КИБЕРНЕТИКА

С.Л. КРЫВЫЙ, А.Н. МАКСИМЕЦ

УДК 51.681.3

ВЕРИФИКАЦИЯ ПРОГРАММ: СОСТОЯНИЕ, ПРОБЛЕМЫ, РЕЗУЛЬТАТЫ. I

Ключевые слова: *верификация, абстрактные интерпретации, транзиционные системы, сети Петри, верификация на моделях.*

Данная работа является продолжением обзоров, начатых в [1, 2], где рассматривались методы верификации программ процедурного типа с ориентацией на использование программных динамических логик (в частности, логики Хоара). В первой части настоящей публикации описывается метод абстрактных интерпретаций анализа семантических свойств последовательных программ, во второй — методы и их приложения, ориентированные на верификацию реактивных и распределенных систем. Выбор этих направлений объясняется тем, что в настоящее время в области верификации активно развиваются и применяются на практике именно эти направления.

Семантический анализ программ заключается в разработке анализатора программ. Под анализатором понимается программа, которая, получая на свой вход в качестве входных данных программу (возможно, аннотированную), выдает на выходе ответы на вопросы о ее свойствах, которые имеют место на всем протяжении выполнения этой программы. Из-за алгоритмической неразрешимости проблемы верификации или из-за огромной сложности процесса верификации эти ответы неизбежно будут частичными, но и они всегда очень важны. Один из подходов к проблеме верификации — метод абстрактных интерпретаций, основные положения которого описаны в данной работе.

1. АБСТРАКТНЫЕ ИНТЕРПРЕТАЦИИ

Абстрактные интерпретации впервые были введены М. Синтцовым [3] для автоматизации процесса анализа программ, затем были развиты в работах [4–6] и [7, 8]. Главная идея абстрактных интерпретаций состоит в нахождении таких ограничений абстрактного характера на семантические структуры программ, чтобы можно было применять средства автоматического доказательства утверждений в логических языках и средства компьютерной алгебры. В этом смысле метод абстрактных интерпретаций относят к таким методам, как анализ потоков данных в программах, смешанные вычисления, трансформации программ в процессе генерации кода [9].

Свойства и их абстракции. Пусть Q — некоторое множество объектов (например, состояний программы, выполняемых трасс и т.п.), P — множество объектов, имеющих определенное свойство, т.е. $P \in B(Q)$, где $B(Q)$ — булеан множества Q. Поскольку свойства определяют подмножества из $B(Q)$, то они образуют полную булевую решетку $G=(B(Q), \subseteq, \varnothing, Q, \bigcup, \bigcap, ')$ относительно теоретико-множественных операций, элементы которой частично упорядочены отношением включения ($\subseteq$) для множеств.

Напомним, что полной решеткой называется алгебра $R=(\mathcal{A}, \{\sqsubseteq, \perp, \top, \sqcup, \sqcap\})$, где $\mathcal{A}$ — частично упорядоченное множество отношением частичного порядка ($\sqsubseteq$), а операции называются операциями взятия верхней грани ($\sqcup$) и нижней грани ($\sqcap$) некоторой совокупности элементов из $\mathcal{A}$ [10]. Элементы $\perp$ и $\top$ соответствуют нулю и единице полной решетки. Это означает, что $\forall a \in \mathcal{A}(\perp \sqsubseteq a) \wedge$ $\wedge\,(a \sqsubseteq \top)$ и для любого подмножества $H \subseteq \mathcal{A}$ существует единственный элемент $a \in \mathcal{A}$ такой, что $a=\bigsqcup_{b \in H} b$ (для операции $\sqcap$ такой элемент определяется двойственным образом).

Далее будем полагать, что имеется полная решетка $(\mathcal{A}, \{\sqsubseteq, \perp, \top, \sqcup, \sqcap\})$, в терминах которой будут определяться все необходимые понятия.

Неформально под абстракцией понимается вывод или (механическое) вычисление на объектах, таких что только некоторые свойства этих объектов могут использоваться. Пусть A — множество конкретных свойств, $\overline{A} \subseteq \mathcal{A}$ — множество абстрактных свойств, которые могут использоваться в выводах или вычислениях. Таким образом, абстракция состоит в аппроксимировании конкретных свойств с помощью их абстрактных свойств.

Существует два возможных направления аппроксимации: аппроксимация сверху и аппроксимация снизу. Аппроксимация сверху множества свойств $P \in A$ с помощью множества абстрактных свойств $\overline{P} \in \mathcal{A}$ влечет $P \sqsubseteq \overline{P}$. В аппроксимации снизу множества свойств $P \in A$ с помощью множества абстрактных свойств $\underline{P} \in \mathcal{A}$ получаем $\underline{P} \sqsubseteq P$. Очевидно, что аппроксимации сверху и снизу двойственные одна другой. Это значит, что аппроксимация сверху/снизу для P является аппроксимацией снизу/сверху для $\neg P$, а отсюда вытекает, что можно рассматривать только одну из них, например аппроксимацию сверху.

Далее, элемент $\top$, являясь элементом $\overline{\mathcal{A}}$, обозначает случай, когда некоторые свойства не имеют абстракции (например, когда $\mathcal{A}=\varnothing$). Следовательно, любое конкретное свойство $P \in \mathcal{A}$ всегда может быть аппроксимировано сверху (с помощью $\top$, т.е. Q истинно или нет). Для более точной аппроксимации используется наименьшая абстракция $\overline{P} \in \overline{\mathcal{A}}$ такая, что $P \sqsubseteq \overline{P}$ и не существует $\overline{P}' \in \mathcal{A} : \overline{P}' \sqsubseteq \overline{P}$.

