СБОРНИК **НАУЧНЫХ ТРУДОВ НГТУ. – 2015. – № 1(79).** – 105–116

УДК 62-50:519.216

АНАЛИЗ СИМВОЛЬНОГО ФРЕЙМВОРКА ДЛЯ СТАТИЧЕСКОГО АНАЛИЗА ИМПЕРАТИВНЫХ ЯЗЫКОВ ПРОГРАММИРОВАНИЯ*

Д.О. РОМАННИКОВ

630073, РФ, г. Новосибирск, пр. Карла Маркса, 20, Новосибирский государственный технический университет, кандидат технических наук, доцент кафедры автоматики. E-mail: rom2006@gmail.ru

Данная статья посвящена анализу работы Баргсталлера, Шольца и Блибегрера по преобразованию программы в символьный домен с целью дальнейшего анализа. В данной работе приводится математический аппарат для описания программы в терминах символьного вычисления: предлагается использовать понятия контекста, состояния, условных переходов, суперконтекста и других для формульного описания программы. Приведены примеры преобразования программы в символьный домен. В работе вышеприведенных авторов основным достижением считается предложенный способ описания циклов. Данный способ позволяет получить не только формальное представление цикла, но и свернуть потенциально бесконечное число вариантов путей программы к конечному числу. В работе рассмотрены примеры из оригинальной статьи, но все примеры достаточно простые и не содержат более 8 строк, что в конечном итоге приводит к тому, что по ним нельзя сделать вывод о применимости метода.

В ходе анализа исследуемой работы был выявлен следующий ряд недостатков: 1) метод «сворачивания» числа путей в цикле не дает заявленного результата в случае, если в цикле есть ветвление; 2) не очевидно, как будет выглядеть предлагаемое формальное описание циклов в случае вложенных циклов. Причем в циклах с большим числом вложенности должны быть зависимости по итерируемым переменным, т. е. выражения вида $a\,[i]\,[j] = b\,[i]\,[j]$ или подобные; 3) отсутствует описание символьного представления для массивов, что особенно важно для перебора элементов массива в циклах в виде a [i].

Ключевые слова: программное обеспечение, тестирование, верификация, поверка моделей, символьный анализ, корректность программ, динамическая верификация, символьный фреймворк, графы, программные циклы, тотальная корректность программ

DOI: 10.17212/2307-6879-2015-1-105-116

ВВЕДЕНИЕ

Данная статья посвящена разбору математического аппарата, используемого для описания символьного фреймворка для статического анализа императивных языков программирования. Стоит отметить, что в настоящее время

* Статья получена 20 января 2015 г.

в направлении анализа программного обеспечения (ПО) есть несколько основных направлений: 1) проверка моделей [11, 12]; 2) статический анализ [7–10, 13, 14] и 3) доказательство корректности, а также менее распространенные [15–24]. Все вышеприведенные направления имеют свои достоинства, недостатки и области применения. Символьные вычисления следует рассматривать как частный случай статического анализа, который на сегодняшний день является наиболее распространенным на практике.

Пример символьного анализа программы можно увидеть в таблице, где приведены значения с подчеркиванием (c, b) для переменных, для которых точное значение либо неизвестно, либо может меняться в зависимости от каких-то условий. Каждой строчке программы соответствует множество значений переменных, где для каждой переменной указано точное значение или выражение с участием переменных.

Далее в работе будут рассмотрены и обсуждены различные математические приемы, с помощью которых может быть выполнен символьный анализ программы, а также подходы к анализу из работы [1].

Пример символьного анализа программы

Программа для анализа	Название и значения переменных
`int a = 1, b, c;`	{a = 1, b = b, c = c}
`a = b + a;`	{a = 1 + b, b = b, c = c}
`b = a * 2;`	{a = 1 + b, b = a*2, c = c}
`c = 100;`	{a = 1 + b, b = b, c = 100}

1. ОБОЗНАЧЕНИЯ

Основной работой для анализа в данной статье является работа [1], в которой предлагается метод уменьшения количества путей для анализа с потенциально бесконечного числа путей до конечного. Стоит отметить, что данные выводы также были получены независимо от [1] в [4, 5].

