Верификация программ методом инвариантов

Миронов А.М.

Излагается метод инвариантов для доказательства правильности компьютерных программ. Основные концепции, связанные с этим методом, иллюстрированы примерами верификации последовательных и параллельных программ.

Ключевые слова: верификация программ, метод Флойда, инварианты.

1. Введение

Проблемы верификации (т.е. доказательства правильности) программ занимают 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 Дейкстра ([1], стр. 41), тестирование может лишь помочь выявить некоторые ошибки, но отнюдь не доказать их отсутствие. Ошибки в программах могут быть весьма тонкими, и чем тоньше ошибка, тем сложнее обнаружить её тестированием. Но во многих программах наличие даже незначительных ошибок категорически недопустимо. Например, наличие даже небольших 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.

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,
- программа 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 $Spec$.

В настоящем тексте излагается 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]. Метод инвариантов основан на понятиях математической логики, с которыми можно познакомиться по книге [5]. Различные изложения метода инвариантов содержатся в [6]–[11].

2. Программы, представленные в виде блок-схем

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

2.1. Вспомогательные понятия

Мы будем предполагать, что заданы следующие множества.

- Множество $\mathcal{T}$, элементы которого называются **типами**. Каждому типу $t \in \mathcal{T}$ сопоставлено множество D_t **значений** типа t.

- Множество $\mathcal{X}$, элементы которого называются **переменными**. Каждой переменной $x \in \mathcal{X}$ сопоставлен тип $t(x) \in \mathcal{T}$. Каждая переменная $x \in \mathcal{X}$ может принимать **значения** в множестве $D_{t(x)}$, т.е. в различные моменты времени переменная x может быть связана с различными элементами множества $D_{t(x)}$.

- Множество $\mathcal{C}$, элементы которого называются **константами**. Каждой константе $c \in \mathcal{C}$ сопоставлены тип $t(c) \in \mathcal{T}$ и значение из $D_{t(c)}$, обозначаемое тем же символом c, и называемое **интерпретацией** константы c.

- Множество $\mathcal{F}$, элементы которого называются **функциональными символами (ФС)**. Каждому ФС $f \in \mathcal{F}$ сопоставлены
  - **функциональный тип** $t(f)$, который представляет собой запись вида $(t_1, \ldots, t_n) \to t$, где $t_1, \ldots, t_n, t \in \mathcal{T}$, и
  - функция вида $D_{t_1} \times \ldots \times D_{t_n} \to D_t$, где $(t_1, \ldots, t_n) \to t \ = t(f)$, данная функция обозначается тем же символом f и называется **интерпретацией** ФС f.

  Ниже мы будем использовать ФС $+, -, \cdot, div, mod$, где
  - ФС $+, -, \cdot$ имеют функциональный тип $(\texttt{int}, \texttt{int}) \to \texttt{int}$, и $\texttt{int}$ – тип, значениями которого являются целые числа,
  - ФС div, mod имеют функциональный тип $(\texttt{int}, \texttt{int}_0) \to \texttt{int}$, и $\texttt{int}_0$ – тип, значениями которого являются целые числа, отличные от 0,

  и функции $+$, $-$, и $\cdot$ представляют собой соответствующие арифметические операции, div и mod вычисляют частное и остаток соответственно от деления первого аргумента на второй.

Выражения строятся из переменных, констант и ФС. Множество всех выражений обозначается символом $\mathcal{E}$. Каждое выражение e имеет тип $t(e) \in \mathcal{T}$, определяемый структурой выражения e. Правила построения выражений имеют следующий вид:

- каждая переменная и константа является выражением того типа, который сопоставлен этой переменной или константе, и
- если $e_1, \ldots, e_n \in \mathcal{E}$, $f \in \mathcal{F}$, и $t(f)$ имеет вид $(t(e_1), \ldots, t(e_n)) \to t$, то знакосочетание $f(e_1, \ldots, e_n)$ является выражением типа t.

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

Среди типов, входящих в $\mathcal{T}$, имеется тип $\texttt{bool}$, множество значений которого имеет вид $\{0, 1\}$. Выражения типа $\texttt{bool}$ называются **формулами**. Множество всех формул обозначается символом $\mathcal{B}$. При построении формул могут использоваться обычные булевские ФС ($\wedge, \vee, \to$ и т.д.), которым соответствуют функции коньюнкции, 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. В некоторых случаях формулы вида $e_1 \wedge \ldots \wedge e_n$ будут записываться в виде $\left[\!\!\left[\begin{array}{c} e_1 \\ \ldots \\ e_n \end{array} \right]\!\!\right]$.

