А. М. Миронов

МЕТОДЫ ВЕРИФИКАЦИИ ПРОГРАММ

ДМК Пресс
издательство
Москва, 2023

УДК 004.052
ББК 32.973.0

М64

Миронов А. М.
М64 Методы верификации программ. — М.: ДМК Пресс. 2023 — 336 с.

ISBN 978-5-93700-278-5

В книге излагаются вопросы моделирования и верификации (т.е. доказательства правильности) различных классов программ. Основные концепции и основанные на них подходы к верификации иллюстрированы примерами верификации различных программ. Для закрепления усвоения изложенного материала в книге приведено большое количество задач.

Книга предназначена для студентов высших учебных заведений, обучающихся по специальностям «теоретические основы информатики», «искусственный интеллект» и «информационная безопасность».

Книга подготовлена при поддержке Междисциплинарной научно-образовательной школы Московского университета «Мозг, когнитивные системы, искусственный интеллект».

УДК 004.052
ББК 32.973.0

ISBN 978-5-93700-278-5

Оглавление

Глава 1

Введение

1.1 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 [1], «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 $Spec$.

1.2 Необходимые математические понятия

1.2.1 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
  $$(\tau_1, \ldots, \tau_n) \to \tau, \tag{1.1}$$
  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 (1.1), 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 (1.1), 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

$$\begin{array}{l}\text{если } e_1 \text{ и } e_2 \text{ – различные подтермы терма } e, \text{ то либо } e_1 \subset e_2,\\ \text{либо } e_2 \subset e_1, \text{ либо } e_1 \text{ и } e_2 \text{ не имеют общих компонентов.}\end{array} \tag{1.2}$$

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).

1.2.2 Примеры типов и функциональных символов

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.

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 ($\neg$, $\wedge$, $\vee$, $\rightarrow$ 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 $e?e_1 : e_2$.

Строковые типы и функциональные символы

$Types$ содержит типы $\mathbf{L}$ и $\mathbf{S}$, 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.

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 $(e_1, \dots, e_n)$.

1.2.3 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

$$x \mapsto \theta(x) \quad \text{или} \quad (\theta(x_1)/x_1, \ldots, \theta(x_n)/x_n), \tag{1.3}$$

  вторая запись в (1.3) 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

$$\forall\, \vec{d} = (d_1, \ldots, d_n) \in D_{\tau(\vec{x})} \quad e(\vec{x}) : \vec{d} \mapsto e^{(d_1/x_1, \ldots, d_n/x_n)}. \tag{1.4}$$

1.2.4 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 $ord(a_{i..j})$, $a_{i..j} \le a_{i'..j'}$, $a_{i..j} \le a_{i'}$ формулы соответственно:

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

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

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.

Часть I

Верификация последовательных и распределённых программ

Глава 2

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**. Одними из первых работ, в которых изложен этот метод, являются статьи [3] и [4]. Различные изложения данного метода можно найти также в книгах [5]–[11].

2.1 Понятие блок-схемы

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

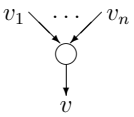
$$\boxed{x := e}\ ,$$

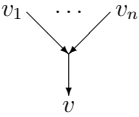
где $x \in Var$, $e \in Tm$, $\tau(x) = \tau(e)$, из каждого присваивания выходит только одно ребро;

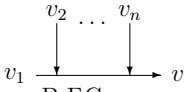
- **условный переход**, вершина такого вида обозначается записью вида β, где $\beta \in Fm$, из каждого условного перехода выходит два ребра с метками 1 и 0 соответственно;

- **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

$$\boxed{(x_1, \ldots, x_n) := (e_1, \ldots, e_n)} \tag{2.1}$$

обозначает последовательность присваиваний вида

$$\boxed{y_1 := e_1} \to \cdots \to \boxed{y_n := e_n} \to \boxed{x_1 := y_1} \to \cdots \to \boxed{x_n := y_n}, \tag{2.2}$$

где переменные $y_1, \ldots, y_n$ – новые, т.е. если какая-либо БС содержит фрагмент вида (2.1), то он рассматривается как последовательность (2.2), где переменные $y_1, \ldots, y_n$ присутствуют только в присваиваниях из (2.2) и не присутствуют в этой БС вне (2.2);

- 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.

2.2 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

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

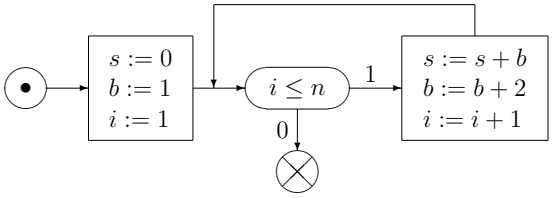
(данное действие заключается в обновлении значения переменной x: после исполнения этого действия значение x становится равным значению терма e на текущих значениях переменных БС P, значения остальных переменных не изменяются);

- если $v_i =$  , 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  , можно рассматривать как перемену местами содержимого переменных x и y.

2.3 Примеры блок-схем

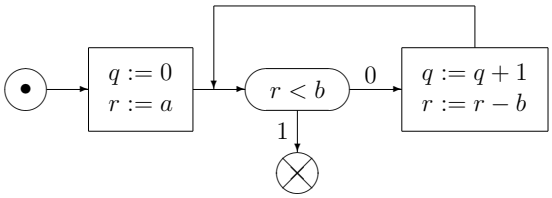
1. БС, вычисляющая сумму $1+3+5\ldots+(2n-1)$, где $n \geq 1$ – входное значение, результат должен быть занесён в переменную s.



(2.3)

   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 $0 \leq r < b$.

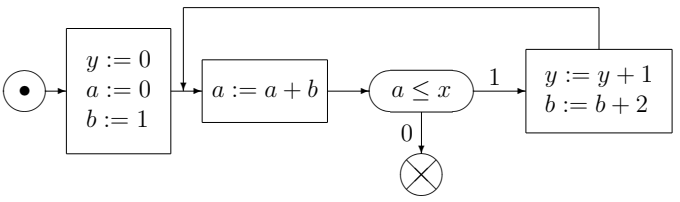


(2.4)

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.

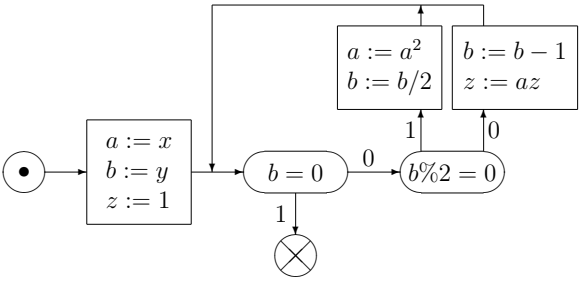


(2.5)

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.



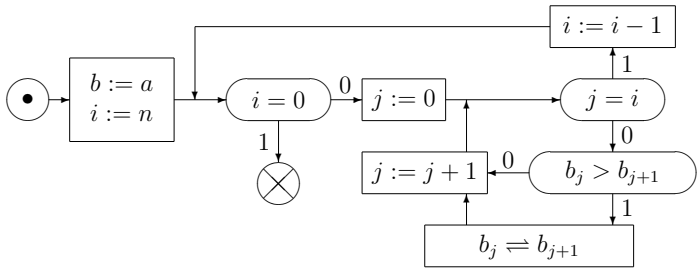
(2.6)

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 $O(\log_2 y)$.

5. БС, реализующая 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 $ord(b)$.



(2.7)

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 (2.7), 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.

2.4 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:

$$\forall\, \theta \in Var(P)^{\bullet}, \quad \text{если } \; Pre^{\theta} = 1, \; \text{ то } \; \theta \xrightarrow{P} \otimes \; \text{ и } \; Post^{P\theta} = 1. \qquad (2.8)$$

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

Глава 3

Метод инвариантов для верификации блок-схем

В этом параграфе излагается метод доказательства утверждений вида $Pre \xrightarrow{P} Post$, называемый **методом инвариантов**. Для его формулировки введём понятия базового множества и базового пути.

3.1 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 $\pi = \alpha_1\alpha_2\ldots$ 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}. Обозначим записью $i_1 i_2 \ldots$ 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 $i_1 i_2 \ldots$ последовательность $\pi_j = \alpha_{i_j} \ldots \alpha_{i_{j+1}}$ 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, … – 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.

3.2 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

$$\varphi^{\theta}_{start(\pi)} = 1 \quad \Rightarrow \quad \varphi^{\theta'}_{end(\pi)} = 1. \tag{3.1}$$

Ниже, при анализе импликаций вида (3.1), $\forall x \in Var(P)$ 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 $u_n^\theta \in L$, 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 [2] и впервые был изложен в [3].

3.3 Обоснование метода инвариантов

Докажем, что если выполнены условия в пунктах 1 и 2 параграфа 3.2, 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 $\alpha_{j_k} \dots \alpha_{j_{k+1}}$ $(k \geq 1)$ являются базовыми относительно N, поэтому, согласно пункту 2 параграфа 3.2, текущие подстановки $\theta_{j_1}, \theta_{j_2}, \dots$ удовлетворяют неравенствам

$$u_{n_1}^{\theta_{j_1}} > u_{n_2}^{\theta_{j_2}} > \dots, \tag{3.2}$$

т.е. в L есть бесконечная строго убывающая последовательность (3.2), что противоречит предположению о фундированности L.

Таким образом, выполнение $Exec$ является конечным. В соответствии со сказанным в конце пункта 3.1 можно представить π в виде конечной последовательности $\pi_1 \dots \pi_k$ путей из $\Pi_{P,M}$.

Согласно определениям выполнения и полного базового множества, а также условиям в пункте 1 параграфа 3.2:

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

Кроме того, $\forall\, i = 1, \dots, k$ текущие подстановки θ_{i-1} и θ_i вычисления $Exec$ до и после прохождения компоненты π_i последовательности $\pi_1 \dots \pi_k$ соответственно удовлетворяют условию $\theta_{i-1} \stackrel{\pi_i}{\to} \theta_i$, поэтому, согласно (3.1),

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

Суммируя все вышесказанное, получаем цепочку импликаций:

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

3.4 Примеры фундированных множеств

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

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

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

Обоснуем фундированность этого множества. Пусть существует бесконечная строго убывающая последовательность

$$(a_1^1, \ldots, a_k^1) > (a_1^2, \ldots, a_k^2) > \ldots \tag{3.3}$$

элементов множества L^k. Из (3.3) и из определения лексикографического порядка следует, что $a_1^1 \geq a_1^2 \geq \ldots$. Поскольку L фундировано, то в этой последовательности неравенств не может быть бесконечного количества строгих неравенств, т.е.

$$\exists\, i_1 :\ a_1^{i_1} = a_1^{i_1+1} = \ldots. \tag{3.4}$$

Из последнего соотношения и из (3.3) следует, что

$$(a_2^{i_1}, \ldots, a_k^{i_1}) > (a_2^{i_1+1}, \ldots, a_k^{i_1+1}) > \ldots \tag{3.5}$$

Применяя изложенные выше рассуждения к (3.5), получаем, что

$$\exists\, i_2 \geq i_1 :\ a_2^{i_2} = a_2^{i_2+1} = \ldots, \tag{3.6}$$

откуда на основании (3.5) 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

$$\exists\, i_k \geq i_{k-1} :\ a_k^{i_k} = a_k^{i_k+1} = \ldots. \tag{3.7}$$

Из (3.4), (3.6), (3.7) следует, что кортежи в (3.3), начиная с кортежа с номером i_k, совпадают, что противоречит предположению. ■

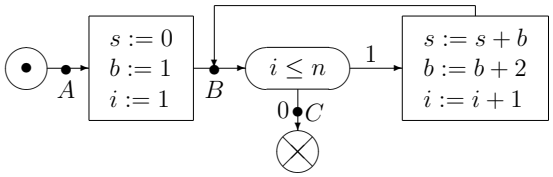
3.5 Применение метода инвариантов

В этом пункте мы рассмотрим применение метода инвариантов для верификации БС, изложенных в пункте 2.3.

3.5.1 Верификация блок-схемы вычисления суммы

Верифицируем БС (2.3) 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:

$$\varphi_A = (n \ge 1) \quad \text{(предусловие)},$$
$$\varphi_B = (s = (i-1)^2) \wedge (b = 2i - 1) \wedge (i \le n + 1),$$
$$\varphi_C = (s = n^2) \quad \text{(постусловие)}.$$

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 (3.1) имеет вид

$$n \ge 1 \;\Rightarrow\; \left\{ \begin{array}{l} s' = (i'-1)^2 \\ b' = 2i' - 1 \\ i' \le n' + 1 \end{array} \right\}. \tag{3.8}$$

   Поскольку $s' = 0$, $b' = 1$, $i' = 1$ и $n' = n$, то (3.8) переписывается в виде

$$n \ge 1 \;\Rightarrow\; \left\{ \begin{array}{l} 0 = 0^2 \\ 1 = 2 \cdot 1 - 1 \\ 1 \le n + 1 \end{array} \right\}, \tag{3.9}$$

   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 ($i \le n$).

   Импликация (3.1) с учетом этого условия имеет вид

$$\left\{ \begin{array}{l} s = (i-1)^2 \\ b = 2i - 1 \\ i \le n + 1 \\ i \le n \end{array} \right\} \;\Rightarrow\; \left\{ \begin{array}{l} s' = (i'-1)^2 \\ b' = 2i' - 1 \\ i' \le n' + 1 \end{array} \right\}. \tag{3.10}$$

Учитывая соотношения $s' = s + b$, $b' = b + 2$ $i' = i + 1$, $n' = n$, заключаем, что импликация (3.10) верна.

3. Рассмотрим этап выполнения БС, на котором происходит переход от B к C. Он возможен только при условии $i > n$. Импликация (3.1) с учетом этого условия имеет вид

$$\left\{ \begin{array}{l} s = (i-1)^2 \\ b = 2i - 1 \\ i \le n + 1 \\ i > n \end{array} \right\} \Rightarrow s' = (n')^2. \tag{3.11}$$

Учитывая равенства $s' = s$, $n' = n$, а также то, что i имеет тип $\mathbf{I}$, и, следовательно, из $i \le n + 1$ и $i > n$ следует, что $i = n + 1$, заключаем, что импликация (3.11) верна.

Для доказательства завершаемости положим

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

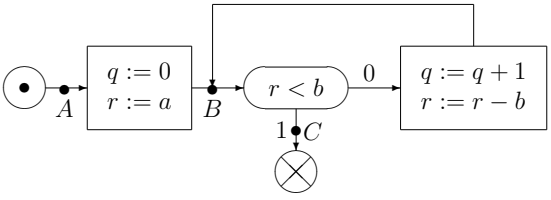
Из доказанного выше соотношения φ_B следует, что при каждом проходе через B текущая подстановка θ удовлетворяет условию $u_B^\theta \in \mathbf{N}$, и при произвольном переходе от B к B по циклу

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

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

3.5.2 Верификация блок-схемы деления с остатком

Верифицируем БС (2.4) относительно предусловия $(a \ge 0) \wedge (b \ge 1)$ и постусловия $(a = bq + r) \wedge (0 \le r < b)$. 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 $(a = bq + r) \wedge (r \ge 0)$.

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 (3.1) имеет вид

$$\left\{ \begin{array}{l} a = bq + r \\ r \geq 0 \\ r \geq b \end{array} \right\} \Rightarrow \left\{ \begin{array}{l} a' = b'q' + r' \\ r' \geq 0 \end{array} \right\}. \tag{3.12}$$

   Учитывая соотношения $a' = a$, $b' = b$, $q' = q + 1$, $r' = r - b$, заключаем, что импликация (3.12) верна.

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

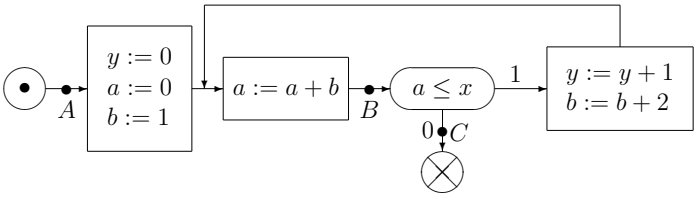
Для доказательства завершаемости положим $N \stackrel{\text{def}}{=} \{B\}$, $L \stackrel{\text{def}}{=} \mathbf{N}$, $u_B \stackrel{\text{def}}{=} r$. Из доказанного выше соотношения φ_B следует, что при каждом проходе через B текущая подстановка θ удовлетворяет условию $u_B^\theta \in \mathbf{N}$, 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.

3.5.3 Верификация блок-схемы извлечения корня

Верифицируем БС (2.5) 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.

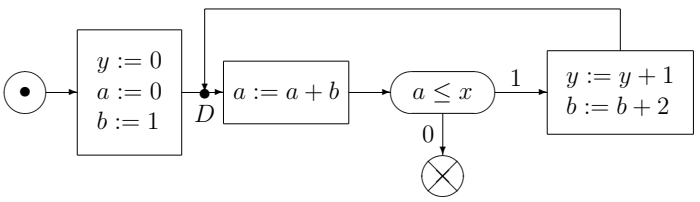
2. При переходе от B к B по циклу верно дополнительное условие $a \leq x$, с учетом которого импликация (3.1) имеет вид

$$\left\{\begin{array}{l} y \geq 0 \\ y^2 \leq x \\ a = (y+1)^2 \\ b = 2y + 1 \\ a \leq x \end{array}\right\} \Rightarrow \left\{\begin{array}{l} y' \geq 0 \\ (y')^2 \leq x' \\ a' = (y'+1)^2 \\ b' = 2y' + 1 \end{array}\right\}. \tag{3.13}$$

   Учитывая соотношения $y' = y + 1$, $b' = b + 2$, $a' = a + b + 2$, $x' = x$, нетрудно установить, что импликация (3.13) 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: $\varphi_D \stackrel{\text{def}}{=} (x \geq a) \wedge (b \geq 1)$.

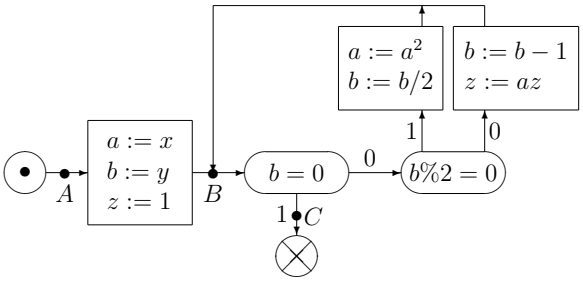
Истинность φ_D при каждом проходе через D следует из того, что

- 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.

3.5.4 Верификация блок-схемы возведения в степень

Верифицируем БС (2.6) относительно предусловия $y \ge 0$ и постусловия $z = x^y$. Выберем точки A, B, C, как показано на рисунке:



Определим инварианты в выбранных точках следующим образом:

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

Нетрудно доказать, что φ_B определен правильно.

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

3.5.5 Верификация блок-схемы сортировки

Верифицируем БС (2.7) относительно предусловия $n \ge 0$ и постусловия $(b = perm(a)) \wedge ord(b)$.

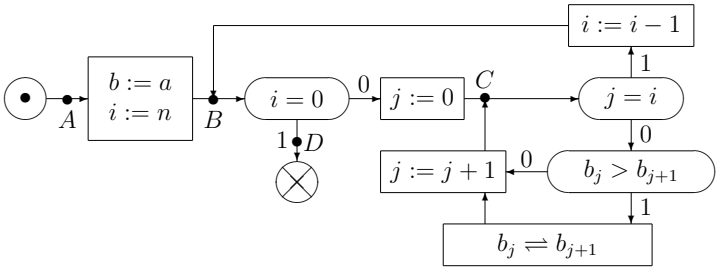
Конъюнктивные члены в постусловии обосновываются раздельно.

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 $ord(b_{0..n})$ соответственно. Инварианты φ_B и φ_C имеют следующий вид:

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

где используются обозначения, введенные в пункте 1.2.4.

Докажем, что инварианты φ_B и φ_C определены правильно.

1. При первом входе в B $i' = n$ и $\varphi_B^{\theta'} = \left\{ \begin{array}{l} 0 \leq n \leq n \\ ord(b_{n..n}) \\ b_{0..n} \leq b_{n+1..n} \end{array} \right\}$. 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 (3.1) имеет вид

$$\left\{ \begin{array}{l} 0 \leq i \leq n \\ ord(b_{i..n}) \\ b_{0..i} \leq b_{i+1..n} \\ i \neq 0 \end{array} \right\} \Rightarrow \left\{ \begin{array}{l} 1 \leq i \leq n \\ 0 \leq 0 \leq i \\ ord(b_{i..n}) \\ b_{0..i} \leq b_{i+1..n} \\ b_{0..0-1} \leq b_0 \end{array} \right\}. \qquad (3.14)$$

Нетрудно видеть, что импликация (3.14) верна.

3. При переходе от C к B верно дополнительное условие $j = i$. Кроме того, меняет свое значение только i. С учетом этого (3.1) 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

$$\left\{\begin{array}{l} 1 \le i \le n \\ ord(b_{i..n}) \\ b_{0..i} \le b_{i+1..n} \\ b_{0..i-1} \le b_i \end{array}\right\} \Rightarrow \left\{\begin{array}{l} 0 \le i-1 \le n \\ ord(b_{i-1..n}) \\ b_{0..i-1} \le b_{i..n} \end{array}\right\}. \qquad (3.15)$$

Нетрудно видеть, что импликация (3.15) верна.

4. При переходе от C к C по короткому циклу верны дополнительные условия $j \ne i$ и $b_j \le b_{j+1}$. Кроме того, меняет свое значение только переменная j ($j' = j+1$). С учетом этого (3.1) имеет вид

$$\left\{\begin{array}{l} 1 \le i \le n \\ 0 \le j < i \\ ord(b_{i..n}) \\ b_{0..i} \le b_{i+1..n} \\ b_{0..j-1} \le b_j \\ b_j \le b_{j+1} \end{array}\right\} \Rightarrow \left\{\begin{array}{l} 1 \le i \le n \\ 0 \le j+1 \le i \\ ord(b_{i..n}) \\ b_{0..i} \le b_{i+1..n} \\ b_{0..j} \le b_{j+1} \end{array}\right\}. \qquad (3.16)$$

Нетрудно видеть, что импликация (3.16) верна.

5. При переходе от C к C по длинному циклу верны дополнительные условия $j \ne i$ 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 (3.1) имеет вид

$$\left\{\begin{array}{l} 1 \le i \le n \\ 0 \le j < i \\ ord(b_{i..n}) \\ b_{0..i} \le b_{i+1..n} \\ b_{0..j-1} \le b_j \\ b_{j+1} < b_j \end{array}\right\} \Rightarrow \left\{\begin{array}{l} 1 \le i \le n \\ 0 \le j+1 \le i \\ ord(b'_{i..n}) \\ b'_{0..i} \le b'_{i+1..n} \\ b'_{0..j} \le b'_{j+1} \end{array}\right\}. \qquad (3.17)$$

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 (3.17).

- Если $j+1=i$, то (3.17) следует из импликации

$$\left\{\begin{array}{l} 1\le i\le n\\ ord(b_{i..n})\\ b_{0..i}\le b_{i+1..n}\\ b_{0..i-2}\le b_{i-1}\\ b_i<b_{i-1}\end{array}\right\}\ \Rightarrow\ \left\{\begin{array}{l} ord(b'_{i..n})\\ b'_{0..i}\le b'_{i+1..n}\\ b'_{0..i-1}\le b'_i\end{array}\right\} \qquad (3.18)$$

  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 (2.7). 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.

Глава 4

Процессные представления блок-схем

Для автоматизации верификации БС P целесообразно сначала преобразовать эту БС в определяемый ниже граф G_P, который называется **процессом**, соответствующим БС P. Метод верификации процессов аналогичен изложенному выше методу верификации БС путем построения утверждений во внутренних точках БС.

4.1 Понятие процесса

4.1.1 Действия

Будем называть **действием** конечную последовательность $A = a_1 \dots a_n$ (где $n \geq 0$, т.е. данная последовательность м.б. пустой, и в этом случае она обозначается символом ε), каждая компонента a_i которой имеет вид $x := e$, где $x \in Var$, $e \in Tm$, $\tau(x) = \tau(e)$, или $\{\beta\}$, где $\beta \in Fm$. Множество всех действий обозначается символом Act. Если $A, A' \in Act$, то запись AA' обозначает конкатенацию последовательностей A и A'. $\forall\, A \in Act$ запись $Var(A)$ обозначает множество всех переменных, входящих в A.

Пусть $A \in Act$, $Var(A) \subseteq X \subseteq Var$ и $\theta, \theta' \in X^\bullet$. Запись

$$\theta \overset{A}{\to} \theta'$$

обозначает следующее утверждение: при выполнении действия A подстановка θ преобразуется в подстановку θ'. Истинность утверждения $\theta \overset{A}{\to} \theta'$ можно определить индукцией по структуре A:

- $\theta \overset{\varepsilon}{\to} \theta'$ эквивалентно равенству $\theta = \theta'$,

- $\theta \overset{\{\beta\}A}{\longrightarrow} \theta'$ эквивалентно утверждениям $\beta^\theta = 1$ и $\theta \overset{A}{\to} \theta'$,
- $\theta \overset{(x:=e)A}{\longrightarrow} \theta'$ эквивалентно утверждению $(e/x)\theta \overset{A}{\to} \theta'$.

Пусть $A \in Act$, $Var(A) \subseteq X \subseteq Var$ и $\varphi, \psi \in Fm(X)$. Запись

$$\varphi \overset{A}{\to} \psi$$

обозначает следующее утверждение:

$$\forall\, \theta, \theta' \in X^\bullet, \text{ если } \theta \overset{A}{\to} \theta', \text{ то из } \varphi^\theta = 1 \text{ следует } \psi^{\theta'} = 1.$$

Утверждение $\varphi \overset{A}{\to} \psi$ можно определить индукцией по структуре A:

- $\varphi \overset{\varepsilon}{\to} \psi$ эквивалентно импликации $\varphi \to \psi$,
- $\varphi \overset{A(x:=e)}{\longrightarrow} \psi$ эквивалентно утверждению $\varphi \overset{A}{\to} \psi^{(e/x)}$,
- $\varphi \overset{A\{\beta\}}{\longrightarrow} \psi$ эквивалентно утверждению $\varphi \overset{A}{\to} (\beta \to \psi)$.

4.1.2 Процессы и их выполнение

Процессом будем называть граф G со следующими свойствами:

- G имеет выделенные вершины $\odot$ и $\otimes$, называемые **начальной** и **терминальной** вершинами соответственно;
- каждому ребру α графа G сопоставлена метка $A_\alpha \in Act$;
- $\otimes$ – единственная вершина G, из которой не выходят рёбра.

Пусть G – процесс. Будем обозначать записями $V(G)$ и $Var(G)$ совокупность всех вершин G и переменных, входящих в G, соответственно. Будем предполагать, что в каждом процессе G среди его переменных присутствует переменная at_G, множеством значений которой является $V(G)$, и метка каждого ребра α содержит присваивание вида $at_G := v$, где v – конец ребра α. В записи меток рёбер процесса данное присваивание будет опускаться.

Выполнение процесса G – это обход вершин графа G, начиная с вершины $\odot$, с выполнением действий, сопоставленных проходимым рёбрам. С каждым шагом $i \geq 0$ выполнения G связаны вершина $v_i \in V(G)$ и подстановка $\theta_i \in Var(G)^\bullet$ (называемые **текущей вершиной** и **текущей подстановкой** на шаге i). Если $v_i = \otimes$, то выполнение G на шаге i завершается, иначе выполнение на шаге i заключается в

- выборе ребра α, выходящего из v_i и удовлетворяющего условию

$$\exists\, \theta \in Var(G)^{\bullet} : \theta_i \xrightarrow{A_\alpha} \theta, \tag{4.1}$$

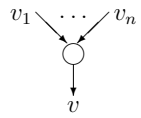
- замене текущей подстановки θ_i на подстановку θ_{i+1}, удовлетворяющей соотношению $\theta_i \xrightarrow{A_\alpha} \theta_{i+1}$, и

- переходе в конец выбранного ребра α, который будет текущей вершиной v_{i+1} на $(i+1)$-м шаге выполнения.

Если на шаге i не существует ребра α, удовлетворяющего условию (4.1), то выполнение процесса G завершается на шаге i.

4.2 Процессы, соответствующие блок-схемам

Пусть задана БС P. Алгоритм построения процесса G_P, соответствующего БС P, состоит из следующих шагов.

1. Удаляются пустые операторы: каждый фрагмент вида 

заменяется на 

.

2. Удаляется начальная вершина P и выходящее из неё ребро, конец этого ребра – начальная вершина $\odot$ графа G_P.

3. Удаляются терминальные вершины P, кроме одной, которая является терминальной вершиной $\otimes$ графа G_P, рёбра с концами в удаляемых вершинах перенаправляются в $\otimes$.

4. Каждая оставшаяся вершина v БС P является вершиной G_P, и

   - если v имеет вид $\boxed{x := e}$ и P содержит ребро $v \to v'$, то G_P содержит ребро $v \xrightarrow{x:=e} v'$,
   - если v имеет вид (β) и P содержит рёбра $v \xrightarrow{1} v'$ и $v \xrightarrow{0} v''$, то G_P содержит рёбра $v \xrightarrow{\{\beta\}} v'$ и $v \xrightarrow{\{\bar{\beta}\}} v''$.

Получившийся процесс G_P может быть редуцирован путем применения операции **удаления вершины**: если v – вершина процесса G_P,

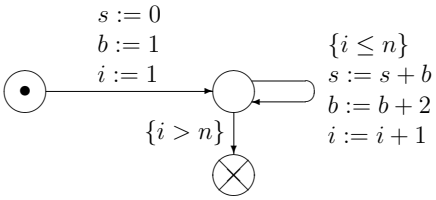
отличная от $\odot$ и $\otimes$, в G_P нет ребер вида $v \stackrel{A}{\to} v$, и списки ребер, входящих в v и выходящих из v, имеют вид

$$v_1 \stackrel{A_1}{\to} v, \ldots, v_k \stackrel{A_k}{\to} v \quad \text{и} \quad v \stackrel{A'_1}{\to} v'_1, \ldots, v \stackrel{A'_{k'}}{\to} v'_{k'} \tag{4.2}$$

соответственно, то редукция G_P путём удаления v из G_P заключается в удалении этой вершины и замене всех ребер из (4.2) на ребра вида $v_i \stackrel{A_i A'_j}{\to} v'_j$, где $i \in \{1, \ldots, k\}$, $j \in \{1, \ldots, k'\}$.

Результат применения этой операции обозначается той же записью G_P и рассматривается как процесс, эквивалентный в некотором смысле исходному процессу. Данная операция м.б. применена несколько раз.

Например, процесс, полученный редукцией процесса, построенного по БС (2.3), имеет следующий вид (компоненты действий записываются столбиком):



Каждому выполнению БС P, заканчивающемуся в терминальной вершине, соответствует некоторое выполнение процесса G_P, заканчивающееся в терминальной вершине, и обратно.

4.3 Верификация процессных представлений блок-схем

Пусть задан процесс G. Для каждого пути $\pi = (v_0 \stackrel{A_1}{\to} \ldots \stackrel{A_k}{\to} v_k)$ в G записи $start(\pi)$ и $end(\pi)$ обозначают начало v_0 и конец v_k пути π соответственно и запись A_π обозначает конкатенацию $A_1 \ldots A_k$. Путь π называется **циклом**, если $start(\pi) = end(\pi)$.

Множество вершин M процесса G называется **базовым**, если для каждого цикла в G хотя бы одна из его вершин лежит в M. Если, кроме того, M содержит $\odot$ и $\otimes$, то M называется **полным** базовым множеством. Путь π в G называется **базовым** относительно базового множества M, если $start(\pi)$ и $end(\pi)$ лежат в M, а остальные вершины π не лежат в M. Нетрудно доказать, что множество $\Pi_{G,M}$ всех базовых относительно базового множества M путей в G конечно.

Теорема 1

Пусть заданы БС P и её спецификация, выражаемая предусловием Pre и постусловием $Post$. Тогда утверждение $Pre \overset{P}{\to} Post$ верно, если

- существует полное базовое множество M вершин G_P, причем с каждой вершиной $m \in M$ связана формула $\varphi_m \in Fm$, и
  - $\varphi_\odot = Pre$, $\varphi_\otimes = Post$,
  - $\forall\, \pi \in \Pi_{G_P,M}$ верно утверждение $\varphi_{start(\pi)} \overset{A_\pi}{\to} \varphi_{end(\pi)}$,
- существуют базовое множество N вершин G_P и фундированное множество L, такие, что с каждой вершиной $n \in N$ связан терм $u_n \in Tm(Var(P))$, причем выполнены условия:
  - если при каком-либо выполнении G_P текущей вершиной в какой-либо момент является вершина $n \in N$, то текущая подстановка θ в этот момент удовлетворяет условию $u_n^\theta \in L$,
  - $\forall\, \pi \in \Pi_{G_P,N}$ верно утверждение

$$(u_{start(\pi)} = x) \overset{A_\pi}{\to} (u_{end(\pi)} < x),$$

где x – новая переменная типа $\tau(u_n)$ (не входящая в $Var(G_P)$).

Доказательство.

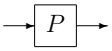
Если существуют множества M, N и L, упоминаемые в формулировке этой теоремы, то по ним нетрудно построить аналогичные множества M_P, N_P, L_P, необходимые для доказательства $Pre \overset{P}{\to} Post$ на основе метода инвариантов. Действительно, если вершина m процесса G_P принадлежит множеству M, то этой вершине соответствует некоторая вершина в исходной БС P. Мы зачисляем в M_P точки на всех рёбрах P, входящих в эту вершину, и сопоставляем каждой такой точке m формулу φ_m. Аналогично определяется N_P. В качестве L_P можно взять L. Проверка всех условий, упоминаемых в формулировке метода инвариантов, является несложной. ■

Глава 5

Верификация операторных программ

5.1 Понятие операторной программы

Еще один способ описания программ заключается в представлении их в операторной форме. Будем называть программы, представленные в операторной форме, **операторными программами**. В настоящий момент операторная форма описания программ является наиболее широко распространенной.

Операторная программа представляет собой оператор из определяемого ниже множества Op операторов. В нижеследующем определении мы поясняем смысл каждого оператора $P \in Op$ путем указания соответствующего ему фрагмента БС, который будем обозначать записью вида 

.

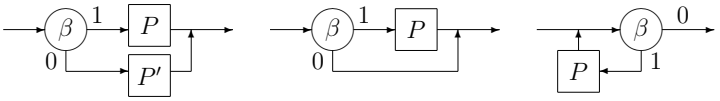
1. $\forall\, x \in Var, \forall\, e \in Tm : \tau(x) = \tau(e)$, запись $x := e$ является оператором из Op, ему соответствует фрагмент БС 

.

2. $\forall\, \beta \in Fm, \forall\, P, P' \in Op$ записи

$$\textbf{if } \beta \textbf{ then } P \textbf{ else } P', \quad \textbf{if } \beta \textbf{ then } P, \quad \textbf{while } \beta \textbf{ do } P \tag{5.1}$$

являются операторами из Op, им соответствуют фрагменты БС

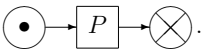


(последняя из программ (5.1) называется **циклом**).

3. $\forall\, P_1, \ldots, P_k \in Op$ $(k \geq 1)$ запись $\{P_1; \ldots; P_k\}$ является оператором, ему соответствует фрагмент БС .

   Данный оператор может обозначаться также другой записью, в которой входящие в него операторы $P_1, \ldots, P_k$ расположены в столбик (при этом фигурные скобки могут отсутствовать), в этом случае точка с запятой для разделения $P_1, \ldots, P_k$ не используется.

$\forall\, P \in Op$ БС, соответствующая этому оператору, имеет вид

 .

Мы будем обозначать эту БС тем же символом P.

Можно доказать, что определенная выше система операторов является алгоритмически полной, т.е. любую функцию, вычислимую по Тьюрингу, можно представить в виде операторной программы.

Наряду с определенными выше видами операторов используются также и другие виды операторов, например:

- **do** P **while** β, где $P \in Op$ и $\beta \in Fm$, этот оператор эквивалентен оператору $\{P;$ **while** β **do** $P\}$;

- **for** $i := e_1$ **step** e_2 **until** e_3 **do** P, где i – переменная типа **I**, e_1, e_2, e_3 – термы типа **I**, и $P \in Op$, этот оператор эквивалентен оператору

$$\{i := e_1;\ \textbf{while}\ i \neq e_3\ \textbf{do}\ \{P;\ i := i + e_2\};\ P\};$$

- **go to** L, где L – метка какого-либо оператора, для возможности использования этого оператора необходимо ввести новое синтаксическое правило: если P – оператор и L – символ, то запись $L : P$ является оператором (в котором L рассматривается как метка);

- **stop**, выполнение этого оператора приводит к завершению работы.

5.2 Примеры операторных программ

Все переменные в излагаемых ниже примерах имеют тип **I**. Для каждой из представленных программ мы указываем её предусловие Pre и постусловие $Post$.

1. Программа P_1, вычисляющая $y = \lfloor \sqrt{x} \rfloor$:

$$\{y := 0;\ a := 1;\ b := 1\}$$
$$\textbf{while}\ (a \le x)\ \textbf{do}\ \left\{ \begin{array}{l} y := y + 1 \\ a := a + b + 2 \\ b := b + 2 \end{array} \right\}$$

   $Pre : x \ge 0$, $Post : (y \ge 0) \wedge (y^2 \le x < (y+1)^2)$.

2. Программа P_2, вычисляющая наибольший общий делитель (НОД) $c = gcd(a, b)$ положительных чисел a и b:

$$\{c := a;\ d := b\}$$
$$\textbf{while}\ (c \ne d)\ \textbf{do}\ \{\textbf{if}\ c > d\ \textbf{then}\ c := c - d\ \textbf{else}\ d := d - c\}$$

   $Pre : a, b > 0$, $Post : c = \max\{x \mid \exists\, a', b' : a = xa', b = xb'\}$.

3. Программа P_3, вычисляющая НОД $c = gcd(a, b)$ положительных чисел a и b:

$$\{x := a;\ y := b;\ z := 1\}$$
$$\textbf{while}\ (x\%2 = 0)\ \textbf{do}\left\{ \begin{array}{l} \textbf{if}\ y\%2 = 0\ \textbf{then}\ (y, z) := (y/2, 2z) \\ x := x/2 \end{array} \right\}$$
$$\textbf{while}\ (y\%2 = 0) \vee (x \ne y)\ \textbf{do}$$
$$\left\{ \begin{array}{l} \textbf{if}\ (y\%2 = 1)\ \textbf{then}\ (x, y) := (y, |x - y|) \\ y := y/2 \end{array} \right\}$$
$$c := xz$$

   Предусловие и постусловие для P_3 – те же, что и для P_2.

5.3 Метод инвариантов для верификации операторных программ

Как и для БС, спецификация программ из Op м.б. задана в виде предусловия и постусловия. Задача верификации программы $P \in Op$ относительно предусловия Pre и постусловия $Post$ из $Fm(Var(P))$ заключается в обосновании утверждения

$$Pre \overset{P}{\to} Post,$$

где $\forall\, P \in Op$, $\forall\, \varphi, \psi \in Fm(Var(P))$ $\varphi \overset{P}{\to} \psi$ обозначает утверждение

$$\forall\, \theta \in Var(P)^{\bullet}, \quad \text{если}\ \ \varphi^{\theta} = 1,\ \ \text{то}\ \ \theta \overset{P}{\to} \otimes\ \ \text{и}\ \ \psi^{P\theta} = 1, \tag{5.2}$$

и P в (5.2) обозначает соответствующую БС.

В описании излагаемого ниже метода инвариантов предполагается, что анализируемый оператор P состоит только из операторов присваивания, операторов вида (5.1) и операторов вида $\{P_1;\ \ldots;\ P_k\}$.

Метод инвариантов обоснования утверждения вида $\varphi \overset{P}{\to} \psi$, где $P \in Op$ и $\varphi, \psi \in Fm(Var(P))$, имеет следующий вид.

1. Если $P = (x := e)$ то обоснование утверждения $\varphi \overset{P}{\to} \psi$ сводится к доказательству импликации $\varphi \to \psi^{(e/x)}$.

2. Если $P =$ **if** β **then** P_1 **else** P_2, то обоснование утверждения $\varphi \overset{P}{\to} \psi$ сводится к обоснованию утверждений $\varphi \wedge \beta \overset{P_1}{\to} \psi$ и $(\varphi \wedge \bar{\beta}) \overset{P_2}{\to} \psi$.

3. Если $P =$ **if** β **then** P_1, то обоснование утверждения $\varphi \overset{P}{\to} \psi$ сводится к обоснованию утверждений $\varphi \wedge \beta \overset{P_1}{\to} \psi$ и $(\varphi \wedge \bar{\beta}) \to \psi$.

4. Если $P =$ **while** β **do** P, то обоснование утверждения $\varphi \overset{P}{\to} \psi$ сводится к построению формулы $I \in Fm$ (называемой **инвариантом цикла**), фундированного множества L и терма u, таких, что

   - верны импликации $\varphi \to I$, $I \to (u \in L)$, $(I \wedge \bar{\beta}) \to \psi$ и
   - верно утверждение

   $$I \wedge \beta \wedge (u = z) \overset{P}{\to} I \wedge (u < z), \tag{5.3}$$

   где z – новая переменная (т.е. не входит в I, β, u, P).

5. Если $P = \{P_1; \ldots; P_k\}$, где $k \geq 2$, то обоснование утверждения $\varphi \overset{P}{\to} \psi$ сводится к построению формулы $\eta \in Fm$, такой, что верны утверждения $\varphi \overset{P'}{\to} \eta$ и $\eta \overset{P_k}{\to} \psi$, где $P' = \{P_1; \ldots; P_{k-1}\}$.

5.4 Пример верификации операторной программы

Приведем пример верификации программы P_1 из пункта 5.2. Данная программа имеет вид $\{B_1; B_2\}$, где B_2 является циклом. Определим

$$\eta \overset{\text{def}}{=} (y^2 \leq x) \wedge (a = (y+1)^2) \wedge (b = 2y+1) \wedge (b > 0) \wedge (x - a + b \geq 0).$$

Нетрудно доказать, что

- $Pre \overset{B_1}{\to} \eta$, т.е. $(x \geq 0) \to \eta^{(0/y,1/a,1/b)}$,
- $\eta \overset{B_2}{\to} Post$,
- в качестве инварианта цикла B_2 можно взять η,
- в качестве L можно взять множество $\mathbf{N}$ натуральных чисел и в качестве u – терм $x - a + b$.

Отметим, что конъюнктивный член $(u < z)$ в заключении импликации (5.3) для данного случая обосновывается импликацией

$$(b > 0) \wedge (x - a + b = z) \to (x - a < z).$$

Глава 6

Распределенные программы

6.1 Понятие распределенной программы

6.1.1 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.

6.1.2 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.

6.1.3 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 $\forall\, i \in I\ at^{\theta}_{P_i} = \otimes$.

6.1.4 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.).

6.1.5 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'.

6.1.6 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.

6.2 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

$$\varphi \overset{A}{\to} \psi \tag{6.1}$$

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

$$(e^{(\vec{x}'/\vec{x})} = e') \to \varphi^{(\vec{x}'/\vec{x})}, \tag{6.2}$$

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 (6.2),
- $(\vec{x}'/\vec{x})$ обозначает подстановку $(x'_1/x_1, \ldots, x'_n/x_n)$.

Утверждение (6.1) 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:

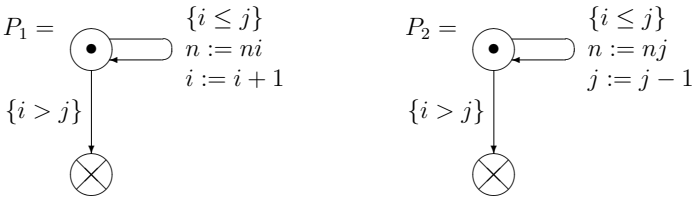
$$\begin{array}{l} Pre \to \varphi, \;\; \varphi \stackrel{A}{\to} \varphi, \;\; \left\{ \begin{array}{l} \varphi \\ \bigwedge_{i \in I} (at_{P_i} = \otimes) \end{array} \right\} \to Post, \\ \varphi \to (u \in L), \;\; \varphi \wedge (u = x) \stackrel{A}{\to} (u < x), \end{array} \tag{6.3}$$

где A – метка какого-либо ребра произвольного процесса из P, x – новая переменная типа $\tau(u)$, не встречающаяся за пределами последней формулы из (6.3).

6.3 Примеры распределенных программ

6.3.1 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:

$\varphi = \left\{ \begin{array}{l} i \le j + 1, \ \ n = (i-1)!\frac{k!}{j!} \\ (at_{P_1} = \otimes) \vee (at_{P_2} = \otimes) \to (i > j) \end{array} \right\}$,

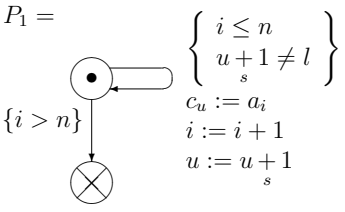
$L = \mathbf{N}$, $u = j + 1 - i + [\![at_{P_1} = \odot]\!] + [\![at_{P_2} = \odot]\!]$.

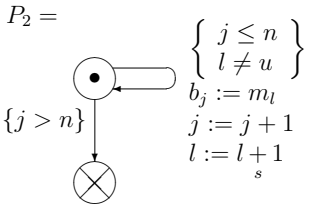
6.3.2 Передача сообщений через буфер

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: $\begin{cases} Pre = \ (i = j = 1) \wedge (u = l = 0) \wedge (n \geq 1) \wedge (s \geq 1), \\ Post = \ \bigwedge_{i=1}^{n} (b_i = a_i). \end{cases}$

Для верификации можно использовать следующий инвариант φ, фундированное множество L и функцию u:

$$\varphi = \left\{ \begin{array}{l} (i \leq n+1) \wedge (j \leq n+1) \\ a_{1..i-1} = b_{1..j-1} c_{l..u\atop s-1} \\ (at_{P_1} = \otimes) \rightarrow (i = n+1) \\ (at_{P_2} = \otimes) \rightarrow (j = n+1) \end{array} \right\},$$

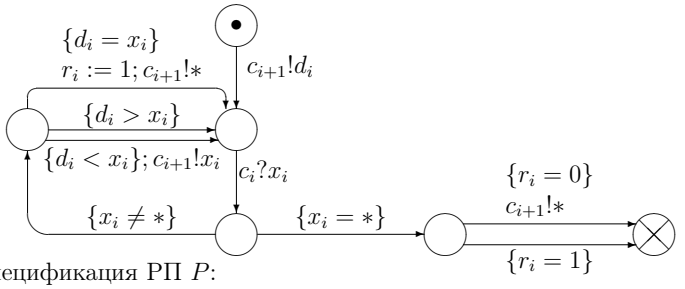
$L = \mathbf{N}$, $u = (n+1-i) + (n+1-j) + [\![at_{P_1} = \odot]\!] + [\![at_{P_2} = \odot]\!]$.

Запись $a_{1..i-1} = b_{1..j-1} c_{l..u\atop s-1}$ 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 $c_{l..u\atop s-1}$ массива c. При этом если $0 < u < l$, то $c_{l..u\atop s-1} \stackrel{\text{def}}{=} c_{l..s-1} c_{0..u-1}$, и в остальных случаях указанная выше запись тоже определяется естественным образом.

6.3.3 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:

$$\begin{cases} Pre = (n \geq 2) \wedge \bigwedge_{i=1}^{n} \Big((r_i = 0) \wedge (d_i \neq *) \wedge (\bigwedge_{j \neq i} (d_j \neq d_i)) \Big), \\ Post = \bigvee_{i=1}^{n} \Big((r_i = 1) \wedge (\bigwedge_{j \neq i} (r_j = 0) \wedge (d_j < d_i)) \Big). \end{cases}$$

6.3.4 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.

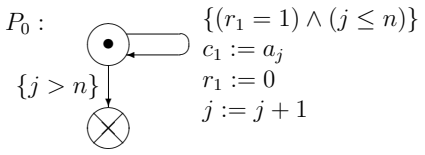
Идея излагаемой ниже РП заключается в следующем. Процесс P_0 получает по очереди элементы массива a, и затем эти элементы передаются процессам $P_1, \ldots, P_n$ по цепочке «слева направо». В каждый момент времени процесс P_i $(i = 1, \ldots, n)$ хранит в b_i некоторый элемент, полученный от предыдущего 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.

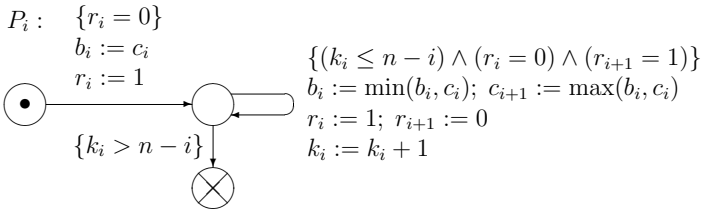
Алгоритм работы процессов $P_0, \ldots, P_n$ имеет следующий вид. P_0 записывает очередной полученный элемент в переменную c_1, после этого

- 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: $\begin{cases} Pre = (n \ge 1) \wedge (j = 1) \wedge \bigwedge_{i=1}^{n}((r_i = 1) \wedge (k_i = 1)), \\ Post = perm(a, b) \wedge ord(b). \end{cases}$

6.4 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 [29] MPI-программы перемножения матриц. РП P имеет вид $\{P_0, \ldots, P_n\}$, где $n \ge 1$. Процесс P_0 называется **менеджером**, процессы $P_1, \ldots, P_n$ называются **работниками**.

6.4.1 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).

6.4.2 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

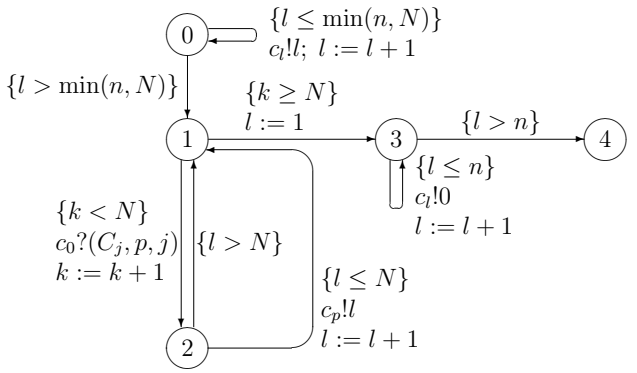
$$\forall\, j = 1, \ldots, N \quad C_j = A_jB. \tag{6.4}$$

6.4.3 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).

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:



(6.5)

$Pre(P_0) = (N \ge 1) \wedge (l = 1) \wedge (k = 0)$.

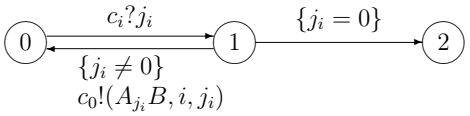
Поясним работу процесса P_0:

- 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.

Процесс P_i для $i > 0$ («работник»)

Процесс P_i имеет следующий вид:



(6.6)

Символ i в (6.6) 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.

6.4.4 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.

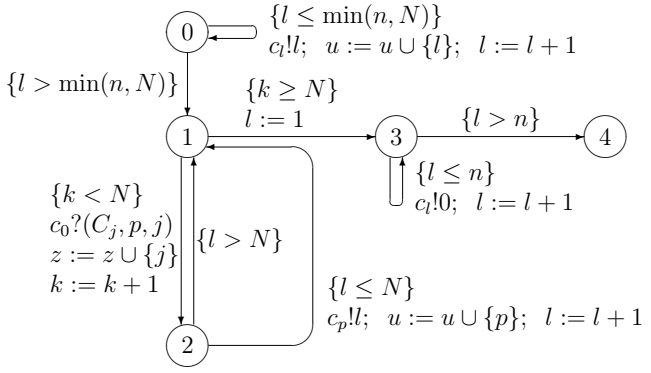
Процессы, получаемые путем добавления вышеуказанных присваиваний, будем называть **дополненными** процессами.

6.4.5 Верификация программы перемножения матриц

Для доказательства утверждения о том, что определенная в пункте 6.4.3 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: $\forall\, \theta \in Var(P)^\bullet$

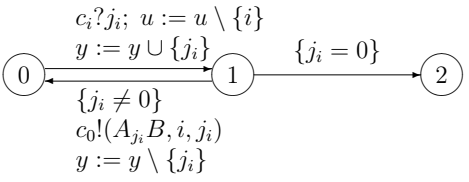
- $u^\theta \subseteq \{1, \ldots, n\}$, u^θ состоит из номеров каналов из $\{c_1, \ldots, c_n\}$ с непустым содержимым в состоянии θ,
- $y^\theta \subseteq \{1, \ldots, N\}$, y^θ состоит из номеров строк матрицы A, для которых в состоянии θ вычисляется их произведение на B, но результат вычисления пока 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:



(6.7)

$\forall\, i = 1, \ldots, n$ дополненный процесс P_i имеет следующий вид:



(6.8)

Теоремы, обосновывающие корректность P

Будем использовать следующее обозначение: если d – множество кортежей, то d^k обозначает множество k–х компонентов кортежей из d.

Теорема 2

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. $\forall\, i = 1,\ldots,n \quad i \in y \;\Leftrightarrow\; at_{P_i} \neq 0$,
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 $\{l \le N\}$ 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 3
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 2, 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

$$\begin{array}{l}\exists\, i \in \{1, \ldots, n\}\text{: } \pi' \text{ содержит бесконечно много} \\ P_i\text{-переходов вида } 0 \to 1.\end{array} \tag{6.9}$$

Поскольку π' не содержит P_0-переходов, то значение $|x^{\theta}_{c_i}|$ не может увеличиваться, и согласно (6.9) оно бесконечно уменьшается, что невозможно. ■

Теорема 4

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 2. Из отсутствия переходов с началом в θ следует, что $k^{\theta} < N$ и $x^{\theta}_{c_0} = \emptyset$, поэтому из утверждения 1 теоремы 2 следует, что в θ верно

$$x_{c_1} \sqcup \ldots \sqcup x_{c_n} \sqcup y \sqcup z = \{1, \ldots, l-1\} \subseteq \{1, \ldots, N\}. \tag{6.10}$$

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 2, что влечет неверное утверждение $0 \in \{1, \ldots, N\}$. Поэтому

$$\forall\, i > 0 \quad at^{\theta}_{P_i} = 0. \tag{6.11}$$

По утверждению 4 теоремы 2, из (6.11) следует, что $y^{\theta} = \emptyset$. Из (6.11) и из отсутствия переходов с началом в θ следует, что для каждого $i = 1, \ldots, n\ |x^{\theta}_{c_i}| = 0$, поэтому из (6.10) и из утверждения 8 теоремы 2 следует, что

$$k^{\theta} = |z^{\theta}| = l^{\theta} - 1 \leq N. \tag{6.12}$$

Докажем, что соотношения (6.12) 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 2 для θ_1 следует, что $l^{\theta_1} - 1 \le N$, что в сочетании с $l^{\theta_1} > N$ дает соотношение $N < l^{\theta_1} \le N + 1$, из которого следует равенство $l^{\theta_1} - 1 = N$. Поскольку при переходах от θ' к θ значение l не убывает, то $l^\theta - 1 \ge N$, что в сочетании с (6.12) 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 (6.12).

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 $k^{\theta'} \ge N$. Поскольку в θ' верны утверждения 1 и 8 теоремы 2, то

$$N \le k^{\theta'} = |z^{\theta'}| \le N, \quad \forall\, i = 0, 1, \ldots, n \;\; x^{\theta'}_{c_i} = \emptyset, \;\; y^{\theta'} = \emptyset,$$

откуда следует $k^{\theta'} = |z^{\theta'}| = N$.

Из истинности в θ' утверждения 4 теоремы 2 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 2, которое в данном случае имеет вид

$$\forall\, j \in z = \{1, \ldots, N\} \;\; C_j = A_j B, \tag{6.13}$$

следует, что и в терминальном состоянии θ верно (6.13). ■

Таким образом, P завершает свою работу после конечного числа шагов, и после завершения выполнения P верно постусловие (6.4). ■

Глава 7

Задачи и исследовательские проблемы

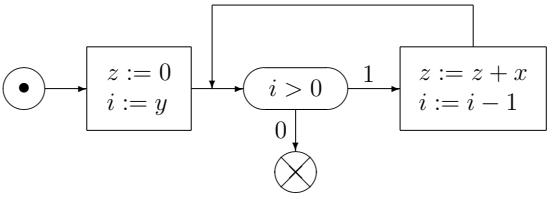
В первых двух параграфах данной главы приводятся задачи доказательства корректности программ, представленых в виде БС или в операторной форме. Каждая из этих задач заключается в верификации программы, для которой задано предусловие Pre и постусловие $Post$. Если тип какой-либо переменной в этих программах не указан явно, то он по умолчанию равен **I**. В некоторых задачах приведено несколько программ, соответствующих одной и той же спецификации. Если какие-либо две из излагаемых ниже программ являются реализацией в виде БС и в операторной форме одного и того же алгоритма, они приводятся в одном и том же пункте. Операция деления целых чисел в задачах понимается как целочисленное деление.

7.1 Верификация программ без массивов

7.1.1 Произведение двух чисел

$$Pre : y \geq 0, \quad Post : z = xy.$$

1.



$\{z := 0;\ i := y\}$
while $i > 0$ **do** $\{z := z + x;\ i := i - 1\}$

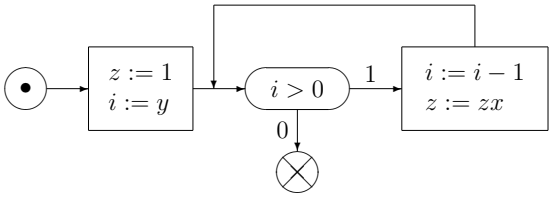
2.

$\{z := 0;\ i := x;\ j := y\}$
while $j \neq 0$ **do** $\left\{ \begin{array}{ll} \textbf{if } j\%2 = 1 & \textbf{then}\{z := z + i;\ j := j - 1\} \\ & \textbf{else } \{i := 2i;\ j := j/2\} \end{array} \right\}$

7.1.2 Возведение в степень

$$Pre : y \geq 0, \quad Post : z = x^y.$$

1.



$\{z := 1;\ i := y\}$
while $i > 0$ **do** $\{i := i - 1;\ z := xz\}$

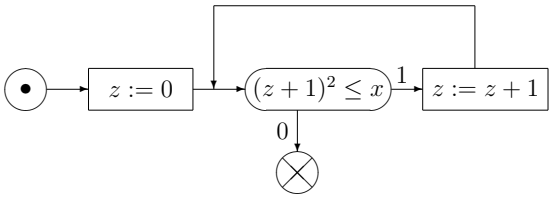
2.

$\{\ z := 1;\ i := x;\ j := y\ \}$
while $j \neq 0$ **do** $\left\{ \begin{array}{ll} \textbf{if } j\%2 = 1 & \textbf{then } \{j := j - 1;\ z := iz\} \\ & \textbf{else } \{i := i^2;\ j := j/2\} \end{array} \right\}$

7.1.3 Извлечение квадратного корня

$$Pre : x \geq 0, \quad Post : z = \lfloor \sqrt{x} \rfloor.$$

1.



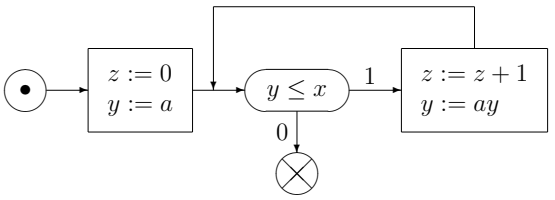
$z := 0$
while $(z + 1)^2 \leq x$ **do** $z := z + 1$

2.

$\{i := 1;\ j := 1;\ a := 0;\ b := 0;\ z := 0\}$
while $j \leq x$ **do** $\{i := 2i;\ j := 4j\}$
while $i > 1$ **do** $\left\{ \begin{array}{l} \{i := i/2;\ j := j/4;\ b := b/2;\ c := a + b + j\} \\ \textbf{if } c \leq x \textbf{ then } \{a := c;\ b := b + 2j;\ z := z + i\} \end{array} \right\}$

7.1.4 Извлечение логарифма

$$Pre : x \geq 1, a \geq 2, \quad Post : z = \lfloor \log_a x \rfloor.$$



$\{z := 0;\ y := a\}$
while $y \neq x$ **do** $\{z := z + 1;\ y := ay\}$

Указание: выразить постусловие соотношением $a^z \leq x < a^{z+1}$.

7.1.5 Вычисление частного и остатка от деления целых чисел

$$Pre : (a \geq 0) \wedge (b > 0), \quad Post : (a = bq + r) \wedge (0 \leq r < b).$$

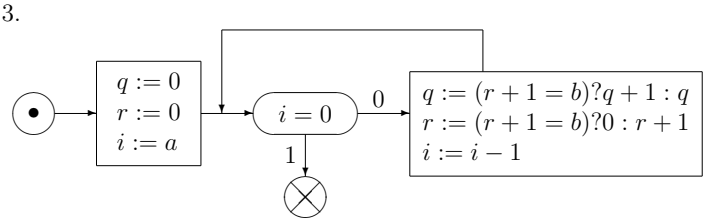
1.

$\{r := a;\ q := 0\}$
while $b \leq r$ **do** $\{r := r - b;\ q := q + 1\}$

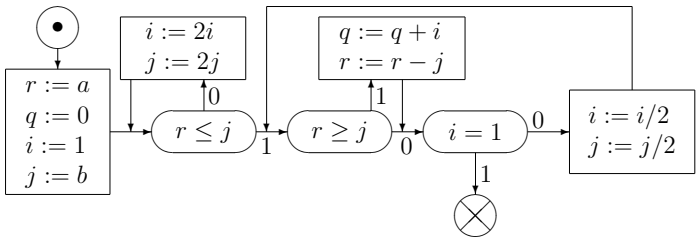
2.

$\{r := a;\ q := 0;\ i := b\}$
while $i \leq r$ **do** $i := 2i$
while $i \neq b$ **do** $\left\{ \begin{array}{l} q := 2q \\ i := i/2 \\ \textbf{if } i \leq r \textbf{ then } \{r := r - i;\ q := q + 1\} \end{array} \right\}$

3.



4. (Аппаратная реализация деления целых чисел.)