Имеется несколько различных аппроксимаций для различных свойств. Во избежание рассмотрения всех возможных аппроксимаций необходимо, чтобы любое конкретное свойство $P \in A$ имело наилучшую абстракцию $\overline{P} \in \overline{\mathcal{P}}$, т.е. если $P \sqsubseteq \overline{P}$, то $\forall \overline{P}' \in \overline{\mathcal{A}}$ должно выполняться $P \sqsubseteq \overline{P}' \rightarrow \overline{P} \sqsubseteq \overline{P}'$. Из определения операции $\sqcap$ следует, что требование наилучшей абстракции эквивалентно пересечению абстрактных свойств $\overline{P}=\sqcap\,\{\overline{P}' \in \overline{\mathcal{A}} : P \sqsubseteq \overline{P}'\} \in \overline{\mathcal{A}}$. Другая точка зрения описана в работах [11, 12].

Проиллюстрируем сказанное примером из работы [9].

Пример 1. Рассмотрим арифметическое выражение $(8+5)$. Следует определить, будет результат четным или нечетным числом. Это можно выяснить с помощью конкретного вычисления числа 13, которое нечетно, а можно также абстрагироваться от конкретных значений 8 и 5 путем введения свойства четности: *even* (четное) и *odd* (нечетное), и далее рассматривать задачу сложения $even + odd$. Это вычисление абстракции дает результат *odd*.

Ясно, что можно было бы рассматривать другие абстракции, получая различную информацию о выражениях, например, какой будет знак в результате вычисления выражения? Тогда рассматривалась бы абстракция $positive + positive = positive$.

Операция +	*even*	*odd*
even	*even*	*odd*
odd	*odd*	*even*

Операция ×	*even*	*odd*
even	*even*	*even*
odd	*even*	*odd*

Рис. 1

Главным здесь является то, что мы имеем абстракции понятия значения (5) и операции на них ($+$). Обобщая эти абстракции, запишем абстрактные операции сложения и умножения на множестве $\{even, odd\}$ следующим образом (рис. 1).

Обозначим эту абстракцию ПН и рассмотрим условное выражение *if u then* 3 *else* 4. Если значение u неизвестно, то и результат этого выражения неизвестен. Для получения абстракции таких выражений и вводится новое абстрактное значение $\top \in$ ПН для обозначения отсутствия знания о значении. Оно удовлетворяет условиям $even \sqsubseteq \top$ и $odd \sqsubseteq \top$. Отсюда следует, что если $s \sqsubseteq t$, т.е. s меньше или равно t, то s более информативно, чем t.

Пусть s, t — два абстрактных значения. Тогда $s \sqcup t$ — наименьшая верхняя грань элементов s и t в нашей решетке, означающая наименьший элемент, который больше или равен s и t. Например, для приведенного выше условного выражения $even \sqcup odd = \top$ будет абстрактным значением этого условного выражения.

Рассмотрим теперь такие две функции:

$$f(x) = x + 1 \text{ и } g(x) = \text{if } x > 5 \text{ then } x \text{ else } 8 + g(5x).$$

Для функции $f(x)$ имеем абстрактную интерпретацию $f^{a(X)} = X + odd$, а для функции $g(x)$ построение абстракции несколько сложнее, так как ее абстракция g^a должна ответить на вопрос, имеет ли решение рекурсивное определение:

$$g^{a(X)} = X \sqcup (even + g(odd \cdot X)).$$

Ответ будет *yes*, если ввести новое значение $\perp$ для обозначения «нет значения» или «не определено». Таким образом, $\perp \in$ ПН и $\perp \sqsubseteq s$ для всех абстрактных значений $s \in$ ПН. В результате получаем решетку $G = (\{\perp, even, odd, \top\}, \{\sqsubseteq, \perp, \top, \sqcup, \sqcap\})$. При этом диаграмма Хассе имеет следующий вид (рис. 2).

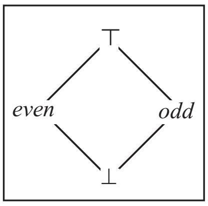

Рис. 2

Запишем абстрактные операции сложения и умножения (рис. 3).

Операция +	$\perp$	*even*	*odd*	$\top$
$\perp$	$\perp$	$\perp$	$\perp$	$\perp$
even	$\perp$	*even*	*odd*	$\top$
odd	$\perp$	*odd*	*even*	$\top$
$\top$	$\perp$	$\top$	$\top$	$\top$

Операция ×	$\perp$	*even*	*odd*	$\top$
$\perp$	$\perp$	$\perp$	$\perp$	$\perp$
even	$\perp$	*even*	*even*	*even*
odd	$\perp$	*even*	*odd*	$\top$
$\top$	$\perp$	*even*	$\top$	$\top$

Рис. 3

Из-за условных выражений и рекурсивных определений функций приходится вводить два абстрактных значения ($\top$ и $\perp$) для обозначения «не определено» и «нет информации».

Подводя первые итоги, отметим, что программы могут быть интерпретированы конкретно или абстрактно. Изучая свойства абстрактной интерпретации программы, мы изучаем свойства самой программы.

Абстрактные и конкретные отображения. Уточним введенные в примере 1 понятия абстракции и конкретизации функции. Оба типа этих отображений могут быть формализованы с помощью функции абстракции α и функции конкретизации γ. Пусть $N = \{0, 1, 2, \ldots\}$ — множество натуральных чисел, $B(N)$ — его булеан, частично упорядоченный отношением включения ($\subseteq$).

Функция конкретизации $\gamma : \text{ПН} \to B(N)$ отображает абстрактное значение s в множество $\gamma(s)$ конкретных чисел, представленных значением s. Функция абстракции $\alpha : B(N) \to$ ПН, наоборот, отображает множество чисел V в (наименьшее) абстрактное значение $s(V)$, представляющее все числа этого множества.