$\forall\, e \in \mathcal{E}$ запись X_e обозначает множество переменных, входящих в e.

$\forall\, X \subseteq \mathcal{X}$ записи $\mathcal{E}(X)$ и $\mathcal{B}(X)$ обозначают множества $\{e \in \mathcal{E} \mid X_e \subseteq X\}$ и $\mathcal{E}(X) \cap \mathcal{B}$ соответственно.

Пусть $X \subseteq \mathcal{X}$. **Означиванием** переменных из множества X называется произвольное соответствие ξ, которое сопоставляет каждой переменной $x \in X$ некоторое значение $x^\xi \in D_{t(x)}$. Множество всех означиваний переменных из X обозначается записью $X^\bullet$.

Если $X \subseteq \mathcal{X}$, $e \in \mathcal{E}(X)$, $\xi \in X^\bullet$, то e^ξ обозначает значение из $D_{t(e)}$, называемое **значением** e **на** ξ, и определяемое следующим образом:

- если $e \in \mathcal{X}$, то e^ξ предполагается заданным,
- если $e \in \mathcal{C}$, то e^ξ является интерпретацией константы e, и
- если $e = f(e_1, \ldots, e_n)$, то $e^\xi = f(e_1^\xi, \ldots, e_n^\xi)$.

Выражения $e_1, e_2 \in \mathcal{E}$ считаются равными, если

$$\forall\, \xi \in (X_{e_1} \cup X_{e_2})^\bullet \;\; e_1^\xi = e_2^\xi.$$

Пусть $X \subseteq \mathcal{X}$, $\xi \in X^\bullet$, $x \in X$ $e \in \mathcal{E}(X)$, $t(x) = t(e)$. Запись $\xi(x := e)$ обозначает означивание из $X^\bullet$, определяемое следующим образом:

$$x^{\xi(x:=e)} = e^\xi, \;\; \forall\, x \in X \setminus \{x\} \;\; x^{\xi(x:=e)} = x^\xi. \tag{1}$$

Если $x \in \mathcal{X}$ $e, e' \in \mathcal{E}$, и $t(x) = t(e)$, то запись $(x := e)e'$ обозначает выражение, получаемое из e' заменой всех вхождений x в e' на e.

2.2. Понятие блок-схемы

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

- **начальная** вершина, она обозначается символом $\bigodot$, в каждой БС имеется только одна начальная вершина, из неё выходит одно ребро, и в неё не входит ни одного ребра,
- **присваивание**, вершина такого типа обозначается записью вида $[x := e]$, где $x \in \mathcal{X}$, $e \in \mathcal{E}$, $t(x) = t(e)$, из каждого присваивания выходит только одно ребро,

- **условный переход**, вершина такого типа обозначается записью вида (b) где $b \in \mathcal{B}$, из каждого условного перехода выходит два ребра, одно из которых имеет метку 1, а другое - метку 0,

- **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.

Пусть P – БС. Будем обозначать записями V_P и X_P совокупность всех вершин P и переменных, входящих в P, соответственно.

Выполнение БС P – это обход вершин P, начиная с начальной вершины (т.е. последовательность переходов по рёбрам от одной вершины P к другой), с выполнением действий, сопоставленных проходимым вершинам. С каждым шагом $i \geq 0$ выполнения P связаны вершина $v_i \in V_P$ и означивание $\xi_i \in X_P^\bullet$ (называемые **текущей вершиной** и **текущим означиванием** на шаге i). Если v_i – нетерминальная вершина, то определен $i+1$-й шаг выполнения P, в P есть ребро из v_i в v_{i+1}, и

- если v_i – начальная вершина или пустой оператор, то $\xi_{i+1} = \xi_i$,

- если $v_i = [x := e]$, то $\xi_{i+1} = \xi_i(x := e)$,
  (можно интерпретировать действие, соответствующее этой вершине, как обновление значения переменной x: после исполнения этого действия значение x становится равным значению выражения e на текущих значениях переменных БС P)

- если $v_i = (b)$, то $\xi_{i+1} = \xi_i$, и ребро из v_i в v_{i+1} имеет метку b^{ξ_i},

а если v_i – терминальная вершина, то выполнение БС на шаге i завершается.

