Методы верификации программ

Версия от 17 ноября 2000 года

Пособие подготовил Жуков Д.Ю.
E-mail: `dmjr@ispras.ru`
`http://www.chat.ru/~dmjr`

Содержание

1 Последовательные программы

1.1 Введение

В этом учебном пособии описаны методы доказательства правильности компьютерных программ, написанных на структурных языках программирования без использования подпрограмм и параллелизма. Мы рассмотрим последовательные программы без использования массивов в разделе 1.2 и программы с массивами в разделе 1.3. Текст пособия в значительной мере базируется на работах [Man74] и [Fra92].

Данная версия текста является предварительной и может содержать ошибки. О них можно сообщать по адресу `dmjr@ispras.ru`. Замечания и предложения читателей могут быть очень полезны автору в его работе над текстом.

1.1.1 Используемые обозначения

При записи логических формул в пособии будут использоваться следующие обозначения. Символ "$\rightarrow$" обозначает логическую импликацию. Если $f_1, \ldots, f_n$ – формулы, то запись

$$\left\{ \begin{array}{c} f_1 \\ \ldots \\ f_n \end{array} \right\}$$

обозначает конъюнкцию $f_1 \wedge \ldots \wedge f_n$, а запись

$$\left[\begin{array}{c} f_1 \\ \ldots \\ f_n \end{array} \right]$$

– дизъюнкцию $f_1 \vee \ldots \vee f_n$. Если предикат p определяется при помощи формулы f (см. раздел 1.2.4), то этот факт записывается как $p \ : \ f$.

1.2 Блок-схемы

1.2.1 Определение блок-схем

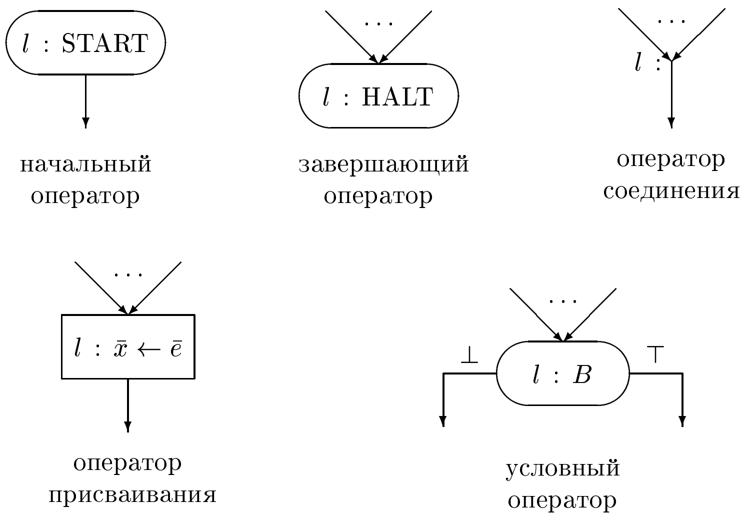
Сперва мы рассмотрим формальную модель очень ограниченного класса программ. Этот класс соответствует программам на структурных языках программирования (типа Pascal, C, Modula-2) без использования подпрограмм. Каждая такая программа оперирует с конечным набором переменных. Мы не будем использовать в программах операторы ввода-вывода. Вместо этого мы будем предполагать, что, когда программа начинает свою работу, ее входные данные уже содержатся в некоторых из ее переменных. Эти переменные мы назовем *входными переменными.* В процессе своей работы программа до своего завершения должна записать результат вычисления в определенные переменные, которые мы будем называть *выходными.* Мы будем допускать программы, в которых одна и та же переменная является и входной, и выходной. Программы могут изменять значения входных переменных в процессе своей работы а также использовать выходные переменные для хранения промежуточных результатов вычислений.

Каждая переменная v может принимать значения из некоторого множества D_v, которое мы будем называть ее *доменом.* Мы будем также считать, что задано некоторое множество *значений данных* D, включающее в себя все домены (это значит, что для любой переменной v выполнено соотношение $D_v \subseteq D$).

В программах мы будем выделять две категории выражений: выражения по данным и логические выражения. Каждое *выражение по данным* принимает значения из множества D. Каждое *логическое выражение* принимает либо значение $\top$ (истина), либо значение $\perp$ (ложь). Если константы $\top$ и $\perp$ принадлежат множеству D, то одно и то же выражение может быть одновременно и выражением по данным, и логическим выражением.

В качестве первой модели для представления программ мы рассмотрим *блок-схемы.* Блок-схема представляет из себя граф, в вершинах которого находятся операторы, а ребра описывают передачу управления между операторами.

Определение 1 *Оператор* блок-схемы представляет из себя последовательность символов вида $l : op$, где l – *метка* оператора, а op – его *тело.* Каждый оператор относится к одному из пяти типов:



Тип оператора	Число входящих ребер	Число выходящих ребер
Начальный	0	1
Присваивания	≥ 1	1
Условный	≥ 1	2
Завершающий	≥ 1	0
Соединения	≥ 1	1

Рис. 1: Операторы блок-схемы

1. *Начальный* оператор l : START.
2. Оператор *присваивания* вида $l : (x_1, \ldots, x_m) \leftarrow (e_1, \ldots, e_m)$. Здесь $x_1, \ldots, x_m$ являются попарно различными переменными. Для каждого i, $1 \leq i \leq m$, e_i – это выражение, принимающее значения из домена D_{x_i}.
3. *Условный* оператор $l : B$. Здесь B является логическим выражением.
4. Оператор *завершения* l : HALT.
5. Оператор *соединения* l : JOIN. □

Определение 2 *Программой в виде блок-схемы* (или просто *блок-схемой*) мы будем называть ориентированный граф, вершинами которого являются операторы. Графическое изображение различных операторов показано на рисунке 1. На граф накладываются следующие ограничения:

1. Для каждого оператора число ребер, входящих в него или выходящих из него, зависит от типа оператора. Эти ограничения на число ребер изображены символически и описаны при помощи таблицы на рисунке 1. Из каждого условного оператора выходит два ребра, одно из которых помечено символом $\top$, а другое – символом $\perp$. При изображении операторов соединения мы будем опускать слово JOIN.
2. В графе имеется ровно один начальный оператор.
3. Любой оператор находится на ориентированном пути из начального оператора в какой-либо из завершающих операторов.
4. Различные операторы имеют различные метки. Множество всех меток, встречающихся в программе P мы будем обозначать через L_P. Меткой $l \in L_P$ мы будем обозначать оператор, помеченный этой меткой там, где это не будет приводить к неоднозначности. Иногда при записи оператора мы будем опускать его метку.

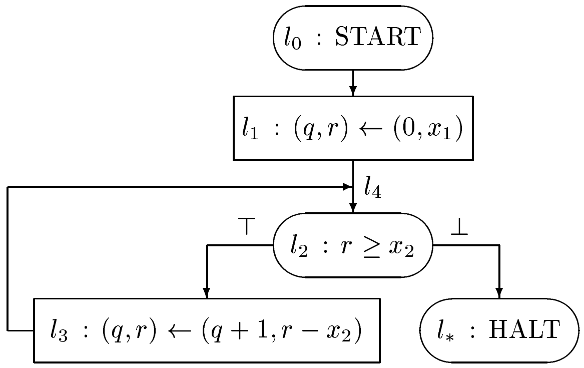


Рис. 2: Программа деления P_{idiv}

Для начального оператора, оператора присваивания или оператора соединения с меткой l выражение $\mathrm{suc}(l)$ обозначает следующий за ним оператор. Если l – условный оператор, то $\mathrm{suc}^{\top}(l)$ обозначает оператор, в который ведет ребро с меткой $\top$ из l. Аналогично, оператор в конце ребра с меткой $\perp$ мы обозначим через $\mathrm{suc}^{\perp}(l)$. □

Соглашение: Мы будем считать, что начальный оператор помечен меткой l_0. Если завершающий оператор в программе всего один, то мы будем считать, что он помечен меткой l_*. □

В качестве примера блок-схемы мы рассмотрим программу P_{idiv} на рисунке 2. Эта программа реализует алгоритм целочисленного деления x_1 на x_2. Доменом каждой переменной в ней является множество целых чисел $\mathbf{Z}$. На выходе алгоритма получается частное q и остаток r.