$\{r := a;\ q := 0;\ i := 1;\ j := b\}$
while $r > j$ **do** $\{i := 2i;\ j := 2j\}$
do $\left\{ \begin{array}{l} \textbf{if } r \geq j \textbf{ then } \{q := q + i;\ r := r - j\} \\ \textbf{if } i \neq 1 \textbf{ then } \{i := i/2;\ j := j/2\} \end{array} \right\}$ **while** $i \neq 1$

7.1.6 Наибольший общий делитель

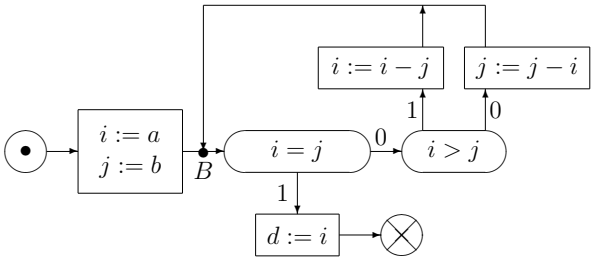
$$Pre : (a > 0) \wedge (b > 0), \quad Post : d = gcd(a, b),$$

где $d = gcd(a, b)$ – сокращенная запись формулы

$$d = \max\{x \mid \exists\, a', b' : a = xa', b = xb'\}$$

(gcd – сокращение от greatest common divisor).

1.



$\{i := a;\ j := b\}$
while $i \neq j$ **do** { **if** $i > j$ **then** $i := i - j$ **else** $j := j - i$ }
$d := i$

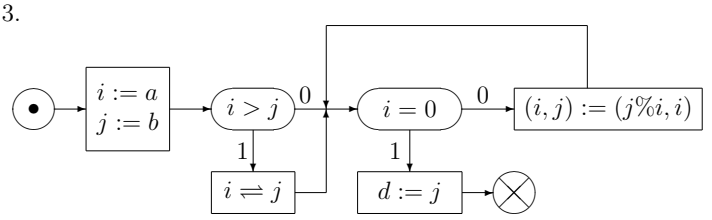
Указание: $\varphi_B = \left\{ \begin{array}{l} (a > 0) \wedge (b > 0) \\ (i > 0) \wedge (j > 0) \\ gcd(i, j) = gcd(a, b) \end{array} \right\}$, $u_B = \max(y_1, y_2)$.

2.

$\{i := a;\ j := b\}$
while $i \neq j$ **do** $\left\{ \begin{array}{l} \textbf{while } i > j \textbf{ do } i := i - j \\ \textbf{while } j > i \textbf{ do } j := j - i \end{array} \right\}$
$d := i$

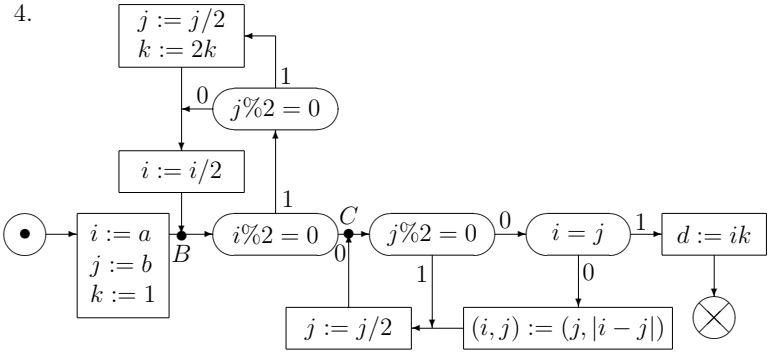
Указание: для доказательства завершаемости в качестве фундированного множества L можно взять множество $\mathbf{N}$ натуральных чисел и в качестве u – терм $i + j$ (для всех трех циклов).

3.



$\{i := a;\ j := b\}$
if $(i > j)$ **then** $i \rightleftharpoons j$
while $(i \neq 0)$ **do** $(i, j) := (j\%i, i)$
$d := j$

4.



$\{i := a;\ j := b;\ k := 1\}$
while $i\%2 = 0$ **do** $\left\{ \begin{array}{l} \textbf{if } j\%2 = 0 \textbf{ then } \{j := j/2;\ k := 2k\} \\ i := i/2 \end{array} \right\}$
while $(j\%2 = 0) \vee (i \neq j)$ **do** $\left\{ \begin{array}{l} \textbf{if } j\%2 = 1 \textbf{ then } (i, j) := (j, |i - j|) \\ j := j/2 \end{array} \right\}$
$d := ik$

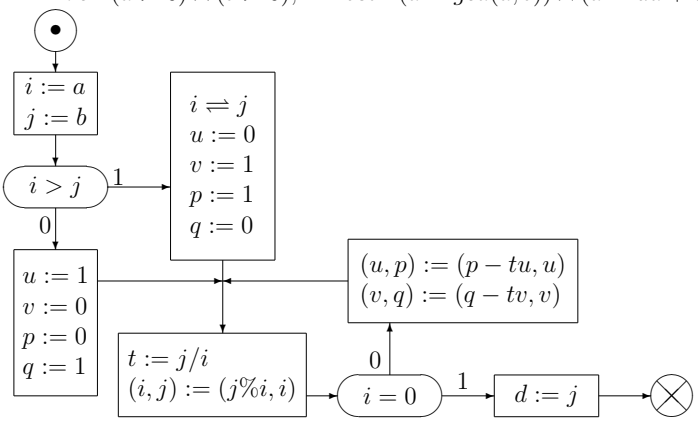
Указание: для доказательства завершаемости рассмотреть следующие инварианты:

$$\varphi_B = \left\{ \begin{array}{l} (a>0)\wedge(b>0) \\ (i>0)\wedge(j>0) \\ gcd(i,j)k = gcd(a,b) \end{array} \right\},\quad \varphi_C = \varphi_B \wedge (i\%2 = 1),$$

и определить u_B и u_C следующим образом: $u_B \stackrel{\text{def}}{=} i$, $u_C \stackrel{\text{def}}{=} i + 2j$.

7.1.7 Представление наибольшего общего делителя линейной формой

$Pre : (a>0)\wedge(b>0), \quad Post : (d = gcd(a,b))\wedge(d = ua + vb).$



7.1.8 Наибольший общий делитель и наименьшее общее кратное

$Pre : (a>0)\wedge(b>0), \quad Post : (d = gcd(a,b))\wedge(m = scm(a,b)),$

где $m = scm(a,b)$ – сокращенная запись формулы

$$m = \min\{x \geq 0 \mid \exists\, a', b' : x = aa' = bb'\}$$

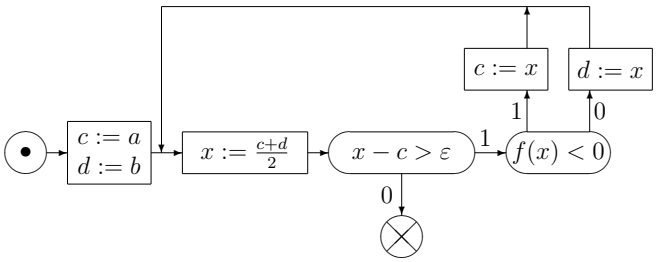
(scm – сокращение от smallest common multiple).

$$\begin{array}{l} \{i := a;\ j := b;\ p := 0;\ q := b\} \\ \textbf{while } i \neq j \textbf{ do } \left\{ \begin{array}{l} \textbf{while } i > j \textbf{ do } (i,p) := (i-j, p+q) \\ \textbf{while } j > i \textbf{ do } (j,q) := (j-i, p+q) \end{array} \right\} \\ \{d := i;\ m := p+q\} \end{array}$$

7.1.9 Приближенное решение уравнения

В излагаемой ниже БС все переменные имеют тип **R**, их значениями являются действительные числа.

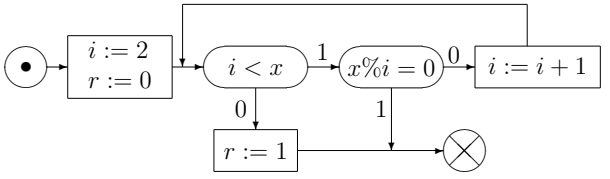
$Pre : (a < b) \wedge (f \in C[a,b]) \wedge (f(a) \le 0) \wedge (f(b) \ge 0) \wedge (\varepsilon > 0)$,
$Post : \exists y \in [a,b] : |x - y| < \varepsilon, f(y) = 0$.



$\{c := a;\ d := b\}$
while $d - c > 2\varepsilon$ **do** $\left\{ \begin{array}{l} x := (c+d)/2 \\ \textbf{if } f(x) < 0 \textbf{ then } c := x \textbf{ else } d := x \end{array} \right\}$
$x := (c+d)/2$

7.1.10 Проверка на простоту

$Pre : x \ge 2$,
$Post : r = \left\{ \begin{array}{l} 1, \text{ если } x \text{ – простое, т.е. } \forall\, i = 2, \ldots, x-1 \ \ x\%i \ne 0, \\ 0, \text{ если } x \text{ не является простым числом.} \end{array} \right.$

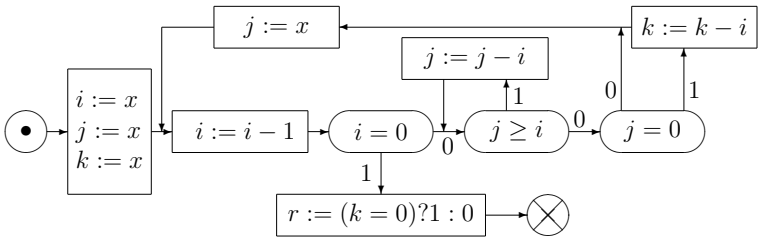


$\{i := 2;\ r := 1\}$
while $(i < x) \wedge (r = 1)$ **do** $\left\{ \begin{array}{ll} \textbf{if } x\%i = 0 & \textbf{then } r := 0 \\ & \textbf{else } i := i + 1 \end{array} \right\}$

7.1.11 Проверка, является ли число совершенным

Целое число называется совершенным, если оно равно сумме своих делителей (например, число $6 = 1 + 2 + 3$ – совершенное).

$$Pre : x > 2, \quad Post : r = \begin{cases} 1, \text{ если } x \sum\limits_{1 \le d < x, d|x} d, \\ 0, \text{ иначе.} \end{cases}$$



$\{i := x - 1;\ j := x;\ k := x\}$
while $i \neq 0$ **do** $\left\{ \begin{array}{l} \textbf{while } j \geq i \textbf{ do } j := j - i \\ \textbf{if } j = 0 \textbf{ then } k := k - i \\ j := x \\ i := i - 1 \end{array} \right\}$
if $k = 0$ **then** $r := 1$ **else** $r := 0$

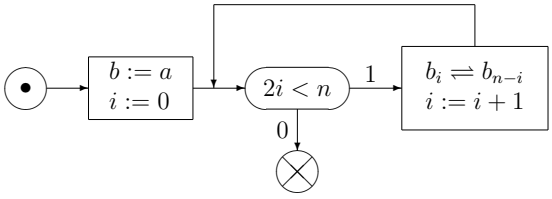
7.2 Верификация программ с массивами

7.2.1 Инвертирование массива

Входной массив – $a_{0..n}$, результат – массив $b_{0..n}$.

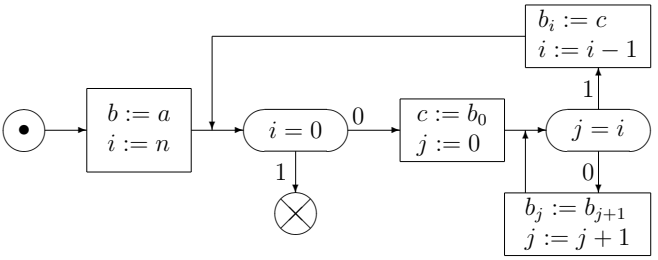
$$Pre : n \geq 0, \quad Post : \forall\, i = 0, \ldots, n \;\; b_i = a_{n-i}.$$

1.



$\{b := a;\ i := 0\}$
while $(2i < n)$ **do** $\{b_i \rightleftharpoons b_{n-i};\ i := i + 1\}$

2.

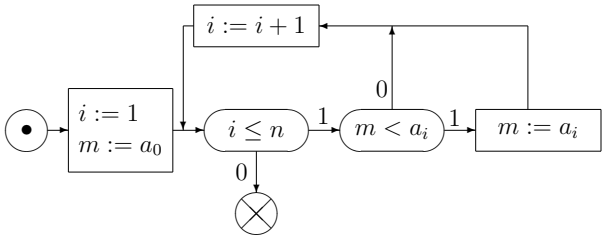


7.2.2 Минимальный элемент массива

Входной массив – $a_{0..n}$.

$$Pre : n \geq 0, \quad Post : m = \min\{a_i \mid i = 0, \ldots, n\}.$$

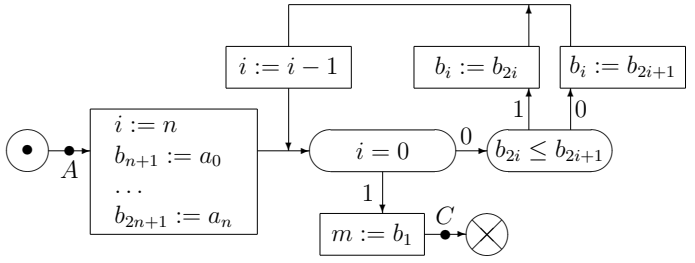
1.



$\{i := 1;\ m := a_0\}$

while $(i \leq n)$ **do** $\left\{ \begin{array}{l} \textbf{if } (m < a_i) \textbf{ then } m := a_i \\ i := i + 1 \end{array} \right\}$

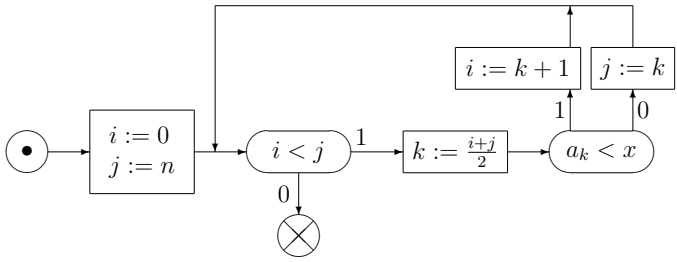
2. (В данной БС используется вспомогательный массив $b_{1..2n+1}$.)



7.2.3 Двоичный поиск

Входной массив – $a_{0..n}$, требуется найти в нем положение элемента x.

$$Pre : (n \geq 0) \wedge (a_0 \leq \ldots \leq a_n) \wedge (\exists\, i : x = a_i),$$
$$Post : (0 \leq i \leq n) \wedge (x = a_i).$$



$\{i := 0;\ j := n\}$

while $(i < j)$ **do** $\left\{ \begin{array}{l} k := (i+j)/2 \\ \textbf{if } (a_k < x) \textbf{ then } i := k+1 \textbf{ else } j := k \end{array} \right\}$

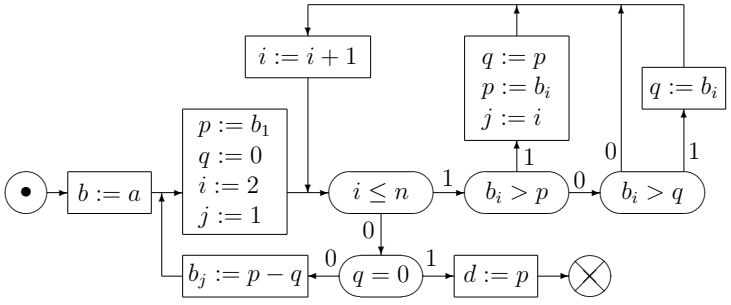
7.2.4 Наибольший общий делитель компонентов массива

Входной массив – $a_{1..n}$.

$$Pre : n \geq 1, \forall\, i\ \ a_i > 0, \quad Post : d = gcd(a_{1..n}),$$

где $d = gcd(a_{1..n})$ – сокращенная запись формулы

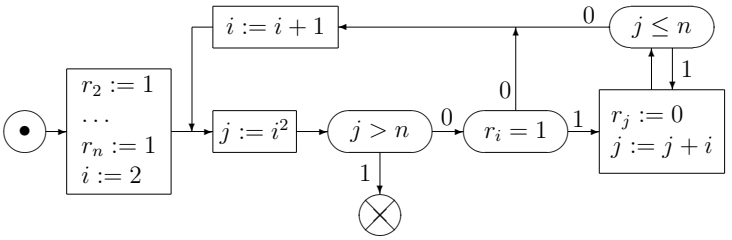
$$d = \max\{x \mid \exists\, a'_1, \ldots, a'_n : a_1 = xa'_1, \ldots, a_n = xa'_n\}.$$



7.2.5 Список простых чисел от 2 до n

Излагаемые ниже программы являются реализацией алгоритма, известного под названием «решето Эратосфена».

$$Pre : n \geq 2,\ Post : \forall\, k = 2, \ldots, n\ \ r_k = \left\{ \begin{array}{l} 1, \text{ если } k \text{ – простое,} \\ 0 \text{ иначе.} \end{array} \right.$$



$\{r_2 := 1;\ \dots;\ r_n := 1;\ i := 2\}$
while $i^2 \le n$ **do**

$$\left\{ \begin{array}{l} \textbf{if } r_i = 1 \textbf{ then } \left\{ \begin{array}{l} j := i^2 \\ \textbf{while } j \le n \textbf{ do } \{r_j := 0;\ j := j + i\} \end{array} \right\} \\ i := i + 1 \end{array} \right\}$$

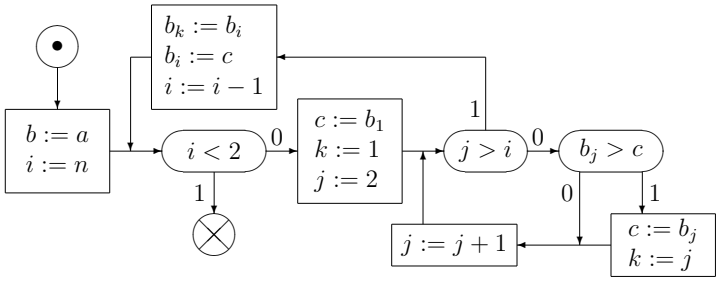
7.2.6 Сортировка массива

Входной массив – $a_{1..n}$. Результат должен быть записан в $b_{1..n}$.

$$Pre : n \ge 1,\ Post : (b = perm(a)) \wedge ord(b)$$

(утверждения вида $b = perm(a)$ и $ord(b)$ определены в пункте 1.2.4).

1.



2. Логика излагаемой ниже программы заключается в том, что массив $a_{1..n}$ рассматривается как дерево, в котором $\forall\, i = 2, \dots, n$ элемент $a_{i/2}$ – родитель a_i.

$b := a$
for $i = n/2$ **step** -1 **until** 2 **do** $\{p := i;\ q := n;\ P\}$
for $i = n$ **step** -1 **until** 2 **do** $\{p := 1;\ q := i;\ P;\ b_1 \rightleftharpoons b_i\}$

где $P = \left\{ \begin{array}{l} x := b_p \\ L : j := 2p \\ \textbf{if } j \le q \textbf{ then} \\ \quad \left\{ \begin{array}{l} \textbf{if } j < q \textbf{ then } \{ \textbf{ if } b_{j+1} > b_j \textbf{ then } j := j+1 \ \} \\ \textbf{if } b_j > x \textbf{ then } \{b_p := b_j;\ p := j;\ \textbf{go to } L\} \end{array} \right. \\ b_p := x \end{array} \right\}$.

Указание: рассмотреть в качестве инварианта одного из циклов формулу, выражающую следующее утверждение:

$$\forall\, i, j : 1 < j \le i \quad b_{j/2} \ge b_j, \text{ и часть } b_{i..n} \text{ отсортирована.}$$

7.2.7 Перестановка массива с заданным условием

Входной массив – $a_{0..n}$, выходной массив – $b_{0..n}$.

$$\begin{array}{l} Pre : n > 0, \\ Post : (b = perm(a)) \wedge (0 \le j < i \le n) \wedge (b_{0..i-1} \le b_{j+1..n}). \end{array}$$

$$\begin{array}{l} \{b := a;\ r := b_{n/2};\ i := 0;\ j := n\} \\ \textbf{while } (i \ne j) \textbf{ do } \left\{ \begin{array}{l} \textbf{while } (b_i < r) \textbf{ do } i := i+1 \\ \textbf{while } (r < b_j) \textbf{ do } j := j-1 \\ \textbf{if } i \le j \textbf{ then } \{b_i \rightleftharpoons b_j;\ i := i+1;\ j := j-1\} \end{array} \right\} \end{array}$$

Указание: рассмотреть в качестве инварианта большого цикла формулу $I = I_i \wedge I_j$, где $\left\{ \begin{array}{l} I_i = (0 \le i) \wedge (b_{0..i-1} \le r), \\ I_j = (j \le n) \wedge (b_{j+1..n} \ge r). \end{array} \right.$

7.2.8 Перестановка массива в заданном порядке

Входной массив – $a_{0..n}$, выходной массив – $b_{0..n}$.

$$\begin{array}{l} Pre : n \ge 0, \quad f - \text{биекция на } \{0, \ldots, n\}, \\ Post : (b = perm(a)) \wedge (\forall\, i = 0, \ldots, n \ \ b_i = a_{f(i)}). \end{array}$$

$$\begin{array}{l} b := a \\ \textbf{for } i := 0 \textbf{ step } 1 \textbf{ until } n \textbf{ do} \\ \qquad \left\{ \begin{array}{l} j := f(i) \\ \textbf{while } j < i \textbf{ do } j := f(j) \\ b_i \rightleftharpoons b_j \end{array} \right. \end{array}$$

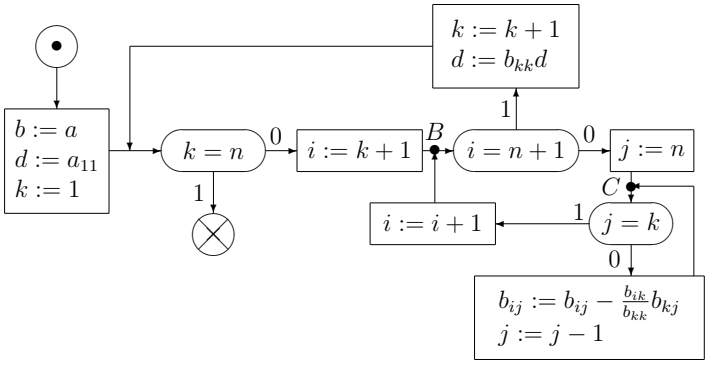
7.2.9 Вычисление определителя матрицы

В излагаемой ниже задаче используется понятие матрицы. Матрицы обозначаются записями вида $a_{m..n,m'..n'}$, где a – имя матрицы, m, n, m', n' – термы типа **I**, обозначающие нижнюю и верхнюю границы по строкам и по столбцам матрицы a соответственно. Матрица $a_{m..n,m'..n'}$ может обозначаться более коротко путем указания лишь её имени, без указания нижней и верхней границ. Для каждой матрицы $a_{m..n,m'..n'}$ и каждых $i \in \{m, \ldots, n\}, j \in \{m', \ldots, n'\}$ определен объект, обозначаемый записью a_{ij} и называемый **компонентой** матрицы a с индексами i и j. Компоненты матрицы a рассматриваются как переменные одинакового типа. Как и для массивов, для матриц определена операция $a := b$ копирования компонентов матрицы b в соответствующие компоненты a.

В представленной ниже БС $a_{1..n,1..n}$ – входная матрица. Все операции (сложение, вычитание, умножение и деление) в излагаемой ниже БС выполняются в поле дробей $\mathbf{R}(a_{11}, \ldots a_{nn})$. Программа вычисляет определитель $det(a)$ входной матрицы путем приведения её к диагональному виду и перемножения элементов на главной диагонали.

$$Pre : n \geq 1, \quad Post : d = det(a) \stackrel{\text{def}}{=} \sum_{\sigma \in S_n} (-1)^\sigma a_{1\sigma(1)} \ldots a_{n\sigma(n)},$$

где S_n – группа всех перестановок на множестве $\{1, \ldots, n\}, \forall \sigma \in S_n \ \ (-1)^\sigma$ – четность перестановки σ.



Указание: при обосновании завершаемости

- доказать, что следующие формулы являются инвариантами в соответствующих точках:

- $\varphi_B : (1 \le k \le n-1) \wedge (2 \le i \le n+1)$,
- $\varphi_C : (1 \le k \le n-1) \wedge (2 \le i \le n) \wedge (k \le j \le n)$;

- определить N, L и $\{u_n \mid n \in N\}$ следующим образом:
  - $N \stackrel{\text{def}}{=} \{B, C\}$,
  - $L \stackrel{\text{def}}{=} \mathbf{N} \times \mathbf{N} \times \mathbf{N}$ (множество троек натуральных чисел с лексикографическим порядком) и
  - $u_B \stackrel{\text{def}}{=} (n-1-k, n+1-i, n+1)$, $u_C \stackrel{\text{def}}{=} (n-1-k, n+1-i, j)$.

7.3 Задачи на составление программ

В этом параграфе требуется составить программы в виде БС или в операторной форме, предназначенные для решения излагаемых ниже задач, и верифицировать эти программы (нужно доказать корректность и завершаемость). Для каждой из этих программ мы указываем требования, которым она должна удовлетворять.

1. Программа получает на вход массив a, результат равен числу различных элементов в a.

2. Программа получает на вход массив a, результат равен 1, если a упорядочен (т.е. верно $ord(a)$), и 0 иначе.

3. Программа получает на вход пару массивов a и b, результат равен 1, если a содержит подмассив, равный b, и 0 иначе.

4. Программа получает на вход пару массивов a и b, результат равен 1, если каждый элемент массива a входит также и в b, и 0 иначе.

5. Программа получает на вход массив $a_{1..n}$, элементами которого являются положительные натуральные числа, и вычисляет представление наибольшего общего делителя $gcd(a)$ элементов массива a в виде линейной формы, т.е. результатом работы программы является массив $b_{1..n}$, удовлетворяющий условию $\sum_{i=1}^{n} a_i b_i = gcd(a)$.

6. Программа получает на вход массив a, результатом является массив, получаемый из a удалением копий тех элементов, которые входят в a более одного раза.

7. Программа получает на вход массив, результатом должен быть отсортированный входной массив (рассмотреть алгоритмы сортировки вставкой, слиянием, а также быструю сортировку Хоара).

8. Программа получает на вход массив a, результатом является пара индексов $i \leq j$, такая, что $a_{i..j}$ – максимальный по размеру подмассив массива a, все элементы которого одинаковы.

9. Программа получает на вход массив a, результатом является массив b, получаемый перестановкой элементов a и такой, что сначала в b идут элементы, имеющие наименьшее число вхождений в a, затем имеющее большее число вхождений, и т.д.

10. Программа получает на вход массив a, результатом является пара индексов $i \leq j$, такая, что $a_{i..j}$ – максимальный по размеру неубывающий подмассив массива a (т.е. $a_i \leq \ldots \leq a_j$).

11. Программа получает на вход массив a, результатом является элемент, имеющий наибольшее число вхождений в a.

12. Программа получает на вход пару упорядоченных массивов a и b, результатом должен быть упорядоченный массив, множество компонентов которого является пересечением (объединением, разностью) множеств компонентов массивов a и b.

13. Программа получает на вход матрицу, элементы которой являются действительными числами. Результатом должен быть определитель этой матрицы, вычисленный путем приведения исходной матрицы к ступенчатому виду.

14. Программа получает на вход матрицу $a_{1..n,1..n}$, $\forall\, i, j = 1, \ldots, n$, коэффициент a_{ij} представляет собой расстояние от вершины с номером i до вершины с номером j некоторого графа, вершины которого занумерованы числами $1, \ldots, n$. Результат – длина минимального пути из вершины с номером 1 в вершину с номером n.

15. Программа получает на вход целое число n, результатом является разложение n на простые множители, т.е. списки простых чисел $p_1, \ldots, p_k$ и натуральных чисел $i_1, \ldots, i_k$, такие что $n = p_1^{i_1} \ldots p_k^{i_k}$.

16. Программа получает на вход ориентированный граф, результат – список сильно связных компонентов этого графа. Программа должна представлять собой реализацию алгоритма Тарьяна [25].

7.4 Верификация распределенных программ

1. Верифицировать все РП из пункта 6.3.

2. Модифицировать РП избрания лидера, изложенную в пункте 6.3.3, в предположении, что среди элементов $d_0, \ldots, d_{n-1}$ м.б. совпадающие. В конце работы каждый из процессов P_i, входящих в эту РП, должен знать номер лидера, и если среди них имеется несколько процессов с максимальным d_i, номер лидера у всех процессов P_i должен быть определён однозначно. Верифицировать эту модифицированную РП.

3. Модифицировать РП сортировки, изложенную в пункте 6.3.4, в предположении, что размер сортируемого массива м.б. произвольным, а число доступных процессоров является фиксированным. Верифицировать эту модифицированную РП.

4. Модифицировать РП перемножения матриц, изложенную в пункте 6.4.5, в предположении, что процессы-работники могут выходить из строя, и через некоторое время могут восстанавливать работоспособность. Для этого ввести понятие предельного времени ожидания для процесса-менеджера, по истечении которого он решает что процесс-работник, которому он послал очередную задачу, вышел из строя, и необходимо послать эту же задачу другому процессу-работнику. Если через некоторое время вышедший из строя работник присылает свое решение, менеджер опять посылает ему очередные задачи. Верифицировать эту модифицированную РП.

5. Модифицировать РП перемножения матриц, изложенную в пункте 6.4.5, в предположении, что процессы взаимодействуют не через общие переменные, а путем пересылки друг другу сообщений с использованием каналов, или путем синхронного взаимодействия путем использования приема и посылки сообщений ($?e$ и $!e$). Верифицировать эту модифицированную РП.

6. Написать РП, являющиеся реализацией параллельных и распределенных алгоритмов из книг [27] и [28], сформулировать их спецификации и верифицировать эти РП.

7.5 Исследовательские проблемы

Наиболее актуальная проблема, связанная с применением метода инвариантов для верификации программ, заключается в автоматизации построения инвариантов (или в автоматизации построения доказательства их отсутствия в том случае, когда верифицируемая программа не удовлетворяет своей спецификации). Некоторые подходы к решению этой проблемы можно найти в [12]–[24]. В общем случае эта проблема алгоритмически неразрешима, поэтому данную проблему целесообразно рассматривать в следующей постановке: разработать средства интерактивного построения требуемых инвариантов, которые бы давали программисту рекомендации по поводу того, какой вид могли бы иметь инварианты, т.е. предлагали бы ему некоторые шаблоны инвариантов, которые он бы уже самостоятельно конкретизировал до формул, обладающих необходимыми свойствами.

Другим направлением исследований в этой области является распространение данного метода на другие классы программ, в том числе на

- программы с потенциально неограниченным количеством взаимодействующих процессов (например, MPI-программы),
- программы, при выполнении которых могут порождаться новые процессы (например, аналогичные тем, которые порождаются функцией `fork` в ОС UNIX),
- сети Петри, программы с использованием нейронных сетей, функциональные и логические программы, объектно-ориентированные программы и т.п.

Отметим, что в параграфе 6.4 была рассмотрена модель MPI-программы, в которой используются наиболее простые функции попарной передачи сообщений между процессами. Требуется построить модель MPI-программ, использующих другие функции передачи сообщений (широковещательной рассылки сообщений, синхронной передачи сообщений и т.п.) и разработать методы верификации таких MPI-программ.

Часть II

Верификация функциональных программ

Глава 8

Введение в функциональное программирование

В этой части рассматриваются математические модели и методы верификации функциональных программ. Основное внимание уделено теории функций, вычисляемых функциональными программами, эти функции называются наименьшими неподвижными точками функциональных программ. Излагаются основные методы верификации функциональных программ: метод вычислительной индукции и метод структурной индукции.

8.1 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 [26]:

> “Функциональное программирование ставит своей целью при-

> дать 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 зарекомендовало 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.

8.2 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.

8.2.1 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

$$\varphi(u, v) = \quad (u = \varepsilon)?v : u_h \varphi(u_t, v). \tag{8.1}$$

ФП (8.1) состоит из одного функционального уравнения, неизвестная величина в котором – функциональная переменная φ. Левая часть уравнения (8.1) состоит из функциональной переменной φ, соответствующей описываемой функции, и списка формальных параметров этой функции (u и v). Правая часть уравнения (8.1) представляет собой выражение, описывающее связь значения описываемой функции на паре аргументов (u, v) с её значениями на других аргументах. Уравнение (8.1) можно рассматривать как рекурсивный алгоритм вычисления функции *append*.

Нетрудно доказать, что функция *append* является единственным решением функционального уравнения (8.1).

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

8.2.2 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.

8.2.3 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

$$(find,\ prefix)$$

($find$ 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

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

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.

Глава 9

Функциональные программы

В этой главе мы определяем понятие функциональной программы. Для этого мы сначала вводим вспомогательные понятия.

9.1 Пополненные домены и функции на них

9.1.1 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: $\forall\, d_1, d_2 \in \tilde{D}_\tau$,

$$d_1 \le d_2 \;\;\Leftrightarrow\;\; d_1 = \omega \;\text{ или }\; d_1 = d_2. \tag{9.1}$$

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

9.1.2 Монотонные функции

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

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

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.

9.1.3 Естественные продолжения

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

$$f:\ D_{\tau_1} \times \ldots \times D_{\tau_n} \to D_\tau,\ \text{где}\ \tau(f) = (\tau_1, \ldots, \tau_n) \to \tau. \tag{9.2}$$

Ниже в этой части будем считать, что каждый ФС $f \in Fun$ связан не с функцией вида (9.2), а с обозначаемой тем же символом f тотальной (т.е. всюду определённой) функцией

$$f : \tilde{D}_{\tau_1} \times \ldots \times \tilde{D}_{\tau_n} \to \tilde{D}_\tau, \tag{9.3}$$

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 (9.2) 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:

$$(1, d_1, d_2) \mapsto d_1,\ \ (0, d_1, d_2) \mapsto d_2,\ \ (\omega, d_1, d_2) \mapsto \omega. \tag{9.4}$$

Нетрудно доказать, что все естественные продолжения и функции (9.4) являются монотонными функциями.

В конце пункта 1.2.3 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 $e(\vec{x})$, определяемая согласно (1.4). Если считать, что ФС, входящим в e, сопоставлены естественные продолжения исходных функций или функции (9.4), то всему терму e м.б. сопоставлена функция

$$e(\vec{x}) : \tilde{D}_{\tau(x_1)} \times \ldots \times \tilde{D}_{\tau(x_n)} \to \tilde{D}_{\tau(e)}, \tag{9.5}$$

определяемая так же, как аналогичная функция (1.4). Нетрудно доказать, что функция (9.5) монотонна.

9.1.4 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

$$f : \tilde{D}_{\tau_1} \times \ldots \times \tilde{D}_{\tau_n} \to \tilde{D}_{\tau}. \tag{9.6}$$

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

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

где отношение неравенства на $\tilde{D}_\tau$ понимается в смысле определения (9.1).

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 $\forall\, f \in D_{ft} \quad \textcircled{\omega} \leq f$.

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

$$f_0(\vec{d}) \leq f_1(\vec{d}) \leq f_2(\vec{d}) \leq \ldots. \tag{9.7}$$

Согласно определению порядка на пополненных доменах, (9.7) эквивалентно условию: либо $\forall\, i \geq 0 \quad f_i(\vec{d}) = \omega$, либо $\exists\, i \geq 0$, $\exists\, d \in D_\tau$:

$$\forall j < i \quad f_j(\vec{d}) = \omega, \quad \forall j \geq i \quad f_j(\vec{d}) = d. \tag{9.8}$$

Обозначим записью $\sup_{i \geq 0} f_i$ функцию вида (9.6), определяемую следующим образом: $\forall\, \vec{d} \in \tilde{D}_{\tau_1} \times \ldots \times \tilde{D}_{\tau_n}$

$$\sup_{i \geq 0} f_i(\vec{d}) \stackrel{\text{def}}{=} \begin{cases} \omega & \text{если } \forall\, i \geq 0 \quad f_i(\vec{d}) = \omega, \\ d & \text{если } \exists\, i \geq 0 : \text{ верно (9.8).} \end{cases} \tag{9.9}$$

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 $f \leq f'$.

9.1.5 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 9.1.4 вытекает, что D_{ft} – полное ЧУМ. Наименьшим элементом D_{ft} является функция $\textcircled{\omega}$.

9.2 Функциональные программы

9.2.1 Понятие функциональной программы

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

$$\left\{ \begin{array}{l} \varphi_1(\vec{x}_1) = e_1 \\ \ldots \\ \varphi_n(\vec{x}_n) = e_n \end{array} \right. , \tag{9.10}$$

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

$$\forall\, i = 1, \ldots, n \quad f_i = e_i^{(f_1/\varphi_1, \ldots, f_n/\varphi_n)}(\vec{x}_i). \tag{9.11}$$

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

9.2.2 Функционал, соответствующий функциональной программе

Пусть Σ – ФП вида (9.10).

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:

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

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

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

  (отметим, что правая часть данного равенства имеет смысл, поскольку из свойства монотонности F следует, что последовательность $\{F(p_i) \mid i \ge 0\}$ 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, ... обозначают функционалы на P, определяемые следующим образом: $\forall p \in P \; F^0(p) \stackrel{\text{def}}{=} p$ и $\forall i \geq 0 \quad F^{i+1}(p) \stackrel{\text{def}}{=} F(F^i(p))$.

9.2.4 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 9.2.1 и 9.2.2 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 5

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

$$\forall\, i \geq 0 \quad p_i \leq p'. \tag{9.12}$$

  (9.12) 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 (9.12) следует искомое неравенство $p = \sup_{i\geq 0} p_i \leq p'$ ■

9.2.5 Непрерывность функционалов, соответствующих функциональным программам

Теорема 6

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

$$\Sigma: \qquad \varphi(\vec{x}) = e. \tag{9.13}$$

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

$$\begin{array}{l} F_{e'}(f)(\vec{d}) = (e')^{(f/\varphi)}(\vec{d}) = g(e_1, \ldots, e_m)^{(f/\varphi)}(\vec{d}) = \\ = g(e_1^{(f/\varphi)}, \ldots, e_m^{(f/\varphi)})(\vec{d}) = g(e_1^{(f/\varphi)}(\vec{d}), \ldots, e_m^{(f/\varphi)}(\vec{d})) = \\ = g(F_{e_1}(f)(\vec{d}), \ldots, F_{e_m}(f)(\vec{d})). \end{array} \qquad (9.14)$$

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 (9.14) и монотонность функции g, получаем:

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

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

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

$$F_{e'}(\sup_{i\geq 0} f_i) = \sup_{i\geq 0} F_{e'}(f_i). \qquad (9.15)$$

$\forall\, i \geq 0$ 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 (9.15) докажем, что верно обратное неравенство:

$$F_{e'}(\sup_{i\geq 0} f_i) \leq \sup_{i\geq 0} F_{e'}(f_i), \tag{9.16}$$

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

$$F_{e'}(\sup_{i\geq 0} f_i)(\vec{d}) \leq \sup_{i\geq 0} F_{e'}(f_i)(\vec{d}). \tag{9.17}$$

Согласно (9.14), левую часть (9.17) можно переписать в виде

$$g(F_{e_1}(\sup_{i\geq 0} f_i)(\vec{d}), \ldots, F_{e_m}(\sup_{i\geq 0} f_i)(\vec{d})). \tag{9.18}$$

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 (9.18) можно переписать в виде

$$g\,(\sup_{i\geq 0} F_{e_1}(f_i)(\vec{d}), \ldots, \sup_{i\geq 0} F_{e_m}(f_i)(\vec{d})). \tag{9.19}$$

Из (9.9) 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 (9.19) можно переписать в виде

$$g\,(F_{e_1}(f_j)(\vec{d}), \ldots, F_{e_m}(f_j)(\vec{d})). \tag{9.20}$$

Согласно (9.14), значение выражения (9.20) равно $F_{e'}(f_j)(\vec{d})$.

Поэтому доказываемое неравенство (9.17) 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 5 и 6, 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 (9.10), 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 σ^s, σ^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.

9.2.6 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 5).

Рассмотрим пример нахождения данным методом ННТ ФП:

$$\Sigma: \quad \left\{ \begin{array}{lll} \varphi(x) & = & (x > 100) \quad ? \, x - 10 \\ & & \qquad\qquad\;\; : \varphi\varphi(x + 11) \end{array} \right. \tag{9.21}$$

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

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

а для $i \geq 11$

$$\begin{array}{lll} e_i & = & (x > 100) \quad ? \, x - 10 \\ & & \qquad\quad : (x > 90 - 11 \cdot (i - 11)) \quad ? \, 91 : \omega \end{array} \tag{9.22}$$

При фиксированном x и достаточно больших i выражение

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

в терме (9.22) будет истинным, поэтому функция σ соответствует терму

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

9.2.7 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
$$\varphi(x,y) \ = \ (x = y) \ \begin{array}{l} ?\ y + 1 \\ :\ \varphi(x, \varphi(x-1, y+1)) \end{array}$$
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 $= (x = 0)?\,1 : \omega$.

9.2.8 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.

9.3 Алгоритмическая полнота функциональных программ

Свойство **алгоритмической полноты** множества ФП заключается в том, что каждая частичная функция на строках, вычислимая в интуитивном смысле, может быть описана некоторой ФП. Мы предполагаем, что для каждой частичной функции f на строках, вычислимой в интуитивном смысле, существует машина Тьюринга M, такая, что f совпадает с функцией, которую вычисляет M.

Теорема 7
Для каждой машины Тьюринга M существует ФП, описывающая ту функцию, которую вычисляет M.

Доказательство.
Пусть M – машина Тьюринга, компонентами которой являются:

- множество состояний $Q = \{q_0, q_1, \ldots, q_n\}$, в котором выделены начальное состояние q_0 и заключительное состояние q_n;
- алфавит символов $A = \{a_1, \ldots, a_m\}$, которые могут быть написаны в ячейках ленты, причём A содержит пробельный символ $\square$;
- отображение переходов $\delta : Q \times A \to Q \times A \times \{\texttt{left}, \texttt{right}\}$.

Сопоставим каждому состоянию $q_i \in Q$ функциональную переменную φ_i, где $\tau(\varphi_i) = \ (\mathbf{S}, \mathbf{S}) \to \mathbf{S}$.

ФП Σ_M, которая описывает функцию, вычисляемую машиной M, состоит из следующих уравнений:

- уравнение, соответствующее той функции, которую вычисляет машина M: $\varphi(x) = \varphi_0(\varepsilon, x)$,
- если q_i – незаключительное состояние и $\begin{cases} \delta(q_i, a_1) = (q_j, a_k, \texttt{right}) \\ \delta(q_i, a_2) = \ldots \\ \ldots \end{cases}$

  то ФП Σ_M содержит уравнение

$$\begin{aligned} \varphi_i(x, y) = \ & (y = \varepsilon) \ ? \ \varphi_i(x, \square) \\ & : (y_h = a_1) \ ? \ \varphi_j(a_k x, y_t) \\ & \qquad : (y_h = a_2) \ ? \ldots \\ & \qquad \ldots \end{aligned}$$

  где

– значения, принимаемые переменной x, соответствуют записям на ленте слева от головки, читаемым справа налево, и

– значения, принимаемые переменной y, соответствуют записям на ленте справа от головки (включая символ под головкой);

- если q_i – незаключительное состояние и $\begin{cases} \delta(q_i, a_1) = (q_j, a_k, \texttt{left}) \\ \delta(q_i, a_2) = \dots \\ \dots \end{cases}$,

  то ФП Σ_M содержит уравнение

$$\begin{array}{l} \varphi_i(x, y) = \\ \quad = (y = \varepsilon) \;\; ? \; \varphi_i(x, \square) \\ \quad : (x = \varepsilon) \;\; ? \; \varphi_i(\square, y) \\ \qquad\qquad : (y_h = a_1) \;\; ? \; \varphi_j(x_t, x_h a_k y_t) \\ \qquad\qquad\qquad\quad : (y_h = a_2) \; ? \dots \\ \qquad\qquad\qquad\qquad \dots \end{array}$$

- уравнение, соответствующее заключительному состоянию q_n:

$$\begin{array}{l} \varphi_n(x, y) = (y = \varepsilon) \; ? \; \varepsilon \\ \quad : (last(y) = \square) \; ? \; \varphi_n(x, rem(y)) : y \end{array}$$

  где

  – функция $last$ возвращает последний символ своего аргумента, она описывается ФП

$$\varphi(x) = (x_t = \varepsilon) \; ? \; x_h : \varphi(x_t),$$

  – функция rem возвращает строку, получаемую удалением последнего символа из своего аргумента, она описывается ФП

$$\varphi(x) = (x_t = \varepsilon) \; ? \; \varepsilon : x_h \varphi(x_t),$$

  – функция, соответствующая функциональной переменной φ_n, возвращает строку, получаемую из значения второго аргумента удалением пробелов в его конце. ■

Глава 10

Вычисление значений наименьших неподвижных точек

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 $FVar(e) \subseteq \{\varphi\}$.

10.1 Постановка задачи

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

$$\Sigma : \quad \varphi(\vec{x}) = e \tag{10.1}$$

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.

10.2 Метод решения

Один из возможных методов решения данной задачи заключается в том, что по заданным ФП Σ вида (10.1), и списку $\vec{d} \in D_{\tau(\vec{x})}$ строится последовательность термов

$$C^0_{\Sigma,\vec{d}} \quad C^1_{\Sigma,\vec{d}} \quad C^2_{\Sigma,\vec{d}} \quad \ldots \tag{10.2}$$

называемая **вычислительной последовательностью**. Каждый её член является в некотором смысле аппроксимацией искомого значения $\sigma(\vec{d})$. Если вычислительная последовательность (10.2) конечна, то её последний элемент должен быть константой, значение которой равно $\sigma(\vec{d})$.

Последовательность (10.2) 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

   $$\varphi(e_1, \ldots, e_n) \tag{10.3}$$

   на термы вида

   $$e^{(e_1/x_1,\ldots,e_n/x_n)}, \tag{10.4}$$

   где $(x_1, \ldots, x_n)$ – это список $\vec{x}$ в (10.1), и

   (b) упрощением получившегося терма.

3. Если терм $C^i_{\Sigma,\vec{d}}$ не содержит φ, то он является последним членом последовательности (10.2).

Ниже мы будем называть

- те подтермы вида (10.3) терма $C^i_{\Sigma,\vec{d}}$, которые выбираются для замены согласно пункту (2a), **раскрываемыми** подтермами, и

- замену (10.3) на (10.4) – **раскрытием** подтерма (10.3).

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.

10.3 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 (10.3) (т.е. не содержащиеся ни в каком подтерме вида (10.3)).

2. **LO** (Left Outermost) – раскрывается самый левый из самых внешних подтермов вида (10.3).

3. **PI** (Parallel Innermost) – раскрываются все самые внутренние подтермы вида (10.3) (т.е. не содержащие подтермов вида (10.3)).

4. **LI** (Left Innermost) – раскрывается самый левый из самых внутренних подтермов вида (10.3).

Мы будем считать, что символ C в (10.2) 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 $\mathbf{PO}^i_{\Sigma,\vec{d}}$).

10.4 Упрощение терма

Упрощение терма, о котором говорится в пункте (2b) описания построения вычислительной последовательности (10.2), представляет собой последовательность **упрощающих преобразований**.

Для определения понятия упрощающего преобразования введём следующие вспомогательные понятия.

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

$$\forall f \in D_{\tau(\varphi)} \quad r^{(f/\varphi)}(\vec{x}) = s^{(f/\varphi)}(\vec{x}), \tag{10.5}$$

   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

$$g(v_1, \ldots, v_n) \quad (g \in Fun) \tag{10.6}$$

то запись $\hat{v}$ обозначает терм

$$g(h_1, \ldots, h_n), \tag{10.7}$$

где $\forall\, i = 1, \ldots, n$,

$$h_i \stackrel{\text{def}}{=} \begin{cases} v_i, & \text{если } v_i \text{ – константа,} \\ x_i & \text{(переменная того же типа,} \\ & \text{что и } v_i\text{) иначе,} \end{cases} \tag{10.8}$$

причём все переменные, входящие в (10.7), различны.

Мы будем говорить, что терм s получен из терма r **упрощающим преобразованием**, если s является результатом замены в r подтерма v вида (10.6) 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 8
Для любого терма r существует терм s, являющийся его упрощением.

Доказательство.
Определим последовательность термов

$$r_1 \;\; r_2 \;\; \ldots \tag{10.9}$$

следующим образом: $r_1 \stackrel{\text{def}}{=} r$, и для каждого терма r_i в (10.9)

- если r_i неупрощаемый, то он – последний элемент в (10.9),
- иначе определяем r_{i+1} как результат какого-либо упрощающего преобразования терма r_i.

Последовательность (10.9) 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 (10.9). ■

Последовательность (10.9) 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 10, последний элемент последовательности (10.9) однозначно определяется её первым элементом.

Мы будем использовать следующие обозначения.

- Для любых термов r и s мы будем обозначать записью

$$r \to s \tag{10.10}$$

  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 9
Если

$$r \to s_1 \quad \text{и} \quad r \to s_2, \tag{10.11}$$

то

$$\exists\, s: \quad s_1 \to s \quad \text{и} \quad s_2 \to s \tag{10.12}$$

(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 (10.6), и h_j – переменная, определённая согласно (10.8),

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

   $$g(v_1, \ldots, v_n) \quad \text{и} \quad g(h_1, \ldots, h_n) \tag{10.13}$$

   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 (10.13).

   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 (10.13) 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 10
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

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

такие, что имеют место соотношения

$$\begin{array}{ccccccc} s_{11} & \to & s_{12} & \to & \ldots & \to & s_{1n} \\ \downarrow & & & & & & \\ s_{21} & & & & & & \\ \downarrow & & & & & & \\ \ldots & & & & & & \\ \downarrow & & & & & & \\ s_{m1} & & & & & & \end{array} \tag{10.14}$$

(мы можем рисовать стрелки, изображающие отношение (10.10), не только горизонтально, но и вертикально).

Диаграмму (10.14) можно достроить до диаграммы

$$\begin{array}{ccccccc} s_{11} & \to & s_{12} & \to & \ldots & \to & s_{1n} \\ \downarrow & & \downarrow & & & & \downarrow \\ s_{21} & \to & s_{22} & \to & \ldots & \to & s_{2n} \\ \downarrow & & \downarrow & & & & \downarrow \\ \ldots & & \ldots & & & & \ldots \\ \downarrow & & \downarrow & & & & \downarrow \\ s_{m1} & \to & s_{m2} & \to & \ldots & \to & s_{mn} \end{array} \tag{10.15}$$

где термы $s_{22}, \ldots, s_{mn}$ 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 (10.15) 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.

10.5 Функция C_Σ

Пусть заданы вычислительное правило C и ФП Σ вида

$$\Sigma: \quad \varphi(\vec{x}) = e. \tag{10.16}$$

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 (10.2). 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 11
Для любого $\vec{d} \in D_{\tau(\vec{x})}$ верно неравенство $C_\Sigma(\vec{d}) \leq \sigma(\vec{d})$.

Доказательство.
Если последовательность (10.2) бесконечна, то $C_\Sigma(\vec{d}) = \omega \leq \sigma(\vec{d})$.

Пусть последовательность (10.2) конечна.

Все термы в последовательности

$$(C^0_{\Sigma,\vec{d}})^{(\sigma/\varphi)} \quad (C^1_{\Sigma,\vec{d}})^{(\sigma/\varphi)} \quad \ldots \tag{10.17}$$

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

Последовательность (10.2) 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 (10.17) является следствием следующих двух утверждений.

1. Если термы s и r не содержат переменных и s получен из r заменой подтерма
$$\varphi(e_1, \ldots, e_n) \tag{10.18}$$
на терм
$$e^{(e_1/x_1,\ldots,e_n/x_n)}, \tag{10.19}$$
то значения термов
$$s^{(\sigma/\varphi)} \quad \text{и} \quad r^{(\sigma/\varphi)} \tag{10.20}$$
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 (10.20) 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 (10.18) на терм (10.19), то $s^{(\sigma/\varphi)}$ получен из $r^{(\sigma/\varphi)}$ заменой подтерма

$$\sigma(e_1^{(\sigma/\varphi)}, \ldots, e_n^{(\sigma/\varphi)}) \tag{10.21}$$

на терм

$$(e^{(\sigma/\varphi)})^{(e_1^{(\sigma/\varphi)}/x_1,\ldots,e_n^{(\sigma/\varphi)}/x_n)}. \tag{10.22}$$

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

$$\sigma(d_1, \ldots, d_n) = (e^{(\sigma/\varphi)})^{(d_1/x_1, \ldots, d_n/x_n)}. \tag{10.23}$$

Полагая в (10.23)

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

получаем, что значения термов (10.21) и (10.22) 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 (10.20) совпадают, т.е. утверждение 1 обосновано. Из утверждений 1 и 2 следует, что значения всех термов в (10.17) совпадают.

Поскольку

- значение первого терма в (10.17) равно $\sigma(\vec{d})$ и
- значение последнего терма в (10.17) равно $C_\Sigma(\vec{d})$,

то мы заключаем, что если вычислительная последовательность (10.2) конечна, то верно равенство $C_\Sigma(\vec{d}) = \sigma(\vec{d})$. ■

10.6 Вспомогательные понятия

В этом параграфе мы определяем понятия и доказываем теоремы, которые будут использоваться в параграфе 10.7 при доказательстве теоремы 16 о свойстве безопасных вычислительных правил.

10.6.1 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 12
Для любого $i \geq 0$ и любой функции $f \in D_{\tau(\varphi)}$

$$(e^{(i)})^{(f/\varphi)}(\vec{x}) = F_\Sigma^i(f). \tag{10.24}$$

Доказательство.
Докажем (10.24) индукцией по i.
При $i = 0$ обе части (10.24) равны функции f.
Пусть (10.24) 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

$$f(e_1, \ldots, e_n) \tag{10.25}$$

на терм

$$(e^{(f/\varphi)})^{(e_1/x_1,\ldots,e_n/x_n)}, \tag{10.26}$$

причём данные замены выполняются по принципу «от меньших подтермов к большим». Из определения функционала F_Σ следует, что терм (10.26) эквивалентен терму

$$F_\Sigma(f)(e_1, \ldots, e_n), \tag{10.27}$$

поэтому терм $(e^{(i+1)})^{(f/\varphi)}$ эквивалентен терму, получаемому из терма $(e^{(i)})^{(f/\varphi)}$ заменой каждого подтерма вида (10.25) на терм (10.27), 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

$$(e^{(i+1)})^{(f/\varphi)}(\vec{x}) = (e^{(i)})^{(F_\Sigma(f)/\varphi)}(\vec{x}). \tag{10.28}$$

Поскольку мы предполагаем, что (10.24) верно для каждой функции f, то, в частности, правая часть в (10.28) равна

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

10.6.2 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 $\{in(r, i) \mid i \geq 0\}$.

10.6.3 Σ-переходы

Пусть задана ФП $\Sigma: \quad \varphi(\vec{x}) = e, \quad$ 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 $r \xrightarrow{\Sigma} s$. 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 $r \overset{\Sigma*}{\to} s$ означает, что существует последовательность термов $r_0, \ldots, r_n$, обладающая свойствами

$$r = r_0 \overset{\Sigma}{\to} \ldots \overset{\Sigma}{\to} r_n = s. \tag{10.29}$$

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

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

(такой порядок называется **лексикографическим**).

Теорема 13
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

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

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

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

$$C^i_{\Sigma,\vec{d}} \overset{\Sigma*}{\to} C^{i+1}_{\Sigma,\vec{d}}. \tag{10.30}$$

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

Теорема 14
Если вычислительная последовательность $\{C^i_{\Sigma,\vec{d}} \mid i \ge 0\}$ бесконечна, то $\forall\, M \ge 0 \quad \exists\, N \ge 0$:

$$\forall\, j = 0, \ldots, M \qquad in(C^N_{\Sigma,\vec{d}}, j) = in(C^{N+1}_{\Sigma,\vec{d}}, j) = \ldots \tag{10.31}$$

Доказательство.
Индукция по M.

1. Если $M = 0$, то $N = 1$.

2. Пусть для некоторого $M \ge 0$ существует номер N, для которого верно (10.31). Тогда из теоремы 13 и из определения лексикографического порядка следует, что

$$in(C^N_{\Sigma,\vec{d}}, M+1) \ge in(C^{N+1}_{\Sigma,\vec{d}}, M+1) \ge \ldots \tag{10.32}$$

Цепочка (10.32) не может бесконечно убывать, т.е. $\exists\, N' \ge N$:

$$in(C^{N'}_{\Sigma,\vec{d}}, M+1) = in(C^{N'+1}_{\Sigma,\vec{d}}, M+1) = \ldots \tag{10.33}$$

Из (10.31) и из $N' \ge N$ следует, что для каждого $j = 0, \ldots, M$ цепочка (10.33), 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 15

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 (10.31).

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 (10.31). ■

10.7 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 $\forall\, i \geq 0$

$$(C^{i,\omega}_{\Sigma,\vec{d}})^{(\sigma/\varphi)} = \omega, \tag{10.34}$$

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 16

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

$$C_\Sigma(\vec{d}) = \sigma(\vec{d}). \tag{10.35}$$

Доказательство.

Как было установлено в теореме 11, равенство (10.35) может нарушаться только в том случае, когда

$$\omega < \sigma(\vec{d}) \tag{10.36}$$

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 $\exists\, n$:

$$\sigma(\vec{d}) = F^n_\Sigma(\textcircled{\omega})(\vec{d}) = (e^{(n)})^{(\textcircled{\omega}/\varphi)}(\vec{d}), \tag{10.37}$$

где $e^{(n)}$ – соответствующий терм из полного раскрытия для Σ (второе равенство верно согласно теореме 12).

Из определения полного раскрытия следует, что существуют ИТ r_0, ..., r_k, такие, что

$$\varphi(\vec{d}) = e^{(0)}(\vec{d}) = r_0 \stackrel{\Sigma}{\to} \ldots \stackrel{\Sigma}{\to} r_k = e^{(n)}(\vec{d}), \tag{10.38}$$

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

Из (10.36), (10.37) и определения r_k следует, что

$$\omega < r_k^{((\!\!\textcircled{\omega}\!\!)/\varphi)}. \tag{10.39}$$

Обозначим символом M максимальную глубину вхождения φ в r_k.

По теореме 15, $\exists N$: 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 (10.30), то существует последовательность ИТ s_0, ..., s_l, такая, что

$$\varphi(\vec{d}) = s_0 \stackrel{\Sigma}{\to} \ldots \stackrel{\Sigma}{\to} s_l = C^N_{\Sigma,\vec{d}}. \tag{10.40}$$

Если в (10.40) есть пара соседних Σ-переходов, первый из которых – упрощение, а следующий за ним – раскрытие, то мы преобразуем (10.40) в соответствии со следующим правилом. Пусть i – наименьший номер, такой, что (10.40) содержит пару

$$s_i \stackrel{\Sigma}{\to} s_{i+1} = s_i^{(u'/u)} \stackrel{\Sigma}{\to} s_{i+2} = s_{i+1}^{(v'/v)}, \tag{10.41}$$

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 (10.40) пару (10.41) 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 (10.40) пару (10.41) 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 (10.40), 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 (10.40), в которой сначала идут раскрытия, а затем упрощения. Нетрудно видеть, что данный процесс завершается после конечного числа шагов.

Пусть s_m – тот член последовательности (10.40), на котором заканчиваются раскрытия и начинаются упрощения, т.е. все Σ-переходы в подпоследовательности

$$\varphi(\vec{d}) = s_0 \overset{\Sigma}{\to} \ldots \overset{\Sigma}{\to} s_m \tag{10.42}$$

являются раскрытиями и все Σ-переходы в подпоследовательности

$$s_m \overset{\Sigma}{\to} \ldots \overset{\Sigma}{\to} s_l = C^N_{\Sigma,\vec{d}} \tag{10.43}$$

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

$$\textcircled{\omega}_M(s_m)^{(\sigma/\varphi)} = \omega \tag{10.44}$$

(это следует из того, что терм $(C^{N,\omega}_{\Sigma,\vec{d}})^{(\sigma/\varphi)}$, значение которого равно ω, можно получить из левой части (10.44) 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 (10.42) Σ-переход

$$s_m \overset{\Sigma}{\to} s_m^{(u'/u)}, \quad \text{где} \;\; u' = e^{(u_1/x_1,\ldots)}. \tag{10.45}$$

Из (10.44) 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 (10.45) к (10.42), той же записью, что и исходную последовательность (10.42). Будем выполнять описанную выше операцию добавления Σ-переходов к (10.42) 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 (10.44), из которого следует равенство

$$s_m^{(\textcircled{\omega}/\varphi)} = \omega. \tag{10.46}$$

Преобразуем последовательности (10.38) и (10.42) в такие последовательности Σ-переходов, в которых каждая пара соседних Σ-переходов удовлетворяет следующему условию: пусть данная пара имеет вид

$$a \overset{\Sigma}{\to} b = a^{(u'/u)} \overset{\Sigma}{\to} c = b^{(v'/v)}, \tag{10.47}$$

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 (10.47). Тогда возможен один из двух случаев.

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

   В этом случае заменим (10.47) 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 (10.47) 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 (10.38) и (10.42) преобразуются в такие последовательности Σ-переходов, которые будут удовлетворять изложенному выше условию. Будем обозначать результаты данных преобразований теми же записями, что и исходные последовательности (10.38) и (10.42). Таким образом, последовательности (10.38) и (10.42) 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;

- (10.38) имеет вид $\varphi(\vec{d}) = r_0 \stackrel{\Sigma}{\to} \ldots \stackrel{\Sigma}{\to} r_k$, и все вхождения φ в r_k имеют глубину $\leq M$;
- (10.42) имеет вид $\varphi(\vec{d}) = s_0 \stackrel{\Sigma}{\to} \ldots \stackrel{\Sigma}{\to} s_m$, 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 $r_i \in$ (10.38), $s_j \in$ (10.42), обладающих следующими свойствами:

- r_i можно представить в виде конкатенации

$$a_1\ u_1\ \ldots\ a_n\ u_n\ a, \tag{10.48}$$

  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

$$a_1\ v_1\ \ldots\ a_n\ v_n\ a, \tag{10.49}$$

  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 (10.48),
  - если $j < m$, то раскрываемый подтерм терма s_j содержится в подстроке a.

Отметим, что

$$(s_0, r_0) \in R \tag{10.50}$$

(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 (10.50) 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 (10.42) (кроме первого) раскрываемый подтерм расположен правее раскрываемого подтерма в предыдущем Σ-переходе, то

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

3. $\alpha > \beta$. Пусть r_i и s_j можно представить в виде конкатенаций (10.48) и (10.49) 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 (10.48) и (10.49) 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 (10.48) и (10.49) 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 (10.39) и (10.46). ■

10.8 Свойства правил PO, LO, PI, LI

10.8.1 Безопасность правила PO

Теорема 17

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 (10.34) выполнено.