Почти все методы анализа ПО используют в основе граф управления (Control Flow Graph (CFG)), который является направленным маркированным графом $G = <N, E, n_e, n_x>$, где N – множество узлов графа, E – множество переходов таких, что $E \subseteq N \times N$. У каждого перехода $e \in E$ есть голова $h(e) \in N$ и хвост $t(e) \in N$. Узлы n_e, n_x – начальный и конечный узлы графа G соответственно. Причем $in(n_e) = 0$ и $out(n_x) = 0$, где операторы in, out – множество входных/выходных узлов соответственно ($in(n) = \{e \in E: t(e) = n\}$, $out(n) =$

= {$e \in E$: $h(e) = n$}). Поскольку граф G – это граф управления программы, то каждый узел n в графе G должен иметь путь от n_e к n_x. Путь будем обозначать как $\pi = <e1, e2, e3, \ldots ek>$, где $t(er) = h(er + 1)\ \forall\ r \in [1, k - 1]$.

В работе [2] показано, как представить программные пути в виде регулярных выражений, а также некоторые правила манипулирования ими. Если $\sum$ является конечным алфавитом, пересекающимся с множеством {Λ, $\emptyset$, (,)}, то регулярным выражением будем называть любое выражение, построенное с применением следующих правил: 1a) «Λ», «$\emptyset$» – атомарные регулярные выражения – $\forall\ a,\ a \in \sum$, «a» – регулярное выражение; 1b) если R_1 и R_2 регулярные выражения, то $(R_1 + R_2)$, (R_1R_2), $(R_1)^*$ – составные регулярные выражения. В регулярных выражениях Λ обозначает пустую строку, $\emptyset$ – пустое множество, + обозначает объединение, R_1R_2 – конкатенация выражений R_1 и R_2. Вышеприведенный математический аппарат позволяет описать путь в графе G как строку π над E, но не все строки над E будут являться путями в графе G. Выражением пути P для (v, w) будем называть регулярное выражение над E такое, что каждая строка в $L(P)$ является программным путем от v до w.

Для описания символьного вычисления программы используется понятие функции состояния $s \in S$, которая позволяет соотнести программную переменную к соответствующему символьному значению. Также необходимо понятие контекста $c \in C \subseteq [S \times SymPred]$, т. е. [$s$, p], где s – состояние, p – условие пути (выражение, которое накладывается на переменную в состоянии s).

Функция $F_s \subseteq \{f\colon C \rightarrow C\}$ определяет изменение контекста при выполнении вычисления при переходе от узлов графа G по дуге e. Так как функция $f \in F_s$ определяет единичное вычисление при переходе по дуге e, то необходимо ввести функцию, которая будет сопоставлять каждой дуге e функцию f: $M_s : E \rightarrow F_s$, т. е. определять символьное выполнение программы на одном из путей. M_s = {0, если π пустое множество, $M_s(e_k)$, $M_s(e_{k-1})$, $M_s(e_{k-2})$,…, $M_s(e_1)$, если $\pi = < e_k , e_{k-1} , e_{k-2} , \ldots e_1 >$}.

Любая промышленная программа из-за наличия условных операторов содержит несколько путей между узлами графа. То есть необходимо определять не просто значение контекста на программном пути, а объединение всех контекстов на различных путях графа программы. Для этого вводится понятие суперконтекста $sc = \{c_1, c_2, c_3, c_k, \ldots\}$, $c \in sc$, $sc = [s, p]$, где s – состояние, а p – условие, выполняемое при переходе программы в состояние s. Вычислить суперконтекст для множества путей в программе можно при помощи формулы [3].

$$mop(n) = \bigcup_{\pi \in Path(n_e, n)} M_s(\pi)(c_e). \qquad (1)$$

2. РЕДУКЦИЯ ЧИСЛА ВОЗМОЖНЫХ ПУТЕЙ ВЫПОЛНЕНИЯ ПРОГРАММЫ

Определение суперконтекста по формуле (1) имеет серьезный побочный эффект, а именно: оно зависит от количества путей между узлами, а их число может быть произвольным и потенциально бесконечным. Следовательно, вычисление *mop*(*n*) может выполняться бесконечно.

Для решения этой проблемы в [1] предлагается использовать выражения диапазона, которые являются символьными выражениями вида $0 \le l \le l_w$, где $l \in L$ – переменная цикла, l_w – верхняя граница переменной цикла. На основе выражения диапазона строится система повторений *rs*(*l*) на основе переменной цикла.