Оператор $(q,r) \leftarrow (0,x_1)$ присваивает пременной q значение 0, а переменной r – значение x_1. Аналогично, оператор $(q,r) \leftarrow (q+1, r-x_2)$ присваивает переменной q значение $q+1$, а переменной r – значение $r-x_2$. Отметим, что при выполнении оператора присваивания общего вида $(x_1,\ldots,x_m) \leftarrow (e_1,\ldots e_m)$ все переменные x_i, $1 \leq i \leq m$, заменяются на значения выражений e_i *одновременно*, то есть значения всех выражений e_i вычисляются до того, как изменяется значение какой-либо из переменных x_i. Из условного оператора $l_2 : r \geq x_2$ вычисление идет по ребру, помеченному символом $\top$, если при достижении этого оператора $r \geq x_2$, и по ребру, помеченному символом $\perp$ в противном случае.

1.2.2 Некоторые определения

В этом подразделе будут введены некоторые определения, которые потребуются нам при работе с последовательными программами. Если $\bar{x} = (x_1,\ldots,x_m)$ – некоторый вектор переменных, то через $D_{\bar{x}}$ мы обозначим множество $D_{\bar{x}} = D_{x_1} \times \cdots \times D_{x_m}$. Его мы будем называть *доменом* вектора $\bar{x}$. Далее, пусть $\bar{v} = (v_1,\ldots,v_n)$ – вектор всех переменных программы. Домен вектора $\bar{v}$ мы обозначим символом $\Sigma \stackrel{def}{=} D_{\bar{v}}$. Множество Σ называется множеством *состояний памяти* программы P. Для произвольного состояния памяти $\sigma \in \Sigma$ и переменной x обозначим через $\sigma[\![x]\!]$ компоненту вектора σ, соответствующую переменной x. Если e – выражение над переменными программы, то $\sigma[\![e]\!]$ обозначает значение выражения e при значениях переменных, определяемых состоянием σ.

Пусть $\bar{x} = (x_1,\ldots,x_m)$ – вектор переменных. Тогда записью $\sigma[\![\bar{x}]\!]$ мы будем обозначать вектор $(\sigma[\![x_1]\!],\ldots,\sigma[\![x_m]\!]) \in D_{\bar{x}}$.

Пусть σ – состояние памяти, v – переменная, а $d \in D_v$ – элемент данных. Запись $\sigma[d \mid v]$ обозначает результат *замены* в σ компоненты, соответствующей v, на d. При этом для любой переменной x

$$\sigma[d \mid v][\![x]\!] \stackrel{def}{=} \begin{cases} d, & \text{если } x \text{ совпадает с } v \\ \sigma[\![x]\!], & \text{иначе} \end{cases}$$

Мы будем использовать также запись $\sigma[\bar{\xi} \mid \bar{x}]$, где $\bar{x} = (x_1,\ldots,x_m)$ – вектор переменных, а $\bar{\xi} = (d_1,\ldots,d_m) \in D_{\bar{x}}$ – вектор значений. При этом

$$\sigma[\bar{\xi} \mid \bar{x}] \stackrel{def}{=} \sigma[d_1 \mid x_1]\cdots[d_m \mid x_m]$$

Иногда мы будем использовать следующее сокращение:

$$\sigma[e \mid v] \stackrel{def}{=} \sigma[\sigma[\![e]\!] \mid v]$$

Здесь v – это переменная, а e – это выражение, принимающее значения из множества D_v. Аналогичное сокращение $\sigma[\bar{e} \mid \bar{v}]$ мы будем использовать для векторов переменных и векторов выражений.

1.2.3 Функционирование программ

Опишем теперь работу произвольной блок-схемы. *Конфигурацией* блок-схемы P мы будем называть пару $C = \langle \sigma, \lambda \rangle$, где σ – состояние памяти, а $\lambda \in L_P$ – метка. Состояние σ определяет значения переменных программы на некотором шаге ее работы, а λ указывает на оператор, который будет выполняться на следующем шаге.

На множестве всех конфигураций программы мы определим бинарное отношение, которое называется *отношением перехода* и обозначается символом "$\rightarrow$". (Запись $C_1 \rightarrow C_2$ читается "C_1 переходит в C_2"). Пусть $C = \langle \sigma, \lambda \rangle$ и $C' = \langle \sigma', \lambda' \rangle$ – две конфигурации. В этом случае $C \rightarrow C'$ тогда и только тогда, когда выполнены следующие условия:

- Если λ – начальный оператор или оператор соединения, то $\sigma' = \sigma$ и $\lambda' = \lambda$.
- Если λ – присваивание $\bar{x} \leftarrow \bar{e}$, то $\lambda' = \mathrm{suc}(\lambda)$ и $\sigma' = \sigma[\bar{e} \mid \bar{x}]$.
- Если λ – условный оператор B, то $\lambda' = \mathrm{suc}^{\sigma[\![B]\!]}(\lambda)$ и $\sigma' = \sigma$.

Конфигурацию $C = \langle \sigma, \lambda \rangle$ мы будем называть *начальной*, если λ – это начальный оператор (то есть $\lambda = l_0$), и *завершающей*, если λ – завершающий оператор.

Определение 3 *Вычислением* программы P называется конечная или бесконечная последовательность конфигураций $\pi = C_0, \ldots, C_n, \ldots$, обладающая следующими свойствами:

1. для любого i такого, что i-я конфигурация вычисления не является в нем последней, выполнено $C_i \rightarrow C_{i+1}$;
2. если вычисление конечно, то его последняя конфигурация является завершающей. □

Определение 4 Бинарное отношение R называется *детерминированным*, если для любого a существует не более одного b, такого что $(a, b) \in R$. Иными словами, из $(a, b_1) \in R$ и $(a, b_2) \in R$ следует $b_1 = b_2$. □

Нетрудно показать, что отношение перехода на конфигурациях блок-схем является детерминированным. Пользуясь этим фактом, можно доказать, что для любой конфигурации существует единственное вычисление, начинающееся с нее. Для любого вычисления состояние памяти в его начальной конфигурации мы будем называть *начальным* состоянием памяти в данном вычислении, а состояние памяти в завершающей конфигурации – *завершающим* состоянием памяти.

Пусть σ – состояние памяти программы P, и пусть C_0 обозначает ее начальную конфигурацию с состоянием памяти σ (то есть $C_0 = \langle \sigma, l_0 \rangle$). Вычисление программы P, начинающееся в C_0, мы обозначим через $\pi(P, \sigma)$. Пусть π – некоторое вычисление. Через $val(\pi)$ мы будем обозначать состояние памяти в последней конфигурации π, если π конечно. Если π бесконечно, то мы положим $val(\pi) \stackrel{def}{=} \omega$, где ω – специальный символ, обозначающий *неопределенное* состояние. *Функцию значения* $\mathbf{M}$ мы определим следующим образом:

$$\mathbf{M}[\![P]\!](\sigma) \stackrel{def}{=} val(\pi(P, \sigma))$$

Мы будем говорить, что программа P *завершается* при начальном состоянии σ_0, если вычисление $\pi(P, \sigma_0)$ конечно (что эквивалентно $\mathbf{M}[\![P]\!](\sigma_0) \neq \omega$). В этом случае функция $\mathbf{M}[\![P]\!]$ определяет соотношение между значениями переменных в начальном состоянии памяти σ_0 и их значениями в завершающем состоянии вычисления.

1.2.4 Спецификации

Спецификацией программы мы будем называть набор требований к ней. Пусть P – программа на некотором языке программирования, а Φ – спецификация. Задача *верификации* программы P относительно спецификации Φ состоит в доказательстве того, что P соответствует всем требованиям, содержащимся в Φ.

В этой главе мы рассмотрим некоторые методы задания спецификаций и методы верификации последовательных программ, в которых отсутствует рекурсия.

Введем несколько дополнительных определений. Пусть M – некоторое множество. *Предикатом* на множестве M мы будем называть функцию $p : M \to \{\top, \bot\}$. Предикаты мы будем задавать при помощи формул логики первого порядка. Мы будем предполагать, что язык логики таков, что любое логическое выражение в программе является формулой без кванторов, а любое выражение в операторах присваивания является термом. Если $\sigma \in \Sigma$ – состояние памяти, а $p : \Sigma \to \{\top, \bot\}$ – предикат, то запись $\sigma \models p$ означает истинность предиката p в состоянии σ (что эквивалентно $p(\sigma) = \top$).

Одно из требований к программам, которое мы будем рассматривать, состоит в том, что после завершения программы ее переменные должны находится в заданном соотношении. Это соотношение можно описать при помощи предиката на множестве состояний Σ. Он определяет, какое состояние памяти должно получиться после завершения работы программы. Такой предикат мы будем называть *выходным предикатом*.

