ломается,
- $P_2 = !\mathit{мон}; ?\mathit{шок}; \mathbf{0}$, т.е. P_2 опускает монету, получает шоколадку и после этого завершает свою работу.

Графовые представления P_1 и P_2 имеют вид



Графовое представление процесса $P_1|P_2$ имеет вид



В данной диаграмме представлены все возможные варианты совместного функционирования P_1 и P_2. Каждое действие процесса $P_1|P_2$ м.б.

- синхронным взаимодействием: первое выполнение действия процессом $P_1|P_2$ м.б. результатом одновременного выполнения действий автоматом и покупателем (переход $(s_{10}, s_{20}) \xrightarrow{\tau} (s_{11}, s_{21})$: покупатель опускает в автомат монету, и автомат её принимает), или
- одиночным выполнением действия каким-либо из процессов P_1, P_2, например
  - переход $(s_{10}, s_{20}) \xrightarrow{!мон} (s_{10}, s_{21})$ интерпретируется следующим образом: покупатель опускает в автомат монету, но эта монета по каким-либо причинам не принимается автоматом, или
  - переход $(s_{11}, s_{21}) \xrightarrow{!шок} (s_{12}, s_{21})$ интерпретируется следующим образом: автомат выдал шоколадку, но покупатель по каким-либо причинам её не получил (например, к автомату подошёл вор и взял эту шоколадку, прежде чем её смог взять покупатель).

4.5 Ограничение

Пусть заданы процесс $P = (S, s^0, R)$ и подмножество $L \subseteq Names$.

Ограничением P по L называется процесс

$$P \setminus L = (S, s^0, \{s \xrightarrow{a} s' \mid a = \tau, \text{ или } name(a) \notin L\}),$$

т.е. $P \setminus L$ получается из P удалением переходов, метки которых содержат имена из L.

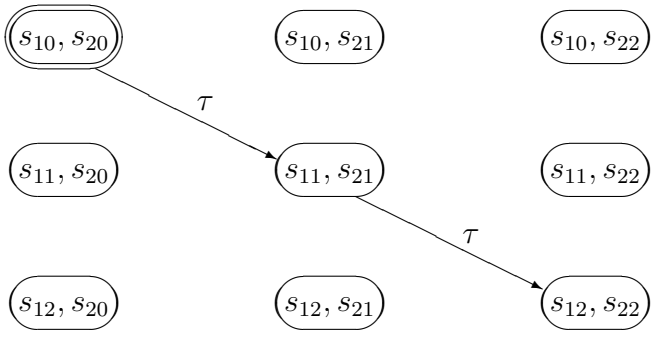
Операция ограничения используется, как правило, совместно с операцией параллельной композиции, для представления процессов, состоящих из нескольких компонентов, взаимодействие между которыми должно удовлетворять ограничениям. Эти ограничения заключаются в невозможности одиночного выполнения некоторых действий.

Например, пусть P_1 и P_2 – торговый автомат и покупатель, которые рассматривались в предыдущем параграфе. Мы хотели бы описать процесс, являющийся моделью такого параллельного функционирования процессов P_1 и P_2, при котором эти процессы могут совершать действия, связанные с покупкой-продажей шоколадки, только совместно.

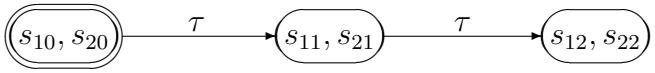
Искомый процесс имеет вид

$$P \stackrel{\text{def}}{=} (P_1 | P_2) \setminus \{мон, шок\} \tag{3}$$

Графовое представление процесса (3) имеет вид



После удаления недостижимых состояний получится процесс, имеющий следующее графовое представление:

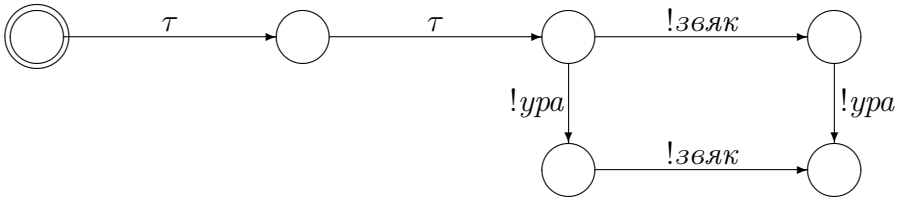


Рассмотрим другой пример. Немного изменим определения торгового автомата и покупателя: пусть они еще и сигнализируют об успешном

выполнении своей работы, т.е. их процессы имеют следующий вид:

$$P_1 \stackrel{\text{def}}{=} ?\textit{мон}; !\textit{шок}; !\textit{звяк}; \mathbf{0}, \quad P_2 \stackrel{\text{def}}{=} !\textit{мон}; ?\textit{шок}; !\textit{ура}; \mathbf{0}$$

В этом случае графовое представление процесса (3) после удаления недостижимых состояний имеет вид



Данный процесс допускает одиночное выполнение тех невнутренних действий, которые не связаны с покупкой и продажей шоколадки.

4.6 Переименование

Пусть заданы процесс $P = (S, s^0, R)$ и функция $\theta : Names \to Names$.

Переименованием P относительно θ называется процесс

$$P^\theta = (S, s^0, \{s \stackrel{a^\theta}{\to} s' \mid s \stackrel{a}{\to} s' \in R\}),$$

где $(!\alpha)^\theta = !\theta(\alpha)$, $(?\alpha)^\theta = ?\theta(\alpha)$, $\tau^\theta = \tau$, т.е. P^θ получается из P заменой имён в метках переходов: каждое имя α заменяется на $\theta(\alpha)$.

Операция переименования позволяет многократно использовать один и тот же процесс P в качестве компоненты при построении более сложного процесса P'. Эта операция используется для предотвращения «конфликтов» между именами действий, используемых в различных вхождениях P в P'.

Если θ действует нетождественно лишь на имена $\alpha_1, \ldots, \alpha_n$ и отображает их в имена $\beta_1, \ldots, \beta_n$ соответственно, то для P^θ используется обозначение $P(\beta_1/\alpha_1, \ldots, \beta_n/\alpha_n)$.

5 Свойства операций на процессах

В этом параграфе мы приводим некоторые свойства определённых выше операций на процессах. В излагаемых ниже свойствах символы P, L и θ (возможно, с индексами) изображают произвольные процессы, множества имен и переименования соответственно.

1. $(P_1 + P_2) + P_3 = P_1 + (P_2 + P_3)$.

   Из данного свойства следует, что допустимы выражения вида

   $$P_1 + \ldots + P_n, \tag{4}$$

   так как при любой расстановке скобок в выражении (4) получится один и тот же процесс. Данный процесс можно описать явно следущим образом. Пусть процессы P_i $(i = 1, \ldots, n)$ имеют вид

   $$P_i = (S_i, s_i^0, R_i) \quad (i = 1, \ldots, n), \tag{5}$$

   причём множества $S_1, \ldots, S_n$ дизъюнктны. Тогда (4) имеет вид

   $$(S_1 \sqcup \ldots \sqcup S_n \sqcup \{s^0\}, s^0, R),$$

   где $\forall\, i = 1, \ldots, n$ R содержит все переходы из R_i и $\forall\, s_i^0 \xrightarrow{a} s \in R_i$ R содержит переход $s^0 \xrightarrow{a} s$.

2. $(P_1 \mid P_2) \mid P_3 = P_1 \mid (P_2 \mid P_3)$.

   Из данного свойства следует, что допустимы выражения вида

   $$P_1 \mid \ldots \mid P_n, \tag{6}$$

   так как при любой расстановке скобок в выражении (6) получится один и тот же процесс. Данный процесс можно описать явно следущим образом. Пусть процессы P_i $(i = 1, \ldots, n)$ имеют вид (5), тогда процесс (6) имеет вид

   $$P = (S_1 \times \ldots \times S_n, (s_1^0, \ldots, s_n^0), R),$$

   где R содержит следующие переходы:

   - $(s_1, \ldots, s_n) \xrightarrow{a} (s_1, \ldots, s_{i-1}, s_i', s_{i+1}, \ldots, s_n)$, где $s_i \xrightarrow{a} s_i' \in R_i$,
   - $(s_1, \ldots, s_n) \xrightarrow{\tau} (s_1, \ldots, s_{i-1}, s_i', s_{i+1}, \ldots, s_{j-1}, s_j', s_{j+1}, \ldots, s_n)$, где для некоторого $a \in Act \setminus \{\tau\}$ $s_i \xrightarrow{a} s_i' \in R_i$, $s_j \xrightarrow{\overline{a}} s_j' \in R_j$.

3. $P_1 + P_2 = P_2 + P_1$.

4. $P_1 \mid P_2 = P_2 \mid P_1$.

5. $P \mid \mathbf{0} = P$.

6. $\mathbf{0} \setminus L = \mathbf{0}$.

7. $\mathbf{0}^\theta = \mathbf{0}$.

8. $P \setminus L = P$, если $L \cap names(Act(P)) = \emptyset$.

9. $(a; P) \setminus L = \begin{cases} \mathbf{0}, & \text{если } a \neq \tau \text{ и } name(a) \in L, \\ a; (P \setminus L) & \text{иначе.} \end{cases}$

10. $(P_1 + P_2) \setminus L = (P_1 \setminus L) + (P_2 \setminus L)$.

11. $(P_1 \mid P_2) \setminus L = (P_1 \setminus L) \mid (P_2 \setminus L)$, если

$$L \cap names(Act(P_1) \cap \overline{Act(P_2)}) = \emptyset.$$

12. $(P \setminus L_1) \setminus L_2 = P \setminus (L_1 \cup L_2)$.

13. $P^\theta \setminus L = (P \setminus \theta^{-1}(L))^\theta$.

14. $P^{\theta_{id}} = P$, где θ_{id} – тождественная функция.

15. $P^\theta = P^{\theta'}$, если $\forall\, \alpha \in names(Act(P))\ \ \theta(\alpha) = \theta'(\alpha)$.

16. $(a; P)^\theta = a^\theta; (P^\theta)$.

17. $(P_1 + P_2)^\theta = P_1^\theta + P_2^\theta$.

18. $(P_1 \mid P_2)^\theta = P_1^\theta \mid P_2^\theta$, если сужение θ на $names(Act(P_1) \cup Act(P_2))$ является инъективной функцией.

19. $(P \setminus L)^\theta = P^\theta \setminus \theta(L)$, если функция θ инъективна.

20. $P^{\theta\theta'} = P^(\theta' \circ \theta)$.

6 Эквивалентности процессов

В этой главе мы излагаем несколько определений понятия эквивалентности процессов. Выбор того или иного варианта определения эквивалентности процессов для его применения в конкретной ситуации должен определяться тем, как именно в данной ситуации понимается одинаковость процессов. В параграфах 6.1 и 6.2 вводятся понятия простой и сильной эквивалентности процессов. Данные понятия используются в тех ситуациях, когда все действия, выполняющиеся в процессах, имеют одинаковый статус. В параграфах **??** и **??** предлагаются другие варианты понятия эквивалентности процессов: наблюдаемая эквивалентность

и наблюдаемая конгруэнция. Данные понятия используются в тех ситуациях, когда действие τ рассматривается как ненаблюдаемое.

С каждым из вариантов понятия эквивалентности процессов связаны естественные задачи: распознавание для двух заданных процессов, являются ли они эквивалентными и построение по заданному процессу P процесса P', имеющего минимальное число состояний среди всех процессов, которые эквивалентны P.

6.1 Простая эквивалентность

6.1.1 Поведение процесса

Будем использовать следующие обозначения: $\forall\, P = (S, s^0, R) \in Proc$

- $\forall\, s \in S$ запись P^s обозначает процесс (S, s, R) (т.е. P^s отличается от P только начальным состоянием),
- $\forall\, P' \in Proc$, $\forall\, a \in Act$ запись $P \xrightarrow{a} P'$ обозначает утверждение: в P имеется ребро вида $s^0 \xrightarrow{a} s$ и $P' = P^s$.

Утверждение $P \xrightarrow{a} P'$ может интерпретироваться следующим образом: P может выполнить действие a, после чего вести себя как P'.

Запись $P \xrightarrow{a} P'$ называется **переходом** из P.

Поведением процесса P называется дерево $Beh(P)$,

- каждая вершина v которого помечена некоторым процессом P^v,
- каждое ребро которого помечено некоторым действием из Act и
- для каждой вершины $v \in Beh(P)$ множество ребер, выходящих из v, находится во взаимно однозначном соответствии с множеством переходов из P^v: ребру $v \xrightarrow{a} v'$ соответствует переход вида $P^v \xrightarrow{a} P^{v'}$.

6.1.2 Понятие простой эквивалентности

Будем говорить, что процессы $P, P' \in Proc$ находятся в отношении **простой эквивалентности**, если деревья $Beh(P)$ и $Beh(P')$ изоморфны, т.е. существует биективное отображение f множества вершин дерева $Beh(P)$ на множество вершин дерева $Beh(P')$, такое, что $\forall\, v, v' \in Beh(P)$, $\forall\, a \in Act$ существует ребро $v \xrightarrow{a} v'$ тогда и только тогда, когда существует ребро $f(v) \xrightarrow{a} f(v')$.

$\forall\, P, P' \in Proc$ запись $P \equiv P'$ обозначает, что процессы P и P' находятся в отношении простой эквивалентности.

6.1.3 Примеры процессов, находящихся в отношении простой эквивалентности

В этом пункте мы приводим несколько примеров процессов, находящихся в отношении простой эквивалентности. Обоснование утверждений о простой эквивалентности этих процессов является несложным.

1. $P \equiv P'$, где P и P' имеют следующий вид:

2. Если графы процессов P и P' изоморфны или P' получается из P удалением недостижимых состояний, то $P \equiv P'$.

3. $\forall\, P \in Proc \;\; P + \mathbf{0} \equiv P$.

4. Если $P = (S, s^0, R)$ и список всех ребер, выходящих из s^0, имеет вид $\{s^0 \xrightarrow{a_i} s_i \mid i = 1, \ldots, n\}$, то $P \equiv a_1; P^{s_1} + \ldots + a_n; P^{s_n}$.

5. Пусть процесс P имеет вид $P_1| \ldots |P_n$, где

$$\forall\, i = 1, \ldots, n \quad P_i \equiv a_{i1}; P_{i1} + \ldots + a_{in_i}; P_{in_i}. \tag{7}$$

   Тогда $P \equiv P'$, где P' является суммой

   - всех процессов вида

$$a_{ij}; (P_1| \ldots |P_{i-1}|P_{ij}|P_{i+1}| \ldots |P_n),$$

     где $i = 1, \ldots, n,\; j = 1, \ldots, n_i$, и
   - всех процессов вида

$$\tau; (P_1| \ldots |P_{i-1}|P_{ik}|P_{i+1}| \ldots |P_{j-1}|P_{jl}|P_{j+1}| \ldots |P_n),$$

     где $1 \le i < j \le n, \quad a_{ik}, a_{jl} \neq \tau$, и $a_{ik} = \overline{a_{jl}}$.

6. **Теорема о разложении**: пусть P имеет вид $(P_1| \ldots |P_n)\backslash L$, причем верно (7). Тогда $P \equiv P'$, где P' является суммой

   - всех процессов вида

$$a_{ij}; \Big((P_1| \ldots |P_{i-1}|P_{ij}|P_{i+1}| \ldots |P_n) \setminus L\Big),$$

     где $i = 1, \ldots, n,\; j = 1, \ldots, n_i$, $a_{ij} = \tau$ или $name(a_{ij}) \notin L$, и
   - всех процессов вида

$$\tau; \Big((P_1| \ldots |P_{i-1}|P_{ik}|P_{i+1}| \ldots |P_{j-1}|P_{jl}|P_{j+1}| \ldots |P_n) \setminus L\Big),$$

     где $1 \le i < j \le n, \quad a_{ik}, a_{jl} \neq \tau$, и $a_{ik} = \overline{a_{jl}}$.