От функций конкретизации и абстракции требуется выполнение таких условий:

1) обе функции монотонны;

2) $\forall s \in V \ \alpha(\gamma(s)) = s$;

3) $\forall V \in B(A) \ \gamma(\alpha(V)) \supseteq V$.

Условие монотонности означает, что большая абстракция представляет большее множество конкретных значений. Из условие 2) следует, что каждое абстрактное значение представляет само себя и не допускает путаницы среди абстрактных значений. Условие 3) означает, что абстрагирование и конкретизирование множества V конкретных значений дает множество, содержащее множество V.

Выполнение всех этих условий вместе гарантирует, что абстракция множества V сохраняет представление всех элементов из V.

Пример 2. В решетке ПН из примера 1 функция конкретизации имеет вид:

$\gamma(\perp) = \varnothing$ — пустое множество;

$\gamma(even) = \{0, 2, 4, \ldots\}$ — множество четных чисел;

$\gamma(odd) = \{1, 2, 5, \ldots\}$ — множество нечетных чисел;

$\gamma(\top) = N$ — множество натуральных чисел,

а функция абстракции будет такой:

$$\alpha(\varnothing) = \perp; \ \alpha(\{0, 2, 4, \ldots\}) = even; \ \alpha(\{1, 3, 5, \ldots\}) = odd; \ \alpha(N) = \top.$$

Используя функции α и γ, можно определить понятие абстрактной функции $f^{a} : A \to \mathcal{A}$. Например, пусть $\mathcal{A} = \text{ПН}$ и $f : N \to N$ — конкретная функция, $n \in N$ и $\alpha(\{n\})$ — его абстрактное значение. Тогда от функции f^{a} требуется, чтобы конкретное значение $f(n)$ этой функцией представлялось как абстрактное значение $f^{a}(\alpha\{(n)\})$, т.е. $\forall n \in N \ (f(n) \in \gamma(f^{a}(\alpha(\{n\}))))$.

Абстрактная операционная семантика программ. Напомним основные понятия λ-исчисления и сделаем некоторые замечания об абстрактном синтаксисе программы. λ-исчисление представляет собой исчисление неименованных функций, оно дает синтаксическое описание функций и синтаксические правила их преобразования. Так, в λ-выражении $\lambda x.[f(x)]$, символ λ читается как «функция от», а точка — как «которая возвращает». Например, $\lambda x.[*(2, x)]$ означает функцию, которая возвращает $2x$. При этом символ x называется связанной переменной λ-абстракции, а выражение справа от точки — телом λ-абстракции. Тело λ-абстракции описывает то, что нужно сделать с параметрами, поступившими на вход функции. Тело λ-абстракции может содержать и другую λ-абстракцию, например $\lambda x.[\lambda y.[*(+(x, y), 2)]]$. Это означает, что функция от x возвращает функцию от y, которая возвращает $2(x + y)$ и является λ-выражением.

Вычисление λ-выражений осуществляется с помощью правил вывода λ-исчисления. Простейший тип λ-выражений — это константы, которые считаются самоопределенными, т.е. их нельзя преобразовать к более простым λ-выражениям. Вычисление константы 3 дает ту же константу 3. Преобразование λ-выражения $+(1, 3)$ в константу 4 выполняется с помощью встроенных δ-правил, например

$$*(+(1,2))(-(4,1)) \xrightarrow{\delta} *(+(1,2),3) \xrightarrow{\delta} *(3,3) \xrightarrow{\delta} 9.$$

Вторая группа правил называется β-правилами. Рассмотрим пример λ-выражения $\lambda x.[+(x,x)]2$. Здесь ситуация аналогична той, которая возникает при вызове процедуры или функции с фактическим параметром, заменяющим ее формальный параметр. Следовательно, сначала нужно заменить формальный параметр x фактическим параметром 2. В данном λ-выражении получаем:

$$\lambda x.[*(x,x)]2 \xrightarrow{\delta} *(2,2) \xrightarrow{\delta} 4.$$

Редукция λ-абстракции, примененная к некоторому аргументу, может дать другую λ-абстракцию, и в этом случае процесс может быть продолжен. Например, вычисление λ-выражения $\lambda x.[\lambda y.[+(x, y)]7]8$ можно начать с подстановки числа 7 вместо x в тело внешней абстракции, т.е. $\lambda y.[+(x, y)]$. В результате получаем $\lambda y.[+(7, y)]8$, и заканчивая применение, получаем выражение $+(7, 8)$, что дает окончательное значение 15.

Представление программ. Представим программу в виде размеченного орграфа с одной начальной вершиной и одной заключительной вершиной. Дуги (ориентированные ребра) графа помечены операторами языка программирования.

Графом программы называется четверка $G_p = (V, a_0, a^*, E)$, где V — конечное множество вершин, $E \subseteq V \times V$ — конечное множество дуг, $a_0 \in V$ — начальная вершина, $a^* \in V$ — заключительная вершина, причем $a_0 \neq a^*$. Если $(a, b) \in E$, то считают, что дуга (a, b) выходит из вершины a и входит в вершину b. Начальная вершина характеризуется тем, что в нее не входит ни одна дуга, а из заключительной вершины не выходит ни одна дуга, а все другие вершины из V находятся на некотором пути из a_0 в a^*.

Пусть v — вектор, координатами которого являются пары (v_i, m_i), где v_i — переменная, а m_i — ее значение, которое берется из некоторого универсального множества U. Множество операторов $Q(U)$ делится на два подмножества:

- подмножество $O_s(U)$ операторов присваивания;
- подмножество $O_t(U)$ операторов проверки условий.

Оператор присваивания $v := y(v)$ представляет собой частичное отображение из множества U в U, а проверка условий — частичное отображение из U в $B = \{true, false\}$.

