УДК 519.713

Г.Н. ЖОЛТКЕВИЧ

Харьковский национальный университет имени В.Н. Каразина, Украина

НОРМАЛИЗАЦИЯ ПРОГРАММ И ПРОГРАММНЫЕ ИНВАРИАНТЫ

В статье предложен метод преобразования программы к виду, содержащему только один цикл. Доказано, что этот метод приводит к эквивалентной программе, названной нормализованной. Нормализованная форма программы является удобным инструментом для решения ряда задач статического анализа программ, в том числе, задачи оценки программных инвариантов.

статический анализ программ, программный граф, эквивалентное преобразование программ, программный инвариант

Введение

Статический анализ программ вот уже тридцать лет рассматривается специалистами как инструмент повышения гарантоспособности программного обеспечения (ПО), в первую очередь, критического. В статье [1] дан подробный анализ причин, которые обусловили такое положение дел. Главной из них является то, что статический анализ, в отличие от других методов оценки гарантоспособности ПО, позволяет достоверно установить отсутствие ошибок того или иного класса.

Развитие методов статического анализа программ привело к выделению двух классов задач: во-первых, задач однопоточных, имеющих, обычно, вычислительный характер, и, во-вторых, задач управления асинхронными потоками вычислений. Первый класс задач анализируется при помощи традиционных логических методов проверки правильности программ [2], в то время как второй использует формализмы темпоральной логики и методы, известные как model checking [3]. Общим методом для обоих классов задач является метод программных инвариантов, заключающийся в проверке инвариантности определенного спецификациями свойства анализируемой программы.

Понятие инварианта было впервые введено, по-видимому, в работах [4, 5]. В работе [6] дан современный обзор теории программных инвариантов.

Анализ этих результатов показывает, что одной из основных трудностей при применении методов, базирующихся на оценке инвариантов, является наличие программных циклов.

В настоящей работе предлагается и обосновывается метод эквивалентного преобразования программы к программе с одним циклом. Процесс, определяемый этим методом, можно назвать нормализацией программы.

Следует отметить, что возможны и другие методы нормализации программы, приводящие, возможно к другой нормальной форме.

Основные определения и обозначения

Напомним, что графом потока управления называется четверка $\langle L, T, b, e \rangle$, где L и T – конечные множества, называемые множеством меток и множеством переходов соответственно, а $b, e : T \to L$ – отображения, при этом для всякого перехода $\tau \in T$ метка $b(\tau)$ интерпретируется как метка начала перехода τ, а $e(\tau)$ – как метка его конца. При этом выполнены следующие условия:

1) существует единственная метка $i \in L$ такая что $i \notin \bigcup_{\tau \in T} e(\tau)$. Она называется начальной меткой;

2) существует единственная метка $h \in L$ такая

что $h \notin \bigcup_{\tau \in T} b(\tau)$. Она называется заключительной меткой;

3) для всякой метки существует хотя бы один путь, проходящий через эту метку, из начальной метки в заключительную метку.

Мы будем считать, что задано множество D, элементы которого представляют данные и конечное множество V, элементы которого соответствуют программным переменным. Состоянием памяти назовем отображение $s: V \to D \cup \{\nabla\}$. При этом для $x \in V$ равенство $s(x) = \nabla$ интерпретируется как «в состоянии s переменная x не определена». Множество состояний памяти будем обозначать через Σ.

Будем считать, что задано конечное множество символов условий $\left\{\beta_i^{n_i} \mid i = 1,\ldots,N\right\}$, причем символу $\beta_i^{n_i}$ поставлено в соответствие множество $\left[\!\left[\beta_i^{n_i}\right]\!\right] \subset D^{n_i}$.