Иногда мы будем предполагать, что данные, которые могут поступать на вход программы, должны удовлетворять некоторому условию. Это условие мы будем также задавать при помощи предиката на Σ. Он будет определять некоторое соотношение между переменными в начальном состоянии программы. Такой предикат называется *входным предикатом*. Можно сказать, что мы заинтересованы в анализе поведения программы только для тех начальных состояний, которые удовлетворяют входному предикату.

Определение 5 Пусть даны

- программа P,
- входной предикат $\phi : \Sigma \to \{\top, \bot\}$.
- выходной предикат $\psi : \Sigma \to \{\top, \bot\}$.

Мы будем говорить, что

- Программа P *частично корректна* относительно ϕ и ψ, если для любого состояния памяти σ, такого что $\sigma \models p$ и $\mathbf{M}[\![P]\!](\sigma) \neq \omega$ выполнено $\mathbf{M}[\![P]\!](\sigma) \models \psi$. Частичную корректность программы P относительно ϕ и ψ мы будем обозначать записью $\{\phi\}P\{\psi\}$.
- Программа P *полностью корректна* относительно ϕ и ψ, если для любого состояния памяти σ, такого что $\sigma \models p$ выполнено $\mathbf{M}[\![P]\!](\sigma) \neq \omega$ и $\mathbf{M}[\![P]\!](\sigma) \models \psi$. Полную корректность программы P относительно ϕ и ψ мы будем обозначать записью $\langle\phi\rangle P\langle\psi\rangle$. □

В качестве *спецификаций* последовательной программы P мы будем рассматривать пары $\Phi = (\phi, \psi)$, где ϕ – входной предикат, а ψ – выходной предикат. Мы будем говорить о частичной или полной корректности программы относительно такой спецификации.

Отметим, что частичная корректность программы P относительно входного предиката ϕ и выходного предиката $\top$ эквивалентна тому, что программа P завершается всегда, когда начальное состояние памяти удовлетворяет ϕ. В этом случае мы будем говорить, что P *завершается* на ϕ.

Лемма 1 *Пусть дана программа P и спецификация $\Phi = (\phi, \psi)$. В этом случае $\langle\phi\rangle P\langle\psi\rangle$ тогда и только тогда, когда $\{\phi\}P\{\psi\}$ и $\langle\phi\rangle P\langle\top\rangle$.* □

Таким образом, доказательство полной корректности программы можно свести к доказательству ее частичной корректности и к доказательству ее завершаемости. Из определения 5 следует, что как частичная, так и полная корректность программ сохраняется

- при замене входного предиката на более сильный, а также
- при замене выходного предиката на более слабый.

Таким образом, верна следующая лемма:

Лемма 2 *Пусть дана программа P. Пусть предикаты ϕ, ϕ', ψ и ψ' таковы, что формулы $\phi' \to \phi$ и $\psi \to \psi'$ истинны. Тогда*

- *Из $\{\phi\}P\{\psi\}$ следует $\{\phi'\}P\{\psi\}$ и $\{\phi\}P\{\psi'\}$.*
- *Из $\langle\phi\rangle P\langle\psi\rangle$ следует $\langle\phi'\rangle P\langle\psi\rangle$ и $\langle\phi\rangle P\langle\psi'\rangle$.* □

Следующая лемма позволяет по нескольким утверждениям о частичной или полной корректности получать новые:

Лемма 3 *Пусть дана программа P. Для любых программы P и предикатов ϕ, ψ_1 и ψ_2 выполнены следующие утверждения:*

- *Из $\{\phi\}P\{\psi_1\}$ и $\{\phi\}P\{\psi_2\}$ следует $\{\phi\}P\{\psi_1 \wedge \psi_2\}$.*
- *Из $\langle\phi\rangle P\langle\psi_1\rangle$ и $\langle\phi\rangle P\langle\psi_2\rangle$ следует $\langle\phi\rangle P\langle\psi_1 \wedge \psi_2\rangle$.* □

Пользуясь этой леммой, можно разбивать на несколько частей доказательство частичной или полной корректности программы.

1.2.5 Верификация программы деления

В качестве примера мы докажем полную корректность программы деления на рисунке 2. Мы будем исследовать ее поведение при неотрицательных значениях x_1 и x_2. Это условие выражается входным предикатом

$$\phi_0 \ : \ (x_1 \geq 0) \wedge (x_2 \geq 0)$$

Пусть X_1 обозначает значение переменной x_1 в начале выполнения программы, а X_2 есть начальное значение x_2. Мы хотим доказать, что если программа завершается, то по ее завершении переменные q и r удовлетворяют соотношению

$$\psi \ : \ (X_1 = q \cdot X_2 + r) \wedge (0 \leq r < X_2)$$

(Предикат ψ является, собственно, определением частного и остатка при делении). Определим предикат ψ_0 следующим образом:

$$\psi_0 \ : \ (x_1 = q \cdot x_2 + r) \wedge (0 \leq r < x_2)$$

В качестве первого варианта спецификации программы мы рассмотрим пару (ϕ_0, ψ_0). У этой спецификации есть существенный недостаток. Пусть программа P выполняет присваивание

$$(x_1, x_2, q, r) \leftarrow (0, 1, 1, 0)$$

и сразу же после этого завершается. Программа P полностью корректна относительно ϕ_0 и ψ_0. Тем не менее она, очевидно, не реализует желаемый алгоритм целочисленного деления.

Для того, чтобы улучшить спецификацию программы, мы введем две дополнительные целочисленные переменные X_1 и X_2. Эти переменные мы будем использовать для хранения начальных значений переменных x_1 и x_2. Мы можем теперь взять следующий предикат ϕ в качестве входного предиката:

$$\phi \ : \ (x_1 = X_1 \geq 0) \wedge (x_2 = X_2 \geq 0)$$

В качестве выходного предиката можно теперь взять предикат ψ, определенный выше.

Как правило, в дальнейшем мы будем часто вводить переменные, которые, подобно X_1 и X_2, не используются в программе, но нужны для задания спецификаций. Они называются *спецификационными* переменными. Мы будем включать их в последовательность $\bar{v}$ вместе с переменными, которые используются в программе. Каждое состояние памяти таким образом, будет определять некоторые значения спецификационных переменных. (Отметим, что значение любой из них будет одинаковым во всех конфигурациях любого вычисления).

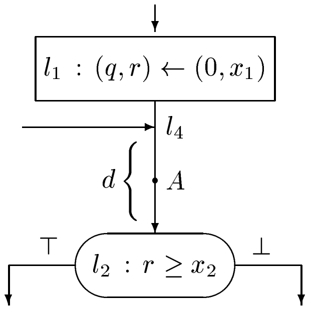
Для программы P_{idiv} мы сначала докажем ее частичную корректность относительно ϕ и ψ. После этого мы покажем, что эта программа завершается на

$$\phi' \ : \ (x_1 \geq 0) \wedge (x_2 > 0)$$

Наконец, из частичной корректности программы относительно ϕ и ψ и ее завершаемости на ϕ' мы сделаем вывод о ее полной корректности относительно $\phi \wedge (X_2 > 0)$ и ψ. Прокомментируем различие между ϕ

и ϕ': при анализе завершаемости важно исключить случай $x_2 = 0$, потому что программа не завершается при $x_2 = 0$.

Частичная корректность. Рассмотрим в программе P_{idiv} ребро $d = (l_4, l_2)$ (то есть ребро, ведущее из оператора l_4 в оператор l_2). Про ребро d мы будем говорить, что на нем имеется *точка сечения.* Эту точку сечения мы обозначим буквой A:



Рассмотрим теперь три пути, на которые точка A разбивает граф программы (мы будем допускать точку A в качестве начальной или конечной точки путей наряду с операторами START и HALT): $\alpha = l_0, l_1, l_4, A$, $\beta = A, l_2, l_3, l_4, A$ и $\gamma = A, l_2, l_5, l_*$. Любое завершающееся вычисление программы сперва идет по пути α, затем проходит путь β несколько раз (возможно, ни разу) и, наконец, заканчивается путем γ.