Программой называется тройка $\mathcal{P} = (G, U, L)$, где G — граф программы, U — универсальное множество и $L : E \to O(U)$ — функция отметок дуг графа G. Вершины этого графа будут также называться состояниями программы. Функция L такая, что для каждой вершины $a \in V$, отличной от a^*, либо из a выходит единственная дуга, отмеченная некоторым оператором присваивания y, либо две дуги, отмеченные условиями u и $\neg u$ соответственно. В работах [1, 2] такая модель программы названа $U-Y$-схемой программы.

Операционная семантика. Операционная семантика синтаксически правильной программы p сопоставляет последовательность достижимых состояний при вычислениях, определяемых этой программой.

Множество состояний S состоит из пар (a, m), где $a \in V \bigcup \{\xi\}$ — состояние программы, $m \in U$ — состояние памяти, а $\xi \notin V$ — вершина, называемая состоянием ошибки. Состояния a_0, a^*, ξ характеризуются такими λ-выражениями:

$$v_{a_0} = \lambda(c, m) . (c = a_0), \quad v_{a^*} = \lambda(c, m) . (c = a^*), \; v_{\xi} = \lambda(c, m) . (c = \xi) .$$

Программа $\pi = (G, U, L)$ определяет функцию переходов $\bar{\tau} : S \to S$ следующим образом:

1) $\bar{\tau}((\xi, m)) = (\xi, m)$;

2) $\bar{\tau}((a^*, m)) = (a^*, m)$;

3) если $(a_1, m) \in S$, где $a_1 \in V$ имеет одну выходную дугу $(a_1, a_2) \in E$, отмеченную оператором присваивания $y \in O_s(U)$, то при $m \in \mathrm{dom}\,(y)$ имеем $\bar{\tau}((a_1, m)) = (a_2, y(m))$, иначе $\bar{\tau}((a_1, m)) = (\xi, m)$;

4) если $(a_1, m) \in S$, где $a_1 \in V$, имеет две выходные дуги: (a_1, a_2), $(a_1, a_3) \in E$, отмеченные u и $\neg\, u$, $u \in O_t(U)$ соответственно, то $m \notin \mathrm{dom}\,(u)$ при $\bar{\tau}((a_1, m)) = (\xi, m)$, иначе если $u(m)$, то $\bar{\tau}((a_1, m)) = (a_2, m)$, иначе $\bar{\tau}((a_1, m)) = (a_3, m)$.

Отношение переходов $\tau \subseteq S \times S$, определяемое программой π, имеет вид

$$\lambda(a_1, a_2) . (a_2 = \bar{\tau}(a_1).$$

Напомним определение рефлексивного и транзитивного замыканий бинарного отношения. Для любого натурального числа $n > 1$ степень бинарного отношения $\alpha \subseteq S \times S$ определяется рекурсивно: $\alpha^0 = i_S$, $\alpha^{n+1} = \alpha * \alpha^n$, где i_S — тождественное отношение на S (диагональ множества S), а $*$ — операция умножения бинарных отношений. Рефлексивным и транзитивным замыканием бинарного отношения α называется объединение $\alpha^* = i_S \bigcup \alpha \bigcup \alpha^2 \bigcup \ldots \bigcup \alpha^n \bigcup \ldots$

Выполнение синтаксически правильной программы π, начинающееся в начальном состоянии $(a_0, m) \in S$, называется ведущим к состоянию ошибки тогда и только тогда, когда $\exists\, a_1 \in S \; : \tau^*(a_0, a_1) \wedge v_{\xi}(a_1)$, и терминальным тогда и только тогда, когда $\exists\, a_1 \in S \; : \tau^*(a_0, a_1) \wedge v_{a^*}(a_1)$. В противном случае говорят, что вычисление программы π расходится. Результат выполнения синтаксически правильной программы π, начинающегося в начальном состоянии (a_0, m), определен тогда и только тогда, когда это выполнение заканчивается со значением $m' \in U$ таким, что $\tau^*((a_0, m), (a^*, m'))$ и m' — этот результат.

Дискретная динамическая система (ДДС) используется в качестве модели для исследования семантических свойств программ.

ДДС называется пятерка $(S, \tau, v_{a_0}, v_{a^*}, \xi)$ такая, что S — непустое множество состояний, $\tau \subseteq S \times S$ — отношение переходов, $v_{a_0} : S \to B$ характеризует начальное состояние, $v_{a^*} : S \to B$ — заключительное состояние, а $v_{\xi} : S \to B$ — состояние ошибки, причем все состояния a_0, a^*, ξ различны.

ДДС называется тотальной, если $\forall c \in S\; \exists\, c_1 \in S(\tau(c, c_1))$, и детерминированной, если $\forall c, c_1, c_2 \in S(\tau(c, c_1) \wedge \tau(c, c_2)) \to (c_1 = c_2)$.

Программа π, определенная выше, задает некоторую ДДС. Более того, выполняются следующие свойства:

$$\forall\, c, c_1 \in S \;\; \tau(c, c_1) \to \neg\, (v_{a_0}(c_1)),$$

$$\forall c, c_1 \in S \;\; \tau(c, c_1) \wedge v_{a^*}(c) \to c = c_1,$$

$$\forall\, c, c_1 \in S \;\; \tau(c, c_1) \wedge v_{\xi}(c) \to v_{\xi}(c_1) .$$

ДДС называется инъективной, если τ^{-1} детерминировано, и инвертируемой, если она инъективна и τ^{-1} — тотальное отношение (в общем случае программа не всегда определяет инъективную ДДС).