2.3. Задача верификации блок-схем

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

- формула Pre называется **предусловием**, и выражает условие, которому должны удовлетворять значения переменных БС P в момент начала её выполнения, и

- формула $Post$ называется **постусловием**, и выражает условие, которому должны удовлетворять значения переменных БС P после завершения её выполнения.

Пусть P – БС, и $\xi \in X_P^\bullet$. Мы будем говорить, что P **выполняется с начальным означиванием** ξ, если в момент начала выполнения P значение каждой переменной $x \in X_P$ было равно x^ξ.

Запись $\xi \xrightarrow{P} \otimes$ обозначает утверждение: если P выполняется с начальным означиванием ξ, то выполнение P когда-либо завершится.

Если верно $\xi \xrightarrow{P} \otimes$, то запись ξP обозначает означивание из $X_P^\bullet$, определяемое следующим образом: пусть P выполняется с начальным означиванием ξ, тогда $\forall\, x \in X_P$ значение $x^{\xi P}$ 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:

$$\forall\, \xi \in X_P^\bullet \quad \text{если} \;\; Pre^\xi = 1, \;\; \text{то} \;\; \xi \xrightarrow{P} \otimes \;\; \text{и} \;\; Post^{\xi P} = 1. \tag{2}$$

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

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 $\pi = \alpha_1\alpha_2 \dots$ рёбер P, в которой каждое ребро α_i является тем ребром, по которому происходит перемещение на шаге i этого выполнения от текущей вершины v_i к вершине v_{i+1}. Обозначим записью $i_1 i_2 \dots$ последовательность номеров тех рёбер из π, на которых есть базовые точки. Согласно определению понятий полного базового множества и выполнения БС, $i_1 = 1$, и для каждой пары (i_j, i_{j+1}) соседних индексов в $i_1 i_2 \dots$ последовательность $\pi_j = \alpha_{i_j} \dots \alpha_{i_{j+1}}$ является базовым относительно M путём. Если путь π конечен, то его последнее ребро входит в терминальную вершину, и, следовательно, содержит точку из M. Таким образом, можно представить π в виде последовательности $\pi_1\pi_2 \dots$, где $\pi_1, \pi_2, \dots$ – базовые относительно M пути. Каждый член π_j этой последовательности мы будем называть **компонентой** выполнения $Exec$.

$\forall\, \pi \in \Pi_{P,M}, \forall\, \xi, \xi' \in X_P^\bullet$ запись $\xi \xrightarrow{\pi} \xi'$ означает, что π – компонента некоторого выполнения БС P, и ξ, ξ' – текущие означивания 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 для верификации блок-схем

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 $\varphi_m \in \mathcal{B}$, 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 означиваний $\xi, \xi' \in X_P^\bullet$, таких, что $\xi \xrightarrow{\pi} \xi'$, верна импликация

   $$\varphi^{\xi}_{start(\pi)} = 1 \quad \Rightarrow \quad \varphi^{\xi'}_{end(\pi)} = 1. \tag{3}$$

   Ниже, при анализе импликаций вида (3), $\forall\, x \in X_P$ будем обозначать значения x^ξ и $x^{\xi'}$ записями x и x' соответственно.

2) Свойство завершаемости P ($\forall\, \xi \in X_P^\bullet \;\; Pre^\xi = 1 \Rightarrow \xi \xrightarrow{P} \otimes$) 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 выражений $\{u_n \in \mathcal{E}(X_P) \mid n \in N\}$,

   таких, что

   - при каждом выполнении P, $\forall\, n \in N$ при каждом проходе через n текущее означивание ξ удовлетворяет условию $u_n^\xi \in L$, и
   - для любого пути $\pi \in \Pi_{P,N}$, и любых означиваний $\xi, \xi' \in X_P^\bullet$, таких, что $\xi \xrightarrow{\pi} \xi'$, верно неравенство $u^{\xi}_{start(\pi)} > u^{\xi'}_{end(\pi)}$.