Найдем теперь предикат $I(\bar{v})$, который истиннен в любом состоянии после выполнения оператора l_4 и перед выполнением оператора l_2. (Про такой предикат I мы будем говорить, что он истиннен при любом прохождении точки A). Для наших целей предикат I можно определить следующим образом:

$$I(x_1, x_2, q, r) \ : \ \left\{ \begin{array}{c} X_1 = q \cdot X_2 + r \\ r \geq 0 \\ x_1 = X_1 \\ x_2 = X_2 \end{array} \right\}$$

Мы будем обычно опускать в списке агрументов предикатов спецификационные переменные и те переменные, от которых данный предикат зависит фиктивно. Рассмотрим произвольное конечное вычисление ρ программы деления. Пусть σ_0 есть его начальное состояние памяти (то есть $\rho = \pi(P_{idiv}, \sigma_0)$). Пусть ρ проходит через точку A количество раз, равное n, и состояние памяти при i-м прохождении точки A есть σ_i. Докажем, что для любого конечного вычисления ρ программы деления выполнены три свойства:

(НАЧАЛО): Из истинности ϕ в начале вычисления следует, что предикат I истиннен при первом прохождении точки A:

$$\sigma_0 \models \phi \Rightarrow \sigma_1 \models I$$

(ЦИКЛ): Если

- при некотором прохождении точки A истиннен предикат I,
- и это прохождение не является последним,

то при следующем прохождении точки A также будет истиннен предикат I:

$$\forall i \ : \ (1 \leq i < n \wedge \sigma_i \models I) \Rightarrow \sigma_{i+1} \models I$$

(ВЫХОД): Если при последнем прохождении точки A был истиннен I, то после завершения работы программы состояние памяти будет удовлетворять ψ:

$$\sigma_n \models I \Rightarrow val(\pi) \models \psi$$

Отметим, что если все эти условия выполнены для любого конечного вычисления ρ, то программа P_{idiv} частично корректна относительно ϕ и ψ. Условие (НАЧАЛО) мы докажем, анализируя путь α. Аналогично, для доказательства других двух условий мы проведем анализ путей β и γ. Для каждого пути мы будем поочередно рассматривать его операторы, двигаясь от конца пути к его началу.

(НАЧАЛО): Найдем условие на значения переменных в начале вычисления, эквивалентное истинности предиката $I(x_1, x_2, q, r)$ в точке A, то есть после выполнения оператора l_4. Таким условием является формула $I(x_1, x_2, 0, x_1)$. Она получается подстановкой в предикат I значения 0 вместо всех вхождений переменной q и переменной x_1 вместо всех вхождений переменной r. Таким образом, для того, чтобы доказать свойство (НАЧАЛО) для любого конечного ρ, достаточно доказать истинность следующей формулы:

$$\phi(x_1, x_2) \to I(x_1, x_2, 0, x_1)$$

или

$$\left\{ \begin{array}{c} x_1 = X_1 \geq 0 \\ x_2 = X_2 \geq 0 \end{array} \right\} \to \left\{ \begin{array}{c} X_1 = 0 \cdot X_2 + X_1 \\ X_1 \geq 0 \\ x_1 = X_1 \\ x_2 = X_2 \end{array} \right\} \tag{1}$$

(ЦИКЛ): Чтобы доказать это свойство, нам нужно проанализировать условный оператор. Пусть при работе программы был сделан виток цикла, то есть был пройден путь β. Истинность предиката $I(x_1, x_2, q, r)$ в точке A после прохождения пути β эквивалентна истинности формулы $I(x_1, x_2, q+1, r-x_2)$ перед выполнением оператора l_3. Докажем истинность следующей формулы:

$$\left\{ \begin{array}{c} I(x_1, x_2, q, r) \\ r \geq x_2 \end{array} \right\} \to I(x_1, x_2, q+1, r-x_2),$$

что эквивалентно

$$\left\{ \begin{array}{c} X_1 = q \cdot X_2 + r \\ r \geq 0 \\ r \geq x_2 \\ x_1 = X_1 \\ x_2 = X_2 \end{array} \right\} \to \left\{ \begin{array}{c} X_1 = (q+1) \cdot X_2 + (r - x_2) \\ r - x_2 \geq 0 \\ x_1 = X_1 \\ x_2 = X_2 \end{array} \right\} \tag{2}$$

Из истинности формулы (2) следует, что свойство (ЦИКЛ) истинно для любого конечного ρ. Действительно, пусть

- формула (2) истинна,
- дано конечное вычисление ρ,
- дано число i, такое что i-е прохождение вычисления ρ через точку A не является последним,
- $\sigma_i \models I$.

Тогда из точки A вычисление после условного оператора идет по дуге с меткой $\top$. Это значит, что $\sigma_i \models r \geq x_2$. Так как $\sigma_i \models I$, истинность формулы (2) дает $\sigma_i \models I(x_1, x_2, q+1, r-x_2)$. Условный оператор не меняет значений переменных, а значит, перед выполнением оператора l_3 состояние памяти по-прежнему будет равно σ_i. Следовательно, $\sigma_{i+1} \models I$.

(ВЫХОД): Для этого свойства рассуждения аналогичны. Для того, чтобы доказать его, достаточно доказать истинность следующей формулы:

$$I(x_1, x_2, q, r) \to \big((r < x_2) \to \psi(q, r)\big).$$

После подстановки выражений для I и ψ получаем

$$\left\{ \begin{array}{c} X_1 = q \cdot X_2 + r \\ r \geq 0 \\ r < x_2 \\ x_1 = X_1 \\ x_2 = X_2 \end{array} \right\} \to \left\{ \begin{array}{c} X_1 = q \cdot X_2 + r \\ 0 \leq r < X_2 \end{array} \right\} \tag{3}$$

Нетрудно убедиться, что каждая из формул (1), (2) и (3) истинна при любых значениях переменных. Отсюда следует частичная корректность программы P_{idiv} относительно ϕ и ψ.

Завершаемость. Фактически, до сих пор мы доказали правильность программы только частично: мы доказали, что если значения входных переменных удовлетворяют входному предикату и если выполнение программы завершается, то по ее завершении будет истиннен выходной предикат. Однако мы еще не доказали, что программа действительно завершает свою работу.

Сейчас мы докажем, что программа завершается при любых значениях входных переменных x_1 и x_2, удовлетворяющих предикату $\phi'(x_1, x_2)$, где

$$\phi'(x_1, x_2) \ : \ (x_1 \geq 0) \wedge (x_2 > 0)$$

(Как было сказано выше, программа не завершается при $x_2 = 0$.) Для начала мы покажем, что предикат

$$J(x_2, r) \ : \ (r \geq 0) \wedge (x_2 > 0)$$

истиннен всегда, когда вычисление проходит точку A. Для этого нам нужно доказать истинность следующих двух формул:

$$\phi'(x_1, x_2) \rightarrow J(x_2, 0) \qquad \text{(Путь } \alpha\text{)}$$

$$\left\{ \begin{array}{c} J(x_2, r) \\ r \geq x_2 \end{array} \right\} \rightarrow J(x_2, r - x_2) \qquad \text{(Путь } \beta\text{)}$$

После подстановки выражений для $\phi'(x_1, x_2)$ и $J(x_2, r)$ эти условия принимают вид

$$\left\{ \begin{array}{c} x_1 \geq 0 \\ x_2 > 0 \end{array} \right\} \quad \rightarrow \quad \left\{ \begin{array}{c} 0 \geq 0 \\ x_2 > 0 \end{array} \right\}$$

$$\left\{ \begin{array}{c} r \geq 0 \\ x_2 > 0 \\ r \geq x_2 \end{array} \right\} \quad \rightarrow \quad \left\{ \begin{array}{c} r - x_2 \geq 0 \\ x_2 > 0 \end{array} \right\}$$

Очевидно, что обе эти формулы истинны. Итак, мы доказали, что предикат $J(x_2, r)$ истиннен при любом прохождении точки A. Он будет также всегда истиннен перед выполнением оператора l_3, так как условный оператор l_2 не меняет значений переменных. Значит, $x_2 > 0$ всегда при выполнении оператора l_3. Этот оператор вычитает из r значение x_2, поэтому при любых двух последовательных прохождениях точки A значение r при втором прохождении будет строго меньше, чем при первом. Мы доказали также, что $r \geq 0$ при *любом* прохождении точки A. Поскольку не существует бесконечной строго убывающей последовательности целых неотрицательных чисел, программа не может выполнять цикл бесконечное число раз (при входных данных, удовлетворяющих ϕ'). А значит, программа P_{idiv} завершается на ϕ'.

Полная корректность. По лемме 2, из $\{\phi\}P_{idiv}\{\psi\}$ и $\phi \wedge (x_2 > 0) \rightarrow \phi$ следует $\{\phi \wedge (x_2 > 0)\}P_{idiv}\{\psi\}$. По той же лемме, из $\langle\phi'\rangle P_{idiv}\langle\top\rangle$ и $\phi \wedge (x_2 > 0) \rightarrow \phi'$ следует $\langle\phi \wedge (x_2 > 0)\rangle P_{idiv}\langle\top\rangle$. Наконец, из

$$\begin{array}{ll} \{\phi \wedge (x_2 > 0)\}P_{idiv}\{\psi\} & \text{и} \\ \langle\phi \wedge (x_2 > 0)\rangle P_{idiv}\langle\top\rangle & \end{array}$$

по лемме 1 следует $\langle\phi \wedge (x_2 > 0)\rangle P_{idiv}\langle\psi\rangle$.