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

$$g(\ldots\varphi(\vec{e}_1)\ldots\varphi(\vec{e}_k)\ldots), \tag{10.51}$$

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

Условие (10.34) для терма (10.51) имеет вид

$$g(\ldots\omega\ldots\omega\ldots) = \omega, \tag{10.52}$$

где терм в левой части является результатом замены подтермов $\varphi(\vec{e}_i)$ терма (10.51) на ω.

Докажем соотношение (10.52). 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

$$g(\ldots b_1\ldots b_k\ldots) = d, \tag{10.53}$$

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 (10.51).

Лемма
Для любого неупрощаемого терма v и любой константы $d \neq \omega$ верна импликация

$$v \equiv d \quad \Rightarrow \quad v = d \tag{10.54}$$

(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

$$\hat{v} = d, \tag{10.55}$$

которое противоречит предположению о неупрощаемости v.
(10.55) следует из доказываемого ниже соотношения

$$\begin{array}{c} g(d_1, \ldots, d_n) \equiv d, \quad \text{где} \ \ \forall\, i = 1, \ldots, n \\ d_i \stackrel{\text{def}}{=} \left\{ \begin{array}{ll} v_i, & \text{если } v_i \text{ – константа} \\ \omega & \text{иначе.} \end{array} \right. \end{array} \tag{10.56}$$

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

$$d \equiv v^{\omega} = g(v_1^{\omega}, \ldots, v_n^{\omega}). \tag{10.57}$$

Докажем, что

$$\forall\, i = 1, \ldots, n \quad v_i^{\omega} \equiv d_i. \tag{10.58}$$

- 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

$$v_i \equiv d' \neq \omega. \tag{10.59}$$

  Поскольку терм v неупрощаемый, то терм v_i тоже неупрощаемый. По индуктивному предположению, верна импликация (10.54), с заменой v на v_i, т.е. из (10.59) следует равенство $v_i = d'$, которое противоречит предположению о том, что v_i – не константа.

Соотношение (10.56) следует из (10.57) и (10.58).

Таким образом, соотношение $v \equiv d$ может быть верно только в том случае, когда v является константой d. ■

10.8.2 Безопасность правила LO

Теорема 18

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 (10.34) 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

$$g(v_1, \ldots, v_m) \quad (g \in Fun), \tag{10.60}$$

  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 $\forall\, j = 1, \ldots, n$

$$r_j^\omega = \omega. \tag{10.61}$$

Для $j = 1$ соотношение (10.61), очевидно, верно.

Пусть для некоторого $j \in \{1, \ldots, n-1\}$

- верно (10.61),
- r_{j+1} имеет вид (10.60) и
- $\exists\, k \in \{1, \ldots, m\} : v_k = r_j$.

тогда

$$r_{j+1}^\omega = g(v_1, \ldots, v_{k-1}, r_j^\omega, v_{k+1}, \ldots, v_m). \tag{10.62}$$

Если $g \neq \mathit{if\_then\_else}$, то соотношение

$$r_{j+1}^\omega = \omega$$

следует из (10.61), (10.62) 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 $r_{j+1} = if\_then\_else(r_j, v_2, v_3)$, откуда следует, что

$$r^{\omega}_{j+1} = if\_then\_else(r^{\omega}_j, v_2, v_3). \tag{10.63}$$

Учитывая индуктивное предположение $r^{\omega}_j = \omega$ и определение функции *if_ then_ else*, заключаем, что (10.63) $= \omega$.

Таким образом, $\forall\, j = 1, \ldots, n$ верно (10.61).

В частности, $r^{\omega}_n = \omega$, поэтому $(C^{i,\omega}_{\Sigma,\vec{d}})^{(\sigma/\varphi)} = (r^{\omega}_n)^{(\sigma/\varphi)} = \omega$. ■

10.8.3 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.

10.8.4 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

$$\begin{array}{l} \mathbf{LI}^0_{\Sigma} = \varphi(1,0) \\ \mathbf{LI}^1_{\Sigma} = \varphi(0, \varphi(1,0)) \\ \mathbf{LI}^2_{\Sigma} = \varphi(0, \varphi(0, \varphi(1,0))) \\ \ldots \end{array}$$

Глава 11

Верификация функциональных программ

11.1 Задача верификации функциональных программ

Задача **верификации** ФП заключается в доказательстве того, что ННТ анализируемых ФП обладают заданными свойствами. Наиболее часто эти свойства представляют собой утверждения вида

$$\forall\, \vec{x}\, e, \tag{11.1}$$

где $\tau(e) = \mathbf{B}$ и $\vec{x} = Var(e)$. Утверждение (11.1) считается верным, если для каждого $\vec{d} \in D_{\tau(\vec{x})}$ значение терма $e(\vec{d})$ равно 1.

К числу основных методов верификации ФП относятся излагаемые ниже методы вычислительной индукции и структурной индукции.

11.2 Метод вычислительной индукции

11.2.1 Описание метода

Многие задачи верификации ФП могут быть сведены к следующей задаче: пусть заданы полное ЧУМ P, непрерывный функционал $F : P \to P$ и подмножество $Q \subseteq P$, которое является **замкнутым**, т.е. удовлетворяет условию: для каждой цепи $p_0 \le p_1 \le \ldots$ элементов P верна импликация

$$(\forall\, i \ge 0 \quad p_i \in Q) \;\Rightarrow\; \sup_{i \ge 0}\, p_i \in Q.$$

Требуется доказать, что ННТ функционала F (которую мы ниже будем обозначать записью $\mathbf{fix}_F$) удовлетворяет соотношению

$$\mathbf{fix}_F \in Q. \tag{11.2}$$

Метод **вычислительной индукции (ВИ)** заключается в сведении задачи доказательства соотношения (11.2) к проверке двух условий:

1. $\mathbf{0} \in Q$ (где $\mathbf{0}$ – наименьший элемент ЧУМ P),
2. $\forall\, p \in Q \quad F(p) \in Q$.

Нетрудно видеть, что если эти условия выполнены, то для каждого $i \geq 0$ верно соотношение $F^i(\mathbf{0}) \in Q$, откуда на основании замкнутости Q следует, что $\mathbf{fix}_F = \sup_{i\geq 0} F^i(\mathbf{0}) \in Q$.

Второе из этих условий можно заменить на следующее условие: для каждого $i > 0$ верна импликация

$$(\forall\, j < i \quad F^j(\mathbf{0}) \in Q) \;\Rightarrow\; F^i(\mathbf{0}) \in Q. \tag{11.3}$$

P и F могут иметь, например, следующий вид:

- $P = P_\Sigma$, где Σ – ФП, и $F = F_\Sigma$, или
- $P = P_\Sigma \times P_{\Sigma'} \times \ldots$, где
  - Σ, Σ', $\ldots$ – ФП,
  - отношение порядка на P определяется покомпонентно:

    $$(p_1, p_1', \ldots) \leq (p_2, p_2', \ldots) \quad \Leftrightarrow \quad p_1 \leq p_2,\;\; p_1' \leq p_2',\;\; \ldots;$$

  - наименьшим элементом P является список наименьших элементов ЧУМ P_Σ, $P_{\Sigma'}$, $\ldots$

  и $\forall\, \vec{f} = (f, f', \ldots) \in P \quad F(\vec{f}) \stackrel{\text{def}}{=} (F_\Sigma(f), F_{\Sigma'}(f'), \ldots)$.

При верификации ФП методом ВИ можно использовать следующие утверждения.

- Если Q_1 и Q_2 – замкнутые подмножества полного ЧУМ P, то подмножество $Q_1 \cap Q_2$ тоже замкнуто.
- Для любых непрерывных функционалов F_1 и F_2 на P подмножества

  $$\{p \in P \mid F_1(p) \leq F_2(p)\} \quad \text{и} \quad \{p \in P \mid F_1(p) = F_2(p)\}$$

  являются замкнутыми.

11.2.2 Примеры верификации функциональных программ методом вычислительной индукции

В этом параграфе излагаются некоторые примеры верификации ФП методом ВИ. Во всех этих примерах проверка первого условия ($\mathbf{0} \in Q$) опущена по причине ее тривиальности. Мы будем предполагать, что функции, используемые в ФП в этом пункте $(p, h, k, \ldots)$, принимают значение ω на аргументе ω.

Пример 1

Требуется доказать, что $\forall\, x \quad \sigma\sigma(x) = \sigma(x)$, где

$$\Sigma: \quad \varphi(x) = p(x)\,?\,x : \varphi\varphi h(x).$$

Определим $Q \stackrel{\text{def}}{=} \{f \in P_\Sigma \mid \sigma f = f\}$.
Пусть $f \in Q$. Докажем, что $F_\Sigma(f) \in Q$, т.е.

$$\sigma F_\Sigma(f) = F_\Sigma(f). \tag{11.4}$$

$$\begin{aligned}
&\text{Лев.Ч.}(11.4) \;= \sigma(p(x)?x : ffh(x)) = \\
&= p(x)?\ \sigma(x) : \sigma ffh(x) = \\
&= p(x)?\ (p(x)?\ x : \sigma\sigma h(x)) : \sigma ffh(x) = \\
&= p(x)?\ x : \sigma ffh(x) = \\
&= p(x)?\ x : ffh(x) = \ \text{Прав.Ч.}(11.4).
\end{aligned}$$

Отметим, что если определить $Q \stackrel{\text{def}}{=} \{f \in P_\Sigma \mid ff = f\}$, то тогда доказать требуемое свойство не получится.

Пример 2

Требуется доказать, что $\forall\, x, y \quad h\,\sigma(x, y) = \sigma(x, h(y))$, где

$$\Sigma: \quad \varphi(x, y) = p(x)?\ y : h\varphi(k(x), y).$$

Определим $Q \stackrel{\text{def}}{=} \{f \in P_\Sigma \mid \forall\, x, y \quad hf(x, y) = f(x, h(y))\}$.
Пусть $f \in Q$. Докажем, что $F_\Sigma(f) \in Q$, т.е. $\forall\, x, y$

$$h(F_\Sigma(f)(x, y)) = F_\Sigma(f)(x, h(y)). \tag{11.5}$$

$$\begin{aligned}
&\text{Лев.Ч.}(11.5) = h(p(x)?\ y : hf(k(x), y)) = \\
&= p(x)?\ h(y) : hhf(k(x), y) = \\
&= p(x)?\ h(y) : hf(k(x), h(y)) = \text{Прав.Ч.}(11.5).
\end{aligned}$$

Пример 3

Требуется доказать, что $\forall\, x \quad \sigma_1(x) = \sigma_3(x)$, где

$$\Sigma: \quad \begin{cases} \varphi_1(x) = p(x)?\ \varphi_1\varphi_2 h(x) : \varphi_2 g(x) \\ \varphi_2(x) = q(x)?\ k\varphi_2\varphi_1(x) : kh(x) \\ \varphi_3(x) = p(x)?\ \varphi_3 k\varphi_4 h(x) : k\varphi_4 g(x) \\ \varphi_4(x) = q(x)?\ k\varphi_4\varphi_3(x) : h(x) \end{cases}.$$

Определим $Q \stackrel{\text{def}}{=} \{\vec{f} \in P_\Sigma \mid (f_1 = f_3) \wedge (f_2 = kf_4)\}$.
Пусть $\vec{f} \in Q$. Докажем, что $F_\Sigma(\vec{f}) \in Q$, т.е.

$$(F_\Sigma(\vec{f}))_1 = (F_\Sigma(\vec{f}))_3, \tag{11.6}$$

$$(F_\Sigma(\vec{f}))_2 = k(F_\Sigma(\vec{f}))_4. \tag{11.7}$$

$$\begin{aligned} &\text{Лев.Ч.}(11.6) = p(x)?\ f_1 f_2 h(x) : f_2 g(x) = \\ &= p(x)?\ f_3 k f_4 h(x) : k f_4 g(x) = \text{Прав.Ч.}(11.6). \\ &\text{Лев.Ч.}(11.7) = q(x)?\ k f_2 f_1(x) : kh(x) = \\ &= k(q(x)?\ f_2 f_1(x) : h(x)) = \\ &= k(q(x)?\ k f_4 f_3(x) : h(x)) = \text{Прав.Ч.}(11.7). \end{aligned}$$

Пример 4

Требуется доказать, что $\forall\, x \quad \sigma(x, 0, x) = \sigma'(x, 0)$, где

$$\begin{aligned} \Sigma: &\quad \varphi(x, y, z) = (x = 0)?\ y : \varphi(x - 1, y + z, z), \\ \Sigma': &\quad \varphi(x, y) = (x = 0)?\ y : \varphi(x - 1, y + 2x - 1). \end{aligned}$$

Мы докажем более общее соотношение:

$$\forall\, x, y \quad \sigma(y, x(x - y), x) = \sigma'(y, x^2 - y^2). \tag{11.8}$$

Искомое соотношение является частным случаем (11.8) (при $y = x$).
Определим

- $P \stackrel{\text{def}}{=} P_\Sigma \times P_{\Sigma'}$;
- $F : P \to P, \quad F(f, f') \stackrel{\text{def}}{=} (F_\Sigma(f), F_{\Sigma'}(f'))$;
- $Q \stackrel{\text{def}}{=} \left\{ (f, f') \in P \;\middle|\; \begin{array}{ll} \forall\, x, y & f(y, x(x - y), x) = \\ & = f'(y, x^2 - y^2) \end{array} \right\}$.

Пусть $(f, f') \in Q$. Докажем, что $(F_\Sigma(f), F_{\Sigma'}(f')) \in Q$, т.е. $\forall\, x, y$

$$F_\Sigma(f)(y, x(x-y), x) = F_{\Sigma'}(f')(y, x^2 - y^2). \tag{11.9}$$

Лев.Ч.(11.9) =
$= (y = 0)?\ x(x-y) : f(y-1, x(x-y)+x, x) =$
$= (y = 0)?\ x^2 : f(y-1, x(x-(y-1)), x) =$
$= (y = 0)?\ x^2 : f'(y-1, x^2 - (y-1)^2) =$
$= (y = 0)?\ x^2 - y^2 : f'(y-1, (x^2 - y^2) + 2y - 1) =$
$=$ Прав.Ч.(11.9).

Пример 5

Требуется доказать, что $\forall\, x, y \quad \sigma(x, y) = \sigma'(x, y)$, где

$$\begin{array}{ll} \Sigma: & \varphi(x, y) = p(x)?\ y : h\varphi(k(x), y), \\ \Sigma': & \varphi(x, y) = p(x)?\ y : \varphi(k(x), h(y)). \end{array}$$

Мы докажем более сильное соотношение:

$$\forall\, x, y \quad \left\{ \begin{array}{l} \sigma(x, y) = \sigma'(x, y) \\ \sigma'(x, h(y)) = h\sigma'(x, y) \end{array} \right\}.$$

Определим

- P и F так же, как в предыдущем примере и
- $Q \stackrel{\text{def}}{=} \{(f, f') \in P \mid \forall\, x, y \quad \left\{ \begin{array}{l} f(x, y) = f'(x, y) \\ f'(x, h(y)) = hf'(x, y) \end{array} \right\}\}$.

Пусть $(f, f') \in Q$. Докажем, что $(F_\Sigma(f), F_{\Sigma'}(f')) \in Q$, т.е. $\forall\, x, y$

$$F_\Sigma(f)(x, y) = F_{\Sigma'}(f')(x, y), \tag{11.10}$$

$$F_{\Sigma'}(f')(x, h(y)) = hF_{\Sigma'}(f')(x, y). \tag{11.11}$$

Лев.Ч.(11.10) $= p(x)?\ y : hf(k(x), y) =$
$= p(x)?\ y : hf'(k(x), y) =$
$= p(x)?\ y : f'(k(x), h(y)) =$ Прав.Ч.(11.10).
Лев.Ч.(11.11) $= p(x)?\ h(y) : f'(k(x), hh(y)) =$
$= p(x)?\ h(y) : hf'(k(x), h(y)) =$
$= h(p(x)?\ y : f'(k(x), h(y))) =$ Прав.Ч.(11.11).

Другой способ доказательства $\sigma = \sigma'$ заключается в том, что вместо условия 2 в определении метода ВИ, т.е. вместо импликации

$$(f, f') \in Q \quad \Rightarrow \quad F(f, f') \in Q,$$

доказывается утверждение: $\forall\, i \geq 0$ верна импликация (11.3), в которой

$$Q \stackrel{\text{def}}{=} \{(f, f') \in P \mid f = f'\}.$$

Для каждого $i \geq 0$ элемент $F^i(\mathbf{0})$ является парой функций, которую мы будем обозначать записью $(f^i, (f')^i)$.

(11.3) доказывается следующим образом.

1. Если $i = 0$, то (11.3) представляет собой утверждение $f^0 = (f')^0$, т.е. $\textcircled{\omega} = \textcircled{\omega}$, что, очевидно, верно.

2. Если $i = 1$, то (11.3) представляет собой утверждение

$$(f^0 = (f')^0) \;\Rightarrow\; (f^1 = (f')^1),$$

   т.е. $F_\Sigma(\textcircled{\omega}) = F_{\Sigma'}(\textcircled{\omega})$. Данное равенство обосновывается следующим образом:
   $F_\Sigma(\textcircled{\omega})(x, y) = p(x)?\ y : h\textcircled{\omega}(k(x), y) =$
   $= p(x)?\ y : h(\omega) = p(x)?\ y : \omega =$
   $= p(x)?\ y : \textcircled{\omega}(k(x), h(y)) = F_{\Sigma'}(\textcircled{\omega})(x, y).$

3. Пусть $i > 1$. Для доказательства равенства $f^i = (f')^i$ мы будем использовать предположение об истинности равенств

$$f^{i-1} = (f')^{i-1} \quad \text{и} \quad f^{i-2} = (f')^{i-2}.$$

   $f^i(x, y) = p(x)?\ y : hf^{i-1}(k(x), y) =$
   $= p(x)?\ y : h(f')^{i-1}(k(x), y) =$
   $= p(x)?\ y : h(pk(x)?\ y : (f')^{i-2}(kk(x), h(y))) =$
   $= p(x)?\ y : h(pk(x)?\ y : f^{i-2}(kk(x), h(y))) =$
   $= p(x)?\ y : (pk(x)?\ h(y) : hf^{i-2}(kk(x), h(y))) =$
   $= p(x)?\ y : f^{i-1}(k(x), h(y)) =$
   $= p(x)?\ y : (f')^{i-1}(k(x), h(y)) = (f')^i(x, y).$

11.3 Метод структурной индукции

11.3.1 Описание метода

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

- доказываемое утверждение о ФП имеет вид (11.1) и

- существует фундированное 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

$$c : D_{\tau(\vec{x})} \to P, \tag{11.12}$$

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

$$(\forall\, \vec{d'} \in D_{\tau(\vec{x})} \quad c(\vec{d'}) < c(\vec{d}) \;\Rightarrow\; e(\vec{d'}) = 1) \quad \Rightarrow \quad e(\vec{d}) = 1. \tag{11.13}$$

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 (11.13) для $\vec{d} = \vec{d}_0$.

11.3.2 Примеры верификации функциональных программ методом структурной индукции

В излагаемых ниже примерах 1, 2 и 3 ФЧУМ P совпадает с $D_{\tau(\vec{x})}$ и (11.12) является тождественной функцией. В примерах 4 и 5 $\tau(\vec{x}) = \mathbf{S}$, $P = D_{\mathbf{N}}$ 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.

Пример 1 (функция Аккермана)

Требуется доказать, что $\forall\, x, y \in D_{\mathbf{N}} \quad \sigma(x,y) \neq \omega$, где

$$\Sigma : \begin{cases} \varphi(x,y) = (x=0) & ?\ \ y+1 \\ & :(y=0) \quad ?\ \varphi(x-1,1) \\ & \qquad\qquad :\varphi(x-1,\varphi(x,y-1)) \end{cases}$$

(переменные x, y имеют тип $\mathbf{N}$).

Поскольку $D_{(x,y)} = D^2_{\mathbf{N}}$ – ФЧУМ относительно лексикографического порядка, то требуемое свойство можно доказать методом СИ.

- Если $x = 0$, то $\sigma(x,y) = y + 1 \neq \omega$.
- Пусть $x > 0$, и $\forall\,(x',y') < (x,y) \quad \sigma(x',y') \neq \omega$. Докажем, что $\sigma(x,y) \neq \omega$.
  - Если $y = 0$, то $\sigma(x,y) = \sigma(x-1,1)$. Поскольку $(x-1,1) < (x,y)$, то, по индуктивному предположению, $\sigma(x-1,1) \neq \omega$.
  - Если $y \neq 0$, то т.к. $(x,y-1) < (x,y)$, по индуктивному предположению, $\sigma(x,y-1) \neq \omega$, откуда, используя неравенство

    $$(x-1,\sigma(x,y-1)) < (x,y),$$

    получаем требуемое соотношение $\sigma(x,y) \neq \omega$.

Пример 2 (вычисление факториала)

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

$$\sigma'(x,0) = \sigma(x), \tag{11.14}$$

где

$$\begin{aligned} \Sigma :&\quad \varphi(x) = (x=0)?\ 1 : x \cdot \varphi(x-1), \\ \Sigma' :&\quad \varphi(x,y) = (x=y)?\ 1 : \varphi(x,y+1) \cdot (y+1) \end{aligned}$$

(переменные x, y имеют тип $\mathbf{N}$).

Мы докажем более общее утверждение: $\forall\, x \in D_{\mathbf{N}}$

$$\forall\, y \in D_{\mathbf{N}} \quad \Big(\sigma'(x+y,y) \cdot \sigma(y) = \sigma(x+y)\Big). \tag{11.15}$$

(11.14) является частным случаем (11.15) (при $y = 0$).

- Если $x = 0$, то (11.15) имеет вид

$$\forall\, y \in D_{\mathbf{N}} \quad (\sigma'(y, y) \cdot \sigma(y) = \sigma(y)),$$

что верно, т.к. $\forall\, y \in D_{\mathbf{N}} \quad \sigma'(y, y) = 1$.

- Пусть $x > 0$, и для каждого натурального $z < x$ верно утверждение

$$\forall\, y \in D_{\mathbf{N}} \quad \Big(\sigma'(z + y, y) \cdot \sigma(y) = \sigma(z + y)\Big).$$

Докажем, что тогда будет верно и (11.15).

Для каждого натурального y

$$\begin{array}{l}\sigma'(x + y, y) \cdot \sigma(y) = \\ = \sigma'(x + y, y + 1) \cdot (y + 1) \cdot \sigma(y) = \\ = \sigma'(x + y, y + 1) \cdot \sigma(y + 1) = \\ = \sigma'((x - 1) + (y + 1), y + 1) \cdot \sigma(y + 1) = \\ = \sigma((x - 1) + (y + 1)) = \sigma(x + y).\end{array}$$

Пример 3 (функция Фибоначчи)

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

$$\sigma'(x, 1, 1) = \sigma(x), \tag{11.16}$$

где

$$\begin{array}{ll}\Sigma: & \varphi(x) = (x \in \{0, 1\})?\ 1 : \varphi(x - 1) + \varphi(x - 2), \\ \Sigma': & \varphi(x, a, b) = (x = 0)?\ a : \varphi(x - 1, b, a + b)\end{array}$$

(переменные x, a, b имеют тип $\mathbf{N}$).

Мы докажем более общее утверждение: $\forall\, x \in D_{\mathbf{N}}$

$$\forall\, y \in D_{\mathbf{N}} \quad \sigma(x + y) = \sigma'(x, \sigma(y), \sigma(y + 1)). \tag{11.17}$$

((11.16) является частным случаем (11.17) (при $y = 0$).

- Если $x = 0$, то (11.17) имеет вид

$$\forall\, y \in D_{\mathbf{N}} \quad \sigma(y) = \sigma'(0, \sigma(y), \sigma(y + 1)),$$

что верно согласно определению ФП Σ'.

- Пусть $x > 0$, и для каждого натурального $z < x$ верно утверждение

$$\forall\, y \in D_{\mathbf{N}} \quad \sigma(z+y) = \sigma'(z, \sigma(y), \sigma(y+1)).$$

Докажем, что тогда будет верно и (11.17).

Для каждого натурального y

$$\begin{array}{l} \sigma(x+y) = \\ = \sigma((x-1)+(y+1)) = \\ = \sigma'(x-1, \sigma(y+1), \sigma(y+2)) = \\ = \sigma'(x-1, \sigma(y+1), \sigma(y+1)+\sigma(y)) = \\ = \sigma'(x, \sigma(y), \sigma(y+1)). \end{array}$$

Пример 4 (функция инвертирования строк)

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

$$r(x) \neq \omega \quad \text{и} \quad rr(x) = x, \tag{11.18}$$

где r – функция инвертирования строк, определяемая следующим образом: $r(x) \stackrel{\text{def}}{=} \sigma(x, \varepsilon)$, где

$$\Sigma: \quad \varphi(x, y) = (x = \varepsilon)?\ y : \varphi(x_t, x_h y)$$

(переменные x, y имеют тип $\mathbf{S}$, x_h и x_t – голова и хвост x соответственно).

Мы докажем более общее утверждение: $\forall\, x \in D_{\mathbf{S}}$

$$\forall\, y \in D_{\mathbf{S}} \quad \sigma(x, y) \neq \omega \quad \text{и} \quad r\sigma(x, y) = \sigma(y, x). \tag{11.19}$$

(11.18) является частным случаем (11.19) (при $y = \varepsilon$).

- Если $x = \varepsilon$, то (11.19) имеет вид

$$\forall\, y \in D_{\mathbf{S}} \quad \sigma(\varepsilon, y) \neq \omega \quad \text{и} \quad r\sigma(\varepsilon, y) = \sigma(y, \varepsilon). \tag{11.20}$$

Согласно определению ФП Σ и функции r, (11.20) можно переписать в виде

$$\forall\, y \in D_{\mathbf{S}} \quad y \neq \omega \quad \text{и} \quad r(y) = r(y),$$

что, очевидно, верно.

- Пусть $x \neq \varepsilon$, и для каждой строки $z < x$ верно утверждение

$$\forall\, y \in D_{\mathbf{S}} \quad \sigma(z, y) \neq \omega \quad \text{и} \quad r\sigma(z, y) = \sigma(y, z). \tag{11.21}$$

Докажем, что тогда будет верно и (11.19).

Используя (11.21) для $z \stackrel{\text{def}}{=} x_t$ (что возможно, т.к. $x_t < x$) и определение ФП Σ, получаем: для каждой строки y

– $\sigma(x,y) = \sigma(x_h x_t, y) = \sigma(x_t, x_h y) \neq \omega$
– $r\sigma(x,y) = r\sigma(x_h x_t, y) = r\sigma(x_t, x_h y) = \sigma(x_h y, x_t) =$
$= \sigma(y, x_h x_t) = \sigma(y,x)$.

Пример 5 (сортировка)

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

$$\mathbf{ord}(\mathbf{sort}(x)) = 1, \tag{11.22}$$

где

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

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

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

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

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 (11.22) переписывается в виде

$$\mathbf{ord}(x_h \to \mathbf{sort}(x_t)) = 1. \tag{11.23}$$

По индуктивному предположению, верно равенство

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

из которого следует (11.23) по нижеследующей лемме.

Лемма
Имеет место импликация

$$\mathbf{ord}(y) = 1 \quad \Rightarrow \quad \mathbf{ord}(a \to y) = 1. \tag{11.24}$$

Доказательство.
Доказываем лемму индукцией по y.
Если $y = \varepsilon$, то правая часть в (11.24) имеет вид

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

что верно по определению **ord**.

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

$$\mathbf{ord}(z) = 1 \quad \Rightarrow \quad \mathbf{ord}(a \to z) = 1. \tag{11.25}$$

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

$$\mathbf{ord}(cd) = 1 \quad \Rightarrow \quad \mathbf{ord}(a \to cd) = 1. \tag{11.26}$$

Для доказательства импликации (11.26) 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

$$\mathbf{ord}(c(a\varepsilon)) = 1. \tag{11.27}$$

  (11.27) 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 $c < a$

$$\mathbf{ord}(c(a \to pq)) = 1. \tag{11.28}$$

  1. Если $a \leq p$, то (11.28) имеет вид

$$\mathbf{ord}(c(a(pq))) = 1. \tag{11.29}$$

  Поскольку $c < a \leq p$, то (11.29) 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 (11.28) имеет вид

$$\mathbf{ord}(cp(a \to q)) = 1. \tag{11.30}$$

Поскольку по предположению

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

то $c \leq p$, и поэтому (11.30) можно переписать в виде

$$\mathbf{ord}(p(a \to q)) = 1. \tag{11.31}$$

При $p < a$

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

поэтому (11.31) можно переписать в виде

$$\mathbf{ord}(a \to d) = 1. \tag{11.32}$$

(11.32) следует по индуктивному предположению для леммы (т.е. из импликации (11.25), в которой $z \stackrel{\text{def}}{=} d$) из равенства

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

которое обосновывается цепочкой равенств

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

11.4 Другие методы верификации функциональных программ

11.4.1 Оценка наименьшей неподвижной точки функциональной программы сверху

Если проверяемое свойство ФП Σ имеет вид $\sigma \leq f$, где

- f – заданная монотонная функция и
- отношение порядка на функциях понимается в смысле определения из пункта 9.1.4,

то для доказательства этого свойства достаточно доказать неравенство

$$F_\Sigma(f) \leq f.$$

Пример

Требуется доказать, что $\sigma \leq f$, где

$$\begin{array}{ll}\Sigma: & \varphi(x) = (x > 100)?\ x - 10 : \varphi\varphi(x + 11)\\ f \stackrel{\text{def}}{=} & (x > 100)?\ x - 10 : 91\end{array}$$

Докажем, что $F_\Sigma(f) \leq f$, т.е.

$$(x > 100)?\ (x - 10) : ff(x + 11) \leq f.$$

Поскольку

$$\begin{array}{l}f(x + 11) = (x + 11 > 100)?\ x + 11 - 10 : 91 =\\ = (x \geq 90)?\ x + 1 : 91\end{array},$$

то

$$\begin{array}{l}ff(x + 11) =\\ = f((x \geq 90)?\ x + 1 : 91) =\\ = (x \geq 90)?\ f(x + 1) : f(91) =\\ = (x \geq 90)?\ f(x + 1) : 91.\end{array}$$

Таким образом,

$$\begin{array}{lll}F_\Sigma(f) = (x > 100) & ?\ (x - 10) & \\ & : (x \geq 90) \quad ?\ f(x + 1) : 91\end{array} \tag{11.33}$$

Подвыражение $f(x+1)$ в (11.33) вычисляется только для $x \in \{90, \ldots, 100\}$, и для всех таких значений x значение $f(x + 1)$ равно 91. Отсюда следует равенство (11.33) $= f$.

11.4.2 Эквивалентные преобразования функциональных программ

Иногда доказательство свойств ННТ ФП можно упростить, если вместо анализируемой ФП Σ рассматривать ФП Σ', получаемую из Σ раскрытием некоторых подтермов, или применением подстановки вида (σ/φ). Можно доказать, что такие Σ и Σ' имеют одинаковые ННТ. Ниже мы доказываем это для случая, когда Σ' получается из Σ раскрытием одного подтерма или применением одной подстановки вида (σ/φ).

Теорема 19

Пусть задана ФП $\Sigma:\ \varphi(\vec{x}) = e$ и терм e' получается из e раскрытием одного подтерма. Тогда $\sigma = \sigma'$, где $\Sigma':\ \varphi(\vec{x}) = e'$.

Доказательство.

Для каждого $\vec{d} \in D_{\tau(\vec{x})}$ терм $e'(\vec{d})$ получается из терма $e(\vec{d})$ заменой подтерма вида $\varphi(e_1, \ldots, e_n)$ на терм $e^{(e_1/x_1,\ldots,e_n/x_n)}$. Согласно утверждению 1 в доказательстве теоремы 11, значения термов $e(\vec{d})^{(\sigma/\varphi)}$ и $e'(\vec{d})^{(\sigma/\varphi)}$ совпадают. Поскольку значение первого из этих термов равно $\sigma(\vec{d})$, то

$$\sigma(\vec{d}) = (e')^{(\sigma/\varphi)}(\vec{d}),$$

т.е. σ – НТ Σ', откуда следует неравенство $\sigma' \leq \sigma$.

Докажем методом ВИ обратное неравенство: $\sigma \leq \sigma'$.

Определим

- $P \stackrel{\text{def}}{=} P_\Sigma \times P_{\Sigma'}$,
- $F : P \to P$, $F(f, f') \stackrel{\text{def}}{=} (F_\Sigma(f), F_{\Sigma'}(f')) = (e^{(f/\varphi)}, (e')^{(f'/\varphi)})$,
- $Q \stackrel{\text{def}}{=} \{(f, f') \in P \mid \left\{ \begin{array}{l} f \leq f' \\ f \leq e^{(f/\varphi)} \end{array} \right\}\}$.

Докажем, что верна импликация $p \in Q \;\Rightarrow\; F(p) \in Q$, т.е.

$$\left\{ \begin{array}{l} f \leq f' \\ f \leq e^{(f/\varphi)} \end{array} \right\} \quad \Rightarrow \quad \left\{ \begin{array}{l} e^{(f/\varphi)} \leq (e')^{(f'/\varphi)} \\ e^{(f/\varphi)} \leq e^{(e^{(f/\varphi)}/\varphi)} \end{array} \right\}. \tag{11.34}$$

Обозначим конъюнктивные члены в правой части (11.34) через A и B и докажем, что каждый из них следует из левой части (11.34).

A: Из $f \leq f'$ и монотонности $F_{\Sigma'}$ следует неравенство

$$(e')^{(f/\varphi)} \leq (e')^{(f'/\varphi)}. \tag{11.35}$$

Искомое неравенство следует из (11.35) и из неравенства

$$e^{(f/\varphi)} \leq (e')^{(f/\varphi)}. \tag{11.36}$$

(11.36) верно потому, что

- $(e')^{(f/\varphi)}$ получается из $e^{(f/\varphi)}$ заменой подтерма вида $f(e_1, \ldots, e_n)$ на $(e^{(f/\varphi)})^{(e_1/x_1,\ldots,e_n/x_n)}$ и
- из $f \leq e^{(f/\varphi)}$ следует неравенство

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

B: Из $f \le e^{(f/\varphi)}$ и монотонности F_Σ следует неравенство

$$e^{(f/\varphi)} \le e^{(e^{(f/\varphi)}/\varphi)}.$$

Таким образом, $\mathbf{fix}_F = (\sigma, \sigma') \in Q$, откуда следует $\sigma \le \sigma'$. ■

Теорема 20

Пусть задана ФП $\Sigma : \varphi(\vec{x}) = e$ и терм e' получается из e заменой одного из вхождений функциональной переменной φ на σ.

Тогда $\sigma = \sigma'$, где $\Sigma' : \varphi(\vec{x}) = e'$.

Доказательство.

Поскольку $(e')^{(\sigma/\varphi)} = e^{(\sigma/\varphi)} = \sigma$, то σ – НТ Σ', поэтому $\sigma' \le \sigma$.

Докажем методом ВИ, что $\sigma \le \sigma'$.

Определим

- $P \stackrel{\text{def}}{=} P_\Sigma \times P_{\Sigma'}$,
- $F : P \to P$, $F(f, f') \stackrel{\text{def}}{=} (F_\Sigma(f), F_{\Sigma'}(f'))$,
- $Q \stackrel{\text{def}}{=} \{(f, f') \in P \mid \left\{ \begin{array}{l} f \le f' \\ f \le \sigma \end{array} \right\}\}$.

Докажем, что верна импликация $p \in Q \;\Rightarrow\; F(p) \in Q$, т.е.

$$\left\{ \begin{array}{l} f \le f' \\ f \le \sigma \end{array} \right\} \Rightarrow \left\{ \begin{array}{l} e^{(f/\varphi)} \le (e')^{(f'/\varphi)} \\ e^{(f/\varphi)} \le \sigma \end{array} \right\}. \tag{11.37}$$

Обозначим конъюнктивные члены в правой части (11.37) через A и B и докажем, что каждый из них следует из левой части (11.37).

A: Из $f \le f'$ и монотонности $F_{\Sigma'}$ следует неравенство

$$(e')^{(f/\varphi)} \le (e')^{(f'/\varphi)}. \tag{11.38}$$

Искомое неравенство следует из (11.38) и из неравенства

$$e^{(f/\varphi)} \le (e')^{(f/\varphi)}. \tag{11.39}$$

(11.39) верно потому, что

- $e^{(f/\varphi)}$ получается из e заменой всех вхождений φ на f,
- $(e')^{(f/\varphi)}$ можно получить из e заменой одного из вхождений φ на σ и остальных вхождений φ – на f,

– $f \le \sigma$.

В: Из $f \le \sigma$ и монотонности F_Σ следует соотношение

$$e^{(f/\varphi)} \le e^{(\sigma/\varphi)} = \sigma.$$

Таким образом, $\mathbf{fix}_F = (\sigma, \sigma') \in Q$, откуда следует $\sigma \le \sigma'$. ■

Приведем два примера верификации ФП с использованием эквивалентных преобразований ФП.

Пример 1

Требуется доказать, что $\sigma = \sigma'$, где

$$\begin{array}{ll} \Sigma: & \varphi(x) = (x > 10)?\ x - 10 : \varphi\varphi(x + 13), \\ \Sigma': & \varphi(x) = (x > 10)?\ x - 10 : \varphi(x + 3), \end{array}$$

где x – типа $\mathbf{N}$.

Обозначим через Σ'' ФП, получаемую из Σ раскрытием подтерма, начинающегося со второго φ:

$$\Sigma'': \quad \varphi(x) = (x > 10)?\ x - 10 : A,$$

где $A = \varphi \begin{pmatrix} (x + 13 > 10) & ?\ x + 13 - 10 \\ & : \varphi\varphi(x + 13 + 13) \end{pmatrix}$.

Согласно теореме 19, $\sigma = \sigma''$.

Поскольку $x \ge 0$, то $x + 13 > 10$, поэтому можно упростить A до $\varphi(x + 3)$, в результате чего получим Σ'.

Пример 2

Требуется доказать, что $\sigma_1 = \sigma_3$, где

$$\Sigma: \quad \left\{ \begin{array}{l} \varphi_1(x) = p(x)?\ \varphi_3\varphi_2\varphi_2 f(x) : g(x) \\ \varphi_2(x) = q(x)?\ \varphi_3 h(x) : k(x) \\ \varphi_3(x) = p(x)?\ \varphi_1\varphi_4 f(x) : g(x) \\ \varphi_4(x) = q(x)?\ \varphi_2\varphi_3 h(x) : \varphi_2 k(x) \end{array} \right. .$$

Заменим в Σ первое вхождение φ_2 в e_1 и оба вхождения φ_2 в e_4 на ФС σ_2. ФП, получившуюся в результате этой замены, обозначим тем же символом Σ. Согласно вышесказанному, ННТ ФП

$$\Sigma: \quad \left\{ \begin{array}{l} \varphi_1(x) = p(x)?\ \varphi_3\sigma_2\varphi_2 f(x) : g(x) \\ \varphi_2(x) = q(x)?\ \varphi_3 h(x) : k(x) \\ \varphi_3(x) = p(x)?\ \varphi_1\varphi_4 f(x) : g(x) \\ \varphi_4(x) = q(x)?\ \sigma_2\varphi_3 h(x) : \sigma_2 k(x) \end{array} \right.$$

совпадает с ННТ исходной ФП.

Определим замкнутое подмножество $Q \subseteq P_\Sigma$:

$$Q \stackrel{\text{def}}{=} \{\vec{f} \in P_\Sigma \mid \left\{ \begin{array}{l} f_1 = f_3 \\ \sigma_2 f_2 = f_4 \end{array} \right\}\}.$$

Нетрудно доказать, что верна импликация

$$p \in Q \;\Rightarrow\; F_\Sigma(p) \in Q,$$

т.е. из $\left\{ \begin{array}{l} f_1 = f_3 \\ \sigma_2 f_2 = f_4 \end{array} \right\}$ следует, что

$$\left\{ \begin{array}{l} p(x)?\ f_3\sigma_2 f_2 f(x) : g(x) = p(x)?\ f_1 f_4 f(x) : g(x) \\ \sigma_2(q(x)?\ f_3 h(x) : k(x)) = q(x)?\ \sigma_2 f_3 h(x) : \sigma_2 k(x) \end{array} \right\}.$$

Из замкнутости Q следует, что $\mathbf{fix}_{F_\Sigma} \in Q$, т.е.

$$\left\{ \begin{array}{l} \sigma_1 = \sigma_3 \\ \sigma_2\sigma_2 = \sigma_4 \end{array} \right\},$$

откуда следует искомое равенство $\sigma_1 = \sigma_3$.

Отметим, что в данном примере доказать требуемое свойство без использования эквивалентных преобразований довольно сложно.

Глава 12

Задачи и исследовательские проблемы

12.1 Нахождение наименьших неподвижных точек функциональных программ

1. Найти ННТ ФП

$$\Sigma:\ \varphi(x,y) = \ (x=0)?\ 0 : 1 + \varphi(x-1, \varphi(y-2, x)),$$

где x, y – типа $\mathbf{I}$.

2. Найти ННТ ФП

$$\Sigma:\ \varphi(x,y) = \ (x=0)?\ 1 : \varphi(x-1, \varphi(x-y, y)),$$

где x, y – типа $\mathbf{I}$.

12.2 Вид наименьших неподвижных точек

1. Доказать, что $\sigma = (xy=0)?\ x+y : 1$, где x, y – типа $\mathbf{N}$ и
$$\Sigma: \begin{cases} \varphi(x,y) = (xy=0) & ?\ x+y \\ & : \varphi(x-1, \varphi(x, y-1)) \end{cases}.$$
2. Доказать, что $\sigma = x+y+1$, где x, y – типа $\mathbf{N}$ и
$$\Sigma: \begin{cases} \varphi(x,y) = (xy=0) & ?\ x+y+1 \\ & : \varphi(\varphi(x-1,1), y-1) \end{cases}.$$
3. Доказать, что $\sigma = (x=0)?\ y : 1$, где x, y – типа $\mathbf{N}$ и
$$\Sigma: \begin{cases} \varphi(x,y) = (x=0) & ?\ y \\ & : (y=0) \quad ?\ \varphi(x-1,1) \\ & \qquad\ : \varphi(x-1, \varphi(x, y-1)) \end{cases}.$$

4. Доказать, что

$$\sigma = \left\{ \begin{array}{ll} (x > a) & ? \; x - b \\ & : a - b + c - rem(a - x, c) \end{array} \right. ,$$

где x – типа $\mathbf{I}$ и

- $\Sigma: \; \varphi(x) = (x > a)? \; x - b : \varphi\varphi(x + b + c)$,
- $a, \; b, \; c \in D_{\mathbf{I}}, \quad b, c > 0$,
- rem – функция взятия остатка от деления нацело.

5. Доказать, что

$$\sigma = (x > 10)? \; x - 10 : rem(x + 1, 3) + 1,$$

где x – типа $\mathbf{N}$ и

- $\Sigma: \; \varphi(x) = (x > 10)? \; x - 10 : \varphi\varphi(x + 13)$,
- rem – функция взятия остатка от деления нацело.

6. Доказать, что $\sigma = (x < 0)? \; x + 1 : 0$, где x – типа $\mathbf{I}$ и

$$\Sigma: \; \varphi(x) = (x < 0)? \; x + 1 : \varphi\varphi(x - 2).$$

7. Доказать, что $\sigma = (x > 100)? \; x - 10 : 91$, где x – типа $\mathbf{I}$ и

$$\Sigma: \quad \left\{ \begin{array}{ll} \varphi(x) = (x > 100) & ? \; x - 10 \\ & : \underbrace{\varphi \ldots \varphi}_{k}(x + 10(k - 1) + 1) \end{array} \right. ,$$

где k – произвольное целое число, такое, что $k \geq 2$.

8. Доказать, что $\sigma_1 = (x \leq 2)? \; x : 4$, где x – типа $\mathbf{N}$ и

$$\Sigma: \left\{ \begin{array}{l} \varphi_1(x) = \varphi_2(x, 0, 0) \\ \varphi_2(x, y, z) = \varphi_3(even(x)? \; \frac{x}{2} : 3x + 1, x, y, z) \\ \varphi_3(u, x, y, z) = (u = z)? \; u : \varphi_2(u, x, y) \end{array} \right. ,$$

где x, y, z, u – типа $\mathbf{N}$ и $even$ – функция проверки чётности.

12.3 Функции, определяемые функциональными программами

1. Верно ли, что $\sigma = \sigma'$, где

$$\Sigma: \left\{ \begin{array}{lll} \varphi(x,y) = & (x=0) & ?\ y \\ & : (x>0) & ?\ \varphi(x-1,y+1) \\ & & : \varphi(x+1,y-1) \end{array} \right. ,$$

$$\Sigma': \left\{ \begin{array}{lll} \varphi(x,y) = & (x=0) & ?\ y \\ & : (x>0) & ?\ \varphi(x-2,y+2)\ ? \\ & & : \varphi(x+2,y-2) \end{array} \right.$$

2. Доказать, что $\sigma = \sigma'_1$, где

$$\Sigma:\ \varphi(x) = (x=0)?\ 0 : \varphi h(x),$$
$$\Sigma': \left\{ \begin{array}{l} \varphi_1(x) = (x=0)?\ 0 : \varphi_1\varphi_2(x) \\ \varphi_2(x) = (x=0)?\ 0 : \varphi_2\varphi_2 h(x) \end{array} \right.$$

(x – типа $\mathbf{N}$).

3. Доказать, что $\sigma = \sigma'_1$, где

$$\Sigma:\ \varphi(x) = (x=0)?\ 0 : (p(x)?\ \varphi g(x) : \varphi h(x)),$$
$$\Sigma': \left\{ \begin{array}{l} \varphi_1(x) = (x=0)?\ 0 : \varphi_1\varphi_2(x) \\ \varphi_2(x) = (x=0)?\ 0 : (p(x)?\ g(x) : \varphi_2\varphi_2 h(x)) \end{array} \right.$$

(x – типа $\mathbf{N}$).

4. Доказать, что $\sigma = \sigma'$, где

$$\Sigma:\ \varphi(x) = p(x)?\ \varphi\varphi f(x) : x,$$
$$\Sigma':\ \varphi(x) = p(x)?\ \varphi f(x) : x.$$

5. Доказать, что $\sigma_1 = \sigma'$, где

$$\Sigma: \left\{ \begin{array}{l} \varphi_1(x) = p(x)?\ \varphi_2\varphi_1 f(x) : x \\ \varphi_2(x) = p(x)?\ x : \varphi_1 g(x) \end{array} \right. ,$$
$$\Sigma':\ \varphi(x) = p(x)?\ \varphi g\varphi f(x) : x.$$

6. Доказать, что $\sigma_1 = \sigma'_1$, где

$$\Sigma: \left\{ \begin{array}{l} \varphi_1(x) = p(x)?\ \varphi_1\varphi_2 f(x) : g(x) \\ \varphi_2(x) = p(x)?\ \varphi_2 h(x) : x \end{array} \right. ,$$
$$\Sigma': \left\{ \begin{array}{l} \varphi_1(x) = p(x)?\ \varphi_2 f(x) : g(x) \\ \varphi_2(x) = p(x)?\ \varphi_2 h(x) : g(x) \end{array} \right. .$$

7. Доказать, что $\sigma_1 = \sigma_1'$, где

$$\Sigma: \begin{cases} \varphi_1(x) = p(x)?\ \varphi_1\varphi_2 h(x) : \varphi_3(x) \\ \varphi_2(x) = q(x)?\ f\varphi_2\varphi_1(x) : fh(x) \\ \varphi_3(x) = qg(x)?\ \varphi_3(x) : lgg(x) \end{cases},$$
$$\Sigma': \begin{cases} \varphi_1(x) = p(x)?\ \varphi_1 f\varphi_2 h(x) : l\varphi_3 g(x) \\ \varphi_2(x) = q(x)?\ f\varphi_2\varphi_1(x) : h(x) \\ \varphi_3(x) = q(x)?\ \varphi_3(x) : g(x) \end{cases}.$$

8. Доказать, что $\sigma_1 = \sigma_1'$, где

$$\Sigma: \begin{cases} \varphi_1(x) = p(x)?\ \varphi_2\varphi_3 f(x) : \varphi_2 f\varphi_4 g(x) \\ \varphi_2(x) = q(x)?\ \varphi_2 f(x) : h(x) \\ \varphi_3(x) = r(x)?\ f\varphi_4 g\varphi_5 f(x) : \varphi_5 gk(x) \\ \varphi_4(x) = s(x)?\ \varphi_4 k(x) : g(x) \\ \varphi_5(x) = s(x)?\ \varphi_5 k(x) : fg(x) \end{cases},$$
$$\Sigma': \begin{cases} \varphi_1(x) = p(x)?\ \varphi_2 g\varphi_3 f(x) : \varphi_2 g(x) \\ \varphi_2(x) = s(x)?\ \varphi_2 k(x) : \varphi_4 fg(x) \\ \varphi_3(x) = r(x)?\ \varphi_5 f(x) : k(x) \\ \varphi_4(x) = q(x)?\ \varphi_4 f(x) : h(x) \\ \varphi_5(x) = s(x)?\ \varphi_5 k(x) : fg(x) \end{cases}.$$

9. Доказать, что $\sigma_1 = \sigma_1'$, где

$$\Sigma: \begin{cases} \varphi_1(x) = \ p(x)\ \ ?\ \begin{pmatrix} q(x) & ?\ \varphi_3\varphi_2 g(x) \\ & :\ \varphi_2\varphi_3 g(x) \end{pmatrix} : h(x) \\ \varphi_2(x) = q(x)?\ \varphi_1 f(x) : k(x) \\ \varphi_3(x) = q(x)?\ \varphi_4 f(x) : k(x) \\ \varphi_4(x) = p(x)?\ \varphi_5 g(x) : h(x) \\ \varphi_5(x) = q(x)?\ \varphi_2\varphi_4 f(x) : \varphi_3 k(x) \end{cases},$$
$$\Sigma': \begin{cases} \varphi_1(x) = p(x)?\ \varphi_2\varphi_2 g(x) : h(x) \\ \varphi_2(x) = q(x)?\ \varphi_1 f(x) : k(x) \end{cases}.$$

10. Доказать, что $\sigma = \sigma'$, где

$$\Sigma:\ \varphi(x) = (x = 0)?\ 1 : 2\varphi(x-1),$$
$$\Sigma':\ \varphi(x) = (x = 0)?\ 1 : \begin{pmatrix} odd(x) & ?\ 2\varphi(\frac{x-1}{2})^2 \\ & :\ \varphi(\frac{x}{2})^2 \end{pmatrix},$$

где $odd(x) = 1$, если x – нечётное, и 0 – иначе.

11. Доказать, что $\forall x \quad \sigma(x,0,0) = \sigma'(x,0)$, где

$$\begin{array}{ll} \Sigma: & \varphi(x,y,z) = \\ & \quad = (y=x)?\ z : \varphi(x, y+1, z+3y(y+1)+1)\ \text{'} \\ \Sigma': & \varphi(x,y) = (x=0)?\ y : \varphi(x-1, y+3x(x-1)+1)\ . \end{array}$$

12. Доказать, что $\forall x \quad \sigma(x) = \sigma'(x,0) = \sigma''(x,1,0)$, где

$$\begin{array}{l} \Sigma: \ \varphi(x) = (x=0)?\ 0 : x + \varphi(x-1), \\ \Sigma': \ \varphi(x,y) = (x=0)?\ y : \varphi(x-1, x+y), \\ \Sigma'': \ \varphi(x,y,z) = (y=x+1)?\ z : \varphi(x, y+1, y+z). \end{array}$$

13. Доказать, что $\forall x, y > 0 \quad \sigma(x,y) = \sigma'(x,y)$, где

$$\begin{array}{ll} \Sigma: & \left\{ \begin{array}{ll} \varphi(x,y) = & (x=0)?\ 0 \\ & \quad : (\varphi(x-1,y)+1=y)?\ 0 \\ & \quad : \varphi(x-1,y)+1 \end{array} \right. , \\ \Sigma': & \varphi(x,y) = (x<y)?\ x : \varphi(x-y, y) \end{array}$$

(обе ФП вычисляют остаток от деления x на y).

14. Доказать, что $\forall x \quad \sigma(x) = \sigma'(x,1) = \sigma''(x,0) = \sigma'''(x,0,1)$, где

$$\begin{array}{l} \Sigma: \ \varphi(x) = (x=0)?\ 1 : x\varphi(x-1), \\ \Sigma': \ \varphi(x,y) = (x=0)?\ y : \varphi(x-1, xy), \\ \Sigma'': \ \varphi(x,y) = (y=x)?\ 1 : (y+1)\varphi(x, y+1), \\ \Sigma''': \ \varphi(x,y,z) = (y=x)?\ z : \varphi(x, y+1, (y+1)z). \end{array}$$

15. Доказать, что $\sigma_1 = r$, где

$$\Sigma: \left\{ \begin{array}{l} \varphi_1(x) = (x=\varepsilon)?\ x : \varphi_2(x_h, \varphi_1(x_t)) \\ \varphi_2(x,y) = (y=\varepsilon)?\ x : y_h\varphi_2(x, y_t) \end{array} \right.$$

и r – функция инвертирования строк, определённая в примере 4 пункта 11.3.2.

16. Доказать, что $\sigma = r$, где

$$\Sigma: \left\{ \begin{array}{l} \varphi(x) = (x=\varepsilon)?\ \varepsilon \\ \qquad : (x'=\varepsilon)?\ x : (\varphi(x_t))_h\varphi(x_h\varphi(\varphi(x_t)_t)) \end{array} \right.$$

и r – функция инвертирования строк, определённая в примере 4 пункта 11.3.2.

17. Доказать, что $\forall\, x \quad \sigma(x) = \sigma'(x, NIL)$, где

$$\Sigma: \begin{cases} \varphi(x) = atom(x) & ?\ cons(x, NIL) \\ & :\ append(\varphi(car(x)), \varphi(cdr(x))) \end{cases},$$
$$\Sigma': \begin{cases} \varphi(x, y) = atom(x) & ?\ cons(x, y) \\ & :\ \varphi(car(x), \varphi(cdr(x), y)) \end{cases}$$

и $atom$, $cons$, car, cdr – функции языка LISP, NIL – константа, которой соответствует пустой список.

(Обе ФП определяют LISP-функции сплющивания дерева, например $(a(bc)(de))$ преобразуется в $(abcde)$.)

12.4 Свойства наименьших неподвижных точек

1. Доказать, что верна импликация

$$x \geq y \quad \Rightarrow \quad \sigma(x) \geq \sigma'(y) \geq 1,$$

где x, y – типа $\mathbf{N}$, и

$$\Sigma:\ \varphi(x) = (x = 0) \vee (x = 1)?\ 1 : \varphi(x-1) + \varphi(x-2),$$
$$\Sigma':\ \varphi(x) = (x = 0) \vee (x = 1)?\ 1 : 2\varphi(x-2).$$

2. Доказать, что верна импликация

$$x > y \quad \Rightarrow \quad \sigma(x, x) > \sigma(y, y),$$

где x, y – типа $\mathbf{N}$ и

$$\Sigma: \begin{cases} \varphi(x, y) = (x = 0) & ?\ y + 1 \\ & :\ (y = 0) \quad ?\ \varphi(x-1, 1) \\ & \qquad\quad :\ \varphi(x-1, \varphi(x, y-1)) \end{cases}.$$

3. Доказать, что верна импликация

$$x > 1 \quad \Rightarrow \quad \sigma(x) = \omega \quad \text{или} \quad \sigma(x) \leq \tfrac{x}{2},$$

где x – типа $\mathbf{I}$ и

- $\Sigma: \begin{cases} \varphi(x) = (x \leq 1) & ?\ 1 \\ & :\ even(x) \quad ?\ div(x, 2) \\ & \qquad\quad :\ \varphi\varphi(3div(x, 2) + 2) \end{cases},$

- $even(x) = 1$, если x – чётное, и 0, если x – нечётное,
- div – функция целочисленного деления.

4. Доказать, что

   (a) $\forall\, x \quad \sigma(x) = \sigma'(x, 0, 1)$,

   (b) $\forall\, x, y \quad \sigma''(x, y) \le \Big(\sigma(x) = \sigma'(x, y, \sigma(y))\Big)$,

   где x, y – типа $\mathbf{N}$ и

   $$\begin{array}{l}\Sigma:\ \varphi(x) = (x = 0)?\ 1 : x\varphi(x-1),\\ \Sigma':\varphi(x,y,z) = (x = y)?\ z : \varphi(x, y+1, (y+1)z),\\ \Sigma'':\varphi(x,y) = (x = y)?\ 1 : \varphi(x, y+1).\end{array}$$

5. Доказать, что $\forall\, x \quad \sigma(x) \ne \omega$, где x – типа $\mathbf{N}$ и

   $$\Sigma:\ \varphi(x) = even(x)?\ \frac{x}{2} : \varphi\varphi(3x+1)$$

   (использовать двоичное представление натуральных чисел).

6. Верно ли, что $\forall\, x > 0 \quad \sigma(x) \ne \omega$, где
   $\Sigma:\ \varphi(x) = (x = 1)?\ 0 : (even(x)?\ \varphi(\frac{x}{2}) : \varphi(3x+1))$?

7. Доказать, что $\forall\, x > 0 \quad \sigma(x) = \sigma'_1(x)$, где x – типа $\mathbf{N}$ и
   $$\Sigma:\ \left\{\begin{array}{ll}\varphi(x) = (x = 1) & ?\ 0\\ & :(even(x)?\ \varphi(\frac{x}{2}) : \varphi(3x+1))\end{array}\right.,$$
   $$\Sigma':\ \left\{\begin{array}{l}\varphi_1(x) = (x = 1)?\ 0 : \varphi_1\varphi_2(x)\\ \varphi_2(x) = (x = 1)?\ 1 : (even(x)?\ \frac{x}{2} : \varphi_2\varphi_2(\frac{3x+1}{2}))\end{array}\right..$$

8. Доказать, что $\sigma(x, h(y)) = h\sigma(x, y)$, где
   $\Sigma:\ \varphi(x, y) = p(x)?\ y : \varphi(k(x), h(y))$.

9. Доказать, что $\sigma_1(x, x) = \sigma_2(x)$, где
   $$\Sigma:\ \left\{\begin{array}{l}\varphi_1(x,y) = p(x)?\ \varphi_2(y) : \varphi_1(h(x), y)\\ \varphi_2(x) = p(x)?\ x : \varphi_2 h(x)\end{array}\right..$$

10. Доказать, что $\sigma_1\sigma_2 = \sigma_3\sigma_4$, где
    $$\Sigma:\ \left\{\begin{array}{l}\varphi_1(x) = q(x)?\ g(x) : \varphi_3\varphi_4 f(x)\\ \varphi_2(x) = p(x)?\ x : \varphi_2 f(x)\\ \varphi_3(x) = p(x)?\ g(x) : \varphi_1\varphi_2 f(x)\\ \varphi_4(x) = q(x)?\ x : \varphi_4 f(x)\end{array}\right..$$

11. Доказать, что $\sigma_1 = \sigma_4$, где

$$\Sigma: \begin{cases} \varphi_1(x) = p(x)?\ \begin{pmatrix} q(x) & ?\ \varphi_3\varphi_2 g(x) \\ & :\ \varphi_2\varphi_3 g(x) \end{pmatrix} : h(x) \\ \varphi_2(x) = q(x)?\ \varphi_1 f(x) : k(x) \\ \varphi_3(x) = q(x)?\ \varphi_4 f(x) : k(x) \\ \varphi_4(x) = p(x)?\ \varphi_5 g(x) : h(x) \\ \varphi_5(x) = q(x)?\ \varphi_2\varphi_4 f(x) : \varphi_3 k(x) \end{cases}.$$

Для этого необходимо сначала доказать, что

$$\sigma_5 = \sigma_3\sigma_2 = \sigma_2\sigma_3 \quad \text{и} \quad \sigma_3 = \sigma_2.$$

12. Доказать, что $\forall\, x, y \in D_{\mathbf{N}}$

    (a) $succ(\sigma(x, y)) = \sigma(x, succ(y))$,

    (b) $\sigma(\sigma(x, y), z) = \sigma(\sigma(x, z), y)$,

    (c) $\sigma(x, y) = \sigma(y, x)$,

    где

    - $succ$ – функция «следующее число»,
    - $pred$ – функция «предыдущее число»,
    - $\Sigma : \varphi(x, y) = (x = 0)?\ y : \varphi(pred(x), succ(y))$
      (σ является сложением натуральных чисел).

13. Доказать, что

$$\forall\, x \neq \varepsilon \quad \sigma(x, \sigma'(x)) = 1,$$

    где x – типа $\mathbf{S}$ и

    - $\Sigma :\ \varphi(x, y) = (x = \varepsilon)?\ 0 : ((x_h = y)?\ 1 : \varphi(x_t, y))$
      где y – типа $\mathbf{L}$,
    - $\Sigma' : \varphi(x) = (x = \varepsilon)?\ \omega : ((x_t = \varepsilon)?\ x_h : \varphi(x_t))$

    ($\sigma(x, y)$ = результат проверки вхождения символа y в строку x, и $\sigma'(x)$ = последний символ в строке x).

14. Доказать следующие свойства функции **sort** (определение которой см. в примере 5 пункта 11.3.2):

    (a) $\forall\, x \in D_{\mathbf{S}} \quad \mathbf{sort}(\mathbf{sort}(x)) = \mathbf{sort}(x)$;

    (b) $\forall\, x, y \in D_{\mathbf{S}} \quad \mathbf{sort}(x * y) = \mathbf{sort}(y * x)$,
    где $*$ – конкатенация строк, описываемая ФП (8.1);

(c) $\forall\, x \in D_{\mathbf{S}} \quad \mathbf{sort}(x * x) = \mathbf{dup}(\mathbf{sort}(x))$,
где функция $\mathbf{dup} : D_{\mathbf{S}} \to D_{\mathbf{S}}$ является ННТ ФП

$$\Sigma :\ \varphi(x) = (x = \varepsilon)?\ \varepsilon : x_h x_h \varphi(x_t);$$

(d) $\forall\, x \in D_{\mathbf{S}} \quad \mathbf{sort}(r(x)) = \mathbf{sort}(x)$,
где $r : D_{\mathbf{S}} \to D_{\mathbf{S}}$ – функция инвертирования строк, определённая в примере 4 пункта 11.3.2.