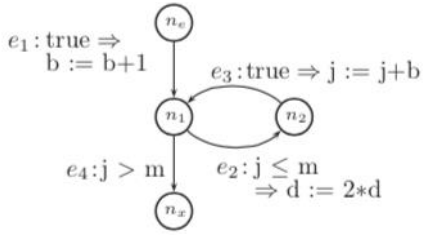
Для описания символьных выражений с учетом наличия выражений диапазона необходимо построить измененный контекст, который будет называться замкнутым (закрытым) контекстом. Замкнутым контекстом будем называть элемент множества $\overline{C} = S \times SymPred \times R$, [*s*, *p*, *r*]. Замкнутый контекст отличается от контекста наличием системы повторений и если $r = \emptyset$, то $\overline{C} = c$.

Основная идея работы [1] заключается в применении оператора $\circledast$, который используется в формуле (2). Данный оператор требуется для того, чтобы отойти от потенциально бесконечного количества вариантов путей в (1):

$$\overline{C}_{out} = \varphi(P_1^*)(\overline{C}_{in}) = \varphi(P_1) \odot (\overline{C}_{in}) = f \odot (\overline{C}_{in}). \quad (2)$$

Далее рассмотрим пример анализа цикла при помощи предлагаемого выражения (2). Перед первым исполнением цикла значение замкнутого контекста будет $\overline{C}_0 = [s_0, p_0, r_0]$ и после первой итерации цикла – $\overline{C}_1 = [s_1, p_1, r_1]$. Таким образом, s_{out} вычисляется на основании s_{in} путем замены символьных выражений, которые определяют значения переменных v_i на соответствующие значения v_i от переменной цикла *i*: $\forall\, v_i \in Dom(s_{in})$: $s_{out} ::= s_{in}\,[v_i \rightarrow v_i(I)]$. Тогда общее условное выражение пути p_{out} будет определяться по формуле

$$p_{in} \wedge (0 \le l \le l_w) \wedge (\wedge_{0<l'<l}\, p(l' - 1)). \quad (3)$$




```
1  b := b + 1;
2  l := 0;
3  while j <= m loop
4    d := 2*d;
5    j := j+b;
6    l := l+1;
7  end loop;
```

Рис. 1. Пример программы с циклом для анализа

Для программы в примере на рис. 1 выражение для p_{out} рассчитывается по формуле (3) и имеет вид: $p_1 = j \le m$, $p(l) = j(0) \le m(0) \wedge j(1) \le m(1) \wedge j(2) \le m(2) \ldots \wedge j(1-1) \le m(1-1) = \Lambda_{0 \le l' \le l} (j(l'-1) \le m(l'-1))$. Система повторений будет иметь вид (4), где верхняя часть выражения определяет переменные v_i со значением входного состояния в начальный момент времени и как функция $\sigma_{sin,\,1}$ для $l + 1$ итерации. Нижняя часть выражения содержит условие (3) для пути от s_{in} до $l + 1$ итерации

$$r = \begin{cases} \forall v_i = \mathrm{IV} : \begin{cases} v_i(0) ::= s_{in}(v_i) \\ v_i(l+1) ::= \sigma_{s_{in},l}(s_1(v_i)) \end{cases}. \\ rc ::= \sigma_{s_{in},l}(p_1). \end{cases} \tag{4}$$

Возвращаясь к примеру на рис. 1, нам необходимо найти mop решение (1) для узла n_1, т. е. выражение пути e_1 $(e_2\ e_3)^*$ для (n_e, n_1). Исходя из значения замкнутого контекста в начальный момент $\bar{c}_e = [s, p, r] = [\{(b = \underline{b}), (d = \underline{d}), (j = \underline{j}), (m = \underline{m})\}, \text{true}, \emptyset]$, можно вычислить $\varphi(e_1\ (e_2\ e_3)^*) = (fe_3\ fe_2)\circledast\ fe_1(\bar{c}_e)$. Учитывая, что функция $fe_1(\bar{c}_e) = \bar{c}_{in} = [\{(b = \underline{b} + 1), (d = \underline{d}), (j = \underline{j}), (m = \underline{m})\}, \text{true}, \emptyset]$, можно переписать выражение в виде $(fe_3\ fe_2)\circledast$. Состояние получают из состояния путем замены символьных выражений, которые описывают значения переменных цикла $v_i \in \mathrm{IV} = \{d, j\}$ на значения соотношений цикла переменных v_i. Согласно (3) условие перехода будет иметь вид

$$p_{out} = \text{true} \wedge (0 \le l \le l_w) \wedge \Lambda_{1 \le l' \le lw} (j(l'-1) \le \underline{m}),$$