Пусть теперь $x_1,\ldots,x_{n_i} \in V$, тога:

$$\left[\!\left[\beta_i^{n_i}(x_1,\ldots,x_{n_i})\right]\!\right] = \{s \in \Sigma \mid s(x_1) \neq \nabla \wedge \ldots \wedge s(x_{n_i}) \neq \nabla \wedge \wedge (s(x_1),\ldots,s(x_{n_i})) \in \left[\!\left[\beta_i^{n_i}\right]\!\right]\} \quad (1)$$

Формула (1) определяет множество состояний памяти, удовлетворяющих простому условию $\beta_i^{n_i}(x_1,\ldots,x_{n_i})$. Сложные условия определяются как обычно, путем построения логических формул с использованием простых условий и пропозициональных связок.

Будем также считать, что задано конечное множество символов операций $\left\{f_j^{m_j} \mid j = 1,\ldots,M\right\}$, причем символу $f_j^{m_j}$ поставлено в соответствие отображение $\left[\!\left[f_j^{m_j}\right]\!\right]: D^{m_j} \to D$.

Пусть $y, x_1,\ldots,x_{m_j} \in V$, тогда:

$$s.\left[y := f_j^{m_j}(x_1,\ldots,x_{m_j})\right](x) = \begin{cases} \nabla, \text{если } s(x_1) = \nabla \vee \ldots \vee s(x_{m_j}) = \nabla, \\ s(x), \text{если } x \neq y, \\ \left[\!\left[f_j^{m_j}\right]\!\right](s(x_1),\ldots,s(x_{m_j})), \text{если } x = y. \end{cases} \quad (2)$$

Последовательность присваиваний определяет преобразование состояний памяти, состоящее в последовательном применении присваиваний в соответствии с формулой (2).

Таким образом, у нас определены множества условий и операций на Σ, которые мы обозначим через $C(\Sigma)$ и $S(\Sigma)$ соответственно.

Введем теперь понятие программного графа, который используется в этой работе как модель программы.

Пусть задан граф потока управления $G = \langle L, T, b, e\rangle$ и отображения $guard: T \to C(\Sigma)$ и $effect: T \to S(\Sigma)$, которые определяют условия выполнения перехода и преобразование состояния памяти при выполнении перехода.

Определение 1. Программным графом будем называть шестерку $\langle L,T,b,e,guard,effect\rangle$, где четверка $G=\langle L,T,b,e\rangle$ образует граф потока управления, а guard и effect – являются отображениями, описанными выше.

Программный граф определяет динамику на множестве состояний памяти следующим образом.

Определение 2. Пусть задан программный граф P. Слово $s_0\tau_1 s_1\tau_2\ldots\tau_n s_n$ будем называть началом вычисления, если выполнены следующие условия:

1) $s_i \in \Sigma(i = 0,\ldots,n), \tau_j \in T(j=1,\ldots,n)$;

2) $e(\tau_i) = b(\tau_{i+1}), i = 1,\ldots,n-1$;

3) $b(\tau_1) = i$;

4) $s_i \in \left[\!\left[guard(\tau_{i+1})\right]\!\right], i = 0,\ldots,n-1$;

5) $s_{i+1} = \mathrm{effect}(\tau_{i+1})(s_i), i = 0,\ldots,n-1$.

Определение 3. Пусть $s_0\tau_1 s_1\tau_2\ldots\tau_n s_n$ – начало вычисления. Оно называется вычислением, если $e(\tau_n) = h$.

Определение 4. Пусть $(s,s') \in \Sigma^2$, будем говорить, что эта пара удовлетворяет спецификации «вход-выход» программного графа, если существует вычисление $s_0\tau_1 s_1\tau_2\ldots\tau_n s_n$, удовлетворяющее условию $s_0 = s \wedge s_n = s'$.