15. Написать ФП, определяющую функцию $\mathbf{equal} :\ D_{\mathbf{S}}^2 \to \{0, 1\}$, принимающую на паре строк $(x, y) \in D_{\mathbf{S}}^2$ значение 1, если каждый символ входит в строку x столько же раз, сколько раз он входит в строку y, и 0 иначе, и доказать следующие свойства:

    (a) $\forall\, x \in D_{\mathbf{S}} \quad \mathbf{equal}(x, \mathbf{sort}(x)) = 1$;

    (b) $\forall\, x \in D_{\mathbf{S}} \quad \mathbf{equal}(x, x) = 1$;

    (c) $\forall\, x, y \in D_{\mathbf{S}} \quad \mathbf{equal}(x, y) = \mathbf{equal}(y, x)$;

    (d) $\forall\, x, y, z \in D_{\mathbf{S}} \quad \mathbf{equal}(x, y)\mathbf{equal}(y, z) \leq \mathbf{equal}(x, z)$.

12.5 Исследовательские проблемы

1. Найти необходимое и достаточное условие, которому должны удовлетворять вычислительное правило C и ФП Σ, при выполнении которого функции C_Σ и σ совпадают.

2. Найти класс ФП, включающий все рассмотренные в этой книге ФП на символьных строках, такой, что проблема распознавания совпадения ННТ двух различных ФП из этого класса алгоритмически разрешима.

3. В пункте 9.2.6 был приведен пример решения задачи нахождения терма, вычисляющего ННТ ФП. Требуется решить более общую задачу: найти достаточно большие классы ФП, для которых разрешима проблема нахождения процессов, вычисляющих ННТ ФП из этих классов (понятие функции, вычисляемой процессом, аналогично понятию функции $\theta \mapsto P\theta$ для БС, см. пункт 2.4).

   Данная проблема имеет название проблемы **синтеза программ**.

Часть III

Model checking

Глава 13

Модели распределённых программ

В этой части мы будем использовать то понятие процесса, которое было определено в параграфе 4.1 (т.е. процессы, рассматриваемые в этой части, могут взаимодействовать только путем использования общих переменных). Под **распределённой программой (РП)** будем понимать конечную совокупность процессов. Рассматриваемые в этой части модели РП представляют собой диаграммы переходов между состояниями анализируемой РП. Излагаемый в этой части подход к верификации РП принято называть англоязычным словосочетанием **model checking**.

Верификация РП на основе метода model checking состоит из следующих этапов.

1. Построение **математической модели** анализируемой РП.
2. Представление проверяемых свойств анализируемой РП в виде формулы темпоральной логики, называемой **спецификацией**.
3. Построение **математического доказательства** наличия или отсутствия у модели РП проверяемых свойств, выраженных в спецификации.

13.1 Верификация распределённых программ

13.1.1 Математические модели распределённых программ

При верификации РП методом model checking **математическая модель РП** (называемая ниже **системой переходов**) представляет собой граф,

- вершины которого называются **состояниями** и изображают ситуации (или классы ситуаций), в которых может находиться эта РП в различные моменты времени, и

- рёбра которого соответствуют переходам, по которым могут происходить изменения состояний во время выполнения этой РП.

Одна и та же РП может быть представлена различными моделями, отражающими разные степени абстракции и разные уровни детализации действий, исполняемых этой РП. При построении модели РП следует руководствоваться следующими принципами:

- модель РП не должна быть чрезмерно детальной, т.к. излишняя сложность модели может вызвать существенные вычислительные проблемы при её формальном анализе;

- модель РП не должна быть чрезмерно упрощённой, она должна отражать те аспекты РП, которые имеют отношение к проверяемым свойствам, и сохранять все свойства моделируемой РП, представляющие интерес для анализа, т.к. в случае несоблюдения этого условия результаты верификации не будут иметь смысла.

13.1.2 Спецификация

Спецификация РП – это описание анализируемых свойств этой РП в виде формального текста, выраженное, например, в виде формулы математической логики.

Если свойство РП изначально было выражено на естественном языке, то при переводе его в спецификацию важно обеспечить соответствие между естественно-языковым описанием этого свойства и его спецификацией, т.к. в случае несоблюдения этого условия результаты верификации не будут иметь смысла.

Одним из инструментов формального описания свойств РП является **темпоральная логика**. Свойства РП выражаются в темпоральной логике **темпоральными формулами**.

Примеры свойств, выражаемых темпоральными формулами:

- РП при любом варианте своего функционирования не будет находиться ни в одном из состояний из заданного класса;

- РП при некотором функционировании когда-нибудь попадёт в некоторое состояние из заданного класса.

Как правило, при проведении рассуждений, связанных с анализом свойств РП, описываемых темпоральными формулами, рассматриваются не всевозможные темпоральные формулы, а только темпоральные формулы из некоторого ограниченного класса. Классы темпоральных формул принято называть **темпоральными логиками**, т.е. словосочетание «темпоральная логика» имеет два значения: в первом значении это язык, на котором можно выражать спецификации в виде темпоральных формул, а во втором – некоторый класс темпоральных формул.

В этом тексте будут рассмотрены следующие темпоральные логики: CTL (Computational Tree Logic), LTL (Linear Temporal Logic), PCTL (Probabilistic Computational Tree Logic).

13.1.3 Построение формальных доказательств

Если анализируемая модель РП удовлетворяет своей спецификации, то доказательство этого свойства методом model checking может быть построено либо путем рассмотрения всех возможных вариантов функционирования анализируемой РП, либо на основе дедуктивных рассуждений. Если же анализируемая модель РП не удовлетворяет своей спецификации, то для обоснования этого факта, как правило, достаточно построить хотя бы один вариант функционирования анализируемой модели (называемый **контрпримером**), при котором нарушается свойство, выражаемое спецификацией.

Главным достоинством метода model checking является возможность полностью автоматического анализа модели РП. Одним из главных недостатков model checking является высокая вычислительная сложность процедуры верификации на основе этого метода.

Среди учебной литературы по model checking в первую очередь следует назвать замечательную книгу Ю. Г. Карпова [30], а также книги [31], [32], [33]. Наиболее известной промышленной системой верификации, основанной на model checking, является система SPIN [34], [35].

13.2 Системы переходов

Ниже предполагается, что задано множество AP, элементы которого называются **атомарными утверждениями** (**atomic propositions**).

13.2.1 Понятие системы переходов

Системой переходов (СП) называется пятёрка $\Sigma = (S, R, L, S^0, \mathcal{F})$, компоненты которой имеют следующий смысл:

- S – конечное множество **состояний**,
- $R \subseteq S \times S$ – **отношение перехода**,
- L – функция (называемая **оценкой**) вида $L : S \times AP_\Sigma \to \{1, 0\}$, где $AP_\Sigma \subseteq AP$,
- $S^0 \subseteq S$ – множество **начальных состояний**,
- $\mathcal{F}$ – некоторая совокупность подмножеств множества S, называемых **условиями справедливости**.

Оценка L имеет следующий смысл: $\forall\, s \in S,\ \forall\, p \in AP_\Sigma$ утверждение p считается **истинным** в s, если $L(s, p) = 1$, и **ложным** в s иначе.

Выражение $L(s, p)$ будем записывать более компактно в виде p^s.

СП $\Sigma = (S, R, L, S^0, \mathcal{F})$ удобно рассматривать как граф (обозначаемый тем же символом Σ) с множеством вершин S. $\forall\, (s, s') \in R$ данный граф содержит ребро из s в s'.

Будем использовать следующие обозначения и соглашения:

- $\forall\, s \in S\ \ R(s) = \{s' \in S \mid (s, s') \in R\},\ \ R^{-1}(s) = \{s' \in S \mid (s', s) \in R\}$;
- если Σ – СП, компоненты которой не указаны, то будем обозначать её компоненты записями S_Σ, R_Σ, L_Σ, S^0_Σ, $\mathcal{F}_\Sigma$ соответственно;
- если СП Σ задана в виде тройки (S, R, L), то $S^0_\Sigma = S$, $\mathcal{F}_\Sigma = \{S\}$;
- если СП Σ задана в виде четверки (S, R, L, S^0), то $\mathcal{F}_\Sigma = \{S\}$.

13.2.2 Пути в системах переходов

Путь в СП Σ – это последовательность $\pi = (s_0, s_1, \ldots)$ состояний из S_Σ, такая, что $\forall\, i \geq 0 \quad s_{i+1} \in R_\Sigma(s_i)$. Если данная последовательность бесконечна, то путь π называется **бесконечным**, иначе путь π называется **конечным**. Если путь π конечный и имеет вид $(s_0, \ldots, s_n)$, то будем говорить, что π – путь из s_0 в s_n. Этот путь называется **циклом**, если $s_n = s_0$, и **пустым**, если $n = 0$.

Если $\pi = (s_0, \ldots)$ и $S \subseteq S_\Sigma$, то $\pi \subseteq S$ означает, что $\forall\, i \geq 0\ s_i \in S$.

Если пути π и π' имеют вид $(s_0, \ldots, s_n)$ и $(s_n, s_{n+1}, \ldots)$ соответственно, то запись $\pi\pi'$ обозначает путь $(s_0, \ldots, s_n, s_{n+1}, \ldots)$, называемый **конкатенацией** π и π'.

Если π – цикл, то π^∞ обозначает бесконечную конкатенацию $\pi\pi\pi\ldots$

Если π имеет вид $(s_0, s_1, \ldots)$, то будем говорить, что π – **путь из** s_0.

По умолчанию под путями будем подразумевать бесконечные пути, а если какой-либо из рассматриваемых путей является конечным, то это будет специально оговариваться.

Для каждого пути π в СП Σ запись $\inf(\pi)$ обозначает множество

$$\{s \in S_\Sigma \mid s \text{ имеет бесконечно много вхождений в } \pi\}.$$

Путь π в СП Σ называется **справедливым**, если

$$\forall\, F \in \mathcal{F}_\Sigma \quad \inf(\pi) \cap F \neq \emptyset.$$

$\forall\, s \in S_\Sigma$ Π_s обозначает совокупность всех справедливых путей из s в Σ, и Π_Σ обозначает совокупность всех справедливых путей в Σ.

Запись Π^0_Σ обозначает множество $\bigcup_{s \in S^0_\Sigma} \Pi_s$.

13.2.3 Построение системы переходов, соответствующей распределенной программе

Распределённой программой (РП) будем называть конечную совокупность процессов $P = (P_1, \ldots, P_k)$. Считаем, что $\forall\, i = 1, \ldots, k$ P_i имеет начальное состояние P_i^0, множество рёбер $Edges_i$, и предусловие Pre_i. Будем использовать следующие обозначения:

- $\forall\, i = 1, \ldots, k$ X_i – множество, состоящее из всех переменных, входящих в P_i, а также дополнительной переменной at_i, принимающей значения в множестве вершин процесса P_i;
- $X = \bigcup_{i=1}^k X_i$, $X' = \{x' \mid x \in X\} \subseteq Var$, где $X \cap X' = \emptyset$;
- $\forall\, (\theta, \theta') \in X^\bullet \times X^\bullet$ $\theta \sqcup \theta'$ – подстановка из $(X \sqcup X')^\bullet$, определяемая следующим образом:

$$\forall\, x \in X \;\; (\theta \sqcup \theta')(x) = \theta(x), \;\; (\theta \sqcup \theta')(x') = \theta'(x');$$

- $\forall\, Y \subseteq X$ запись $same(Y)$ обозначает формулу $\bigwedge_{x \in Y}(x' = x)$;
- $Edges = \bigcup_{i=1}^k Edges_i$;

- $\forall\, \gamma \in Edges$ запись R_γ обозначает формулу, определяемую следующим образом: пусть n и n' – начало и конец γ соответственно, тогда
  - если метка ребра γ имеет вид $x := e$, то
    $$R_\gamma = (at_i = n) \wedge (at_i' = n') \wedge (x' = e) \wedge same(X \setminus \{x, at_i\}),$$
  - если метка ребра γ имеет вид $\{\beta\}$, то
    $$R_\gamma = (at_i = n) \wedge (at_i' = n') \wedge \beta \wedge same(X \setminus \{at_i\});$$
- символ R обозначает формулу $\bigvee_{\gamma \in Edges} R_\gamma$;
- Pre обозначает формулу $\bigwedge_{i=1}^{k} \Big(Pre_i \wedge (at_i = P_i^0)\Big)$.

Для любых $\theta, \theta' \in X^\bullet$:

- $Pre^\theta = 1$ тогда и только тогда, когда θ является одной из возможных подстановок из $X^\bullet$ в начальный момент выполнения P и
- $R^{\theta \sqcup \theta'} = 1$ тогда и только тогда, когда верно утверждение:
  - если перед выполнением какого-либо действия РП P каждая переменная $x \in X$ имела значение $\theta(x)$,
  - то после выполнения этого действия каждая переменная $x \in X$ будет иметь значение $\theta'(x)$.

СП Σ, соответствущая РП P, имеет следующие компоненты.

- $S_\Sigma \stackrel{\text{def}}{=} X^\bullet$, $S_\Sigma^0 \stackrel{\text{def}}{=} \{\theta \in X^\bullet \mid Pre^\theta = 1\}$.
- $R_\Sigma \stackrel{\text{def}}{=} \{(\theta, \theta') \in X^\bullet \times X^\bullet \mid R^{\theta \sqcup \theta'} = 1\}$.

  Из определения R_Σ и приведенного выше замечания следует, что каждому выполнению P соответствует некоторый путь в Σ.
- $AP_\Sigma \stackrel{\text{def}}{=} Fm(X)$, $\forall\, \theta \in S_\Sigma$, $\forall\, \beta \in Fm(X)$ $\;L_\Sigma(\theta, \beta) \stackrel{\text{def}}{=} \beta^\theta$.
- Выбор множества $\mathcal{F}_\Sigma$ условий справедливости индивидуален для каждой РП. На реальные выполнения РП P могут быть наложены дополнительные условия, и $\mathcal{F}_\Sigma$ должно быть выбрано с таким расчетом, чтобы множество Π_Σ^0 не содержало путей, не соответствующих никакому реальному выполнению (т.е. не удовлетворяющему этим дополнительным условиям) РП P.

  Такие дополнительные условия могут иметь следующий вид.

– Не должно быть таких выполнений P, в которых один из входящих в неё процессов не совершает никаких действий после некоторого момента времени.

– Один из процессов в P (обозначим его P_i) может обращаться с запросами к другому процессу (обозначим его P_j), и процесс P_j может посылать процессу P_i ответы на эти запросы.

  Не должно быть таких бесконечных выполнений P, в которых

  * одно из действий представляет собой запрос от P_i к P_j и
  * все последующие действия P_j не являются посылкой ответа от P_j к P_i на этот запрос.

– Один из процессов в P может посылать сообщения другому процессу, причём сообщения при пересылке могут пропадать.

  Не должно быть таких выполнений P, в которых один из процессов в P бесконечно много раз посылает сообщения другому процессу в P и все эти сообщения пропадают.

Глава 14

Model checking на основе CTL

В этой главе изучается темпоральная логика CTL (Computational Tree Logic), используемая для формальной спецификации свойств РП.

Основными объектами всех темпоральных логик являются **темпоральные формулы** (называемые также просто **формулами**).

Будем считать, что для каждой из рассматриваемых в этом тексте темпоральных логик TL верно следующее:

- каждое утверждение из множества AP атомарных утверждений, введённого в пункте 13.2, является формулой логики TL и
- TL замкнута относительно **булевых комбинаций**, т.е. среди формул логики TL есть константы $\top$ и $\bot$, и если TL содержит формулы B и C, то TL также содержит формулы $\bar{B},\ B \wedge C,\ B \vee C$.

Запись вида $B \to C$, где $B, C \in$ TL, обозначает формулу $\bar{B} \vee C$.

14.1 Темпоральная логика CTL

14.1.1 Формулы темпоральной логики CTL

Совокупность **формул** темпоральной логики CTL, называемых также **CTL-формулами**, обозначается записью Fm_{CTL}. Как было отмечено выше, Fm_{CTL} содержит AP и замкнуто относительно булевых комбинаций. Кроме того, если Fm_{CTL} содержит формулы B и C, то Fm_{CTL} содержит также формулы

$$\mathbf{AX}B,\ \mathbf{EX}B,\ \mathbf{AF}B,\ \mathbf{EF}B,\ \mathbf{AG}B,\ \mathbf{EG}B,\ \mathbf{AU}(B,C),\ \mathbf{EU}(B,C).$$

Жирные символы (**AX** и т.д.) в данных формулах называются **темпоральными операторами**.

14.1.2 Значения CTL-формул

Пусть задана СП Σ.

$\forall\, s \in S_\Sigma, \forall\, B \in Fm_{CTL}$ **значение** $B^s \in \{1,0\}$ формулы B в состоянии s определяется индуктивно:

- если $B = p \in AP$, то $B^s = \begin{cases} L_\Sigma(s,p), & \text{если } p \in AP_\Sigma, \\ 0, & \text{если } p \notin AP_\Sigma, \end{cases}$
- значения булевых комбинаций определяются стандартно:

$$\top^s = 1,\ \ \bot^s = 0,$$
$$(\bar{B})^s = \overline{B^s},\ \ (B \wedge C)^s = B^s \wedge C^s \text{ и т.д.}$$

- значения CTL-формул, начинающихся с темпорального оператора, определяются следующим образом:
  - $(\mathbf{AX}B)^s = [\![\, \forall\, s' \in R(s)\ \ B^{s'} = 1 \,]\!]$,
  - $(\mathbf{EX}B)^s = [\![\, \exists\, s' \in R(s) : B^{s'} = 1 \,]\!]$,
  - $(\mathbf{AF}B)^s = [\![\, \forall\, \pi = (s_0, \ldots) \in \Pi_s\ \exists\, i \geq 0 : B^{s_i} = 1 \,]\!]$,
  - $(\mathbf{EF}B)^s = [\![\, \exists\, \pi = (s_0, \ldots) \in \Pi_s,\ \exists\, i \geq 0 : B^{s_i} = 1 \,]\!]$,
  - $(\mathbf{AG}B)^s = [\![\, \forall\, \pi = (s_0, \ldots) \in \Pi_s\ \forall\, i \geq 0 : B^{s_i} = 1 \,]\!]$,
  - $(\mathbf{EG}B)^s = [\![\, \exists\, \pi = (s_0, \ldots) \in \Pi_s :\ \forall\, i \geq 0 : B^{s_i} = 1 \,]\!]$,
  - $(\mathbf{AU}(B,C))^s = [\![\, \forall\, \pi = (s_0, \ldots) \in \Pi_s\ \exists\, i \geq 0 : C^{s_i} = 1, \forall\, j < i \quad B^{s_j} = 1 \,]\!]$,
  - $(\mathbf{EU}(B,C))^s = [\![\, \exists\, \pi = (s_0, \ldots) \in \Pi_s, \exists\, i \geq 0 : C^{s_i} = 1, \forall\, j < i \quad B^{s_j} = 1 \,]\!]$.

Значением CTL-формулы B в СП Σ называется множество

$$B^{S_\Sigma} \stackrel{\text{def}}{=} \{s \in S_\Sigma \mid B^s = 1\}.$$

14.1.3 Эквивалентность CTL-формул

Будем называть CTL-формулы B и C **эквивалентными**, если для каждой СП Σ верно равенство $B^{S_\Sigma} = C^{S_\Sigma}$. Если CTL-формулы B и C эквивалентны, то будем обозначать этот факт записью $B = C$.

Нетрудно доказать, что верны равенства:

- законы де Моргана: $\overline{B \wedge C} = \bar{B} \vee \bar{C}$, $\overline{B \vee C} = \bar{B} \wedge \bar{C}$, $\overline{\overline{B}} = B$,
- $\overline{\mathbf{AX}B} = \mathbf{EX}\bar{B}$,

- $\mathbf{EF}B = \mathbf{EU}(\top, B)$, $\overline{\mathbf{AF}B} = \mathbf{EG}\bar{B}$, $\overline{\mathbf{AG}B} = \mathbf{EF}\bar{B}$,
- $\overline{\mathbf{AU}(B,C)} = \mathbf{EU}(\bar{C}, \bar{B} \wedge \bar{C}) \vee \mathbf{EG}\bar{C}$.

Данные равенства доказываются легко, за исключением последнего.

Докажем последнее равенство. Данное равенство эквивалентно следующему утверждению: для любой СП Σ и любого $s \in S_\Sigma$

$$\Big(\overline{\mathbf{AU}(B,C)} \Big)^s = 0 \;\Leftrightarrow\; \Big(\mathbf{EU}(\bar{C}, \bar{B} \wedge \bar{C}) \vee \mathbf{EG}\bar{C}\Big)^s = 0. \tag{14.1}$$

Левая часть (14.1) равносильна равенству $(\mathbf{AU}(B,C))^s = 1$, которое равносильно следующему утверждению: $\forall\, \pi = (s_0, \ldots) \in \Pi_s$

$$\exists\, i \geq 0 :\ C^{s_i} = 1,\ \forall\, j < i\ \ (B^{s_j} = 1) \wedge (C^{s_j} = 0). \tag{14.2}$$

Правая часть (14.1) равносильна конъюнкции равенств

$$(\mathbf{EU}(\bar{C}, \bar{B} \wedge \bar{C}))^s = 0 \quad \text{и} \quad (\mathbf{EG}\bar{C})^s = 0,$$

которая равносильна конъюнкции утверждений

$$\neg \left(\begin{array}{l} \exists\, \pi = (s_0, \ldots) \in \Pi_s : \exists\, i \geq 0 : (\bar{B} \wedge \bar{C})^{s_i} = 1, \\ \forall\, j < i\ (\bar{C})^{s_j} = 1,\ (\bar{B} \wedge \bar{C})^{s_j} = 0 \end{array} \right)$$
$$\neg\Big(\exists\, \pi = (s_0, \ldots) \in \Pi_s : \forall\, i \geq 0 : C^{s_i} = 0\Big),$$

что равносильно следующему утверждению: $\forall\, \pi = (s_0, \ldots) \in \Pi_s$ верна конъюнкция

$$\left\{ \begin{array}{l} \neg \left(\begin{array}{l} \exists\, i \geq 0 : B^{s_i} = 0, C^{s_i} = 0, \\ \forall\, j < i\ C^{s_j} = 0,\ B^{s_j} = 1 \end{array} \right) \\ \neg\Big(\forall\, i \geq 0 : C^{s_i} = 0\Big) \end{array} \right\},$$

которую можно переписать следующим образом:

$$\left\{ \begin{array}{l} \forall\, i \geq 0 \quad \Big((B^{s_i} = 0) \wedge (C^{s_i} = 0)\Big) \rightarrow \\ \qquad\qquad \rightarrow\ \exists\, j < i : \Big(B^{s_j} = 0 \text{ или } C^{s_j} = 1\Big) \\ \exists\, i \geq 0 : C^{s_i} = 1 \end{array} \right\}. \tag{14.3}$$

Докажем, что утверждения (14.2) и (14.3) равносильны.

- Пусть верно (14.2), тогда верен второй конъюнктивный член в (14.3).

  Докажем, что первый член в (14.3) тоже будет верен. Обозначим символом A импликацию в этом члене и записью i_0 – тот номер, существование которого утверждается в (14.2). Если $i \leq i_0$, то A верна, потому что ложна её посылка. Если $i > i_0$, то A верна, т.к. верно её заключение: в качестве j можно взять i_0.

- Пусть верно (14.3). Докажем, что тогда будет верно (14.2). Пусть i_0 – наименьший из номеров i, таких, что $C^{s_i} = 1$ (такой номер существует согласно второму конъюнктивному члену в (14.3)).

  Докажем от противного, что $\forall\, j < i_0\ B^{s_j} = 1$. Предположим, что $\exists\, j < i_0 : B^{s_j} = 0$, и пусть j_0 – наименьший из таких j. Согласно выбору i_0 и j_0, имеем свойство $B^{s_{j_0}} = 0$ и $C^{s_{j_0}} = 0$. Согласно импликации в первом конъюнктивном члене в (14.3) для $i = j_0$, получаем: $\exists\, j < j_0 : \Big(B^{s_j} = 0$ или $C^{s_j} = 1\Big)$. Однако равенство $C^{s_j} = 1$ невозможно в силу выбора i_0, поэтому $B^{s_j} = 0$, что тоже невозможно в силу выбора j_0. ■

Из приведенных выше соотношений следует, что для любой CTL-формулы существует эквивалентная ей CTL-формула, в которую входят только следующие CTL-операторы: **EX**, **EG**, **EU**.

14.1.4 Примеры свойств распределённых программ, выражаемых CTL-формулами

Пусть AP содержит утверждения Start, Ready, Request, Acknowledgement, DeviceEnabled, Restart. Используя данные утверждения, можно написать следующие CTL-формулы, выражающие определенные свойства РП (после каждой из приводимых ниже CTL-формул мы неформально поясняем ее смысл):

- **EF** (Start $\wedge$ $\overline{\text{Ready}}$)
  (возможно, что будет достижимо состояние, в котором свойство Start выполняется, а условие Ready – не выполняется);

- **AG** (Request $\rightarrow$ **AF** Acknowledgement)
  (если получен запрос, то когда-нибудь на него будет дан ответ);

- **AG** (**AF** DeviceEnabled)
  (при любом функционировании РП условие DeviceEnabled выполнено бесконечно много раз);

- **AG** (**EF** Restart)
  (из каждого состояния возможно будет достижимо состояние, в котором выполняется свойство Restart).

14.2 Задача model checking для CTL

Одна из возможных форм задачи **model checking** для CTL (которую мы будем ниже обозначать знакосочетанием **MC-CTL**) заключается в том, чтобы по заданным СП $\Sigma = (S, R, L, S^0, \mathcal{F})$ и CTL-формуле A вычислить множество A^S. Можно считать, что формула A содержит только CTL-операторы вида **EX**, **EG**, **EU**.

Для вычисления A^S можно использовать излагаемый ниже рекурсивный алгоритм, сводящий вычисление множества A^S к вычислению множеств вида B^S, где B – подформула формулы A.

- Если $A = p \in AP$, то $A^S = \begin{cases} \{s \in S \mid L(s,p) = 1\}, & \text{если } p \in AP_\Sigma, \\ \emptyset, & \text{если } p \notin AP_\Sigma. \end{cases}$

- Если A – булева комбинация, то A^S вычисляется так:

  $$\top^S = S, \;\; \bot^S = \emptyset,$$
  $$(\bar{B})^S = S \setminus B^S, \;\; (B \wedge C)^S = B^S \cap C^S \text{ и т.д.}$$

- Если $A = \mathbf{EX}B$, то $A^S = R^{-1}(B^S) = \{s \in S \mid R(s) \cap B^S \neq \emptyset\}$.

- Если $A = \mathbf{EU}(B, C)$, то A^S вычисляется следующим алгоритмом:
  $A^S := S' := C^S \cap (\mathbf{EG}\top)^S$,
  while $(S' \neq \emptyset)$
  { выбираем $s \in S'$ и удаляем s из S',
  $\forall\, s' \in R^{-1}(s)$, **если** $\left\{ \begin{array}{l} s' \in B^S \\ s' \notin A^S \end{array} \right\}$, **то** добавляем s' к A^S и к S'
  }

- Пусть $A = \mathbf{EG}B$.

  Согласно определению, равенство $(\mathbf{EG}B)^s = 1$ означает, что

  $$\exists\, \pi = (s_0, \ldots) \in \Pi_s : \; \pi \subseteq B^S. \tag{14.4}$$

  Поскольку $|S| < \infty$, то $\exists\, i \geq 0 : \inf(\pi) = \{s_j \mid j \geq i\}$.

  Будем обозначать записью B^S не только множество состояний, но также и подграф графа Σ, вершинами которого являются состояния из множества B^S, и рёбрами – рёбра графа Σ, начало и конец которых лежат в B^S. Нетрудно видеть, что

  – $\inf(\pi)$ – сильно связный подграф графа B^S (т.е. такой подграф, что для каждой пары s, s' его вершин существует непустой путь из s в s'), причём $\forall\, F \in \mathcal{F} \;\; \inf(\pi) \cap F \neq \emptyset$, и

– этот подграф содержится в некотором максимальном по включению сильно связном подграфе $\mathbf{C}$ графа B^S (такие подграфы называются **сильно связными компонентами (strongly connected components, SCC**), причем

$$\forall F \in \mathcal{F} \quad \mathbf{C} \cap F \neq \emptyset. \tag{14.5}$$

Таким образом, из (14.4) следует, что в графе B^S существуют

– SCC $\mathbf{C}$, такая, что выполнено (14.5), и

– конечный путь $\hat{\pi}$ из s в некоторое состояние $s' \in \mathbf{C}$.

Верно и обратное – из последнего утверждения следует (14.4): в качестве искомого пути π можно взять путь $\hat{\pi}\tilde{\pi}^\infty$, где $\tilde{\pi}$ – цикл из s' в s', проходящий через каждую вершину $\mathbf{C}$.

Таким образом, возможен следующий алгоритм вычисления A^S:

– находим в B^S все SCC (можно использовать **алгоритм Тарьяна** нахождения всех SCC в произвольном графе G, его сложность равна $O(|G|)$, где $|G|$ – число вершин и рёбер в G);

– оставляем только те из найденных SCC, которые удовлетворяют условию (14.5);

– полагаем $A^S := S' :=$ множеству всех состояний, входящих в эти оставшиеся SCC, и затем работает цикл, аналогичный циклу в предыдущем пункте.

Нетрудно доказать, что сложность данного алгоритма – $O(|A|\,|\Sigma|\,|\mathcal{F}|)$, где $|A|$ – размер формулы A, $|\Sigma|$ – число вершин и рёбер в Σ, $|\mathcal{F}|$ – число условий справедливости в Σ.

14.3 MC-CTL на основе понятия неподвижной точки

14.3.1 Неподвижные точки монотонных операторов

Пусть S – конечное множество и $\mathbf{2}^S$ – множество всех подмножеств S.

Будем называть **оператором** на $\mathbf{2}^S$ отображение вида $f : \mathbf{2}^S \to \mathbf{2}^S$.

Оператор f на $\mathbf{2}^S$ называется **монотонным**, если

$$\forall A, B \subseteq S \quad A \subseteq B \Rightarrow f(A) \subseteq f(B).$$

Множество $A \subseteq S$ называется **неподвижной точкой (fixpoint, FP)** оператора f, если верно равенство $A = f(A)$.

FP оператора f на $\mathbf{2}^S$ называется **наименьшей** (**наибольшей**) FP и обозначается $\mu x.f(x)$ ($\nu x.f(x)$), если она – наименьшая (наибольшая) по включению FP f соответственно.

Теорема 21

Если S – конечное множество и f – монотонный оператор на $\mathbf{2}^S$, то существуют подмножества $A, B \subseteq S$, являющиеся наименьшей и наибольшей FP f соответственно.

Доказательство.

Определим последовательности $A_0, A_1, \ldots$ и $B_0, B_1, \ldots$ подмножеств множества S следующим образом:

$$A_0 = \emptyset,\ B_0 = S,\ \forall\, i \geq 0\ \ A_{i+1} = f(A_i),\ B_{i+1} = f(B_i).$$

Поскольку $A_0 = \emptyset \subseteq A_1$, то из монотонности f следует, что $f(A_0) \subseteq f(A_1)$, т.е. $A_1 \subseteq A_2$. Аналогично получаем: $A_2 = f(A_1) \subseteq f(A_2) = A_3$ и т.д., т.е. последовательность $A_0, A_1, \ldots$ – неубывающая цепь: $A_0 \subseteq A_1 \subseteq \ldots$.

Аналогично из $B_0 = S \supseteq B_1$ и монотонности f следует $f(B_0) \supseteq f(B_1)$, т.е. $B_1 \supseteq B_2$ и т.д., т.е. $B_0, B_1, \ldots$ – невозрастающая цепь: $B_0 \supseteq B_1 \supseteq \ldots$.

Поскольку $\forall\, i \geq 0\ \ A_i, B_i \subseteq S$ и множество S конечно, то

$$\exists\, i_0, j_0 \geq 0 :\ \ A_{i_0} = A_{i_0+1},\ \ B_{j_0} = B_{j_0+1},$$

или $A_{i_0} = f(A_{i_0})$ и $B_{j_0} = f(B_{j_0})$, т.е. A_{i_0} и B_{j_0} – FP f.

Докажем, что A_{i_0} и B_{j_0} – наименьшая и наибольшая FP f соответственно. Пусть C – произвольная FP f, т.е. $f(C) = C$. Из включений

$$A_0 = \emptyset \subseteq C \subseteq S = B_0$$

и монотонности f следует, что $f(A_0) \subseteq f(C) \subseteq f(B_0)$, т.е. $A_1 \subseteq C \subseteq B_1$.

Аналогично получаем $A_2 \subseteq C \subseteq B_2$ и т.д., т.е. $\forall\, i \geq 0\ \ A_i \subseteq C \subseteq B_i$. В частности, $A_{i_0} \subseteq C \subseteq B_{j_0}$. ■

14.3.2 Вычисление множеств $(\mathbf{EU}(B, C))^S$ и $(\mathbf{EG}B)^S$ на основе понятия неподвижной точки

Пусть задана СП $\Sigma = (S, R, \ldots)$.

Нетрудно доказать, что следующие операторы монотонны:

- операторы $A\cap$ и $A\cup : \mathbf{2}^S \to \mathbf{2}^S$ (где $A \subseteq S$ – фиксированное подмножество), сопоставляющие каждому $x \subseteq S$ множества $A \cap x$ и $A \cup x$ соответственно, и

- оператор $\mathbf{EX} : \mathbf{2}^S \to \mathbf{2}^S$, сопоставляющий каждому $x \subseteq S$ множество $R^{-1}(x) = \{s \in S \mid R(s) \cap x \neq \emptyset\}$.

Поскольку композиция монотонных операторов является монотонным оператором, то, в частности, операторы $f_1, f_2 : \mathbf{2}^S \to \mathbf{2}^S$, такие, что

$$\forall\, x \subseteq S \quad f_1(x) = A \cup (B \cap \mathbf{EX}(x)), \quad f_2(x) = B \cap \mathbf{EX}(x)$$

(где $A, B \subseteq S$ – фиксированные множества), являются монотонными.

Излагаемая ниже теорема описывает свойства этих операторов.

Теорема 22

Пусть задана СП $\Sigma = (S, R, L)$, причем $\forall\, s \in S \;\; R(s) \neq \emptyset$.

Тогда $\forall\, B, C \in Fm_{CTL}$

$$\begin{aligned} (\mathbf{EU}(B,C))^S &= \mu x.\Big(C^S \cup (B^S \cap \mathbf{EX}(x))\Big), \\ (\mathbf{EG}B)^S &= \nu x.\Big(B^S \cap \mathbf{EX}(x)\Big). \end{aligned} \tag{14.6}$$

Доказательство.

Пусть f_1 и f_2 – операторы, соответствующие правым частям равенств в (14.6), и $A_0, A_1, \ldots$ и $B_0, B_1, \ldots$ – последовательности подмножеств S, такие, что $A_0 = \emptyset$, $B_0 = S$, и $\forall\, i \geq 0$ $A_{i+1} = f_1(A_i)$ и $B_{i+1} = f_2(B_i)$.

Из данных определений следует, что $\forall\, i \geq 0$

- $A_{i+1} = \{s \in S \mid \exists\, j \leq i, \exists$ путь $(s = s_0, \ldots, s_j) :$ $\{s_0, \ldots, s_{j-1}\} \subseteq B^S, s_j \in C^S\}$,

- $B_{i+1} = \{s \in S \mid \exists$ путь $(s = s_0, \ldots, s_i) \subseteq B^S\}$.

Поэтому правые части равенств в (14.6) имеют вид

$$\bigcup_{i\geq 0} A_i = \{s \in S \mid \exists\, i \geq 0 : \exists\, j \leq i, \exists \text{ путь } (s = s_0, \ldots, s_j) : \{s_0, \ldots, s_{j-1}\} \subseteq B^S, s_j \in C^S\}, \tag{14.7}$$

$$\bigcap_{i\geq 0} B_i = \{s \in S \mid \forall\, i \geq 0\ \exists \text{ путь } (s = s_0, \ldots, s_i) \subseteq B^S\}. \tag{14.8}$$

Принимая во внимание условие $\forall\, s \in S \;\; R(s) \neq \emptyset$, заключаем, что (14.7) $= (\mathbf{EU}(B,C))^S$.

Докажем, что (14.8) $= (\mathbf{EG}B)^S$.

- Пусть $s \in (\mathbf{EG}B)^S$, тогда $\exists$ бесконечный путь $(s = s_0, \ldots) \subseteq B^S$, поэтому $s \in (14.8)$.
- Пусть $s \in (14.8)$, тогда $\exists \pi = (s = s_0, \ldots, s_i) \subseteq B^S$, где $i > |S|$.

  Пусть s_j – состояние, входящее в π более одного раза. Обозначим

  – записью $\hat{\pi}$ префикс $(s_0, \ldots, s_j)$ пути π и
  – записью $\tilde{\pi}$ – цикл из s_j в s_j, содержащийся в π.

  Имеем: $\hat{\pi}\tilde{\pi}^\infty$ – бесконечный путь, все состояния которого $\in B^S$, поэтому $s \in (\mathbf{EG}B)^S$. ■

Если множество $\mathcal{F}_\Sigma$ нетривиально, то процедура вычисления множеств $(\mathbf{EU}(B, C))^S$ и $(\mathbf{EG}B)^S$ имеет более сложный вид.

Ниже $\forall\, B, C \subseteq S$ запись $\mathbf{EU}(B, C)$ обозначает множество

$$\mu x.(C \cup (B \cap \mathbf{EX}(x))).$$

Нетрудно доказать, что оператор

$$f : \mathbf{2}^S \to \mathbf{2}^S, \quad \forall\, x \subseteq S \;\; f(x) = \mathbf{EU}(B, x),$$

где $B \subseteq S$ – фиксированное подмножество, является монотонным.

Теорема 23

Пусть задана СП $\Sigma = (S, \ldots)$, причём $\mathcal{F}_\Sigma = \{F_1, \ldots, F_k\}$.
Тогда $\forall\, B \in Fm_{CTL} \quad (\mathbf{EG}B)^S = \nu x.f(x)$, где

$$f(x) = B^S \cap \textstyle\bigcap_{i=1}^{k} \mathbf{EX}\, \mathbf{EU}(B^S, F_i \cap x).$$

Доказательство.

По определению, $(\mathbf{EG}B)^S = \{s \in S \mid \exists \pi \in \Pi_s : \pi \subseteq B^S\}$.
Включение $(\mathbf{EG}B)^S \subseteq \nu x.f(x)$ следует из того, что $(\mathbf{EG}B)^S$ – FP f:

$$(\mathbf{EG}B)^S = f((\mathbf{EG}B)^S).$$

Включение $\nu x.f(x) \subseteq (\mathbf{EG}B)^S$ следует из импликации

$$x = f(x) \;\Rightarrow\; x \subseteq (\mathbf{EG}B)^S. \blacksquare$$

Если для какого-либо $s \in S$ есть предположение, что $(\mathbf{EG}B)^s = 1$, и для обоснования этого требуется построить $\pi \in \Pi_s : \pi \subseteq B^S$, то это можно делать в процессе вычисления соответствующих неподвижных точек (не дожидаясь окончания их построения) следующим образом:

- вычисляя $\mathbf{EU}(B^S, F_1 \cap x)$, строим $\pi_1 \subseteq B^S$ из s в $s_1 \in F_1 \cap x$,
- так же строим $\pi_2 \subseteq B^S$ из s_1 в $s_2 \in F_2 \cap x$ и т.д.

Используя $\pi_1, \pi_2, \ldots$, строим искомый путь π в виде $\hat{\pi}\tilde{\pi}^\infty$.

14.3.3 Алгоритм решения задачи MC-CTL на основе понятия неподвижной точки

Пусть заданы СП $\Sigma = (S, R, L)$ и формула $A \in Fm_{CTL}$.

Используя изложенные выше методы, можно построить рекурсивный алгоритм вычисления множества A^S. Данный алгоритм отличается от алгоритма из пункта 14.2 лишь в тех пунктах, которые связаны с вычислением множеств вида $(\mathbf{EU}(B, C))^S$ и $(\mathbf{EG}B)^S$. Для вычисления этих множеств можно использовать соотношения (14.6).

14.4 μ-исчисление

CTL можно вложить в более выразительную логику, называемую **μ-исчислением**. Это позволяет свести задачу MC-CTL к некоторой задаче μ-исчисления.

μ-исчисление позволяет описывать свойства СП с несколькими отношениями перехода, в которых вместо одного отношения перехода R_Σ имеется совокупность $\{R^a \subseteq S_\Sigma \times S_\Sigma \mid a \in T\}$, где T – заданное множество, элементы которого называются **переходами**.

14.4.1 μ-формулы

Множество Fm_μ формул μ-исчисления (называемых **μ-формулами**) содержит AP, замкнуто относительно булевых комбинаций и также

- содержит множество RV, элементы которого называются **реляционными переменными (РП)**,
- $\forall\, a \in T,\ \forall\, A \in Fm_\mu \quad [a]A \in Fm_\mu$ и $\langle a\rangle A \in Fm_\mu$,
- $\forall\, x \in RV,\ \forall\, A \in Fm_\mu \quad \mu x.A \in Fm_\mu$ и $\nu x.A \in Fm_\mu$.

Вхождения РП в μ-формулы бывают **свободными** и **связанными**:

- $\forall\, x \in RV$ вхождение x в μ-формулу x – свободное,
- если A имеет вид $\bar{B}$, $B \wedge C$, $B \vee C$, $[a]B$, $\langle a\rangle B$, то каждому вхождению каждой РП x в A соответствует некоторое вхождение x в B или C и каждое вхождение x в A имеет тот же статус (свободное или связанное), который имеет соответствующее вхождение x в B или C;
- если A имеет вид $\mu x.B$ или $\nu x.B$, то

– все свободные вхождения x в B, а также вхождение x рядом с μ или ν становятся связанными в A и образуют (вместе с первым вхождением x в A) одну **группу связанности** и

– все остальные вхождения РП в A имеют тот же статус, который имеют соответствующие им вхождения этих РП в B.

Для каждой μ-формулы A

- $fv(A)$ обозначает совокупность всех РП, имеющих свободные вхождения в A, и
- $Sub(A)$ обозначает совокупность всех подформул формулы A.

μ-формула называется **правильной**, если для каждой её подформулы вида $\mu x.B$ или $\nu x.B$ число отрицаний в B, располагающихся над каждым свободным вхождением x в B, является чётным. Ниже каждая рассматриваемая μ-формула предполагается правильной.

14.4.2 Значения μ-формул

Пусть задана СП $\Sigma = (S, \{R^a \mid a \in T\}, L, S^0)$.

Подстановкой РП в СП Σ называется отображение $\zeta : RV \to \mathbf{2}^S$. Множество всех подстановок РП в СП Σ обозначается записью $RV_\Sigma^\bullet$.

$RV_\Sigma^\bullet$ можно рассматривать как частично упорядоченное множество:

$$\forall\, \zeta, \zeta' \in RV_\Sigma^\bullet \quad \zeta \le \zeta' \;\Leftrightarrow\; \forall\, x \in RV \quad \zeta(x) \subseteq \zeta'(x). \tag{14.9}$$

Значением μ-формулы A в СП Σ на подстановке $\zeta \in RV_\Sigma^\bullet$ называется подмножество $A^\zeta \subseteq S$, определяемое рекурсивно следующим образом:

- $\forall\, p \in AP \;\; p^\zeta = p^S = \begin{cases} \{s \in S \mid L(s,p) = 1\}, & \text{если } p \in AP_\Sigma, \\ \emptyset, & \text{если } p \notin AP_\Sigma, \end{cases}$
- $\forall\, x \in RV \;\; x^\zeta = \zeta(x)$,
- $\top^\zeta = S$, $\bot^\zeta = \emptyset$, $\bar{B}^\zeta = S \setminus B^\zeta$, $(B \wedge C)^\zeta = B^\zeta \cap C^\zeta$ и т.д.,
- $([a]B)^\zeta = \{s \in S \mid R^a(s) \subseteq B^\zeta\}$,
- $(\langle a \rangle B)^\zeta = (R^a)^{-1}(B^\zeta) = \{s \in S \mid R^a(s) \cap B^\zeta \neq \emptyset\}$,
- $(\mu x.B)^\zeta = \mu x.(y \mapsto B^{(y/x)\zeta})$, $(\nu x.B)^\zeta = \nu x.(y \mapsto B^{(y/x)\zeta})$, где

$$y \mapsto B^{(y/x)\zeta} \tag{14.10}$$

обозначает оператор, сопоставляющий каждому $S' \in \mathbf{2}^S$ множество $B^{(S'/x)\zeta} \in \mathbf{2}^S$ (монотонность (14.10) будет доказана в теореме 24), и

$$x^{(S'/x)\zeta} = S', \;\; \forall\, y \in RV \setminus \{x\} \;\; y^{(S'/x)\zeta} = y^\zeta.$$

Заметим, что A^ζ зависит только от значений ζ на РП из $fv(A)$.

Будем обозначать записью Σ_A функцию вида $RV_\Sigma^\bullet \to \mathbf{2}^S$, сопоставляющую каждой подстановке $\zeta \in RV_\Sigma^\bullet$ значение A^ζ.

Будем называть μ-формулы A и B **эквивалентными**, если для каждой СП Σ фунции Σ_A и Σ_B совпадают. Если μ-формулы A и B эквивалентны, то будем обозначать этот факт записью $A = B$.

Нетрудно доказать, что

$$\forall\, A \in Fm_\mu,\ \forall\, a \in T \quad \overline{[a]A} = \langle a\rangle \bar{A},\;\; \overline{\langle a\rangle A} = [a]\bar{A}. \tag{14.11}$$

Теорема 24

Пусть формула $A \in Fm_\mu$ такова, что $\forall\, x \in RV$ число отрицаний в A, располагающихся над каждым свободным вхождением x в A, чётное.

Тогда для каждой СП Σ функция Σ_A – монотонная.

Доказательство.

Обозначим записью $\iota(A)$ число вхождений в A связок $\wedge$, $\vee$, $[a]$, $\langle a\rangle$, μx, νx. Будем доказывать теорему индукцией по $\iota(A)$.

- Если A имеет вид $\top$, $\bot$, p, или x (где $p \in AP$, $x \in RV$), то монотонность Σ_A тривиальна.

- Пусть $A = B \wedge C$ или $B \vee C$.

  B и C удовлетворяют условию теоремы, и $\iota(B) < \iota(A)$, $\iota(C) < \iota(A)$, поэтому по и.п. (т.е. индуктивному предположению) функции Σ_B и Σ_C монотонны.

  Σ_A – суперпозиция этих функций и монотонных функций $\cap$ или $\cup : \mathbf{2}^S \to \mathbf{2}^S$ соответственно, поэтому Σ_A тоже монотонна.

- Пусть $A = [a]B$ или $\langle a\rangle B$.

  Формула B удовлетворяет условию теоремы, и $\iota(B) < \iota(A)$, поэтому по и.п. функция Σ_B монотонна.

  Σ_A – суперпозиция функции Σ_B и монотонных функций $[a]$ или $\langle a\rangle : \mathbf{2}^S \to \mathbf{2}^S$ соответственно, где $\forall\, x \in \mathbf{2}^S$

  $$[a](x) = \{s \in S \mid R^a(s) \subseteq x\}, \quad \langle a\rangle(x) = (R^a)^{-1}(x),$$

  поэтому Σ_A тоже монотонна.

- Пусть $A = \mu x.B$ или $\nu x.B$.

  Из условия теоремы и определения правильной формулы следует, что формула B удовлетворяет условию теоремы, и $\iota(B) < \iota(A)$, поэтому по и.п. функция Σ_B монотонна. В частности, $\forall\, \zeta : RV \to \mathbf{2}^S$ оператор $y \mapsto B^{(y/x)\zeta}$ монотонный.

  Докажем монотонность функции Σ_A, т.е. утверждение

  $$\forall\, \zeta, \zeta' \in RV_\Sigma^\bullet \quad \text{если } \zeta \le \zeta', \text{ то } A^\zeta \subseteq A^{\zeta'}.$$

  Рассмотрим лишь случай $A = \mu x.B$ (случай $A = \nu x.B$ рассматривается аналогично).

  По определению значения формул вида $\mu x.B$, $\exists\, n \ge 0$: $A^\zeta = S_n$, $A^{\zeta'} = S'_n$ где S_n, S'_n – последние члены последовательностей

  $$S_0 = \emptyset,\ S_1 = B^{(S_0/x)\zeta},\ S_2 = B^{(S_1/x)\zeta},\ \ldots, S_{n+1} = B^{(S_n/x)\zeta} = S_n,$$
  $$S'_0 = \emptyset,\ S'_1 = B^{(S'_0/x)\zeta'},\ S'_2 = B^{(S'_1/x)\zeta'},\ \ldots, S'_{n+1} = B^{(S'_n/x)\zeta'} = S'_n.$$

  $\forall\, i = 0, \ldots, n$ из включения $S_i \subseteq S'_i$ и неравенства $\zeta \le \zeta'$ следует неравенство $(S_i/x)\zeta \le (S'_i/x)\zeta'$, откуда, учитывая монотонность функции Σ_B, получаем: $S_{i+1} = B^{(S_i/x)\zeta} \subseteq B^{(S'_i/x)\zeta'} = S'_{i+1}$.

  Таким образом, поскольку $S_0 = \emptyset \subseteq S'_0$, то $\forall\, i = 0, \ldots, n\ \ S_i \subseteq S'_i$.

  В частности, $A^\zeta = S_n \subseteq S'_n = A^{\zeta'}$.

- Пусть $A = \bar{B}$.

  Случаи $B = p \in AP$, $B = \top$ или $B = \bot$ тривиальны.

  Случай $B = x \in RV$ невозможен по условию теоремы.

  Если $B = \bar{C}$, то вместо формулы A рассматриваем эквивалентную ей формулу C (которая имеет меньший размер, удовлетворяет условию теоремы, и $\iota(C) = \iota(A)$).

  Если B имеет вид $C \wedge D$, $C \vee D$, $[a]C$, или $\langle a\rangle C$, то вместо формулы A рассматриваем эквивалентную ей формулу $\bar{C} \vee \bar{D}$, $\bar{C} \wedge \bar{D}$, $\langle a\rangle \bar{C}$, или $[a]\bar{C}$, соответственно. Формулы $\bar{C}$ и $\bar{D}$, удовлетворяют условию теоремы, $\iota(\bar{C}) < \iota(A)$ и $\iota(\bar{D}) < \iota(A)$, далее рассуждаем, как в соответствующих пунктах выше.

- Случаи $B = \mu x.C$ и $B = \nu x.C$. Рассмотрим лишь первый случай (второй случай рассматривается аналогично), т.е. $A = \overline{\mu x.C}$.

  Обозначим записью $C(\bar{x})$ формулу, получаемую из C заменой всех свободных вхождений x на $\bar{x}$. Из условия теоремы и определения

правильной формулы следует, что $\overline{C(\bar{x})}$ удовлетворяет условию теоремы, и $\iota(\,\overline{C(\bar{x})}\,) < \iota(A)$, поэтому по и.п. функция $\Sigma_{\overline{C(\bar{x})}}$ монотонна.

$\forall\,\zeta : RV \to \mathbf{2}^S$ оператор $y \mapsto C^{(y/x)\zeta}$ монотонный, т.к. если $y_1 \subseteq y_2$, то $\bar{y}_1 \supseteq \bar{y}_2$, и из монотонности функции $\Sigma_{\overline{C(\bar{x})}}$ следует, что

$$(\,\overline{C}\,)^{(y_1/x)\zeta} = (\,\overline{C(\bar{x})}\,)^{(\bar{y}_1/x)\zeta} \supseteq (\,\overline{C(\bar{x})}\,)^{(\bar{y}_2/x)\zeta} = (\,\overline{C}\,)^{(y_2/x)\zeta},$$

т.е. $C^{(y_1/x)\zeta} \subseteq C^{(y_2/x)\zeta}$.

Докажем монотонность $\Sigma_A = \Sigma_{\overline{\mu x.C}}$, т.е. утверждение

$$\forall\,\zeta,\zeta' \in RV_\Sigma^\bullet \quad \text{если } \zeta \le \zeta', \text{ то } (\mu x.C)^\zeta \supseteq (\mu x.C)^{\zeta'}.$$

По определению значения формул вида $\mu x.C$, $\exists\, n \ge 0$: $(\mu x.C)^\zeta$ равно последнему члену последовательности

$$\begin{array}{l} S_0 = \emptyset,\ S_1 = C^{(S_0/x)\zeta},\ S_2 = C^{(S_1/x)\zeta},\ \ldots, \\ S_{n+1} = C^{(S_n/x)\zeta} = S_n. \end{array} \tag{14.12}$$

Нетрудно видеть, что $\forall\, i \ge 0$ равенство $S_{i+1} = C^{(S_i/x)\zeta}$ равносильно равенству $S \setminus S_{i+1} = (\,\overline{C(\bar{x})}\,)^{((S\setminus S_i)/x)\zeta}$, поэтому можно переписать (14.12) следующим образом:

$$\begin{array}{l} S \setminus S_0 = S,\ S \setminus S_1 = (\,\overline{C(\bar{x})}\,)^{((S\setminus S_0)/x)\zeta}, \\ S \setminus S_2 = (\,\overline{C(\bar{x})}\,)^{((S\setminus S_1)/x)\zeta},\ \ldots, \\ S \setminus S_{n+1} = (\,\overline{C(\bar{x})}\,)^{((S\setminus S_n)/x)\zeta} = S \setminus S_n. \end{array} \tag{14.13}$$

Из (14.13) и из определения значений формул вида $\nu x.A$ следует, что $S \setminus S_n = (\nu x.\overline{C(\bar{x})}\,)^\zeta$, т.е. $(\mu x.C)^\zeta = S \setminus (\nu x.\overline{C(\bar{x})}\,)^\zeta$.

Выше было установлено, что из монотонности функции $\Sigma_{\overline{C(\bar{x})}}$ следует монотонность функции $\Sigma_{\nu x.\overline{C(\bar{x})}}$, поэтому из $\zeta \le \zeta'$ следует включение $(\nu x.\overline{C(\bar{x})}\,)^\zeta \subseteq (\nu x.\overline{C(\bar{x})}\,)^{\zeta'}$, из которого следует желаемое соотношение

$$(\mu x.C)^\zeta = S \setminus (\nu x.\overline{C(\bar{x})}\,)^\zeta \supseteq S \setminus (\nu x.\overline{C(\bar{x})}\,)^{\zeta'} = (\mu x.C)^{\zeta'}. \blacksquare$$

Отметим, что мы попутно доказали утверждение:

$$\forall\, A \in Fm_\mu \quad \overline{\mu x.A} = \nu x.\overline{A(\bar{x})},\ \ \overline{\nu x.A} = \mu x.\overline{A(\bar{x})}, \tag{14.14}$$

где $A(\bar{x})$ получается из A заменой всех свободных вхождений x на $\bar{x}$.

14.4.3 Ускоренное вычисление значений μ-формул

Определение значения A^ζ в пункте 14.4.2 является также и алгоритмом вычисления этого значения. Этот алгоритм имеет сложность $O(|\Sigma|^{|A|})$.

Вычисление значений формул вида $\mu x.B$ и $\nu x.B$ можно ускорить. Ниже излагается рекурсивный алгоритм вычисления значения A^ζ для формулы A вида $\mu x.B$, сложность которого – $O(|\Sigma|^d)$, где d – **глубина чередования** формулы A, определяемая как максимальная длина последовательности $B_1, \ldots, B_k$ подформул формулы A, каждая из которых начинается с μ или ν, $\forall\, i = 1, \ldots, k-1$ B_{i+1} – подформула формулы B_i, и если B_i начинается с μ, то B_{i+1} начинается с ν, а если B_i начинается с ν, то B_{i+1} начинается с μ.

Алгоритм имеет следующий вид.

1. Пронесём в A все отрицания вниз, используя законы де Моргана и утверждения (14.11) и (14.14). Получившуюся формулу обозначим тем же символом A.

   Из определения правильности μ-формулы следует, что после этого все отрицания будут располагаться только над атомарными утверждениями и свободными вхождениями РП.

   Переименуем связанные РП (если это необходимо) так, чтобы в каждой группе связанности в A та РП, которая входит в эту группу, отличалась бы от любой РП, имеющей вхождение в A вне этой группы.

2. Пусть список всех подформул формулы A (включая саму A), которые начинаются с μ и не содержатся в подформулах, начинающихся с ν, имеет вид $\{\mu x.B_x \mid x \in X\}$.

   Будем использовать следующие обозначения:

   - $RV^\bullet_{\Sigma,X} = \{\sigma \in RV^\bullet_\Sigma \mid \forall\, x \in RV \setminus X \;\; \sigma(x) = \zeta(x)\}$,
   - $\forall\, B \in Sub(A)$ запись $\hat{B}$ обозначает формулу, получаемую из B удалением входящих в нее записей $\mu x.$, где $x \in X$, имеем:

     $$\forall\, B \in Sub(A) \quad fv(\hat{B}) \subseteq fv(A) \sqcup X,$$

   - $f_{\hat{A}}$ – оператор вида $RV^\bullet_{\Sigma,X} \to RV^\bullet_{\Sigma,X}$, определяемый следующим образом:

     $$\forall\, \sigma \in RV^\bullet_{\Sigma,X},\ \forall\, x \in X \quad f_{\hat{A}}(\sigma)(x) = \hat{B}^\sigma_x. \tag{14.15}$$

3. Вычисляем последовательность $\sigma_0, \sigma_1, \ldots$ подстановок из $RV^{\bullet}_{\Sigma,X}$:

$$\forall\, x \in X \;\; \sigma_0(x) = \emptyset, \quad \forall i \geq 0 \;\; \sigma_{i+1} = f_{\hat{A}}(\sigma_i).$$

Если для некоторого i верно равенство $\sigma_{i+1} = \sigma_i$, то алгоритм завершает работу, искомое значение A^{ς} полагается равным $\hat{A}^{\sigma_i}$.

Вычисление значений $(\nu x.B)^{\varsigma}$ производится двойственным образом.

Теорема 25

Приведенный выше алгоритм всегда завершается и вычисляет правильное значение A^{ς}.

Доказательство.

Используя теорему 24, нетрудно доказать, что оператор $f_{\hat{A}}$ является монотонным относительно порядка (14.9).

Поскольку σ_0 – наименьший элемент относительно этого порядка, то последовательность $\sigma_0, \sigma_1, \ldots$ является цепью.

Поскольку множество $RV^{\bullet}_{\Sigma,X}$ конечно, то $\exists\, i \geq 0 : \sigma_i = \sigma_{i+1}$, т.е. описанный выше алгоритм всегда завершается.

Докажем равенство $\hat{A}^{\sigma_i} = A^{\varsigma}$.

Из алгоритма вычисления значения A^{ς} следует, что

$$\exists\, \sigma_* \in RV^{\bullet}_{\Sigma,X} : \quad A^{\varsigma} = \hat{A}^{\sigma_*} \quad \text{и} \quad \forall\, x \in X \;\; \sigma_*(x) = \hat{B}^{\sigma_*}_x.$$

Из (14.15) следует, что $\sigma_* = f_{\hat{A}}(\sigma_*)$, т.е. σ_* – неподвижная точка $f_{\hat{A}}$.

По построению, σ_i – наименьшая неподвижная точка $f_{\hat{A}}$, т.е.

$$\forall\, x \in X \;\; \hat{B}^{\sigma_i}_x = \sigma_i(x) \subseteq \sigma_*(x) = \hat{B}^{\sigma_*}_x.$$

С другой стороны, $\forall\, x \in X \;\; \hat{B}^{\sigma_*}_x \subseteq \hat{B}^{\sigma_i}_x$, т.к. σ_* получается в результате итераций, начиная с подстановки σ_0 ($\forall\, x \in X \;\; \sigma_0(x) = \emptyset$), которая $\leq \sigma_i$, и каждый шаг итерации заключается в переходе от подстановки $\sigma \leq \sigma_i$ к подстановке σ', которая тоже будет $\leq \sigma_i$, т.к. $\forall\, x \in X$ либо $\sigma'(x) = \sigma(x)$, либо $\sigma'(x) = \emptyset$, либо $\sigma'(x) = \hat{B}^{\sigma}_x \leq \hat{B}^{\sigma_i}_x = \sigma_i(x)$, поэтому $\sigma_* \leq \sigma_i$.

Таким образом, $\forall\, x \in X \;\; \hat{B}^{\sigma_*}_x = \hat{B}^{\sigma_i}_x$.

Поскольку $A = B_x$ для некоторого $x \in X$, то $A^{\varsigma} = \hat{A}^{\sigma_*} = \hat{A}^{\sigma_i}$. ■

14.4.4 Вложение CTL в μ-исчисление

Пусть $T = \{a\}$.

Каждой CTL-формуле A, в которую входят только CTL-операторы **EX**, **EG**, **EU**, можно сопоставить замкнутую μ-формулу A_μ, получаемую

из A заменой в ней подформул, начинающихся с CTL-операторов, на μ-формулы, по правилу:

- $\mathbf{EX}B$ заменяется на $\langle a\rangle B$,
- $\mathbf{EU}(B, C)$ заменяется на $\mu x.(C \vee (B \wedge \langle a\rangle x))$,
- $\mathbf{EG}B$ заменяется на $\nu x.(B \wedge \langle a\rangle x)$.

Пусть задана СП $\Sigma = (S, R, L)$, где $\forall\, s \in S\ R(s) \neq \emptyset$. Обозначим записью Σ_μ СП $(S, \{R^a\}, L)$, где $R^a = R$. Из теоремы 22 следует, что $\forall\, A \in Fm_{CTL}$ значение A^S в Σ совпадает со значением A_μ в Σ_μ.

Таким образом, задача MC-CTL может быть сведена к задаче вычисления значений μ-формул.

Глава 15

Бинарные диаграммы решений

15.1 Бинарные диаграммы решений и их свойства

15.1.1 Определение бинарной диаграммы решений

Пусть задано множество переменных $X \subseteq Var_{\mathbf{B}} = \{x \in Var \mid \tau(x) = \mathbf{B}\}$.

Бинарная диаграмма решений (Binary Decision Diagram, BDD) над множеством переменных X – это ациклический граф Δ с выделенной начальной вершиной Δ^0. Вершины Δ делятся на два класса:

- **терминальные**, из них не выходит ни одного рёбра, и
- **нетерминальные**, из каждой из них выходят два ребра, одно из которых имеет метку 0, а другое – метку 1.