6.2 Сильная эквивалентность

6.2.1 Понятие сильной эквивалентности

Другим вариантом понятия эквивалентности процессов является **сильная эквивалентность**. Данное понятие основано на следующем понимании эквивалентности процессов: если мы рассматриваем процессы P_1 и P_2 как эквивалентные, то должно быть выполнено условие: если один из этих процессов (P_i) может выполнить некоторое действие $a \in Act$ и после этого вести себя как некоторый процесс P_i' (т.е. $P_i \xrightarrow{a} P_i'$), то другой процесс (P_j) должен обладать способностью выполнить то же самое действие a, после чего вести себя как процесс P_j' (т.е. $P_j \xrightarrow{a} P_j'$), который эквивалентен P_i'.

Обозначим символом $\mathcal{M}$ совокупность всех отношений μ на множестве $Proc$, удовлетворяющих условию: $\forall\,(P_1, P_2) \in \mu$

$$\begin{array}{l} \forall\, a \in Act, \forall\, P_1' : P_1 \xrightarrow{a} P_1' \;\; \exists\, P_2' : P_2 \xrightarrow{a} P_2' \text{ и } (P_1', P_2') \in \mu, \\ \forall\, a \in Act, \forall\, P_2' : P_2 \xrightarrow{a} P_2' \;\; \exists\, P_1' : P_1 \xrightarrow{a} P_1' \text{ и } (P_1', P_2') \in \mu. \end{array} \tag{8}$$

Естественно считать процессы P_1, P_2 эквивалентными, если существует хотя бы одно отношение $\mu \in \mathcal{M}$, содержащее пару (P_1, P_2). Данное свойство можно выразить соотношением $(P_1, P_2) \in \bigcup_{\mu\in\mathcal{M}} \mu$.

Определим $\sim$ как отношение $\bigcup_{\mu\in\mathcal{M}} \mu$. Данное отношение называется **сильной эквивалентностью**. Нетрудно видеть, что $\sim \in \mathcal{M}$.

Теорема 1
$\sim$ является отношением эквивалентности.

Доказательство.

- $\sim$ рефлексивно, т.к. $Id_{Proc} = \{(P, P) \mid P \in Proc\} \in \mathcal{M}$,
- $\sim$ симметрично, т.к. из того, что $\forall\, \mu \in \mathcal{M} \;\; \mu^{-1} \in \mathcal{M}$, следует, что

$$\sim^{-1} = (\bigcup_{\mu\in\mathcal{M}} \mu)^{-1} = \bigcup_{\mu\in\mathcal{M}} (\mu^{-1}) \subseteq \bigcup_{\mu\in\mathcal{M}} \mu = \sim,$$

- $\sim$ транзитивно, т.к. если $\mu_1, \mu_2 \in \mathcal{M}$, то $\mu_1 \circ \mu_2 \in \mathcal{M}$, поэтому

$$\sim \circ \sim = (\bigcup_{\mu_1\in\mathcal{M}} \mu_1) \circ (\bigcup_{\mu_2\in\mathcal{M}} \mu_2) = \bigcup_{\mu_1,\mu_2\in\mathcal{M}} (\mu_1 \circ \mu_2) \subseteq \bigcup_{\mu\in\mathcal{M}} \mu = \sim . \;\blacksquare$$

Процессы $P_1, P_2 \in Proc$ называются **сильно эквивалентными**, если $(P_1, P_2) \in \sim$. Если процессы P_1 и P_2 сильно эквивалентны, то этот факт обозначается записью $P_1 \sim P_2$.

Нетрудно доказать, что

$$\text{если } P \equiv P', \text{ то } P \sim P'. \tag{9}$$

6.2.2 Критерий сильной эквивалентности, основанный на понятии бимоделирования

Пусть заданы два процесса: $P_i = (S_i, s_i^0, R_i)$ $(i = 1, 2)$. **Бимоделированием (БМ)** между P_1 и P_2 называется отношение $\rho \subseteq S_1 \times S_2$, содержащее пару (s_1^0, s_2^0) и удовлетворяющее условию: $\forall\, (s_1, s_2) \in \rho$

$$\begin{array}{l} \forall\, a \in Act, \forall\, s_1' : s_1 \xrightarrow{a} s_1' \;\; \exists\, s_2' : s_2 \xrightarrow{a} s_2' \text{ и } (s_1', s_2') \in \rho, \\ \forall\, a \in Act, \forall\, s_2' : s_2 \xrightarrow{a} s_2' \;\; \exists\, s_1' : s_1 \xrightarrow{a} s_1' \text{ и } (s_1', s_2') \in \rho. \end{array} \tag{10}$$

Теорема 2
$P_1 \sim P_2$ тогда и только тогда, когда существует БМ между P_1 и P_2.

Доказательство.

1. Пусть $P_1 \sim P_2$. Определим отношение $\rho \subseteq S_1 \times S_2$:

$$\rho \stackrel{\text{def}}{=} \{(s_1, s_2) \in S_1 \times S_2 \mid P_1^{s_1} \sim P_2^{s_2}\}.$$

Докажем что ρ является БМ между P_1 и P_2.

Так как $P_1 = P_1^{s_1^0}$ и $P_2 = P_2^{s_2^0}$, то из $P_1 \sim P_2$ следует, что $(s_1^0, s_2^0) \in \rho$.

Докажем первое свойство в (10): из $(s_1, s_2) \in \rho$ и $s_1 \xrightarrow{a} s_1'$ следует

$$\exists\, s_2' : s_2 \xrightarrow{a} s_2', (s_1', s_2') \in \rho. \tag{11}$$

Из $(s_1, s_2) \in \rho$ следует $P_1^{s_1} \sim P_2^{s_2}$. Из $s_1 \xrightarrow{a} s_1'$ следует $P_1^{s_1} \xrightarrow{a} P_1^{s_1'}$.

Так как $\sim \in \mathcal{M}$, то $\exists\, P_2' : P_2^{s_2} \xrightarrow{a} P_2'$ и $P_1^{s_1'} \sim P_2'$.

Согласно определению понятия перехода из процесса, из $P_2^{s_2} \xrightarrow{a} P_2'$ следует, что $\exists\, s_2' : s_2 \xrightarrow{a} s_2'$ и $P_2' = P_2^{s_2'}$.

Из $P_1^{s_1'} \sim P_2^{s_2'}$ следует, что $(s_1', s_2') \in \rho$. Таким образом, верно (11).

Аналогично доказывается второе свойство в (10).

2. Пусть существует БМ ρ между P_1 и P_2. Докажем, что $P_1 \sim P_2$ путем построения отношения $\mu \in \mathcal{M}$, содержащего пару (P_1, P_2).

   Определим $\mu \stackrel{\text{def}}{=} \{(P_1^{s_1}, P_2^{s_2}) \mid (s_1, s_2) \in \rho\}$.

   Так как $(s_1^0, s_2^0) \in \rho$, то $(P_1, P_2) = (P_1^{s_1^0}, P_2^{s_2^0}) \in \mu$.

   Для доказательства соотношения $\mu \in \mathcal{M}$ докажем первое свойство в (8): из соотношений $(P_1^{s_1}, P_2^{s_2}) \in \mu$ и $P_1^{s_1} \stackrel{a}{\to} P_1'$ следует, что

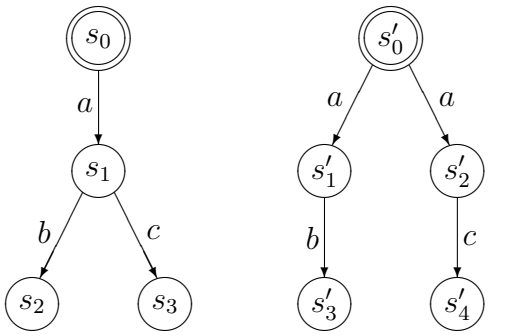
$$\exists\, P_2' : P_2^{s_2} \stackrel{a}{\to} P_2' \text{ и } (P_1', P_2') \in \mu. \tag{12}$$

   Так как $(s_1, s_2) \in \rho$ и из $P_1^{s_1} \stackrel{a}{\to} P_1'$ следует, что $\exists\, s_1' : s_1 \stackrel{a}{\to} s_1'$ и $P_1' = P_1^{s_1'}$, то по определению БМ $\exists\, s_2' : s_2 \stackrel{a}{\to} s_2'$, $(s_1', s_2') \in \rho$, поэтому в качестве P_2' в (12) можно взять $P_2^{s_2'}$.

   Аналогично доказывается второе свойство в (8). ■

Используя теорему 2, можно обосновать следующие утверждения.

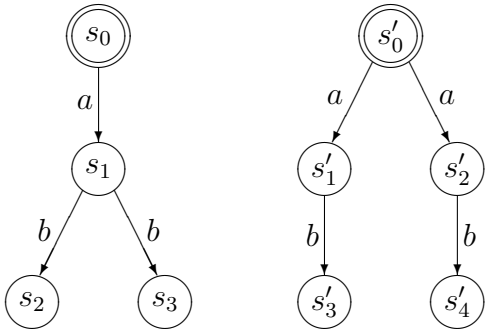
1. Процессы



(13)

   не являются сильно эквивалентными. Действительно, если они сильно эквивалентны, то по теореме 2 существует БМ ρ, содержащее пару (s_0, s_0'). Из $(s_0, s_0') \in \rho$ и $s_0' \stackrel{a}{\to} s_1'$ следует, что $\exists\, s : s_0 \stackrel{a}{\to} s$ и $(s, s_1') \in \rho$. Так как из s_0 выходит единственное ребро, то $s = s_1$. Из $(s_1, s_1') \in \rho$ и $s_1 \stackrel{c}{\to} s_3$ следует, что $\exists\, s : s_1' \stackrel{c}{\to} s$ и $(s_3, s) \in \rho$. Однако ребра вида $s_1' \stackrel{c}{\to} s$ не существует.

2. Процессы



сильно эквивалентны. БМ, обосновывающее это утверждение, имеет вид $\{(s_0, s'_0), (s_1, s'_1), (s_1, s'_2), (s_2, s'_3), (s_2, s'_4), (s_3, s'_3), (s_3, s'_4)\}$.

Межфакультетский курс «Формальная семантика и верификация программного обеспечения» Лекция 10 Бимоделирование. Критерий сильной эквивалентности процессов. Наблюдаемая эквивалентность

А.М.Миронов

1 Критерий сильной эквивалентности, основанный на понятии бимоделирования

Пусть заданы два процесса: $P_i = (S_i, s_i^0, R_i)$ $(i = 1, 2)$. **Бимоделированием (БМ)** между P_1 и P_2 называется отношение $\rho \subseteq S_1 \times S_2$, содержащее пару (s_1^0, s_2^0) и удовлетворяющее условию: $\forall\, (s_1, s_2) \in \rho$

$$\begin{array}{l} \forall\, a \in Act, \forall\, s_1' : s_1 \stackrel{a}{\to} s_1' \;\; \exists\, s_2' : s_2 \stackrel{a}{\to} s_2' \text{ и } (s_1', s_2') \in \rho, \\ \forall\, a \in Act, \forall\, s_2' : s_2 \stackrel{a}{\to} s_2' \;\; \exists\, s_1' : s_1 \stackrel{a}{\to} s_1' \text{ и } (s_1', s_2') \in \rho. \end{array} \qquad (1)$$

Теорема 1
$P_1 \sim P_2$ тогда и только тогда, когда существует БМ между P_1 и P_2.

Доказательство.

1. Пусть $P_1 \sim P_2$. Определим отношение $\rho \subseteq S_1 \times S_2$:

$$\rho \stackrel{\text{def}}{=} \{(s_1, s_2) \in S_1 \times S_2 \mid P_1^{s_1} \sim P_2^{s_2}\}.$$

Докажем что ρ является БМ между P_1 и P_2.

Так как $P_1 = P_1^{s_1^0}$ и $P_2 = P_2^{s_2^0}$, то из $P_1 \sim P_2$ следует, что $(s_1^0, s_2^0) \in \rho$.

Докажем первое свойство в (1): из $(s_1, s_2) \in \rho$ и $s_1 \stackrel{a}{\to} s_1'$ следует

$$\exists\, s_2' : s_2 \stackrel{a}{\to} s_2', (s_1', s_2') \in \rho. \tag{2}$$

Из $(s_1, s_2) \in \rho$ следует $P_1^{s_1} \sim P_2^{s_2}$. Из $s_1 \stackrel{a}{\to} s_1'$ следует $P_1^{s_1} \stackrel{a}{\to} P_1^{s_1'}$.

Так как $\sim \in \mathcal{M}$, то $\exists\, P_2' : P_2^{s_2} \stackrel{a}{\to} P_2'$ и $P_1^{s_1'} \sim P_2'$.

Согласно определению понятия перехода из процесса, из $P_2^{s_2} \stackrel{a}{\to} P_2'$ следует, что $\exists\, s_2' : s_2 \stackrel{a}{\to} s_2'$ и $P_2' = P_2^{s_2'}$.

Из $P_1^{s_1'} \sim P_2^{s_2'}$ следует, что $(s_1', s_2') \in \rho$. Таким образом, верно (2).

Аналогично доказывается второе свойство в (1).

2. Пусть существует БМ ρ между P_1 и P_2. Докажем, что $P_1 \sim P_2$ путем построения отношения $\mu \in \mathcal{M}$, содержащего пару (P_1, P_2).

   Определим $\mu \stackrel{\text{def}}{=} \{(P_1^{s_1}, P_2^{s_2}) \mid (s_1, s_2) \in \rho\}$.

   Так как $(s_1^0, s_2^0) \in \rho$, то $(P_1, P_2) = (P_1^{s_1^0}, P_2^{s_2^0}) \in \mu$.

   Для доказательства соотношения $\mu \in \mathcal{M}$ докажем, что из соотношений $(P_1^{s_1}, P_2^{s_2}) \in \mu$ и $P_1^{s_1} \stackrel{a}{\to} P_1'$ следует, что

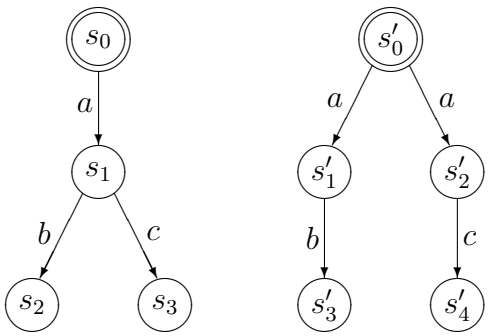
$$\exists\, P_2' : P_2^{s_2} \stackrel{a}{\to} P_2' \text{ и } (P_1', P_2') \in \mu. \tag{3}$$

   Так как $(s_1, s_2) \in \rho$ и из $P_1^{s_1} \stackrel{a}{\to} P_1'$ следует, что $\exists\, s_1' : s_1 \stackrel{a}{\to} s_1'$ и $P_1' = P_1^{s_1'}$, то по определению БМ $\exists\, s_2' : s_2 \stackrel{a}{\to} s_2', (s_1', s_2') \in \rho$, поэтому в качестве P_2' в (3) можно взять $P_2^{s_2'}$.

   Аналогично доказывается другое свойство из определения сильной эквивалентности. ■

Используя теорему 1, можно обосновать следующие утверждения.