1.2.6 Индуктивные утверждения: доказательство частичной корректности

В этом разделе мы рассмотрим общий метод доказательства частичной корректности блок-схем. Этот метод носит название *метод индуктивных утверждений Флойда.* Его мы будем также обозначать символом **F**. В этом методе используется идеи, примененные при верификации программы деления. Пусть дана последовательная программа P, входной предикат ϕ и выходной предикат ψ. Доказательство частичной корректности P относительно ϕ и ψ мы будем проводить следующим образом.

Шаг 1. Точки сечения. Сначала мы выбираем множество точек сечения на ребрах блок-схемы. Точки сечения выбираются так, чтобы были выполнены следующие условия:

- Каждый цикл в программе содержит по крайней мере одну точку сечения.
- На ребре, ведущем из начального оператора, имеется точка сечения. Ее мы будем называть *начальной точкой.*

- Для каждого завершающего оператора мы поместим точку сечения на ребро, ведущее в этот оператор. Такие точки мы будем называть *завершающими*.

Каждую точку сечения мы будем обозначать при помощи уникального имени.

Мы будем рассматривать пути в программы,

- в концах которых находятся точки сечения, и
- которые не содержат промежуточных точек сечения.

Такие пути мы будем называть *базовыми*. Отметим, что начальная точка базового пути может совпадать с завершающей (примером такого пути является путь β в программе деления). Поскольку каждый цикл в программе содержит хотя бы одну точку сечения, любой базовый путь конечен, и всего существует конечное число базовых путей.

Для каждого базового пути α из точки сечения s в точку сечения t мы определим предикат $R_\alpha(\bar{v})$ и функцию $T_\alpha(\bar{v})$. Предикат $R_\alpha(\bar{v}) : \Sigma \to \{\top, \bot\}$ определяет, каково должно быть состояние памяти при прохождении точки s, чтобы вычисление шло из этой точки по пути α. Функции $T_\alpha : \Sigma \to \Sigma$ сопоставляет состоянию памяти перед прохождением пути α, соответствующее ему состояние после прохождения этого пути.

Как $R_\alpha(\bar{v})$, так и $T_\alpha(\bar{v})$ могут быть выражены через функции и предикаты, входящие в операторы пути α; простым способом их вычисления является *метод обратных подстановок*.

Пусть последовательность операторов пути α есть $\lambda_1, \ldots, \lambda_n$, а ребро с точкой t ведет в оператор λ_{n+1}. Для каждого m, $1 \le m \le n+1$, мы определим предикат $R^m_\alpha(\bar{v})$ и функцию $T^m_\alpha(\bar{v})$. Если $m \ne n+1$, то предикат R^m_α определяет, какие значения должны принимать переменные перед выполнением оператора λ_m, чтобы вычисление после этого оператора шло по пути α до точки t. Функция T^m_α при $m \ne n+1$ определяет преобразование переменных при выполнении последовательности операторов от λ_m до λ_n. Из этих определений следует, что

$$R_\alpha(v) \stackrel{def}{=} R^1_\alpha(\bar{v}) \tag{4}$$

$$T_\alpha(v) \stackrel{def}{=} T^1_\alpha(\bar{v}) \tag{5}$$

Предикаты R^m_α и функции T^m_α мы определим нисходящей индукцией по m, двигаясь от $n+1$ к 1.

Базис индукции:

$$R^{n+1}_\alpha(\bar{v}) = \top \qquad T^{n+1}_\alpha(\bar{v}) = \bar{v}$$

Индуктивное предположение: Предположим, что $R^{m+1}_\alpha(\bar{v})$ и $T^{m+1}_\alpha(\bar{v})$ уже определены для некоторого m, $1 \le m \le n$.

Индуктивный переход: Определим $R^m_\alpha(\bar{v})$ и $T^m_\alpha(\bar{v})$ в зависимости от оператора λ_m:

- Если это оператор присваивания вида $\lambda_m : \bar{x} \leftarrow \bar{e}$, то положим

$$R^m_\alpha(\bar{v}) \stackrel{def}{=} R^{m+1}_\alpha(\bar{v}[\bar{e} \mid \bar{x}])$$

$$T^m_\alpha(\bar{v}) \stackrel{def}{=} T^{m+1}_\alpha(\bar{v}[\bar{e} \mid \bar{x}])$$

- Если это условный оператор $\lambda_m : B$, то положим

$$T^m_\alpha(\bar{v}) \stackrel{def}{=} T^{m+1}_\alpha(\bar{v})$$

  Предикат $R^m_\alpha(\bar{v})$ мы определим следующим образом. Если ребро, принадлежащее пути α и ведущее из оператора λ_m в оператор λ_{m+1}, помечено символом $\top$, то положим

$$R^m_\alpha \stackrel{def}{=} R^{m+1}_\alpha \wedge B$$

  Если же это ребро помечено символом $\bot$, то положим

$$R^m_\alpha \stackrel{def}{=} R^{m+1}_\alpha \wedge \neg B$$

- Если это оператор соединения λ_m : JOIN, то положим

$$\begin{array}{rcl} R_\alpha^m(\bar{v}) & \stackrel{def}{=} & R_\alpha^{m+1}(\bar{v}) \\ T_\alpha^m(\bar{v}) & \stackrel{def}{=} & T_\alpha^{m+1}(\bar{v}) \end{array}$$

После построения всех предикатов R_α^m и функций T_α^m определим R_α и T_α в соответствии с (4) и (5).

Шаг 2. Индуктивные утверждения. На втором шаге мы выбираем для каждой точки сечения s некоторый предикат $I_s(\bar{v})$, описывающий соотношения между переменными при прохождении точки s (при условии, что начальное состояние удовлетворяет входному предикату ϕ). Предикат $I_s(\bar{v})$ мы будем называть *индуктивным утверждением* для s. Индуктивным утверждением для начальной точки является входной предикат ϕ, а для любой завершающей – выходной предикат ψ.

Шаг 3. Условия верификации. На третьем шаге для каждого базового пути α, ведущего из точки s в точку t, строится *условие верификации*

$$\forall \bar{v} \ : \ \left\{ \begin{array}{c} I_s(\bar{v}) \\ R_\alpha(\bar{v}) \end{array} \right\} \rightarrow I_t\big(T_\alpha(\bar{v})\big) \tag{6}$$

Так как базовых путей в программе конечное число, мы получим конечное число условий верификации.

Завершающий шаг состоит в доказательстве истинности этих условий. Если все они истинны, то следующее утверждение позволяет сделать важный вывод об индуктивных утверждениях:

Утверждение 4 *Пусть все условия верификации (6) истинны. Пусть дано вычисление π программы P, начальное состояние которого удовлетворяет предикату ϕ. Тогда всегда, когда вычисление π проходит точку сечения s, предикат $I_s(\bar{v})$ истиннен при текущих значениях переменных.* □

Из этого утверждения следует, что если

- все условия верификации истинны,
- начальное состояние вычисления удовлетворяет ϕ, и
- программа завершает работу,

то при достижении завершающей точки сечения будет истиннен выходной предикат ψ. Мы получаем, что истинность условий верификации влечет частичную корректность P относительно ϕ и ψ.

Подводя итог, сформулируем следующую теорему:

Теорема 5 (Метод индуктивных утверждений, Флойд) *Пусть дана блок-схема P, входной предикат ϕ и выходной предикат ψ. Выполним следующие действия:*

1. *Выберем точки сечения;*
2. *Найдем подходящий набор индуктивных утверждений;*
3. *Построим условия верификации (6) для всех базовых путей.*

Если все условия верификации истинны, то блок-схема P частично корректна относительно ϕ и ψ. □

1.2.7 Пример: частичная корректность программы извлечения квадратного корня

Программа P_{sqrt} над целыми числами, изображенная на рисунке 3, вычисляет целую часть $z = \lfloor\sqrt{x}\rfloor$ квадратного корня неотрицательного целого x. Для этого она последовательно перебирает квадраты целых чисел, начиная с 0, до тех пор, пока квадрат очередного числа не превысит x. Текущее число хранится в переменной z. Переменная y_1 в процессе работы программы иногда принимает значение z^2, а иногда – $(z+1)^2$. При увеличении z на единицу значение z^2 должно увеличиться на

$$(z+1)^2 - z^2 = 2z + 1$$

Это приращение, равное $2z+1$, хранится в переменной y_2.