а согласно (4) система повторений имеет вид

$$r' = \begin{cases} \begin{cases} d(0) ::= d \\ d(l+1) :: 2d(l) \end{cases} \\ \begin{cases} j(0) ::= j \\ j(l+1) ::= j(l) + b + 1 \end{cases} \\ rc ::= j(l) \le m \end{cases}$$

а после упрощения – вид

$$r = \begin{cases} d(l) ::= 2^l d \\ j(l) ::= j + l(b+1). \\ rc ::= j(l) \le m. \end{cases}$$

В итоге, комбинируя выражения p_{out}, r, s_{out}, получим решение для замкнутого состояния, которое также является *mop* решением для узла n_1.

Далее рассмотрим общий случай построения повторяющейся системы в ситуации, когда цикл содержит вложенные пути, – выражение (5). Для вычисления замкнутого контекста в повторяющейся системе с вложенными путями его необходимо расширить переменной для выбора пути $\rho \in P$. Тогда, исходя из значений переменной цикла l и переменной для выбора пути, можно выбрать один путь из всего множества путей x^l_w:

$$r = \begin{cases} \forall v_i \in IV : v_i(0) = s_{in}(v_i), \\ \forall v_i \in IV : \forall [s_x, p_x, r_x] \in \overline{sc}, \\ v_i(l+1) = \sigma_{s_{in},l}(s_x(v_i)) if\, \sigma_{s_{in},l}(p_x), \\ rc ::= \vee_{1 \le x \le x_w} (\sigma_{s_{in},l}(p_x)). \end{cases} \quad (5)$$

```
1   while e > 0 loop
2     if e mod 2 = 1 then
3         y := y * f;
4     end if;
5     f := f * f;
6     e := e / 2;
7   end loop;
```

Рис. 2. Пример программы с циклом для анализа с двумя путями

Для примера рассмотрим программу на рис. 2 с двумя путями p_0 и p_1. Выражение условного перехода будет иметь вид $p_{in} \wedge (0 \le l \le l_w) \wedge (0 \le \mathfrak{p} \le x^l_w) \Lambda_{1 \le l' \le l} \mathrm{p}(l' - 1, \mathfrak{p})$. Условие $p\ (l' - 1, \mathfrak{p})$ вычисляется как $\sigma_{s_{in},(l'-1)}(p_k)$, где индекс $k = (\mathfrak{p}\ x_w^{-(l-1')})$ *mod* x_w. Рассмотрим следующий путь в программе: $x_w = 2$, $l = 3$, $l' = \{1, 2, 3\}$, $k = \{1, 0, 1\}$, $p(0, 5) = \sigma_{s_{in},0}(p_1) = e(0) > 0 \wedge e(0)$ *mod* 2 != 1, $p(1, 5) = \sigma_{s_{in},1}(p_0) = e(1) > 0 \wedge e(1)$ *mod* 2 != 1, $p(2, 5) = \sigma_{s_{in},2}(p_1) = e(2) > 0 \wedge e(2)$ *mod* 2 != 1.

2. АНАЛИЗ И ПРЕДЛОЖЕНИЯ

В данном разделе представлен анализ вышепредлагаемых методов и подходов к проверке ПО на наличие ошибок, а также приведены некоторые рекомендации по их устранению. Описание выражений диапазона в виде $0 \le l \le l_w$ может применяться не для всех циклов, т. е. в цикле может выполняться не-

равномерное приращение переменной цикла (не i++ для каждой итерации, а, например, $i > 10$? i++ : i+=2 или еще более непредсказуемый вариант: $i = \text{rand}(0, l_w)$).

Второй проблемой является то, что введенный оператор $\circledast$ не уменьшает количество путей до конечного числа. В примере на рис. 2 число путей увеличивается в два раза на каждой итерации. При «удачно» подобранных значениях легко добиться роста числа вариантов до очень больших значений в цикле из семи строк. Также можно подобрать пример цикла, возможно, с вложенными циклами, где такой рост будет неограничен.