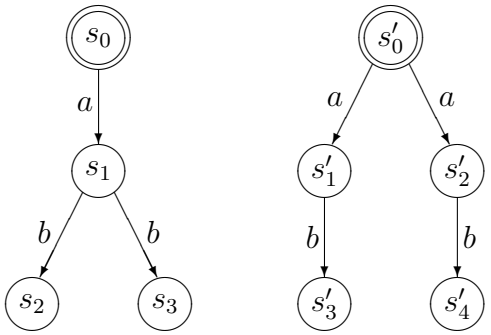
1. Процессы



(4)

не являются сильно эквивалентными. Действительно, если они сильно эквивалентны, то по теореме 1 существует БМ ρ, содержащее пару (s_0, s'_0). Из $(s_0, s'_0) \in \rho$ и $s'_0 \xrightarrow{a} s'_1$ следует, что $\exists\, s : s_0 \xrightarrow{a} s$ и $(s, s'_1) \in \rho$. Так как из s_0 выходит единственное ребро, то $s = s_1$. Из $(s_1, s'_1) \in \rho$ и $s_1 \xrightarrow{c} s_3$ следует, что $\exists\, s : s'_1 \xrightarrow{c} s$ и $(s_3, s) \in \rho$. Однако ребра вида $s'_1 \xrightarrow{c} s$ не существует.

2. Процессы



сильно эквивалентны. БМ, обосновывающее это утверждение, имеет вид $\{(s_0, s'_0), (s_1, s'_1), (s_1, s'_2), (s_2, s'_3), (s_2, s'_4), (s_3, s'_3), (s_3, s'_4)\}$.

2 Алгебраические свойства сильной эквивалентности

Теорема 2

$\sim$ является конгруэнцией, т.е. если $P_1 \sim P_2$, то

- $\forall\, a \in Act \quad a; P_1 \sim a; P_2$,
- $\forall\, P \in Proc \quad P_1 + P \sim P_2 + P$,
- $\forall\, P \in Proc \quad P_1 | P \sim P_2 | P$,
- $\forall\, L \subseteq Names \quad P_1 \setminus L \sim P_2 \setminus L$,
- $\forall\, \theta : Names \to Names \quad P_1\theta \sim P_2\theta$.

Доказательство.

Пусть процесс P_i имеет вид $(S_i, s_i^0, R_i) \;\; (i = 1, 2)$. Согласно теореме 1, из $P_1 \sim P_2$ следует, что существует БМ $\rho \subseteq S_1 \times S_2$ между P_1 и P_2. Используя это ρ, определим БМ для обоснования каждого из вышеприведённых соотношений.

- $\{(s^0_{(1)}, s^0_{(2)})\} \cup \rho$ – БМ между $a; P_1$ и $a; P_2$, где $s^0_{(i)}$ – начальное состояние процесса $a; P_i \ \ (i = 1, 2)$.

- Пусть процесс P имеет вид (S, s^0, R), тогда

  - $\{(s^0_{(1)}, s^0_{(2)})\} \cup \rho \cup \{(s, s) \mid s \in S\}$ – БМ между $P_1 + P$ и $P_2 + P$, где $s^0_{(i)}$ – начальное состояние процесса $P_i + P \ \ (i = 1, 2)$,
  - $\{((s_1, s), (s_2, s)) \mid (s_1, s_2) \in \rho, s \in S\}$ – БМ между $P_1|P$ и $P_2|P$.

- ρ является БМ между $P_1 \setminus L$ и $P_2 \setminus L$ и между P_1^θ и P_2^θ. ■

Теорема 3
Для каждого процесса P верно соотношение $P + P \sim P$.

Доказательство.
Пусть P имеет вид (S, s^0, R). Будем рассматривать процессы в левой части доказываемого соотношения как две дизъюнктные копии процесса P, состояния в этих копиях имеют вид $s_{(i)}$, где $s \in S, i = 1, 2$.

БМ между $P + P$ и P имеет вид $\{(s^0_0, s^0)\} \cup \{(s_{(i)}, s) \mid s \in S, \ i = 1, 2\}$, где s^0_0 – начальное состояние процесса $P + P$. ■

3 Распознавание сильной эквивалентности

В этом пункте рассматривается следующая задача: определить для заданых процессов $P_1, P_2 \in Proc$ верно ли, что $P_1 \sim P_2$.

Пусть $P_i = (S_i, s^0_i, R_i)$ $(i = 1, 2)$. Определим функцию $'$ на множестве всех отношений из S_1 в S_2, которая сопоставляет каждому отношению $\rho \subseteq S_1 \times S_2$ отношение $\rho' \subseteq S_1 \times S_2$, определяемое следующим образом:

$$\rho' \stackrel{\text{def}}{=} \{(s_1, s_2) \in S_1 \times S_2 \mid \text{ верно (1)}\}.$$

Очевидно, что функция $'$ монотонна, т.е. если $\rho_1 \subseteq \rho_2$, то $\rho'_1 \subseteq \rho'_2$.

Нетрудно видеть, что отношение $\rho \subseteq S_1 \times S_2$ является БМ между P_1 и P_2 тогда и только тогда, когда $(s^0_1, s^0_2) \in \rho$ и $\rho \subseteq \rho'$.

Определим ρ_{max} как объединение всех отношений из совокупности

$$\{\rho \subseteq S_1 \times S_2 \mid \rho \subseteq \rho'\}. \tag{5}$$

$\rho_{max} \in (5)$, т.к. $\forall\, \rho \in (5)$ из включения $\rho \subseteq (\bigcup\limits_{\rho \in (5)} \rho) = \rho_{max}$ и монотонности функции $'$ следует, что $\forall\, \rho \in (5)\, \rho \subseteq \rho' \subseteq \rho'_{max}$, поэтому $\rho_{max} = \bigcup\limits_{\rho \in (5)} \rho \subseteq \rho'_{max}$, т.е. $\rho_{max} \in (5)$.

Из включения $\rho_{max} \subseteq \rho'_{max}$ и монотонности функции $'$ следует включение $\rho'_{max} \subseteq \rho''_{max}$, т.е. $\rho'_{max} \in (5)$, откуда, в силу максимальности ρ_{max}, следует включение $\rho'_{max} \subseteq \rho_{max}$. Поэтому $\rho_{max} = \rho'_{max}$.

Таким образом, отношение ρ_{max} является наибольшим элементом совокупности (5) и наибольшей неподвижной точкой функции $'$. Будем обозначать это отношение записью $\rho(P_1, P_2)$.

Из теоремы 1 следует, что

$$P_1 \sim P_2 \;\Leftrightarrow\; (s_1^0, s_2^0) \in \rho(P_1, P_2), \tag{6}$$

откуда следует, что $\rho(P_1, P_2) = \{(s_1, s_2) \in S_1 \times S_2 \mid P_1^{s_1} \sim P_2^{s_2}\}$.

Нетрудно доказать, что

$$\forall\, P_1, P_2, P_3 \in Proc \;\; \rho(P_1, P_2) \circ \rho(P_2, P_3) \subseteq \rho(P_1, P_3). \tag{7}$$

Согласно (6), доказательство соотношения $P_1 \sim P_2$ сводится к вычислению $\rho(P_1, P_2)$ и проверке правой части (6). Ниже излагается алгоритм вычисления $\rho(P_1, P_2)$ (в случае когда P_1 и P_2 конечны).

Пусть $\{\rho_i \mid i \geq 1\}$ – последовательность отношений, определяемая следующим образом: $\rho_1 \stackrel{\text{def}}{=} S_1 \times S_2$, $\forall\, i \geq 1 \quad \rho_{i+1} \stackrel{\text{def}}{=} \rho'_i$.

Из включения $\rho_1 \supseteq \rho_2$ и монотонности функции $'$ следует, что

$$\rho_2 = \rho'_1 \supseteq \rho'_2 = \rho_3, \;\; \rho_3 = \rho'_2 \supseteq \rho'_3 = \rho_4 \;\text{ и т.д.}$$

Таким образом, последовательность $\{\rho_i \mid i \geq 1\}$ монотонна: $\rho_1 \supseteq \rho_2 \supseteq \ldots$

Поскольку все члены последовательности $\{\rho_i \mid i \geq 1\}$ являются подмножествами конечного множества $S_1 \times S_2$, то данная последовательность не может бесконечно убывать, она стабилизируется на некотором члене, т.е. $\exists\, i \geq 1 : \rho_i = \rho_{i+1} = \rho_{i+2} = \ldots$. Докажем, что член ρ_i, на котором наступает стабилизация, совпадает с $\rho(P_1, P_2)$.

Включение $\rho_i \subseteq \rho(P_1, P_2)$ следует из равенств $\rho_i = \rho_{i+1} = \rho'_i$ и того, что $\rho(P_1, P_2)$ – наибольшая неподвижная точка функции $'$.

Обратное включение $\rho(P_1, P_2) \subseteq \rho_i$ следует из того, что $\forall\, j \geq 1$ верно включение $\rho(P_1, P_2) \subseteq \rho_j$, т.к. данное включение верно для $j = 1$, и если оно верно для некоторого j, то в силу монотонности функции $'$ имеем:

$$\rho(P_1, P_2) = \rho(P_1, P_2)' \subseteq \rho'_j = \rho_{j+1},$$

т.е. данное включение верно для $j + 1$.

Для вычисления членов последовательности $\{\rho_i \mid i \geq 1\}$ можно использовать следующий алгоритм, который по отношению $\rho \subseteq S_1 \times S_2$

вычисляет отношение ρ':

$\rho' := \emptyset$
цикл для каждого $(s_1, s_2) \in \rho$
| включить $:= \top$
| **цикл для каждого** $s_1', a : s_1 \xrightarrow{a} s_1'$
| | найдено $:= \bot$
| | **цикл для каждого** $s_2' : s_2 \xrightarrow{a} s_2'$
| | | найдено $:=$ найдено $\vee (s_1', s_2') \in \rho$
| | **конец цикла**
| | включить $:=$ включить $\wedge$ найдено
| **конец цикла**
| **цикл для каждого** $s_2', a : s_2 \xrightarrow{a} s_2'$
| | найдено $:= \bot$
| | **цикл для каждого** $s_1' : s_1 \xrightarrow{a} s_1'$
| | | найдено $:=$ найдено $\vee (s_1', s_2') \in \rho$
| | **конец цикла**
| | включить $:=$ включить $\wedge$ найдено
| **конец цикла**
| **если** включить **то** $\rho' := \rho' \cup \{(s_1, s_2)\}$
конец цикла

Данный алгоритм корректен, только если $\rho' \subseteq \rho$ (что верно в том случае, когда этот алгоритм используется для вычисления членов последовательности $\{\rho_i \mid i \geq 1\}$). В общей ситуации внешний цикл должен иметь вид

цикл для каждого $(s_1, s_2) \in S_1 \times S_2$.

Оценим сложность данного алгоритма. Обозначим символом A число

$$A \stackrel{\text{def}}{=} \max(|Act(P_1)|, |Act(P_2)|) + 1.$$

Внешний цикл делает не более $|S_1| \cdot |S_2|$ итераций, и оба цикла, содержащихся во внешнем цикле, делают не более $|S_1| \cdot |S_2| \cdot A$ итераций, поэтому сложность этого алгоритма – $O(|S_1|^2 \cdot |S_2|^2 \cdot A)$.

Поскольку для вычисления члена последовательности $\{\rho_i \mid i \geq 1\}$, на котором наступает её стабилизация, нужно вычислить не больше чем $|S_1| \cdot |S_2|$ членов этой последовательности, то искомое отношение $\rho(P_1, P_2)$ м.б. вычислено за время $O(|S_1|^3 \cdot |S_2|^3 \cdot A)$.

4 Наблюдаемая эквивалентность

4.1 Определение наблюдаемой эквивалентности

Ещё одним вариантом понятия эквивалентности процессов является **наблюдаемая эквивалентность**. Данное понятие используется в тех ситуациях, когда действие τ рассматривается как ненаблюдаемое.

Введём вспомогательные обозначения. Пусть $P, P' \in Proc$.

1. Запись $P \stackrel{\tau^*}{\to} P'$ означает, что либо $P = P'$, либо существует последовательность процессов $P_1, \ldots, P_n$ таких, что $P_1 = P, P_n = P'$, и $\forall\, i = 1, \ldots, n-1 \;\; P_i \stackrel{\tau}{\to} P_{i+1}$,

   $P \stackrel{\tau^*}{\to} P'$ можно интерпретировать как утверждение о том, что P может незаметным для наблюдателя образом превратиться в P'.

2. $\forall\, a \in Act$ запись $P \stackrel{\tau^* a \tau^*}{\longrightarrow} P'$ означает, что существуют процессы P_1 и P_2 со следующими свойствами: $P \stackrel{\tau^*}{\to} P_1$, $P_1 \stackrel{a}{\to} P_2$, $P_2 \stackrel{\tau^*}{\to} P'$.

   Если $a \neq \tau$, то $P \stackrel{\tau^* a \tau^*}{\longrightarrow} P'$ можно интерпретировать как утверждение о том, что процесс P может эволюционировать так, что внешним проявлением этой эволюции будет лишь исполнение действия a, после чего вести себя как процесс P'.

Если имеет место $P \stackrel{\tau^* a \tau^*}{\longrightarrow} P'$, где $a \in Act \setminus \{\tau\}$, то будем говорить, что P может **наблюдаемо выполнить** a и после этого вести себя как P'.

Понятие наблюдаемой эквивалентности основано на следующем понимании эквивалентности процессов: если мы рассматриваем процессы P_1 и P_2 как эквивалентные, то должны быть выполнены условия:

- если один из этих процессов (P_i) может незаметно превратиться в некоторый процесс P_i', то и другой процесс (P_j) должен обладать способностью незаметно превратиться в такой процесс P_j', который эквивалентен P_i', и

- если один из этих процессов (P_i) может наблюдаемо выполнить действие $a \in Act \setminus \{\tau\}$ и после этого вести себя как некоторый процесс P_i' (т.е. $P_i \stackrel{\tau^* a \tau^*}{\longrightarrow} P_i'$), то другой процесс ($P_j$) должен обладать способностью наблюдаемо выполнить то же действие a, после чего вести себя как такой процесс P_j', который эквивалентен P_i'.

Определим $\mathcal{M}_\tau$ как множество всех отношений $\mu \subseteq Proc \times Proc$, удовлетворяющих условию: $\forall\,(P_1, P_2) \in \mu$

$$\begin{array}{l}
\forall\, P_1' : P_1 \stackrel{\tau}{\to} P_1' \;\; \exists\, P_2' : P_2 \stackrel{\tau^*}{\to} P_2' \text{ и } (P_1', P_2') \in \mu, \\
\forall\, P_2' : P_2 \stackrel{\tau}{\to} P_2' \;\; \exists\, P_1' : P_1 \stackrel{\tau^*}{\to} P_1' \text{ и } (P_1', P_2') \in \mu, \\
\forall\, a \in Act \setminus \{\tau\}, \forall\, P_1' : P_1 \stackrel{a}{\to} P_1' \;\; \exists\, P_2' : P_2 \stackrel{\tau^* a \tau^*}{\longrightarrow} P_2' \text{ и } (P_1', P_2') \in \mu, \\
\forall\, a \in Act \setminus \{\tau\}, \forall\, P_2' : P_2 \stackrel{a}{\to} P_2' \;\; \exists\, P_1' : P_1 \stackrel{\tau^* a \tau^*}{\longrightarrow} P_1' \text{ и } (P_1', P_2') \in \mu.
\end{array} \tag{8}$$

Будем считать P_1 и P_2 наблюдаемо эквивалентными, если существует хотя бы одно отношение $\mu \in \mathcal{M}_\tau$, содержащее пару (P_1, P_2). Данное свойство можно выразить соотношением $(P_1, P_2) \in \bigcup_{\mu \in \mathcal{M}_\tau} \mu$.