Мы докажем, что программа частично корректна относительно входного предиката

$$\phi(x) \ : \ x = X \geq 0$$

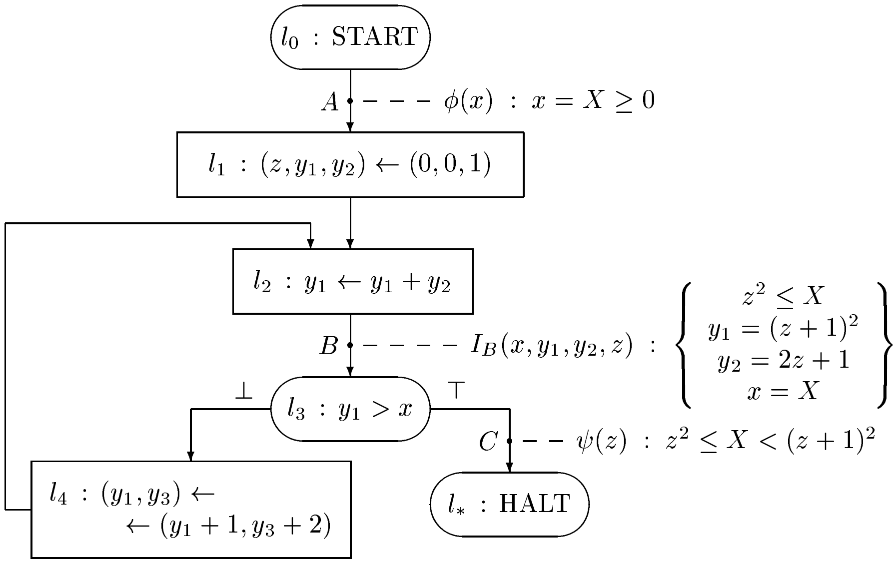


Рис. 3: Программа P_{sqrt} извлечения квадратного корня (частичная корректность)

и выходного предиката

$$\psi(z) \ : \ z^2 \leq X < (z+1)^2$$

Воспользуемся методом индуктивных утверждений. Разобъем единственный цикл программы точкой сечения B. Начальную и завершающую точки мы обозначим символами A и C. При таком выборе точек сечения в программе имеются три базовых пути: $\alpha = A, l_1, l_2, B$, $\quad \beta = B, l_3, l_4, l_2, B$ и $\gamma = B, l_3, C$. Найдем предикаты R и функции T для каждого из этих путей методом обратных подстановок.

Для представления отображений из множества Σ в себя мы будем использовать особую форму записи. Пусть $\bar{x} = (x_1, \ldots, x_m)$ – вектор переменных. Пусть также для любого i, $1 \leq i \leq m$, g_i является функцией, действующей из Σ в D_{x_i}. Тогда запись $T(\bar{v})[\![\bar{x}]\!] = \big(g_1(\bar{v}), \ldots, g_m(\bar{v})\big)$ означает, что

$$T(\sigma)[\![v]\!] \stackrel{def}{=} \left\{ \begin{array}{ll} g_i(\sigma), & \text{если } v = x_i \\ \sigma[\![v]\!], & \text{если } v \notin \{x_1, \ldots, x_m\} \end{array} \right.$$

для любого состояния памяти $\sigma \in \Sigma$ и для любой переменной v.

На рисунке 4 показаны предикаты R^m и функции T^m, возникающие при применении метода обратных подстановок. Итак, искомые предикаты R и функции T определяются следующими соотношениями:

$$\begin{array}{rcl} R_\alpha(\bar{v}) & : & \top \\ T_\alpha(\bar{v})[\![y_1, y_2, z]\!] & = & (1, 1, 0) \\ R_\beta(\bar{v}) & : & y_2 \leq x \\ T_\beta(\bar{v})[\![y_1, y_2, z]\!] & = & (y_1 + y_2 + 2, y_2 + 2, z + 1) \\ R_\gamma(\bar{v}) & : & y_2 > x \\ T_\gamma(\bar{v}) & = & \bar{v} \end{array}$$

Выберем теперь для точки сечения B следующее индуктивное утверждение

$$I_B(x_1, y_1, y_2, z) \ : \ \left\{ \begin{array}{c} z^2 \leq x \\ y_1 = (z+1)^2 \\ y_2 = 2z+1 \\ X = x \end{array} \right\} \tag{7}$$

Кроме того, точку A мы пометим входным предикатом $\phi(x)$, а точку C – выходным предикатом $\psi(z)$.

Путь α:

$$\begin{tikzcd}
A \arrow[d] \arrow[r, dash, dashed] & \begin{cases} R^1_\alpha(\bar{v}) \ : \ \top \\ T^1_\alpha(\bar{v})\llbracket y_1, y_2, z \rrbracket = (1, 1, 0) \end{cases} \\
l_1 \arrow[d] \arrow[r, dash, dashed] & \begin{cases} R^2_\alpha(\bar{v}) \ : \ \top \\ T^2_\alpha\llbracket y_1 \rrbracket = y_1 + y_2 \end{cases} \\
l_2 \arrow[d] & \\
B \arrow[r, dash, dashed] & \begin{cases} R^3_\alpha(\bar{v}) \ : \ \top \\ T^3_\alpha(\bar{v}) \ : \ \bar{v} \end{cases}
\end{tikzcd}$$

Путь γ:

$$\begin{tikzcd}
B \arrow[d] \arrow[r, dash, dashed] & \begin{cases} R^1_\gamma(\bar{v}) \ : \ y_1 > x \\ T^1_\gamma(\bar{v}) = \bar{v} \end{cases} \\
l_4 \arrow[d, "\top"'] & \\
C \arrow[r, dash, dashed] & \begin{cases} R^2_\gamma(\bar{v}) \ : \ \top \\ T^2_\gamma(\bar{v}) = \bar{v} \end{cases}
\end{tikzcd}$$

Путь β:

$$\begin{tikzcd}
B \arrow[d] \arrow[r, dash, dashed] & \begin{cases} R^1_\beta(\bar{v}) \ : \ y_1 \le x \\ T^1_\beta(\bar{v})\llbracket y_1, y_2, z \rrbracket = (y_1 + y_2 + 2, y_2 + 2, z + 1) \end{cases} \\
l_3 \arrow[d, "\perp"'] \arrow[r, dash, dashed] & \begin{cases} R^2_\beta(\bar{v}) \ : \ \top \\ T^2_\beta(\bar{v})\llbracket y_1, y_2, z \rrbracket = (y_1 + y_2 + 2, y_2 + 2, z + 1) \end{cases} \\
l_4 \arrow[d] \arrow[r, dash, dashed] & \begin{cases} R^3_\beta(\bar{v}) \ : \ \top \\ T^3_\beta(\bar{v})\llbracket y_1 \rrbracket = y_1 + y_2 \end{cases} \\
l_2 \arrow[d] & \\
B \arrow[r, dash, dashed] & \begin{cases} R^4_\beta(\bar{v}) \ : \ \top \\ T^4_\beta(\bar{v}) = \bar{v} \end{cases}
\end{tikzcd}$$

Рис. 4: Анализ базовых путей в программе P_{sqrt}

Нам остается доказать истинность трех условий верификации для любых целых чисел x, y_1, y_2 и y_3.

Для пути α:

$$\big(\phi(x) \land \top\big) \to I_B(x, 1, 1, 0)$$

то есть

$$(x = X \geq 0) \to \left\{ \begin{array}{c} 0^2 \leq x \\ 1 = (0+1)^2 \\ 1 = 2 \cdot 0 + 1 \\ x = X \end{array} \right\} \tag{8}$$

Для пути β:

$$\left\{ \begin{array}{c} I_B(x, y_1, y_2, z) \\ y_1 \leq x \end{array} \right\} \to I_B(x, y_1 + y_2 + 2, y_2 + 2, z + 1)$$

то есть

$$\left\{ \begin{array}{c} z^2 \leq x \\ y_1 = (z+1)^2 \\ y_2 = 2z + 1 \\ X = x \\ y_1 \leq x \end{array} \right\} \to \left\{ \begin{array}{c} (z+1)^2 \leq x \\ y_1 + y_2 + 2 = (z+2)^2 \\ y_2 + 2 = 2(z+1) + 1 \\ X = x \end{array} \right\} \tag{9}$$

Для пути γ:

$$\left\{ \begin{array}{c} I_B(x, y_1, y_2, z) \\ y_1 > x \end{array} \right\} \to \psi(z)$$

то есть

$$\left\{ \begin{array}{c} z^2 \leq x \\ y_1 = (z+1)^2 \\ y_2 = 2z + 1 \\ X = x \\ y_1 > x \end{array} \right\} \to \big(z^2 \leq X < (z+1)^2\big) \tag{10}$$

Поскольку формулы (8), (9) и (10) истинны, программа P_{sqrt} частично корректна относительно входного предиката $x = X \geq 0$ и выходного предиката $z^2 \leq X < (z+1)^2$.