Нормализация программного графа

Пусть $\langle L,T,b,e,guard,effect\rangle$ программный граф с множеством переменных V и множеством данных D. Построим новый программный граф, определив:

1) $\overline{L} = \{start, execute, halt\}$;

2) $\overline{T} = \{\tau_{in}, \tau_{out}\} \cup \{\overline{\tau} \mid \tau \in T\}$, где $b(\tau_{in}) = start$, $e(\tau_{in}) = execute$, $b(\tau_{out}) = execute$, $e(\tau_{out}) = halt$, для всякого $\tau \in T$: $b(\overline{\tau}) = e(\overline{\tau}) = execute$;

3) $\overline{V} = V \amalg \{p\}$, $\overline{D} = D \amalg L$; для всякого символа условия $\beta_i^{n_i}$ определим $\overline{\beta_i^{n_i}}$ и его интерпретацию $[\![\overline{\beta_i^{n_i}}]\!] = [\![\beta_i^{n_i}]\!]$; для всякого символа операции $f_j^{m_j}$ определим $\overline{f_j^{m_j}}$ и его интерпретацию

$$[\![\overline{f_j^{m_j}}]\!](v_1,\ldots,v_{m_j}) = \begin{cases} \nabla, \text{если } v_1 = \nabla \vee \ldots \vee v_{m_j} = \nabla \\ \nabla, \text{если } v_1 \in \mathbf{L} \vee \ldots \vee v_{m_j} \in \mathbf{L} \\ [\![f_j^{m_j}]\!](v_1,\ldots,v_{m_j}) \end{cases}$$

4) $guard(\tau_{in}) \equiv p = \nabla$, $guard(\tau_{out}) \equiv p = h$, для всякого $\tau \in T$: $guard(\overline{\tau}) \equiv p = b(\tau) \wedge guard(\tau)$

5) $effect(\tau_{in}) \equiv p := i$, $effect(\tau_{out}) \equiv id$, для всякого $\tau \in T$: $effect(\overline{\tau}) \equiv p := e(\tau); effect(\tau)$

Утверждение 1.

Пусть $\overline{P} = \langle \overline{L},\overline{T},b,e,guard,effect\rangle$ является шестеркой, построенной выше, тогда $\overline{P}$ – программный граф.

Доказательство проводится простой проверкой.

Утверждение 2.

Пусть $s_0\tau_1 s_1\tau_2\ldots\tau_n s_n$ – начало вычисления для программного графа P, тогда $\overline{s_{-1}}\tau_{in}\overline{s_0}\tau_1 s_1\ldots\overline{\tau_n}\overline{s_n}$ является началом вычисления для программного графа $\overline{P}$, где $\overline{s_{-1}}(x) = s_0(x)$, $x \in V$ и $\overline{s_{-1}}(p) = \nabla$, $\overline{s_0}(x) = s_0(x)$, $x \in V$ и $\overline{s_0}(p) = i$, $\overline{s_i}(x) = s_i(x)$, $x \in V$ и $\overline{s_i}(p) = b(\tau_i)$ для $i = 1,\ldots,n$.

Доказательство проведем, используя математическую индукцию по n. Прежде всего заметим, что начальной меткой графа потока управления, соответствующего $\overline{P}$, является метка start, а конечной – метка halt.

Для $n = 0$ необходимо доказать, что $\overline{s_{-1}}\tau_{in}\overline{s_0}$ является началом вычисления. Но действительно, условия определения 2 выполнены по построению.

Пусть утверждение верно для $m < n$. Тогда для начала вычисления $\overline{s_{-1}}\tau_{in}\overline{s_0}\tau_1 s_1\ldots\overline{\tau_n}\overline{s_n}$ условия 1) и 3) определения 2, очевидно, верны. Для доказательства истинности условия 2) определения 2 достаточно доказать, что $e(\overline{\tau_{n-1}}) = b(\overline{\tau_n})$, но это верно, поскольку $\overline{\tau_{n-1}}$ и $\overline{\tau_n}$ соответствуют переходам τ_{n-1} и τ_n программного графа P, а значит, $e(\overline{\tau_{n-1}}) = b(\overline{\tau_n}) = execute$ по построению. Для доказательства истинности условия 4) определения 2 достаточно доказать, что $\overline{s_{n-1}} \in guard(\overline{\tau_n})$. Но по построению $guard(\overline{\tau_n}) \equiv p = b(\tau_n) \wedge guard(\tau_n)$, а $\overline{s_{n-1}} \mid V = s_{n-1}$, а $\overline{s_{n-1}}(p) = b(\tau_n)$. Для доказательства истинности

условия 5) определения 2 достаточно доказать, что $\overline{s_n} = \text{effect}\left(\overline{\tau_n}\right)\left(\overline{s_{n-1}}\right)$, но это также следует из построения.