Определим $\approx$ как отношение $\bigcup_{\mu \in \mathcal{M}_\tau} \mu$. Данное отношение называется **наблюдаемой эквивалентностью**. Нетрудно видеть, что $\approx \in \mathcal{M}_\tau$.

Нетрудно доказать, что $\approx$ является отношением эквивалентности, доказательство этого утверждения получается из доказательства теоремы **??** путем замены $\mathcal{M}$ на $\mathcal{M}_\tau$.

Процессы $P_1, P_2 \in Proc$ называются **наблюдаемо эквивалентными**, если $(P_1, P_2) \in \approx$. Если процессы P_1 и P_2 наблюдаемо эквивалентны, то этот факт обозначается знакосочетанием $P_1 \approx P_2$.

Нетрудно доказать, что если $P_1 \sim P_2$, то $P_1 \approx P_2$.

4.2 Критерий наблюдаемой эквивалентности, основанный на понятии наблюдаемого бимоделирования

Для отношения $\approx$ также имеет место аналог критерия, основанного на понятии БМ (теорема 1 из параграфа 1). Для его формулировки мы введём вспомогательные обозначения.

Пусть $P = (S, s^0, R) \in Proc$ и $s_1, s_2 \in S$, тогда

- запись $s \stackrel{\tau^*}{\to} s'$ означает, что либо $s = s'$, либо существуют состояния $s_1, \ldots, s_n$, такие, что $s_1 = s,\ s_n = s'$ и $\forall\, i = 1, \ldots, n-1\ s_i \stackrel{\tau}{\to} s_{i+1}$,
- $\forall\, a \in Act \setminus \{\tau\}$ запись $s \stackrel{\tau^* a \tau^*}{\longrightarrow}$ означает, что существуют состояния s_1 и s_2 со следующими свойствами: $s \stackrel{\tau^*}{\to} s_1$, $s_1 \stackrel{a}{\to} s_2$, $s_2 \stackrel{\tau^*}{\to} s'$.

Пусть заданы процессы $P_i = (S_i, s_i^0, R_i)\ (i = 1, 2)$. **Наблюдаемое бимоделирование (НБМ)** между P_1 и P_2 – это отношение $\rho \subseteq S_1 \times S_2$,

содержащее пару (s_1^0, s_2^0) и удовлетворяющее условию: $\forall\,(s_1, s_2) \in \rho$

$$\begin{cases} \forall\, s_1' : s_1 \xrightarrow{\tau} s_1' \;\; \exists\, s_2' : s_2 \xrightarrow{\tau^*} s_2' \text{ и } (s_1', s_2') \in \rho, \\ \forall\, s_2' : s_2 \xrightarrow{\tau} s_2' \;\; \exists\, s_1' : s_1 \xrightarrow{\tau^*} s_1' \text{ и } (s_1', s_2') \in \rho, \\ \forall\, a \in Act \setminus \{\tau\}, \forall\, s_1' : s_1 \xrightarrow{a} s_1' \;\; \exists\, s_2' : s_2 \xrightarrow{\tau^* a \tau^*} s_2' \text{ и } (s_1', s_2') \in \rho, \\ \forall\, a \in Act \setminus \{\tau\}, \forall\, s_2' : s_2 \xrightarrow{a} s_2' \;\; \exists\, s_1' : s_1 \xrightarrow{\tau^* a \tau^*} s_1' \text{ и } (s_1', s_2') \in \rho. \end{cases} \tag{9}$$

Нетрудно доказать, что $P_1 \approx P_2$ тогда и только тогда, когда существует НБМ между P_1 и P_2 (доказательство этого утверждения выглядит аналогично доказательству теоремы 1).

Используя вышеупомянутое утверждение, можно доказать, что

$$\forall\, P \in Proc \quad P \;\approx\; \tau; P. \tag{10}$$

4.3 Алгебраические свойства наблюдаемой эквивалентности

Теорема 4

$\approx$ сохраняется всеми операциями на процессах, за исключением операции $+$, т.е. если $P_1 \approx P_2$, то

- $\forall\, a \in Act \quad a; P_1 \approx a; P_2$,
- $\forall\, P \in Proc \quad P_1 | P \approx P_2 | P$,
- $\forall\, L \subseteq Names \quad P_1 \setminus L \approx P_2 \setminus L$,
- $\forall\, \theta : Names \to Names \quad P_1^\theta \approx P_2^\theta$. ■

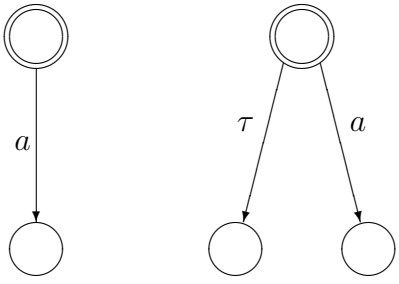
Доказательство этой теоремы аналогично доказательству теоремы 2.

Приведем два примера несохраняемости отношения $\approx$ операцией $+$.

1. Согласно (10), $\mathbf{0} \;\approx\; \tau; \mathbf{0}$, однако если $a \in Act \setminus \{\tau\}$, то

$$\mathbf{0} + a; \mathbf{0} \;\not\approx\; \tau; \mathbf{0} + a; \mathbf{0}, \tag{11}$$

в чём нетрудно убедиться при рассмотрении графового представления левой и правой частей в (11):



2. Другой пример: если $a, b \in Act \setminus \{\tau\}$ и $a \neq b$, то

$$a;\mathbf{0} + b;\mathbf{0} \quad \not\approx \quad \tau;a;\mathbf{0} + \tau;b;\mathbf{0},$$

хотя $a;\mathbf{0} \approx \tau;a;\mathbf{0}$ и $b;\mathbf{0} \approx \tau;b;\mathbf{0}$.

4.4 Распознавание наблюдаемой эквивалентности

Для решения задачи распознавания для двух заданных конечных процессов, являются ли они наблюдаемо эквивалентными, может быть построена теория, аналогичная теории для сильной эквивалентности. Мы не будем детально излагать эту теорию, т.к. она почти дословно повторяет соответствующую теорию для случая $\sim$. В этой теории для каждой пары процессов $P_i = (S_i, s_i^0, R_i)$ $(i = 1, 2)$ определяется функция, сопоставляющая каждому $\rho \subseteq S_1 \times S_2$ отношение

$$\rho'_\tau \stackrel{\text{def}}{=} \{(s_1, s_2) \in S_1 \times S_2 \mid \text{ верно (9)}\}.$$

Нетрудно доказать, что эта функция монотонна и ρ является НБМ между P_1 и P_2 тогда и только тогда, когда $(s_1^0, s_2^0) \in \rho$ и $\rho \subseteq \rho'_\tau$.

Определим $\rho_\tau(P_1, P_2)$ как объединение всех отношений из совокупности $\{\rho \subseteq S_1 \times S_2 \mid \rho \subseteq \rho'_\tau\}$. Это отношение является наибольшим элементом данной совокупности и обладает свойством

$$P_1 \approx P_2 \quad \Leftrightarrow \quad (s_1^0, s_2^0) \in \rho_\tau(P_1, P_2).$$

Из определения отношения $\rho_\tau(P_1, P_2)$ вытекает, что оно состоит из всех пар $(s_1, s_2) \in S_1 \times S_2$, таких, что $P_1^{s_1} \approx P_2^{s_2}$.

При построении полиномиального алгоритма вычисления отношения $\rho_\tau(P_1, P_2)$, аналогичного алгоритму из параграфа 3, следует учитывать следующее соображение: всякий раз, когда для заданной пары s, s' состояний некоторого процесса P требуется проверить условие $s \stackrel{\tau^*}{\to} s'$, достаточно рассматривать последовательности переходов $s \stackrel{\tau}{\to} s_1 \stackrel{\tau}{\to} \ldots$, длина которых не превосходит числа состояний процесса P.

Межфакультетский курс «Формальная семантика и верификация программного обеспечения» Лекция 11 Планировщик процессов (пример)

А.М.Миронов

1 Планировщик

Будем использовать следующие определения и обозначения.

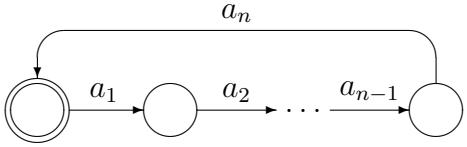
- Пусть задан процесс $P = (S, s^0, R)$. Запись Π_P^0 обозначает множество всех бесконечных путей в P вида

$$s^0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} s_2 \xrightarrow{a_3} \ldots \tag{1}$$

  Если путь $\pi \in \Pi_P^0$ имеет вид (1), то запись $Act(\pi)$ обозначает последовательность $a_1 a_2 a_3 \ldots$ меток переходов в π.

- Для каждой последовательности A действий из Act и каждого подмножества $L \subseteq Names$ запись A_L обозначает последовательность, получаемую из A удалением τ и действий с именами не из L.

- Пусть конечная последовательность A действий из Act имеет вид $a_1 \ldots a_n$. Тогда запись A^* обозначает процесс



  а также бесконечную последовательность $a_1 \ldots a_n a_1 \ldots a_n \ldots$.

Рассмотрим следующую задачу. Пусть задано множество действий

$$\{!\alpha_1, \ldots, !\alpha_n, !\beta_1, \ldots, !\beta_n\}, \tag{2}$$

где $\forall\, i = 1, \ldots, n$, действия $!\alpha_i$ и $!\beta_i$ интерпретируются как разрешение начать и кончить соответственно очередной сеанс работы для некоторого процесса P_i. Требуется построить детерминированный процесс Sch, называемый **планировщиком**, удовлетворяющий следующим условиям:

1. $Act(Sch) = (2)$, Sch не содержит терминальных состояний,
2. $\forall\, \pi \in \Pi^0_{Sch}$, если $A = Act(\pi)$, то

$$\left.\begin{array}{l} A_{\{\alpha_1,\ldots,\alpha_n\}} = (!\alpha_1 \ldots !\alpha_n)^* \\ \forall\, i = 1, \ldots, n \;\; A_{\{\alpha_i,\beta_i\}} = (!\alpha_i\, !\beta_i)^* \end{array}\right\} \qquad (3)$$

3. Для каждой последовательности A действий из (2), обладающей свойствами (3), существует путь $\pi \in \Pi^0_{Sch}$, такой, что $A = Act(\pi)$.

Свойства (3) выражают следующие требования:

- процессы $P_1, \ldots, P_n$ должны начинать свои новые сеансы только по очереди, в циклическом порядке,
- $\forall\, k \geq 1$ каждый из процессов $P_1, \ldots, P_n$ должен иметь возможность получить разрешение начать свой $(k+1)$-й сеанс только после того, как он завершит свой k-й сеанс, и не должен повторно завершать уже завершенный сеанс.

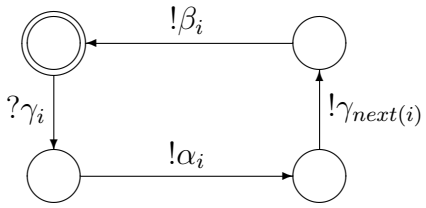
Свойства (3) можно выразить равносильным образом в виде наблюдаемой эквивалентности между некоторыми процессами.

В описании планировщика Sch мы будем использовать новые имена $\gamma_1, \ldots, \gamma_n$, обозначим множество этих имён символом Γ.

Искомый процесс Sch определяется следующим образом:

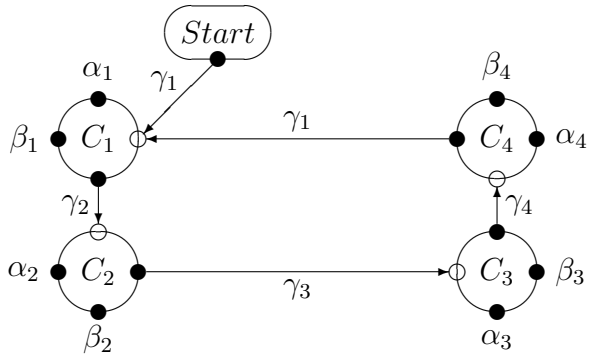
$$Sch \stackrel{\text{def}}{=} (Start \,|\, C_1 \,|\, \ldots \,|\, C_n) \setminus \Gamma, \qquad (4)$$

где $Start = !\gamma_1; \mathbf{0}$, $\forall\, i = 1, \ldots, n$ C_i имеет вид



и $next(i) = i + 1$ при $i < n$, $next(n) = 1$.

Потоковый граф процесса Sch в случае $n = 4$ имеет следующий вид:



Приведём неформальное пояснение функционирования процесса Sch.

Назовём процессы $C_1, \ldots, C_n$, участвующие в определении планировщика Sch, **циклерами**. Циклер C_i называется выключенным, если он находится в своём начальном состоянии, и включённым – иначе.

Процесс $Start$ включает первый циклер C_1 и после этого «умирает».

Каждый циклер C_i отвечает за работу процесса P_i. Циклер C_i

- включает следующий циклер $C_{next(i)}$ после того, как он дал разрешение процессу P_i начать очередной сеанс работы, и
- выключается после того, как он дал разрешение процессу P_i закончить очередной сеанс работы.