Множество вершин BDD Δ обозначается тем же символом Δ. Каждая вершина $v \in \Delta$ имеет метку $l(v)$. Если v терминальная, то $l(v) \in \{0, 1\}$, а если v нетерминальная, то $l(v) \in X$.

$\forall\, \theta \in X^\bullet$ определено значение $\Delta^\theta \in \{0, 1\}$, которое вычисляется путем прохода по Δ, начиная с Δ^0. В каждый момент этого прохода

- если мы находимся в нетерминальной вершине v, то переходим к следующей вершине по ребру с меткой $\theta(l(v))$, и
- если мы находимся в терминальной вершине, то вычисление заканчивается и в качестве Δ^θ выдаётся значение $l(v)$.

$\forall\, X \subseteq Var_{\mathbf{B}}$ будем обозначать записью $\Delta(X)$ совокупность всех BDD над множеством переменных X.

15.1.2 Эквивалентность и изоморфность бинарных диаграмм решений

Пусть задано множество переменных $X \subseteq Var_{\mathbf{B}}$.

BDD Δ_1 и Δ_2 из $\Delta(X)$ называются

- **эквивалентными**, если $\forall\, \theta \in X^\bullet \;\; \Delta_1^\theta = \Delta_2^\theta$, и
- **изоморфными**, если существует биекция $\varphi : \Delta_1 \to \Delta_2$, такая, что
  - $\varphi(\Delta_1^0) = \Delta_2^0, \quad \forall\, v \in \Delta_1 \;\; l(v) = l(\varphi(v))$, и
  - Δ_1 содержит ребро из v в v' с меткой $l \in \{0, 1\}$ тогда и только тогда, когда Δ_2 содержит ребро $\varphi(v)$ в $\varphi(v')$ с меткой l.

Очевидно, что если две BDD изоморфны, то они эквивалентны.

Нетрудно построить алгоритм проверки изоморфности BDD Δ_1 и Δ_2 со сложностью $O(|\Delta_1| + |\Delta_2|)$, заключающийся в попытке определить отображение $\varphi : \Delta_1 \to \Delta_2$, обладающее указанными выше свойствами:

- сначала значение φ определяется на Δ_1^0 и
- затем выполняется цикл: если для какой-либо вершины $v \in \Delta_1$ значение $\varphi(v)$ уже определено, то однозначно определяется значение φ на концах рёбер, выходящих из v, так, чтобы выполнялись условия на φ из предыдущего абзаца: $\forall\, b \in \{0, 1\}$ если v_b – конец выходящего из v ребра с меткой b, то $\varphi(v_b)$ – конец выходящего из $\varphi(v)$ ребра с меткой b.

15.1.3 Представление множеств замкнутых подстановок бинарными диаграммами решений

Пусть заданы множество переменных $X \subseteq Var_{\mathbf{B}}$, BDD $\Delta \in \Delta(X)$, и подмножество $S \subseteq X^\bullet$.

Будем говорить что Δ является **представлением** множества S, если

$$S = \{\theta \in X^\bullet \mid \Delta^\theta = 1\}.$$

Понятие представления множеств замкнутых подстановок при помощи BDD можно распространить на множества замкнутых подстановок $S \subseteq X^\bullet$, где не все переменные в X имеют тип $\mathbf{B}$, излагаемым ниже образом.

Пусть значениями каждой переменной $x \in X$ являются битовые строки длины d_x. Обозначим записью X_x совокупность из d_x новых переменных типа **B** и записью $\tilde{X}$ – множество $\bigsqcup_{x \in X} X_x$. Нетрудно видеть, что имеется естественная биекция

$$f : X^\bullet \to \tilde{X}^\bullet,$$

сопоставляющая каждой $\theta \in X^\bullet$ подстановку $f(\theta) : \bigsqcup_{x \in X} X_x \to \{0, 1\}$, где $\forall\, x \in X \;\; f(\theta)$ отображает каждую переменную из X_x в соответствующий бит значения $\theta(x)$.

Будем говорить, что BDD $\Delta \in \Delta_{\tilde{X}}$ представляет множество $S \subseteq X^\bullet$, если эта BDD является представлением множества

$$f(S) = \{f(\theta) \mid \theta \in S\} \subseteq \tilde{X}^\bullet.$$

15.1.4 Редукция бинарных диаграмм решений

Редукцией BDD называется преобразование её в эквивалентную ей BDD меньшего размера. Существуют три операции редукции:

1. если BDD содержит пару v, v' вершин, таких, что $l(v) = l(v')$, и если v и v' нетерминальны, то концы выходящих из v и v' рёбер с одинаковыми метками совпадают, тогда можно

   удалить v и выходящие из неё рёбра и
   рёбра, входящие в v, перенаправить в v',
   если v – начальная вершина исходной BDD,
   то начальной вершиной новой BDD будет v'; (15.1)

2. если BDD содержит вершину v, такую, что выходящие из v рёбра имеют один и тот же конец v', то можно сделать (15.1);

3. если BDD содержит недостижимую вершину (т.е. такую вершину, в которую не существует пути из начальной вершины), то можно удалить эту вершину и все связанные с нею рёбра.

Нетрудно доказать, что при применении каждой из описанных выше операций редукции к какой-либо BDD получившаяся BDD будет эквивалентна исходной BDD.

Редуцированием BDD Δ называется итеративный процесс применения описанных выше операций редукции к Δ (на каждом шаге итерации текущая операция применяется к результату предыдущего шага) до тех пор, пока их будет возможно применять.

BDD называется **нередуцируемой**, если к ней невозможно применить никакую из описанных выше операций редукции.

15.1.5 Представление булевых функций

Пусть $X = \{x_1, \ldots, x_k\} \subseteq Var_{\mathbf{B}}$.

С каждой BDD $\Delta \in \Delta_X$ связана булева функция

$$f_\Delta : X^\bullet \to \{0,1\},$$

сопоставляющая каждой подстановке $\theta \in X^\bullet$ значение $f_\Delta(\theta) \stackrel{\text{def}}{=} \Delta^\theta$.

С другой стороны, для каждой булевой функции $f : X^\bullet \to \{0,1\}$ существует BDD $\Delta_{(f)} \in \Delta_X$, такая, что $f_{\Delta_{(f)}} = f$ – это полное бинарное дерево глубины k, каждое ребро которого имеет метку 0 или 1 и

- корень которого помечен переменной x_1,
- вершины следующего уровня – переменной x_2 и т.д.,
- вершины уровня $k-1$ – переменной x_k и
- каждая вершина v уровня k – значением $f(\theta)$, где $(\theta(x_1), \ldots, \theta(x_k))$ – последовательность меток ребер, образующих путь из корня в v.

15.1.6 Подстановка значений вместо переменных

Пусть Δ – BDD и $b \in \{0,1\}$, тогда

- если $v \in \Delta$ – нетерминальная вершина, то запись v_b обозначает конец выходящего из v ребра с меткой b, и
- $\forall\, x \in Var_{\mathbf{B}}$ запись $\Delta(b/x)$ обозначает BDD, получаемую из Δ удалением всех вершин с меткой x и выходящих из них рёбер, причём перед удалением каждой такой вершины все входящие в неё рёбра перенаправляются в v_b. Если удаляемая вершина – начальная в Δ, то начальной вершиной в $\Delta(b/x)$ будет v_b.

Пусть заданы множество $X \subseteq Var_{\mathbf{B}}$ и функция $f : X^\bullet \to \{0,1\}$. $\forall\, x \in Var_{\mathbf{B}}$, $\forall\, b \in \{0,1\}$ запись $f(b/x)$ обозначает функцию вида

$$X^\bullet \to \{0,1\},$$

сопоставляющую каждому $\theta \in X^\bullet$ значение $f^{(b/x)\theta}$, где

$$x^{(b/x)\theta} = b, \;\; \forall\, y \in X \setminus \{x\} \;\; y^{(b/x)\theta} = y^\theta.$$

Нетрудно доказать, что

$$\begin{array}{l} \forall\, f : X^\bullet \to \{0,1\}, \forall\, x \in Var_{\mathbf{B}} \quad f = \bigvee\limits_{b \in \{0,1\}} ((x = b) \wedge f(b/x)), \\ \forall\, \Delta \in \Delta(X), \forall\, x \in Var_{\mathbf{B}}, \forall\, b \in \{0,1\} \quad f_{\Delta(b/x)} = f_\Delta(b/x). \end{array} \tag{15.2}$$

15.2 Согласованность с порядком переменных

Пусть задано множество $X \subseteq Var_{\mathbf{B}}$ и на X задан линейный порядок ρ.

BDD $\Delta \in \Delta(X)$ называется **согласованной** с порядком ρ, если для каждого ребра из одной нетерминальной вершины $v \in \Delta$ в другую нетерминальную вершину v' имеет место неравенство $l(v) < l(v')$. Запись $\Delta_{X,\rho}$ обозначает совокупность всех $\Delta \in \Delta(X)$, согласованных с ρ.

Пусть заданы множество подстановок $S \subseteq X^\bullet$ и линейный порядок ρ на X. Запись $size(S, \rho)$ обозначает наименьший возможный размер BDD $\Delta \in \Delta_{X,\rho}$, которая представляет S.

Значения $size(S, \rho)$ могут существенно различаться при разных ρ. Например, если $X = \{x_1, y_1, \ldots x_k, y_k\}$ и

$$S = \{\theta \in X^\bullet \mid \Big(\bigwedge_{i=1}^{k} (x_i = y_i) \Big)^\theta = 1\},$$

то $size(S, \rho) = \begin{cases} 3k + 2, \text{ если } \rho = (x_1 < y_1 < \ldots < x_k < y_k), \\ 3 \cdot 2^k - 1, \text{ если } \rho = (x_1 < \ldots < x_k < y_1 < \ldots < y_k). \end{cases}$

Порядок ρ на X называется **оптимальным** для представления S, если для любого порядка ρ' на X $\; size(S, \rho) \leq size(S, \rho')$. Задача проверки оптимальности выбранного порядка является NP-полной.

При выборе порядка на X для представления $S \subseteq X^\bullet$ рекомендуется располагать ближе друг к другу те переменные, которые связаны между собой по смыслу.

Теорема 26

Пусть заданы множество $X \subseteq Var_{\mathbf{B}}$, линейный порядок ρ на X и две нередуцируемые BDD $\Delta_1, \Delta_2 \in \Delta_{X,\rho}$.

Если Δ_1 и Δ_2 эквивалентны, то они изоморфны.

Доказательство.

Для доказательства теоремы сначала докажем лемму.

Лемма

Пусть $\Delta \in \Delta_{X,\rho}$ – нередуцируемая BDD. Свяжем с каждой вершиной $v \in \Delta$

- BDD Δ_v, отличающуюся от Δ только начальной вершиной: $\Delta_v^0 = v$, и
- функцию $f_v : X^\bullet \to \{0, 1\} : \; \forall\, \theta \in X^\bullet \;\; f_v(\theta) = \Delta_v^\theta$.

Тогда $\forall\, v, v' \in \Delta$, если $f_v = f_{v'}$, то $v = v'$.

Доказательство леммы.

Будем доказывать утверждение $v = v'$ индукцией по числу $d(v, v')$, которое равно сумме длин максимальных путей из v и v' в терминальную вершину.

Рассмотрим отдельно следующие случаи.

1. v и v' – терминальные вершины.

   Из $f_v = f_{v'}$ следует, что $l(v) = l(v')$. Так как Δ нередуцируема, то это возможно, только если $v = v'$.

2. Одна из вершин v, v' нетерминальная, другая – терминальная.

   Пусть, например, v – нетерминальная, v' – терминальная.

   В этом случае функция f_v – константа, равная $l(v')$, и все пути из v ведут в v'. Пусть π – путь из v в v' максимальной длины и v'' – последняя нетерминальная вершина на этом пути. Оба ребра, выходящих из v'', ведут в v' (т.к. иначе имеем противоречие с максимальностью π), но это невозможно по причине нередуцируемости Δ.

3. v и v' – нетерминальные вершины.

   Обозначим $x = l(v)$, $y = l(v')$, v_b и v'_b – концы ребер с меткой b ($b = 0, 1$), выходящих из v и v' соответственно.

   Предположим, что $x \neq y$. Пусть, например, $x < y$. В этом случае функция $f_v = f_{v'}$ не зависит существенно от x, поэтому $f_{v_0} = f_{v_1} = f_v = f_{v'}$. Имеем пары вершин (v_b, v'), такие, что $f_{v_b} = f_{v'}$, и $d(v_b, v') < d(v, v')$ ($b = 0, 1$). По и.п. (индуктивному предположению) верны равенства $v_0 = v'$ и $v_1 = v'$, откуда следует, что $v_0 = v_1$, что противоречит нередуцируемости Δ.

   Таким образом, $x = y$.

   Предположим, что $v \neq v'$.

   Если бы были верны равенства $v_0 = v'_0$ и $v_1 = v'_1$, то к Δ можно было бы применить первое правило редукции, что противоречит нередуцируемости Δ, поэтому $\exists\, b \in \{0, 1\} : v_b \neq v'_b$.

   Нетрудно видеть, что $f_{v_b} = f_{v'_b}$ и $d(v_b, v'_b) < d(v, v')$, откуда по и.п. следует, равенство $v_b = v'_b$, которое по предположению неверно.

   Таким образом, предположение $v \neq v'$ неверно. ■

Переходим к доказательству теоремы 26.
Будем вести доказательство индукцией по $|X|$.

Если $|X| = 0$, то $\Delta_1 = \Delta_1^0$, $\Delta_2 = \Delta_2^0$ и $l(\Delta_1^0) = l(\Delta_2^0)$. Такие BDD очевидно изоморфны.

Пусть $|X| = n > 0$, $X = \{x_1, \ldots, x_n\}$ и $x_1 < \ldots < x_n$.

Отдельно рассматриваем случаи, когда:

- одна из вершин Δ_1^0, Δ_2^0 терминальна,
- Δ_1^0 и Δ_2^0 нетерминальны и имеют разные метки,
- Δ_1^0 и Δ_2^0 нетерминальны и имеют одинаковые метки.

В первых двух случаях, рассуждая аналогично тому, как в лемме, получаем, что либо Δ_1 и Δ_2 изоморфны, либо одна из них редуцируема, что по предположению невозможно.

Рассмотрим третий случай: $l(\Delta_1^0) = l(\Delta_2^0)$.

Если $l(\Delta_1^0) = l(\Delta_2^0) \neq x_1$, то $\Delta_1, \Delta_2 \in \Delta_{X\setminus\{x_1\}}$, откуда по и.п. получаем доказываемое утверждение.

Пусть $l(\Delta_1^0) = l(\Delta_2^0) = x_1$ и $\forall\, i = 1, 2$,

- v_{i0} и v_{i1} – концы рёбер с началом в Δ_i^0 и
- Δ_{i0} и Δ_{i1} – BDD, состоящие из тех вершин BDD Δ_i, которые достижимы из v_{i0} и v_{i1} соответственно, $\Delta_{i0}^0 = v_{i0}$ и $\Delta_{i1}^0 = v_{i1}$.

Нетрудно видеть, что $\forall\, b = 0, 1$ $\Delta_{1b}, \Delta_{2b} \in \Delta_{X\setminus\{x_1\},\rho}$ – нередуцируемые эквивалентные BDD. Следовательно, по и.п. эти BDD изоморфны, т.е. $\exists\, \varphi_b : \Delta_{1b} \to \Delta_{2b} : \varphi(v_{1b}) = v_{2b}$ $(b = 0, 1)$, и выполнены другие свойства φ_b из определения изоморфности BDD (пункт 15.1.2).

Докажем, что $\forall\, v \in \Delta_{10} \cap \Delta_{11}$ $\varphi_0(v) = \varphi_1(v)$. Согласно лемме, для этого достаточно доказать равенство функций $f_{\varphi_0(v)}$ и $f_{\varphi_1(v)}$. Обозначим записями Δ_{1v} и $\Delta_{2\varphi_b(v)}$ $(b = 0, 1)$ BDD, состоящие из тех вершин BDD Δ_1 и Δ_2, которые достижимы из v и $\varphi_b(v)$ соответственно, $\Delta_{1v}^0 = v$ и $\Delta_{2\varphi_b(v)}^0 = v_{\varphi_b(v)}$. $\forall\, b = 0, 1$ ограничение φ_b на Δ_{1v} задает изоморфизм между Δ_{1v} и $\Delta_{2\varphi_b(v)}$, откуда нетрудно получить равенства $f_v = f_{\varphi_0(v)}$ и $f_v = f_{\varphi_1(v)}$, откуда следует желаемое равенство $f_{\varphi_0(v)} = f_{\varphi_1(v)}$.

Искомое отображение $\varphi : \Delta_1 \to \Delta_2$ определяется так:

$$\varphi(\Delta_1^0) \stackrel{\text{def}}{=} \Delta_2^0, \quad \forall\, b = 0, 1, \forall\, v \in \Delta_{1b}\ \ \varphi(v) \stackrel{\text{def}}{=} \varphi_b(v).$$

Корректность данного определения следует из приведенных выше рассуждений. Нетрудно проверить, что φ обладается свойствами, изложенными в определении изоморфности BDD в пункте 15.1.2. ■

15.3 Алгебраические операции на BDD

15.3.1 Булевы операции

На BDD можно определить булевы операции: нульарные операции (т.е. константы) 0 и 1, унарную операцию отрицания $\bar{}$, бинарные операции $\wedge$ и $\vee$, – причем данные операции будут согласованы с операциями на булевых функциях, соответствующих BDD, т.е. будут удовлетворять условиям

$$f_0 = 0, \quad f_1 = 1, \quad f_{\overline{\Delta}} = \overline{f_\Delta}, \quad f_{\Delta_1 \wedge \Delta_2} = f_{\Delta_1} \wedge f_{\Delta_2} \quad \text{и т.д.}$$

Данные операции определяются следующим образом.

- BDD, соответствующая булевой константе 0 или 1, обозначается так же, как эта константа, и состоит из одной вершины с меткой 0 или 1 соответственно.

- Для каждой BDD Δ BDD $\overline{\Delta}$ получается из Δ заменой меток терминальных вершин: 0 заменяется на 1, а 1 – на 0.

- Пусть заданы множество $X \subseteq Var_{\mathbf{B}}$, линейный порядок ρ на X и пара BDD $\Delta_1, \Delta_2 \in \Delta_{X,\rho}$.

  Конъюнкция $\Delta_1 \wedge \Delta_2 \in \Delta_{X,\rho}$ определяется рекурсивно следующим образом.

  - Если Δ_1 или Δ_2 является константой, то $\Delta_1 \wedge \Delta_2$ – или константа, или совпадает либо с одной из Δ_i (в соответствии с определением конъюнкции).
  - Пусть Δ_1 и Δ_2 не константы. Обозначим $x = \Delta_1^0$, $y = \Delta_2^0$.
    - Если $x = y$, то $(\Delta_1 \wedge \Delta_2)^0$ имеет метку x, из неё выходит ребро с меткой b в $(\Delta_1(b/x) \wedge \Delta_2(b/x))^0$ (где $b = 0, 1$).
    - Если $x < y$, то $(\Delta_1 \wedge \Delta_2)^0$ имеет метку x, из неё выходит ребро с меткой b в $(\Delta_1(b/x) \wedge \Delta_2)^0$ (где $b = 0, 1$).
    - Если $y < x$, то $\Delta_1 \wedge \Delta_2$ определяется аналогично.

  Дизъюнкция $\Delta_1 \vee \Delta_2$ определяется аналогично (в вышеприведенном определении все символы $\wedge$ заменяются на $\vee$).

Заметим, что вспомогательные BDD $\Delta_i(b/x)$, участвующие в определении $\Delta_1 \wedge \Delta_2$ и $\Delta_1 \vee \Delta_2$, являются подграфами Δ_1 и Δ_2 и полностью определяются теми вершинами Δ_1 и Δ_2, которые являются начальными в этих вспомогательных BDD. Это позволяет во всех выражениях, в

которых участвуют вспомогательные BDD, вместо самих этих BDD записывать только определяющие их вершины. Используя данное соображение, нетрудно доказать, что сложность задачи вычисления бинарных булевых операций на BDD имеет верхнюю оценку $O(|\Delta_1| \cdot |\Delta_2|)$.

15.3.2 Произведение

Пусть заданы BDD $\Delta_1 \in \Delta_{X_1}$ и $\Delta_2 \in \Delta_{X_2}$, которые согласованы с заданным линейным порядком ρ на $X_1 \cup X_2$.

Произведение $\Delta_1 \underset{X}{\wedge} \Delta_2$, где $X = \{x_1, \ldots, x_k\} = X_1 \cap X_2$ – это BDD из $\Delta_{(X_1 \cup X_2) \setminus X, \rho}$, эквивалентная BDD

$$\exists\, x_1 \ldots \exists\, x_k (\Delta_1 \wedge \Delta_2),$$

где для каждой BDD Δ запись $\exists\, x\, \Delta$ обозначает BDD $\Delta(0/x) \vee \Delta(1/x)$.

Ниже излагается рекурсивная программа **Prod**, которая по паре аргументов (Δ_1, Δ_2) указанного выше вида возвращает результат $\Delta_1 \underset{X}{\wedge} \Delta_2$.

В данной программе используется вспомогательная переменная $Cache$, в которой хранятся тройки вида $(\Delta_1, \Delta_2, \mathbf{Prod}(\Delta_1, \Delta_2))$.

Программа **Prod** имеет следующий вид:

Prod$(\Delta_1, \Delta_2) =$

1. если $\Delta_1 = 0$ или $\Delta_2 = 0$, то **return** 0,
2. если $\Delta_1 = 1$ и $\Delta_2 = 1$, то **return** 1,
3. если $(\Delta_1, \Delta_2, \Delta) \in Cache$ то **return** Δ,
4. иначе выполняем следующую последовательность действий:
   (a) $x :=$ максимальная (в смысле указанного выше линейного порядка ρ) из переменных, входящих в Δ_1 или Δ_2,
   (b) $\Delta_{xb} := \mathbf{Prod}(\Delta_1(b/x), \Delta_2(b/x)) \quad (b = 0, 1)$,
   (c) $\Delta := \begin{cases} \Delta_{x0} \vee \Delta_{x1}, & \text{если } x \in X, \\ (\Delta_{(x)} \wedge \Delta_{x1}) \vee (\Delta_{(\bar{x})} \wedge \Delta_{x0}), & \text{если } x \notin X, \end{cases}$
   (d) добавляем $(\Delta_1, \Delta_2, \Delta)$ в $Cache$,
   (e) **return** Δ.

В наихудшем случае сложность этого алгоритма экспоненциальна.

Теорема 27
BDD $\mathbf{Prod}(\Delta_1, \Delta_2)$, вычисляемая описанным выше алгоритмом, эквивалентна BDD $\Delta_1 \underset{X}{\wedge} \Delta_2$.

Доказательство.
Очевидно, что достаточно доказать следующее утверждение: если $\Delta_{xb} = \Delta_1(b/x) \underset{X}{\wedge} \Delta_2(b/x))$, то BDD Δ, вычисляемая в пункте (4c) описанного выше алгоритма, эквивалентна $\Delta_1 \underset{X}{\wedge} \Delta_2$.

Для этого достаточно доказать совпадение булевых функций, соответствующих рассматриваемым BDD.

Обозначим символами f и g функции $f_{\Delta_1 \underset{X}{\wedge} \Delta_2}$ и $f_{\Delta_1} \wedge f_{\Delta_2}$ соответственно. В этих обозначениях

$$\begin{aligned} f &= \bigvee_{(b_1 \ldots b_k) \in \{0,1\}^k} g(b_1/x_1) \ldots (b_k/x_k), \\ f_{\Delta_{xb}} &= \bigvee_{(b_1 \ldots b_k) \in \{0,1\}^k} g(b/x)(b_1/x_1) \ldots (b_k/x_k). \end{aligned} \tag{15.3}$$

Требуется доказать, что $\forall\, x \in Var_{\mathbf{B}}$

$$\left.\begin{array}{l} x \in X \;\Rightarrow\; f = f_{\Delta_{x0}} \vee f_{\Delta_{x1}} \\ x \notin X \;\Rightarrow\; f = ((x = 1) \wedge f_{\Delta_{x1}}) \vee ((x = 0) \wedge f_{\Delta_{x0}}) \end{array}\right\}. \tag{15.4}$$

(15.4) можно переписать в виде

$$\left.\begin{array}{l} x \in X \;\Rightarrow\; f = \bigvee\limits_{b \in \{0,1\}} f_{\Delta_{xb}} \\ x \notin X \;\Rightarrow\; f = \bigvee\limits_{b \in \{0,1\}} ((x = b) \wedge f_{\Delta_{xb}}) \end{array}\right\}. \tag{15.5}$$

Согласно первому утверждению в (15.2), $\forall\, x \in Var_{\mathbf{B}}$

$$g = \bigvee_{b \in \{0,1\}} ((x = b) \wedge g(b/x)). \tag{15.6}$$

Обозначим последовательность подстановок $(b_1/x_1) \ldots (b_k/x_k)$ символом θ. Применяя θ к обоим частям (15.6), получаем

$$x \notin X \;\Rightarrow\; g\theta = \bigvee_{b \in \{0,1\}} ((x = b) \wedge g(b/x))\theta. \tag{15.7}$$

Кроме того, нетрудно видеть, что

$$x \in X \;\Rightarrow \bigvee_{(b_1 \ldots b_k) \in \{0,1\}^k} g\theta = \bigvee_{(b_1 \ldots b_k) \in \{0,1\}^k} \bigvee_{b \in \{0,1\}} g(b/x)\theta. \tag{15.8}$$

Из (15.7) и (15.8), учитывая (15.3), получаем (15.5). ■

15.4 MC-CTL с использованием бинарных диаграмм решений

Напомним, что одна из форм задачи MC-CTL заключается в вычислении по заданным СП $\Sigma = (S, R, L)$ и формуле $A \in Fm_{CTL}$ множества A^S.

В том случае, когда

- $S = X^\bullet$, где $X \subseteq Var_{\mathbf{B}}$ – конечное множество переменных, и
- компоненты СП Σ определяются формулами из множеств $Fm(X)$ и $Fm(X \sqcup X')$,

данная задача м.б. решена методом, связанным с понятием неподвижной точки (изложенным в параграфе 14.3), с использованием BDD.

Идея излагаемого ниже подхода заключается в том, чтобы представлять каждое подмножество $S' \subseteq S = X^\bullet$, возникающее в процессе вычисления A^S, не явным образом, а в виде BDD, представляющей S' (в соответствии с тем, как это было определено в пункте 15.1.3).

Как и раньше, без ограничения общности мы можем предполагать, что в A могут входить лишь CTL-операторы вида **EX**, **EG**, **EU**. Вычисление множества A^S заключается в вычислении множеств вида B^S для всех подформул формулы A.

Предположим, что

- для каждого атомарного утверждения $p \in AP$, входящего в A, задана BDD $\Delta(p) \in \Delta(X)$, представляющая множество p^S, и
- задана BDD $\Delta(R) \in \Delta_{X \sqcup X'}$, представляющая отношение перехода $R \subseteq X^\bullet \times X^\bullet$,
- $X = \{x_1, \ldots, x_k\}$, и все упомянутые в этом абзаце BDD согласованы с подрядком $x_1 < \ldots < x_k < x'_1 < \ldots < x'_k$.

Тогда для вычисления BDD $\Delta(A)$, представляющей множество A^S, можно использовать излагаемый ниже рекурсивный алгоритм. Данный алгоритм сводит вычисление BDD $\Delta(A)$ к вычислению BDD $\Delta(B)$, представляющих множества вида B^S, где B – подформула формулы A.

1. Если $A = p \in AP$, то BDD $\Delta(p)$ предполагается заданной.
2. Если A – булева комбинация, то $\Delta(A)$ вычисляется путем применения соответствующих булевых операций к BDD, соответствующим компонентам этой булевой комбинации:

$$\Delta(\top) = 1, \;\; \Delta(\bot) = 0,$$
$$\Delta(\bar{B}) = \overline{\Delta(B)}, \;\; \Delta(B \wedge C) = \Delta(B) \wedge \Delta(C) \text{ и т.д.}$$

3. Пусть $A = \mathbf{EX}B$ и имеется BDD $\Delta(B)$, представляющая B^S.

   Из определения множества

   $$(\mathbf{EX}B)^S = R^{-1}(B^S) = \{\theta \in X^\bullet \mid R(\theta) \cap B^S \neq \emptyset\}$$

   следует, что $\forall\, \theta \in X^\bullet$

   $$\theta \in (\mathbf{EX}B)^S \;\; \Leftrightarrow \;\; \exists\, \theta' \in X^\bullet : (\theta, \theta') \in R, \;\; \theta' \in B^S. \tag{15.9}$$

   Соотношение (15.9) можно переписать в терминах BDD, представляющих упомянутые в нём множества:

   $$\Delta(\mathbf{EX}B)^\theta = 1 \;\; \Leftrightarrow \;\; \exists\, \theta' \in X^\bullet : \Delta(R)^{\theta \sqcup \theta'} = 1 \;\text{ и }\; \Delta(B)^{\theta'} = 1. \tag{15.10}$$

   Истинность (15.10) $\forall\, \theta \in X^\bullet$ равносильна эквивалентности

   $$\Delta(\mathbf{EX}B) \;=\; \exists\, x'_1 \ldots \exists\, x'_k \, \Big(\Delta(R) \wedge \Delta(B)(x'_1, \ldots, x'_k)\Big),$$

   где $X = \{x_1, \ldots, x_k\}$ и $\Delta(B)(x'_1, \ldots, x'_k)$ получается из $\Delta(B)$ заменой каждой переменной $x_i \in X$ на соответствующую переменную x'_i $(i = 1, \ldots, x_k)$. Будем обозначать данную BDD записью $\Delta(B)'$.

   Таким образом, вычисление $\Delta(\mathbf{EX}B)$ сводится к вычислению произведения $\Delta(R) \underset{X'}{\wedge} \Delta(B)'$. Будем обозначать его записью $\mathbf{EX}(\Delta(B))$.

4. Если $A = \mathbf{EU}(B, C)$ или $\mathbf{EG}B$, то $\Delta(A)$ вычисляется согласно (14.6):

   $$\begin{aligned} &\Delta(\mathbf{EU}(B, C)) = \mu x.\Big(\Delta(C) \vee (\Delta(B) \wedge \mathbf{EX}(x))\Big), \\ &\Delta(\mathbf{EG}B) = \nu x.\Big(\Delta(B) \wedge \mathbf{EX}(x)\Big), \end{aligned} \tag{15.11}$$

   где $\forall\, \Delta \in \Delta(X)$ запись $\mathbf{EX}(\Delta)$ обозначает произведение $\Delta(R) \underset{X'}{\wedge} \Delta'$ и Δ' определяется аналогично $\Delta(B)'$ в предыдущем пункте.

   Неподвижные точки в (15.11) вычисляются путем построения соответствующих последовательностей BDD, начиная с 0 или 1. После вычисления каждого нового члена последовательности производится его редукция и проверка его эквивалентности предыдущему члену этой последовательности. Согласно теореме 26, данная проверка сводится к проверке изоморфности этих членов. Построение последовательности заканчивается, когда новый построенный её член изоморфен предыдущему её члену.

15.5 Оптимизирующие преобразования

Если СП Σ строится по РП согласно определениям из пункта 13.2.3, то её отношение перехода определяется формулой $\bigvee_{i=1}^{n} R_i$. Поскольку квантор $\exists$ коммутирует с дизъюнкцией, то $\forall\, \Delta \in \Delta(X)$ для вычисления $\mathbf{EX}(\Delta)$ можно вычислять BDD, определяемую выражением

$$\bigvee_{i=1}^{n} \exists\, x'_1 \ldots \exists\, x'_k \left(\Delta_{(R_i)} \wedge \Delta(x'_1, \ldots, x'_k) \right). \tag{15.12}$$

$\forall\, i = 1, \ldots, n$ обозначим записью C_i i-й дизъюнктивный член в (15.12). Можно оптимизировать выражение C_i (т.е. преобразовать его в такое выражение $\tilde{C}_i$, по которому соответствующая BDD вычисляется быстрее) следующим образом. Пусть $R_i = \bigwedge_{j=1}^{n_i} R_{ij}$, где некоторые из R_{ij} имеют вид $x' = x$ или $x' = b$, где $b \in \{0, 1\}$, или не содержат переменных из X'. Тогда можно выполнить следующие преобразования выражения C_i.

- Каждый конъюнктивный член в R_i вида $x' = x$ удаляется, и компонента $\Delta(x'_1, \ldots, x'_k)$ выражения C_i преобразуется путем замен всех вхождений переменной x' на x.

  Обозначим записью $\tilde{\Delta}$ результат этих преобразований.

- Каждый конъюнктивный член в R_i вида $x' = b$, где $b \in \{0, 1\}$, удаляется, компонента $\tilde{\Delta}$ преобразуется в $\tilde{\Delta}(b/x')$.

  Будем обозначать получившуюся BDD той же записью $\tilde{\Delta}$.

- Каждый конъюнктивный член R_{ij} в R_i, не содержащий переменных из X', удаляется из R_i, и в выражении C_i перед кванторами существования приписывается конъюнктивный член $\Delta_{R_{ij}}$.

  Обозначим записью $\tilde{R}_i$ конъюнкцию оставшихся членов в R_i и записью $\tilde{C}_i$ – преобразованное выражение C_i.

- Если какая-либо из переменных $x' \in X'$ не входит в $\tilde{R}_i$ и $\tilde{\Delta}$, то фрагмент $\exists\, x'$ удаляется из $\tilde{C}_i$.

Нетрудно видеть, что C_i и $\tilde{C}_i$ определяют эквивалентные BDD.

Глава 16

Model checking на основе LTL

16.1 Формулы темпоральной логики LTL

Совокупность **формул** темпоральной логики LTL (Linear Temporal Logic), называемых также **LTL-формулами**, обозначается записью Fm_{LTL} и определяется следующим образом:

- $AP \subseteq Fm_{LTL}$,
- Fm_{LTL} замкнуто относительно булевых комбинаций,
- $\forall\, B \in Fm_{LTL}$ записи $\mathbf{X}B,\ \mathbf{F}B,\ \mathbf{G}B$ являются LTL-формулами,
- $\forall\, B, C \in Fm_{LTL}$ запись $\mathbf{U}(B, C)$ является LTL-формулой.

Жирные символы ($\mathbf{X}$, $\mathbf{F}$, $\mathbf{G}$, $\mathbf{U}$) в LTL-формулах называются **темпоральными операторами**.

Пусть задана СП Σ и $\pi = (s_0, s_1, \ldots)$ – путь в Σ. Для каждой LTL-формулы B **значение** $B^\pi \in \{0, 1\}$ формулы B на пути π определяется ниже индуктивно. $\forall\, i \geq 0$ будем обоначать записью π_i хвост пути π, начинающийся с позиции i: $\pi_i = (s_i, s_{i+1}, \ldots)$.

- Если $B = p \in AP$, то $B^\pi \stackrel{\text{def}}{=} \begin{cases} p^{s_0} = L_\Sigma(s_0, p), \text{ если } p \in AP_\Sigma, \\ 0, \text{ если } p \notin AP_\Sigma. \end{cases}$
- Значения булевых комбинаций определяются стандартно:

$$\top^\pi = 1,\ \ \bot^\pi = 0,$$
$$(\bar{B})^\pi = \overline{B^\pi},\ \ (B \wedge C)^\pi = B^\pi \wedge C^\pi \text{ и т.д.}$$

- Значения LTL-формул, начинающихся с темпорального оператора, определяются следующим образом:

– $(\mathbf{X}B)^\pi \stackrel{\text{def}}{=} B^{\pi_1}$,
– $(\mathbf{F}B)^\pi = 1$, если $\exists\, i \geq 0 :\ B^{\pi_i} = 1$,
– $(\mathbf{G}B)^\pi = 1$, если $\forall\, i \geq 0 \quad B^{\pi_i} = 1$,
– $(\mathbf{U}(B,C))^\pi = 1$, если либо $C^\pi = 1$, либо $\exists\, i > 0$:

$$C^{\pi_i} = 1, \forall\, j < i \quad B^{\pi_j} = 1.$$

Будем называть LTL-формулы A и B **эквивалентными**, если для каждой СП Σ, $\forall\, \pi \in \Pi_\Sigma$ верно равенство $A^\pi = B^\pi$. Запись $A = B$, где $A, B \in Fm_{LTL}$, означает что A и B эквивалентны.

Нетрудно доказать, что верны соотношения

$$\mathbf{F}B = \mathbf{U}(\top, B), \quad \overline{\mathbf{X}B} = \mathbf{X}\bar{B}, \quad \overline{\mathbf{G}B} = \mathbf{F}\bar{B}.$$

Следовательно, для любой LTL-формулы существует эквивалентная ей LTL-формула, в которую входят только связки $^-, \vee, \mathbf{X}, \mathbf{U}$.

Кроме того, нетрудно доказать, что $\forall\, B, C \in Fm_{LTL}$ верно равенство

$$\mathbf{U}(B,C) = C \vee \Big(B \wedge \mathbf{X}\mathbf{U}(B,C)\Big). \tag{16.1}$$

Ниже будет использоваться соглашение: знакосочетание $\mathbf{U}(B,C)$ будет обозначаться символом $*$. В этих обозначениях (16.1) записывается в виде равенства $* = C \vee \Big(B \wedge \mathbf{X} * \Big)$.

16.2 Квантифицированные LTL-формулы

Совокупность **квантифицированных LTL-формул** обозначается записью QFm_{LTL} и определяется индуктивно:

- записи вида $\mathbf{A}B$ или $\mathbf{E}B$, где B – LTL-формула, являются квантифицированными LTL-формулами,
- QFm_{LTL} замкнуто относительно булевых комбинаций.

Пусть задана СП $\Sigma = (S, \ldots)$. $\forall\, s \in S, \forall\, A \in QFm_{LTL}$ **значение** A в s обозначается знакосочетанием A^s и определяется следующим образом:

$$(\mathbf{A}B)^s = 1, \ \text{ если } \ \forall\, \pi \in \Pi_s \ \ B^\pi = 1,$$
$$(\mathbf{E}B)^s = 1, \ \text{ если } \ \exists\, \pi \in \Pi_s : B^\pi = 1,$$

и значения булевых комбинаций определются стандартным образом.

Будем называть формулы $A, B \in QFm_{LTL}$ **эквивалентными**, если для каждого состояния s произвольной СП верно равенство $A^s = B^s$. Если формулы $A, B \in QFm_{LTL}$ эквивалентны, то будем обозначать этот факт знакосочетанием $A = B$.

Из этого определения следует, что

$$\forall\, B \in Fm_{LTL} \quad \overline{\mathbf{A}B} = \mathbf{E}\overline{B}. \tag{16.2}$$

Аналогично определяется эквивалентность между CTL-формулами и квантифицированными LTL-формулами. Можно доказать, что

- CTL-формула $\mathbf{AG}(\mathbf{EF}p)$ не эквивалентна ни одной квантифицированной LTL-формуле,
- квантифицированная LTL-формула $\mathbf{A}(\mathbf{FG}p)$ не эквивалентна ни одной CTL-формуле,
- дизъюнкция приведённых выше формул не эквивалентна ни одной квантифицированной LTL-формуле и ни одной CTL-формуле.

16.3 Задачи model checking для LTL

16.3.1 Система переходов Σ_A

Ниже предполагаем, что каждая рассматриваемая LTL-формула содержит связки только из множества $\{^{-}, \vee, \mathbf{X}, \mathbf{U}\}$.

Будем использовать следующие понятия и обозначения:

- $\forall\, A \in Fm_{LTL}$ запись $\langle A\rangle$ обозначает наименьшее (по отношению включения) множество формул, удовлетворяющее условиям:
  - $\langle A\rangle$ содержит все подформулы формулы A,
  - если $* \in \langle A\rangle$, то $\mathbf{X}* \in \langle A\rangle$,

  множество $\langle A\rangle$ называется **замыканием** формулы A;
- множество $AP \cap \langle A\rangle$ будем обозначать записью AP_A.

С каждой формулой $A \in Fm_{LTL}$ связана СП

$$\Sigma_A = (S_A, R_A, L_A, S_A^0, \mathcal{F}_A),$$

определяемая следующим образом:

- S_A состоит из функций $f : \langle A\rangle \to \{0,1\}$, таких, что

$$\begin{array}{l}\forall\, \bar{B} \in \langle A\rangle \quad f(\bar{B}) = \overline{f(B)},\\ \forall\, (B \vee C) \in \langle A\rangle \quad f(B \vee C) = f(B) \vee f(C)\\ \forall * \in \langle A\rangle \quad f(*) = f(C) \vee \Big(f(B) \wedge f(\mathbf{X}*)\Big);\end{array} \tag{16.3}$$

- R_A состоит из всех пар $(f, f') \in S_A \times S_A$, таких, что

$$\forall\, \mathbf{X}B \in \langle A\rangle \quad f(\mathbf{X}B) = f'(B); \tag{16.4}$$

- $AP_{\Sigma_A} = AP_A, \forall\, f \in S_A, \forall p \in AP_A \quad L_A(f,p) = f(p)$;
- $S_A^0 \stackrel{\text{def}}{=} \{f \in S_A \mid f(A) = 1\}$;
- $\mathcal{F}_A = \{F_* \mid * \in \langle A\rangle\}$, где для каждой формулы из $\langle A\rangle$ вида $*$

$$F_* \stackrel{\text{def}}{=} \{f \in S_A \mid f(*) \le f(C)\}. \tag{16.5}$$

Лемма 1
Для каждого пути $\varphi = (f_0, \ldots)$ в СП Σ_A

$$\varphi \text{ – справедлив} \quad \Leftrightarrow \quad \forall\, i \ge 0,\ \forall * \in \langle A\rangle \quad f_i(*) \le \bigvee\nolimits_{j \ge i} f_j(C). \tag{16.6}$$

Доказательство.
Пусть левая часть (16.6) верна, а правая – нет, т.е. $\exists\, i \ge 0,\ \exists * \in \langle A\rangle$:

$$f_i(*) = 1, \forall\, j \ge i \quad f_j(C) = 0. \tag{16.7}$$

Из (16.7) следует, что

$$\begin{array}{l}\forall\, j \ge i \quad 1 = f_i(*) = f_i(C) \vee (f_i(B) \wedge f_i(\mathbf{X}*)) =\\ = f_i(B) \wedge f_i(\mathbf{X}*) = f_i(B) \wedge f_{i+1}(*),\end{array}$$

поэтому $f_{i+1}(*) = 1$, т.е. верно утверждение (16.7) с заменой i на $i+1$. Отсюда следует, что $f_{i+2}(*) = 1$ и т.д., т.е.

$$\forall\, j \ge i \quad f_j(*) = 1. \tag{16.8}$$

Так как путь φ справедлив, то, в частности, $\exists\, j \ge i : f_j \in F_*$, т.е.

$$f_j(*) \le f_j(C). \tag{16.9}$$

(16.9) противоречит (16.8) и свойству $\forall\, j \ge i \quad f_j(C) = 0$.

Обратно, пусть правая часть (16.6) верна, а левая – нет, т.е. путь φ не справедлив, что означает

$$\exists * \in \langle A\rangle,\ \exists\, i \ge 0 : \forall\, j \ge i \ \ f_j \notin F_*, \tag{16.10}$$

т.е. $\forall\, j \ge i \ \ f_j(*) = 1,\ f_j(C) = 0$, откуда следует, что

$$f_i(*) = 1, \quad \bigvee\nolimits_{j \ge i} f_j(C) = 0, \tag{16.11}$$

что противоречит правой части (16.6). ■

16.3.2 Система переходов $\Sigma \times \Sigma_A$

Пусть заданы СП $\Sigma = (S_\Sigma, R_\Sigma, L_\Sigma)$ и LTL-формула A.

Запись $\Sigma \times \Sigma_A$ обозначает СП $(S, R, L, S^0, \mathcal{F})$, где

- $S = \{(s, f) \in S_\Sigma \times S_A \mid \forall\, p \in AP_A \;\; p^s = f(p)\}$,
- $R = \{((s, f), (s', f')) \mid (s, s') \in R_\Sigma, (f, f') \in R_A\}$,
- $\forall\, (s, f) \in S, \forall\, p \in AP_A \quad L((s, f), p) = p^s = f(p)$,
- $S^0 = S^0_\Sigma \times S^0_A$,
- $\mathcal{F} = \{S_\Sigma \times F_* \mid * \in \langle A \rangle\}$, где множества F_* определяются в (16.5).

Нетрудно видеть, что путь ψ в $\Sigma \times \Sigma_A$ справедлив тогда и только тогда, когда последовательность его вторых компонентов – справедливый путь в Σ_A.

Каждый путь π в Σ определяет путь $\varphi_\pi = (f_0, \ldots)$ в Σ_A, где

$$\forall\, A' \in \langle A \rangle, \quad \forall\, i \geq 0 \qquad f_i(A') = (A')^{\pi_i}. \tag{16.12}$$

Нетрудно видеть, что $\forall\, i \geq 0 \;\; f_i \in S_A$ и $(f_i, f_{i+1}) \in R_A$.

Лемма 2
φ_π – справедливый путь в Σ_A.

Доказательство.
Если φ_π не справедлив, то верно (16.10) и (16.11), откуда, учитывая (16.12), получаем:

$$\exists * \in \langle A \rangle,\; \exists\, i \geq 0 : \forall\, j \geq i \quad *^{\pi_j} = 1,\; C^{\pi_j} = 0. \tag{16.13}$$

Но т.к. $*^{\pi_i} = 1$, то $\exists\, j \geq i : C^{\pi_j} = 1$, что противоречит (16.13). ■

$\forall\, \pi \in \Pi_\Sigma$ обозначим записью ψ_π путь в $\Sigma \times \Sigma_A$, имеющий вид

$$((s_0, f_0), (s_1, f_1), \ldots), \quad \text{где } \pi = (s_0, s_1, \ldots),\; \varphi_\pi = (f_0, f_1, \ldots). \tag{16.14}$$

Из леммы 2 следует, что путь ψ_π справедлив.

Лемма 3
Соответствие $\pi \mapsto \psi_\pi$ из Π_Σ в $\Pi_{\Sigma \times \Sigma_A}$ биективно.

Доказательство.

Инъективность данного соответствия очевидна.

Докажем его сюръективность, т.е. докажем, что для каждого пути

$$\psi = ((s_0, f_0), (s_1, f_1), \ldots) \in \Pi_{\Sigma \times \Sigma_A}$$

верно утверждение (16.12), где $\pi = (s_0, s_1, \ldots)$. Доказательство проведем индукцией по структуре формулы A'. Предполагаем, что для собственных подформул формулы A' это утверждение верно.

1. $A' = p \in AP$: в этом случае, по определению состояний СП $\Sigma \times \Sigma_A$,

$$f_i(A') = f_i(p) = p^{s_i} = p^{\pi_i} = (A')^{\pi_i}.$$

2. A' имеет вид булевой комбинации: в этом случае доказательство (16.12) не представляет особой сложности.

3. $A' = \mathbf{X}B$: $f_i(\mathbf{X}B) = f_{i+1}(B) = B^{\pi_{i+1}} = (\mathbf{X}B)^{\pi_i}$.

4. $A' = *$: докажем равенство $f_i(*) = *^{\pi_i}$.

   - Предположим, что $f_i(*) = 1$. Так как путь $(f_0, f_1, \ldots)$ справедлив, то по лемме 1 верна правая часть (16.6), поэтому из $f_i(*) = 1$ следует, что $\exists\, j \geq i : f_j(C) = 1$. Будем считать, что j – наименьший индекс $\geq i$, такой, что $f_j(C) = 1$. Если $j > i$, то из последнего равенства в (16.3), где $f = f_i$, следует, что $f_i(B) = 1$, $f_{i+1}(*) = 1$. Если $j > i + 1$, то аналогично получаем $f_{i+1}(B) = 1$, $f_{i+2}(*) = 1$ и т.д. Таким образом, имеем равенство

   $$f_i(B) \wedge \ldots \wedge f_{j-1}(B) \wedge f_j(C) = 1. \tag{16.15}$$

   По и.п. данное равенство равносильно равенству

   $$B^{\pi_i} \wedge \ldots \wedge B^{\pi_{j-1}} \wedge C^{\pi_j} = 1, \tag{16.16}$$

   откуда по определению значения $*^{\pi_i}$ следует, что $*^{\pi_i} = 1$.

   - Предположим, что $*^{\pi_i} = 1$. По определению значения $*^{\pi_i}$, отсюда следует, что $\exists\, j \geq i$: верно равенство (16.16). Используя и.п., из (16.16) можно получить (16.15), из которого нетрудно получить равенство $f_i(*) = 1$. ■

16.3.3 Первая задача model checking для LTL

Одна из задач **MC-LTL** имеет следующий вид: заданы СП $\Sigma = (S, R, L)$, состояние $s \in S$ и квантифицированная LTL-формула Q. Требуется вычислить значение Q^s. Из определения понятия квантифицированной LTL-формулы и свойства (16.2) следует, что данная задача сводится к аналогичной задаче для случая, когда Q имеет вид $\mathbf{E}A$.

Согласно определению, равенство $(\mathbf{E}A)^s = 1$ означает, что

$$\exists\, \pi \in \Pi_s :\ A^\pi = 1. \tag{16.17}$$

Как было сказано выше, каждому пути $\pi \in \Pi_s$ соответствует путь

$$\psi_\pi \in \Pi_{(s,f_0)}, \tag{16.18}$$

где состояние $f_0 \in S_A$ определяется соотношением (16.12). Частным случаем этого соотношения является равенство $f_0(A) = A^\pi$.

Теорема 28

Для проверки свойства (16.17) можно использовать следующий метод: $\forall\, f \in S_A^0$ проверить существование пути

$$\psi \in \Pi_{(s,f)}; \tag{16.19}$$

если для какого-либо $f \in S_A^0$ такой путь существует, то ответ положителен (т.е. свойство (16.17) верно), иначе ответ отрицателен.

Доказательство.

- Если существует путь $\pi \in \Pi_s$, такой, что $A^\pi = 1$, то упомянутое выше состояние $f_0 \in S_A$, определяемое этим путем, обладает свойством $f_0(A) = A^\pi = 1$, т.е. $f_0 \in S_A^0$. Поскольку верно свойство (16.18), то, следовательно, существует путь (16.19) (это будет путь ψ_π), т.е. этот метод даст положительный ответ.

- Обратно, предположим, что описанный выше метод даст положительный ответ, т.е. $\exists\, f \in S_A^0$, $\exists\, \psi$: (16.19). По лемме 3, $\exists\, \pi \in \Pi_s$: $\psi = \psi_\pi$. В частности, $f(A) = A^\pi$. Так как $f \in S_A^0$, то $f(A) = 1$, откуда следует, что $A^\pi = 1$. ■

Задача проверки существования пути (16.19) может решаться различными способами. Один из них – рассматривать её как задачу MC-CTL, которая заключается в вычислении значения CTL-формулы $\mathbf{EG}\top$

в состоянии (s, f) СП $\Sigma \times \Sigma_A$. Эту задачу можно решать алгоритмом MC-CTL, основанным на использовании BDD, его сложность – $O(|\Sigma| \cdot 2^{|A|})$. Отметим, что результатом работы данного алгоритма является множество $(\mathbf{EG}\top)^{S_{\Sigma \times \Sigma_A}}$, из которого может быть получено множество $(\mathbf{E}A)^S$.

Существование пути ψ, удовлетворяющего условию (16.19), может быть проверено также с использованием следующей теоремы.

Теорема 29

Для каждого состояния (s, f) СП $\Sigma \times \Sigma_A$ следующие условия эквивалентны:

1. существует путь ψ, удовлетворяющий условию (16.19);
2. существует конечный путь из (s, f) в некоторую SCC $\mathbf{C}$, которая является **справедливой**, т.е. удовлетворяет условию

$$\forall * \in \langle A \rangle \;\; \mathbf{C} \cap F_* \neq \emptyset, \quad \text{где } F_* \stackrel{\text{def}}{=} \{f \in S_A \mid f(*) \leq f(C)\}. \quad (16.20)$$

Доказательство.

(1) $\Rightarrow$ (2): $\exists\, i$: $\psi_i = \inf(\psi) \subseteq$ SCC $\mathbf{C}$. Нетрудно видеть, что $\mathbf{C}$ удовлетворяет условию (16.20).

(2) $\Rightarrow$ (1): $\psi \stackrel{\text{def}}{=} \hat{\psi} \cdot \tilde{\psi}^{\infty}$, где $\hat{\psi}$ – конечный путь из (s, f) в $(s', f') \in \mathbf{C}$ и $\tilde{\psi}$ – цикл из (s', f') в (s', f'), содержащий все состояния из $\mathbf{C}$. ■

Можно доказать, что описанная выше задача MC-LTL PSPACE-полна.

16.3.4 Вторая задача model checking для LTL

Пусть заданы СП $\Sigma = (S, R, L, S^0)$ и LTL-формула A.

Вторая задача **MC-LTL** имеет следующий вид: доказать, что

$$\forall\, \pi \in \Pi^0_\Sigma \quad A^\pi = 1. \quad (16.21)$$

Данная задача сводится к задаче, связанной с излагаемым в пункте 16.4 понятием **автомата Бюхи**. Для такого сведения мы сформулируем вспомогательные понятия и приведём связанные с ними утверждения.

Будем использовать следующие обозначения: пусть $A \in Fm_{LTL}$, тогда

- запись $[A]$ обозначает множество функций вида $g : AP_A \to \{0, 1\}$,
- запись $[A]^\infty$ обозначает множество всех бесконечных последовательностей элементов множества $[A]$,

- для произвольных СП Σ и пути $\pi = (s_0, \ldots) \in \Pi_\Sigma$ запись $\mathcal{L}(\pi)$ обозначает последовательность

$$(g_0, g_1, \ldots) \in [A]^\infty, \text{ где } \forall\, i \geq 0,\ \forall\, p \in AP_A \quad g_i(p) = p^{\pi_i} = p^{s_i}.$$

Лемма 4
$\forall\, A \in Fm_{LTL}$, $\forall$ СП Σ, $\forall\, \pi \in \Pi_\Sigma, \forall\, \varphi \in \Pi_{\Sigma_A}$

$$\mathcal{L}(\pi) = \mathcal{L}(\varphi) \quad \Leftrightarrow \quad \varphi_\pi = \varphi.$$

Доказательство.
Пусть π и φ имеют вид $(s_0, s_1, \ldots)$ и $(f_0, f_1, \ldots)$ соответственно.

- Импликация «$\Rightarrow$» следует из леммы 3 и из того, что путь

$$((s_0, f_0), (s_1, f_1), \ldots)$$

является справедливым путём в $\Sigma \times \Sigma_A$.

- Импликация «$\Leftarrow$», т.е. равенство $\mathcal{L}(\pi) = \mathcal{L}(\varphi_\pi)$, следует из определения 16.12 пути φ_π. ■

Будем обозначать множество $\{\mathcal{L}(\pi) \mid \pi \in \Pi^0_\Sigma\}$ (где Σ – произвольная СП) записью $\mathcal{L}(\Sigma)$.

Лемма 5
Свойство (16.21) равносильно включению $\mathcal{L}(\Sigma) \subseteq \mathcal{L}(\Sigma_A)$.

Доказательство.

- Пусть верно (16.21). Докажем, что $\forall\, \pi \in \Pi^0_\Sigma$

$$\mathcal{L}(\pi) \in \mathcal{L}(\Sigma_A). \tag{16.22}$$

Согласно (16.21), из $\pi \in \Pi^0_\Sigma$ следует, что $A^\pi = 1$. По лемме 4 верно равенство $\mathcal{L}(\pi) = \mathcal{L}(\varphi_\pi)$, поэтому для доказательства (16.22) нужно доказать, что $\varphi_\pi \in \Pi^0_{\Sigma_A}$, т.е. если φ_π имеет вид $(f_0, f_1, \ldots)$, то $f_0(A) = 1$. Из (16.12) следует, что $f_0(A) = A^{\pi_0} = A^\pi = 1$.

- Обратно, пусть $\mathcal{L}(\Sigma) \subseteq \mathcal{L}(\Sigma_A)$, т.е. $\forall\, \pi \in \Pi^0_\Sigma\ \mathcal{L}(\pi) \in \mathcal{L}(\Sigma_A)$.

Докажем, что верно (16.21), т.е. $\forall\, \pi \in \Pi^0_\Sigma \quad A^\pi = 1$.

Из $\mathcal{L}(\pi) \in \mathcal{L}(\Sigma_A)$ следует, что $\exists\, \varphi \in \Pi^0_{\Sigma_A} : \mathcal{L}(\pi) = \mathcal{L}(\varphi)$. По лемме 4 из этого равенства следует, что $\varphi = \varphi_\pi$. Из (16.12) следует, что если $(f_0, f_1, \ldots) = \varphi = \varphi_\pi$, то $A^\pi = A^{\pi_0} = f_0(A) = 1$. ■

Теорема 30
Свойство (16.21) равносильно равенству

$$\mathcal{L}(\Sigma) \cap \mathcal{L}(\Sigma_{\bar{A}}) = \emptyset. \tag{16.23}$$

Доказательство.
Утверждение теоремы следует из леммы 5 и из соотношений

$$\begin{array}{l} \mathcal{L}(\Sigma_A) \cap \mathcal{L}(\Sigma_{\bar{A}}) = \emptyset, \\ \forall\, g = (g_0, g_1, \ldots) \in [A]^\infty \quad g \in \mathcal{L}(\Sigma_A) \;\text{ или }\; g \in \mathcal{L}(\Sigma_{\bar{A}}). \end{array} \tag{16.24}$$

Для доказательства этих соотношений введём следующие обозначения: если $g = (g_0, \ldots) \in [A]^\infty$, то

- запись Σ_g обозначает СП (S, R, L, S^0), где
  - $S = \{s_0, s_1, \ldots\}$, $R = \{(s_i, s_{i+1}) \mid i \geq 0\}$, $S^0 \stackrel{\text{def}}{=} \{s_0\}$,
  - $\forall\, i \geq 0, \forall\, p \in AP_A \;\; L(s_i, p) \stackrel{\text{def}}{=} g_i(p)$,
- символ π_g обозначает путь $(s_0, s_1, \ldots) \in \Pi_{\Sigma_g}$,
- записи $\varphi_{\pi_g} = (f_0, \ldots)$ и $\varphi'_{\pi_g} = (f'_0, \ldots)$ обозначают пути в Π_{Σ_A} и $\Pi_{\Sigma_{\bar{A}}}$ соответственно, соответствующие пути π_g.

Из (16.12) следуют равенства:

- $g = \mathcal{L}(\varphi_{\pi_g}) = \mathcal{L}(\varphi'_{\pi_g})$, т.к. $\forall\, i \geq 0,\ \forall\, p \in AP_A = AP_{\bar{A}}$
$$g_i(p) = L(s_i, p) = p^{(\pi_g)_i} = f_i(p) = f'_i(p),$$
- $f_0(A) = A^{\pi_g}$, $f'_0(\bar{A}) = (\bar{A})^{\pi_g}$, откуда следует равенство
$$\overline{f_0(A)} = f'_0(\bar{A}). \tag{16.25}$$

1. Обоснуем первое соотношение в (16.24). Пусть оно неверно, т.е.
$$\exists\, \varphi \in \Pi^0_{\Sigma_A},\ \exists\, \varphi' \in \Pi^0_{\Sigma_{\bar{A}}} :\ \mathcal{L}(\varphi) = \mathcal{L}(\varphi').$$
Обозначим $g \stackrel{\text{def}}{=} \mathcal{L}(\varphi) = \mathcal{L}(\varphi')$. Так как $f_0 \in S^0_{\Sigma_A}$ и $f'_0 \in S^0_{\Sigma_{\bar{A}}}$, то $f_0(A) = 1$ и $f'_0(\bar{A}) = 1$, что противоречит (16.25).

2. Обоснуем второе соотношение в (16.24).
Если $f_0(A) = 1$, то $g = \mathcal{L}(\varphi_{\pi_g}) \in \mathcal{L}(\Sigma_A)$. Если же $f_0(A) = 0$, то из (16.25) следует, что $f'_0(\bar{A}) = 1$, поэтому $g = \mathcal{L}(\varphi'_{\pi_g}) \in \mathcal{L}(\Sigma_{\bar{A}})$. ■

Таким образом, исходная задача доказательства (или опровержения) утверждения (16.21) сводится к проверке равенства (16.23). Проверка данного равенства м.б. выполнена на основе излагаемого в следующем пункте понятия автомата Бюхи.

16.4 Автоматы Бюхи

16.4.1 Понятие автомата Бюхи

Автомат Бюхи (называемый ниже просто **автоматом**) – это пятёрка

$$\mathcal{B} = (\mathcal{A}, S, R, S^0, \mathcal{F}), \tag{16.26}$$

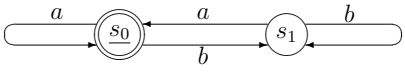
компоненты которой имеют следующий смысл:

- $\mathcal{A}$ – множество, называемое **алфавитом** автомата $\mathcal{B}$,
- S – множество, элементы которого называются **состояниями**,
- $R \subseteq S \times \mathcal{A} \times S$ – **отношение перехода**,
- $S^0 \subseteq S$ – множество **начальных состояний**,
- $\mathcal{F} = \{F_i \subseteq S \mid i = 1, \ldots, n\}$ – совокупность **условий справедливости**.

Элементы R называются **переходами**. Каждый переход (s, a, s') из R может обозначаться записью $s \xrightarrow{a} s'$.

Автомат (16.26) можно представить в виде графа (обозначаемого тем же символом $\mathcal{B}$), вершинами которого являются состояния этого автомата. $\forall\, (s \xrightarrow{a} s') \in R$ граф $\mathcal{B}$ содержит ребро с меткой a из s в s'.

Пример автомата:



(16.27)

Данный автомат имеет следующие компоненты:

- $\mathcal{A} = \{a, b\}$, $S = \{s_0, s_1\}$, $S^0 = \{s_0\}$ (начальные состояния обозначаются двойными кружочками),
- $\mathcal{F} = \{\{s_0\}\}$ (состояния из условий справедливости выделяются подчёркиванием).

16.4.2 Язык автомата

Пусть задан автомат $\mathcal{B} = (\mathcal{A}, S, R, S^0, \mathcal{F})$.

Будем использовать следующие понятия и обозначения: если $\pi = (s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} \ldots)$ – путь в $\mathcal{B}$, то

- первое состояние в π (т.е. s_0) называется **началом** пути π,
- запись $\inf(\pi)$ обозначает множество всех состояний, которые входят в π бесконечное число раз,
- π называется **справедливым путём**, если

$$\forall\, F \in \mathcal{F} \quad \inf(\pi) \cap F \neq \emptyset,$$

- запись $\mathcal{L}(\pi)$ обозначает последовательность меток рёбер, из которых состоит π, т.е. $\mathcal{L}(\pi) = (a_1, a_2, \ldots)$.