Таким образом, доказательство завершено.

Утверждение 3.

Пусть $s_0\tau_1 s_1\tau_2 \ldots \tau_n s_n$ – вычисление для P, тогда $\overline{s_{-1}}\,\overline{\tau_{in}}\,\overline{s_0}\,\overline{\tau_1}\,\overline{s_1} \ldots \overline{\tau_n}\,\overline{s_n}\,\overline{\tau_{out}}\,\overline{s_{n+1}}$ является вычислением для $\overline{P}$, где $\overline{s_{n+1}} \mid V = s_n$, $\overline{s_{n+1}}(p) = \nabla$.

Доказательство. Утверждение следует из утверждения 2 и построения.

Утверждение 4.

Пусть $\overline{s_{-1}}\,\overline{\tau_{in}}\,\overline{s_0}\,\overline{\tau_1}\,\overline{s_1} \ldots \overline{\tau_n}\,\overline{s_n}\,\overline{\tau_{out}}\,\overline{s_{n+1}}$ – вычисление для $\overline{P}$, для которого $\overline{s_{-1}}(p) = \nabla$, тогда $s_0\tau_1 s_1\tau_2 \ldots \tau_n s_n$ – вычисление для P.

Доказательство аналогично предыдущим.

Пример нормализации

В качестве примера рассмотрим программный граф алгоритма сортировки пузырьком (рис. 1).

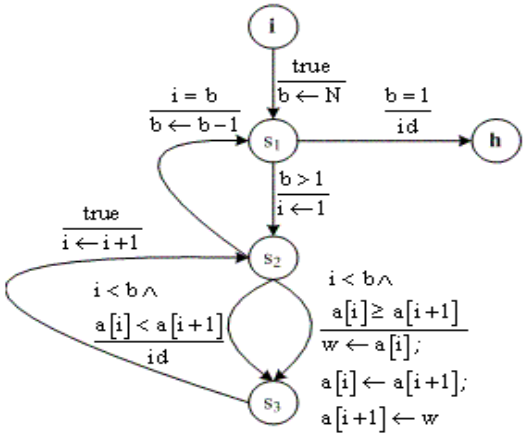

Рис. 1. Программный граф алгоритма сортировки пузырьком

На рис. 2 приведен результат нормализации этого программного графа.

Программные инварианты исходного и нормализованного графов

Напомним, что программным инвариантом для программного графа P называется элемент ψ из $C(\Sigma)$, удовлетворяющий условию: для любого начала выполнения $s_0\tau_1 s_1\tau_2 \ldots \tau_n s_n$ такого, что $s_0 \in [\![\psi]\!]$, выполняется $s_n \in [\![\psi]\!]$. Связь между программными инвариантами графов P и $\overline{P}$ устанавливается следующей теоремой.

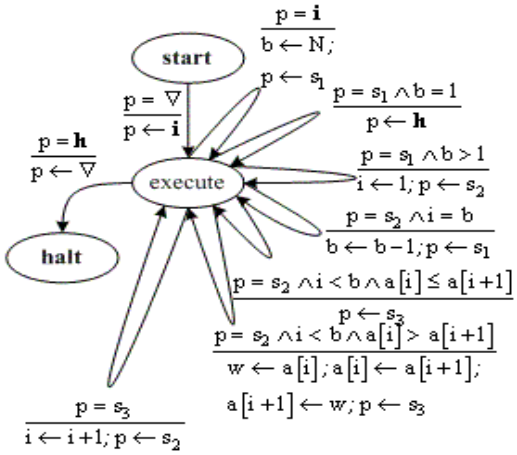

Рис. 2. Нормализованный программный граф алгоритма сортировки пузырьком

Теорема.