Первое свойство в (3) можно выразить следующим соотношением:

$$(Sch \,|\, (?\beta_1)^* \,|\, \ldots \,|\, (?\beta_n)^*) \setminus \{\beta_1, \ldots, \beta_n\} \approx (!\alpha_1 \ldots !\alpha_n)^*. \tag{5}$$

Обозначим левую часть (5) записью Sch'. Процесс Sch' можно рассматривать как процесс, получаемый из Sch заменой меток переходов: все метки переходов в Sch, принадлежащие множеству $\{!\beta_1, \ldots, !\beta_n\}$, заменяются на τ. Для доказательства (5) достаточно доказать, что

$$Sch' \approx !\alpha_1; \ldots !\alpha_n; Sch'. \tag{6}$$

Мы будем преобразовывать левую часть соотношения (6) так, чтобы получилась правая часть этого соотношения. Используя свойства операций на процессах, можно преобразовать Sch' следующим образом:

$$\begin{array}{l} Sch' = (Sch \,|\, (?\beta_1)^* \,|\, \ldots \,|\, (?\beta_n)^*) \setminus \{\beta_1, \ldots, \beta_n\} = \\ = \big(\; ((Start \,|\, C_1 \,|\, \ldots \,|\, C_n) \setminus \Gamma) \,|\, (?\beta_1)^* \,|\, \ldots \,|\, (?\beta_n)^* \; \big) \setminus \{\beta_1, \ldots, \beta_n\} = \\ = (Start \,|\, C_1' \,|\, \ldots \,|\, C_n') \setminus \Gamma, \text{ где } C_i' = (C_i \,|\, (?\beta_i)^*) \setminus \{\beta_i\}. \end{array} \tag{7}$$

Заметим, что для каждого $i = 1, \ldots, n$ имеет место соотношение

$$C'_i \approx ?\gamma_i; !\alpha_i; !\gamma_{next(i)}; C'_i. \tag{8}$$

Действительно, $C_i \equiv ?\gamma_i; !\alpha_i; !\gamma_{next(i)}; !\beta_i; C_i$, и по теореме о разложении

$$\begin{array}{l} C'_i \equiv ((?\gamma_i; !\alpha_i; !\gamma_{next(i)}; !\beta_i; C_i) \,|\, (?\beta_i)^*) \setminus \{\beta_i\} \equiv \\ \equiv ?\gamma_i; !\alpha_i; !\gamma_{next(i)}; \tau; C'_i \approx \text{ правая часть (8).} \end{array}$$

Используя данное замечание и теорему о разложении, цепочку равенств (7) можно продолжить следующим образом:

$$\begin{array}{l} (Start \,|\, C'_1 \,|\, C'_2 \,|\, \ldots \,|\, C'_n) \setminus \Gamma \approx \\ \approx (\underbrace{!\gamma_1; \mathbf{0}}_{=Start} \mid \underbrace{?\gamma_1; !\alpha_1; !\gamma_2; C'_1}_{\approx C'_1} \,|\, C'_2 \,|\, \ldots \,|\, C'_n) \setminus \Gamma \approx \\ \approx \tau; (\mathbf{0} \,|\, !\alpha_1; !\gamma_2; C'_1 \,|\, C'_2 \,|\, \ldots \,|\, C'_n) \setminus \Gamma = \\ = \tau; (!\alpha_1; !\gamma_2; C'_1 \,|\, C'_2 \,|\, \ldots \,|\, C'_n) \setminus \Gamma \approx \\ \approx \tau; !\alpha_1; (!\gamma_2; C'_1 \,|\, C'_2 \,|\, \ldots \,|\, C'_n) \setminus \Gamma \approx \\ \approx \tau; !\alpha_1; (!\gamma_2; C'_1 \,|\, \underbrace{?\gamma_2; !\alpha_2; !\gamma_3; C'_2}_{\approx C'_2} \,|\, \ldots \,|\, C'_n) \setminus \Gamma \approx \\ \approx \tau; !\alpha_1; \tau; (C'_1 \,|\, !\alpha_2; !\gamma_3; C'_2 \,|\, \ldots \,|\, C'_n) \setminus \Gamma \approx \ldots \approx \\ \approx \tau; !\alpha_1; \tau; !\alpha_2; \ldots \tau; !\alpha_n; (C'_1 \,|\, \ldots \,|\, !\gamma_1; C'_n) \setminus \Gamma \approx \\ \approx !\alpha_1; \ldots !\alpha_n; (C'_1 \,|\, \ldots \,|\, !\gamma_1; C'_n) \setminus \Gamma \approx \\ \approx !\alpha_1; \ldots !\alpha_n; (\underbrace{?\gamma_1; !\alpha_1; !\gamma_2; C'_1}_{\approx C'_1} \,|\, \ldots \,|\, !\gamma_1; C'_n) \setminus \Gamma \approx \\ \approx !\alpha_1; \ldots !\alpha_n; \underbrace{\tau; (!\alpha_1; !\gamma_2; C'_1 \,|\, \ldots \,|\, C'_n) \setminus \Gamma}. \end{array} \tag{9}$$

Выражение, подчёркнутое фигурной скобкой в последней строке данной цепочки, совпадает с выражением в четвёртой строке этой цепочки, которое, в свою очередь, находится в отношении $\approx$ с Sch'.

Мы получили, что последнее выражение в цепочке (9)

- находится в отношении $\approx$ с правой частью соотношения (6) и
- с другой стороны, данное выражение находится в отношении $\approx$ с левой частью соотношения (6).

Таким образом, соотношение (6) доказано. ■

Читателю предлагается самостоятельно доказать второе свойство в (3), а также детерминированность Sch, отсутствие в Sch терминальных состояний, и свойство 3 в списке свойств процесса Sch.

Межфакультетский курс «Формальная семантика и верификация программного обеспечения» Лекция 12 Формальная модель криптографических протоколов. Протокол Yaholom

А.М.Миронов

Процессная модель криптографических протоколов отличается от описанной выше процессной модели для обычных процессов. Ниже мы вводим все необходимые понятия для процессной модели криптографических протоколов с самого начала.

1 Термы и связанные с ними понятия

1.1 Типы, переменные, константы и функциональные символы

Будем предполагать, что заданы следующие множества.

- Множество $Types$, его элементы называются **типами**. Будем понимать типы так же, как понимаются типы данных в языках программирования. Каждому типу τ из $Types$ сопоставлено множество D_τ **значений** типа τ.
- Множества Var и Con, их элементы называются **переменными** и **константами** соответственно. Каждой переменной $x \in Var$ и константе $c \in Con$ сопоставлен тип $\tau(x)$ и $\tau(c) \in Types$ соответственно. Каждая переменная x может принимать **значения** в домене $D_{\tau(x)}$, т.е. в различные моменты времени переменная x может быть связана с различными элементами домена $D_{\tau(x)}$.

- Множество Fun, его элементы называются **функциональными символами (ФС)**. Каждому $f \in Fun$ сопоставлены
  - **функциональный тип (ФТ)** $\tau(f)$, который представляет собой запись вида
    $$(\tau_1, \ldots, \tau_n) \to \tau, \tag{1}$$
    где $\tau_1, \ldots, \tau_n, \tau \in Types$, и
  - частичная функция вида $D_{\tau_1} \times \ldots \times D_{\tau_n} \to D_\tau$, где $\tau(f)$ имеет вид (1), данная функция обозначается тем же символом f и называется **интерпретацией** ФС f. (Напомним, что функция $f : D \to D'$ называется **частичной**, если $\forall\, d \in D$ значение $f(d)$ м.б. не определено.)

Будем считать, что $Types$ содержит следующие типы:

- $\mathbf{A}$, значения этого типа называются **агентами**,
- $\mathbf{K}$, значения этого типа называются **ключами**, они обозначают криптографические ключи, которые агенты могут использовать для шифрования или расшифрования сообщений,
- $\mathbf{M}$, значения этого типа называются **сообщениями**, они обозначают сообщения, которые агенты могут пересылать друг другу во время своей работы,
- $\mathbf{B}$ (булев тип), значения этого типа – 0 и 1,
- для каждого типа τ множество $Types$ содержит тип $\mathbf{2}^\tau$, значениями этого типа являются подмножества множества D_τ.

Будем предполагать, что Fun содержит следующие ФС.

- ФС *encrypt* типа $(\mathbf{K}, \mathbf{M}) \to \mathbf{M}$.

  Терм вида $encrypt(k, e)$ обозначает сообщение, получаемое шифрованием сообщения e на ключе k. Будем обозначать такой терм записью $k(e)$ и называть его **шифрованным сообщением (ШС)**.

- ФС *shared_key* типа $(\mathbf{2}^{\mathbf{A}}) \to \mathbf{K}$.

  Терм вида $shared\_key(\{A_1, \ldots, A_n\})$ называется **разделяемым ключом** агентов $A_1, \ldots, A_n$ и будет обозначаться записью $k_{A_1 \ldots A_n}$.

1.2 Термы

Термы строятся из переменных, констант и ФС. Множество всех термов обозначается символом Tm. Каждый терм e имеет тип $\tau(e) \in Types$, определяемый структурой терма e.

Правила построения термов имеют следующий вид:

- каждая переменная и константа является термом того типа, который сопоставлен этой переменной или константе, и
- если $e_1, \ldots, e_n \in Tm$, $f \in Fun$, и $\tau(f)$ имеет вид (1), где $\tau_1 = \tau(e_1)$, $\ldots$, $\tau_n = \tau(e_n)$, то запись $f(e_1, \ldots, e_n)$ – терм типа τ.

Терм $e \in Tm$ называется **подтермом** терма $e' \in Tm$, если либо $e = e'$, либо $e' = f(e_1, \ldots, e_n)$, и $\exists\, i \in \{1, \ldots, n\}$: e – подтерм терма e_i. Запись $e \subseteq e'$, где $e, e' \in Tm$, означает, что e является подтермом терма e'. Запись $e \subset e'$, где $e, e' \in Tm$, означает, что $e \subseteq e'$ и $e \neq e'$. Совокупность всех подтермов терма e обозначается $Sub(e)$.

Индукцией по структуре терма $e \in Tm$ нетрудно доказать, что

$$\begin{array}{l}\text{если } e_1 \text{ и } e_2 \text{ – различные подтермы терма } e,\\ \text{то либо } e_1 \subset e_2, \text{ либо } e_2 \subset e_1,\\ \text{либо } e_1 \text{ и } e_2 \text{ не имеют общих компонентов.}\end{array} \qquad (2)$$

Будем использовать следующие обозначения:

- $\forall\, x \in Var, e \in Tm$ запись $x \in e$ означает, что x входит в e,
- $\forall\, e_1, \ldots, e_n \in Tm$

$$Var(e_1, \ldots, e_n) = \{x \in Var \mid \exists\, i \in \{1, \ldots, n\} : x \in e_i\},$$

- $\forall\, E \subseteq Tm$ множество $Tm(E)$ определяется следующим образом:
  - $E \subseteq Tm(E)$, $Con \subseteq Tm(E)$,
  - для каждого терма вида $f(e_1, \ldots, e_n)$, если $e_1, \ldots, e_n \in Tm(E)$, то $f(e_1, \ldots, e_n) \in Tm(E)$.
- $\forall\, \tau \in Types$, $\forall\, E \subseteq Tm$ $E_\tau = \{e \in E \mid \tau(e) = \tau\}$,
- для каждого списка типов $\tau_1, \ldots, \tau_n$ множество $Types$ содержит тип $(\tau_1, \ldots, \tau_n)$, и $D_{(\tau_1, \ldots, \tau_n)} = D_{\tau_1} \times \ldots \times D_{\tau_n}$, Fun содержит ФС $tuple$ типа $(\tau_1, \ldots, \tau_n) \to (\tau_1, \ldots, \tau_n)$, ему соответствует тождественная функция, терм $tuple(e_1, \ldots, e_n)$ обозначается записью $(e_1, \ldots, e_n)$.

Термы типа **B** называются **формулами**. Множество всех формул обозначается Fm. $\forall X \subseteq Var\ Fm(X) = Tm(X) \cap Fm$. При построении формул могут использоваться обычные булевы ФС ($\neg$, $\wedge$, $\vee$, $\to$ и т.д.), которым соответствуют функции отрицания, конъюнкции, дизъюнкции и т.д. Символы 1 и 0 обозначают тождественно истинную и ложную формулу соответственно. Формулы вида $\wedge(e_1, e_2)$ будем записывать в более привычном виде $e_1 \wedge e_2$. Формулы вида $e_1 \wedge \ldots \wedge e_n$ могут также записываться в виде $\left\{ \begin{array}{c} e_1 \\ \vdots \\ e_n \end{array} \right\}$ или $\{e_1, \ldots, e_n\}$. Возможна конъюнкция произвольного семейства формул $\{e_i \mid i \in I\}$, она обозначается $\bigwedge_{i \in I} e_i$.

Терм $e \in Tm$ **замкнут**, если $Var(e) = \emptyset$. Каждому замкнутому терму e соответствует объект $eval(e)$, называемый **значением** данного терма, и либо является элементом $D_{\tau(e)}$, либо не определён. Если e – константа, то $eval(e)$ – интерпретация этой константы, и если e имеет вид $f(e_1, \ldots, e_n)$, то $eval(e)$ определён, только если определено значение функции f на кортеже $(eval(e_1), \ldots, eval(e_n))$, и в этом случае $eval(e)$ равен этому значению. Для каждого замкнутого терма e будем обозначать объект $eval(e)$ той же записью, что и сам терм (т.е. e).

1.3 Подстановки

Подстановкой называется функция $\theta : Var \to Tm$, такая, что для каждой $x \in Var\ \tau(x) = \tau(\theta(x))$. Будем говорить, что подстановка θ заменяет переменную $x \in Var$ на терм $\theta(x)$.

Будем использовать следующие обозначения:

- множество всех подстановок обозначается символом Θ;
- $\forall\, \theta \in \Theta\ Var(\theta) = \{x \in Var \mid \theta(x) \neq x\}$;
- $\forall\, X \subseteq Var \quad \Theta(X) = \{\theta \in \Theta \mid Var(\theta) \subseteq X\}$;
- подстановка $\theta \in \Theta$ может обозначаться записями

$$x \mapsto \theta(x) \quad \text{или} \quad (\theta(x_1)/x_1, \ldots, \theta(x_n)/x_n), \tag{3}$$

  (вторая запись в (3) используется, если $Var(\theta) = \{x_1, \ldots, x_n\}$);
- $\forall\, \theta \in \Theta,\ \forall\, e \in Tm$ запись e^θ обозначает терм, получаемый из e заменой $\forall\, x \in Var(e)$ каждого вхождения x в e на терм $\theta(x)$;
- $\forall\, \theta \in \Theta,\ \forall\, E \subseteq Tm\ E^\theta = \{e^\theta \mid e \in E\}$,
- $\forall\, \theta, \theta' \in \Theta$ запись $\theta\theta'$ обозначает подстановку $x \mapsto (x^\theta)^{\theta'}$.

Подстановка θ' называется **продолжением** подстановки θ, если верно включение $Var(\theta) \subseteq Var(\theta')$ и $\forall\, x \in Var(\theta)\ \ \theta(x) = \theta'(x)$.

Подстановка θ **замкнута**, если $\forall\, x \in Var(\theta)\ Var(x^\theta) = \emptyset$. Множество всех замкнутых подстановок из $\Theta(X)$ обозначается $X^\bullet$.

Термы e_1 и e_2 считаются равными, если $\forall\, \theta \in Var(e_1, e_2)^\bullet\ e_1^\theta = e_2^\theta$.

2 Последовательные и распределенные процессы

2.1 Действия

Действие – это запись одного из следующих видов:

$$\circ!e, \quad \circ?e, \quad e := e', \quad \text{где } e, e' \in Tm,$$

которые называются **выводом** сообщения e в открытый канал $\circ$, **вводом** сообщения e из открытого канала $\circ$, и **присваиванием**, соответственно. Множество всех действий обозначается Act.

2.2 Последовательные процессы

Последовательным процессом (или просто **процессом**) будем называть граф P со следующими свойствами:

- P имеет выделенные вершины $\odot$ и $\otimes$, называемые **начальной** и **терминальной** вершинами соответственно, из $\otimes$ не выходят рёбра,
- каждому ребру графа P сопоставлена метка $a \in Act$, ребро процесса P представляется записью $v \xrightarrow{a} v'$, где v и v' – начало и конец ребра, a – метка ребра.

Процесс является описанием поведения дискретной динамической системы, работа которой заключается в последовательном выполнении действий, связанных с вводом и выводом сообщений и изменением значений переменных. С каждым процессом P связаны

- агент $Agent(P) \in Var_{\mathbf{A}}$ (от имени которого выполняется процесс),
- множество $Var(P)$ переменных процесса P, являющееся дизъюнктным объединением следующих множеств:
  - множество $Input(P)$ **входных переменных**,

– множество $Unique(P)$ **уникальных переменных** (инициализированных уникальными значениями), и обозначают криптографические ключи, или переменные, называемые **нонсами**,

– множество $Private(P)$ **внутренних переменных**,

– $\{at_P\}$, значения at_P – вершины графа P,

– $\{x_P\}$, значения x_P – подмножества $Var(P)$, их элементы называются **инициализированными переменными**,

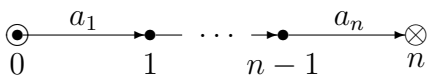
– $\{x_\circ\}$, $\tau(x_\circ) = \mathbf{2^M}$, значения $x_\circ$ интерпретируются как **содержимое открытого канала**.

- **предусловие** $Pre(P) \in Fm$, это конъюнкция формул, содержащая следующие конъюнктивные члены:

$$x_P = Input(P) \cup Unique(P) \cup \{at_P, x_P, x_\circ\},\ at_P = \odot,\ x_\circ = \emptyset$$

(в $Pre(P)$ м.б. и другие конъюнктивные члены).