Изложенный в этом пункте метод инвариантов был 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. $\forall\, \xi \in X_P^\bullet$ из $Pre^\xi = 1$ следует $\xi \xrightarrow{P} \otimes$ и $Post^{\xi P} = 1$.

Пусть $Exec$ – произвольное выполнение БС P с начальным означиванием ξ, удовлетворящим условию $Pre^\xi = 1$. 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 $n_1, n_2, \ldots$. Пути $\alpha_{j_k} \ldots \alpha_{j_{k+1}}$ $(k \geq 1)$ являются базовыми относительно N, поэтому, согласно пункту 2 параграфа 3.2, текущие означивания $\xi_{j_1}, \xi_{j_2}, \ldots$ удовлетворяют неравенствам

$$u_{n_1}^{\xi_{j_1}} > u_{n_2}^{\xi_{j_2}} > \ldots \tag{4}$$

т.е. в L есть бесконечная строго убывающая последовательность (4), 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 $\pi_1 \ldots \pi_k$ путей из $\Pi_{P,M}$.

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

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

Кроме того, $\forall\, i = 1, \ldots, k$ текущие означивания ξ_{i-1} и ξ_i вычисления $Exec$ до и после прохождения компоненты π_i последовательности $\pi_1 \ldots \pi_k$ соответственно удовлетворяют условию $\xi_{i-1} \xrightarrow{\pi_i} \xi_i$, поэтому, согласно (3),

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

Суммируя все вышесказанное, получаем цепочку импликаций:

$$(Pre^{\xi} = 1) \Rightarrow (\varphi^{\xi}_{m_0} = 1) \Rightarrow (\varphi^{\xi_1}_{m_1} = 1) \Rightarrow \ldots$$
$$\ldots \Rightarrow (\varphi^{\xi_k}_{m_k} = 1) \Rightarrow (Post^{\xi P} = 1). \blacksquare$$

3.4. Примеры фундированных множеств

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

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

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

  Обоснуем фундированность этого множества. Пусть существует бесконечная строго убывающая последовательность

  $$(a^1_1, \ldots, a^1_k) > (a^2_1, \ldots, a^2_k) > \ldots \tag{5}$$

  элементов множества L^k. Из (5) и из определения лексикографического порядка следует, что $a^1_1 \geq a^2_1 \geq \ldots$. Поскольку L фундировано, то в этой последовательности неравенств не может быть бесконечного количества строгих неравенств, т.е.

  $$\exists\, i_1 :\ a^{i_1}_1 = a^{i_1+1}_1 = \ldots. \tag{6}$$

  Из последнего соотношения и из (5) следует, что

  $$(a^{i_1}_2, \ldots, a^{i_1}_k) > (a^{i_1+1}_2, \ldots, a^{i_1+1}_k) > \ldots \tag{7}$$

  Применяя изложенные выше рассуждения к (7), получаем, что

  $$\exists\, i_2 \geq i_1 :\ a^{i_2}_2 = a^{i_2+1}_2 = \ldots, \tag{8}$$

  откуда на основании (7) следует, что

  $$(a^{i_2}_3, \ldots, a^{i_2}_k) > (a^{i_2+1}_3, \ldots, a^{i_2+1}_k) > \ldots.$$

  Продолжая так и дальше, в конце концов получим, что

  $$\exists\, i_k \geq i_{k-1} :\ a^{i_k}_k = a^{i_k+1}_k = \ldots. \tag{9}$$

  Из (6), (8), ..., (9) следует, что кортежи в (5), начиная с кортежа с номером i_k, совпадают, что противоречит предположению. $\blacksquare$

4. Процессные представления блок-схем

Для автоматизации верификации БС P методом инвариантов целесообразно сначала преобразовать эту БС в определяемый ниже граф G_P, который называется **процессным представлением** БС P. Для определения процессного представления БС необходимо ввести излагаемые в следующих пунктах вспомогательные понятия.

4.1. Действия и последовательности действий

Будем понимать под **действием** запись $x := e$ или $[\![b]\!]$, где $x \in \mathcal{X}$, $e \in \mathcal{E}$, $t(x) = t(e)$, $b \in \mathcal{B}$. Действие вида $x := e$ называется **присваиванием** переменной x значения выражения e, а действие вида $[\![b]\!]$ называется **проверкой условия**, выражаемого формулой b.