Инвариант $\psi \in C(\Sigma)$ программного графа P является инвариантом программного графа $\overline{P}$ и наоборот инвариант $\overline{\psi} \in C(\overline{\Sigma})$ программного графа $\overline{P}$, который обладает следующими свойствами:

1) если $\overline{s_1} \in [\![\overline{\psi}]\!]$, $\overline{s_2} \in C(\overline{\Sigma})$ и $\overline{s_1} \mid V = \overline{s_2} \mid V$, тогда $\overline{s_2} \in [\![\overline{\psi}]\!]$;

2) если $\bar{s} \in [\![\overline{\psi}]\!]$, тогда для всякого $x \in V$ выполнено $\bar{s}(x) \in D \cup \{\nabla\}$, тогда условие ψ, определенное формулой

$$[\![\psi]\!] = \left\{ s \in \Sigma \mid \bar{s}(p) = \nabla \wedge \bar{s} \mid V = s \wedge \bar{s} \in [\![\overline{\psi}]\!] \right\}$$

является программным инвариантом графа **P**.

Доказательство следует из утверждений 2 и 3.

Выводы

Приведенное преобразование программного графа, несмотря на расширение множества возможных значений данных, приводит к эквивалентной про-

грамме, если нас интересуют только свойства, определенные в терминах исходного множества данных.

Вид нормализованной программы является очень простым: конструкция цикла, телом которого является конструкция выбора.

Такой вид программы позволяет локализовать проблемы, возникающие при ее анализе за счет наличия многих циклов.

Такой специальный вид программы позволяет строить инварианты даже без учета условий завершения цикла в том случае, когда эти инварианты могут быть выражены в терминах абстракций [7, 8] области данных, обладающих свойством обрыва убывающих цепей. Именно такая ситуация возникает при контроле инвариантности физической размерности программных переменных.

Важным с точки зрения автора преимуществом нормализованной программы является выделение из ее текста конструкции выбора, которая сосредотачивает в себе всю логику программы, и которую можно рассматривать как оператор на памяти, играющий роль аналогичную инфинитизимальному генератору динамической системы. По аналогии его можно назвать генератором.

Тогда вопрос о программных инвариантах аналогичен вопросу об аттракторах динамической системы.

В этом направлении открывается путь для ослабления понятия инварианта до понятия, например, асимптотического инварианта, т.е. условия, которое приобретает свойство инварианта после выполнения конечного числа проходов генератора по циклу.

Отмеченная аналогия позволяет применять методы динамических систем для анализа программ, вводя известные для динамических систем числовые инварианты, такие как энтропия, свободная энергия, температура и т.п. В этой связи вызывает интерес вопрос об их содержательности для статического анализа программ.

Литература

1. Binkley D. Static Code Analysis: A Road Map // Future of Software Engineering, 2007 (FOSE'07). – IEEE, 2007. – P. 104-119.

2. Hoare C.A.R., Jones. Essays of Computing Science. – N. York: Prentice Hall, 1989. – 412 p.

3. Clarke E.M. Jr., Grumberg O., Peled D. Model Checking. – Cambridge, MA: MIT Press, 2001. – 416 p.

4. Floyd R.W. Assigning meanings to programs // Proc. Symposia in Applied Mathematics. – Vol. 19, 1967. – P. 19-32.

5. Hoare C.A.R. An axiomatic basis for computer programming. Communications of the ACM. – Vol. 10, N 12, 1969. – P. 576-580.

6. Sankaranarayanan S. Mathematical Analysis of Programs // A Dissertation for The Degree of Doctor of Philosophy. – Stanford University, 2005. – 163 p.

7. Cousot P., Cousot R. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints // 4th POPL. – ACM Press, 1977. – P. 238-252.

8. Cousot P., Cousot R. Abstract interpretation framework. – J. Logic Programming. – Vol. 13, NN 2, 3, 1992. – P. 103-179.

Поступила в редакцию 24.01.2008

Рецензент: д-р техн. наук, проф. Б.М. Конорев, Национальный аэрокосмический университет им. Н.Е. Жуковского «ХАИ», Харьков.