Для каждого состояния s автомата $\mathcal{B}$ запись Π_s обозначает множество всех справедливых путей, началом которых является s.

Язык автомата $\mathcal{B}$ – это множество

$$\mathcal{L}(\mathcal{B}) = \{\mathcal{L}(\pi) \mid \exists\, s \in S^0, \exists\, \pi \in \Pi_s\}.$$

Например, язык автомата (16.27) состоит из всех бесконечных конкатенаций $u_1 u_2 \ldots$, где $\forall\, i \geq 1$ – цепочка вида $b \ldots ba$, состоящая из ≥ 0 символов b, после которых следует символ a. Данный язык обозначается записью $(b^*a)^\infty$.

Теорема 31
Пусть задан автомат $\mathcal{B}$ вида (16.26), причем $\mathcal{F} = \{F\}$.
Следующие условия эквивалентны:

- $\mathcal{L}(\mathcal{B}) \neq \emptyset$,
- существует конечный путь из некоторого состояния $s_0 \in S^0$ в некоторую SCC $\mathbf{C}$ графа $\mathcal{B}$, такую, что $\mathbf{C} \cap F \neq \emptyset$,
- существует конечный путь из некоторого состояния $s_0 \in S^0$ в некоторое состояние $s \in F$, через которое проходит цикл графа $\mathcal{B}$. ■

16.4.3 Эквивалентность автоматов

Автоматы $\mathcal{B}_1$ и $\mathcal{B}_2$ называются **эквивалентными**, если $\mathcal{L}(\mathcal{B}_1) = \mathcal{L}(\mathcal{B}_2)$.

Теорема 32
Для каждого автомата (16.26) существует эквивалентный ему автомат

$$\mathcal{B}_1 = (\mathcal{A}, S_1, R_1, S_1^0, \mathcal{F}_1)$$

такой, что $\mathcal{F}_1$ состоит только из одного множества.

Доказательство.
Компоненты автомата $\mathcal{B}_1$ можно определить, например, так:

- $S_1 = S \times \{0, 1, \ldots, n\}$, $S_1^0 = S^0 \times \{0\}$, $\mathcal{F}_1 = \{S \times \{n\}\}$,
- R_1 состоит из переходов $(s, j) \xrightarrow{a} (s', j')$, таких, что $s \xrightarrow{a} s'$ и

$$j' = \begin{cases} k, & \text{если } s' \in F_k \text{ и } j = k - 1, \\ 0, & \text{если } j = n, \\ j, & \text{в остальных случаях.} \end{cases} \blacksquare$$

Ниже для каждого автомата, у которого вид совокупности $\mathcal{F}$ его условий справедливости не описан явно, будем предполагать, что

- совокупность $\mathcal{F}$ состоит из одного множества и
- эта совокупность будет обозначаться той же записью, что и то единственное множество, из которого состоит $\mathcal{F}$.

16.4.4 Пересечение автоматов

Теорема 33
Для каждой пары автоматов $\mathcal{B}_1, \mathcal{B}_2$ с общим алфавитом, где

$$\mathcal{B}_i = (\mathcal{A}, S_i, R_i, S_i^0, F_i) \qquad (i = 1, 2),$$

существует автомат $\mathcal{B}_1 \cap \mathcal{B}_2 = (\mathcal{A}, S, R, S^0, F)$, называемый **пересечением** $\mathcal{B}_1$ и $\mathcal{B}_2$, и обладающий следующим свойством:

$$\mathcal{L}(\mathcal{B}_1 \cap \mathcal{B}_2) = \mathcal{L}(\mathcal{B}_1) \cap \mathcal{L}(\mathcal{B}_2). \tag{16.28}$$

Доказательство.
Компоненты автомата $\mathcal{B}_1 \cap \mathcal{B}_2$ можно определить, например, так:

- $S = S_1 \times S_2 \times \{0, 1, 2\}$, $S^0 = S_1^0 \times S_2^0 \times \{0\}$, $F = S_1 \times S_2 \times \{2\}$,
- R состоит из переходов $(s_1, s_2, j) \xrightarrow{a} (s_1', s_2', j')$, таких, что

$$s_i \xrightarrow{a} s_i' \quad (i = 1, 2), \quad j' = \begin{cases} 1, & \text{если } j = 0 \text{ и } s_1' \in F_1, \\ 2, & \text{если } j = 1 \text{ и } s_2' \in F_2, \\ 0, & \text{если } j = 2, \\ j, & \text{в остальных случаях.} \end{cases}$$

Если $F_1 = S_1$, то компоненты автомата $\mathcal{B}_1 \cap \mathcal{B}_2$ можно определить проще:

- $S = S_1 \times S_2, \quad S^0 = S_1^0 \times S_2^0, \quad F = S_1 \times F_2,$
- $R \stackrel{\text{def}}{=} \{(s_1, s_2) \xrightarrow{a} (s_1', s_2') \mid s_i \xrightarrow{a} s_i' \quad (i = 1, 2)\}$. $\blacksquare$

16.4.5 Использование автоматов Бюхи для MC-LTL

Лемма 6

Пусть заданы СП Σ и LTL-формула A.

Имеет место равенство $\mathcal{L}(\Sigma) = \mathcal{L}(\mathcal{B}_\Sigma)$, где $\mathcal{B}_\Sigma = (\mathcal{A}, S, R, S^0, \mathcal{F})$ – автомат Бюхи, компоненты которого определяются следующим образом:

$$\mathcal{A} = [A],\ \ S = S_\Sigma \sqcup \{init\},\ \ S^0 = \{init\},\ \ \mathcal{F} = \mathcal{F}_\Sigma,$$
$$R = \{s \xrightarrow{g_{s'}} s' \mid (s \to s') \in R_\Sigma\} \cup \{init \xrightarrow{g_s} s \mid s \in S^0_\Sigma\}$$

где $\forall\, s \in S_\Sigma, \forall\, p \in AP_A\ \ g_s(p) = s(p)$. ■

Основанный на данной лемме метод проверки соотношения (16.23) заключается в построении автоматов $\mathcal{B}_\Sigma$, $\mathcal{B}_{\Sigma_{\bar{A}}}$ и проверке пустоты языка автомата $\mathcal{B}_\Sigma \cap \mathcal{B}_{\Sigma_{\bar{A}}}$. Согласно теореме 31, язык этого автомата непуст тогда и только тогда, когда существует путь из его начального состояния в какое-либо справедливое состояние, через которое проходит цикл.

Автомат $\mathcal{B}_\Sigma \cap \mathcal{B}_{\Sigma_{\bar{A}}}$ можно строить «на лету» («on-the-fly»): сначала строится $\mathcal{B}_{\Sigma_{\bar{A}}}$, который используется в процессе построения $\mathcal{B}_\Sigma$. Пусть состояниями $\mathcal{B}_\Sigma \cap \mathcal{B}_{\Sigma_{\bar{A}}}$ являются пары $(s, s_{\bar{A}})$, где s – состояние $\mathcal{B}_\Sigma$ и $s_{\bar{A}}$ – состояние $\mathcal{B}_{\Sigma_{\bar{A}}}$. Если уже построено некоторое состояние s автомата $\mathcal{B}_\Sigma$, то к тому фрагменту автомата $\mathcal{B}_\Sigma$, который уже построен, добавляются только такие состояния s', что

$$\exists\, a \in \mathcal{A}, \exists\, s'_{\bar{A}} : \quad s \xrightarrow{a} s', s_{\bar{A}} \xrightarrow{a} s'_{\bar{A}}.$$

Если построена часть $\mathcal{B}_\Sigma \cap \mathcal{B}_{\Sigma_{\bar{A}}}$, содержащая путь, упомянутый в теореме 31, то в дальнейшем построении $\mathcal{B}_\Sigma$ уже нет необходимости.

16.4.6 Оптимизация построения автомата $\mathcal{B}_{\bar{A}}$

Автомат $\mathcal{B}_{\bar{A}}$ можно строить не по $\Sigma_{\bar{A}}$, а по более компактной СП Σ, такой, что $\mathcal{L}(\Sigma) = \mathcal{L}(\Sigma_{\bar{A}})$. Ниже излагается алгоритм построения такой СП.

Будем использовать соглашение: $\forall\, B, C \in Fm_{LTL}$ запись $\mathbf{R}(B, C)$ рассматривается как LTL-формула, которая по определению равна формуле $\overline{\mathbf{U}(\overline{B}, \overline{C})}$. Из этого соглашения следует, что $\forall\, B, C \in Fm_{LTL}$

$$\begin{aligned} &\bar{*} = \mathbf{R}(\bar{B}, \bar{C}), \\ &\mathbf{R}(B, C) = C \wedge \Big(B \vee \mathbf{XR}(B, C)\Big). \end{aligned} \tag{16.29}$$

Преобразуем $\bar{A}$ в эквивалентную ей формулу $\tilde{A}$, в которой отрицания располагаются только над атомарными утверждениями, используя законы де Моргана и равенства $\overline{\mathbf{X}B} = \mathbf{X}\bar{B}$ и $\bar{*} = \mathbf{R}(\bar{B}, \bar{C})$.

Далее строим СП, каждое состояние которой является подмножеством множества $\langle \tilde{A} \rangle$. На каждом шаге построения данная СП является аппроксимацией искомой СП Σ. Мы будем обозначать одним и тем же символом Σ данную СП на каждом шаге ее построения. В конце построения данная СП будет представлять собой искомую СП. На каждом шаге построения СП Σ должны быть выполнены следующие условия:

- $\forall\, s \in S_\Sigma$ указано разбиение $s = \mathrm{New}(s) \sqcup \mathrm{Old}(s)$,
- $\forall\, s \in S_\Sigma,\ \forall\, p \in AP_A \quad L_\Sigma(s,p)$ равно 1, если $p \in s$, и 0 – если $\bar{p} \in s$,
- $\mathcal{F}_\Sigma = \{F_* \mid * \in \langle \tilde{A} \rangle\}$, где $\forall * \in \langle \tilde{A} \rangle \quad F_* \stackrel{\text{def}}{=} \{s \in S \mid * \in s \ \Rightarrow\ C \in s\}$,
- $\forall\, s \in S_\Sigma,\ \forall\, B \in s,\ \forall\, \pi \in \Pi_s \quad \pi(B) = 1$.

Сначала $S_\Sigma \stackrel{\text{def}}{=} \{\tilde{A}\} = \mathrm{New}(s_0)$.

На очередном шаге построения выбираем произвольное $s \in S_\Sigma$, такое, что $\mathrm{New}(s) \neq \emptyset$ (если таких s нет, то построение закончено), и выполняем одно из двух следующих действий:

1. если $\exists\, D \in \mathrm{New}(s)$, не начинающаяся с $\mathbf{X}$, то переносим D из $\mathrm{New}(s)$ в $\mathrm{Old}(s)$, после чего
   (a) если $D = B \wedge C$, то добавляем B и C к $\mathrm{New}(s)$;
   (b) если $D = B \vee C$, то
      - добавляем дубликат s' состояния s (с теми же New и Old) и для каждого ребра, ведущего в s, добавляем новое ребро с тем же началом, но с концом в s',
      - добавляем B к $\mathrm{New}(s)$, C к $\mathrm{New}(s')$;
   (c) если $D = *$, то обрабатываем её как $C \vee \Big(B \wedge \mathbf{X} * \Big)$, т.е.
      - добавляем дубликат s' состояния s и рёбра в s',
      - добавляем C к $\mathrm{New}(s)$, B и $\mathbf{X}*$ к $\mathrm{New}(s')$;
   (d) если $D = \mathbf{R}(B,C)$, то обрабатываем её как $C \wedge \Big(B \vee \mathbf{X}D\Big)$, т.е.
      - добавляем дубликат s' состояния s и рёбра в s',
      - добавляем C и B к $\mathrm{New}(s)$, C и $\mathbf{X}D$ к $\mathrm{New}(s')$,

   после чего
   - если s содержит $\bot$ или пару формул вида $p, \bar{p}$, то удаляем s и все ведущие в него рёбра,

- если в s входит $\top$, то удаляем $\top$ из s,
- если в s входит пара одинаковых формул, то удаляем ту из них, которая входит в $\mathrm{New}(s)$,
- в случаях (b–d) выполняем те же действия для s';

2. если все формулы в $\mathrm{New}(s)$ начинаются с $\mathbf{X}$, то
   (a) если $\exists\, s' \neq s$: $\mathrm{New}(s) \subseteq \mathrm{New}(s')$ и $\mathrm{Old}(s) = \mathrm{Old}(s')$, то удаляем s и перенаправляем в s' все рёбра, ведущие в s,
   (b) иначе добавляем $s' = \{B \mid \mathbf{X}B \in \mathrm{New}(s)\} = \mathrm{New}(s')$ и ребро из s в s' и удаляем все формулы из $\mathrm{New}(s)$.

После завершения построения S_Σ полагаем $S_\Sigma^0 \stackrel{\text{def}}{=} \{s \in S_\Sigma \mid \tilde{A} \in s\}$.

Глава 17

Вероятностный model checking

17.1 Введение

Наряду с изложенными выше моделями дискретных динамических систем используются и другие модели, среди которых наибольшую популярность получили **вероятностные системы переходов (ВСП)**.

ВСП являются эффективным средством моделирования таких РП, в которых присутствуют ненадежные компоненты, выходящие из строя с некоторой вероятностью, или возникают различные случайные события (например, потеря сообщений в каналах связи), или в алгоритмах явно присутствует рандомизация.

В этой главе мы будем изучать логический язык описания свойств ВСП и методы их анализа, называемые **вероятностным model checking** (**probabilistic model checking, PMC**). В настоящее время PMC является одним из наиболее широко используемых методов моделирования и верификации вычислительных систем.

Понятие ВСП является обобщением понятия цепи Маркова [36], которое имеет широкие применения в естественных и гуманитарных науках. Понятие ВСП можно рассматривать также как частный случай понятия вероятностного автомата [37].

Одной из главных причин актуальности ВСП в задачах верификации РП является существенно меньшая (по сравнению с моделями в виде СП) сложность вероятностных моделей анализируемых РП. Однако построение и анализ моделей анализируемых РП в виде ВСП является нетривиальной задачей по следующим причинам:

- для получения численных значений вероятностей переходов в модели анализируемой РП в виде ВСП необходимо проведение достаточно трудоемких экспериментов с исходной РП;

- свойства модели в виде ВСП могут отличаться от свойств исходной РП, это приводит к проблеме оценки меры соответствия свойств исходной РП и свойств ее модели в виде ВСП.

Первые алгоритмы PMC были предложены в 1980-е годы в работах [38], [39], [40]. Данные алгоритмы были предназначены для верификации качественных вероятностных свойств (то есть таких, которые выполняются с вероятностью 1 или 0). Затем эти алгоритмы были обобщены на случай верификации количественных вероятностных свойств. Первые промышленные системы PMC были разработаны в 2000-х годах. Эти системы успешно применяются во многих областях, в том числе анализе распределенных алгоритмов, телекоммуникационных протоколов, компьютерной безопасности, криптографических протоколах, моделировании биологических процессов. С использованием этих систем PMC были обнаружены уязвимости и аномальные поведения анализируемых систем, некоторые подробности см. в [41] и [42]. При помощи систем PMC могут быть вычислены такие характеристики программных систем, как, например, вероятность вторжения злоумышленника в компьютерную сеть, мат. ожидание времени отклика веб-сервиса, и другие количественные и качественные характеристики.

Наиболее популярной практической системой PMC является система PRISM [43], [42], разработанная на факультете компьютерных наук Оксфордского университета (Великобритания) в группе Quantitative Analysis and Verification под руководством Марты Квятковской [44].

17.2 Вероятностные системы переходов

17.2.1 Понятие вероятностной системы переходов

Вероятностная система переходов (ВСП) (называемая также **Discrete Time Markov Chain**) – это четверка Σ вида

$$\Sigma = (S, P, L, s^0), \tag{17.1}$$

компоненты которой имеют следующий смысл:

- S – множество **состояний** ВСП Σ,
- P – функция вида $P : S \times S \to [0, 1]$, называемая **функцией переходов** ВСП Σ и удовлетворяющая условию:

$$\forall\, s \in S \quad \sum_{s' \in S} P(s, s') = 1,$$

- L – функция вида $L : S \times AP \to \{0, 1\}$, называемая **оценкой**,
- $s^0 \in S$ – **начальное состояние** ВСП Σ.

$\forall\, (s_1, s_2) \in S \times S$ число $P(s_1, s_2)$ понимается как вероятность того, что если в текущий момент времени Σ находится в состоянии s_1, то через один такт времени Σ будет находиться в состоянии s_2.

Оценка L имеет следующий смысл: $\forall\, s \in S$, $\forall\, p \in AP$ утверждение p считается **истинным** в s, если $L(s, p) = 1$, и **ложным** иначе.

ВСП $\Sigma = (S, P, L, s^0)$ удобно рассматривать как граф (обозначаемый тем же символом Σ), в котором

- вершинами являются состояния из S и
- для каждой пары $(s_1, s_2) \in S \times S$ такой, что $P(s_1, s_2) > 0$, имеется ребро из s_1 в s_2 с меткой $P(s_1, s_2)$.

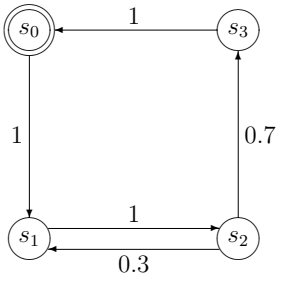
Рёбра данного графа будем называть **переходами** ВСП Σ.

17.2.2 Примеры вероятностных систем переходов

1. Упрощённая модель протокола передачи сообщений через ненадёжный канал, в котором сообщения могут пропадать. Протокол представляет собой систему, состоящую из

   - двух агентов – отправителя и получателя, а также
   - канала, в который помещаются сообщения, пересылаемые от одного агента другому.

   Граф, представляющий эту ВСП, имеет следующий вид:



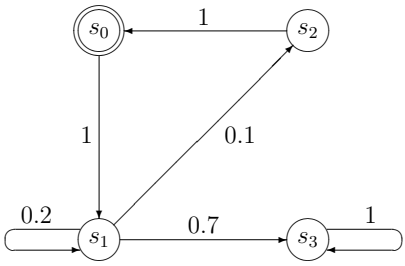
(17.2)

   Переходы этой ВСП имеют следующий смысл.

- Переход $s_0 \xrightarrow{1} s_1$ заключается в получении отправителем от внешнего источника сообщения, которое должно быть передано через канал получателю.
- Переход $s_1 \xrightarrow{1} s_2$ заключается в помещении сообщения в канал отправителем.
- Переход $s_2 \xrightarrow{0.3} s_1$ заключается в потере сообщения в канале.
- Переход $s_2 \xrightarrow{0.7} s_3$ заключается в передаче сообщения из канала получателю.
- Переход $s_3 \xrightarrow{1} s_0$ заключается в получении сообщения получателем и посылке им подтверждения отправителю.

2. Упрощённая модель агента, работа которого представляет собой последовательность сеансов. В каждом из этих сеансов агент пытается выполнить некоторое действие. Если действие было выполнено, то он переходит к следующему сеансу, а если не было выполнено, то он завершает свою работу.

   Граф, представляющий эту ВСП, имеет следующий вид:



   Состояния этой ВСП имеют следующий смысл:

   - В состоянии s_0 агент начинает очередной сеанс.
   - В состоянии s_1 агент предпринимает попытку выполнения действия (переход $s_1 \xrightarrow{0.2} s_1$ означает, что попытка выполнения действия пока не осуществима).
   - В состояние s_2 агент попадает тогда, когда действие было выполнено успешно.
   - В состоянии s_3 агент оказывается тогда, когда его попытка выполнить действие закончилась неудачей.

17.3 Темпоральная логика PCTL

17.3.1 Свойства вероятностных систем переходов

Одним из логических языков, предназначенных для формального описания свойств поведения ВСП, является темпоральная логика **PCTL** (**Probabilistic Computation Tree Logic**). Логика PCTL была введена Х. Ханссоном (H. Hansson) и Б. Джонссоном (B. Jonsson) в работе [45].

Формулы логики PCTL могут отражать различные вероятностные аспекты поведения систем, к числу которых относятся, например:

- частота выполнения тех или иных действий или переходов,
- вероятность отказа компонентов систем,
- вероятностный характер взаимодействия системы с её окружением, например частота поступления входных запросов или сообщений, частота получения искажённых сообщений (для протоколов передачи сообщений в компьютерных сетях) и т.п.

Примеры свойств, которые можно описать формулами PCTL:

- вероятность доставки сообщения в заданном временном интервале $[t_1, t_2]$ не меньше 0.999,
- в результате работы алгоритма избрания процесса-лидера избрание такого процесса завершится с вероятностью 1,
- вероятность успешной атаки на криптографический протокол не превышает 0.0001,
- вероятность отклика веб-сервиса в течение 5ms – не менее 0.8,
- ожидаемое время в худшем случае для доставки пакета данных при помощи протокола беспроводной связи – 1 с,
- максимальное ожидаемое энергопотребление за 24 часа для устройства с электропитанием от батареи – не более 190J,
- вероятность того, что система после любого отказа восстановится за $\leq n$ шагов, не менее $1 - \varepsilon$,
- вероятность того что сигнал ready будет получен в течение следующих n единиц времени, больше половины.

17.3.2 Формулы логики PCTL

Совокупность **формул** логики PCTL, называемых также **PCTL-формулами**, обозначается записью Fm_{PCTL}. Формулы логики PCTL делятся на два класса: Fm^{state}_{PCTL} (**state-формулы**) и Fm^{path}_{PCTL} (**path-формулы**), определяемые следующим образом:

- $AP \subseteq Fm^{state}_{PCTL}$,
- Fm^{state}_{PCTL} замкнут относительно булевых комбинаций,
- $\forall\, A \in Fm_{PCTL}$ запись $\mathbf{X}A$ является path-формулой,
- $\forall\, A, B \in Fm^{state}_{PCTL},\ \forall\, n \geq 0$ записи $\mathbf{U}^{\leq n}(A, B)$ и $\mathbf{U}(A, B)$ являются path-формулами,
- $\forall\, A \in Fm^{path}_{PCTL},\ \forall\, a \in [0, 1]$ записи $[\![A]\!]_{\geq a}$, $[\![A]\!]_{>a}$, $[\![A]\!]_{\leq a}$ $[\![A]\!]_{<a}$, являются state-формулами.

$\forall\, A \in Fm^{state}_{PCTL},\ \forall\, n \geq 0$ записи $\mathbf{F}^{\leq n}A$ и $\mathbf{F}A$ обозначают формулы $\mathbf{U}^{\leq n}(\top, A)$ и $\mathbf{U}(\top, A)$ соответственно, и записи $\mathbf{G}^{\leq n}A$ и $\mathbf{G}A$ обозначают формулы $\overline{\mathbf{F}^{\leq n}\overline{A}}$ и $\overline{\mathbf{F}\ \overline{A}}$ соответственно.

17.3.3 Значения формул логики PCTL в состояниях вероятностных систем переходов

Пусть задана ВСП $\Sigma = (S, P, L, s^0)$.

$\forall\, s \in S,\ \forall\, A \in Fm_{PCTL}$ ниже определяется **значение** формулы A в состоянии s, которое обозначается записью A^s, и

- если $A \in Fm^{state}_{PCTL}$, то $A^s \in \{0, 1\}$,
- если $A \in Fm^{path}_{PCTL}$, то $A^s \in [0, 1]$, A^s интерпретируется как вероятность того, что формула A истинна в состоянии s.

В излагаемом ниже определении значений вида A^s будем использовать следующие обозначения. Пусть S имеет вид $\{s_1, \ldots, s_n\}$.

- $\forall\, A \in Fm_{PCTL}$ будем обозначать записью A^S столбец $\begin{pmatrix} A^{s_1} \\ \ldots \\ A^{s_n} \end{pmatrix}$.

- Для любых столбцов $U = \begin{pmatrix} u_1 \\ \dots \\ u_n \end{pmatrix}$, $V = \begin{pmatrix} v_1 \\ \dots \\ v_n \end{pmatrix}$ из $[0,1]^n$ записи $\max(U,V)$ и UV обозначают столбцы $\begin{pmatrix} \max(u_1,v_1) \\ \dots \\ \max(u_n,v_n) \end{pmatrix}$ и $\begin{pmatrix} u_1 v_1 \\ \dots \\ u_n v_n \end{pmatrix}$ соответственно.

- Символ P обозначает матрицу порядка $n \times n$, в которой элемент в i-й строке и j-м столбце равен $P(s_i, s_j)$.

- Для любого столбца $U \in [0,1]^n$ запись $[P^* U]$ обозначает столбец из $\{0,1\}^n$, получаемый заменой всех ненулевых компонентов матрицы P и столбца U на 1 и вычислением $(\sum_{i \geq 0} P^i)U$, где сложение понимается как дизъюнкция (т.е. $1 + 1 = 1$ и сумма $\sum_{i \geq 0} P^i$ конечна).

Значения формул логики PCTL в состояниях ВСП определяются индукцией по структуре формул в соответствии с излагаемыми ниже правилами. В одних из этих правил мы определяем значение A^s, в других определяем столбец A^S целиком.

- $\forall\, s \in S,\ \forall\, p \in AP \quad p^s = L(s,p)$.

- $\forall\, s \in S \ \ \top^s = 1,\ \bot^s = 0$.

- Значения булевых комбинаций определяются стандартным образом: $\forall\, s \in S \ \ (\bar{A})^s = \overline{A^s}, (A \wedge B)^s = A^s \wedge B^s$ и т.д.

- $(\mathbf{X}A)^S = PA^S$ (произведение матрицы P на столбец A^S).

- $(\mathbf{U}^{\leq 0}(A,B))^S = B^S$,
  $\forall\, n > 0 \quad (\mathbf{U}^{\leq n}(A,B))^S = \max\Big(B^S,\ A^S(\mathbf{X}\mathbf{U}^{\leq n-1}(A,B))^S\Big)$.

- $(\mathbf{U}(A,B))^S$ определяется системой линейных уравнений

$$(\mathbf{U}(A,B))^S = \max\Big(B^S,\ [P^* B^S] A^S (P(\mathbf{U}(A,B))^S)\Big).$$

- $\forall\, s \in S \ \ (\llbracket A \rrbracket_{\geq a})^s = \llbracket A^s \geq a \rrbracket$, т.е. $(\llbracket A \rrbracket_{\geq a})^s = \begin{cases} 1, \text{ если } A^s \geq a, \\ 0 \text{ иначе,} \end{cases}$

- аналогичное определение для формул вида $\llbracket A \rrbracket_{>a}$, $\llbracket A \rrbracket_{\leq a}$, $\llbracket A \rrbracket_{<a}$.

17.3.4 Интерпретация значений формул логики PCTL

Пусть задана ВСП Σ. $\forall\, s \in S_\Sigma$, $\forall\, A \in Fm_{PCTL}$ значение A^s интерпретируется следующим образом:

- если $A \in Fm^{state}_{PCTL}$, то формула A истинна в s при $A^s = 1$ и ложна в s при $A^s = 0$,
- если $A = \mathbf{X}B$, то A^s интерпретируется как математическое ожидание значения формулы B в том состоянии, в которое будет совершён переход из состояния s за один такт времени,
- если $A = \mathbf{U}^{\leq n}(B, C)$, то A^s интерпретируется как вероятность того, что для произвольного пути π в графе Σ, выходящего из s, существует состояние s' на этом пути, такое, что длина отрезка от s до s' пути π не превосходит n, и
  - в каждом состоянии этого отрезка, кроме м.б. s', истинна B,
  - в состоянии s' истинна C,
- если $A = \mathbf{U}(B, C)$, то A^s интерпретируется так же, как значение предыдущей формулы, без упоминания того, что длина пути из s в s' не превосходит n.

На основе этой интерпретации можно описывать свойства ВСП формулами логики PCTL. Эти свойства могут выражать динамические аспекты поведения ВСП, т.е. описывать зависимость значения какого-либо утверждения в некотором состоянии рассматриваемой ВСП от значений других утверждений в других состояниях этой ВСП. Например, свойство протокола из пункта 17.2.2, представленного ВСП (17.2):

> каждое сообщение, полученное отправителем от внешнего источника, будет доставлено получателю за не более чем 5 шагов с вероятностью ≥ 0.9,

выражается формулой $\mathbf{G}\Big(p_0 \to [\![\mathbf{F}^{\leq 5} p_3]\!]_{\geq 0.9}\Big)$, где $s_i(p_j) = [\![i = j]\!]$.

Глава 18

Задачи и исследовательские проблемы

18.1 Задачи

1. Доказать, что результатом алгоритма оптимизирующих преобразований из пункта 15.5 является BDD, эквивалентная исходной BDD.

2. Доказать, что

   - CTL-формула $\mathbf{AG}(\mathbf{EF}p)$ не эквивалентна ни одной квантифицированной LTL-формуле,
   - квантифицированная LTL-формула $\mathbf{A}(\mathbf{FG}p)$ не эквивалентна ни одной CTL-формуле,
   - дизъюнкция приведённых выше формул не эквивалентна ни одной квантифицированной LTL-формуле и ни одной CTL-формуле.

3. Доказать теорему 31.

4. Доказать, что автомат Бюхи $\mathcal{B}_1$, определённый в доказательстве теоремы 32, эквивалентен исходному автомату Бюхи (16.26).

5. Доказать, что автомат Бюхи $\mathcal{B}_1 \cap \mathcal{B}_2$, определённый в доказательстве теоремы 33, обладает свойством (16.28).

6. Доказать лемму 6 в пункте 16.4.5.

7. Доказать, что СП Σ, алгоритм построения которой изложен в пункте 16.4.6, обладает свойством $\mathcal{L}(\Sigma) = \mathcal{L}(\Sigma_{\bar{A}})$.

8. Доказать что определяемое в пункте 17.3.3 значение $A^s \in [0,1]$ формулы $A \in Fm^{path}_{PCTL}$ действительно является вероятностью того, что для произвольного пути π в графе Σ, выходящего из s, формула A истинна на π.

18.2 Исследовательские проблемы

За пределами рассмотрений в данной части остались такие важные разделы Model Checking, как методы редукции (т.е. сокращения размеров) моделей РП, называемые **редукцией частичных порядков (partial order reduction)**, и модели РП с учетом длительности исполняемых действий (**timed automata**). Представление о данных направлениях можно получить, например, в упомянутой выше книге [32]. С данными направлениями связаны актуальные исследовательские проблемы: построение новых методов редукции моделей анализируемых РП, с сохранением проверяемых свойств у редуцированных моделей, и построение новых моделей систем с учетом различных сложностных характеристик выполняемых действий, методов описания их свойств и методов их верификации.

Также имеют высокую актуальность следующие проблемы:

1. Рассмотрение в качестве языков спецификаций таких темпоральных логик, в которых присутствуют темпоральные операторы не только в «будущее», но и в «прошлое», построение методов верификации свойств, выражаемых в таких темпоральных логиках.

2. Нахождение новых форм представления булевых функций, обладающих теми же свойствами, что и BDD (возможность выполнения булевых операций и распознавания эквивалентности), сложность выполнения операций с которыми была бы существенно меньше, чем сложность выполнения операций с BDD.

3. Построение ускоренных алгоритмов вычисления неподвижных точек операторов, используемых для MC-CTL на основе понятия неподвижной точки.

4. Разработка новых языков спецификаций РП, учитывающих различные (в том числе нечисловые) меры неопределённости, доверия, правдоподобия и т.п.

5. Построение аналогов понятия автомата Бюхи для верификации вероятностных систем переходов и разработка на основе этого поня-

тия методов построения фрагментов вероятностных систем переходов, содержащих контрпример к проверяемой спецификации (т.е. вероятностное обобщение построения частичных систем переходов, связанных с построением автомата $\mathcal{B}_\Sigma \cap \mathcal{B}_{\Sigma_{\bar{A}}}$ в пункте 16.4.5).

6. Построение моделей гибридных программно-аппаратных систем (в составе которых могут быть дискретные компоненты, непрерывные компоненты, вероятностные компоненты, нечёткие компоненты, нейросетевые компоненты, компоненты с криптографическими преобразованиями и т.п.), разработка методов идентификации, спецификации и верификации таких систем.

Часть IV

Теория процессов

Глава 19

Понятие процесса

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**, см. [46]–[49].

19.1 Неформальное понятие процесса и примеры процессов

Прежде чем сформулировать формальное понятие процесса, мы опишем данное понятие неформально и рассмотрим простейшие примеры процессов.

19.1.1 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.

19.1.2 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.

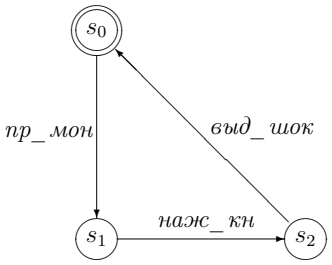
Обозначим действия короткими именами:

- 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:



(19.1)

Диаграмма (19.1) 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.

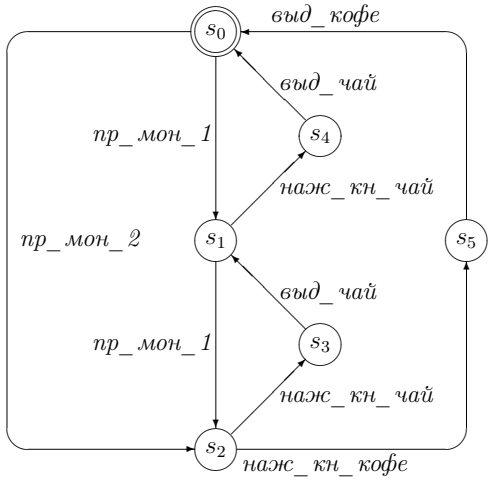
19.1.3 Другой пример процесса

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 19.2.

19.2 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. $name(?\alpha) = name(!\alpha) = \alpha$,
- $\forall\, L \subseteq Act \setminus \{\tau\}\ \ \overline{L} \stackrel{\text{def}}{=} \{\bar{a} \mid a \in L\}$, $names(L) \stackrel{\text{def}}{=} \{name(a) \mid a \in L\}$.

19.3 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 $s_1 \stackrel{a}{\to} s_2$.

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 $s_1 \stackrel{a}{\to} s_2$, 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 $s^0 \stackrel{a_0}{\longrightarrow} s_1 \stackrel{a_1}{\longrightarrow} s_2 \stackrel{a_2}{\longrightarrow} \ldots$ и выполнении действий $a_0, a_1, a_2 \ldots$, 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 состояние 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:

$$Act(P) = \{a \in Act \setminus \{\tau\} \mid \exists\, s \stackrel{a}{\to} s' \in R\}.$$

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 $s \xrightarrow{a} s' \in R$.

Глава 20

Операции на процессах

В этой главе мы определим операции на процессах, при помощи которых из одних процессов можно строить другие, более сложные процессы.

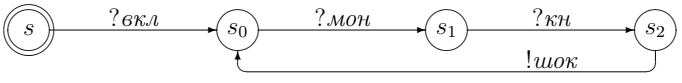
20.1 Префиксное действие

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 19.1.2. 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

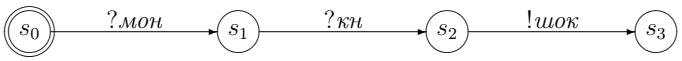


20.2 Пустой процесс

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) **0**.

Возвращаясь к примерам с торговыми автоматами, можно сказать, что процесс **0** представляет поведение сломанного автомата, то есть такого автомата, который вообще не может выполнять никаких действий.

Путем применения операций префиксного действия к процессу **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.

20.3 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

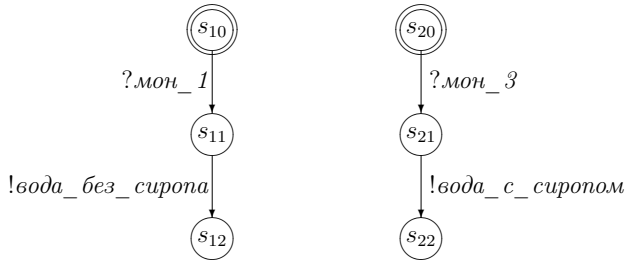
- если покупатель опускает в него монету *мон_ 1* достоинством в 1 копейку, то автомат выдаёт стакан воды без сиропа,
- а если покупатель опускает в него монету *мон_ 3* достоинством в 3 копейки, то автомат выдаёт стакан воды с сиропом

и сразу после продажи одного стакана воды автомат ломается.

Поведение данного автомата описывается следующим процессом:

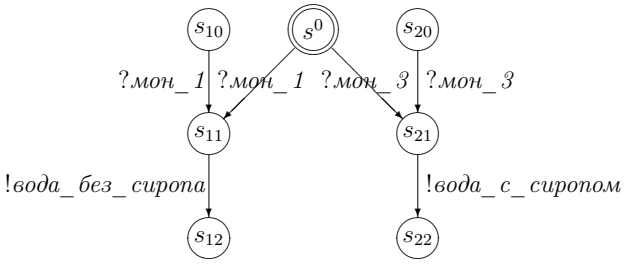
$$\begin{array}{rcl} P_{\text{газ\_вода}} & = & ?\textit{мон\_ 1}; !\textit{вода\_ без\_ сиропа}; \mathbf{0} \;+ \\ & + & ?\textit{мон\_ 3}; !\textit{вода\_ с\_ сиропом}; \mathbf{0} \end{array} \qquad (20.1)$$

Графовые представления слагаемых в сумме (20.1) имеют вид

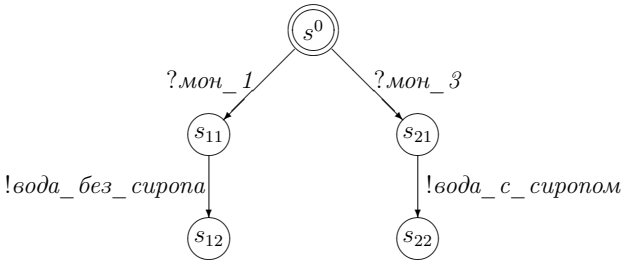


Согласно определению альтернативной композиции, графовое представление процесса (20.1) получается добавлением к предыдущей диаграмме нового состояния и переходов из этого состояния, дублирующих

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 (20.1).

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.

20.4 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

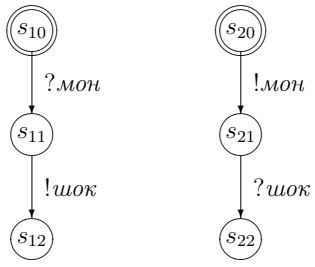
- либо одиночное выполнение действия одним из процессов P_1, P_2, во время выполнения которого другой из этих процессов приостанавливает свою работу,
- либо одновременное совместное выполнение действий обоими процессами P_1, P_2, которое заключается в том, что
  - один из этих процессов (P_i) передаёт 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**.

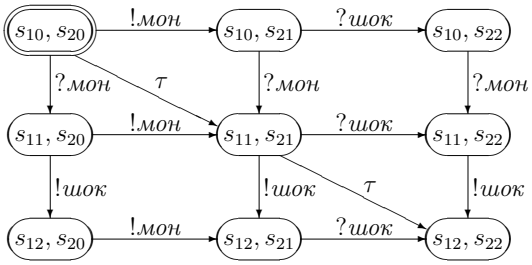
Приведём в качестве примера процесс $P_1 \,|\, P_2$, где процессы P_1 и P_2 представляют поведение торгового автомата и покупателя:

- $P_1 = ?\mathit{мон}; !\mathit{шок}; \mathbf{0}$, 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).

20.5 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.

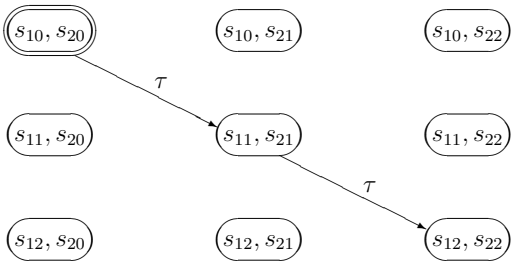
Операция ограничения используется, как правило, совместно с операцией параллельной композиции, для представления процессов, состоящих из нескольких компонентов, взаимодействие между которыми должно удовлетворять ограничениям. Эти ограничения заключаются в невозможности одиночного выполнения некоторых 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.

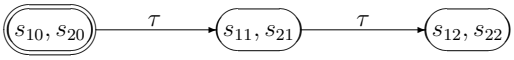
Искомый процесс имеет вид

$$P \stackrel{\text{def}}{=} (P_1|P_2) \setminus \{\textit{мон}, \textit{шок}\} \tag{20.2}$$

Графовое представление процесса (20.2) имеет вид



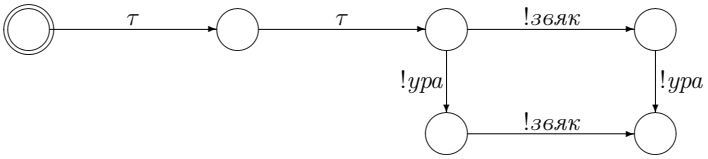
После удаления недостижимых состояний получится процесс, имеющий следующее графовое представление:



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 (20.2) после удаления недостижимых состояний имеет вид



Данный процесс допускает одиночное выполнение тех невнутренних действий, которые не связаны с покупкой и продажей шоколадки.

20.6 Переименование

Пусть заданы процесс $P = (S, s^0, R)$ и функция $\theta : Names \to Names$.

Переименованием P относительно θ называется процесс

$$P^\theta = (S, s^0, \{s \xrightarrow{a^\theta} s' \mid s \xrightarrow{a} s' \in R\}),$$

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 $P(\beta_1/\alpha_1, \ldots, \beta_n/\alpha_n)$.

20.7 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

$$P_1 + \ldots + P_n, \tag{20.3}$$

так как при любой расстановке скобок в выражении (20.3) получится один и тот же процесс. Данный процесс можно описать явно следущим образом. Пусть процессы P_i $(i = 1, \ldots, n)$ имеют вид

$$P_i = (S_i, s_i^0, R_i) \quad (i = 1, \ldots, n), \tag{20.4}$$

причём множества $S_1, \ldots, S_n$ дизъюнктны. Тогда (20.3) 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. $(P_1 \,|\, P_2) \,|\, P_3 = P_1 \,|\, (P_2 \,|\, P_3)$.

   Из данного свойства следует, что допустимы выражения вида

   $$P_1 \,|\, \ldots \,|\, P_n, \tag{20.5}$$

   так как при любой расстановке скобок в выражении (20.5) получится один и тот же процесс. Данный процесс можно описать явно следущим образом. Пусть процессы P_i $(i = 1, \ldots, n)$ имеют вид (20.4), тогда процесс (20.5) 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. $P_1 \,|\, P_2 = P_2 \,|\, P_1$.
5. $P \,|\, \mathbf{0} = P$.
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. $P^{\theta\theta'} = P^(\theta' \circ \theta)$.

Глава 21

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 21.1 и 21.2 вводятся понятия простой и сильной эквивалентности процессов. Данные понятия используются в тех ситуациях, когда все действия, выполняющиеся в процессах, имеют одинаковый статус. В параграфах 21.3 и 21.4 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.

21.1 Простая эквивалентность

21.1.1 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 $P \stackrel{a}{\rightarrow} P'$ обозначает утверждение: в P имеется ребро вида $s^0 \stackrel{a}{\rightarrow} s$ 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 $P^v \xrightarrow{a} P^{v'}$.

21.1.2 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.

21.1.3 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

$$\forall\, i = 1, \ldots, n \quad P_i \equiv a_{i1}; P_{i1} + \ldots + a_{in_i}; P_{in_i}. \tag{21.1}$$

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 (21.1). 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 $a_{ik} = \overline{a_{jl}}$.

21.2 Сильная эквивалентность

21.2.1 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. $P_j \stackrel{a}{\to} P'_j$), который эквивалентен P'_i.

Обозначим символом $\mathcal{M}$ совокупность всех отношений μ на множестве $Proc$, удовлетворяющих условию: $\forall\,(P_1, P_2) \in \mu$

$$\begin{array}{l} \forall\, a \in Act, \forall\, P'_1 : P_1 \stackrel{a}{\to} P'_1 \;\; \exists\, P'_2 : P_2 \stackrel{a}{\to} P'_2 \text{ и } (P'_1, P'_2) \in \mu, \\ \forall\, a \in Act, \forall\, P'_2 : P_2 \stackrel{a}{\to} P'_2 \;\; \exists\, P'_1 : P_1 \stackrel{a}{\to} P'_1 \text{ и } (P'_1, P'_2) \in \mu. \end{array} \tag{21.2}$$

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 34
$\sim$ 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

$$\text{если } P \equiv P', \text{ то } P \sim P'. \tag{21.3}$$

21.2.2 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 $\rho \subseteq S_1 \times S_2$, содержа-

щее пару (s_1^0, s_2^0) и удовлетворяющее условию: $\forall\,(s_1, s_2) \in \rho$

$$\begin{array}{l} \forall\, a \in Act, \forall\, s_1' : s_1 \stackrel{a}{\to} s_1' \;\; \exists\, s_2' : s_2 \stackrel{a}{\to} s_2' \text{ и } (s_1', s_2') \in \rho, \\ \forall\, a \in Act, \forall\, s_2' : s_2 \stackrel{a}{\to} s_2' \;\; \exists\, s_1' : s_1 \stackrel{a}{\to} s_1' \text{ и } (s_1', s_2') \in \rho. \end{array} \tag{21.4}$$

Теорема 35
$P_1 \sim P_2$ 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 (21.4): из $(s_1, s_2) \in \rho$ и $s_1 \stackrel{a}{\to} s_1'$ следует

   $$\exists\, s_2' : s_2 \stackrel{a}{\to} s_2', (s_1', s_2') \in \rho. \tag{21.5}$$

   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 (21.5).

   Аналогично доказывается второе свойство в (21.4).

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 (21.2): из соотношений $(P_1^{s_1}, P_2^{s_2}) \in \mu$ и $P_1^{s_1} \stackrel{a}{\to} P_1'$ следует, что

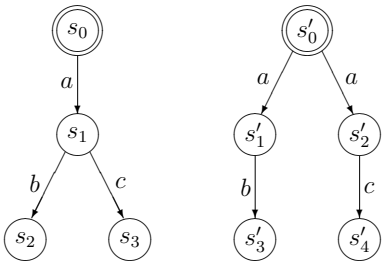
   $$\exists\, P_2' : P_2^{s_2} \stackrel{a}{\to} P_2' \text{ и } (P_1', P_2') \in \mu. \tag{21.6}$$

   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 (21.6) можно взять $P_2^{s_2'}$.

   Аналогично доказывается второе свойство в (21.2). ■

Используя теорему 35, можно обосновать следующие утверждения.

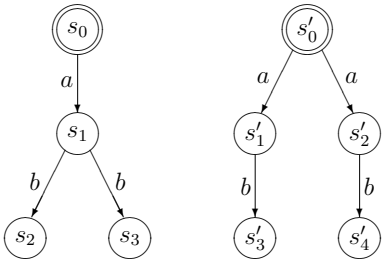
1. Процессы



(21.7)

не являются сильно эквивалентными. Действительно, если они сильно эквивалентны, то по теореме 35 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 $\{(s_0, s'_0), (s_1, s'_1), (s_1, s'_2), (s_2, s'_3), (s_2, s'_4), (s_3, s'_3), (s_3, s'_4)\}$.

21.2.3 Логический критерий сильной эквивалентности

Определим множество Fm **модальных формул (МФ)** следующим образом:

- Fm содержит символы $\top$ и $\bot$,
- если $\varphi \in Fm$, то $\neg\varphi \in Fm$,

- если $\varphi, \psi \in Fm$, то $\varphi \wedge \psi \in Fm$,
- если $\varphi \in Fm$ и $a \in Act$, то $\Diamond_a \varphi \in Fm$.

Значение МФ $\varphi \in Fm$ на процессе $P \in Proc$ – это элемент $\varphi^P \in \{0, 1\}$, определяемый следующим образом:

- $\top^P = 1$, $\bot^P = 0$, $(\neg\varphi)^P = 1 - \varphi^P$, $(\varphi \wedge \psi)^P = \varphi^P \cdot \psi^P$,
- $(\Diamond_a \varphi)^P = \begin{cases} 1, & \text{если } \exists\, P' \in Proc : P \xrightarrow{a} P', \varphi^{P'} = 1, \\ 0 & \text{в противном случае} \end{cases}$

Теорема 36
Пусть процессы P_1 и P_2 конечны. Тогда

$$P_1 \sim P_2 \quad \Leftrightarrow \quad \forall\, \varphi \in Fm \;\; \varphi^{P_1} = \varphi^{P_2}. \tag{21.8}$$

Доказательство.
Импликация «$\Rightarrow$» доказывается индукцией по структуре МФ φ.
Докажем импликацию «$\Leftarrow$». Определим

$$\mu \stackrel{\text{def}}{=} \{(Q_1, Q_2) \in Proc \times Proc \mid \forall\, \varphi \in Fm \;\; \varphi^{Q_1} = \varphi^{Q_2}\}.$$

Докажем, что $\mu \in \mathcal{M}$. Если это не так, то $\exists\, (Q_1, Q_2) \in \mu$, $\exists\, a \in Act$: например, верно утверждение:

$$\begin{array}{l} \exists\, Q_1' : Q_1 \xrightarrow{a} Q_1', \\ \forall\, Q_2' \;\; (Q_2 \xrightarrow{a} Q_2' \;\Rightarrow\; (Q_1', Q_2') \notin \mu). \end{array} \tag{21.9}$$

Из первого соотношения в (21.9) следует, что $(\Diamond_a \top)^{Q_1} = 1$, откуда, учитывая определение μ, получаем: $(\Diamond_a \top)^{Q_2} = 1$, т.е. $\exists\, Q_2' : Q_2 \xrightarrow{a} Q_2'$.

Пусть множество $\{Q \in Proc \mid Q_2 \xrightarrow{a} Q\}$ имеет вид $\{Q_{21}, \ldots, Q_{2n}\}$. Из второго соотношения в (21.9) следует, что $\forall\, i = 1, \ldots, n$ $(Q_1', Q_{2i}) \notin \mu$, т.е. $\exists\, \varphi_i \in Fm$: $\varphi_i^{Q_1'} = 1$, $\varphi_i^{Q_{2i}} = 0$. Полагая $\varphi = \Diamond_a(\varphi_1 \wedge \ldots \wedge \varphi_n)$, получаем: $\varphi^{Q_1} = 1$, $\varphi^{Q_2} = 0$, что противоречит предположению $(Q_1, Q_2) \in \mu$.

Таким образом, $\mu \in \mathcal{M}$. Если верна правая часть (21.8), то $(P_1, P_2) \in \mu$, поэтому $(P_1, P_2) \in \sim$. ■

Приведем пример МФ, которая принимает разные значения на процессах P_1 и P_2 на рисунке (21.7). Как было установлено выше, $P_1 \not\sim P_2$. Определим $\varphi \stackrel{\text{def}}{=} \Diamond_a(\Diamond_b \top \wedge \Diamond_c \top)$. Нетрудно видеть, что $\varphi^{P_1} = 1$ и $\varphi^{P_2} = 0$.

21.2.4 Алгебраические свойства сильной эквивалентности

Теорема 37

$\sim$ 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 35, 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.

- $\{(s_{(1)}^0, s_{(2)}^0)\} \cup \rho$ – БМ между $a; P_1$ и $a; P_2$, где $s_{(i)}^0$ – начальное состояние процесса $a; P_i \;\; (i = 1, 2)$.
- Пусть процесс P имеет вид (S, s^0, R), тогда
  - $\{(s_{(1)}^0, s_{(2)}^0)\} \cup \rho \cup \{(s, s) \mid s \in S\}$ – БМ между $P_1 + P$ и $P_2 + P$, где $s_{(i)}^0$ – 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 38

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 $P + P$. ■

21.2.5 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 $P_i = (S_i, s_i^0, R_i)$ $(i = 1, 2)$. 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:

$$\rho' \stackrel{\text{def}}{=} \{(s_1, s_2) \in S_1 \times S_2 \mid \text{ верно (21.4)}\}.$$

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 $(s_1^0, s_2^0) \in \rho$ и $\rho \subseteq \rho'$.

Определим ρ_{max} как объединение всех отношений из совокупности

$$\{\rho \subseteq S_1 \times S_2 \mid \rho \subseteq \rho'\}. \tag{21.10}$$

$\rho_{max} \in (21.10)$, т.к. $\forall\, \rho \in (21.10)$ из включения $\rho \subseteq (\bigcup\limits_{\rho \in (21.10)} \rho) = \rho_{max}$ и монотонности функции $'$ следует, что $\forall\, \rho \in (21.10)\, \rho \subseteq \rho' \subseteq \rho'_{max}$, поэтому $\rho_{max} = \bigcup\limits_{\rho \in (21.10)} \rho \subseteq \rho'_{max}$, т.е. $\rho_{max} \in (21.10)$.

Из включения $\rho_{max} \subseteq \rho'_{max}$ и монотонности функции $'$ следует включение $\rho'_{max} \subseteq \rho''_{max}$, т.е. $\rho'_{max} \in (21.10)$, 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 (21.10) и наибольшей неподвижной точкой функции $'$. Будем обозначать это отношение записью $\rho(P_1, P_2)$.

Из теоремы 35 следует, что

$$P_1 \sim P_2 \;\Leftrightarrow\; (s_1^0, s_2^0) \in \rho(P_1, P_2), \tag{21.11}$$

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

$$\forall\, P_1, P_2, P_3 \in Proc \;\; \rho(P_1, P_2) \circ \rho(P_2, P_3) \subseteq \rho(P_1, P_3). \tag{21.12}$$

Согласно (21.11), доказательство соотношения $P_1 \sim P_2$ сводится к вычислению $\rho(P_1, P_2)$ и проверке правой части (21.11). 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 $O(|S_1|^3 \cdot |S_2|^3 \cdot A)$.

21.2.6 Минимизация процессов относительно сильной эквивалентности

В этом пункте рассматривается задача построения по заданному конечному процессу P процесса P_{min} с наименьшим числом состояний, удовлетворяющего условию $P_{min} \sim P$. Процедура построения такого процесса P_{min} называется **минимизацией** процесса P (относительно $\sim$).

Первый этап минимизации заключается в удалении из P всех недостижимых состояний. Как было отмечено в примере 2 в пункте 21.1.3, в результате такого удаления получается процесс P', обладающий свойством $P' \equiv P$, из которого, согласно (21.3), следует свойство $P' \sim P$.

Множество всех достижимых состояний процесса P может быть построено, например, следующим образом.

Пусть $P = (S, s^0, R)$. Определим последовательность $\{S_i \mid i \geq 0\}$ подмножеств множества S: $S_0 \stackrel{\text{def}}{=} \{s^0\}$, и $\forall\, i \geq 0$ S_{i+1} получается добавлением к S_i всех состояний $s' \in S$, таких, что $\exists\, s \in S_i$, $\exists\, a \in Act$: $s \stackrel{a}{\to} s'$. По построению, верны включения $S_0 \subseteq S_1 \subseteq \ldots$. Нетрудно видеть, что $\forall\, s \in S$ состояние s достижимо тогда и только тогда, когда $\exists\, i : s \in S_i$.

Поскольку S конечно, то последовательность $\{S_i \mid i \geq 0\}$ не может неограниченно возрастать, т.е. множество всех достижимых состояний процесса P является тем членом последовательности $\{S_i \mid i \geq 0\}$, на котором наступает её стабилизация (т.е. $S_i = S_{i+1}$).

Будем обозначать процесс, получаемый удалением из исходного процесса P недостижимых состояний, тем же символом P.

Второй этап минимизации связан с использованием $\rho(P, P)$.

Теорема 39
$\forall\, P \in Proc$ отношение $\rho(P,P)$ является эквивалентностью.

Доказательство.

1. **Рефлексивность** отношения $\rho(P,P)$ следует из того, что тождественное отношение $Id_S = \{(s,s) \mid s \in S\}$, где S – множество состояний процесса P, обладает свойством $Id_S \subseteq Id'_S$, т.е. $Id_S \in$ (21.10).

2. **Симметричность** отношения $\rho(P,P)$ следует из того, что если отношение $\rho \subseteq S \times S$ обладало свойством $\rho \in$ (21.10), т.е. $\rho \subseteq \rho'$, то ρ^{-1} обладает свойством $\rho^{-1} \subseteq (\rho^{-1})'$ т.е. $\rho^{-1} \in$ (21.10).

3. **Транзитивность** отношения $\rho(P,P)$ следует из (21.12). ■

Пусть $P = (S, s^0, R)$. Определим $P_\sim$ как процесс $(S_\sim, s^0_\sim, R_\sim)$, где

- $S_\sim$ состоит из классов эквивалентности, на которые разбивается S по эквивалентности $\rho(P,P)$, ниже $\forall\, s \in S$ запись $[s]$ обозначает класс этой эквивалентности, содержащий s,

- $s^0_\sim = [s^0]$, $R_\sim = \{[s] \xrightarrow{a} [s'] \mid s \xrightarrow{a} s' \in R\}$.

Теорема 40
Для каждого процесса P верно свойство $P \sim P_\sim$.

Доказательство.
Пусть $P = (S, s^0, R)$. Докажем, что отношение $\rho \stackrel{\text{def}}{=} \{\ (s,[s])\ \mid s \in S\}$ является БМ между P и $P_\sim$.

Свойство $(s^0, s^0_\sim) \in \rho$ верно по определению состояния $s^0_\sim$.

Первое свойство в (21.4) верно по определению множества $R_\sim$.

Докажем второе свойство в (21.4), которое в рассматриваемом случае выглядит следующим образом: для каждого перехода $[s] \xrightarrow{a} [s'] \in R_\sim$ существует переход в R вида $s \xrightarrow{a} s''$, такой, что $[s''] = [s']$.

Из определения $R_\sim$ следует, что R содержит переход вида $s_1 \xrightarrow{a} s'_1$ где $[s_1] = [s]$ и $[s'_1] = [s']$, т.е. $(s_1, s) \in \rho(P,P)$ и $(s'_1, s') \in \rho(P,P)$.

Так как $\rho(P,P)$ – БМ, то из $s_1 \xrightarrow{a} s'_1 \in R$ и $(s_1, s) \in \rho(P,P)$ следует, что R содержит переход $s \xrightarrow{a} s''_1$, где $(s''_1, s'_1) \in \rho(P,P)$.

Так как $\rho(P,P)$ транзитивно, то из $(s''_1, s'_1) \in \rho(P,P)$ и $(s'_1, s') \in \rho(P,P)$ следует, что $(s''_1, s') \in \rho(P,P)$, т.е. $[s''_1] = [s']$.

Таким образом, в качестве искомого s'' можно взять s''_1. ■

Будем называть процесс $P = (S, s^0, R)$ **минимальным**, если каждое его состояние достижимо и $\rho(P, P) = Id_S$.

Теорема 41
Если каждое состояние процесса P достижимо, то $P_\sim$ минимален.

Доказательство.
Из достижимости каждого состояния процесса P и из определения процесса $P_\sim$ следует, что каждое состояние процесса $P_\sim$ достижимо.

Пусть $P = (S, s^0, R)$. Равенство $\rho(P_\sim, P_\sim) = Id_{S_\sim}$ верно потому, что $\forall\, s, s' \in S$ из $([s], [s']) \in \rho(P_\sim, P_\sim)$ следует равенство $[s] = [s']$.

Поскольку $(s, [s]) \in \rho(P, P_\sim)$ и $([s'], s') \in \rho(P_\sim, P)$, то

$$(s, s') \in \rho(P, P_\sim) \circ \rho(P_\sim, P_\sim) \circ \rho(P_\sim, P) \subseteq \rho(P, P) \tag{21.13}$$

(включение в (21.13) следует из (21.12)) и следовательно, верно соотношение $(s, s') \in \rho(P, P)$, которое равносильно доказываемому равенству $[s] = [s']$. ■

Теорема 42
Если процессы P_1 и P_2 минимальны и $P_1 \sim P_2$, то P_1 и P_2 изоморфны.

Доказательство.
Пусть $P_i = (S_i, s_i^0, R_i)$ $(i = 1, 2)$. Согласно (21.12),

$$\begin{array}{l} \rho(P_1, P_2) \circ \rho(P_2, P_1) \subseteq \rho(P_1, P_1) = Id_{S_1}, \\ \rho(P_2, P_1) \circ \rho(P_1, P_2) \subseteq \rho(P_2, P_2) = Id_{S_2}. \end{array} \tag{21.14}$$

Докажем, что $\forall\, s_1 \in S_1\ \exists\, s_2 \in S_2 : (s_1, s_2) \in \rho(P_1, P_2)$.

- Если $s_1 = s_1^0$, то $s_2 = s_2^0$. Так как $P_1 \sim P_2$, то $(s_1^0, s_2^0) \in \rho(P_1, P_2)$.
- Если $s_1 \neq s_1^0$, то достижимость s_1 означает, что в P_1 существует путь вида $s_1^0 \xrightarrow{a_1} \ldots \xrightarrow{a_n} s_1$, откуда, на основании $(s_1^0, s_2^0) \in \rho(P_1, P_2)$, нетрудно получить, что в P_2 существует путь вида $s_2^0 \xrightarrow{a_1} \ldots \xrightarrow{a_n} s_2$, причем $(s_1, s_2) \in \rho(P_1, P_2)$.

Докажем, что $\forall\, s_1 \in S_1$ существует единственный $s_2 \in S_2$ со свойством $(s_1, s_2) \in \rho(P_1, P_2)$. Если элемент $s_2' \in S_2$ таков, что $(s_1, s_2') \in \rho(P_1, P_2)$, то поскольку $(s_2', s_1) \in (\rho(P_1, P_2))^{-1} \subseteq \rho(P_2, P_1)$, тогда

$$(s_2', s_2) \in \rho(P_2, P_1) \circ \rho(P_1, P_2) \subseteq Id_{S_2},$$

откуда следует, что $s_2' = s_2$. Аналогично доказывается, что $\forall\, s_2 \in S_2$ существует единственный $s_1 \in S_1$ со свойством $(s_2, s_1) \in \rho(P_2, P_1)$.

Из доказанного выше, учитывая (21.14), нетрудно заключить, что отношения $\rho(P_1, P_2)$ и $\rho(P_2, P_1)$ определяют взаимно обратные функции. Изоморфность P_1 и P_2 следует из того, что $\rho(P_1, P_2)$ и $\rho(P_2, P_1)$ – БМ. ■

Теорема 43

Если каждое состояние конечного процесса P достижимо, то процесс $P_\sim$ имеет наименьшее число состояний среди всех конечных процессов, сильно эквивалентных P.