2. ХАРАКТЕРИСТИКА ДДС С ПОЗИЦИЙ НЕПОДВИЖНЫХ ТОЧЕК

Теоремы о неподвижной точке отображений в полных решетках. Приведем некоторые определения и понятия из теории полных решеток. Пусть $R = (\mathcal{A}, \{\sqsubseteq, \perp, \top, \sqcap, \sqcup\})$ — полная решетка и $f : R \to R$ — отображение. Отображение f называется строгим, если $f(\perp) = \perp$, и изотонным, если $\forall\, a, b \in G[(a \sqsubseteq b) \to (f(a) \sqsubseteq f(b)]$.

Элемент $a \in R$ называется неподвижной точкой отображения $f : R \to R$, если $f(a) = a$. Поскольку полная решетка является частично упорядоченным множеством, то неподвижная точка может быть не единственной, и тогда естественно ввести понятие наименьшей (lfp) и двойственное понятие наибольшей неподвижной точки (gfp). Исходя из определения операции $\sqcap$, можем записать

$$lfp(f) = \sqcap\ \{x \in R : f(x) \sqsubseteq x\} \ \text{ и } \ gfp(f) = \sqcup\ \{x \in R\ : x \sqsubseteq f(x)\}.$$

Имеют место следующие утверждения [10].

Теорема 1 (о неподвижной точке). Если $R = (\mathcal{A}, \{\sqsubseteq, \perp, \top, \sqcap, \sqcup\})$ — полная решетка и $f : R \to R$ — изотонное отображение, то существует элемент $a \in R$ такой, что $f(a) = a$.

Теорема 2 (Тарского). Множество неподвижных точек изотонного отображения полной решетки $R = (\mathcal{A}, \{\sqsubseteq, \perp, \top, \sqcap, \sqcup\})$ является полной решеткой относительно того же порядка $\sqsubseteq$.

Из теоремы Тарского вытекает следующее утверждение.

Теорема 3 (о рекурсивном принципе индукции). Если $R = (\mathcal{A}, \{\sqsubseteq, \perp, \top, \sqcap, \sqcup\})$ — полная решетка, то $\forall x \in R\ ((f(x) \sqsubseteq x) \to (lfp(f) \sqsubseteq x))$ и двойственно $\forall x \in R$ $((x \sqsubseteq f(x)) \to (x \sqsubseteq gfp(f))$.

Если $R = (\mathcal{A}, \{\sqsubseteq, \perp, \top, \sqcap, \sqcup\})$ и $R_1 = (\mathcal{B}, \{\sqsubseteq', \perp', \top', \sqcap', \sqcup'\})$ — полные решетки, то множество F отображений из R в R_1 является полной решеткой $\overline{R} = (F, \{\sqsubseteq', \perp', \top', \sqcap', \sqcup'\})$ относительно отношения $\forall f, g \in F\ \ (f \sqsubseteq' g)$ тогда и только тогда, когда $\forall x \in R\ \ (f(x) \sqsubseteq' g(x))$.

Множество n-ок элементов полной решетки R образуют снова полную решетку относительно отношения покомпонентного сравнения, т.е. $(a_1, a_2, \ldots, a_n) \sqsubseteq (b_1, b_2, \ldots, b_n)$ тогда и только тогда, когда $a_i \sqsubseteq b_i$ для $i = 1, 2, \ldots, n$. Булеан $B(R)$ полной решетки R снова будет полной решеткой $(B(R), \subseteq, \varnothing, R, \bigcap, \bigcap, ')$. Отображение $f : R \to R_1$ расширяется на решетки R^n и R_1^n очевидным образом: $f((a_1, \ldots, a_n)) = (f(a_1), \ldots, f(a_n))$ и на булеаны $f : B(R) \to B(R_1)$, где $f(S) = \{f(x) : x \in S\}$, $S \in B(R)$, $f(S) \in B(R_1)$.

Последовательность элементов $x_0, x_1, \ldots, x_n, \ldots$ частично упорядоченного множества $(R, \sqsubseteq)$ называется возрастающей цепью, если $x_0 \sqsubseteq x_1 \sqsubseteq \ldots \sqsubseteq x_n \sqsubseteq \ldots$ Функция $f : R \to R$ на решетке $R = (\mathcal{A}, \{\sqsubseteq, \perp, \top, \sqcap, \sqcup\})$ называется полу-$\sqcup$-непрерывной тогда и только тогда, когда для любой цепи $C = \{x_i \mid i \in I \subseteq R\}$ выполняется равенство $f(\sqcup C) = \sqcup\ f(C)$. Имеет место следующая теорема.

Теорема 4 (Клини). Наименьшая фиксированная точка полу-$\sqcup$-непрерывной функции f на $R = (\mathcal{A}, \{\sqsubseteq, \perp, \top, \sqcap, \sqcup\})$ совпадает со значением $\sqcup f^{\,i}$, где $f^{\,i}$ определяется рекуррентной зависимостью $f^{\,0} = \lambda x\,.[x]$, $f^{\,i+1} = \lambda x\,.[f(f^{\,i}(x))]$.

Частично упорядоченное множество называется множеством, удовлетворяющим условию обрыва возрастающих цепей, если любая возрастающая цепь конечна, т.е. существует такой индекс m, что

$$x_0 \sqsubseteq x_1 \sqsubseteq x_2 \sqsubseteq \ldots \sqsubseteq x_m = x_{m+1} = x_{m+2} = \ldots$$

Очевидно, что полу-$\sqcup$-непрерывная функция будет изотонным отображением, но обратное в общем случае неверно. Однако если f — изотонное отображение полной решетки в себя, удовлетворяющей условию обрыва возрастающих цепей, то f будет полу-$\sqcup$-непрерывной функцией. В этом случае такая функция будет полным-$\sqcup$-морфизмом, т.е. $\forall H \subseteq R\ f(\sqcup H) = \sqcup f(H)$, а отсюда очевидным образом следует полу-$\sqcup$-непрерывность f.

