УДК 51.681.3

ПОШУК ІНВАРІАНТІВ U-Y-ПРОГРАМ ІТЕРАЦІЙНИМ АЛГОРИТМОМ НАД АБСОЛЮТНО ВІЛЬНИМИ АЛГЕБРАМИ ДАННИХ

О.М. Максимець

Київський національний університет ім. Тараса Шевченка,
03680, м. Київ, проспект Глушкова, 2, корпус 6,
E-mail: maksymets@gmail.com

Розглядається проблема пошуку програмних інваріантів у програмах над вільними алгебрами даних. Для отримання більш релевантних результатів при верифікації програмного забезпечення сучасним «пруверам» необхідні інваріанти програм. Представлено реалізацію ітераційного алгоритму генерації інваріантів на абсолютно вільних алгебрах. Показано, що алгоритм генерує релевантні інваріанти для відповідної вільної алгебри.

The problem of generating program invariants on free algebras is considered. Modern provers need program invariants as input to get more relevant results for software verification. Implementation of iterative algorithm for generating invariants on absolutely free algebras is presented. We show that the algorithm generates relevant invariants corresponding to free algebras.

Вступ

В стані програми інваріант являє собою твердження щодо змінних програми, яке є вірним завжди, коли процес обчислень потрапляє у цей стан. Будь-який прогрес у проблемах пошуку інваріантів є важливим для перевірки правильності програм у теорії верифікації [1]. Пошук інваріантів є полем для застосування алгоритмів на алгебрах, автоматичних «пруверів», абстрактних методів представлення і перевірки моделей.

Особливої уваги заслуговує задача пошуку інваріантів для циклів. Розв'язок цієї задачі дозволяє знаходити вічні цикли на етапі компіляції і перевіряти більш складні програми [2].

У даній роботі розглянута узагальнена задача пошуку інваріантів, не фокусуючись на інваріантах циклів. Вибрана мова виразів не дозволяє покрити множину виразів, які використовуються в сучасних мовах програмування, але є наближенням до них. Новизною цієї роботи є реалізація ітераційного алгоритму [3] і демонстрація його на прикладі.

Визначення

Розглянемо U-Y програму на пам'яті змінних $R = \{r_1,...,r_m\}$, які визначені на алгебрі даних (D,Ω). $K(D,\Omega)$ є клас алгебр, який включає в себе алгебру (D,Ω) і визначається набором рівностей Eq [4]. $T(\Omega,R)$ є вільною алгеброю термів, на R з класу $K(D,\Omega)$. Ми розглядаємо абсолютно вільну алгебру, тому множина відомих співвідношень є пустою $Eq = \varnothing$.

Нехай $A = \{a_0, a_1,..., a_*\}$ – множина вершин U-Y програми. $N_{a0}, N_{a1},..., N_{a*}$ – множини базисів співвідношень, які є вірними в відповідній вершині програми на поточному кроці виконання методу.

Розглядається задача пошуку інваріантів на мові L. L складається з термів визначених як рівності виду $r_i := h(r)$, де $h(r) \in T(\Omega,R)$, $r = (r_1,...,r_m)$.

Для демонстрації структур даних і алгоритму розглянемо приклад програми, яка обчислює суму $\sum_{y=1}^{n} \frac{\sum_{z=1}^{m-1} z^y}{m^y}$ по вхідних цілих m, n. Одна з U-Y програм, яка розв'язує цю задачу (рис. 1).

Ребро U-Y програми складаються з умов переходу u, які знаходяться зліва від ребра на рис. 1 і операторів присвоєння y, що знаходяться справа від ребра. N_a позначає базис співвідношень для вершини програми a, який справджується на конкретному кроці алгоритму.

Розглядається метод верхньої апроксимації (МВА) [5] для генерації інваріантів на вище наведеному прикладі. МВА ігнорує умови u на ребрах, а ті які мають вигляд $r_i = h(r)$ переносяться в присвоєння y цього ребра у вигляді $r_i := h(r)$. Таким чином, ребро між вершинами a_3 і a_4, яке складається з $u = (z = m)$ і $y = (x := m \wedge y, sum_2 := sum_2 + sum_1 / x)$ буде враховано, як $y = (x := m \wedge y, sum_2 := sum_2 + sum_1 / x, z := m)$.