Процесс называется **линейным**, если он имеет вид



(4)

2.3 Процесс противника

Процесс противника – это процесс $P_\dagger$, обладающий свойствами:

- граф процесса $P_\dagger$ состоит из единственной вершины,
- $\forall\, a \in Act$ граф $P_\dagger$ содержит ребро с меткой a.

Ниже будем предполагать, что $P_\dagger$ – единственный из всех рассматриваемых процессов, граф которого имеет циклы.

2.4 Распределенные процессы

Распределенный процесс (РП) – это семейство $\mathcal{P} = \{P_i \mid i \in I\}$ процессов, таких, что компоненты семейства

$$\{Unique(P_i) \cup Private(P_i) \cup \{at_{P_i}, x_{P_i}\} \mid i \in I\} \tag{5}$$

дизъюнктны (если это условие не выполняется, то соответствующие переменные в процессах P_i переименовываются).

Если $\{\mathcal{P}_i \mid i \in I\}$ – семейство РП, и $\forall\, i \in I \quad \mathcal{P}_i = \{P_{i'} \mid i' \in I_i\}$, то запись $\{\mathcal{P}_i \mid i \in I\}$ обозначает также РП $\{P_{i'} \mid i' \in \bigsqcup_{i\in I} I_i\}$.

Множество $Var(\mathcal{P})$ переменных РП $\mathcal{P}$ определяется как объединение $\bigcup_{i\in I} Var(P_i)$. **Предусловие** $Pre(\mathcal{P})$ РП $\mathcal{P}$ определяется как $\bigwedge_{i\in I} Pre(P_i)$.

Состояние РП $\mathcal{P}$ – это подстановка $\theta \in Var(\mathcal{P})^\bullet$. Состояние θ является частичной функцией, определённой только для тех $x \in Var(\mathcal{P})$, которые входят в $\bigcup_{i\in I} x^\theta_{P_i}$. Состояние θ называется **начальным** (и обозначается θ_0), если $Pre(\mathcal{P})^\theta = 1$.

РП $\mathcal{P} = \{P_i \mid i \in I\}$ обозначается записью P^*, если I – множество натуральных чисел, и все процессы, входящие в $\mathcal{P}$, совпадают с P.

Будем использовать следующее соглашение:

- если в каком-либо рассуждении, связанном с РП вида P^*, некоторый процесс является первым из рассматриваемых процессов, входящих в P^*, то этот процесс и все его переменные обозначаются теми же записями, которые используются в процессе P,
- если кроме этого процесса рассматривается другой процесс, входящий в P^*, то он обозначается $\grave{P}$, и в обозначениях тех его переменных, которые являются дубликатами переменных из $Unique(P) \cup Private(P) \cup \{at_P, x_P\}$, используются обратные штрихи (например, дубликат переменной x в $\grave{P}$ будет обозначаться записью $\grave{x}$), в следующем процессе ($\grave{\grave{P}}$) соответствующие переменные будут обозначаться с двойными обратными штрихами, и т.д.

Для каждого РП $\mathcal{P}$ запись $\mathcal{P}_\dagger$ обозначает РП $\{\mathcal{P}, \{P_\dagger\}\}$.

2.5 Переходы в распределенных процессах

Переход в РП $\mathcal{P} = \{P_i \mid i \in I\}$ – это утверждение, обозначаемое записью $\theta \overset{a_{P_i}}{\to} \theta'$, где $\theta, \theta' \in Var(\mathcal{P})^\bullet$ (θ называется **началом** данного перехода, а θ' – его **концом**) и a – метка некоторого ребра $v \overset{a}{\to} v'$ процесса $P_i \in \mathcal{P}$, причем выполнены следующие условия: $v = at^\theta_{P_i}$ и

- если $a = \circ?e$ или $(e := e')$, и $k(\ldots) \subseteq e$, где $k \in Tm_{\mathbf{K}}$, то

$$\left.\begin{array}{l} \text{либо } k \in x^\theta_P, \\ \text{либо } k = shared\_key(\ldots) \text{ и } Agent(P) \in k \end{array}\right\} \qquad (6)$$

- если $a = \circ!e$, то

$$\left.\begin{array}{l} Var(e) \subseteq x^\theta_P \\ \theta' = (x_\circ \cup \{e^\theta\}/x_\circ, v'/at_{P_i})\theta \end{array}\right\} \qquad (7)$$

- если $a = \circ?e$, то

$$\left.\begin{array}{l} \exists\,\tilde{\theta} \in \Theta(\,Var(e) \setminus x_P^\theta) : e^{\tilde{\theta}\theta} \in x_\circ^\theta \\ \theta' = (x_P^\theta \cup\, Var(e)/x_P, v'/at_{P_i})\tilde{\theta}\theta \end{array}\right\} \tag{8}$$

- если $a = (e := e')$, то

$$\left.\begin{array}{l} Var(e') \subseteq x_P^\theta, \exists\,\tilde{\theta} \in \Theta(\,Var(e) \setminus x_P^\theta) : e^{\tilde{\theta}\theta} = (e')^\theta \\ \theta' = (x_P^\theta \cup\, Var(e)/x_P, v'/at_{P_i})\tilde{\theta}\theta \end{array}\right\}$$

Переход $\theta \stackrel{a_{P_i}}{\to} \theta'$ РП $\mathcal{P}$ интерпретируется как выполнение процессом $P_i \in \mathcal{P}$ действия a, в результате чего $\mathcal{P}$ переходит из состояния θ в состояние θ'. Если в текущий момент $\mathcal{P}$ находится в состоянии θ, и в этот момент некоторый процесс P_i, входящий в $\mathcal{P}$, содержит ребро $at_{P_i}^\theta \stackrel{a}{\to} v'$, причем выполнены условия из вышеприведённого списка, то мы считаем, что РП $\mathcal{P}$, находясь в состоянии θ, может выполнить действие a после чего перейти в состояние θ', при этом

- при выполнении действия $\circ!e$ происходит добавление терма e^θ к содержимому открытого канала $\circ$,

- при выполнении действия $\circ?e$ или $e := e'$ происходит либо чтение некоторого терма из содержимого канала $\circ$, либо присваивание соответственно, путем инициализации неинициализированных в текущий момент переменных из $Var(e)$: терм e рассматривается как шаблон, которому должен соответствовать некоторый терм в $x_\circ^\theta$ или терм $(e')^\theta$ соответственно, и выполняемое действие заключается в определении подходящего продолжения θ' подстановки θ путем определения подходящих значений переменных из $Var(e)$, не вошедших в x_P^θ, с таким расчетом, чтобы значение $e^{\theta'}$ было бы равно некоторому терму из $x_\circ^\theta$ или терму $(e')^\theta$ соответственно.

2.6 Выполнение распределенного процесса

Выполнение РП $\mathcal{P}$ – это последовательность состояний $\pi = (\theta_0, \theta_1, \ldots)$ РП $\mathcal{P}$, такая, что θ_0 – начальное состояние и для каждой пары θ, θ' соседних членов этой последовательности имеется переход $\theta \stackrel{a_P}{\to} \theta'$, где P – какой-либо процесс из $\mathcal{P}$.

Если задано выполнение $\pi = (\theta_0, \theta_1, \ldots)$ и θ, θ' – состояния, входящие в π, то запись $\theta <_\pi \theta'$ означает, что $\theta = \theta_i$ и $\theta' = \theta_j$ для некоторых индексов $i < j$. Запись $\theta \leq_\pi \theta'$ означает, что $\theta <_\pi \theta'$ или $\theta = \theta'$.s Если путь π ясен из контекста, то обозначение π в записях $<_\pi$ и $\leq_\pi$ м.б. опущено.

Состояние θ РП $\mathcal{P}$ называется **достижимым** состоянием, если оно входит в некоторое выполнение РП $\mathcal{P}$. Множество достижимых состояний РП $\mathcal{P}$ обозначается $\Sigma_{\mathcal{P}}$.

Каждое выполнение РП $\mathcal{P}$ можно интерпретировать как путь в графе с множеством вершин $\Sigma_{\mathcal{P}}$, ребра которого соответствуют переходам.

В начальный момент каждого выполнения РП $\mathcal{P} = \{P_i \mid i \in I\}$ переменные из $\bigcup_{i \in I} Unique(P_i)$ инициализированы **уникальными значениями**, т.е. такими значениями, которые никогда не встречались среди всех значений, используемых до начала выполнения $\mathcal{P}$.

3 Свойство защищённости

3.1 Определение свойства защищённости

В рассуждениях, связанных с верификацией РП, будем использовать свойство **защищённости**, обозначаемое записью

$$E \perp P, \text{ где } E \subseteq Tm \text{ и } P \text{ – процесс.} \tag{9}$$

Данное свойство имеет следующий неформальный смысл: переменные из $E \cap Var$ доступны процессу P только в «защищённом» виде, т.е. внутри ШС, которые зашифрованы на ключах, недоступных для P.

Формальное определение данного свойства имеет следующий вид: свойство $E \perp P$ истинно в состоянии $\theta \in \Theta$ (что обозначается $\theta \models E \perp P$), если $\forall\, e \in E^\theta \;\; Agent(P) \notin e$ и

$$\begin{array}{l} \forall\, x \in E^\theta \cap Var,\ \forall\, e \in (x_P^\theta)^\theta \cup x_\circ^\theta \text{ каждое вхождение } x \text{ в } e \\ \text{содержится в подтерме } k(\ldots) \subseteq e, \text{ где } k \in (E^\theta)_{\mathbf{K}}. \end{array} \tag{10}$$

3.2 Теорема о сохранении свойства защищённости при переходах распределённых процессов

В этом параграфе доказывается теорема о сохранении свойства защищённости $E \perp P$ при переходах РП. Данная теорема м.б. интерпретирована как следующее утверждение: если в текущем состоянии θ РП $\mathcal{P}$ верно $E \perp P$, где $P \in \mathcal{P}$, $E \subseteq Tm$, то никакая собственная активность P, начиная с состояния θ, не приведет к тому, что сообщения, представляемые переменными из E, когда-нибудь станут доступны процессу P.

Теорема 1

Пусть заданы РП $\mathcal{P}$, процесс $P \in \mathcal{P}$, и состояния $\theta, \theta' \in \Sigma_{\mathcal{P}}$ таковы, что $\theta \xrightarrow{a_P} \theta'$. Тогда $\forall\, E \subseteq Tm(x_{\mathcal{P}}^{\theta_0})$, где θ_0 – начальное состояние, верна импликация

$$\theta \models E \perp P \Rightarrow \theta' \models E \perp P. \tag{11}$$

Доказательство.

Из $E \subseteq Tm(x_{\mathcal{P}}^{\theta_0})$ следует, что $E = E^{\theta} = E^{\theta'}$.

Пусть верна посылка импликации (11). Докажем, что тогда будет верно её заключение, т.е. $\forall\, x \in E \cap Var$

$$\begin{array}{l}(1)\ \text{каждое вхождение } x \text{ в какой-либо терм } e \in (x_P^{\theta'})^{\theta'} \\ \qquad \text{содержится в подтерме } k(\ldots) \subseteq e,\ \text{где } k \in E_{\mathbf{K}}, \\ (2)\ \text{каждое вхождение } x \text{ в какой-либо терм } e \in x_{\circ}^{\theta'} \\ \qquad \text{содержится в подтерме } k(\ldots) \subseteq e,\ \text{где } k \in E_{\mathbf{K}}.\end{array} \tag{12}$$

1. Если первое утверждение в (12) является неверным, то из (10) следует, что $x_P^{\theta} \subset x_P^{\theta'}$, и имеет место один из двух случаев:

   (a) Первый случай: верно утверждение

   $$\left.\begin{array}{l}\alpha \text{ имеет вид } \circ?e, e^{\theta'} \in x_{\circ}^{\theta}, x_P^{\theta'} = x_P^{\theta} \cup Var(e), \\ \exists\, y \in Var(e) :\ \exists \text{ вхождение } x \text{ в } y^{\theta'}, \text{ которое} \\ \text{не содержится ни в каком подтерме вида} \\ k(\ldots) \subseteq y^{\theta'}, \text{ где } k \in E_{\mathbf{K}}.\end{array}\right\} \tag{13}$$

   Поскольку упомянутое в (13) вхождение x содержится в терме $y^{\theta'} \subseteq e^{\theta'} \in x_{\circ}^{\theta}$, то из (10) следует, что это вхождение x содержится в подтерме $k(\tilde{e}) \subseteq e^{\theta'}$, где $k \in E_{\mathbf{K}}$.

   Из (13) следует, что $k(\tilde{e})$ не м.б. подтермом терма $y^{\theta'}$. Поскольку термы $k(\tilde{e})$ и $y^{\theta'}$ имеют непустое пересечение (оба содержат вышеупомянутое вхождение x), то из (2) следует, что $y^{\theta'} \subset k(\tilde{e})$. Таким образом,

   $$y^{\theta'} \subset k(\tilde{e}) \subseteq e^{\theta'}. \tag{14}$$

   Докажем индукцией по структуре терма e, что из (14) следует

   $$\exists\, z \in Var(e) :\ \ k(\tilde{e}) \subseteq z^{\theta'} \subseteq e^{\theta'}. \tag{15}$$

   Если $e \in Var$, то $z = e$. Если $e \in Con$, то (14) не м.б. верно.

   Если e имеет вид $f(e_1, \ldots, e_n)$, где $f \in Fun$, то

   - если $f = encrypt$, т.е. e имеет вид $k_1(e_1)$, то $k_1 \in Sub(e)_{\mathbf{K}}$, согласно (6), $k_1 \in x_P^{\theta}$, и возможны следующие случаи:

- $k(\tilde{e}) = e^{\theta'} = k_1^{\theta'}(e_1^{\theta'})$, в этом случае $k = k_1^{\theta'} = k_1^{\theta} \in x_P^{\theta}$, однако поскольку $k \in E_{\mathbf{K}}$, то по (10) вхождение k в k должно содержаться в подтерме вида $k'(\ldots) \subseteq k$, что невозможно,
- $k(\tilde{e}) \subseteq k_1^{\theta'}$, данный случай невозможен по определению термов типа $\mathbf{K}$,
- $k(\tilde{e}) \subseteq e_1^{\theta'}$, в данном случае утверждение (15) следует из индуктивного предположения,

• если $f \neq encrypt$, то $\exists\, i \in \{1, \ldots, n\} : k(\tilde{e}) \subseteq e_i^{\theta'}$, и утверждение (15) следует из индуктивного предположения.

Из (14) и (15) следует, что

$$y^{\theta'} \subset k(\tilde{e}) \subseteq z^{\theta'} \subseteq e^{\theta'}. \tag{16}$$

Таким образом, терм e содержит вхождения переменных y и z, обладающие следующим свойством: $y^{\theta'} \subset z^{\theta'}$, откуда для данных вхождений следует включение $y \subset z$, что невозможно.

(b) Второй случай: верно утверждение

$$\left.\begin{array}{l} \alpha \text{ имеет вид } e := e', \text{ где } e' \in Tm(x_P^{\theta}), \\ e^{\theta'} = (e')^{\theta},\ \ x_P^{\theta'} = x_P^{\theta} \cup Var(e), \exists\, y \in Var(e) : \\ \exists \text{ вхождение } x \text{ в } y^{\theta'}, \text{ которое не содержится} \\ \text{ни в каком подтерме вида } k(\ldots) \subseteq y^{\theta'},\ \text{ где } k \in E_{\mathbf{K}}. \end{array}\right\} \tag{17}$$

Поскольку упомянутое в (17) вхождение x содержится в терме $y^{\theta'} \subseteq e^{\theta'} = (e')^{\theta}$, то это вхождение x содержится в подтерме $(z')^{\theta} \subseteq (e')^{\theta}$, где $z' \in Var(e')$.