Действие вида $[\![b]\!]$, где формула b сама имеет вид $[\![b']\!]$ (т.е. является конъюнкцией), будет обозначаться без дублирующих квадратных скобок (т.е. записью $[\![b']\!]$).

Множество всех действий обозначается символом $\mathcal{A}$, множество всех конечных последовательностей действий обозначается записью $\mathcal{A}^*$. Если $a \in \mathcal{A}$ и $A, A' \in \mathcal{A}^*$, то записи aA, Aa и AA' обозначают конкатенации действия a и последовательности A, или A и A', соответственно.

$\forall\, a \in \mathcal{A}$ и $\forall\, A \in \mathcal{A}^*$ записи X_a и X_A обозначают множества переменных, содержащихся в a и A соответственно.

$\forall\, a \in \mathcal{A}$, $\forall\, X : X_a \subseteq X \subseteq \mathcal{X}$, $\forall\, \xi \in X^\bullet$ ξa обозначает объект, который

- равен означиванию $\xi(x := e)$ (см. (1)), если $a = (x := e)$,
- равен означиванию ξ, если $a = [\![b]\!]$ и $b^\xi = 1$
- не определен, если $a = [\![b]\!]$ и $b^\xi = 0$.

$\forall A \in \mathcal{A}^*$, $\forall X : X_A \subseteq X \subseteq \mathcal{X}$, $\forall \xi \in X^\bullet$ ξA обозначает объект, который

- совпадает с $(\ldots((\xi a_1)a_2)\ldots)a_k$, если $A = a_1 \ldots a_k$, и все объекты ξa_1, $(\xi a_1)a_2$, …, $(\ldots((\xi a_1)a_2)\ldots)a_k$ определены,
- не определен, в противном случае.

Последовательность из $\mathcal{A}^*$ называется **редуцируемой**, если она имеет вид $Aa[\![b]\!]A'$, где $A, A' \in \mathcal{A}^*$, $a \in \mathcal{A}$. **Редукцией** последовательности $Aa[\![b]\!]A'$ называется последовательность

- $A[\![b' \wedge b]\!]A'$, если $a = [\![b']\!]$,

- $A[\![(x := e)b]\!](x := e)A'$, если $a = (x := e)$
  (определение выражений вида $(x := e)e'$ см. в конце пункта 2.1).

Нетрудно доказать, что $\forall\, A \in \mathcal{A}^*$

- если A' – редукция A, то $\forall X : X_A \subseteq X \subseteq \mathcal{X}$, $\forall \xi \in X^\bullet$ $\xi A = \xi A'$ (т.е. ξA и $\xi A'$ либо оба не определены, либо определены и совпадают),

- $\exists\, A_1, \ldots, A_k\ (k \geq 1) : A_1 = A$, $\forall\, i = 1, \ldots, k-1$ A_{i+1} – редукция A_i, и A_k нередуцируема.

Последовательность A_k из предыдущего абзаца называется **нормальной формой** последовательности A, и обозначается записью $NF(A)$. Множество всех нередуцируемых последовательностей из $\mathcal{A}^*$ обозначается записью $\mathcal{A}^*_n$. Из определения понятия редукции следует, что каждая последовательность из $\mathcal{A}^*_n$ содержит не более одной проверки условия.

$\forall\, A \in \mathcal{A}^*_n$ запись $[\![A]\!]$ обозначает действие $[\![b]\!]$, если оно является первым в A, и действие $[\![1]\!]$, если A не содержит проверки условия.

$\forall A \in \mathcal{A}^*_n$, $\forall e \in \mathcal{E}$ запись Ae обозначает выражение

$$(x_1 := e_1) \ldots (x_k := e_k)e,$$

где $(x_1 := e_1) \ldots (x_k := e_k)$ – список всех присваиваний, входящих в A.

$\forall A \in \mathcal{A}^*_n, \forall\, \varphi, \psi \in \mathcal{B}$ запись $\varphi \overset{A}{\to} \psi$ обозначает формулу

$$\varphi \wedge [\![A]\!] \to A\psi,$$

которая выражает утверждение: $\forall\, \xi \in X^\bullet$, где $X_\varphi \cup X_A \cup X_\psi \subseteq X \subseteq \mathcal{X}$, если $\varphi^\xi = 1$ и ξA определено, то $\psi^{\xi A} = 1$.