Доказательство.

Пусть P' – конечный процесс, такой, что $P' \sim P$. Удалим из P' недостижимые состояния, получившийся процесс обозначим той же записью P'. Согласно теореме 40, $P \sim P_\sim$ и $P' \sim P'_\sim$, поэтому $P_\sim \sim P'_\sim$.

Согласно теореме 41, процессы $P_\sim$ и $P'_\sim$ минимальны. Отсюда по теореме 42 следует, что $P_\sim$ и $P'_\sim$ изоморфны. В частности, они имеют одинаковое число состояний. Следовательно, число состояний процесса $P_\sim$ не превосходит числа состояний процесса P' (поскольку состояния процесса $P'_\sim$ являются классами разбиения множества состояний процесса P'). ■

Таким образом, искомым процессом P_{min} является процесс $P_\sim$.

Теорема 44

Пусть конечный процесс P имеет множество состояний S. Для нахождения отношения $\rho(P, P)$ нужно вычислить не более $|S|$ членов последовательности $\{\rho_i \mid i \geq 1\}$, в которой $\rho_1 \stackrel{\text{def}}{=} S \times S$, и $\forall\, i \geq 1 \quad \rho_{i+1} \stackrel{\text{def}}{=} \rho_i'$.

Доказательство.

Каждое из отношений в последовательности $\{\rho_i \mid i \geq 1\}$ является эквивалентностью (так как если $\rho \subseteq S \times S$ – эквивалентность, то ρ' тоже будет эквивалентностью). Поэтому каждый член последовательности $\{\rho_i \mid i \geq 1\}$ определяет разбиение множества S и для каждого $i \geq 1$ если $\rho_{i+1} \neq \rho_i$, то из включения $\rho_{i+1} \subseteq \rho_i$ следует, что разбиение, соответствующее отношению ρ_{i+1}, является измельчением разбиения, соответствующего отношению ρ_i. Нетрудно видеть, что таких измельчений может быть не больше, чем количество элементов в множестве S. ■

21.3 Наблюдаемая эквивалентность

21.3.1 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: $\forall\,(P_1, P_2) \in \mu$

$$\begin{array}{l}
\forall\, P_1' : P_1 \xrightarrow{\tau} P_1' \;\; \exists\, P_2' : P_2 \xrightarrow{\tau^*} P_2' \text{ и } (P_1', P_2') \in \mu, \\
\forall\, P_2' : P_2 \xrightarrow{\tau} P_2' \;\; \exists\, P_1' : P_1 \xrightarrow{\tau^*} P_1' \text{ и } (P_1', P_2') \in \mu, \\
\forall\, a \in Act \setminus \{\tau\}, \forall\, P_1' : P_1 \xrightarrow{a} P_1' \;\; \exists\, P_2' : P_2 \xrightarrow{\tau^* a \tau^*} P_2' \text{ и } (P_1', P_2') \in \mu, \\
\forall\, a \in Act \setminus \{\tau\}, \forall\, P_2' : P_2 \xrightarrow{a} P_2' \;\; \exists\, P_1' : P_1 \xrightarrow{\tau^* a \tau^*} P_1' \text{ и } (P_1', P_2') \in \mu.
\end{array} \tag{21.15}$$

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 34 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 $P_1 \approx P_2$.

21.3.2 Критерий наблюдаемой эквивалентности, основанный на понятии наблюдаемого бимоделирования

Для отношения $\approx$ также имеет место аналог критерия, основанного на понятии БМ (теорема 35 из параграфа 21.2.2). Для его формулировки мы введём вспомогательные обозначения.

Пусть $P = (S, s^0, R) \in Proc$ и $s_1, s_2 \in S$, тогда

- запись $s \xrightarrow{\tau^*} s'$ означает, что либо $s = s'$, либо существуют состояния $s_1, \ldots, s_n$, такие, что $s_1 = s,\ s_n = s'$ и $\forall\, i = 1, \ldots, n-1\ s_i \xrightarrow{\tau} s_{i+1}$,
- $\forall\, a \in Act \setminus \{\tau\}$ запись $s \xrightarrow{\tau^* a \tau^*}$ означает, что существуют состояния s_1 и s_2 со следующими свойствами: $s \xrightarrow{\tau^*} s_1,\ s_1 \xrightarrow{a} s_2,\ s_2 \xrightarrow{\tau^*} s'$.

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: $\forall\,(s_1, s_2) \in \rho$

$$\begin{cases} \forall\, s_1' : s_1 \xrightarrow{\tau} s_1' \;\; \exists\, s_2' : s_2 \xrightarrow{\tau^*} s_2' \text{ и } (s_1', s_2') \in \rho, \\ \forall\, s_2' : s_2 \xrightarrow{\tau} s_2' \;\; \exists\, s_1' : s_1 \xrightarrow{\tau^*} s_1' \text{ и } (s_1', s_2') \in \rho, \\ \forall\, a \in Act \setminus \{\tau\}, \forall\, s_1' : s_1 \xrightarrow{a} s_1' \;\; \exists\, s_2' : s_2 \xrightarrow{\tau^* a \tau^*} s_2' \text{ и } (s_1', s_2') \in \rho, \\ \forall\, a \in Act \setminus \{\tau\}, \forall\, s_2' : s_2 \xrightarrow{a} s_2' \;\; \exists\, s_1' : s_1 \xrightarrow{\tau^* a \tau^*} s_1' \text{ и } (s_1', s_2') \in \rho. \end{cases} \tag{21.16}$$

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 35).

Используя вышеупомянутое утверждение, можно доказать, что

$$\forall\, P \in Proc \quad P \approx \tau; P. \tag{21.17}$$

21.3.3 Логический критерий наблюдаемой эквивалентности процессов

Логический критерий наблюдаемой эквивалентности аналогичен критерию из параграфа 21.2.3. В данном критерии используется то же самое множество МФ. Для каждой МФ $\varphi \in Fm$ определяется ее значение на процессе $P \in Proc$, которое обозначается записью φ_τ^P и определение которого отличается от определения аналогичного понятия в параграфе 21.2.3 лишь для МФ вида $\Diamond_a\varphi$:

- $(\Diamond_\tau\varphi)_\tau^P = \begin{cases} 1, & \text{если } \exists\, P' \in Proc : P \xrightarrow{\tau^*} P', \varphi_\tau^{P'} = 1, \\ 0 & \text{в противном случае,} \end{cases}$
- $\forall\, a \in Act\setminus\{\tau\} \;\; (\Diamond_a\varphi)_\tau^P = \begin{cases} 1, & \text{если } \exists\, P' \in Proc : P \xrightarrow{\tau^* a \tau^*} P', \varphi_\tau^{P'} = 1, \\ 0 & \text{в противном случае.} \end{cases}$

Теорема 45
Пусть процессы P_1 и P_2 конечны. Тогда

$$P_1 \approx P_2 \quad \Leftrightarrow \quad \forall\, \varphi \in Fm \;\; \varphi_\tau^{P_1} = \varphi_\tau^{P_2}. \;\blacksquare$$

Доказательство данной теоремы аналогично доказательству теоремы 36.

21.3.4 Алгебраические свойства наблюдаемой эквивалентности

Теорема 46

$\approx$ сохраняется всеми операциями на процессах, за исключением операции $+$, т.е. если $P_1 \approx P_2$, то

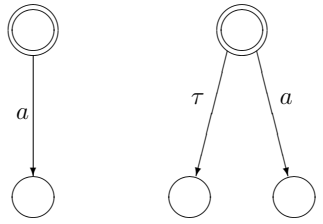
- $\forall\, a \in Act \quad a; P_1 \approx a; P_2$,
- $\forall\, P \in Proc \quad P_1|P \approx P_2|P$,
- $\forall\, L \subseteq Names \quad P_1 \setminus L \approx P_2 \setminus L$,
- $\forall\, \theta : Names \to Names \quad P_1^\theta \approx P_2^\theta$. ■

Доказательство этой теоремы аналогично доказательству теоремы 37. Приведем два примера несохраняемости отношения $\approx$ операцией $+$.

1. Согласно (21.17), $\mathbf{0} \approx \tau; \mathbf{0}$, однако если $a \in Act \setminus \{\tau\}$, то

$$\mathbf{0} + a; \mathbf{0} \not\approx \tau; \mathbf{0} + a; \mathbf{0}, \tag{21.18}$$

в чём нетрудно убедиться при рассмотрении графового представления левой и правой частей в (21.18):



2. Другой пример: если $a, b \in Act \setminus \{\tau\}$ и $a \neq b$, то

$$a; \mathbf{0} + b; \mathbf{0} \not\approx \tau; a; \mathbf{0} + \tau; b; \mathbf{0},$$

хотя $a; \mathbf{0} \approx \tau; a; \mathbf{0}$ и $b; \mathbf{0} \approx \tau; b; \mathbf{0}$.

21.3.5 Распознавание наблюдаемой эквивалентности

Для решения задачи распознавания для двух заданных конечных процессов, являются ли они наблюдаемо эквивалентными, может быть построена теория, аналогичная изложенной в параграфах 21.2.5 и 21.2.6. 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

$$\rho'_\tau \stackrel{\text{def}}{=} \{(s_1, s_2) \in S_1 \times S_2 \mid \text{ верно (21.16)}\}.$$

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 21.2.5, 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.

21.4 Наблюдаемая конгруэнция

21.4.1 Мотивировка наблюдаемой конгруэнции

В этом параграфе мы определяем ещё один вид эквивалентности процессов, называемый **наблюдаемой конгруэнцией**. Данная эквивалентность обозначается символом $\simeq$. Мы определяем эту эквивалентность исходя из следующих условий:

- процессы, находящиеся в отношении $\simeq$, должны быть наблюдаемо эквивалентными и
- $\simeq$ должна быть конгруэнцией относительно операций на процессах.

Мотивировка второго из этих условий заключается в следующем. Если какой-либо процесс P определен в виде выражения (будем обозначать его тем же символом P) с операциями на процессах, то подвыражения этого выражения можно интерпретировать как компоненты процесса P. В некоторых ситуациях желательно наличие следующего свойства: если какую-либо компоненту процесса P заменить на эквивалентную ей, то получившийся процесс должен быть эквивалентным исходному процессу P. Такая замена может пониматься как замена в выражении P некоторого его подвыражения P' на выражение P'', в результате чего из процесса P получается процесс $P(P''/P')$. Формализация описанного выше условия имеет следующий вид: если $P' \simeq P''$, то $P \simeq P(P''/P')$. Нетрудно видеть, что данное условие равносильно тому, что $\simeq$ сохраняется всеми операциями на процессах, т.е. является конгруэнцией.

Сформулированные выше условия определяют искомую эквивалентность $\simeq$ неоднозначно. Например, им удовлетворяют тождественное отношение Id_{Proc} и сильная эквивалентность $\sim$. Отметим, что $\approx$ не удовлетворяет данным условиям, т.к. не сохраняется операцией $+$. Ниже мы докажем, что среди всех эквивалентностей на $Proc$, удовлетворяющих данным условиям, существует наибольшая (относительно включения). Естественно считать эту эквивалентность искомой эквивалентностью $\simeq$.

21.4.2 Определение наблюдаемой конгруэнции

Будем говорить, что процессы P_1 и P_2 находятся в отношении **наблюдаемой конгруэнции**, и обозначать этот факт записью $P_1 \simeq P_2$, если выполнены следующие условия:

$$\forall\, a \in Act, \forall\, P_1' : P_1 \xrightarrow{a} P_1' \;\; \exists\, P_2' : P_2 \xrightarrow{\tau^* a \tau^*} P_2' \text{ и } P_1' \approx P_2',$$
$$\forall\, a \in Act, \forall\, P_2' : P_2 \xrightarrow{a} P_2' \;\; \exists\, P_1' : P_1 \xrightarrow{\tau^* a \tau^*} P_1' \text{ и } P_1' \approx P_2'.$$

Нетрудно доказать, что

- отношение $\simeq = \{(P_1, P_2) \mid P_1 \simeq P_2\}$ – эквивалентность,
- $\sim \subseteq \;\simeq\; \subseteq \approx$, причем оба включения собственные:

$$\tau;\tau;\mathbf{0} \not\sim \tau;\mathbf{0}, \text{ но } \tau;\tau;\mathbf{0} \simeq \tau;\mathbf{0},$$
$$\tau;\mathbf{0} \not\approx \mathbf{0}, \text{ но } \tau;\mathbf{0} \approx \mathbf{0}.$$

Теорема 47
Пусть заданы два процесса $P_i = (S_i, s_i^0, R_i) \;\; (i = 1, 2)$.
$P_1 \simeq P_2$ тогда и только тогда, когда $(s_1^0, s_2^0) \in \rho_\tau(P_1, P_2)$ и

$$\left.\begin{array}{l} \forall\, s_1 : s_1^0 \xrightarrow{\tau} s_1 \;\; \exists\, s_2 : s_2^0 \xrightarrow{\tau^*\tau\tau^*} s_2 \text{ и } (s_1, s_2) \in \rho_\tau(P_1, P_2), \\ \forall\, s_2 : s_2^0 \xrightarrow{\tau} s_2 \;\; \exists\, s_1 : s_1^0 \xrightarrow{\tau^*\tau\tau^*} s_1 \text{ и } (s_1, s_2) \in \rho_\tau(P_1, P_2). \end{array}\right\} \qquad (21.19)$$

Теорема 48
$\simeq$ является конгруэнцией, т.е. если $P_1 \simeq P_2$, то

- для каждого $a \in Act \quad a; P_1 \simeq a; P_2$,
- для каждого процесса $P \quad P_1 + P \simeq P_2 + P$ и $P_1 | P \simeq P_2 | P$,
- для каждого $L \subseteq Names \quad P_1 \setminus L \simeq P_2 \setminus L$,
- для каждого переименования $\theta : Names \to Names \quad P_1^\theta \simeq P_2^\theta$.

Доказательство.

- Из определения отношения $\simeq$ непосредственно следует, что если $P_1 \approx P_2$, то $\forall\, a \in Act\ \ a; P_1 \simeq a; P_2$.
- Пусть S – множество состояний процесса P, s_1^0 и s_2^0 – начальные состояния $P_1 + P$ и $P_2 + P$ соответственно, тогда

$$\{(s_1^0, s_2^0)\} \cup \rho_\tau(P_1, P_2) \cup Id_S \subseteq \rho_\tau(P_1 + P, P_2 + P),$$
$$\{((s_1, s), (s_2, s)) \mid (s_1, s_2) \in \rho_\tau(P_1, P_2),\ s \in S\} \subseteq \rho_\tau(P_1|P, P_2|P).$$

- $\rho_\tau(P_1, P_2) \subseteq \rho_\tau(P_1 \setminus L, P_2 \setminus L)$ и $\rho_\tau(P_1, P_2) \subseteq \rho_\tau(P_1^\theta, P_2^\theta)$.

Кроме того, для каждой пары Q_1, Q_2 процессов, наблюдаемая конгруэнтность которых утверждается в теореме (48), отношение $\rho_\tau(Q_1, Q_2)$ обладает свойством (21.19). ■

21.4.3 Связь между наблюдаемой эквивалентностью и наблюдаемой конгруэнцией

Теорема 49
$\forall\, P_1, P_2 \in Proc \quad P_1 \approx P_2 \ \Leftrightarrow\ (P_1 \simeq P_2) \vee (P_1 \simeq \tau; P_2) \vee (\tau; P_1 \simeq P_2)$.

Доказательство.
Импликация «$\Leftarrow$» следует из свойств $\simeq \subseteq \approx$ и $\forall\, P \in Proc\ \ P \approx \tau; P$.
Докажем импликацию «$\Rightarrow$». Предположим, что $P_1 \approx P_2$, но $P_1 \not\simeq P_2$.
Соотношение $P_1 \not\simeq P_2$ имеет место, если верно одно из утверждений:

$$\begin{array}{l} \exists\, P_1' : P_1 \stackrel{\tau}{\to} P_1' \quad \nexists\, P_2' : P_2 \stackrel{\tau^*\tau\tau^*}{\longrightarrow} P_2' \text{ и } P_1' \approx P_2' \text{ или} \\ \exists\, P_2' : P_2 \stackrel{\tau}{\to} P_2' \quad \nexists\, P_1' : P_1 \stackrel{\tau^*\tau\tau^*}{\longrightarrow} P_1' \text{ и } P_1' \approx P_2'. \end{array} \tag{21.20}$$

Пусть, например, верно первое утверждение в (21.20). Докажем, что в этом случае $P_1 \simeq \tau; P_2$. Так как $P_1 \approx \tau; P_2$, то достаточно доказать, что

$$\begin{array}{l} \forall\, P_1' : P_1 \stackrel{\tau}{\to} P_1' \ \exists\, P_2' : \tau; P_2 \stackrel{\tau^*\tau\tau^*}{\longrightarrow} P_2' \text{ и } P_1' \approx P_2', \\ \forall\, P_2' : \tau; P_2 \stackrel{\tau}{\to} P_2' \ \exists\, P_1' : P_1 \stackrel{\tau^*\tau\tau^*}{\longrightarrow} P_1' \text{ и } P_1' \approx P_2'. \end{array}$$

Данные свойства можно переписать следующим образом:

$$\begin{array}{l} \forall\, P_1' : P_1 \stackrel{\tau}{\to} P_1' \ \exists\, P_2' : P_2 \stackrel{\tau^*}{\longrightarrow} P_2' \text{ и } P_1' \approx P_2', \\ \exists\, P_1' : P_1 \stackrel{\tau^*\tau\tau^*}{\longrightarrow} P_1' \text{ и } P_1' \approx P_2. \end{array} \tag{21.21}$$

Первое свойство в (21.21) следует из предположения $P_1 \approx P_2$.

Обоснуем второе свойство в (21.21). Согласно первому утверждению в (21.20), $\exists\, P_1' : P_1 \stackrel{\tau}{\to} P_1'$, откуда следует, что $\exists\, P_2' : P_2 \stackrel{\tau^*}{\to} P_2'$, и $P_1' \approx P_2'$ (т.к. $P_1 \approx P_2$). Случай $P_2' \neq P_2$ невозможен по первому утверждению в (21.20).

Если же верно второе утверждение в (21.20), то аналогичными рассуждениями можно доказать, что $\tau; P_1 \simeq P_2$. ■

Теорема 50

$\simeq = \stackrel{+}{\approx}$, где $\stackrel{+}{\approx} = \{(P_1, P_2) \mid \forall\, P \in Proc \;\; P_1 + P \approx P_2 + P\}$.

Доказательство.

Включение $\simeq \subseteq \stackrel{+}{\approx}$ следует из того, что $\simeq$ – конгруэнция (т.е., в частности, $\simeq$ сохраняет операцию $+$) и $\simeq \subseteq \approx$.

Докажем включение $\stackrel{+}{\approx} \subseteq \simeq$. Пусть $(P_1, P_2) \in \stackrel{+}{\approx}$, докажем, что $P_1 \simeq P_2$.

Из определения $\stackrel{+}{\approx}$ следует, что $P_1 + \mathbf{0} \approx P_2 + \mathbf{0}$, откуда следует, что $P_1 \approx P_2$. Если $P_1 \not\simeq P_2$, то по теореме 49 из $P_1 \approx P_2$ следует, что либо $P_1 \simeq \tau; P_2$ либо $\tau; P_1 \simeq P_2$. Пусть, например, имеет место первый случай: $P_1 \simeq \tau; P_2$ (второй случай рассматривается аналогично). Докажем, что в этом случае верно соотношение $\tau; P_2 \simeq P_2$ (из которого, учитывая предположение $P_1 \simeq \tau; P_2$, получаем доказываемое соотношение $P_1 \simeq P_2$).

Так как $\tau; P_2 \approx P_2$, то для доказательства соотношения $\tau; P_2 \simeq P_2$ достаточно доказать, что $\exists\, P_2' \approx P_2$: $P_2 \stackrel{\tau^*\tau\tau^*}{\longrightarrow} P_2'$.

Так как $\simeq$ конгруэнция и $\simeq \subseteq \approx$, то из $P_1 \simeq \tau; P_2$ следует: $\forall\, P \in Proc$

$$P_2 + P \approx P_1 + P \approx \tau; P_2 + P. \tag{21.22}$$

Рассмотрим случай, когда P в (21.22) имеет вид $a; \mathbf{0}$, где $a \neq \tau$ и a не входит в P_2. Поскольку $\tau; P_2 + a; \mathbf{0} \stackrel{\tau}{\to} P_2$, то по определению наблюдаемой эквивалентности $\exists\, P_2' \approx P_2 : P_2 + a; \mathbf{0} \stackrel{\tau^*}{\to} P_2'$.

Случай $P_2 + a; \mathbf{0} = P_2'$ невозможен, так как иначе верно $P_2' \stackrel{a}{\to} \mathbf{0}$, откуда по $P_2' \approx P_2$ следует, что $P_2 \stackrel{\tau^* a \tau^*}{\longrightarrow} P$ для некоторого P, что невозможно. Следовательно, верно свойство $P_2 + a; \mathbf{0} \stackrel{\tau^*\tau\tau^*}{\to} P_2'$. По определению операции $+$, отсюда следует доказываемое свойство $P_2 \stackrel{\tau^*\tau\tau^*}{\longrightarrow} P_2'$. ■

Теорема 51

$\simeq$ является наибольшей конгруэнцией на $Proc$, содержащейся в $\approx$.

Доказательство.

Пусть $\mu \subseteq \approx$, и μ – конгруэнция. Докажем, что из $(P_1, P_2) \in \mu$ следует $P_1 \simeq P_2$. Так как μ конгруэнция, то $\forall\, P \in Proc \;\; (P_1 + P, P_2 + P) \in \mu \subseteq \approx$, откуда по теореме 50 следует, что $P_1 \simeq P_2$. ■

21.5 Другие эквивалентности процессов

Наряду с определенными выше эквивалентностями процессов можно ещё определить, например, такие эквивалентности, которые

- учитывают длительность исполнения действий, т.е., в частности, одно из условий эквивалентности процессов P_1 и P_2 имеет вид:
  - если один из этих процессов (P_i) может в течение некоторого промежутка времени незаметно превратиться в процесс P_i',
  - то и другой процесс (P_j) тоже должен обладать способностью в течение примерно такого же промежутка времени незаметно превратиться в процесс P_j', который эквивалентен P_i' (понятие «примерно такого же промежутка времени» может уточняться различным образом, в частности может иметь вероятностный характер),
- или учитывают свойство **справедливости (fairness)** в поведении процессов, т.е. не позволяют рассматривать как эквивалентные такие два процесса, один из которых обладает свойством справедливости, а другой – не обладает, где одно из определений свойства справедливости имеет следующий вид: процесс называется **справедливым**, если не существует бесконечной последовательности переходов этого процесса вида

  $$s_1 \xrightarrow{\tau} s_2 \xrightarrow{\tau} s_3 \xrightarrow{\tau} \dots$$

  такой, что $\forall\, i \geq 1$ из s_i выходит ребро с меткой из $Act \setminus \{\tau\}$,
- и т.д.

Отметим, что отношение наблюдаемой эквивалентности не учитывает свойство справедливости: существуют два процесса P_1 и P_2, такие, что $P_1 \approx P_2$, но P_1 обладает свойством справедливости, а P_2 не обладает этим свойством, например

- в качестве P_1 можно взять процесс $a;\mathbf{0}$, где $a \in Act \setminus \{\tau\}$,
- а в качестве P_2 – процесс $a;\mathbf{0} \,|\, \tau^*$, где процесс τ^* имеет одно состояние и один переход с меткой τ.

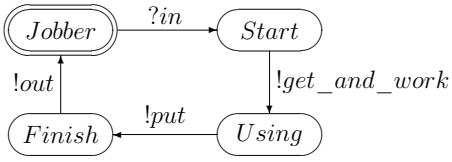
Решение о том, какое именно из понятий эквивалентности между процессами следует использовать в конкретной ситуации, существенно зависит от целей, для достижения которых предназначено данное понятие.

Глава 22

Примеры доказательства свойств процессов

22.1 Мастерская

Рассмотрим модель мастерской, в которой работают двое рабочих, пользующиеся для работы одним молотком. Поведение каждого рабочего в мастерской описывается процессом *Jobber*

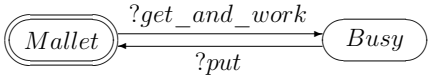


где

- действия $?in$ и $!out$ используются для взаимодействия рабочего с заказчиком и обозначают соответственно прием материала и выдачу готового изделия,
- действия $!get\_and\_work$ и $!put$ используются для взаимодействия рабочего с молотком и обозначают соответственно взятие молотка и выполнение с его помощью некоторых действий, затем возвращение молотка на место.

Согласно процессу *Jobber*, рабочий сначала принимает материал, затем берет молоток и работает, после чего кладет молоток, выдает готовое изделие и возвращается в исходное состояние.

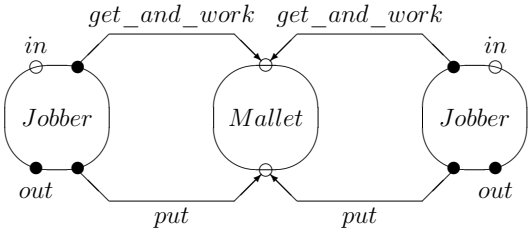
Поведение молотка представляется процессом *Mallet*:



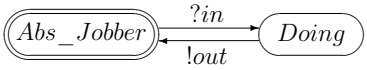
Функционирование мастерской определяется процессом

$$Job\_Shop = (Jobber \mid Jobber \mid Mallet) \setminus \{get\_and\_work,\ put\}. \quad (22.1)$$

Взаимодействие процессов, входящих в (22.1), можно изобразить в виде схемы, называемой **потоковым графом**. В нём каждый процесс, входящий в параллельную композицию, изображается овалом, на границе которого нарисованы белые и черные кружочки, представляющие выполняемые действия (белые кружочки представляют приемы, а черные – посылки), и стрелки соединяют совместно выполняемые действия.



Введем теперь понятие «абстрактного рабочего», про которого известно, что он циклически принимает материал и выдает готовые изделия, но ничего неизвестно о подробностях его работы. Его поведение описывается процессом Abs_Jobber:



Поведение «абстрактной мастерской» описывается процессом

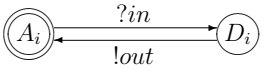
$$Abs\_Job\_Shop = Abs\_Jobber \mid Abs\_Jobber \quad (22.2)$$

Процесс (22.2) является **спецификацией** мастерской, он представляет поведение мастерской без учета деталей ее реализации.

Докажем соответствие мастерской ее спецификации, т.е. свойство

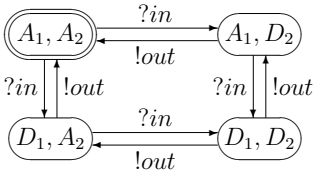
$$Job\_Shop \approx Abs\_Job\_Shop \quad (22.3)$$

Процесс (22.2) является параллельной композицией двух одинаковых процессов Abs_Jobber. В целях предотвращения коллизии в обозначениях мы выберем различные идентификаторы для обозначения состояний этих процессов. Пусть, например, эти процессы имеют вид

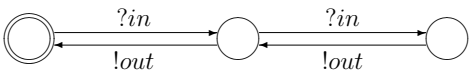


(где $i = 1, 2$).

Параллельная композиция этих процессов имеет вид



Применяя к данному процессу процедуру минимизации относительно наблюдаемой эквивалентности, мы получим процесс



(22.4)

Процесс Job_Shop имеет $4 \cdot 4 \cdot 2 = 32$ состояния, и мы не приводим его здесь ввиду его большой громоздкости. Если минимизировать этот процесс относительно наблюдаемой эквивалентности, то получится процесс, изоморфный процессу (22.4). Это означает, что верно свойство (22.3).

22.2 Планировщик

Будем использовать следующие определения и обозначения.

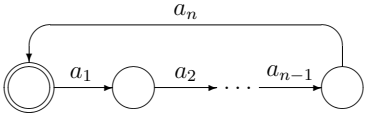
- Пусть задан процесс $P = (S, s^0, R)$. Запись Π_P^0 обозначает множество всех бесконечных путей в P вида

$$s^0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} s_2 \xrightarrow{a_3} \dots \tag{22.5}$$

  Если путь $\pi \in \Pi_P^0$ имеет вид (22.5), то запись $Act(\pi)$ обозначает последовательность $a_1 a_2 a_3 \dots$ 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 $a_1 \dots a_n$. Тогда запись A^* обозначает процесс



  а также бесконечную последовательность $a_1 \dots a_n a_1 \dots a_n \dots$.

Рассмотрим следующую задачу. Пусть задано множество действий

$$\{!\alpha_1, \ldots, !\alpha_n, !\beta_1, \ldots, !\beta_n\}, \tag{22.6}$$

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. $Act(Sch) = (22.6)$, Sch не содержит терминальных состояний,
2. $\forall\, \pi \in \Pi^0_{Sch}$, если $A = Act(\pi)$, то

$$\left.\begin{array}{l} A_{\{\alpha_1,\ldots,\alpha_n\}} = (!\alpha_1 \ldots !\alpha_n)^* \\ \forall\, i = 1, \ldots, n \;\; A_{\{\alpha_i,\beta_i\}} = (!\alpha_i\, !\beta_i)^* \end{array}\right\} \tag{22.7}$$

3. Для каждой последовательности A действий из (22.6), обладающей свойствами (22.7), существует путь $\pi \in \Pi^0_{Sch}$, такой, что $A = Act(\pi)$.

Свойства (22.7) 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.

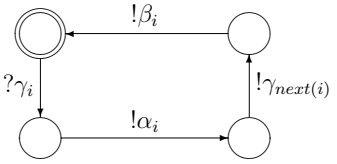
Свойства (22.7) можно выразить равносильным образом в виде наблюдаемой эквивалентности между некоторыми процессами.

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:

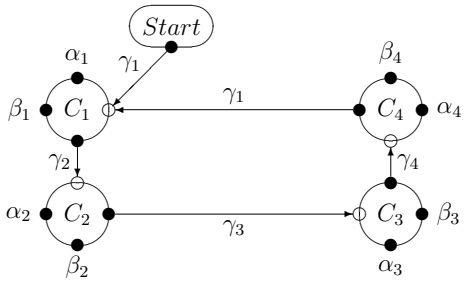
$$Sch \stackrel{\text{def}}{=} (Start \,|\, C_1 \,|\, \ldots \,|\, C_n) \setminus \Gamma, \tag{22.8}$$

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 (22.7) можно выразить следующим соотношением:

$$(Sch \,|\, (?\beta_1)^* \,|\, \ldots \,|\, (?\beta_n)^*) \setminus \{\beta_1, \ldots, \beta_n\} \approx (!\alpha_1 \ldots !\alpha_n)^*. \tag{22.9}$$

Обозначим левую часть (22.9) 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 (22.9) достаточно доказать, что

$$Sch' \approx !\alpha_1; \ldots !\alpha_n; Sch'. \tag{22.10}$$

Мы будем преобразовывать левую часть соотношения (22.10) так, чтобы получилась правая часть этого соотношения. Используя свойства 8, 11 и 12 операций на процессах, сформулированные в параграфе 20.7, можно преобразовать Sch' следующим образом:

$$\begin{aligned} &Sch' = (Sch \,|\, (?\beta_1)^* \,|\, \ldots \,|\, (?\beta_n)^*) \setminus \{\beta_1, \ldots, \beta_n\} = \\ &= \big(\; ((Start \,|\, C_1 \,|\, \ldots \,|\, C_n) \setminus \Gamma) \,|\, (?\beta_1)^* \,|\, \ldots \,|\, (?\beta_n)^* \; \big) \setminus \{\beta_1, \ldots, \beta_n\} = \\ &= (Start \,|\, C_1' \,|\, \ldots \,|\, C_n') \setminus \Gamma, \text{ где } C_i' = (C_i \,|\, (?\beta_i)^*) \setminus \{\beta_i\}. \end{aligned} \tag{22.11}$$

Заметим, что для каждого $i = 1, \ldots, n$ имеет место соотношение

$$C'_i \approx ?\gamma_i; !\alpha_i; !\gamma_{next(i)}; C'_i. \tag{22.12}$$

Действительно, $C_i \equiv ?\gamma_i; !\alpha_i; !\gamma_{next(i)}; !\beta_i; C_i$, и по теореме о разложении

$$\begin{array}{l} C'_i \equiv ((?\gamma_i; !\alpha_i; !\gamma_{next(i)}; !\beta_i; C_i) \,|\, (?\beta_i)^*) \setminus \{\beta_i\} \equiv \\ \equiv ?\gamma_i; !\alpha_i; !\gamma_{next(i)}; \tau; C'_i \approx \text{ правая часть (22.12).} \end{array}$$

Используя данное замечание и теорему о разложении, цепочку равенств (22.11) можно продолжить следующим образом:

$$\begin{array}{l} (Start \,|\, C'_1 \,|\, C'_2 \,|\, \ldots \,|\, C'_n) \setminus \Gamma \approx \\ \approx (\underbrace{!\gamma_1; \mathbf{0}}_{=Start} \mid \underbrace{?\gamma_1; !\alpha_1; !\gamma_2; C'_1}_{\approx C'_1} \,|\, C'_2 \,|\, \ldots \,|\, C'_n) \setminus \Gamma \approx \\ \approx \tau; (\mathbf{0} \,|\, !\alpha_1; !\gamma_2; C'_1 \,|\, C'_2 \,|\, \ldots \,|\, C'_n) \setminus \Gamma = \\ = \tau; (!\alpha_1; !\gamma_2; C'_1 \,|\, C'_2 \,|\, \ldots \,|\, C'_n) \setminus \Gamma \approx \\ \approx \tau; !\alpha_1; (!\gamma_2; C'_1 \,|\, C'_2 \,|\, \ldots \,|\, C'_n) \setminus \Gamma \approx \\ \approx \tau; !\alpha_1; (!\gamma_2; C'_1 \,|\, \underbrace{?\gamma_2; !\alpha_2; !\gamma_3; C'_2}_{\approx C'_2} \,|\, \ldots \,|\, C'_n) \setminus \Gamma \approx \\ \approx \tau; !\alpha_1; \tau; (C'_1 \,|\, !\alpha_2; !\gamma_3; C'_2 \,|\, \ldots \,|\, C'_n) \setminus \Gamma \approx \ldots \approx \\ \approx \tau; !\alpha_1; \tau; !\alpha_2; \ldots \tau; !\alpha_n; (C'_1 \,|\, \ldots \,|\, !\gamma_1; C'_n) \setminus \Gamma \approx \\ \approx !\alpha_1; \ldots !\alpha_n; (C'_1 \,|\, \ldots \,|\, !\gamma_1; C'_n) \setminus \Gamma \approx \\ \approx !\alpha_1; \ldots !\alpha_n; (\underbrace{?\gamma_1; !\alpha_1; !\gamma_2; C'_1}_{\approx C'_1} \mid \ldots \,|\, !\gamma_1; C'_n) \setminus \Gamma \approx \\ \approx !\alpha_1; \ldots !\alpha_n; \underbrace{\tau; (!\alpha_1; !\gamma_2; C'_1 \,|\, \ldots \,|\, C'_n) \setminus \Gamma}. \end{array} \tag{22.13}$$

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 (22.13)

- находится в отношении $\approx$ с правой частью соотношения (22.10) и
- с другой стороны, данное выражение находится в отношении $\approx$ с левой частью соотношения (22.10).

Таким образом, соотношение (22.10) доказано. ■

Читателю предлагается самостоятельно доказать второе свойство в (22.7), а также детерминированность Sch, отсутствие в Sch терминальных состояний, и свойство 3 в списке свойств процесса Sch.

Глава 23

Процессы с передачей сообщений

23.1 Действия с передачей сообщений

В этой главе мы рассматриваем процессы более общего вида, чем процессы, рассмотренные выше. Данные процессы будем называть **процессами с передачей сообщений (ППС)**. Главная особенность ППС заключается в том, что метками переходов в них являются не конкретные действия, а схемы действий, которые мы будем называть **операторами**. Операторы представляют собой последовательности **атомарных операторов (АО)**, каждый из которых относится к одному из трех классов: прием, посылка или внутреннее действие. При выполнении каждого оператора порождается действие, имеющее один из следующих видов:

$$?(\alpha, d), \quad !(\alpha, d), \quad \tau, \quad \text{где } \alpha \in Names \text{ и } d \in \mathcal{D} - \textbf{сообщение}. \qquad (23.1)$$

Действия $?(\alpha, d)$ и $!(\alpha, d)$ понимаются как прием и посылка соответственно сообщения d с тегом (т.е. меткой) α. Ниже в этой главе Act обозначает множество действий видов (23.1).

23.2 Процессы с передачей сообщений

23.2.1 Операторы

Будем считать, что задано множество $Names$, элементы которого рассматриваются как теги (т.е. метки) сообщений, которые могут принимать или посылать процессы.

Атомарные операторы (АО) – это записи следующих видов:

$$?(\alpha, x),\ \ !(\alpha, e),\ \ x := e,\quad \{\varphi\},\\ \text{где } \alpha \in Names,\ x \in Var,\ e \in Tm,\ \tau(x) = \tau(e),\ \varphi \in Fm, \tag{23.2}$$

которые называются **приемом** сообщения x с тегом α, **посылкой** сообщения e с тегом α, **присваиванием** и **условным переходом** соответственно. **Оператор** – это последовательность АО $o_1 \dots o_n$, в которой имеется не более одного приема или посылки. Данная последовательность м.б. пустой, пустая последовательность АО обозначается символом ε. Для удобства восприятия атомарные операторы, входящие в какой-либо оператор, могут разделяться точкой с запятой, кроме того, они могут записываться не в виде последовательности, а в столбик. Оператор, не содержащий приема или посылки, называется **внутренним оператором**. Множество всех операторов обозначается символом $\mathcal{A}$.

Если $A = o_1 \dots o_n$ и $A' = o'_1 \dots o'_{n'}$ – операторы, то AA' обозначает **конкатенацию** A и A', т.е. последовательность $o_1 \dots o_n o'_1 \dots o'_{n'}$, аналогично определяются конкатенации вида Ao, где o – АО.

23.2.2 Понятие процесса с передачей сообщений

Процесс с передачей сообщений (ППС) (называемый ниже просто **процессом**) – это пара $(P, Init(P))$, где

- P – граф с выделенной вершиной P^0 (называемой **начальной вершиной**), каждому ребру которого сопоставлена метка $A \in \mathcal{A}$,
- $Init(P) \in Fm$ – **начальное условие** процесса P.

Для каждого процесса $(P, Init(P))$

- данный процесс обозначается тем же символом P, что и его граф, множество вершин графа P также обозначается символом P, рёбра этого графа обозначаются записями вида $v \xrightarrow{A} v'$, где v, v' – начало и конец ребра соответственно, A – его метка,
- $Var(P)$ обозначает множество всех переменных, входящих в P.

Будем предполагать, что для каждого процесса P

- $Var(P)$ содержит переменную at_P, значениями которой являются вершины графа P,

- для каждого ребра $v \stackrel{A}{\to} v'$ графа P первый АО в метке a этого ребра имеет вид $\{at_P = v\}$, последний АО в метке A этого ребра имеет вид $at_P := v'$, эти АО не будут указываться явно.

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

Процессы P и P' считаются равными, если P' получается из P **переименованием переменных** на новые переменные, которое производится следующим образом: задаётся инъективная функция

$$\eta : Var(P) \to Var \setminus Var(P),$$

и P' получается из P заменой $\forall\, x \in Var(P)$ каждого вхождения x в P на $\eta(x)$.

Ниже будет использоваться следующее соглашение: всякий раз, когда рассматривается какая-либо совокупность процессов, предполагается, что множества переменных процессов из этой совокупности дизъюнктны (если совокупность процессов не обладает этим свойством, то производится подходящее переименование переменных).

23.2.3 Операции на процессах

На процессах с передачей сообщений можно определить операции, аналогичные опредёленным в главе 20 операциям на обычных процессах.

- Пусть заданы процесс P и оператор $A \in \mathcal{A}$. Применением операции **префиксного действия** к паре (A, P) является процесс $A; P$, получаемый добавлением к P новой вершины v и ребра $v \stackrel{A}{\to} P^0$. $(A; P)^0 = v$, $Init(A; P) = Init(P)$.

- Пусть процессы P_1 и P_2 не имеют общих вершин, тогда их **альтернативная композиция** – это процесс $P_1 + P_2$, граф которого получается добавлением к объединению графов P_1 и P_2 новой вершины P^0 и рёбер вида $P^0 \stackrel{A}{\to} v$ для каждого ребра вида $P_i^0 \stackrel{A}{\to} v$ $(i = 1, 2)$. $(P_1 + P_2)^0 = v$, $Init(P_1 + P_2) = Init(P_1) \wedge Init(P_2)$.

- **Параллельной композицией** процессов P_1 и P_2 называется процесс $P_1 \,|\, P_2$, вершинами которого являются пары вида (v_1, v_2), где $v_i \in P_i$ $(i = 1, 2)$ и который содержит следующие ребра:

– $(v_1, v_2) \xrightarrow{A} (v_1', v_2)$, где $v_1 \xrightarrow{A} v_1'$ – ребро P_1 и $v_2 \in P_2$,

– $(v_1, v_2) \xrightarrow{A} (v_1, v_2')$, где $v_1 \in P_1$ и $v_2 \xrightarrow{A} v_2'$ – ребро P_2,

– $(v_1, v_2) \xrightarrow{A} (v_1', v_2')$, если P_i содержит ребро $v_i \xrightarrow{A_i} v_i'$ $(i = 1, 2)$ и один из операторов A_1, A_2 представляется в виде конкатенации $A_1'?(\alpha, x)A_1''$, а другой – в виде конкатенации $A_2'!(\alpha, e)A_2''$, где $\tau(x) = \tau(e)$, $A = A_1'A_2'(x := e)A_1''A_2''$.

$(P_1|P_2)^0 = (P_1^0, P_2^0)$, $Init(P_1|P_2) = Init(P_1) \wedge Init(P_2)$.

- Пусть заданы процесс P и подмножество $L \subseteq Names$.

  Ограничением P по L называется процесс $P \setminus L$, получаемый из P удалением рёбер, метки которых содержат имена из L.

- Пусть заданы процесс P и функция $\theta : Names \to Names$.

  Переименованием P относительно θ называется процесс P^θ, получаемый из P заменой имён в метках рёбер: каждое имя α заменяется на $\theta(\alpha)$.

23.2.4 Редукция процессов

Пусть заданы процесс P и вершина $v \in P$. Если множества всех ребер в P с концом в v и с началом в v непусты и имеют вид

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

соответственно, где v отличается от всех вершин v_i и v_i' и либо все операторы $A_1, \ldots, A_n$ внутренние, либо все операторы $A_1', \ldots, A_{n'}'$ внутренние, то к графу P можно применить операцию **редукции**, которая заключается в преобразовании P путем удаления вершины v и связанных с нею ребер и добавления ребер вида $v_i \xrightarrow{A_iA_{i'}'} v_{i'}'$, где $i = 1, \ldots, n, i' = 1, \ldots, n'$, и $A_iA_{i'}'$ – конкатенация последовательностей A_i и $A_{i'}'$.

23.3 Наблюдаемая эквивалентность процессов

23.3.1 Система переходов процесса

Состояние процесса P – это произвольная подстановка из $Var(P)^\bullet$.

Системой переходов процесса P называется граф Σ_P, вершинами которого являются состояния процесса P, а ребра помечены действиями

$$?(\alpha, d), \quad !(\alpha, d), \quad \text{или} \quad \tau, \quad \text{где } \alpha \in Names \text{ и } d \in \mathcal{D}. \tag{23.3}$$

Рёбра графа Σ_P называются **переходами**. Множество вершин графа Σ_P обозначается той же записью Σ_P. Переход с началом θ, концом θ' и меткой a обозначается записью $\theta \stackrel{a}{\to} \theta'$. Связь между состояниями θ и θ', входящими в переход $\theta \stackrel{a}{\to} \theta'$, понимается следующим образом: если P находится в состоянии θ, то он может выполнить действие a, после чего перейти в состояние θ'. Σ_P содержит переход $\theta \stackrel{a}{\to} \theta'$, если P содержит ребро $at_P^{\theta} \stackrel{A}{\to} at_P^{\theta'}$, обладающее следующим свойством: пусть A имеет вид $o_1 \dots o_n$, тогда существует последовательность состояний $\theta_1, \theta_2, \dots, \theta_{n+1}$ процесса P, такая, что $\theta_1 = \theta$, $\theta_{n+1} = \theta'$ и $\forall\, i = 1, \dots, n$

$$\begin{array}{l} \text{если } o_i = ?(\alpha, x), \text{ то } a = ?(\alpha, d),\ \theta_{i+1} = (d/x)\theta_i, \\ \text{если } o_i = !(\alpha, e), \text{ то } a = !(\alpha, e^{\theta_i}),\ \theta_{i+1} = \theta_i, \\ \text{если } o_i = (x := e), \text{ то } \theta_{i+1} = (e/x)\theta_i, \\ \text{если } o_i = \{\varphi\}, \text{ то } \varphi^{\theta_i} = 1,\ \theta_{i+1} = \theta_i, \end{array} \tag{23.4}$$

и если A не содержит приема или посылки, то $a = \tau$.

Заметим, что если переход $\theta \stackrel{\tau}{\to} \theta'$ в Σ_P определяется ребром $v \stackrel{A}{\to} v'$ в P, то состояние θ' однозначно определяется состоянием θ и оператором A, мы будем обозначать данную связь записью $\theta' = \theta A$.

Добавим к графу Σ_P новую вершину θ_P^0 и рёбра вида $\theta_P^0 \stackrel{a}{\to} \theta'$ для каждого ребра графа Σ_P вида $\theta \stackrel{a}{\to} \theta'$, где $Init(P^{\theta}) = 1$, $at_P^{\theta} = P^0$. Получившийся граф можно рассматривать как процесс в исходном смысле этого понятия (см. пункт 19.3) над множеством Act действий вида (23.3) с начальным состоянием θ_P^0. Данный процесс будем обозначать Σ_P^0.

Будем называть процессы P и P' **наблюдаемо эквивалентными**, если $\Sigma_P^0 \approx \Sigma_{P'}^0$, где $\approx$ – отношение наблюдаемой эквивалентности на обычных процессах, определённое в пункте 21.3.1.

23.3.2 Метод доказательства наблюдаемой эквивалентности процессов

Один из методов доказательства наблюдаемой эквивалентности процессов основан на нижеследующей теореме 52. Для её формулировки и доказательства введём вспомогательные понятия и обозначения.

1. Пусть заданы процесс P и пара вершин $v, v' \in P$.

Простой путь из v в v' – это последовательность π рёбер P вида

$$v = v_0 \stackrel{A_1}{\to} v_1, \quad v_1 \stackrel{A_2}{\to} v_2, \quad \ldots \quad v_{n-1} \stackrel{A_n}{\to} v_n = v' \tag{23.5}$$

такая, что среди АО, входящих в конкатенацию $A_1 \ldots A_n$, имеется не более одного приема или посылки. Последовательность (23.5) м.б. пустой, в этом случае $v = v'$.

Если π имеет вид (23.5), то A_π обозначает конкатенацию $A_1 \ldots A_n$.

Запись $v \stackrel{\pi}{\to} v'$ означает, что π – простой путь из v в v'.

2. Пусть заданы формула φ и операторы A_1, A_2, тогда $[A_1, A_2]\varphi$ обозначает формулу Φ, определяемую рекурсивно:
   - если A_1 и A_2 – пустые последовательности, то $\Phi = \varphi$,
   - пусть один из операторов A_1, A_2 является конкатенацией Ao, а другой – конкатенацией $A'o'$, тогда
     - если $o = ?(\alpha, x)$, $o' = ?(\alpha, y)$, $\tau(x) = \tau(y)$, то $\Phi = [A, A']\varphi^{(z/x, z/y)}$, где z – новая переменная (т.е. z не входит в A_1, A_2, φ),
     - если $o = !(\alpha, e)$, $o' = !(\alpha, e')$, то $\Phi = [A, A']\{e = e', \varphi\}$,
   - пусть один из операторов A_1, A_2 является конкатенацией Ao, где o – присваивание или условный переход, и A' – другой из этих операторов, тогда $\Phi = [A, A']o(\varphi)$, где

$$o(\varphi) = \begin{cases} \varphi^{(e/x)} & \text{в случае } o = (x := e) \text{ и} \\ \psi \to \varphi & \text{в случае } o = \{\psi\}, \end{cases}$$

   - иначе $\Phi = 0$,
3. $\forall\, \varphi, \psi \in Fm$ запись $\varphi \le \psi$ означает, что $(\varphi \to \psi) = 1$.

Теорема 52

Процессы P_1 и P_2 наблюдаемо эквивалентны, если существует совокупность M формул из $Fm(Var(P_1) \sqcup Var(P_2))$, индексированных некоторыми парами $v_1 v_2$, где $v_1 \in P_1$, $v_2 \in P_2$, и обладающая свойствами:

(0) $\mu_{P_1^0 P_2^0} \in M$, $0 \ne \{Init(P_1) \wedge Init(P_2)\} \le \mu_{P_1^0 P_2^0}$,

(1) $\forall\, \mu_{v_1 v_2} \in M$, если P_1 содержит ребро $v_1 \stackrel{A}{\to} v_1'$, то существует совокупность $\{v_2 \stackrel{\pi_i}{\to} v_2^i \mid i \in I\}$ простых путей в P_2, такая, что

$$\mu_{v_1 v_2} \le \bigvee\nolimits_{i \in I} [A, A_{\pi_i}] \mu_{v_1' v_2^i}, \tag{23.6}$$

(2) свойство, симметричное предыдущему свойству: $\forall\, \mu_{v_1v_2} \in M$, если P_2 содержит ребро $v_2 \stackrel{A}{\to} v_2'$, то существует совокупность простых путей в P_1 $\{v_1 \stackrel{\pi_i}{\to} v_1^i \mid i \in I\}$, такая, что

$$\mu_{v_1v_2} \le \bigvee\nolimits_{i\in I} [A_{\pi_i}, A]\mu_{v_1^i v_2'}. \tag{23.7}$$

Доказательство.

Существует биекция между $(Var(P_1) \sqcup Var(P_2))^\bullet$ и $\Sigma_{P_1} \times \Sigma_{P_2}$, поэтому каждую пару $(\theta_1, \theta_2) \in \Sigma_{P_1} \times \Sigma_{P_2}$ можно рассматривать как подстановку из $(Var(P_1) \sqcup Var(P_2))^\bullet$. Определим отношение $\rho \subseteq \Sigma_{P_1} \times \Sigma_{P_2}$:

$$\begin{aligned}\rho = \{(\theta_1, \theta_2) \in \Sigma_{P_1} \times \Sigma_{P_2} \mid \\ \exists\, \mu_{v_1v_2} \in M : v_i = at_{P_i}^{\theta_i}\ (i = 1, 2), \mu_{v_1v_2}^{(\theta_1,\theta_2)} = 1\}.\end{aligned} \tag{23.8}$$

Пусть $(\theta_1, \theta_2) \in \rho$, т.е. $\exists\, \mu_{v_1v_2} \in M : v_i = at_{P_i}^{\theta_i}\ (i = 1, 2), \mu_{v_1v_2}^{(\theta_1,\theta_2)} = 1$.

Докажем, что выполнены свойства (21.16) из пункта 21.3.2. Мы рассмотрим лишь половину этих свойств:

$$\begin{cases} \forall\, \theta_1' : \theta_1 \stackrel{\tau}{\to} \theta_1'\ \ \exists\, \theta_2' : \theta_2 \stackrel{\tau^*}{\to} \theta_2' \text{ и } (\theta_1', \theta_2') \in \rho, \\ \forall\, a \in Act \setminus \{\tau\}, \forall\, \theta_1' : \theta_1 \stackrel{a}{\to} \theta_1'\ \ \exists\, \theta_2' : \theta_2 \stackrel{\tau^* a \tau^*}{\longrightarrow} \theta_2' \text{ и } (\theta_1', \theta_2') \in \rho \end{cases} \tag{23.9}$$

(рассуждения для другой половины свойств (21.16) аналогичны).

Из $\mu_{v_1v_2}^{(\theta_1,\theta_2)} = 1$ и (23.6) следует, что $\exists\, i \in I$:

$$\Big([A, A_{\pi_i}]\mu_{v_1'v_2^i}\Big)^{(\theta_1,\theta_2)} = 1. \tag{23.10}$$

Докажем первое свойство в (23.9). Из $\theta_1 \stackrel{\tau}{\to} \theta_1'$ следует, что в P_1 есть ребро $v_1 \stackrel{A}{\to} v_1'$, где A – внутренний, $v_1' = at_{P_1}^{\theta_1'}$, $\theta_1' = \theta_1 A$.

Рассмотрим раздельно случаи, когда путь π_i пуст и когда он непуст.

- π_i пуст, тогда $v_2 = v_2^i$, $\theta_2 = \theta_2'$ и $\Big([A, \varepsilon]\mu_{v_1'v_2}\Big)^{(\theta_1,\theta_2)} = 1$.

  Пусть $A = o_1 \dots o_n$, тогда $1 = \Big([A, \varepsilon]\mu_{v_1'v_2}\Big)^{(\theta_1,\theta_2)} =$
  $= \Big([o_1 \dots o_n, \varepsilon]\mu_{v_1'v_2}\Big)^{(\theta_1,\theta_2)} = \Big([\varepsilon, \varepsilon]o_1(\dots o_n(\mu_{v_1'v_2})\dots)\Big)^{(\theta_1,\theta_2)} =$
  $= o_1(\dots o_n(\mu_{v_1'v_2})\dots)^{(\theta_1,\theta_2)} = \mu_{v_1'v_2}^{\theta_1 A, \theta_2} = \mu_{v_1'v_2}^{\theta_1', \theta_2}$.

- π_i непуст, и верно (23.10). Определим $\theta_2' \stackrel{\text{def}}{=} \theta_2 A_{\pi_i}$. Нетрудно доказать, что $\theta_2 \stackrel{\tau^*}{\to} \theta_2'$, и $(\theta_1', \theta_2') \in \rho$, т.к.

$$1 = \Big([A, A_{\pi_i}]\mu_{v_1'v_2^i}\Big)^{(\theta_1,\theta_2)} = \mu_{v_1'v_2^i}{}^{(\theta_1 A, \theta_2 A_{\pi_i})} = \mu_{v_1'v_2^i}^{(\theta_1', \theta_2')}. \tag{23.11}$$

Докажем второе свойство в (23.9). Рассмотрим возможные виды a.

- $a = ?(\alpha, d)$. Из $\theta_1 \stackrel{?(\alpha,d)}{\longrightarrow} \theta_1'$ следует, что в P_1 есть ребро $v_1 \stackrel{A}{\to} v_1'$, где
  - $A = A_1'?(\alpha, x)A_1''$, причем A_1' и A_1'' – внутренние,
  - $v_1' = at_{P_1}^{\theta_1'}$, $\theta_1' = ((d/x)(\theta_1 A_1'))A_1''$.

  Из свойства (23.10), определения формул вида $[A_1, A_2]\varphi$ и определения системы переходов процесса следует, что A_{π_i} можно представить в виде конкатенации $A_2'?(\alpha, y)A_2''$, где A_2' и A_2'' – внутренние.

  Определим $\theta_2' = ((d/y)(\theta_2 A_2'))A_2''$.

  Нетрудно доказать, что $\theta_2 \stackrel{\tau^* ?(\alpha,d) \tau^*}{\longrightarrow} \theta_2'$.

  Докажем, что $(\theta_1', \theta_2') \in \rho$, т.е. $\mu_{v_1' v_2^i}^{(\theta_1', \theta_2')} = 1$.

  Из определения формул вида $[A_1, A_2]\varphi$ следует, что

$$\begin{array}{l} [A, A_{\pi_i}]\mu_{v_1' v_2^i} = \\ = [A_1'?(\alpha, x)A_1'', A_2'?(\alpha, y)A_2'']\mu_{v_1' v_2^i} = \\ = [A_1'?(\alpha, x), A_2'?(\alpha, y)][A_1'', A_2'']\mu_{v_1' v_2^i} = \\ = [A_1', A_2']([A_1'', A_2'']\mu_{v_1' v_2^i})^{(z/x, z/y)}. \end{array} \tag{23.12}$$

  Из (23.12) и (23.10) следует равенство

$$\left(([A_1'', A_2'']\mu_{v_1' v_2^i})^{(z/x, z/y)}\right)^{(\theta_1 A_1', \theta_2 A_2')} = 1,$$

  частным случаем которого является равенство

$$\left(([A_1'', A_2'']\mu_{v_1' v_2^i})^{(d/x, d/y)}\right)^{(\theta_1 A_1', \theta_2 A_2')} = 1.$$

  Последнее равенство можно переписать в виде

$$\left([A_1'', A_2'']\mu_{v_1' v_2^i}\right)^{((d/x)(\theta_1 A_1'), (d/y)(\theta_2 A_2'))} = 1,$$

  откуда следует, что $\mu_{v_1' v_2^i}^{(\theta_1', \theta_2')} = \mu_{v_1' v_2^i}^{(((d/x)(\theta_1 A_1'))A_1'', ((d/y)(\theta_2 A_2'))A_2'')} = 1$.

- $a = !(\alpha, d)$. Из $\theta_1 \stackrel{!(\alpha,d)}{\longrightarrow} \theta_1'$ следует, что в P_1 есть ребро $v_1 \stackrel{A}{\to} v_1'$, где
  - $A = A_1'!(\alpha, e_1)A_1''$, причем A_1' и A_1'' – внутренние,
  - $v_1' = at_{P_1}^{\theta_1'}$, $d = e_1^{\theta_1 A_1'}$, $\theta_1' = (\theta_1 A_1')A_1''$.

Из свойства (23.10), определения формул вида $[A_1, A_2]\varphi$ и определения системы переходов процесса следует, что A_{π_i} можно представить в виде конкатенации $A'_2!(\alpha, e_2)A''_2$, где A'_2 и A''_2 – внутренние.

Из определения формул вида $[A_1, A_2]\varphi$ следует, что

$$\begin{array}{l}[A, A_{\pi_i}]\mu_{v'_1 v^i_2} = [A'_1!(\alpha, e_1)A''_1, A'_2!(\alpha, e_2)A''_2]\mu_{v'_1 v^i_2} = \\ = [A'_1!(\alpha, e_1), A'_2!(\alpha, e_2)][A''_1, A''_2]\mu_{v'_1 v^i_2} = \\ = [A'_1, A'_2]\{e_1 = e_2, [A''_1, A''_2]\mu_{v'_1 v^i_2}\}.\end{array} \tag{23.13}$$

Из (23.10) и (23.13) следует, что

$$\{e_1 = e_2, [A''_1, A''_2]\mu_{v'_1 v^i_2}\}^{(\theta_1 A'_1, \theta_2 A'_2)} = 1. \tag{23.14}$$

Из (23.14) следует, что

$$\begin{array}{l} d = e_1^{\theta_1 A'_1} = e_2^{\theta_2 A'_2}, \\ \Big([A''_1, A''_2]\mu_{v'_1 v^i_2}\Big)^{(\theta_1 A'_1, \theta_2 A'_2)} = 1. \end{array} \tag{23.15}$$

Определим $\theta'_2 = (\theta_2 A'_2)A''_2$.

Из первой строки в (23.15) следует, что $\theta_2 \overset{\tau^*!(\alpha,d)\tau^*}{\longrightarrow} \theta'_2$.

Из второй строки в (23.15) следует, что

$$\mu^{(\theta'_1, \theta'_2)}_{v'_1 v^i_2} = \Big([A''_1, A''_2]\mu_{v'_1 v^i_2}\Big)^{(\theta_1 A'_1, \theta_2 A'_2)} = 1,$$

т.е. $(\theta'_1, \theta'_2) \in \rho$.

Таким образом, доказано, что отношение $\rho \subseteq \Sigma_{P_1} \times \Sigma_{P_2}$, определенное в (23.8), обладает свойством: $\forall\, (\theta_1, \theta_2) \in \rho$ выполнены свойства (21.16) из пункта 21.3.2. Используя условие (0) в формулировке теоремы, отсюда нетрудно заключить, что $\rho \cup \{(\theta^0_{P_1}, \theta^0_{P_2})\}$ является НБМ между $\Sigma^0_{P_1}$ и $\Sigma^0_{P_2}$, т.е. P_1 и P_2 наблюдаемо эквивалентны. ■

23.4 Примеры верификации процессов с передачей сообщений

В этом параграфе мы излагаем несколько примеров процессов с передачей сообщений и рассматриваем задачи доказательства некоторых свойств

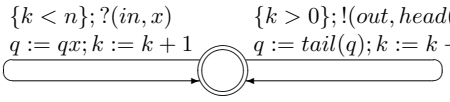
этих процессов. Будем называть имена, входящие в действия этих процессов, **портами**, и действия вида $?(\alpha, x)$ и $!(\alpha, e)$ будем называть приемом и посылкой соответственно значения с порта α. Как и для обычных процессов, будем использовать потоковые графы для наглядного представления взаимодействия процессов, входящих в параллельную композицию.

23.4.1 Последовательная композиция буферов

В этом пункте рассматривается процесс Buffer_n, моделирующий работу буфера, хранящего не более n значений. В каждый момент времени буфер содержит список q значений длины не более n и может выполнить следующие действия:

- прием значения с порта in и запись этого значения в конец q и
- посылка $head(q)$ с порта out, и удаление этого значения из q.

Граф процесса Buffer_n имеет вид



(23.16)

Переменные, входящие в Buffer_n, имеют следующий смысл:

- n и k – переменные типа $\mathbf{N}$, содержащие максимальное и текущее соответственно количество значений, находящихся в буфере,
- x – переменная, в которую записывается очередное значение, поступившее в буфер, q – переменная типа $*$, в которой хранится список принятых, но пока не выведенных значений.

Начальное условие: $\mathit{Init}(\mathit{Buffer}_n) = \{n > 0, k = 0, q = \varepsilon\}$.

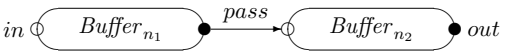
Buffer_n имеет два перехода: левый переход в (23.16) связан с выполнением приёма (который может быть выполнен, только если $k < n$, т.е. в буфере есть место для записи нового значения), а правый переход в (23.16) – с выполнением посылки (которая может быть выполнена, только если $k > 0$, т.е. буфер непуст).