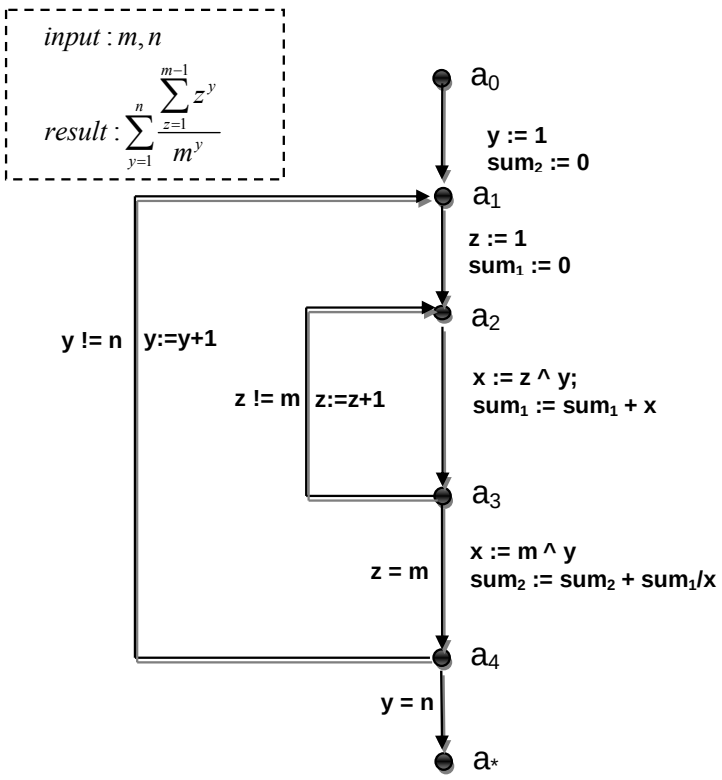

Рис. 1. Приклад U-Y програми

Метод використовує три нетривіальні операції над базисами: перетин, ефект $ef\left(N_a, y\right)$ і порівняння. Операція перетину $N_A \bigcap N_B$ базисів будує найбільший базис простору, який є підпростором N_A і N_B. Оператор $ef\left(N_a, y\right)$ за базисом співвідношень N_a, які вірні перед виконанням операторів присвоювання y, будує базис співвідношень, які вірні на перетвореному оператором y стані пам'яті. Оператор порівняння двох базисів співвідношень $N_A <> N_B$ перевіряє чи збігаються простори співвідношень описані цими базисами, враховуючи, що в базисах можуть використовуватися різні набори змінних. Більш детально ці операції і їхня оцінка складності описані в [6].

Початкове наближення

За означенням U-Y програма має тільки одну початкову вершину a_0. На початкову вершину може подаватися вхідний базис співвідношень N_{a0}, який є правильним до виконання програми. Решту вершин мають порожні базиси співвідношень $N_{a1},...,N_{a*}$. Для розглядуваного прикладу $N_{a0} = \varnothing$. Метод називається методом верхньої апроксимації тому, що ми будуємо початкові базиси співвідношень, які з кожним кроком звужуються. Початкове наближення цих базисів і є верхньою границею. Початкове наближення отримується в результаті виконання першого етапу методу.

Далі приведена послідовність дій виконання першого етапу МВА (рис. 2) для пошуку початкового наближення:

```
Na0 := N0
ToVisit.push(a0)
Visited := {}
while (ToVisit <> ∅)
        c := ToVisit.pop()
        Visited := Visited + c
        for every (c, y, a')
                if Not a' in Visited
                        Na':=ef(Nc, y)
                        ToVisit.push(a')
```

Рис. 2. Перший етап МВА

На кожному кроці зберігається стек вершин, які потрібно відвідати *ToVisit* і множину, які вже були відвідані *Visited*. На першому кроці методу стек *ToVisit* складається з a_0, а множина *Visited* порожня. На кожній ітерації методу розглядається одна з вершин стеку *ToVisit* c. Розглядаються всі ребра (c, y, a'), які виходять з цієї вершини і присвоюються $N_{a'}$ результуючий базис ефекту $ef(N_c, y)$. Додаємо a' до множини *ToVisit* для розгляду на наступних кроках. Етап закінчується, коли множина розглядуваних вершин *ToVisit* є порожньою.