1.3 Последовательные программы с массивами

Массивом называется структура данных в языках программирования, состоящая из набора компонент, называемых *элементами* массива. Все элементы массива имеют один и тот же тип, который называется *базовым* типом массива. Отдельную компоненту массива можно выделить при помощи целого числа, называемого *индексом*.

Для того, чтобы описывать работу с массивами в последовательных программах, мы расширим синтаксис для операторов. Для каждого массива, встречающегося в программе, мы будем отводить отдельную переменную типа массив, в которой будет храниться содержимое массива. Если X – это переменная типа массив, а i – целочисленное выражение, то через $X[i]$ мы будем обозначать элемент массива X с индексом i. Выражения вида $X[i]$ мы будем использовать в условных операторах а также в левых и правых частях операторов присваивания. Вот некоторые примеры использования массивов (здесь i – целочисленная переменная):

- Оператор $y \leftarrow S[i]$ заносит в переменную y значение i-го элемента массива S.
- Оператор $S[i] \leftarrow 0$ заносит в i-й элемент массива S значение 0.
- Оператор $(S[i], S[i+1]) \leftarrow (S[i+1], S[i])$ меняет местами содержимое i-го и $i+1$-го элеметов массива S.
- Если S и T – два массива с одинаковым базовым типом, то присваивание $S \leftarrow T$ заносит в массив S содержимое массива T.

Индекс массива не может быть произвольным числом, а должен принадлежать некоторому диапазону. А именно, число i может быть индексом элемента массива X только если оно удовлетворяет неравенству $L_X \le i \le H_X$, где L_X и H_X – *нижняя* и *верхняя* границы массива X. При этом может случиться, например, что в некотором операторе мы использовали выражение $X[i]$, а при выполнении этого оператора значение i оказалось за пределами границ массива. Мы не будем определять, как ведет себя программа в подобных случаях. При верификации программ иногда требуется доказать отсутствие выхода индекса за границы массива. Эту задачу можно решить методом индуктивных утверждений Флойда.

Пусть, например,

- в программе имеется оператор присваивания $l\ :\ y \leftarrow X[i+j]$,
- массив X содержит 21 элемент $X[0], \dots, X[20]$,
- задан некоторый входной предикат ϕ.

Предположим, что нам нужно доказать, что если начальное состояние памяти удовлетворяет ϕ, то при выполнении оператора l не происходит выхода индекса за границы массива.

Поместим точку сечения A на ребро, ведущее в оператор l. Выберем на ребрах программы другие точки сечения и укажем для них индуктивные утверждения в соответствии с методом Флойда. Пусть выполнены следующие условия:

- Индуктивное утверждение I_A для точки A таково, что истинна формула
$$I_A \to \big(0 \le i + j \le 20\big)$$
- Все условия верификации (6) истинны.
- При запуске программы начальное состояние памяти удовлетворяет входному предикату.

Согласно утверждению 4 из истинности данных условий следует, что при выполнении оператора l не происходит выхода индекса за границы массива.

1.3.1 Пример: частичная корректность программы пузырьковой сортировки

Рассмотрим программу P_{sort}, изображенную на рисунке 5. (Утверждения, помечающие ее ребра, можно пока не принимать во внимание). Эта программа сортирует по возрастанию массив S из $n+1$ $(n \ge 0)$ действительных чисел $S[0], \dots, S[n]$, используя метод пузырьковой сортировки.

Для задания спецификации этой программы введем две спецификационные переменные:

- Переменная N хранит начальное значение n.
- Переменная X хранит начальное значение массива S.

Докажем, что по завершении программы P_{sort} выполнены следующие два условия:

1. Элементы массива S (в количестве $n+1$) являются перестановкой элементов исходного массива X. Для того, чтобы описать это условие при помощи формулы, введем предикат $perm(X, S, k, l)$, где X и S – массивы, а k и l – целые числа. Предикат $perm(X, S, k, l)$ истиннен тогда и только тогда, когда определенный участок массива S получается из участка массива X перестановкой элементов. Эти участки в массивах состоят из элементов с индексами от k до l включительно. При этом мы будем считать, что $perm(X, S, k, l) = \top$ если $k \ge l$. Итак, первое свойство, которое должно быть выполнено по завершении программы, описывается предикатом
$$\psi_1(S)\ :\ perm(X, S, 0, N)$$

2. Элементы массива S упорядочены по возрастанию. Введем предикат $ordered(S, k, l)$, который истиннен тогда и только тогда, когда участок массива S, начинающийся с индекса k и кончающийся индексом l, упорядочен по возрастанию. При этом мы будем считать, что $ordered(X, S, k, l) = \top$ если $k \ge l$. Предикат $ordered(S, k, l)$, таким образом, имеет следующее определение:
$$ordered(S, k, l)\ :\ (k \ge l) \to \Big(S[k] \le S[k+1] \le \dots S[l]\Big)$$
Таким образом, второе требование к завершающему состоянию памяти описывается следующей формулой:
$$\psi_2(S)\ :\ ordered(S, 0, N)$$

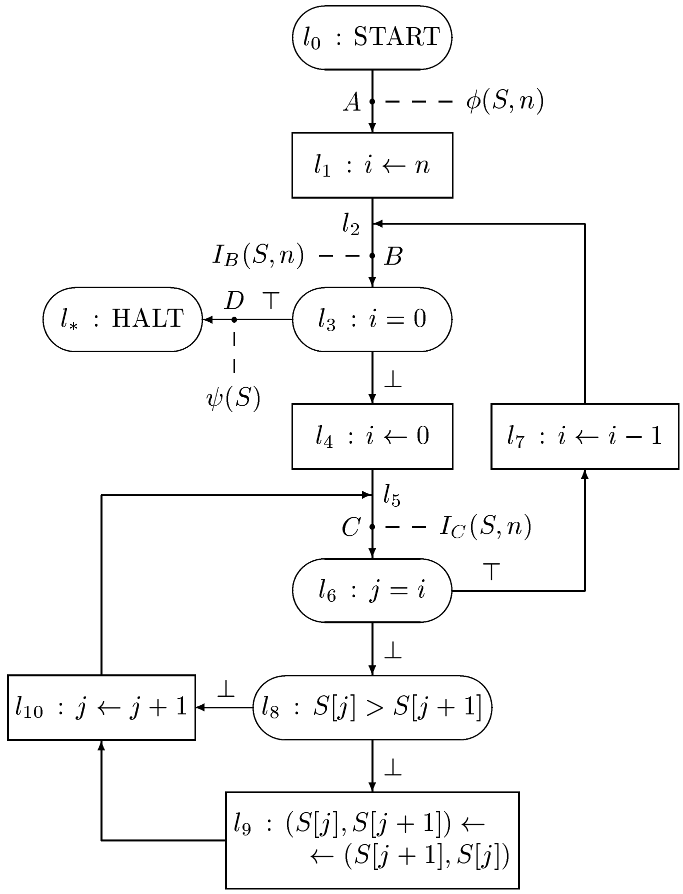


Рис. 5: Программа P_{sort} пузырьковой сортировки массива.

Итак, выходной предикат для программы сортировки есть

$$\psi(S) \; : \; \psi_1(S) \wedge \psi_2(S)$$

Мы будем использовать следующий входной предикат:

$$\phi(S,n) \; : \; \left\{ \begin{array}{c} X = S \\ n = N \geq 0 \end{array} \right\}$$

По лемме 3, для доказательства $\{\phi\}P_{sort}\{\psi\}$ нам достаточно доказать $\{\phi\}P_{sort}\{\psi_1\}$ и $\{\phi\}P_{sort}\{\psi_2\}$.

Для того, чтобы обосновать[1] $\{\phi\}P_{sort}\{\psi_1\}$, предположим, что в начальном состоянии памяти при работе программы сортировки истиннен предикат ϕ. Тогда будут верны следующие утверждения:

- Так как формула $(S = X) \wedge (N \geq 0) \to perm(X, S, 0, N)$ является тавтологией, предикат $perm(X, S, 0, N)$ будет истиннен в начальном состоянии памяти.
- Единственным оператором, который может менять содержимое массива S, является оператор l_9. Этот оператор сохраняет содержимое массива S с точностью до перестановки его элементов. Поэтому предикат $perm(X, S, 0, N)$ будет истиннен в любом состоянии памяти при работе программы, а значит и в завершающем состоянии.