Пусть заданы два буфера: Buffer_{n_1} и Buffer_{n_2}. Эты буферы можно объединить в последовательную композицию путем соединения выхода первого буфера с входом второго буфера (т.е. значения, которые

посылает первый буфер, сразу же принимает второй буфер). Процесс $SeqComp_{n_1,n_2}$, соответствующий такому объединению, имеет вид

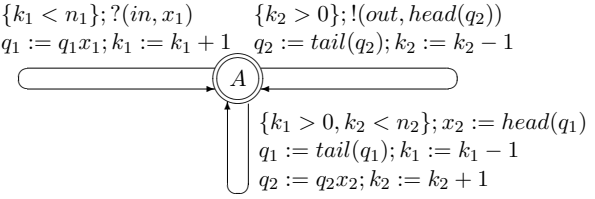
$$(Buffer_{n_1}(pass/out) \mid Buffer_{n_2}(pass/in)) \setminus \{pass\}, \tag{23.17}$$

где $pass \notin \{in, out\}$. Потоковый граф процесса $SeqComp_{n_1,n_2}$ имеет вид



Докажем, что $SeqComp_{n_1,n_2}$ и $Buffer_{n_1+n_2}$ наблюдаемо эквивалентны.

Согласно определению операций на процессах, граф процесса (23.17) выглядит следующим образом (мы обозначаем переменные в $Buffer_{n_i}$ $(i = 1, 2)$ теми же символами, что и соответствующие переменные в $Buffer_n$, с добавлением индекса i):

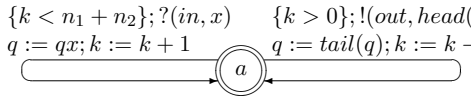


(23.18)

Начальное условие процесса $SeqComp_{n_1,n_2}$ имеет вид

$$Init(SeqComp_{n_1,n_2}) = \{Init(Buffer_{n_1}), Init(Buffer_{n_2})\} = \\ = \left\{ \begin{array}{l} n_1 > 0, k_1 = 0, q_1 = \varepsilon, \\ n_2 > 0, k_2 = 0, q_2 = \varepsilon \end{array} \right\}.$$

Граф процесса $Buffer_{n_1+n_2}$ имеет вид



(23.19)

Начальное условие процесса $Buffer_{n_1+n_2}$ имеет вид

$$Init(Buffer_{n_1+n_2}) = \{n_1 + n_2 > 0, k = 0, q = \varepsilon\}.$$

Нетрудно проверить, что формула

$$Inv \stackrel{\text{def}}{=} \left\{ \begin{array}{l} 0 \le k_1 \le n_1, |q_1| = k_1, n_1 > 0 \\ 0 \le k_2 \le n_2, |q_2| = k_2, n_2 > 0 \end{array} \right\}$$

является инвариантом процесса $SeqComp_{n_1,n_2}$ (т.е. истинна в каждый момент выполнения этого процесса).

Для доказательства наблюдаемой эквивалентности процессов (23.18) и (23.19) на основе теоремы 52 определим $M = \{\mu_{Aa}\}$, где

$$\mu_{Aa} \stackrel{\text{def}}{=} \{q = q_2 q_1, k = k_2 + k_1, Inv\}.$$

Проверим свойства (0), (1), (2) в формулировке теоремы 52.

- Свойство (0) следует из неравенства

$$\left\{ \begin{array}{l} n_1 > 0, k_1 = 0, q_1 = \varepsilon, \\ n_2 > 0, k_2 = 0, q_2 = \varepsilon, \\ n_1 + n_2 > 0, k = 0, q = \varepsilon \end{array} \right\} \leq \{q = q_2 q_1, k = k_2 + k_1\}.$$

- Проверим свойство (1).

  – Для левого ребра в (23.18) совокупность $\{v_2 \stackrel{\pi_i}{\to} v_2^i \mid i \in I\}$ состоит из левого ребра в (23.19). Неравенство (23.6) в данном случае следует из неравенства

$$\left\{ \begin{array}{l} k_1 < n_1 \\ q = q_2 q_1 \\ k = k_2 + k_1 \\ Inv \end{array} \right\} \leq \left\{ \begin{array}{l} k < n_1 + n_2 \\ qz = q_2 q_1 z \\ k + 1 = k_2 + k_1 + 1 \end{array} \right\}.$$

  – Для среднего ребра в (23.18) совокупность $\{v_2 \stackrel{\pi_i}{\to} v_2^i \mid i \in I\}$ пуста. Неравенство (23.6) в данном случае следует из неравенства

$$\left\{ \begin{array}{l} q = q_2 q_1 \\ k = k_2 + k_1 \\ Inv \\ k_1 > 0 \\ k_2 < n_2 \end{array} \right\} \leq \left\{ \begin{array}{l} q = q_2\, head(q_1)\, tail(q_1) \\ k = k_2 + 1 + k_1 - 1 \end{array} \right\}.$$

  – Для правого ребра в (23.18) совокупность $\{v_2 \stackrel{\pi_i}{\to} v_2^i \mid i \in I\}$ состоит из правого ребра в (23.19). Неравенство (23.6) в данном случае следует из неравенства

$$\left\{ \begin{array}{l} k_2 > 0 \\ q = q_2 q_1 \\ k = k_2 + k_1 \\ Inv \end{array} \right\} \leq \left\{ \begin{array}{l} k > 0 \\ head(q_2) = head(q) \\ tail(q) = tail(q_2) q_1 \\ k - 1 = k_2 - 1 + k_1 \end{array} \right\}.$$

- Проверим свойство (2).

  – Для левого ребра в (23.19) совокупность $\{v_1 \stackrel{\pi_i}{\to} v_1^i \mid i \in I\}$ состоит из пары путей, первый из которых – левое ребро в (23.18), а второй состоит из среднего и левого ребра в (23.18). Неравенство (23.6) в данном случае следует из неравенства

$$\left\{\begin{array}{l} k < n_1 + n_2 \\ q = q_2 q_1 \\ k = k_2 + k_1 \\ Inv \end{array}\right\} \le (k_1 < n_1) \ \vee \ \left\{\begin{array}{l} k_1 > 0 \\ k_2 < n_2 \\ k_1 - 1 < n_1 \end{array}\right\}.$$

  – Для правого ребра в (23.19) совокупность $\{v_1 \stackrel{\pi_i}{\to} v_1^i \mid i \in I\}$ состоит из пары путей, первый из которых – правое ребро в (23.18), а второй состоит из среднего и правого ребра в (23.18). Неравенство (23.6) в данном случае следует из неравенства

$$\left\{\begin{array}{l} k > 0 \\ q = q_2 q_1 \\ k = k_2 + k_1 \\ Inv \end{array}\right\} \le (k_2 > 0) \ \vee \ \left\{\begin{array}{l} k_1 > 0 \\ k_2 < n_2 \\ k_2 + 1 > 0 \end{array}\right\}.$$

23.4.2 Разделение мультимножеств

Напомним, что **мультимножеством** называется произвольная индексированная совокупность $X = \{x_i \mid i \in I\}$ каких-либо объектов, в которой объекты с разными индексами могут совпадать. Мультимножество называется конечным, если множество I индексов конечно, и в этом случае $|X|$ обозначает число элементов в I. Обозначим символом $\mathcal{U}$ совокупность конечных мультимножеств действительных чисел. $\forall\, X, Y \in \mathcal{U}$ запись $X \cup Y$ обозначает мультимножество, число вхождений в которое каждого элемента x равно сумме чисел вхождений x в X и Y.

$\forall\, X, Y \in \mathcal{U}$ записи вида $X \le Y$, $\max(X)$ и т.п. понимаются естественным образом (т.е. как утверждение $\forall\, x \in X, \forall\, y \in Y\ x \le y$ и т.д.).

Задача разделения мультимножеств имеет следующий вид. Задана пара L, R мультимножеств из $\mathcal{U}$ (будем называть их **левым** и **правым** мультимножеством соответственно). Требуется преобразовать эту пару в такую пару мультимножеств L', R', чтобы

$$L' \cup R' = L \cup R, \ \ |L'| = |L|, \ \ |R'| = |R|, \ \ L' \le R'. \tag{23.20}$$

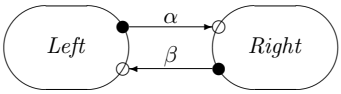
Предлагаемый метод решения данной задачи заключается в выполнении нескольких сеансов обмена элементами между левым и правым

мультимножествами, где каждый сеанс состоит из следующих действий: нахождение максимального элемента mx в левом мультимножестве, нахождение минимального элемента mn в правом мультимножестве и перенесение mx из левого мультимножества в правое, а mn – из правого мультимножества в левое.

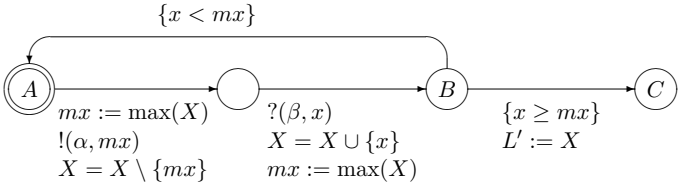
Для решения этой задачи предлагается использовать процесс

$$(\mathit{Left} \mid \mathit{Right}) \setminus \{\alpha, \beta\}, \tag{23.21}$$

где процессы *Left* и *Right* выполняют операции, связанные с левым и правым мультимножеством соответственно. Потоковый граф процесса (23.21) имеет вид

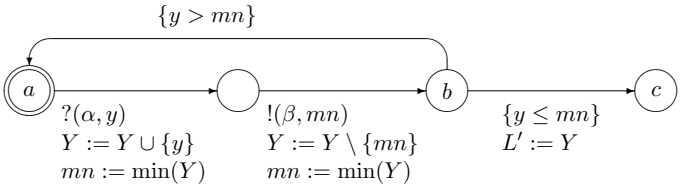


Процесс *Left* использует переменную X для хранения текущего левого мультимножества, имеет начальное условие $\mathit{Init}(\mathit{Left}) = (X = L)$, и его граф имеет следующий вид:



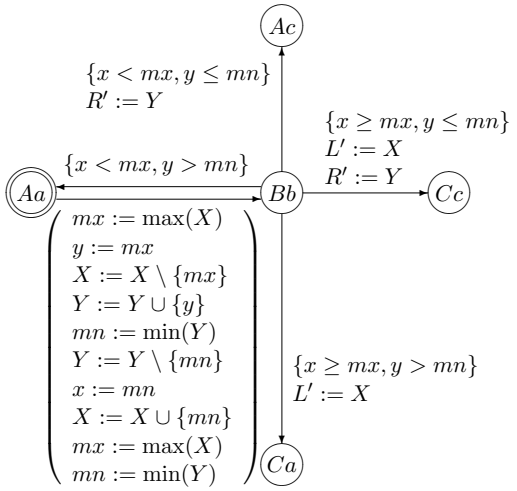
(23.22)

Процесс *Right* использует переменную Y для хранения текущего правого мультимножества, имеет начальное условие $\mathit{Init}(\mathit{Right}) = (Y = R)$, и его граф имеет следующий вид:



(23.23)

Редуцированный процесс (23.21) выглядит следующим образом:



(23.24)

На данном графе видно, что у процесса (23.24) имеются такие вершины (Ac и Ca), из которых не выходит ни одного ребра, т.е., попав в которые, процесс не может продолжать свою работу, но попадание в эти вершины не является нормальным завершением работы процесса (такие вершины называются **тупиковыми**). Нормальным завершением является попадание в вершину Cc, что означает, что оба процесса *Left* и *Right* достигли своих терминальных вершин C и c соответственно.

Процесс (23.21) действительно может попасть в одну из тупиковых вершин, например при $L = \{3\}$, $R = \{1, 2\}$.

Тем не менее процесс (23.21) обладает следующими свойствами:

- процесс (23.21) всегда попадает в одну из вершин Ac, Ca, Cc и
- при попадании процесса в какую-либо из этих вершин выполняются соотношения

$$X \cup Y = L \cup R, \;\; |X| = |L|, \;\; |Y| = |R|, \;\; X \leq Y. \tag{23.25}$$

Для обоснования этих свойств мы будем использовать функцию

$$f(X, Y) \stackrel{\text{def}}{=} |\{(x, y) \in X \times Y \mid x > y\}|.$$

При анализе последовательности операторов присваивания, выполняемых при переходе от Aa к Bb, удобно представлять эту последовательность схематически как последовательность действий:

1. $X \xrightarrow{y:=\max(X)} Y$ (перенесение элемента $y := \max(X)$ из X в Y).
2. $Y \xrightarrow{x:=\min(Y)} X$ (перенесение элемента $x := \min(Y)$ из Y в X).
3. $mx := \max(X)$, $mn := \min(Y)$.

Нетрудно установить, что имеют место следующие утверждения.

1. Если в текущий момент времени i процесс находится в вершине Aa и значения X_i, Y_i переменных X и Y в момент i таковы, что $f(X_i, Y_i) = 0$, (т.е. $X_i \le Y_i$), то $X_{i+1} = X_i$ и $Y_{i+1} = Y_i$. Кроме того, после выполнения перехода от Aa к Bb значения переменных x, y, mx и mn будут удовлетворять соотношениям $y = x = mx \le mn$, и, таким образом, следующим переходом будет переход из Bb в Cc, т.е. процесс нормально завершит свою работу. При этом значения L' и R' равны X_i и Y_i соответственно, поэтому верно (23.20).

2. Если в текущий момент i процесс находится в Aa и $f(X_i, Y_i) > 0$, то после выполнения перехода от вершины Aa к вершине Bb (т.е. в момент $i+1$) новые значения X_{i+1}, Y_{i+1} переменных X и Y будут удовлетворять неравенству

$$f(X_{i+1}, Y_{i+1}) < f(X_i, Y_i). \tag{23.26}$$

   Кроме того, значения переменных x, y, mx, mn в момент $i+1$ будут удовлетворять соотношениям

   $$y = \max(X_i), \quad x = \min(Y_i), \quad mx = \max(X_{i+1}), \quad mn = \min(Y_{i+1}),$$
   $$x \le y, \quad x \le mx, \quad mn \le y.$$

   Отсюда следует, что если в момент $i+1$ процесс будет переходить из Bb в одну из вершин Ac, Cc или Ca, то это возможно:

   (a) либо если $x = mx$, в этом случае $X_{i+1} \le mx = x \le Y_i$, откуда, используя $x \le y$ и $Y_{i+1} \subseteq Y_i \cup \{y\}$, получаем:

   $$X_{i+1} \le Y_{i+1}, \tag{23.27}$$

   (b) либо если $y = mn$, в этом случае $X_i \le y = mn \le Y_{i+1}$, откуда, используя $x \le y$ и $X_{i+1} \subseteq X_i \cup \{x\}$, получаем (23.27).

Таким образом, при попадании в какую-либо из вершин Ac, Cc или Ca верно неравенство $X \le Y$. Истинность других соотношений в (23.25) усматривается непосредственно.

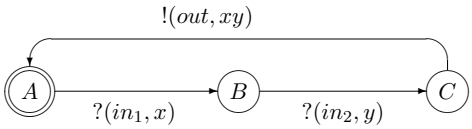
Из вышесказанного следует, что данный процесс не может зациклиться, так как зацикливание возможно только в том случае, когда процесс бесконечно часто попадает в Aa, и при каждом попадании в Aa значение функции f на текущих значениях переменных X, Y положительно. Невозможность данной ситуации следует из (23.26), и свойства фундированности множества натуральных чисел (т.е. из того, что в данном множестве нет бесконечных убывающих цепей).

Читателю предлагается самостоятельно

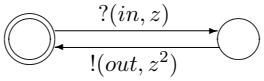
- найти необходимые и достаточные условия, которым должны удовлетворять L и R, чтобы приведённый выше алгоритм завершал свою работу с этими L и R нормально (т.е. в вершине Cc), и
- разработать такой алгоритм разделения мультимножеств, который бы работал корректно на любых L и R.

23.4.3 Вычисление квадрата

Предположим, что мы имеем систему «умножитель», работа которой заключается в том, что она периодически получает на порты in_1 и in_2 два значения и выдаёт на порте out их произведение. Поведение умножителя описывается процессом Mul:

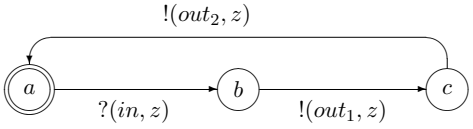


Используя этот умножитель, мы хотим построить систему «вычислитель квадрата», поведение которой описывается процессом $Square\_Spec$:



Искомую систему мы построим как композицию

- вспомогательной системы «дубликатор», поведение которой описывается процессом Dup:



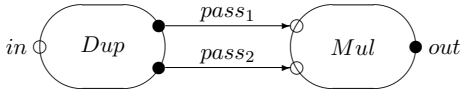
т.е. дубликатор копирует свой вход на два выхода, и

- умножителя Mul, который будет получать на свои входные порты те значения, которые будет выдавать дубликатор.

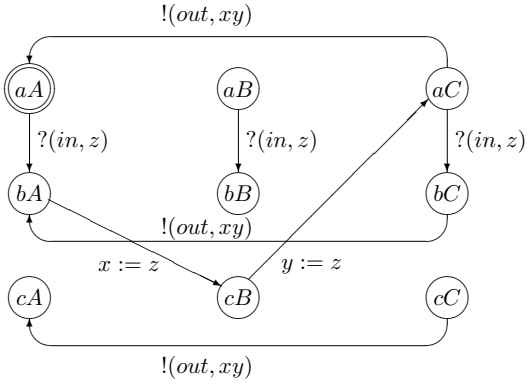
Процесс $Square$, соответствующий такой композиции, определяется следующим образом:

$$Square \stackrel{\text{def}}{=} \left(\begin{array}{l} Dup(pass_1/out_1, pass_2/out_2) \ \mid \\ \mid \ Mul(pass_1/in_1, pass_2/in_2) \end{array} \right) \setminus \{pass_1, pass_2\} \ .$$

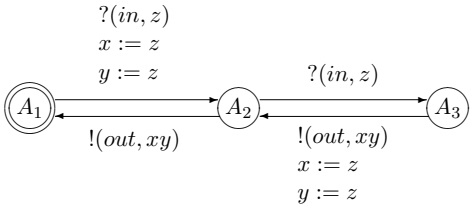
Потоковый граф процесса $Square$ имеет вид



Однако процесс $Square$ не соответствует спецификации $Square\_Spec$. Данный факт нетрудно установить при помощи построения графа процесса $Square$, который, согласно определению операций параллельной композиции, ограничения и переименования, имеет вид



После редукции данного процесса мы получим процесс



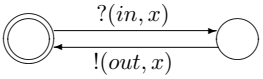
(23.28)

Из процесса (23.28) видно, что процесс $Square$ может последовательно совершить два приема, не выполняя между ними посылки, а процесс $Square\_Spec$ этого сделать не может.

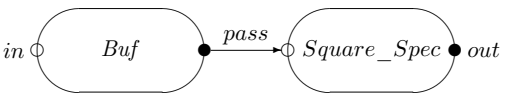
Процесс $Square$ соответствует другой спецификации:

$$Square\_Spec' \stackrel{\text{def}}{=} \left(\begin{array}{l} Buf(pass/out) \ \mid \\ \mid \ Square\_Spec(pass/in) \end{array} \right) \setminus \{pass\},$$

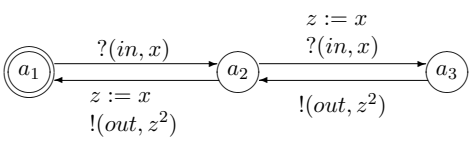
где Buf – буфер, поведение которого изображается диаграммой



Потоковый граф процесса $Square\_Spec'$ имеет вид

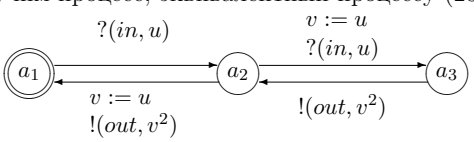


Редуцированный процесс $Square\_Spec'$ имеет вид



(23.29)

Соответствие процесса $Square$ спецификации $Square\_Spec'$ можно понимать как наблюдаемую эквивалентность процессов (23.28) и (23.29). Доказать это можно, например, при помощи теоремы 52. Для того чтобы её можно было применить, переименуем переменные в процессе (23.29) (чтобы множества переменных анализируемых процессов не пересекались). Мы получим процесс, эквивалентный процессу (23.29):



(23.30)

Для доказательства наблюдаемой эквивалентности (23.28) и (23.30) при помощи теоремы 52 определим $M = \{\mu_{A_1a_1}, \mu_{A_2a_2}, \mu_{A_3a_3}\}$, где

$$\mu_{A_1a_1} \stackrel{\text{def}}{=} 1, \quad \mu_{A_2a_2} \stackrel{\text{def}}{=} (x = y = z = u) \quad \mu_{A_3a_3} \stackrel{\text{def}}{=} \{x = y = v, z = u\}.$$

Глава 24

Процессы с асинхронным взаимодействием

24.1 Понятие асинхронного взаимодействия

В рассмотренном выше понятии процессов с передачей сообщений предполагается, что взаимодействие между процессами является **синхронным**, т.е. если процессы P_1 и P_2 взаимодействуют так, что

- P_1 передает сообщение P_2 путем выполнения действия $!(\alpha, e)$, и
- P_2 принимает это сообщение путем выполнения действия $?(\alpha, x)$,

то данные действия выполняются в один и тот же момент времени.

В некоторых ситуациях взаимодействие между вычислительными системами может выполняться также и в **асинхронном** режиме, когда события передачи сообщения и приема этого сообщения выполняются в различные моменты времени и происходят путем передачи сообщений через посредников, называемых **каналами**: отправитель помещает отправляемое сообщение в канал, и получатель принимает это сообщение из канала. Данная ситуация требует развития понятия процесса с передачей сообщений, чтобы обеспечить возможность формального представления асинхронного взаимодействия процессов при помощи каналов.

В разных ситуациях используются разные модели каналов. В одних случаях функционирование канала аналогично функционированию буфера (т.е. сообщения, поступившие в канал, не пропадают, не искажаются и не переупорядочиваются), в других случаях сообщения, поступившие в канал, могут пропадать, искажаться и переупорядочиваться. Также могут накладываться ограничения на максимальное количество

сообщений, которые могут содержаться в канале. Кроме того, условия на функционирование канала могут носить вероятностный характер.

Содержимое канала в каждый момент времени представляет собой список значений. Будем предполагать, что $\mathcal{D}$ содержит особое значение $\boxtimes$, которое будет интерпретироваться как искаженное сообщение.

24.2 Понятие процесса с асинхронным взаимодействием

Будем предполагать, что

- множество типов $Types$ содержит типы $\mathbf{C}$ и $*$, значениями типа $*$ являются списки значений произвольного типа,
- множество Con констант содержит подмножество $Chan$, элементы которого называются **каналами**, и $\forall\, c \in Chan\ \tau(c) = \mathbf{C}$,
- $\forall\, c \in Chan$ множество Var содержит переменную x_c типа $*$.

Обозначим записью $\mathcal{AC}$ совокупность **операторов**, которые определяются так же, как операторы в пункте 23.2.1, со следующей модификацией: к АО, перечисленным в (23.2), добавляются АО вида

$$c?x, \quad c!e, \quad \text{где } c \in Chan,\ x \in Var,\ e \in Tm,\ \tau(x) = \tau(e), \tag{24.1}$$

которые называются **вводом** сообщения x из канала c и **выводом** сообщения e в канал c соответственно.

Понятие **процесса с асинхронным взаимодействием (ПАВ)** определяется аналогично понятию ППС в пункте 23.2, с единственным отличием: метки ребер ПАВ являются операторами из $\mathcal{AC}$. Для каждого ПАВ P $Chan(P)$ обозначает совокупность всех каналов, входящих в P. Для каждого $c \in Chan(P)$ множество $Var(P)$ содержит переменную x_c.

Каждому ПАВ соответствует **система переходов**, которая определяется аналогично соответствующему понятию для ППС в пункте 23.3.1. Система переходов ПАВ P представляет собой граф с множеством вершин $\Sigma_P = Var(P)^\bullet$, данный граф обозначается той же записью Σ_P. Определение понятия перехода в Σ_P получается модификацией соответствующего определения в пункте 23.3.1 путем добавления к условиям (23.4) следующих условий:

$$\begin{array}{l}\text{если } o_i = c?x, \text{ то } x_c^{\theta_i} \neq \varepsilon \text{ и } \theta_{i+1} = (head(x_c^{\theta_i})/x, tail(x_c^{\theta_i})/x_c)\theta_i,\\ \text{если } o_i = c!e, \text{ то } \theta_{i+1} = ((x_c^{\theta_i} e^{\theta_i})/x_c)\theta_i,\end{array} \tag{24.2}$$

т.е. $c?x$ записывает в x первую компоненту содержимого x_c канала c и удаляет её из x_c, а $c!e$ записывает e^{θ_i} в конец содержимого канала c.

Будем предполагать, что система переходов Σ_P наряду с вышеупомянутыми переходами (будем называть их **явными** переходами) может содержать переходы, называемые **неявными** переходами, которые отражают различные преобразования сообщений в каналах. Например, если P – модель системы, в каналах которой возможны искажения или потеря сообщений, то будем предполагать что Σ_P содержит неявные переходы $\theta \xrightarrow{\tau} \theta'$, если $\exists\, c \in Chan(P)$: $\forall\, x \in Var(P) \setminus \{x_c\}\ x^{\theta'} = x^{\theta}$, $x_c^{\theta'}$ получается из x_c^{θ} удалением некоторых компонентов или заменой их на $\boxtimes$.

Ниже будет использоваться соглашение: всякий раз, когда рассматривается какая-либо совокупность ПАВ вида $\{P_i \mid i \in I\}$, предполагается, что множества в семействе $\{Var(P_i) \mid i \in I\}$ дизъюнктны (если это свойство отсутствует, то производится подходящее переименование переменных).

Операции на ПАВ и понятие наблюдаемой эквивалентности для ПАВ определяются аналогично соответствующим понятиям для ППС.

24.3 Протоколы передачи данных

Важными примерами ПАВ являются **протоколы передачи данных** (называемые также просто **протоколами**). В этом параграфе рассматриваются некоторые протоколы передачи данных.

Каждый протокол представляет собой параллельную композицию процессов, выполняющих формирование, отправление, приём и обработку сообщений. Такие процессы называются **агентами** протокола, а сообщения, пересылаемые от одного агента другому, называются **кадрами** (**frame**). Агенты пересылают свои сообщения друг другу через каналы. Кадры в каналах могут искажаться или пропадать (например, в результате воздействия радиопомех). Поэтому каждый кадр должен содержать не только ту информацию, которую один агент желает передать другому, но также и средства, позволяющие получателю этого кадра выяснить, был ли этот кадр искажён в процессе передачи.

24.3.1 Однонаправленный протокол с чередующимися битами

Первый из рассматриваемых нами протоколов передачи данннх состоит из двух агентов: **отправителя** (*Sender*) и **получателя** (*Receiver*). Задачей протокола является организация надёжной доставки кадров от

отправителя к получателю через ненадёжный канал связи (который может искажать и терять передаваемые кадры). Данный протокол называется **однонаправленным протоколом с чередующимися битами**, в англоязычной литературе он называется **one-way Alternating Bit Protocol**, или, сокращённо, **one-way ABP**.

Работа протокола происходит следующим образом.

1. **Отправитель** получает сообщения от своего источника данных, эти сообщения называются **пакетами**. Задача отправителя заключается в периодическом выполнении следующих действий:

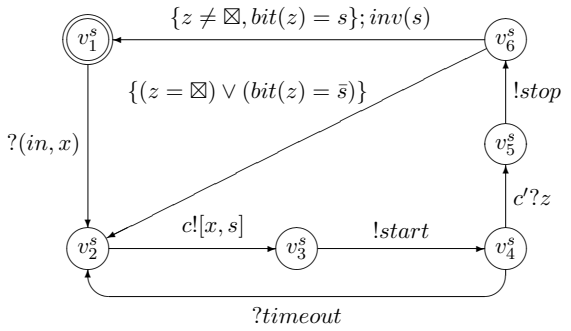
   - получить с порта in очередной пакет от своего источника данных, сформировать из этого пакета кадр, послать этот кадр в канал c и включить таймер действием $!start$,
   - если таймер пришлёт сигнал $timeout$ (который означает, что время ожидания подтверждения посланного кадра закончилось и, видимо, этот кадр до получателя не дошёл), то послать этот кадр в канал c ещё раз,
   - если придёт подтверждение от получателя (через канал c'), то это означает, что текущий кадр дошёл до него успешно, и нужно выключить таймер действием $!stop$, получить следующий пакет от своего источника данных и т.д.

   Механизм, с помощью которого получатель может отличить новый кадр от кадра, переданного повторно, реализован в данном протоколе следующим образом: среди переменных отправителя и получателя присутствуют булевы переменные s и r соответственно, где s – чётность номера очередного кадра, которого пытается послать отправитель, и r – чётность номера очередного кадра, которого ожидает получатель. В начальный момент $s = r = 0$.

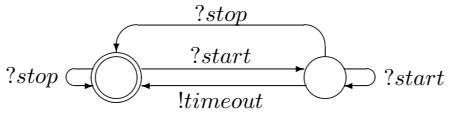
   Будем обозначать кадр, соответствующий пакету x, с которым связан бит s, записью $[x, s]$. Для извлечения пакета и бита из кадра используются функции $info$ и bit соответственно, обладающие свойствами:

   $$info([x, s]) = x, \quad bit([x, s]) = s.$$

   Работа отправителя представляется нижеследующим процессом P, в котором запись $inv(s)$ обозначает присваивание $s := 1 - s$ и запись $\bar{s}$ обозначает терм $1 - s$. Ниже записи вида $inv(x)$ и $\bar{x}$, где x – булева переменная, имеют тот же смысл.



Поведение таймера, с которым взаимодействует P, представляется процессом $Timer$, он имеет вид

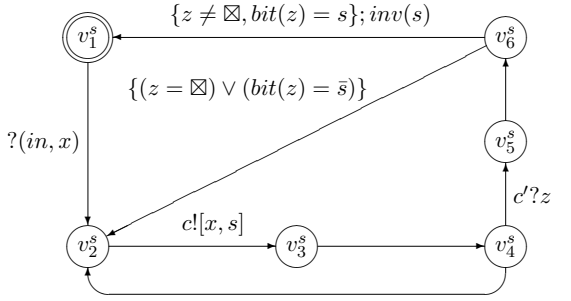


(24.3)

Процесс отправителя $Sender$ имеет вид

$$(P|Timer) \setminus \{start, stop, timeout\}.$$

Нетрудно доказать, что $Sender$ наблюдаемо эквивалентен процессу

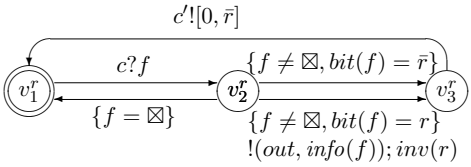


2. **Получатель** периодически выполняет следующие действия:

   - получение из канала c очередного кадра f и проверка наличия искажений в полученном кадре f,
   - если кадр f искажён, то получатель его игнорирует (предполагая, что отправитель, не дождавшись подтверждения, пошлёт это кадр ещё раз),

- если кадр f не искажён, то извлечение из него бита $bit(f)$, и
  - если $bit(f)$ совпадает с ожидаемым битом r, то извлечение из этого кадра пакета $info(f)$ и передача этого пакета с порта *out* потребителю данных и инвертирование бита r,
  - посылка отправителю через канал c' подтверждения полученного кадра (которое имеет вид $[0, \bar{r}]$).

Описанный выше алгоритм представляется процессом *Receiver*:

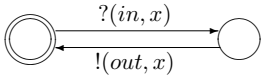


(24.4)

Поведение всего протокола описывается процессом

$$\begin{array}{l} Protocol \stackrel{\text{def}}{=} (Sender \,|\, Receiver), \\ Init(Protocol) = \{s = r = 0, c = c' = \varepsilon\}. \end{array} \tag{24.5}$$

Спецификация процесса (24.5) заключается в том, что данный процесс наблюдаемо эквивалентен следующему процессу *Buffer*:



(24.6)

24.3.2 Двунаправленный протокол передачи сообщений с чередующимися битами

В большинстве ситуаций пересылки данных между двумя агентами требуется **двунаправленная передача**, т.е. передача кадров в обоих направлениях, где каждый из агентов выступает как в роли отправителя, так и в роли получателя. Работа агентов в таких ситуациях выглядит следующим образом: если агент B успешно принял кадр f от агента A, то он посылает подтверждение получения кадра f в составе своего кадра, содержащего пакет для A.

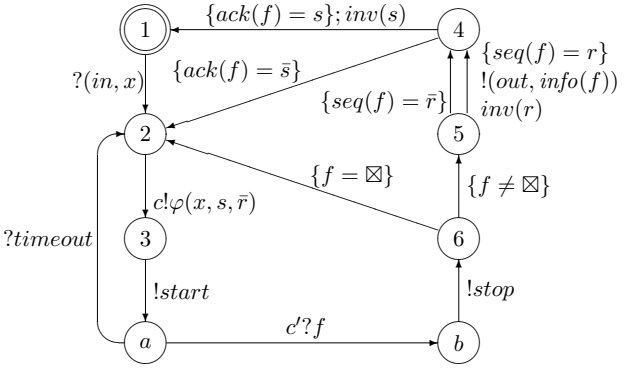
Простейшим протоколом двунаправленной надежной передачи является **двунаправленный протокол с чередующимися битами** (two-way ABP). Поведение каждого из участников этого протокола описывается одним и тем же процессом P, совмещающим в себе функции отправителя и получателя. У каждого из участников имеется свой источник данных (от которого он получает пакеты, пересылаемые в составе кадров

другому участнику) и свой потребитель данных (которому он передает пакеты, получаемые из кадров от другого участника).

Каждый пересылаемый кадр f имеет вид $[x, s, r]$, где x – пакет, s – бит, сопоставленный пакету x, r – бит подтверждения последнего полученного неискажённого кадра. Для извлечения пакетов и битов из кадров используются функции *info*, *seq* и *ack*, обладающие свойствами:

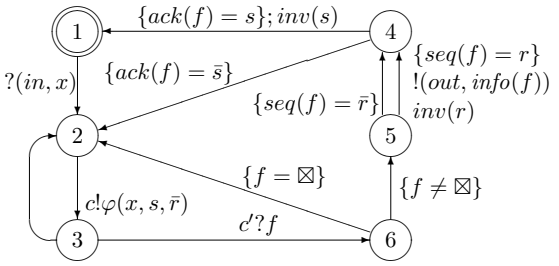
$$info([x, s, r]) = x, \quad seq([x, s, r]) = s, \quad ack([x, s, r]) = r.$$

Алгоритм работы участников данного протокола является объединением алгоритмов работы отправителя и получателя в протоколе из предыдущего пункта и представляется следующим процессом P:



Процесс *Agent*, описывающий поведение агента данного протокола, имеет вид $(P|Timer) \setminus \{start, stop, timeout\}$, где $Timer$ – процесс (24.3).

Нетрудно доказать, что *Agent* наблюдаемо эквивалентен процессу



(24.7)

Поведение всего протокола описывается процессом

$$\begin{array}{l} Protocol \stackrel{\text{def}}{=} \ (Agent_1 \,|\, Agent_2), \\ Init(Protocol) = \{s_1 = 0, r_1 = 0, s_2 = 0, r_2 = 0, c_{12} = \varepsilon, c_{21} = \varepsilon\}, \end{array} \tag{24.8}$$

где $Agent_1$ и $Agent_2$ получаются из (24.7) приписыванием индекса 1 или 2 соответственно к каждому имени и к каждой переменной, не являющейся каналом, заменой в $Agent_1$ каналов c и c' на c_{12} и c_{21} соответственно и заменой в $Agent_2$ каналов c и c' на c_{21} и c_{12} соответственно.

24.3.3 Протокол скользящего окна с возвратом

В рассмотренных выше протоколах каждый следующий кадр посылается только после получения подтверждения об успешной доставке текущего кадра. В этом и следующем пунктах рассматриваются протоколы двунаправленной передачи, в которых отправитель может послать в канал несколько кадров подряд, не дожидаясь подтверждений.

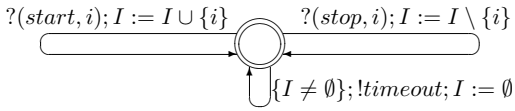
Первый из таких протоколов называется **протоколом скользящего окна (ПСО) с возвратом**. Так же как и в предыдущем протоколе, каждый из участников этого протокола совмещает функции отправителя и получателя. Процесс P, описывающий поведение участника этого протокола, содержит среди своих переменных массив x размера n, в компонентах $x_0, \ldots, x_{n-1}$ которого могут содержаться отправленные, но ещё не подтверждённые пакеты. Совокупность компонентов массива x, в которых содержатся такие пакеты в текущий момент времени, называется **окном**. Мы рассматриваем совокупность $x_0, \ldots, x_{n-1}$ компонентов массива x как «свёрнутую в кольцо», и окно является связным подмножеством этого кольца. С окном связаны две переменные из $Var(P)$: b (нижняя граница окна) и s (верхняя граница окна), их значениями являются элементы $\mathbf{Z}_n = \{0, 1, \ldots, n-1\}$. Если в текущий момент $b = s$, то окно является пустым, иначе окно имеет вид $\{x_b, x_{b+1}, \ldots, x_{s-1}\}$, где все арифметические операции в этом и следующем параграфе выполняются по модулю n. Запись $[b, s[$ обозначает множество $\{b, b+1, \ldots, s-1\}$, т.е. совокупность индексов тех компонентов массива x, которые входят в окно. $Var(P)$ содержит переменную w, значение которой равно размеру окна (т.е. числу компонентов массива x, входящих в окно), $b + w = s$.

Каждый пересылаемый кадр f имеет вид $[x, s, r]$, где x – пакет, $s, r \in \mathbf{Z}_n$, s – номер, сопоставленный пакету x и кадру f, r – номер последнего полученного неискажённого кадра. Для извлечения пакетов и номеров из кадров используются функции *info*, *seq* и *ack*, где

$$info([x, s, r]) = x, \quad seq([x, s, r]) = s, \quad ack([x, s, r]) = r.$$

Каждая компонента i массива x связана с соответствующим таймером, который определяет продолжительность ожидания подтверждения от другого агента получения им пакета x_i. Поведение совокупности этих

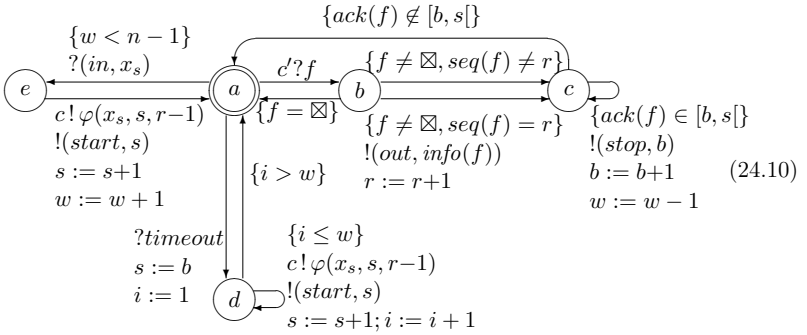
таймеров представляется процессом $Timers$, который имеет вид



(24.9)

$Timers$ имеет переменную i со значениями в $\mathbf{Z}_n$ и переменную I со значениями в $\mathbf{2}^{\mathbf{Z}_n}$, значением I в текущий момент является множество номеров включенных таймеров в этот момент. $Init(Timers) = \{I = \emptyset\}$.

Поведение каждого из участников этого протокола описывается следующим процессом P:



(24.10)

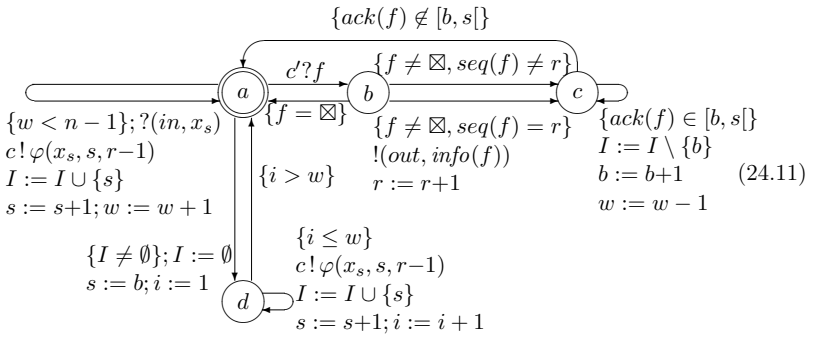
Поясним смысл некоторых компонентов процесса P.

- Значение переменной r равно номеру ожидаемого кадра.
- Переход $a \to e$ означает получение нового пакета от источника данных, который записывается в x_s, s считается номером этого пакета.
- Переход $c \to c$ означает удаление подтверждённого пакета из окна. Если агент получает кадр, третья компонента r которого (т.е. номер подтверждения) такова, что $r \in [b, s[$, то все пакеты с номерами из $[b, r[$ тоже считаются подтверждёнными и удаляются из окна (даже если их подтверждения не дошли).
- Переход $d \to d$ означает повторную передачу всех пакетов из окна, это происходит в том случае, когда заканчивается лимит времени ожидания подтверждения какого-либо пакета из окна.

Процесс $Agent$, описывающий поведение агента данного протокола, имеет вид

$$(P|Timers) \setminus \{start, stop, timeout\}.$$

Нетрудно доказать, что *Agent* наблюдаемо эквивалентен процессу



(24.11)

Поведение всего протокола описывается процессом

$$\begin{aligned} &Protocol \stackrel{\text{def}}{=} \ (Agent_1 \,|\, Agent_2),\\ &Init(Protocol) = \{w_i = b_i = s_i = r_i = 0, I_i = \emptyset \ (i = 1, 2),\\ &\qquad\qquad c_{12} = c_{21} = \varepsilon\}, \end{aligned} \tag{24.12}$$

где процессы $Agent_1$ и $Agent_2$ определяются аналогично соответствующим процессам из предыдущего пункта.

24.3.4 Протокол скользящего окна с заданным повтором

В этом пункте рассматривается другой протокол двунаправленной передачи, в котором отправитель тоже может послать в канал несколько кадров подряд, не дожидаясь подтверждений. Этот протокол называется **протоколом скользящего окна (ПСО) с заданным повтором**. Так же, как и в предыдущем протоколе, каждый из участников этого протокола совмещает функции отправителя и получателя, и номера пересылаемых кадров являются элементами $\mathbf{Z}_n$, где n – четное число. Однако, в отличие от предыдущего протокола, у агента этого протокола имеется не одно, а два окна, называемых **посылающим** и **принимающим** окнами. В посылающем окне содержатся отправленные, но еще не подтвержденные пакеты. Принимающее окно предназначено для размещения пакетов, полученных от другого агента, которые пока не м.б. переданы потребителю данных, т.к. ещё не пришли некоторые пакеты с меньшими номерами. Главное отличие от предыдущего протокола заключается в том, что в случае обнаружения искажения или потери

ожидаемого кадра получатель посылает запрос отправителю на повторную передачу этого кадра, данный запрос обозначается **NAK** (**Negative Acknowledgement**), в то время как в предыдущем протоколе в данном случае происходит повторная посылка всех кадров из окна отправителя.

Среди переменных агента протокола имеются массивы x, $timers$, y, $arrived$ размера $m = n/2$, индексированные элементами $\mathbf{Z}_m$. Данные массивы имеют следующий смысл.

- В x размещается посылающее окно. Если агент посылает в канал кадр номер s, то соответствующий этому кадру пакет записывается в компоненту массива x с номером $s\%m$, где $\%m$ обозначает операцию взятия остатка по модулю m. Как и в предыдущем протоколе, мы рассматриваем совокупность $x_0, \ldots, x_{m-1}$ компонентов массива x как «свёрнутую в кольцо», и посылающее окно является связным подмножеством этого кольца. С этим окном связаны две переменные агента протокола: b (нижняя граница окна) и s (верхняя граница окна), их значениями являются элементы $\mathbf{Z}_n$. Если в текущий момент $b = s$, то посылающее окно является пустым, иначе оно имеет вид $\{x_{b\%m}, x_{(b+1)\%m}, \ldots, x_{(s-1)\%m}\}$. Агент протокола содержит переменную w, значение которой равно размеру окна (т.е. числу компонентов массива x, входящих в окно), $b + w = s$.

- В $timers$ размещаются булевы значения, соответствующие таймерам: если для некоторого $i \in \mathbf{Z}_m$ $timers_i = 1$, то таймер номер i включён, иначе он выключен. Каждый из этих таймеров предназначен для оповещения агента о том, что время ожидания подтверждения пакета с соответствующим номером из посылающего окна закончилось и необходимо послать кадр с этим пакетом ещё раз.

- В y размещается принимающее окно, его размер всегда равен m. В это окно помещаются пакеты, извлеченные из принятых кадров. Пакет, извлеченный из кадра номер i, размещается в компоненте $y_{i\%m}$. С этим окном связаны переменные r (нижняя граница окна) и u (верхняя граница окна), их значения – элементы $\mathbf{Z}_n$, $u = r + m$.

- В $arrived$ размещаются булевы значения, связанные с соответствующими компонентами принимающего окна и имеющие следующий смысл: $\forall\, i = 0, \ldots, m-1$ $arrived_i = 1$, если в i-й компоненте принимающего окна содержится пакет, который агент пока ещё не передал своему потребителю данных.

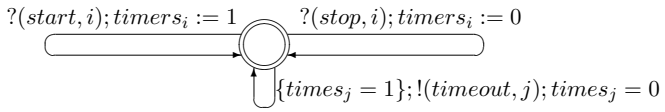
  Если в принимающем окне присутствует пакет с номером r, то

агент может передать этот пакет своему потребителю данных, после чего он увеличивает r и u на 1.

Каждый пересылаемый кадр f имеет вид $[k, x, s, r]$, где k – тип кадра, он может иметь следующие значения: *data* (информационный кадр), *ack* (кадр, содержащий лишь подтверждение), *nak* (кадр с запросом на повторную передачу), x – пакет, $s, r \in \mathbf{Z}_n$, s – номер, сопоставленный пакету x и кадру f, r – номер последнего полученного неискажённого кадра. У кадров типа *ack* и *nak* компоненты x и s фиктивны и обозначаются символом $*$. Для извлечения пакетов и номеров из кадров используются функции *kind*, *info*, *seq* и *ack*, где

$$kind([k,x,s,r]) = k, \quad info([k,x,s,r]) = x,$$
$$seq([k,x,s,r]) = s, \quad ack([k,x,s,r]) = r.$$

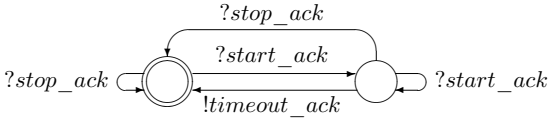
Поведение таймеров представляется процессом $Timers$, который использует переменную i со значениями в $\mathbf{Z}_m$ и вышеупомянутый массив $timers$, представляющий множество номеров включенных таймеров в текущий момент. Процесс $Timers$ имеет вид



(24.13)

Средняя стрелка в (24.13) обозначает m переходов, соответствующих значениям $j = 0, \ldots, m-1$. $Init(Timers) = \{timers = (0, \ldots, 0)\}$.

Наряду с вышеупомянутыми таймерами агент протокола использует дополнительный таймер, поведение которого описывается процессом Add_Timer, имеющим следующий вид:



Дополнительный таймер используется в том случае, когда агенту необходимо послать подтверждение a: он включает дополнительный таймер ($!start\_ack$), и если до сигнала $timeout\_ack$ агент получит новый пакет x от своего источника данных, он посылает кадр $[data, x, s, a]$ (т.е. посылает это подтверждение a вместе с пакетом x) и выключает дополнительный таймер, иначе он посылает кадр $[ack, *, *, a]$.

Поведение каждого из участников этого протокола описывается нижеследующим процессом P, который мы для удобства восприятия представляем двумя графами. В этих графах используются следующие обозначения и соглашения:

- вершины с метками a_0 и b_0 в первом графе совпадают с вершинами с соответствующими метками во втором графе,

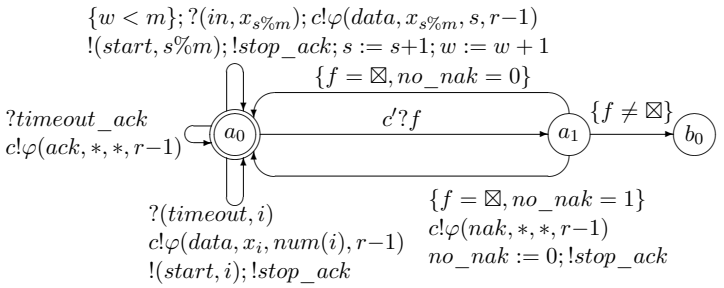
- если какое-либо ребро $v \to v'$ помечено последовательностью АО A, в которой имеется более одного АО приема или посылки, то данное ребро является сокращенным обозначением пути $v \stackrel{A_1}{\to} \dots \stackrel{A_n}{\to} v'$, где A является конкатенацией $A_1 \dots A_n$,

- $num(i)$ обозначает число $j \in [b, s[$, такое, что $j\%m = i$,

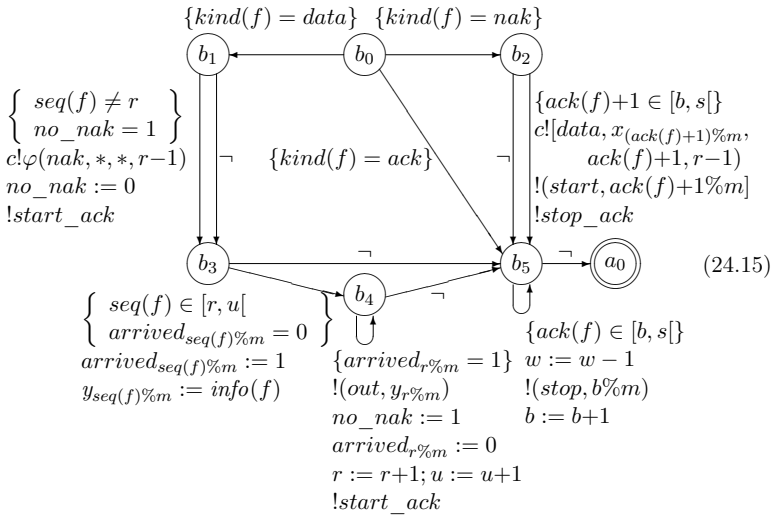
- символ $\neg$ на ребре *edge* какого-либо из данных графов имеет следующий смысл: пусть v – начало данного ребра, тогда существует другое ребро с началом в v, первый АО в его метке имеет вид $\{\varphi\}$, и метка ребра *edge* имеет вид $\{\neg\varphi\}$,

- P имеет булеву переменную no_nak, предназначенную для предотвращения нескольких запросов на повторную передачу ожидаемого кадра (его номер равен значению переменной r): $no\_nak = 1$, если запрос на повторную передачу кадра номер r ещё не был послан, при получении искаженного кадра $\boxtimes$ или кадра с номером $\neq r$ агент посылает запрос NAK на повторную передачу кадра номер r.

Графы, представляющие процесс P, имеют следующий вид (во втором графе представлены различные варианты обработки полученного неискаженного кадра):



(24.14)



Поясним смысл некоторых переходов процесса P.

- Верхний переход $a_0 \to a_0$: получение нового пакета от источника данных, который записывается в $x_{s\%m}$, s считается номером этого пакета и кадра, содержащего этот пакет.

- Переход $b_3 \to b_4$: запись принятого пакета в принимающее окно.

- Переход $b_4 \to b_4$: передача потребителю данных тех принятых пакетов, которые можно передать, и их удаление из принимающего окна.

- Переход $b_5 \to b_5$: удаление подтвержденных пакетов из посылающего окна. Если агент получает кадр, четвертая компонента a которого (номер подтверждения) такова, что $a \in [b, s[$, то все пакеты с номерами из $[b, a[$ тоже считаются подтверждёнными и удаляются из посылающего окна (даже если их подтверждения не дошли).

Процесс $Agent$, описывающий поведение агента, имеет вид

$$(P|Timers|Add\_Timer) \setminus \left\{ \begin{array}{l} start, stop, timeout \\ start\_ack, stop\_ack, timeout\_ack \end{array} \right\}$$

Поведение всего протокола описывается процессом

$$\begin{aligned} &Protocol \stackrel{\text{def}}{=} (Agent_1 \,|\, Agent_2),\\ &Init(Protocol) = \{w_i = b_i = s_i = r_i = 0, u_i = m = n/2, no\_nak_i = 1,\\ &\qquad arrived_i = (0 \ldots 0), timers_i = (0, \ldots, 0)\ (i = 1, 2),\\ &\qquad c_{12} = c_{21} = \varepsilon\}, \end{aligned} \tag{24.16}$$

где процессы $Agent_1$ и $Agent_2$ определяются аналогично соответствующим процессам из предыдущего пункта.

Глава 25

Задачи и исследовательские проблемы

1. Доказать свойства операций на процессах из пункта 20.7.
2. Доказать свойства простой эквивалентности из пункта 21.1.3.
3. Доказать теоремы 45. и 46.
4. Для процесса Sch из пункта 22.2 доказать второе свойство в (22.7), а также детерминированность Sch, отсутствие в Sch терминальных состояний и свойство 3 в списке свойств процесса Sch.
5. Для процесса 23.21 из пункта 23.4.2
   - найти необходимые и достаточные условия, которым должны удовлетворять L и R, чтобы процесс завершал свою работу с этими L и R нормально (т.е. в вершине Cc), и
   - разработать такой процесс разделения мультимножеств, который бы работал корректно на любых L и R.
6. Доказать, что процесс 24.3.1, моделирующий протокол однонаправленной передачи данных в пункте 24.3.1, наблюдаемо эквивалентен процессу 24.6.
7. Найти необходимые и достаточные условия, при которых процесс P', получаемый применением к процессу P операций редукции из пункта 23.2.4, удовлетворял бы условию $P' \approx P$.
8. Найти нередуцируемые процессы, наблюдаемо эквивалентные процессам, описывающим двунаправленные протоколы передачи данных из пунктов 24.3.2, 24.3.3 и 24.3.4.

9. Обобщить теорему 52 путем нахождения необходимых и достаточных условий наблюдаемой эквивалентности процессов с передачей сообщений.

10. Найти необходимые и достаточные условия наблюдаемой эквивалентности процессов с асинхронным взаимодействием.

11. Обобщить понятие процесса с асинхронным взаимодействием, допуская, что имена каналов м.б. переменными и их значения могут передаваться в сообщениях. Определить понятие наблюдаемой эквивалентности для таких процессов и найти необходимые и достаточные условия наблюдаемой эквивалентности таких процессов.

Литература

[1] Лекции лауреатов премии Тьюринга. М.: Мир, 1993.

[2] Страничка Р. Флойда в Википедии:
`https://ru.wikipedia.org/wiki/Флойд,_Роберт`

[3] *Floyd R. W.* Assigning meanings to programs. In J. T. Schwartz, editor, Proc. Symp. Appl. Math., volume 19 of Mathematical Aspects of Computer Science, pages 19–32, Providence, R.I., 1967.

[4] *Hoare C. A. R.* An axiomatic basis for computer programming. Communications of the ACM, 12(10):576580, October 1969.

[5] *Андерсон Р.* Доказательство правильности программ. М.: Мир, 1982.

[6] *Абрамов С. А.* Элементы анализа программ. Частичные функции на множестве состояний. М.: Наука, 1986.

[7] *Непомнящий В. А., Рякин О. М.* Прикладные методы верификации программ. М.: Радио и связь, 1988.

[8] *Камкин А. С.* Введение в формальные методы верификации программ: учебное пособие. М: МАКС Пресс, 2018.

[9] *Francez N.* Verification of programs. Addison-Wesley Publishers Ltd., 1992.

[10] *Loeckx J., and Sieber K.* The Foundations of Program Verication. Wiley, Teubner, Stuttgart, 1984.

[11] *Manna Z.* Mathematical theory of computation. McGraw-Hill, 1974.

[12] *Bjorner N., Browne A., Manna Z.* Automatic generation of invariants and intermediate assertions. Theoretical Computer Science, Volume 173, Issue 1, 1997, pp. 49-87.

[13] *Bensalem S., Lakhnech Y., Saidi H.* Powerful techniques for the automatic generation of invariants. Proc. 8th Internat. Conf. on Computer Aided Verification, Lecture Notes in Computer Science, Vol. 1102, Springer, Berlin (1996), pp. 323–335.

[14] *Chadha R., Plaisted D. A.* On the mechanical derivation of loop invariants. J. Symbol. Comput., 15 (5) (1993), pp. 705–744.

[15] *Cousot P., Cousot R.* Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fix points. 4th ACM Symp. Principles of Programming Languages, ACM Press, New York (1977), pp. 238–252.

[16] *Cousot P., Halbwachs N.* Automatic discovery of linear restraints among the variables of a program. 5th ACM Symp. Principles of Programming Languages (1978), pp. 84–97.

[17] *Dill D., Wong-Toi H.* Verification of real-time systems by successive over and under approximation. Proc. 7th Internat. Conf. on Computer Aided Verification. Lecture Notes in Computer Science, Springer, Berlin (1995), pp. 409–422.

[18] *German S. M., Wegbreit B.* A synthesizer of inductive assertions. IEEE Trans. Software Eng., 1 (1975), pp. 68–75.

[19] *Halbwachs N., Raymond P., Proy Y.-E.* Verification of linear hybrid systems by means of convex approximations. 1st Internat. Static Analysis Symp., Lecture Notes in Computer Science, Vol. 864, Springer, Berlin (1994), pp. 223–237.

[20] *Bensalem S., Bozga M., Fernandez J.-C., Ghirvu L., Lakhnech Y.* A transformational approach for generating non-linear invariants. In SAS, 2000, pp. 58–74.

[21] *Sankaranarayanan S., Sipma H. B., Manna Z.* Non-linear loop invariant generation using grobner bases. 2004.

[22] *Tiwari A., Rueb H., Saidi H., Shankar N.* A technique for invariant generation. In TACAS 2001 – Tools and Algorithms for the Construction and Analysis of Systems, ser. LNCS, vol. 2031. Genova, Italy: Springer-Verlag, Apr. 2001, pp. 113–127.

[23] *Ernst M. D., Perkins J. H., Guo P. J., McCamant S., Pacheco C., Tschantz M. S., Xiao C.* The Daikon system for dynamic detection of

likely invariants. Science of Computer Programming, vol. 69, no. 1–3, pp. 35–45, Dec. 2007.

[24] *Gupta A., Rybalchenko A.* Invgen: An efficient invariant generator. In Computer Aided Verification. Springer, 2009, pp. 634–640.

[25] `https://ru.wikipedia.org/wiki/Алгоритм_Тарьяна`

[26] *Paulson L. C.* ML for the working programmers, 2nd edition. Cambridge University Press, 2000.

[27] *Akl S. G.* The Design and Analysis of Parallel Algorithms. Prentice-Hall, 1989.

[28] *Тель Ж.* Введение в распределенные алгоритмы. М.: МЦНМО, 2009.

[29] *Siegel S. F. and Gopalakrishnan G.* Formal Analysis of Message Passing. In R. Jhala and D. Schmidt, editors, Verification, Model Checking, and Abstract Interpretation: 12th International Conference, VMCAI 2011, volume 6538 of Lecture Notes in Computer Science, pages 2–18, (2011).

[30] *Карпов Ю. Г.* Model Checking. Верификация параллельных и распределенных программных систем. СПб.: БХВ-Петербург, 2010.

[31] *Clarke E., Grumberg O., Peled D.* Model checking. The MIT Press, 2001.

[32] *Baier C. and Katoen J.-P.* Principles of Model Checking The MIT Press, 2008.

[33] *Clarke E.M. et al.* Handbook of Model Checking. Springer International Publishing AG, 2018.

[34] `http://spinroot.com/spin/whatispin.html`

[35] *Шошмина И. В., Карпов Ю. Г.* Введение в язык Promela и систему комплексной верификации Spin: учеб. пособие. СПб: Изд-во Политехн. ун-та, 2010.

[36] *Кемени Дж., Снелл Дж.* Конечные цепи Маркова. М.: Наука, 1970.

[37] *Бухараев Р. Г.* Основы теории вероятностных автоматов. М.: Наука, 1985.

[38] *Hart S., Sharir M., Pnueli A.* Termination of probabilistic concurrent programs. ACM Transactions on Programming Languages and Systems, 5(3):356–380, 1983.

[39] *Vardi M.* Automatic verification of probabilistic concurrent finite state programs. In Proc. 26th Annual Symposium on Foundations of Computer Science (FOCS'85), pages 327–338. IEEE Computer Society Press, 1985.

[40] *Courcoubetis C. and Yannakakis M.* Verifying temporal properties of finite state probabilistic programs. In Proc. 29th Annual Symposium on Foundations of Computer Science (FOCS'88), pages 338–345. IEEE Computer Society Press, 1988.

[41] *Kwiatkowska M., Norman G., and Parker D.* Probabilistic model checking in practice: Case studies with PRISM. ACM SIGMETRICS Performance Evaluation Review, 32(4):16–21, 2005.

[42] `http://www.prismmodelchecker.org/`

[43] *Kwiatkowska M., Norman G., and Parker D.* PRISM 4.0: Verification of probabilistic real-time systems. In Gopalakrishnan G. and Qadeer S., editors, Proc. 23rd International Conference on Computer Aided Verification (CAV'11), volume 6806 of LNCS, pages 585–591. Springer, 2011.

[44] `http://qav.comlab.ox.ac.uk/`

[45] *Hansson H. and Jonsson B.* A logic for reasoning about time and reliability. Formal Aspects of Computing, 6(5):512–535, 1994.

[46] *Milner R.* A Calculus of Communicating Systems. Number 92 in Lecture Notes in Computer Science. Springer Verlag, 1980.

[47] *Milner R.* Calculi for synchrony and asynchrony. Theoretical Computer Science, 25:267–310, 1983.

[48] *Milner R.* Communication and Concurrency. Prentice Hall, 1989.

[49] *Milner R.* Communicating and Mobile Systems: the π-Calculus. Cambridge University Press, ISBN 052164320, 1999.

Миронов Андрей Михайлович

МЕТОДЫ ВЕРИФИКАЦИИ ПРОГРАММ

Главный редактор *Мовчан Д. А.*
dmkpress@gmail.com

Корректор *Синяева Г. И.*
Верстка *Миронов А. М.*
Дизайн обложки *Мовчан А. Г.*

Формат 70×100 1/16.
Гарнитура «Петербург». Печать офсетная.
Усл. печ. л. 31,5. Тираж 100 экз.

Сайт издательства: www.дмк.рф