Проведемо обчислення методу для розглянутого прикладу U-Y програми:

Беремо вершину a_0 з стеку *ToVisit*, розглянемо ребро $(a_c, (sum2 := 0; y := 1), a_1)$, $N_1 := ef(N_0, (sum2 := 0; y := 1)) := (sum2 := 0; y := 1)$, бо $N_0 = \varnothing$.

Додаємо a_0 до множини *Visited*, a_1 до стеку *ToVisit*.

Беремо вершину a_1 з стеку *ToVisit*, розглянемо ребро $(a_1, (sum1 := 0; z := 1), a_2)$:

$N_1 := (sum2 := 0; y := 1)$,

$N_2 := ef(N_1, (sum1 := 0; z := 1)) := (sum2 := 0; sum1 := sum2; z := 1; y := z)$.

Додаємо a_1 до множини *Visited*, a_2 до стеку *ToVisit*.

Беремо вершину a_2 з стеку *ToVisit*, розглянемо ребро $(a_2, (sum1 := sum1 + z \wedge y; x := z \wedge y), a_3)$.

Продовжуючи виконання далі за методом отримуємо наступні базиси співвідношень, що приведені в табл. 1.

Таблиця 1. Базиси початкового наближення

Вершина	**Базис співвідношень**
N_{a0}	$\varnothing$
N_{a1}	$sum2 := 0$ $y := 1$
N_{a2}	$sum2 := 0$ $sum1 := sum2$ $z := 1$ $y := z$
N_{a3}	$sum2 := 0$ $sum1 := sum2 + x$ $z := 1$ $y := z$ $x := z \wedge z$
N_{a4}	$sum2 := 0 + sum1$ $sum1 := 0 + z \wedge z$ $z := m$ $y := 1$ $x := z \wedge y$
N_{a*}	$sum2 := 0 + 0 + z \wedge z$ $sum1 := 0 + z \wedge z$ $z := m$ $y := n$ $x := z \wedge 1$

Етап стабілізації розв'язку

На кожному кроці зберігається множина вершин, які потрібно відвідати *ToVisit*. На першому кроці методу множина *ToVisit* складається з усіх вершин множини A окрім a_0. На кожній ітерації методу розглядається одна з вершин стеку *ToVisit* c. Базис поточної множини співвідношень присвоюється базис множини співвідношень вершини $c: N := N_c$. Розглядаються всі ребра (a', y, c), які входять в вершину c і для кожного ребра поточний базис множини співвідношень N перетинається з базисом $ef(N_c, y)$. Після проходу по всім перерахованим ребрам $N <> N_c$, то множина *ToVisit* поповнюється всіма вершинами, для яких є

ребра, які ведуть з c: $\{a \mid (c, y, a) \in S\}$, де S – множина ребер U-Y програми. Етап методу закінчує виконання, якщо множина розглядуваних вершин *ToVisit* є порожньою.

Послідовність дій виконання другого етапу МВА (рис. 3).

```
ToVisit := A\{a0}
while (ToVisit <> ∅)
    c := take from ToVisit
        if (Nc <> ∅)
                N := Nc
                for every (a', y, c)
                        N := N ⋂ ef(Nc, y)
                if (N <> Nc)
                        Nc := N
                        ToVisit := ToVisit + {a | for every (c, y, a)}
```

Рис. 3. Другий етап МВА

Розглянемо виконання 2-го етапу методу (рис. 3) на розглядуваному прикладі програми (рис. 1). Множина *ToVisit* містить $\{a_1, a_2, a_3, a_4, a_*\}$.

Розглянемо вершину a_1, $N := N_1$.

Розглянемо ребро $(0, (sum2 := 0; y := 1), 1)$:

$ef(N_0, (sum2 := 0; y := 1)) = (sum2 := 0; y := 1)$.