Еще одна проблема, решение которой было предложено в [4, 5], это описание массивов. Безусловно, запись вида v_i предполагает описание всех переменных, в том числе и в массиве, но не учитывает особенности массивов: возможности сравнения с учетом индекса, т. е. в виде $a[i] > 0$ или $a[i + 1] > b[j]$.

Данные недостатки были частично рассмотрены в работах [4, 5]. Там же были предложены варианты решения этих проблем, которые должны быть переработаны с учетом используемого в [1] математического аппарата. Но тем не менее в работах [4, 5] предлагаются подходы, которые могут устранить вышеприведенные недостатки.

ВЫВОД

В данной работе был рассмотрен метод [1] символьного анализа программы, в котором предлагается способ приведения потенциально бесконечного множества путей в программе к конечному. Для того чтобы понять суть предлагаемого подхода, в данной работе был рассмотрен математический аппарат символьного описания программ. Данный аппарат должен быть применен для более формального описания идей и подходов в работах [4, 5], которые являются продолжением и улучшением идей из работы [1].

Существенным недостатком, по мнению авторов работы [1], является то, что область применения заявленного оператора для приведения потенциально бесконечного количества путей к конечному существенного ограничена и не может быть применена на практике. Также к недостаткам следует отнести то, что в работе приведены примеры преобразования тривиальных программ в символьный домен, по которым сложно судить о предлагаемом подходе.

СПИСОК ЛИТЕРАТУРЫ

1. *Burgstaller B., Scholz B., Blieberger J.* A symbolic analysis framework for static analysis of imperative programming languages // Journal of Systems and Software. – 2012. – Vol. 85, iss. 6. – P. 1418–1439. – doi: 10.1016/j.jss.2011.11.1039.

2. *Tarjan R.E.* A unified approach to path problems // Journal of the ACM (JACM). – 1981. – Vol. 28, iss. 3. – P. 577–593. – doi: 10.1145/322261.322272.

3. Hecht M.S. Flow analysis of computer programs. – New York: Elsevier: North Holland, 1977. – 232 p.

4. *Воевода А.А., Романников Д.О.* О методе анализа программ // Сборник научных трудов НГТУ. – 2014. – № 4 (78). – С. 125–138. – doi: 10.17212/2307-6879-2014-4-125-138.

5. *Воевода А.А., Романников Д.О.* Способы представления программ и их анализ // Сборник научных трудов НГТУ. – 2014. – № 3 (76). – С. 81–98.

6. *Романников Д.О.* О поиске входных интервалов // Сборник научных трудов НГТУ. – 2014. – № 1 (75). – С. 140–145.

7. Abramsky S., Hankin C. An introduction to abstract interpretation [Electronic resource]. – URL: https://www.cs.virginia.edu/~weimer/2007-615/reading/AbramskiAI.pdf (accessed 20.04.2015).

8. *Cousot P., Cousot R.* A gentle introduction to formal verification of computer systems by abstract interpretation // Logics and Languages for Reliability and Security. – Amsterdam: IOS Press, 2010. – P. 1–29. – (NATO Science Series – D: Information and Communication Security; vol. 25). – doi: 10.3233/978-1-60750-100-8-1.

9. *Глухих М.И., Ицыксон В.М., Цесько В.А.* Использование зависимостей для повышения точности статического анализа программ // Моделирование и анализ информационных систем. – 2011. – Т. 18, № 4. – С. 68–79.

10. *Bush W.R., Pincus J.D., Sielaff D.J.* A static analyzer for finding dynamic programming errors // Software: Practice and Experience. – 2000. – Vol. 30, iss. 7. – P. 775–802. – doi: 10.1002/(SICI)1097-024X(200006)30:7<775::AID-SPE309>3.0.CO;2-H.

11. *Кларк Э., Грамберг О., Пелед Д.* Верификация моделей программ. Model Checking: пер. с англ. – М.: МЦНМО, 2002. – 416 с.

12. *Карпов Ю.Г.* Model Checking. Верификация параллельных и распределенных программных систем. – СПб.: БХВ-Петербург, 2010. – 560 с.