Так как $e' \in Tm(x_P^{\theta})$, то $Var(e') \subseteq x_P^{\theta}$, следовательно, $z' \in x_P^{\theta}$. Из (10) следует, что упомянутое в (17) вхождение x в $(z')^{\theta}$ содержится в некотором подтерме $k(\tilde{e}) \subseteq (z')^{\theta}$, где $k \in E_{\mathbf{K}}$.

Из (17) следует, что $k(\tilde{e})$ не м.б. подтермом терма $y^{\theta'}$.

Поскольку термы $k(\tilde{e})$ и $y^{\theta'}$ имеют непустое пересечение (оба содержат упомянутое в (17) вхождение x), то из (2) следует, что $y^{\theta'} \subset k(\tilde{e})$.

Из равенства $e' = e^{\theta}$ следует, что $\exists\, z \in Var(e)$: вышеупомянутое вхождение z' в e' входит в подтерм $z^{\theta} \subseteq e^{\theta} = e'$. Следовательно,

$$(z')^{\theta} \subseteq (z^{\theta})^{\theta} = z^{\theta'} \subseteq e^{\theta'}.$$

Таким образом, получаем:

$$y^{\theta'} \subset k(\tilde{e}) \subseteq (z')^{\theta} \subseteq z^{\theta'} \subseteq e^{\theta'}. \tag{18}$$

Как и в предыдущем пункте, на основании (18) заключаем, что терм e содержит вхождения переменных y и z, обладающие следующим свойством: $y^{\theta'} \subset z^{\theta'}$, откуда для данных вхождений следует включение $y \subset z$, что невозможно.

2. Если второе утверждение в (12) неверно, то из (10) следует, что

$$\left.\begin{array}{l} \alpha \text{ имеет вид } \circ!e, \text{ где } e \in Tm(x_P^\theta), \\ \exists \text{ вхождение } x \text{ в } e^\theta, \text{ которое не содержится} \\ \text{ни в каком подтерме вида } k(\ldots) \subseteq e^\theta, \text{ где } k \in E_{\mathbf{K}}. \end{array}\right\} \qquad (19)$$

Так как $e \in Tm(x_P^\theta)$, то упомянутое в (19) вхождение x в e^θ содержится в подтерме вида y^θ терма e^θ, где y – некоторая переменная из x_P^θ. Согласно (10), это вхождение x в y^θ содержится в подтерме вида $k(\ldots) \subseteq y^\theta \subseteq e^\theta$, где $k \in E_{\mathbf{K}}$, что противоречит (19). ■

3.3 Теорема для обоснования свойства соответствия

Теорема, излагаемая в этом параграфе, может использоваться для обоснования **свойства соответствия** протоколов аутентификации, которое имеет следующий смысл: если один из участников протокола аутентификации после выполнения этого протокола пришел к выводу, что другой участник этого протокола является подлинным (т.е. объявленные им свое имя и параметры совпадают с его реальными именем и параметрами), то это действительно верно. Доказываемая ниже теорема применяется для обоснования того, что если в некотором состоянии $\theta \in \Sigma_{\mathcal{P}}$ в канале $\circ$ содержится сообщение, содержащее подтерм $k(e)$, где ключ k недоступен в этом состоянии для некоторого процесса P, входящего в $\mathcal{P}$, то в некотором состоянии $\theta' <_\pi \theta$ другой процесс $P' \neq P$ из $\mathcal{P}$ послал в открытый канал $\circ$ сообщение, содержащее этот подтерм $k(e)$.

Теорема 2.

Пусть заданы РП $\mathcal{P}$, процесс $P \in \mathcal{P}$, начальное состояние θ_0 РП $\mathcal{P}$, подмножество E множества $Tm(x_{\mathcal{P}}^{\theta_0})$ и состояние $\theta \in \Sigma_{\mathcal{P}}$, причем

- $\theta \models E \perp P$, и
- $x_\circ^\theta$ содержит терм с подтермом $k(e)$, где $k \in E_{\mathbf{K}}$.

Тогда для каждого пути π из начального состояния θ_0 РП $\mathcal{P}$ в состояние θ существует процесс $P' \in \mathcal{P} \setminus \{P\}$, такой, что π содержит ребро

$$\dot{\theta} \xrightarrow{(!\dot{e})_{P'}} \theta', \quad \text{где } k(e) \subseteq \dot{e}^{\dot{\theta}}. \qquad (20)$$

Доказательство.

Обозначим записью θ' первое состояние на пути π, такое, что $x_\circ^{\theta'}$ содержит терм e' с подтермом $k(e)$. Т.к. $x_\circ^0 = \emptyset$, то $\theta' \neq \theta_0$.

Пусть ребро на пути π с концом в θ' имеет вид $\dot\theta \stackrel{\alpha_{P'}}{\to} \theta'$. Т.к. $e' \notin x_\circ^{\dot\theta}$, то $\alpha = !\dot e$, где $\dot e^{\dot\theta} = e'$. Если $P' \neq P$, то теорема доказана. Докажем, что случай ($P' = P$) невозможен.

Пусть $P' = P$, т.е. $\dot\theta \stackrel{(!\dot e)_P}{\longrightarrow} \theta'$. Докажем, что $k \in E \cap Var$.

Если это неверно, т.е. k – разделяемый ключ, то, согласно определению выполнения процесса, должно быть верно $Agent(P) \in k \in E_{\mathbf{K}}$, что противоречит предположению $\forall\, \tilde e \in E \;\; Agent(P) \notin \tilde e$.

Из $\theta \models E \perp P$ следует, что $\dot\theta \models E \perp P$, откуда получаем $k \notin x_P^{\dot\theta}$, т.к. если $k \in x_P^{\dot\theta}$, то, согласно (10), вхождение k в $k^{\dot\theta} = k$ должно содержаться в подтерме вида $k'(\ldots)$, что невозможно.

Из соотношения $k \notin x_P^{\dot\theta}$ и из условия $\dot e \in Tm(x_P^{\dot\theta})$, которое верно согласно (7), следует, что $k \notin \dot e$.

Аналогично доказательству импликации (14) $\Rightarrow$ (15) в теореме 1, можно доказать, что из свойств $k(e) \subseteq e' = \dot e^{\dot\theta}$ и $k \notin \dot e$ следует, что

$$\exists\, x \in Var(\dot e) \subseteq x_P^{\dot\theta} : k(e) \subseteq x^{\dot\theta} \in x_P^{\dot\theta}. \tag{21}$$

Обозначим записью θ'' первое состояние на пути π, такое, что $(x_P^{\theta''})^{\theta''}$ содержит терм с подтермом $k(e)$, т.е.

$$\exists\, x \in x_P^{\theta''} : k(e) \subseteq x^{\theta''}. \tag{22}$$

Из (21) следует, что θ'' находится на пути π левее θ'. Нетрудно видеть, что θ'' – не начальное состояние, поэтому на пути π существует ребро вида $\ddot\theta \stackrel{\alpha_{P''}}{\longrightarrow} \theta''$. Из выбора состояния θ'' следует, что $x \notin x_P^{\ddot\theta}$, поэтому $P'' = P$, и возможны два случая:

1. $\alpha = \circ?\ddot e, x \in Var(\ddot e), \ddot e^{\theta''} \in x_\circ^{\ddot\theta}$,

   т.к. $k(e) \subseteq x^{\theta''} \subseteq \ddot e^{\theta''} \in x_\circ^{\ddot\theta}$, то получаем противоречие с выбором θ' как самого первого состояния на пути π, такого, что $x_\circ^{\theta'}$ содержит терм e' с подтермом $k(e)$: состояние $\ddot\theta$ имеет аналогичное свойство, и находится левее θ',

2. $\alpha = (\ddot e := \tilde e), x \in Var(\ddot e), \tilde e \in Tm(x_P^{\ddot\theta}), \ddot e^{\theta''} = \tilde e^{\ddot\theta}$,

   поскольку

   - $k(e) \subseteq x^{\theta''} \subseteq \ddot e^{\theta''} = \tilde e^{\ddot\theta}$ и

- $\tilde{e}$ не содержит k, т.к. выше было установлено, что $k \notin x_P^{\dot{\theta}}$, поэтому, учитывая свойство $\ddot{\theta} \leq \dot{\theta}$, из которого следует включение $x_P^{\ddot{\theta}} \subseteq x_P^{\dot{\theta}}$, получаем: $k \notin x_P^{\ddot{\theta}}$, и, следовательно, терм $\tilde{e} \in Tm(x_P^{\ddot{\theta}})$ тоже не содержит k,

то, аналогично доказательству импликации (14) $\Rightarrow$ (15) в теореме 1, можно доказать, что $\exists\, y \in x_P^{\ddot{\theta}} : k(e) \subseteq y^{\ddot{\theta}}$, что противоречит выбору θ'' как самого первого состояния на пути π со свойством (22): $\ddot{\theta}$ имеет аналогичное свойство, и находится левее θ''. ■

4 Верификация протокола Yahalom

В этом параграфе изложенная выше математическая модель РП и связанные с ней теоремы 1 и 2 применяются для верификации свойств соответствия и секретности протокола аутентификации Yahalom.

4.1 Описание протокола Yahalom

Протокол Yahalom предназначен для аутентификации (т.е. проверки подлинности) агентов, взаимодействующих по открытому каналу $\circ$, и передачи сеансовых ключей между этими агентами. Предполагается что

- заданы множество агентов Ag, а также агент J, называемый **доверенным посредником**, данные агенты могут взаимодействовать друг с другом по открытому каналу $\circ$,
- каждый агент $A \in Ag$ имеет разделяемый ключ k_{AJ} с доверенным посредником J, на котором A и J могут шифровать и дешифровать сообщения, используя симметричную систему шифрования.

В каждом сеансе протокола Yahalom принимают участие следующие агенты: **инициатор** $A \in Ag$, доверенный посредник J, и **респондер** $B \in Ag$. Каждый агент из Ag в одних сеансах м.б. инициатором, а в других – респондером. Выполнение сеанса протокола Yahalom с инициатором A, респондером B и доверенным посредником J представляет собой совокупность четырех пересылок сообщений:

$$\begin{array}{lll} 1. & A \to B & : \quad A, n_A \\ 2. & B \to J & : \quad B, k_{BJ}(A, n_A, n_B) \\ 3. & J \to A & : \quad k_{AJ}(B, k, n_A, n_B), k_{BJ}(A, k) \\ 4. & A \to B & : \quad k_{BJ}(A, k), k(n_B) \end{array} \tag{23}$$

Пересылки в (23) имеют следующий смысл.

1. A посылает B запрос на аутентификацию и генерацию сеансового ключа k, этот запрос состоит из имени агента A и нонса n_A.

2. B посылает J запрос на генерацию сеансового ключа k, в свой запрос он включает своё имя, имя агента A, для связи с которым нужен этот ключ, полученный нонс n_A, и свой нонс n_B.

3. J генерирует сеансовый ключ k и посылает A пару сообщений, из первого сообщения A может извлечь сеансовый ключ k, а второе предназначено для того, чтобы A переслал его B.

4. A посылает B пару сообщений,

   - первое из которых было получено им от J, B может извлечь из этого сообщения сеансовый ключ k, и
   - используя ключ k, B дешифрует второе сообщение.

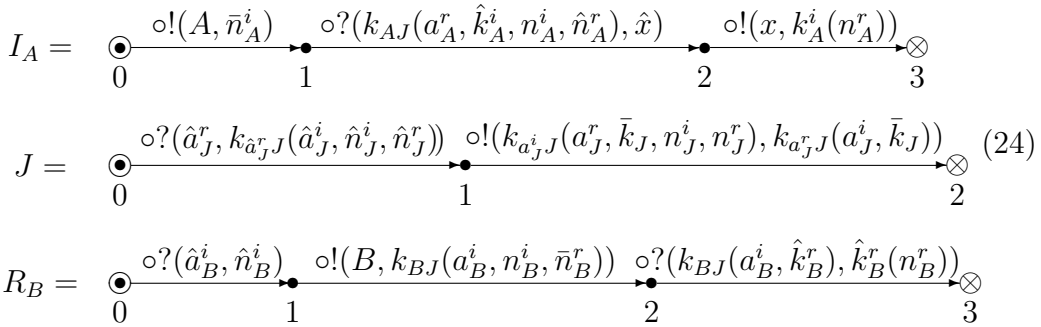
   Если результат дешифрования совпадает с n_B, то это является для B доказательством того, что отправителем этого сообщения был A.

4.2 Формальное описание процессов, входящих в протокол Yahalom

В целях большей наглядности будем использовать следующее соглашение в обозначениях переменных: пусть P – процесс вида (4), и переменная x входит в действие a_i, причём $\forall\, j \in \{1, \ldots, i-1\}$ x не входит в a_j, тогда

- если $x \in Unique(P)$, то будем указывать горизонтальную черту над всеми вхождениями x в a_i (т.е. обозначать их $\bar{x}$) и
- если $x \in Private(P)$, то будем указывать уголок над всеми вхождениями x в a_i (т.е. обозначать их $\hat{x}$).

Описание сеанса протокола Yahalom изображается схемой



(24)

В этой схеме первая и третья диаграммы соответствуют процессам I_A и R_B, описывающим поведение инициатора A и респондера B соответственно, вторая диаграмма соответствует процессу, описывающему поведение посредника J, этот процесс обозначается символом J. Смысл переменных в этих процессах усматривается из сопоставления действий в этих процессах с соответствующими действиями в (23). Верхний индекс i или r при какой-либо переменной означает, что она содержит информацию об инициаторе (i) или респондере (r) данного сеанса. Предполагаем, что $Agent(I_A) = A$, $Agent(R_B) = B$, $Agent(J) = J$.

РП $\mathcal{P}$, соответствующий протоколу Yahalom, имеет вид

$$\mathcal{P} = \{\{I_A^* \mid A \in Ag\}, \{R_B^* \mid B \in Ag\}, J^*\}, \tag{25}$$

т.е. каждый агент может участвовать в неограниченном числе сеансов как в качестве инициатора, так и в качестве респондера.

Ниже будем использовать следующие обозначения:

- если $\mathcal{P}$ – РП, и π – выполнение $\mathcal{P}$, то запись

  $$\pi \ni P^{i,i'} : \theta \xrightarrow{a} \theta'$$

  означает, что π содержит ребро $\theta \xrightarrow{a_P} \theta'$, и $at_P^{\theta} = i$, $at_P^{\theta'} = i'$,

- $\theta \models E \perp e$ означает, что $\forall\, x \in E \cap Var$ каждое вхождение x в e^θ содержится в подтерме $k(\ldots) \subseteq e^\theta$, где $k \in E^\theta_{\mathbf{K}}$.

Нетрудно доказать, что

$$\theta \models E \perp (e, e') \Leftrightarrow \theta \models E \perp e \quad \text{и} \quad \theta \models E \perp e'. \tag{26}$$

4.3 Свойства протокола Yahalom

В этом пункте приводится формальное описание и верификация трех свойств протокола (25): секретность ключей k_J и нонсов n_B^r, аутентификация инициатора перед респондером и аутентификация респондера перед инициатором. В доказательствах теорем 3, 4, 5 при каждом применении теоремы 2 имеется единственный вариант обоснования существования ребра (20) в графе $\Sigma_{\mathcal{P}_\dagger}$, и мы будем сразу будем излагать это обоснование, без упоминания о единственности варианта такого обоснования.