Присвоюємо:

$N := (sum2 := 0; y := 1) \cap (sum2 := 0; y := 1) := (sum2 := 0; y := 1)$

Розглянемо ребро $(a_4, y := y + 1, a_1)$:

$N_4 := (sum2 := 0 + sum1; sum1 := 0 + z \wedge z; z := m; y := 1; x := z \wedge y)$,

$ef(N_4, y := y + 1) = (sum2 := 0 + 0 + z \wedge z; sum1 := 0 + z \wedge z; z := m; y := 1 + 1; x := z \wedge 1)$.

Присвоюємо:

$N := (sum2 := 0; y := 1) \bigcap (sum2 := 0 + 0 + z \wedge z; sum1 := 0 + z \wedge z; z := m; y := 1 + 1; x := z \wedge 1) = \varnothing$.

Базис $N_1 <> N$, тому до розгляду додаються, вершини до яких є ребра з вершини a_1: $ToVisit := ToVisit + \{a_2\}$.

Вершина a_2 уже знаходиться в *ToVisit*.

Розглянемо вершину a_2, $N := N_2$.

Розглянемо ребро $(1, (sum1 := 0; z := 1), 2)$:

$ef(N_1, sum1 := 0, z := 1) = (sum1 := 0; z := 1)$,

$N_2 := (sum2 := 0; sum1 := sum2; z := 1; y := z) \bigcap (sum1 := 0; z := 1) := (sum1 := 0; z := 1)$.

Розглянемо ребро $(3, z := z + 1, 2)$:

$N_3 := (sum2 := 0; sum1 := sum2 + x; z := 1; y := z; x := z \wedge z)$,

$ef(N_3, z := z + 1) = (sum2 := 0; sum1 := sum2 + z \wedge z; z := y + y; y := 1; x := y \wedge y)$,

$N_2 := (sum1 := 0; z := 1) \bigcap (sum2 := 0; sum1 := sum2 + z \wedge z; z := y + y; y := 1; x := y \wedge y) := \varnothing$.

Базис $N_2 <> N$, тому до розгляду додаються, вершини до яких є ребра з вершини a_2: $ToVisit := ToVisit + \{a_3\}$.

Вершина a_3 уже знаходиться в *ToVisit*.

Розглянемо вершину a_3, $N := N_3$.

Розглянемо ребро $(2, sum1 := sum1 + z \wedge y; z := z \wedge y, 3)$:

$ef(N_2, sum1 := sum1 + z \wedge y; x := z \wedge y) = (sum1 := sum1 + x; x := z \wedge y)$

Присвоюємо:

$N := (sum2 := 0; sum1 := sum2 + x; z := 1; y := z; x := z \wedge z) \bigcap (sum1 := sum1 + x; x := z \wedge y) = (x := z \wedge y)$

Базис $N_3 <> N$, тому до розгляду додаються, вершини до яких є ребра з вершини a_3: $ToVisit := ToVisit + \{a_2, a_4\}$. Вершина a_4 уже знаходиться в *ToVisit*, а a_2 ні.

Розглянемо вершину a_2, $N := N_2$.

Так як $N_2 := \varnothing$ переходимо до наступної вершини.

Розглянемо вершину a_4, $N := N_4$.

Розглянемо ребро $(2, sum1 := sum1 + z \wedge y; x := z \wedge y, 4)$:

$ef(N_3, sum2 := sum2 + sum1; z := m; x := m \wedge y) = (sum2 := sum2 + sum1 + x; z := m; x := z \wedge y)$

Присвоюємо:

$N := (sum2 := 0 + sum1; sum1 := 0 + z \wedge z; z := m; y := 1; x := z \wedge y) \bigcap (sum2 := sum2 + sum1; z := m; x := z \wedge y) =$

$= (z := m; x := z \wedge y)$.

Базис $N_4 <> N$, тому додаємо до розгляду вершини до яких є виходи з вершини N_4: $ToVisit := ToVisit + \{a_1, a_5\}$. Вершина a_5 уже знаходиться в $ToVisit$, а a_1 буде додана.

Розглянемо вершину a_2, $N := N_1$.