4.2. Понятие процессного графа

Процессным графом (ПГ) называется конечный граф G со следующими свойствами:

- граф G имеет выделенные вершины G^0 и $G^\otimes$, называемые **начальной** и **терминальной** вершинами соответственно, при изображении ПГ в виде рисунка данные вершины обозначаются символами $\odot$ и $\otimes$ соответственно,

- каждому ребру α графа G сопоставлена метка $A_\alpha \in \mathcal{A}^*_n$,

- $G^{\otimes}$ – единственная вершина G, из которой не выходят рёбра.

Пусть G – ПГ. Будем обозначать записями V_G и X_G совокупность всех вершин G и переменных, входящих в G, соответственно. Будем предполагать, что в каждом ПГ 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 ПГ данное присваивание будет опускаться.

Выполнение ПГ G – это обход вершин G, начиная с G^0, с выполнением действий, сопоставленных проходимым рёбрам. С каждым шагом $i \geq 0$ выполнения G связаны вершина v_i этого графа и означивание $\xi_i \in X_G^{\bullet}$ (называемые **текущей вершиной** и **текущим означиванием** на шаге i). Если $v_i = G^{\otimes}$, то выполнение G на шаге i завершается, иначе выполнение на шаге i заключается в

- выборе произвольного ребра α, которое выходит из v_i и удовлетворяет условию $[\![A_\alpha]\!]^{\xi_i} = 1$,
- замене текущего означивания ξ_i на означивание $\xi_{i+1} = \xi_i A_\alpha$, и
- переходе в конец выбранного ребра, который будет текущей вершиной v_{i+1} на $i+1$–м шаге выполнения.

4.3. Построение процессного представления блок-схем

Пусть задана БС P. Алгоритм построения ПГ G_P, называемого **процессным представлением** БС P, состоит из следующих шагов.

1) Удаляется начальная вершина P и выходящее из неё ребро, конец этого ребра – начальная вершина G_P^0 графа G_P.

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

3) Каждая оставшаяся вершина v БС P является вершиной ПГ G_P, и

   - если v имеет вид $[x := e]$, и P содержит ребро $v \to v'$, то G_P содержит ребро $v \overset{x:=e}{\to} v'$
   - если v имеет вид (b), и P содержит рёбра $v \overset{1}{\to} v'$ и $v \overset{0}{\to} v''$, то G_P содержит рёбра $v \overset{[\![b]\!]}{\to} v'$ и $v \overset{[\![\neg b]\!]}{\to} v''$.

Получившийся ПГ G_P может быть редуцирован путем применения операции **удаления вершины**: если v – вершина ПГ G_P, отличная от G_P^0 и $G_P^\otimes$, в G_P нет ребер вида $v \xrightarrow{A} v$, и списки ребер, входящих в v и выходящих из v, имеют вид

$$v_1 \xrightarrow{A_1} v, \ldots, v_k \xrightarrow{A_k} v \quad \text{и} \quad v \xrightarrow{A'_1} v'_1, \ldots, v \xrightarrow{A'_{k'}} v'_{k'} \tag{10}$$

соответственно, то операция удаления v из G_P заключается в удалении этой вершины и замене всех ребер из (10) на ребра вида $v_i \xrightarrow{A_{ij}} v'_j$, где $i \in \{1, \ldots, k\}$, $j \in \{1, \ldots, k'\}$, и $A_{ij} = NF(A_i A'_j)$.

Результат применения этой операции обозначается той же записью G_P и рассматривается как ПГ, эквивалентный в некотором смысле исходному ПГ. Данная операция может быть применена несколько раз.

4.4. Верификация блок-схем с использованием процессного представления

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

Множество вершин M ПГ G называется **базовым**, если для каждого цикла в G хотя бы одна из его вершин лежит в M. Если, кроме того, M содержит G^0 и $G^\otimes$, то M называется **полным** базовым множеством. Путь π в G называется **базовым** относительно M, если он непуст, $start(\pi)$ и $end(\pi)$ лежат в M, а остальные вершины π не лежат в M. Нетрудно доказать, что множество $\Pi_{G,M}$ всех базовых относительно M путей в G конечно.

Теорема.