13. *Horwitz S., Reps T., Binkley D.* Interprocedural slicing using dependence graphs // ACM Transactions on Programming Languages and Systems. – 1990. – Vol. 12, iss. 1. – P. 26–60. – doi: 10.1145/77606.77608.

14. *Ferrante J., Ottenstein K.J., Warren J.D.* The program dependence graph and its use in optimization // ACM Transactions on Programming Languages and Systems (TOPLAS). – 1987. – Vol. 9, iss. 3. – 1987. – P. 319–349. – doi: 10.1145/24039.24041.

15. *Воевода А.А., Марков А.В., Романников Д.О.* Разработка программного обеспечения: проектирование с использованием UML диаграмм и сетей Петри на примере АСУ ТП водонапорной станции // Труды СПИИРАН. – 2014. – Вып. 3 (34). – С. 218–231.

16. *Романников Д.О.* Нахождение ошибок обращения к несуществующим элементам массива на основании результатов анализа сети Петри // Сборник научных трудов НГТУ. – 2012. – № 1 (67). – С. 115–120.

17. *Марков А.В., Романников Д.О.* Алгоритм трансляции диаграммы активности в сеть Петри // Доклады Академии наук высшей школы Российской Федерации. – 2014. – № 1 (22). – С. 104–112.

18. *Воевода А.А., Романников Д.О.* Редуцирование пространства состояний сети Петри для объектов из одного класса // Научный вестник НГТУ. – 2011. – № 4 (45). – С. 146–150.

19. *Романников Д.О., Марков А.В., Зимаев И.В.* Обзор работ, посвященных разработке ПО с использованием UML и сетей Петри // Сборник научных трудов НГТУ. – 2011. – № 1 (63). – С. 91–104.

20. *Марков А.В.* Анализ отдельных частей дерева достижимости сетей Петри // Сборник научных трудов НГТУ. – 2013. – № 3 (73). – С. 58–74.

21. *Воевода А.А., Марков А.В.* Рекурсия в сетях Петри // Сборник научных трудов НГТУ. – 2012. – № 3 (69). – С. 115–122.

22. *Воевода А.А., Марков А.В.* Тестирование UML-диаграмм с помощью аппарата сетей Петри на примере разработки по для игры "Змейка" // Сборник научных трудов НГТУ. – 2010. – № 3 (61). – С. 51–60.

23. *Воевода А.А., Зимаев И.В.* Верификация workflow-моделей с применением сетей Петри // Научный вестник НГТУ. – 2010. – № 4 (41). – С. 151–154.

24. *Воевода А.А., Саркенов Д.О.,* Хассоунех В. Моделирование протоколов с учетом времени на цветных сетях Петри // Сборник научных трудов НГТУ. – 2004. – № 3 (37). – С. 133–136.

Романников Дмитрий Олегович, кандидат технических наук, доцент кафедры автоматики НГТУ. Основные направления научных исследований: формальная верификация, проверка моделей. Имеет 34 публикации. E-mail: rom2006@gmail.ru

The analysis of a symbolic framework for static analysis of imperative programming languages[*]

D.O. Romannikov

Novosibirsk State Technical University, 20 K. Marks prospekt, Novosibirsk, 630073, Russian Federation, PhD (Eng.), associate professor of the automation department. E-mail: rom2006@gmail.ru

This paper is devoted of analysis of a work of Burgstaller B., Scholz B. and Blieberger J. about program transformation into symbolic domain for future analysis. There is mathematical apparatus for a program description in the terms of symbolic computation in the paper: au-

[*] *Received 20 January 2015.*

thors suggest using terms of context, program state, and super-context for formulas program description. There are examples of transformation of a program to symbolic domain. Proposed approach of loops description is a main achievement of this work. That way allows getting formal cycle representation and reducing potentially infinite number of ways to the end of the program.
Examples of operation of the original source are analyzed, but, and this is certainly a drawback of the original work, all the examples are quite simple, and does not contain more than 8 lines, which ultimately leads to that it is impossible to draw a conclusion about the applicability of the method.
During the analysis of the study work has identified a number of shortcomings: 1) the method of "folding" the number of paths in a loop does not result in the claimed if the cycle is branching; 2) is not obvious how it will look proposed a formal description of the cycles in the case of nested loops. And in cycles with a large number of nesting should be based on the iterable variables, ie expression of the form: a[i][j] = b[i][j] or the like; 3) there is no description of the character representation for arrays and is especially important to iterate over the array elements in cycles in the form of a[i].