Теорема 3 (секретность ключей k_J и нонсов n_B^r)

РП (25) обладает следующим свойством:

$$\forall\,\theta \in \Sigma_{\mathcal{P}_\dagger} \quad \theta \models E \perp P_\dagger, \text{ где } E = \{k_{BJ}, k_J, n^r_B \mid B \in Ag\}. \tag{27}$$

Доказательство.

Докажем (27) от противного.

Предположим, что $S = \{\theta \in \Sigma_{\mathcal{P}_\dagger} \mid \theta \not\models E \perp P_\dagger\} \neq \emptyset$.

$\forall\,\theta \in S$ обозначим π_θ путь минимальной длины из θ_0 в θ. Пусть θ – состояние из S с наименьшей длиной π_θ. Т.к. $\theta_0 \models E \perp P_\dagger$, то $\theta \neq \theta_0$.

Пусть $\theta' \stackrel{a_P}{\to} \theta$ – ребро из π_θ с концом в θ.

Из определения θ следует, что $\theta' \models E \perp P_\dagger$, $\theta \not\models E \perp P_\dagger$. Если бы было верно $P = P_\dagger$, то из теоремы 1 следует, что $\theta \models E \perp P_\dagger$, т.е. имеем противоречие. Поэтому $P \in \{I_A, R_B, J \mid A, B \in Ag\}$, и

$$a_P = \circ!e,\ x^\theta_\circ = x^{\theta'}_\circ \cup \{e\},\ \exists\, y \in E \cap Var : \neg(y \perp_E e^\theta), \tag{28}$$

где $y \perp_E e^\theta$, означает, что

$$\begin{array}{l}\text{каждое вхождение } y \text{ в } e^\theta \text{ содержится}\\ \text{в подтерме } k(\ldots) \subseteq e^\theta, \text{ где } k \in (E^\theta)_{\mathbf{K}}\end{array} \tag{29}$$

Перебором всех вариантов обоснования существования ребра $\theta' \stackrel{a_P}{\to} \theta$ со свойствами (28) находим единственное возможное обоснование:

$$\pi_\theta \ni I^{2,3}_A : \theta' \stackrel{\circ!e}{\to} \theta, \text{ где } e = (x, k^i_A(n^r_A)). \tag{30}$$

Т.к. $\theta' \models at_{I_A} = 2$, то $\exists\,\theta_1 \leq_{\pi_\theta} \theta'$:

$$\pi_\theta \ni I^{1,2}_A : \theta'_1 \stackrel{\circ?e_1}{\to} \theta_1, \text{ где } e_1 = (k_{AJ}(a^r_A, \hat{k}^i_A, n^i_A, \hat{n}^r_A), \hat{x}). \tag{31}$$

Т.к. $\theta_1 \leq_{\pi_\theta} \theta'$ и $\theta' \models E \perp P_\dagger$, то $\theta_1 \models E \perp P_\dagger$. В частности, $\theta_1 \models E \perp e_1$. По (26), отсюда следует, что $\theta_1 \models E \perp x$.

По теореме 2, из $\theta_1 \models E \perp P_\dagger$, $e^\theta_1 \in x^{\theta_1}_\circ$ и $k_{AJ} \in E$ следует, что $\exists\,\theta_2 \leq_{\pi_\theta} \theta'_1$: π_θ содержит ребро $\theta'_2 \stackrel{(\circ!e_2)_P}{\longrightarrow} \theta_2$, где $P \in \mathcal{P}$ и первая компонента $k_{AJ}(\ldots)$ терма e^θ_1 входит в e^θ_2. Перебором всех вариантов обоснования существования ребра с такими свойствами находим единственное обоснование:

$$\left\{\begin{array}{l}\pi_\theta \ni J^{1,2} : \theta'_2 \stackrel{\circ!e_2}{\to} \theta_2, \text{ где } e_2 = (k_{a^i_J J}(a^r_J, \bar{k}_J, n^i_J, n^r_J), \ldots)\\ k_{(a^i_J)^\theta J}((a^r_J)^\theta, \bar{k}_J, \ldots) = k_{AJ}(a^r_A, (k^i_A)^\theta, \ldots)\end{array}\right. \tag{32}$$

(многоточие в (32) и ниже обозначает компоненту пары, не представляющую интерес для рассмотрения). Из (32) следует, что $(k^i_A)^\theta = \bar{k}_J$, поэтому

$\theta \models E \perp k_A^i(n_A^r)$. Учитывая установленное выше свойство $\theta_1 \models E \perp x$, на основании (26) получаем: $\theta \models E \perp (x, k_A^i(n_A^r))$, т.е. $\theta \models E \perp P$, что противоречит предположению $\theta \not\models E \perp P$. ■

Доказанное свойство $\forall\, \theta \in \Sigma_{\mathcal{P}_\dagger}\ \ \theta \models E \perp P_\dagger$ используется ниже.

Теорема 4 (аутентификация инициатора перед респондером)
Процесс (25) обладает следующим свойством: $\forall\, R_B \in \mathcal{P}$, $\theta \in \Sigma_{\mathcal{P}_\dagger}$, если $\theta \models at_{R_B} = 3$, то $\exists\ I_A \in \mathcal{P}$:

$$\theta \models \left\{ \begin{array}{l} at_{I_A} = 3 \\ a_A^r = B,\ a_B^i = A \\ n_A^i = n_B^i,\ n_A^r = n_B^r \\ k_A^i = k_B^r \end{array} \right\} \tag{33}$$

Доказательство.
Пусть процесс $R_B \in \mathcal{P}$ и состояние $\theta \in \Sigma_{\mathcal{P}_\dagger}$ таковы, что $\theta \models at_{R_B} = 3$. Докажем, что $\exists\, I_A \in \mathcal{P}$: выполнено (33).
Пусть π – путь из θ_0 в θ. Из $\theta \models at_{R_B} = 3$ следует: $\exists\, \theta_1 \leq_\pi \theta$:

$$\pi \ni R_B^{2,3} : \theta_1' \overset{\circ ? e_1}{\to} \theta_1,\ \text{где}\ e_1 = (k_{BJ}(a_B^i, \hat{k}_B^r), \hat{k}_B^r(n_B^r)).$$

По теореме 2, из $\theta_1 \models E \perp P_\dagger$, $e_1^\theta \in x_\circ^{\theta_1}$ и $k_{BJ} \in E$ следует: $\exists\, \theta_2 \leq_\pi \theta_1'$:

$$\left\{ \begin{array}{l} \pi \ni J^{1,2} : \theta_2' \overset{\circ ! e_2}{\to} \theta_2,\ \ \text{где}\ e_2 = (\ldots, k_{a_J^r J}(a_J^i, \bar{k}_J)) \\ k_{(a_J^r)^\theta J}((a_J^i)^\theta, \bar{k}_J) = k_{BJ}((a_B^i)^\theta, (k_B^r)^\theta) \end{array} \right. \tag{34}$$

Из второго равенства в (34) следует, что

$$(a_J^r)^\theta = B, (a_J^i)^\theta = (a_B^i)^\theta, \bar{k}_J = (k_B^r)^\theta. \tag{35}$$

Из $\theta_2' \models at_J = 1$ следует, что $\exists\, \theta_3 \leq_\pi \theta_2'$:

$$\pi \ni J^{0,1} : \theta_3' \overset{\circ ? e_3}{\to} \theta_3,\ \text{где}\ e_3 = (\ldots, k_{\hat{a}_J^r J}(\hat{a}_J^i, \hat{n}_J^i, \hat{n}_J^r)). \tag{36}$$

Из (35) и (36) следует, что $k_{BJ}(*, *, *) \subseteq e_3^\theta \in x_\circ^{\theta_3}$ (где звёздочки обозначают некоторые термы), откуда по теореме 2, с учетом соотношения $\theta_3 \models E \perp P_\dagger$ и $k_{BJ} \in E$ получаем: $\exists\, \theta_4 \leq_\pi \theta_3'$:

$$\left\{ \begin{array}{l} \pi \ni R_{\grave{B}}^{1,2} : \theta_4' \overset{\circ ! e_4}{\to} \theta_4,\ \text{где}\ e_4 = (\ldots, k_{\grave{B}J}(a_{\grave{B}}^i, n_{\grave{B}}^i, \bar{n}_{\grave{B}}^r)) \\ k_{\grave{B}J}((a_{\grave{B}}^i)^\theta, (n_{\grave{B}}^i)^\theta, \bar{n}_{\grave{B}}^r) = k_{BJ}((a_B^i)^\theta, (n_J^i)^\theta, (n_J^r)^\theta) \end{array} \right. \tag{37}$$

Из второго равенства в (37) следует, что

$$\grave{B} = B, (n_B^i)^\theta = (n_J^i)^\theta, \bar{n}_B^r = (n_J^r)^\theta. \tag{38}$$

По теореме 2, из $\theta_1 \models E \perp P_\dagger$, $(k_B^r(n_B^r))^\theta \subseteq e_1^\theta \in x_\circ^{\theta_1}$, и $(k_B^r)^\theta = \bar{k}_J \in E$ следует, что $\exists\, \theta_5 \leq_\pi \theta_1'$:

$$\left\{ \begin{array}{l} \pi \ni I_A^{2,3} : \theta_5' \stackrel{\circ!e_5}{\rightarrow} \theta_5, \text{ где } e_5 = (\ldots, k_A^i(n_A^r)) \\ (k_A^i(n_A^r))^\theta = \bar{k}_J(n_B^r) \end{array} \right. \tag{39}$$

Из второго равенства в (39) следует, что

$$(k_A^i)^\theta = \bar{k}_J, (n_A^r)^\theta = n_B^r. \tag{40}$$

Из $\theta_5 \models at_{I_A} = 2$ следует, что $\exists\, \theta_6 \leq_\pi \theta_5'$:

$$\pi \ni I_A^{1,2} : \theta_6' \stackrel{\circ?e_6}{\rightarrow} \theta_6, \text{ где } e_6 = (k_{AJ}(a_A^r, \hat{k}_A^i, n_A^i, \hat{n}_A^r), \ldots). \tag{41}$$

Из (40) и (41) следует, что

$$k_{AJ}(a_A^r, (k_A^i)^\theta, n_A^i, (n_A^r)^\theta) = k_{AJ}(a_A^r, \bar{k}_J, n_A^i, n_B^r) \subseteq e_6^\theta \in x_\circ^{\theta_6}. \tag{42}$$

По теореме 2, из $\theta_6 \models E \perp P_\dagger$, $k_{AJ} \in E$, и (42) следует, что $\exists\, \theta_7 \leq_\pi \theta_6'$:

$$\left\{ \begin{array}{l} \pi \ni \grave{J}^{1,2} : \theta_7' \stackrel{\circ!e_7}{\rightarrow} \theta_7, \text{ где } e_7 = (k_{a_{\grave{J}}^i \grave{J}}(a_{\grave{J}}^r, \bar{k}_{\grave{J}}, n_{\grave{J}}^i, n_{\grave{J}}^r), \ldots) \\ k_{(a_{\grave{J}}^i)^\theta J}((a_{\grave{J}}^r)^\theta, \bar{k}_{\grave{J}}, (n_{\grave{J}}^i)^\theta, (n_{\grave{J}}^r)^\theta) = k_{AJ}(a_A^r, \bar{k}_J, n_A^i, n_B^r) \end{array} \right. \tag{43}$$

Из второго равенства в (43) следует, что

$$(a_{\grave{J}}^i)^\theta = A, (a_{\grave{J}}^r)^\theta = a_A^r, \grave{J} = J, (n_J^i)^\theta = n_A^i, (n_J^r)^\theta = \bar{n}_B^r. \tag{44}$$

Свойство (33) следует из (35), (38), (40), (44). ■

Теорема 5 (аутентификация респондера перед инициатором)
Процесс (25) обладает следующим свойством: $\forall\, I_A \in \mathcal{P}$, $\theta \in \Sigma_{\mathcal{P}_\dagger}$, если $\theta \models at_{I_A} = 2$, то $\exists\, R_B \in \mathcal{P}$:

$$\theta \models \left\{ \begin{array}{l} at_{R_B} = 2, \\ a_A^r = B, a_B^i = A, \\ n_A^i = n_B^i, n_A^r = n_B^r \end{array} \right\}. \tag{45}$$

Доказательство.
Пусть процесс $I_A \in \mathcal{P}$ и состояние $\theta \in \Sigma_{\mathcal{P}_\dagger}$ таковы, что $\theta \models at_{I_A} = 2$.

Докажем, что $\exists\, R_B \in \mathcal{P}$: выполнено (45).

Пусть π – путь из θ_0 в θ. Из $\theta \models at_{I_A} = 2$ следует, что $\exists\, \theta_1 \leq_\pi \theta$:

$$\pi \ni I_A^{1,2} : \theta_1' \overset{\circ?e_1}{\to} \theta_1, \text{ где } e_1 = (k_{AJ}(a_A^r, \hat{k}_A^i, n_A^i, \hat{n}_A^r), \ldots). \tag{46}$$

По теореме 2, из $\theta_1 \models E \perp P_\dagger$, $k_{AJ}(*,*,*,*) \subseteq e_1^\theta \in x_\circ^{\theta_1}$ (где звёздочки обозначают некоторые термы) и $k_{AJ} \in E$, следует, что $\exists\, \theta_2 \leq_\pi \theta_1'$:

$$\begin{cases} \pi \ni J^{1,2} : \theta_2' \overset{\circ!e_2}{\to} \theta_2, \text{ где } e_2 = (k_{a_J^i J}(a_J^r, \bar{k}_J, n_J^i, n_J^r), \ldots) \\ k_{(a_J^i)^\theta J}((a_J^r)^\theta, \bar{k}_J, (n_J^i)^\theta, (n_J^r)^\theta) = k_{AJ}(a_A^r, (k_A^i)^\theta, n_A^i, (n_A^r)^\theta) \end{cases} \tag{47}$$

Из второго равенства в (47) следует, что

$$(a_J^i)^\theta = A, (a_J^r)^\theta = a_A^r, \bar{k}_J = (k_A^i)^\theta, (n_J^i)^\theta = n_A^i, (n_J^r)^\theta = (n_A^r)^\theta. \tag{48}$$

Из $\theta_2 \models at_J = 1$ следует, что $\exists\, \theta_3 \leq_\pi \theta_2'$:

$$\pi \ni J^{0,1} : \theta_3' \overset{\circ?e_3}{\to} \theta_3, \text{ где } e_3 = (\ldots, k_{\hat{a}_J^r J}(\hat{a}_J^i, \hat{n}_J^i, \hat{n}_J^r)). \tag{49}$$

Из (48) и (49) следует, что $k_{a_A^r J}(A, n_A^i, (n_A^r)^\theta) \subseteq e_3^\theta \in x_\circ^{\theta_3}$, откуда по теореме 2, учитывая $\theta_3 \models E \perp P_\dagger$, и $k_{a_A^r J} \in E$, получаем: $\exists\, \theta_4 \leq_\pi \theta_3'$:

$$\begin{cases} \pi \ni R_B^{1,2} : \theta_4' \overset{\circ!e_4}{\to} \theta_4, \text{ где } e_4 = (\ldots, k_{BJ}(a_B^i, n_B^i, \bar{n}_B^r)) \\ k_{BJ}((a_B^i)^\theta, (n_B^i)^\theta, \bar{n}_B^r) = k_{a_A^r J}(A, n_A^i, (n_A^r)^\theta). \end{cases} \tag{50}$$

Второе равенство в (50) влечёт равенства, из которых следует (45): $B = a_A^r, (a_B^i)^\theta = A, (n_B^i)^\theta = n_A^i, \bar{n}_B^r = (n_A^r)^\theta$. ■