Двойственным образом определяются убывающие цепи, условие обрыва убывающих цепей, полу-$\sqcap$-непрерывность и полный $\sqcap$-морфизм.

Если $R = (\mathcal{A}, \{\sqsubseteq, \bot, \top, \sqcap, \sqcup\})$ и $R_1 = (\mathcal{B}, \{\sqsubseteq', \bot', \top', \sqcap', \sqcup'\})$ — полные решетки, $f : R \to R$ и $g : R' \to R'$ — изотонные отображения, а $h : R \to R'$ — строгое полу-$\sqcup$-непрерывное отображение (т.е. $h(\bot) = \bot'$), то $h(lfp(f)) = lfp(g)$.

Если a — элемент полной решетки, то его дополнением называется такой элемент b, что $a \sqcup b = \top$ и $a \sqcap b = \bot$. В общем случае дополнение элемента в полной решетке неединственно, но если оно единственно, то такая решетка называется решеткой с единственным дополнением и обозначается $R = (\mathcal{A}, \{\sqsubseteq, \bot, \top, \sqcap, \sqcup, \neg\})$, а дополнение элемента a обозначается $\neg a$.

Теорема 5 (Парик). Если $f : R \to R$ — изотонное отображение полной решетки R с единственным дополнением, то $\neg f(\neg x)$ тоже будет изотонным отображением R и $gfp(f) = \neg lfp(\lambda x . [\neg f(\neg x)])$.

Пусть $R = (\mathcal{A}, \{\sqsubseteq, \bot, \top, \sqcap, \sqcup\})$ — полная решетка, $n \geq 1$ и $F : R^n \to R^n$ — полу-$\sqcup$-непрерывное отображение. Пусть $X = (X_1, \ldots, X_n)$ — вектор из R^n. Система уравнений

$$X = F(X) \text{ (т.е. } X_j = F_j(X_1, \ldots, X_n),\ j = 1, \ldots, n)$$

имеет по крайней мере одно решение, которое является наименьшей верхней гранью последовательности $\{X^i \mid i \geq 0\}$, где $X^0 = (\bot, \ldots, \bot)$ и $X^{i+1} = F(X^i)$, т.е. $X_j^{i+1} = F_j(X_1^i, \ldots, X_n^i)$, $j = 1, \ldots, n$.

Характеристика свойств ДДС. Пусть $(S, \tau, \upsilon_{a_0}, \upsilon_{a^*}, \upsilon_\xi)$ — ДДС и множество достижимых состояний, удовлетворяющих условию β, определяется следующим образом:

$$\lambda s_2 . [\exists s_1 \in S : (\beta(s_1) l \wedge \tau^*(s_1, s_2)] = post(\tau^*(\beta),$$

где $post = \lambda\theta . [\lambda\beta . [\lambda s_2 . \exists s_1 \in S : \beta(s_1) \wedge \theta(s_1, s_2)]]]$.

Пример 3. Пусть π — программа, которой соответствует полностью определенная ДДС $(S, \tau, \upsilon_{a_0}, \upsilon_{a^*}, \upsilon_\xi)$, а φ и ψ — некоторые условия. Тогда частичная корректность программы π записывается в виде выражения

$$\upsilon_{a^*} \wedge post(\tau^*)(\upsilon_{a_0} \wedge \varphi) \to \psi,$$

которое означает, что каждое выполнение программы π, начинающееся в начальном состоянии a_0, где истинно условие φ, заканчивается в заключительном состоянии a^*, где истинно условие ψ. Вопрос терминальности программы не рассматривается, чем оправдывается слово «частичная» корректность.

Следующее утверждение показывает, что $post(\tau^*)(\beta)$ является решением уравнения $\alpha = \beta \vee post(\tau)(\alpha)$ на полной решетке $R = (\mathcal{A}, \{\to, false, true, \wedge, \vee, \neg\})$.

Теорема 6: а) решетка $R = (\mathcal{A}, \{\to, \mathit{false}, \mathit{true}, \wedge, \vee, \neg\})$ — полная решетка с единственным дополнением;

б) для любого θ отображение $post(\theta)$ является полным строгим $\vee$-морфизмом, и для любого условия β отображение $\lambda\theta.[post(\theta)(\beta)]$ — тоже строгий полный $\vee$-морфизм;

в) для любого τ и любого β имеет место

$$post(\tau^*)(\beta) = \bigvee_{n\geq 0} post(\tau^n)(\beta) = lfp(\lambda\alpha.[\beta \vee post(\tau)(\alpha)]).$$

Пример 4. В методе Флойда–Наура при доказательстве частичной правильности программы π относительно начальных условий φ и заключительных условий ψ корректность индуктивных предположений сводится к доказательству утверждений $(((v_{a_0} \wedge \varphi) \to t) \wedge post(\tau)(t) \to t) \wedge ((v_{a^*} \wedge t) \to \psi))$ при условии справедливости утверждения t.

Применяя рекурсивный индуктивный принцип, получаем, что из $(((v_{a_0} \wedge \varphi) \to t) \wedge (post(\tau)(t) \to t))$ следует $lfp(\lambda\alpha.[(v_{a_0} \wedge \varphi) \vee post(\tau)(\alpha)]) \to t)$ в силу п. в) предыдущей теоремы. Действительно, согласно этому свойству $(v_{a^*} \wedge post(\tau^*)(v_{a_0} \wedge \varphi)) \to (v_{a^*} \wedge t) \to \psi$.

Наоборот, если π — частично правильная программа относительно условий φ и ψ, то это может быть доказано с помощью метода Флойда–Наура. Это следует из того, что можно выбрать t как $lfp(\lambda\alpha.[v_{a_0} \wedge \varphi) \vee post(\tau)(\alpha)])$.