Keywords: software, testing, input intervals, formal verification, dynamic verification, verification, model checking, software models, graphs, total correctness of programs

DOI: 10.17212/2307-6879-2015-1-105-116

REFERENCES

1. Burgstaller B., Scholz B., Blieberger J. A symbolic analysis framework for static analysis of imperative programming languages. *Journal of Systems and Software*, 2012, vol. 85, iss. 6, pp. 1418–1439. doi: 10.1016/j.jss.2011.11.1039

2. Tarjan R.E. A unified approach to path problems. *Journal of the ACM (JACM)*, 1981, vol. 28, iss. 3, pp. 577–593. doi: 10.1145/322261.322272

3. Hecht M.S. *Flow analysis of compute programs*. New York, Elsevier, North Holland, 1977. 232 p.

4. Voevoda A.A., Romannikov D.O. O metode analiza programm [About the method of program analysis]. *Sbornik nauchnyh trudov Novosibirskogo gosudarstvennogo tekhnicheskogo universiteta – Transaction of scientific papers of the Novosibirsk state technical university*, 2014, no. 4 (78), pp. 125–138. doi: 10.17212/2307-6879-2014-4-125-138

5. Voevoda A.A., Romannikov D.O. Sposoby predstavleniya programm i ikh analiz [Methods of program representation and analysis]. *Sbornik nauchnyh trudov Novosibirskogo gosudarstvennogo tekhnicheskogo universiteta – Transaction of scientific papers of the Novosibirsk state technical university*, 2014, no. 3 (76), pp. 81–98.

6. Romannikov D.O. O poiske vkhodnykh intervalov [On the search for input intervals]. *Sbornik nauchnyh trudov Novosibirskogo gosudarstvennogo tekhnicheskogo universiteta – Transaction of scientific papers of the Novosibirsk state technical university*, 2014, no. 1 (75), pp. 140–145.

7. Abramsky S., Hankin C. *An introduction to abstract interpretation*. Available at: https://www.cs.virginia.edu/~weimer/2007-615/reading/AbramskiAI.pdf (accessed 20.04.2015)

8. Cousot P., Cousot R. A gentle introduction to formal verification of computer systems by abstract interpretation. *NATO Science Series – D: Information and Communication Security*. Vol. 25: *Logics and Languages for Reliability and Security*. Amsterdam, IOS Press, 2010, pp. 1–29. doi: 10.3233/978-1-60750-100-8-1.

9. Glukhikh M.I., Itsykson V.M., Tses'ko V.A. Ispol'zovanie zavisimostei dlya povysheniya tochnosti staticheskogo analiza programm [Using dependencies to improve precision of code analysis]. *Modelirovanie i analiz informatsionnykh sistem – Modeling and Analisys of Information System*, 2011, vol. 18, no. 4, pp. 68–79.

10. Bush W.R., Pincus J.D., Sielaff D.J. A static analyzer for finding dynamic programming errors. *Software: Practice and Experience*, 2000, vol. 30, iss. 7, pp. 775–802. doi: 10.1002/(SICI)1097-024X(200006)30:7<775::AID-SPE309>3.0.CO;2-H

11. Clarke E.M., Grumberg O., Peled D. *Model checking*. Cambridge, London, MIT Press, 2001 (Russ. ed.: Klark E., Gramberg O., Peled D. *Verifikatsiya modelei programm: Model checking*. Moscow, MTsNMO Publ., 2002. 416 p.).

12. Karpov Yu.G. Model Checking. Verifikatsiya parallel'nykh i raspredelennykh programmnykh sistem [Model Checking. Verification of parallel and distributed software systems]. St. Petersburg, BHV-Petersburg Publ., 2010. 560 p.