Для доказательства $\{\phi\}P_{sort}\{\psi_2\}$ мы разобьем три цикла программы точками сечения B и C и найдем для этих точек подходящие индуктивные утверждения. Мы будем использовать выражения $\{S[0], \ldots, S[l]\}$ для обозначения множества $\{S[m] \mid k \leq m \leq l\}$ (при $k > l$ это множество пусто). Далее, для любых двух множеств действительных чисел T_1 и T_2 выражение $T_1 \leq T_2$ означает, что любой элемент множества T_1 меньше или равен любому элементу множества T_2 (при этом $T_1 \leq T_2$ истинно если $T_1 = \emptyset$ или $T_2 = \emptyset$).

Выберем индуктивные утверждения для точек сечения B и C. С использованием вышеприведенных обозначений они записываются следующим образом:

$$p_B(n,S,i) \quad : \quad \left\{ \begin{array}{c} n = N \geq 0 \\ 0 \leq i \leq N \\ ordered(S,i,N) \\ \{S[0], \ldots, S[i]\} \leq \{S[i+1], \ldots, S[N]\} \end{array} \right\}$$

$$p_C(n,S,i,j) \quad : \quad \left\{ \begin{array}{c} n = N \geq 0 \\ 0 \leq j \leq i \leq N \\ ordered(S,i,N) \\ \{S[0], \ldots, S[i]\} \leq \{S[i+1], \ldots, S[N]\} \\ \{S[0], \ldots, S[j-1]\} \leq \{S[j]\} \end{array} \right\}$$

В программе имеется шесть базовых путей:

α – из A в B	β_3 – из C в C через l_9
β_1 – из B в C	β_4 – из C в B
β_2 – из C в C минуя l_9	γ – из B в D

Шесть условий верификации, которые нужно доказать, имеют следующий вид:

Для пути α:
$$\phi(S,n) \quad \to \quad p_B(n,S,n) \tag{11}$$

Для пути γ:
$$\left\{ \begin{array}{c} p_B(n,S,i) \\ i = 0 \end{array} \right\} \quad \to \quad \psi(S) \tag{12}$$

Для пути β_1:
$$\left\{ \begin{array}{c} p_B(n,S,i) \\ i \neq 0 \end{array} \right\} \quad \to \quad p_C(n,S,i,0) \tag{13}$$

Для пути β_2:
$$\left\{ \begin{array}{c} p_C(n,S,i,j) \\ S[j] \leq S[j+1] \\ j \neq i \end{array} \right\} \quad \to \quad p_C(n,S,i,j+1) \tag{14}$$

[1] Доказательство $\{\phi\}P_{sort}\{\psi_1\}$ можно провести методом индуктивных утверждений Флойда, но здесь это доказательство не приводится. Вместо этого используются обычные математические рассуждения.

Для пути β_3: $$\left\{\begin{array}{c} p_C(n,S,i,j) \\ S[j] > S[j+1] \\ j \neq i \end{array}\right\} \quad\rightarrow\quad p_C\bigl(n, ex(S,j,j+1), i, j+1\bigr) \qquad (15)$$

Здесь $ex(S,k,l)$ обозначает массив S после перестановки элементов $S[k]$ и $S[l]$.

Для пути β_4: $$\left\{\begin{array}{c} p_C(n,S,i,j) \\ j = i \end{array}\right\} \quad\rightarrow\quad p_B(n,S,i-1) \qquad (16)$$

Мы не будем рассматривать доказательства всех этих условий верификации, а докажем только условие (16) и укажем некоторые приемы, которые могут использоваться для доказательства условия (15). После подстановки выражений для p_B и p_C условие (16) принимает вид

$$\left\{\begin{array}{c} n = N \geq 0 \\ 1 \leq i \leq N \\ 0 \leq j \leq i \\ ordered(S,i,N) \\ \{S[0],\ldots,S[i]\} \leq \{S[i+1],\ldots,S[N]\} \\ \{S[0],\ldots,S[j-1]\} \leq \{S[j]\} \\ j = i \end{array}\right\} \rightarrow \left\{\begin{array}{c} n = N \geq 0 \\ 0 \leq i-1 \leq N \\ ordered(S,i-1,N) \\ \{S[0],\ldots,S[i-1]\} \leq \{S[i],\ldots,S[N]\} \end{array}\right\}$$

Докажем истинность данной формулы.

- Из $1 \leq i \leq N$ очевидным образом следует $0 \leq i-1 \leq N$.
- Ввиду $S[j-1] \leq S[j]$ и $j = i$ мы имеем $S[i-1] \leq S[i]$.
- Из $ordered(S,i,N)$ и $S[i-1] \leq S[i]$ вытекает $ordered(S,i-1,N)$.
- Наконец $\{S[0],\ldots,S[i]\} \leq \{S[i+1],\ldots,S[N]\}$ и $\{S[0],\ldots,S[i-1]\} \leq \{S[i]\}$ влечет $\{S[0],\ldots,S[i-1]\} \leq \{S[i],\ldots,S[N]\}$, что завершает доказательство условия (16).

Доказательство условий (11) – (14) проводится аналогичным образом. Условие (15) ссылается на содержимое массива S как до, так и после перестановки в нем элементов $S[j]$ и $S[j+1]$. Рассмотрим некоторые методы, облегчающие доказательство условий такого вида. Докажем истинность следующей формулы:

$$\left\{\begin{array}{c} \{S[0],\ldots,S[j-1]\} \leq \{S[j]\} \\ S[j] > S[j+1] \\ 0 \leq j < N \end{array}\right\} \rightarrow \Bigl(\{S^*[0],\ldots,S^*[j]\} \leq \{S^*[j+1]\}\Bigr) \qquad (17)$$

где $S^* = ex(S,j,j+1)$. Формулу (17) можно использовать при доказательстве условия (15).

Во-первых, отметим, что неравенство слева от знака импликации

$$\{S[0],\ldots,S[j-1]\} \leq \{S[j]\}$$

можно записать при помощи кванторов следующим образом:

$$\forall k \,:\, (0 \leq k < j) \rightarrow \bigl(S[k] \leq S[j]\bigr)$$

Аналогичное преобразование можно применить и к неравенству справа от знака импликации. Итак, с использованием кванторов формулу (17) можно переписать следующим образом:

$$\left\{\begin{array}{c} \forall k \,:\, (0 \leq k < j) \rightarrow \bigl(S[k] \leq S[j]\bigr) \\ S[j] > S[j+1] \\ 0 \leq j < N \end{array}\right\} \rightarrow \forall k \,:\, (0 \leq k \leq j) \rightarrow \bigl(S^*[k] \leq S^*[j+1]\bigr) \qquad (18)$$

Далее, заметим, что для любого k значение $S^*[k]$ вычисляется следующим образом:

$$S^*[k] = \left\{\begin{array}{ll} S[j], & \text{если } k = j+1 \\ S[j+1], & \text{если } k = j \\ S[k], & \text{иначе} \end{array}\right.$$

К части (18) справа от знака импликации применим следующее преобразование. Неравенство $S^*[j] \leq S^*[j+1]$ выделим в виде отдельной подформулы. Правая часть (18) примет следующий вид:

$$\left\{ \begin{array}{c} S^*[j] \leq S^*[j+1] \\ \forall k\,:\,(0 \leq k < j) \rightarrow \big(S^*[k] \leq S^*[j+1]\big) \end{array} \right\}$$

Подставим сюда выражение для $S^*[k]$. В результате формула (18) преобразуется к следующему виду:

$$\left\{ \begin{array}{c} \forall k\,:\,(0 \leq k < j) \rightarrow \big(S[k] \leq S[j]\big) \\ S[j] > S[j+1] \\ 0 \leq j < N \end{array} \right\} \rightarrow \left\{ \begin{array}{c} \forall k\,:\,(0 \leq k < j) \rightarrow \big(S[k] \leq S[j]\big) \\ S[j+1] \leq S[j] \end{array} \right\}$$

Последняя формула, очевидно, истинна. Итак, истинность формулы (17) доказана.

Список литературы

[Анд82] Р. Андерсон. *Доказательство правильности программ.* “Мир”, Москва, 1982.

[Fra92] N. Francez. *Verification of programs.* Addison-Wesley Publishers Ltd., 1992.

[Man74] Z. Manna. *Mathematical theory of computation.* McGraw-Hill, 1974.