Рассмотренная выше характеристика ДДС выполнялась с помощью *post*-отображения, но характеризовать ДДС можно и с помощью *pre*-отображения.

Пусть β — некоторое условие вида

$$\lambda s_1.[\exists s_2 \in S : \tau^*(s_1, s_2) \wedge \beta(s_2)] = pre(\tau^*)(\beta).$$

Можем записать, что

$$pre = \lambda\theta.[\lambda\beta.[\lambda s_1[\exists s_2 \in S : \tau^*(s_1, s_2) \wedge \beta(s_2)]]].$$

Пример 5. Пусть программа π определяет полную детерминированную ДДС $(S, \tau, v_{a_0}, v_{a^*}, v_\xi)$, а φ и ψ являются входным и выходным условиями соответственно.Тогда доказательство тотальной корректности программы π сводится к доказательству импликации

$$(v_{a_0} \wedge \varphi) \to pre(\tau^*)(v_{a^*} \wedge \psi).$$

Другими словами, каждое вычисление программы π, начинающееся в состоянии a_0, где выполняется условие φ, приводит к состоянию a^*, где выполняется условие ψ.

Математические свойства отображения *pre* получаются двойственным образом из свойств *post*, поскольку $pre(\theta)(\beta) = post(\theta^{-1})(\beta)$ и $post(\theta)(\beta) = pre(\theta^{-1})(\beta)$.

Имеет место следующая теорема.

Терема 7: а) для любого θ отображение $pre(\theta)$ является полным строгим $\vee$-морфизмом и для любого условия β отображение $\lambda\theta.[pre(\theta)(\beta)]$ — тоже строгий полный $\vee$-морфизм;

б) для любого τ и любого β имеет место

$$pre(\tau^*)(\beta) = \bigvee_{n\geq 0} pre(\tau^n)(\beta) = lfp(\lambda\alpha.[\beta \vee pre(\tau)(\alpha)]).$$

Вычисления в детерминированной ДДС $(S, \tau, \upsilon_{a_0}, \upsilon_{a^*}, \upsilon_\xi)$, которые не ведут в состояние ошибки, удовлетворяют условию $\upsilon_\xi \wedge \neg\, pre(\tau^*)(\upsilon_\xi)$. Отсюда следует теорема 8.

Теорема 8. Пусть τ — тотальное детерминированное отношение переходов. Тогда для любого условия β имеет место

$$\neg\, pre(\tau^*)(\beta) = gfp(\lambda\alpha\,.[\neg\beta \wedge pre(\tau)(\alpha)])\,.$$

Таким образом, поведенческие свойства детерминированной ДДС, которая соответствует программе π, получаются как решения в виде неподвижных точек следующих уравнений: пусть φ и ψ — входное и выходное условия для детерминированной ДДС $(S, \tau, \upsilon_{a_0}, \upsilon_{a^*}, \upsilon_\xi)$, тогда:

1) для множества состояний, которые удовлетворяют начальному условию φ,

$$post(\tau^*)(\upsilon_{a_0} l \wedge \varphi) = lfp(\lambda\alpha\,.[(\upsilon_{a_0} \wedge \varphi) \vee post(\tau)(\alpha)]);$$

2) для множества состояний, которые удовлетворяют заключительному условию ψ,

$$pre(\tau^*)(\upsilon_{a^*} \wedge \psi) = lfp(\lambda\alpha\,.[(\upsilon_{a^*} \wedge \psi) \vee pre(\tau)(\alpha)]);$$

3) для множества состояний, которые приводят к состоянию ошибки,

$$pre(\tau^*)(\upsilon_\xi) = lfp(\lambda\alpha\,.[\upsilon_\xi \vee pre(\tau)(\alpha)])\,;$$

4) для множества состояний, которые не приводят к состоянию ошибки,

$$\neg\, pre(\tau^*)(\upsilon_\xi) = gfp(\lambda\alpha\,.[\neg\upsilon_\xi \wedge pre(\tau)(\alpha)])\,;$$

5) для множества состояний, которые приводят к неопределенности,

$$pre(\tau^*)(\upsilon_{a^*} \vee \upsilon_\xi) = gfp(\lambda\alpha\,.[\neg\upsilon_{a^*} \wedge \neg\upsilon_\xi \wedge pre(\tau)(\alpha)])\,.$$

3. ЗАКЛЮЧИТЕЛЬНЫЙ ПРИМЕР

Проиллюстрируем на примере теории программных инвариантов построение иерархии абстракций, которые возникают при поиске соотношений в программах. В процессе поиска соотношений в программах имеем дело со следующими объектами:

а) программой в некотором языке программирования и ее областью данных;

б) языком соотношений;

в) генератором соотношений (алгоритмом поиска соотношений) [1, 2].

Абстракции, касающиеся программы и ее данных, состоят в следующем:

а1) абстрагирование от структурных переменных программы (массивов, курсоров, указателей и т.п.) и учет только простых переменных и действий над ними;

а2) абстракция области данных (например, принимается, что область данных является абелевой группой или векторным пространством).

Абстракции, касающиеся языка соотношений, состоят в том, что принимается соглашение о языке только равенств или только линейных равенств и неравенств. Среди этих абстракций имеется иерархия следующего вида:

б1) язык полиномиальных равенств [13];

б2) язык полиномиальных равенств ограниченной степени [14];

б3) язык линейных равенств и неравенств [15];

б4) язык линейных равенств [16];

б5) язык равенств вида $r = c$, где r — переменная, а c — константа [17].

Абстракции, касающиеся генератора инвариантов, вытекают из абстракций языка соотношений и алгебры данных. Если принимается абстракция языка линейных равенств, то генератор не учитывает условий в условных операторах, не являющихся линейными равенствами.