13. Horwitz S., Reps T., Binkley D. Interprocedural slicing using dependence graphs. *ACM Transactions on Programming Languages and Systems*, 1990, vol. 12, iss. 1, pp. 26–60. doi: 10.1145/77606.77608

14. Ferrante J., Ottenstein K.J., Warren J.D. The program dependence graph and its use in optimization. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 1987, vol. 9, iss. 3, pp. 319–349. doi: 10.1145/24039.24041

15. Voevoda A.A., Markov A.V., Romannikov D.O. Razrabotka programmnogo obespecheniya: proektirovanie s ispol'zovaniem UML diagramm i setei Petri na primere ASU TP vodonapornoi stantsii [Software development: software design using UML diagrams and Petri nets for example automated process control system of pumping station]. *Trudy SPIIRAN – SPIIRAS Proceedings*, 2014, iss. 3 (34), pp. 218–231.

16. Romannikov D.O. Nakhozhdenie oshibok obrashcheniya k nesushchestvuyushchim elementam massiva na osnovanii rezul'tatov analiza seti Petri [Finding errors or nonexistent array's elements based on the results of the analysis Petri nets]. *Sbornik nauchnyh trudov Novosibirskogo gosudarstvennogo tekhnicheskogo universiteta – Transaction of scientific papers of the Novosibirsk state technical university*, 2012, no. 1 (67), pp. 115–120.

17. Markov A.V., Romannikov D.O. Algoritm translyatsii diagrammy aktivnosti v set' Petri [Algorithm of automatic conversion of the activity diagram into petri-net structure formats]. *Doklady Akademii nauk vysshei shkoly Rossiiskoi Federatsii – Proceedings of the Russian higher school Academy of sciences*, 2014, no. 1 (22), pp. 104–112.

18. Voevoda A.A., Romannikov D.O. Redutsirovanie prostranstva sostoyanii seti Petri dlya ob"ektov iz odnogo klassa [Reducing the state space of Petri nets for objects of one class]. *Nauchnyi vestnik Novosibirskogo gosudarstvennogo tekhnicheskogo universiteta – Science bulletin of the Novosibirsk state technical university*, 2011, no. 4 (45), pp. 146–150.

19. Romannikov D.O., Markov A.V., Zimaev I.V. Obzor rabot, posvyashchennykh razrabotke PO s ispol'zovaniem UML i setei Petri [The review of works devoted to development of the software with use UML and Petri nets]. *Sbornik nauchnyh trudov Novosibirskogo gosudarstvennogo tekhnicheskogo universiteta – Transaction of scientific papers of the Novosibirsk state technical university*, 2011, no. 1 (63), pp. 91–104.

20. Markov A.V. Analiz otdel'nykh chastei dereva dostizhimosti setei Petri [Analysis of individual pieces of wood reachability Petri nets]. *Sbornik nauchnyh trudov Novosibirskogo gosudarstvennogo tekhnicheskogo universiteta – Transaction of scientific papers of the Novosibirsk state technical university*, 2013, no. 3 (73), pp. 58–74.

21. Voevoda A.A., Markov A.V. Rekursiya v setyakh Petri [The concepts recursion in Petri net]. *Sbornik nauchnyh trudov Novosibirskogo gosudarstvennogo tekhnicheskogo universiteta – Transaction of scientific papers of the Novosibirsk state technical university*, 2012, no. 3 (69), pp. 115–122.

22. Voevoda A.A., Markov A.V. Testirovanie UML-diagramm s pomoshch'yu apparata setei Petri na primere razrabotki po dlya igry "Zmeika" [About testing UML-diagrams by means of the device of Petri nets on the example of software engineering for game "Snake"]. *Sbornik nauchnyh trudov Novosibirskogo gosudarstvennogo tekhnicheskogo universiteta – Transaction of scientific papers of the Novosibirsk state technical university*, 2010, no. 3 (61), pp. 51–60.

23. Voevoda A.A., Zimaev I.V. Verifikatsiya workflow-modelei s primeneniem setei Petri [Verification of workflow-models using Petri-nets]. *Nauchnyi vestnik Novosibirskogo gosudarstvennogo tekhnicheskogo universiteta – Science bulletin of the Novosibirsk state technical university*, 2010, no. 4 (41), pp. 151–154.

24. Voevoda A.A., Sarkenov D.O., Khassounekh V. Modelirovanie protokolov s uchetom vremeni na tsvetnykh setyakh Petri [Protocol modeling with regards of time on colored Petri nets] *Sbornik nauchnyh trudov Novosibirskogo gosudarstvennogo tekhnicheskogo universiteta – Transaction of scientific papers of the Novosibirsk state technical university*, 2004, no. 3 (37), pp. 133–136.