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

- существует полное базовое множество M вершин G_P, причем с каждой вершиной $m \in M$ связана формула $\varphi_m \in \mathcal{B}$, и
  - $\varphi_{G_P^0} = Pre$, $\varphi_{G_P^\otimes} = Post$,
  - $\forall\, \pi \in \Pi_{G_P,M} \;\; (\varphi_{start(\pi)} \xrightarrow{A_\pi} \varphi_{end(\pi)}) = 1$,

- существуют базовое множество N вершин G_P и фундированное множество L, такие, что с каждой вершиной $n \in N$ связано выражение $u_n \in \mathcal{E}(X_P)$, причем выполнены условия:

  – если при каком-либо выполнении G_P текущей вершиной в какой-либо момент является вершина $n \in N$, то текущее означивание ξ в этот момент удовлетворяет условию $u_n^{\xi} \in L$,

  – $\forall\, \pi \in \Pi_{G_P,N} \quad \Big([\![A_\pi]\!] \to \big(u_{start(\pi)} > A_\pi u_{end(\pi)} \big) \Big) = 1.$

Для доказательства теоремы заметим, что 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 $Pre \xrightarrow{P} Post$ на основе метода инвариантов. Действительно, если вершина m графа 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 формулу φ_m. Аналогично определяется N_P. В качестве L_P можно взять L. Проверка всех условий, упоминаемых в формулировке метода инвариантов, является несложной. ■

5. Заключение

В настоящем тексте изложено описание метода инвариантов для верификации различных классов программ, и приведено несколько иллюстраций применения этого метода.

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). 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 ему некоторые шаблоны инвариантов, которые он бы уже самостоятельно конкретизировал до формул, обладающих необходимыми свойствами. Некоторые продвижения в решении данной проблемы можно найти в работах [12]–[24].

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, и т.п.

Список литературы

[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] Мендельсон Э. Введение в математическую логику. - М.: Наука, 1984.

[6] Андерсон Р. Доказательство правильности программ. - М.: Мир, 1982.

[7] Абрамов С. А. Элементы анализа программ. Частичные функции на множестве состояний. - М.: Наука, 1986.

[8] Непомнящий В. А., Рякин О. М. Прикладные методы верификации программ. - М.: Радио и связь, 1988.

[9] Francez N. Verication 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., and Manna Z. Automatic generation of invariants and intermediate assertions. Theoretical Computer Science, Volume 173, Issue 1, 1997, P. 49–87.

[13] S. Bensalem, Y. Lakhnech, H. Saidi. 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] R. Chadha, D.A. Plaisted. On the mechanical derivation of loop invariants. J. Symbol. Comput., 15 (5) (1993), pp. 705-744

[15] P. Cousot, R. Cousot. 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] P. Cousot, N. Halbwachs. Automatic discovery of linear restraints among the variables of a program. 5th ACM Symp. Principles of Programming Languages (1978), pp. 84-97

[17] D. Dill, H. Wong-Toi. 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] S.M. German, B. Wegbreit. A synthesizer of inductive assertions. IEEE Trans. Software Eng., 1 (1975), pp. 68-75.

[19] N. Halbwachs, P. Raymond, Y.-E. Proy. 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] S. Bensalem, M. Bozga, J.-C. Fernandez, L. Ghirvu, and Y. Lakhnech. A transformational approach for generating non-linear invariants. In SAS, 2000, pp. 58–74.

[21] S. Sankaranarayanan, H. B. Sipma, and Z. Manna. Non-linear loop invariant generation using grobner bases. 2004.

[22] A. Tiwari, H. Rueb, H. Saidi, and N. Shankar. 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] M. D. Ernst, J. H. Perkins, P. J. Guo, S. McCamant, C. Pacheco, M. S. Tschantz, and C. Xiao. The Daikon system for dynamic detection of likely invariants. Science of Computer Programming, vol. 69, no. 1–3, pp. 35–45, Dec. 2007.

[24] A. Gupta and A. Rybalchenko. Invgen: An efficient invariant generator. In Computer Aided Verification. Springer, 2009, pp. 634–640.

Verification of programs by the method of invariants
Mironov A.M.

We present a method of invariants for proving the correctness of computer programs. The basic concepts associated with this method are illustrated by examples of verification of sequential and parallel programs.

Keywords: program verification, Floyd's method, invariants