Необходимость в таких абстракциях состоит в том, что при наличии той или иной абстракции процесс поиска соотношений может успешно заканчиваться, а не продолжаться бесконечно. При этом генератор соотношений вычисляет наименьшую неподвижную точку. Так, генератор, реализующий метод верхней аппроксимации (алгоритм МВА [1, 2]), вычисляет наименьшую неподвижную точку, являющуюся решением системы уравнений

$$X_a = \begin{cases} N_0, & \text{если } a = a_0, \\ X_a \bigcap\limits_{(a'u,y,a)\in S} ef(X_{a'}, y) & \text{в противном случае,} \end{cases}$$

для всех состояний a, a' и переходов $(a', u, y, a) \in S$ программы, где a_0 — начальное состояние, N_0 — множество соотношений в начальном состоянии, а функция ef вычисляет множество соотношений на выходе оператора присваивания y с учетом на его входе множества соотношений $X_{a'}$. Соотношения, входящие в результирующие множества, и будут инвариантными соотношениями анализируемой программы.

При этом получаемые в процессе поиска соотношения составляют решетку относительно операций объединения и пересечения. Полнота множества соотношений зависит от того, будет ли дистрибутивной функция ef относительно операции пересечения.

В связи со сказанным выше заметим, что семантический анализ программ требует изучения дисциплин, которые исследуют разнородные семейства методов и алгоритмов. Это связано с тем, что методы анализа программ независимо могут применяться к языкам, объектному коду, семантике, свойствам спецификаций, свойствам объектного кода, абстракциям и т.п. Такой анализ возможен на аппроксимациях структур, входящих в семантические спецификации. С практической точки зрения, эта методология может применяться в процессе разработки иерархии семантик и иерархии абстрактных алгебр на разных уровнях абстракции [18]. Цель такой методологии — создание средств такой семантической системы анализаторов, которая (по возможности) автоматически выполняла бы анализ спецификаций.

Дальнейшие исследования проблемы и результаты в этой области можно найти в работах [6, 11, 12, 19, 20].

СПИСОК ЛИТЕРАТУРЫ

1. Максимец А.Н. Поиск программных инвариантов в виде полиномов // Докл. НАН Украины. — 2013. — № 9. — С. 44–50.

2. Крывый С.Л., Максимец А.Н. Формальные методы верификации на основе сетей Петри // Тр. 10-й междунар. конф. «Теоретико-прикладные аспекты построения програм. систем. TAAbD’2013». — Ялта, 2013. — С. 75–80.

3. Sintzoff F. Calculating properties of programs by variations on specific models // ACM Conf. on Proving Assertions about Programs; Sigplan Notices. — 1972. — **7**, N 1. — P. 203–207.

4. Cousot P. Semantic foundations of program analysis // S.S. Muchnik and N.D. Jones, editors, Program Flow Analysis: Theory and Appl. — Prentice-Hall, 1981. — **10**. — P. 303–342.
5. Cousot P. Abstract interpretation: a unified lattice model for static analysis of programs by construction of fixpoints. // 4-th ACM Symposium on Principles on Program. Languages. — Los Angeles: ACM, 1977. — P. 238–252.
6. Cousot P. Verification by abstract Interpretation // Lecture Notes in Comput. Sci. — 2003. — N 2772. — P. 243–268.
7. Nielson F. A denotation framework for data flow analisys // Acta Inform. — 1982. — N 18. — P. 265–287.
8. Nielson F. Two-level semantics and abstract interpretation // Theor. Comput. Sci. (Fundamental Studies). — 1989.— N 69. — P. 117–242.
9. Jones N.D., Gomard C.K., Sestoft P. Partial evaluation and automatic program generation. — Techn. Un-t of Denmark, 1993. — 414 p.
10. Биркгоф Г. Теория решеток. — М.: Наука, 1984. — 564 с.
11. Cousot P., Cousot R. Abstract Interpretation frameworks // Journ. of Logic and Comput. — 1992. — **2**, N 4. — P. 511–547.
12. Cousot P., Cousot R. Modular static program analysis. Invited paper // Lecture Notes in Comput. Sci. — 2002. — N 2304. — P. 159–178.
13. Rodriguez-Carbonell E., Kapur D. An abstract interpretation approach for automatic generation of polynomial invariants // Proc. Static Analysis Sympos. (SAS). — Italy. — August 2004. — P. 1–8.
14. Lvov M.S. About one algorithm of program polynomial invariants generation // Proc. Workshop on Invariant Generation: (Techn. rep.) Univ. of Linz; Eds. M. Giese, T. Jebelean. N 0707 (RISC Rep. Ser.). Linz (Austria), 2007. — P. 85–99.
15. Кривой С.Л., Ракша С.Г. Поиск линейных инвариантных зависимостей в программах // Кибернетика. — 1984. — № 6. — С. 23–28.
16. Krivoi S.L. About one invariant search algorithm in programs // Cybernetics and Systems Analysis. — 1981. — N 5. — P. 12–18.
17. Kildall G.A. A unified approach to program optimization // Conf. Rec. of ACM Symp. on Principles of Program Languages. — Boston. — October 1–3. — 1973. — P. 194–206.
18. Cousot P. Abstract Interpretation Persective // ACM Workshop on Strategic Directions in Comput. Res. MIT Laboratory for Comput. Sci.: Cambridge, Massachusetts, USA,1996. — P. 1–8.
19. Cousot P. Abstract interpretation based formal methods and future challenges. — http://www.di.ens.fr/~cousot/. — 2003. — P. 131–151. Chap. 10. — P. 303–342.
20. Saddek B., Graf S., Lakhnech Y. Abstraction as the key for invariant verification // Lecture Notes in Comput. Sci. — 2003. — N 2772. — P. 67–99.

Поступила 15.04.2013