Так як $N_1 := \varnothing$ переходимо до наступної вершини.

Розглянемо вершину a_5, $N := N_5$. Розглянемо ребро $(4, y := n, 5)$:

$ef(N_4, y := n) = (z := m, y := n; x := z \wedge y)$.

Присвоюємо:

$N := (sum2 := 0 + 0 + z \wedge z; sum1 := 0 + z \wedge z; z := m; y := n; x := z \wedge 1) \bigcap (z := m; y := n; x := z \wedge y) = (z := m; y := n)$.

Базис $N_5 <> N$, але з вершини a_5 немає виходів, тому що ця вершина кінцева.

Множина $ToVisit$ порожня, робота методу закінчилась.

Результатом виконання методу є наступна множина базисів інваріантів (табл. 2).

Таблиця 2. Базиси інваріантів

Вершина	Базис інваріантів
N_{a0}	$\varnothing$
N_{a1}	$\varnothing$
N_{a2}	$\varnothing$
N_{a3}	$x = z \wedge y$
N_{a4}	$x = z \wedge y$ $z = m$
N_{a*}	$y = n$ $z = m$

Індуктивне доведення інваріантів

Як бачимо з прикладу вище отримати доведення вірності U-Y програми не вдається, оскільки з базису інваріантів неможливо вивести вираз, який обчислює програма.

Розглянемо приклад програми, яка обчислює добуток матриць $C := A \times B$, де $c(i,j) = \sum_{r=1}^{n} a(i,r) * b(r,j)$:

```
for(i := 1; n)
    for(j := 1; n)
        for(l := 1; n)
            c(i, j) := c(i, j) + a(i, l) * b(l, j)
```

Рис. 4. Приклад програми множення матриць

МВА на даному прикладі не вдається отримати інших інваріантів, окрім умов виходу з циклів. Скористаємося аналітичним методом для доведення інваріантів. Методом математичної індукції доведемо, що в рядку 4, гнізді циклів, виконується інваріант $c(i,j) = \sum_{r=1}^{l} a(i,r) * b(r,j)$.

Для $l = 1$ твердження індукції виконується $c(i,j) = a(i,1) \cdot b(1,j)$.

Для $l = 2$ твердження індукції виконується $c(i,j) = a(i,1) \cdot b(1,j) + a(i,2) \cdot b(2,j)$.

Нехай твердження виконується на кроці $l = k$: $c(i,j) = \sum_{r=1}^{k} a(i,r) * b(r,j)$. Розглянемо співвідношення на кроці $l = k+1$: $c(i,j) = \sum_{r=1}^{k} a(i,r) \cdot b(r,j) + a(i,k+1) \cdot b(k+1,j) = \sum_{r=1}^{k+1} a(i,r) \cdot b(r,j)$.

Таким чином, на всіх кроках $l = 1..n$ в рядку 4 виконується $c(i,j) = \sum_{r=1}^{l} a(i,r) * b(r,j)$. Отже, після закінчення виконання циклу за умовою його закінчення $l = n$ і справджується $c(i,j) = \sum_{r=1}^{n} a(i,r) * b(r,j)$, що і доводить правильність програми.

Одна з особливостей МВА – це можливість при генерації інваріантів ураховувати більш складні інваріанти, виведені експертом, наприклад, методом математичної індукції. У такому випадку ці інваріанти додаються до вхідного базису співвідношень N_{a0}.

Висновки

У даній роботі описується реалізація алгоритму МВА [1]. Представлений результат виконання генерації інваріантів програмою, яка реалізує метод верхньої апроксимації для U-Y програм над вільною алгеброю.

Отримана множина базисів інваріантів може слугувати вхідними параметрами для «прувера», який перевіряє правильність програми. Оскільки в вершині a_3 виконується основна операція сумування $sum1 := sum1 + x$, то інваріант $x = z \wedge y$ вказує на те, що сума рахується правильно. У вершині a_4 виконується операцію ділення на x і справджуються умови $x = z \wedge y$ і $z = m$, тобто ділення правильне щодо постановки задачі. Інваріанти в вершині a_5 надають змогу перевірити умови виходу з програми.

Змінюючи три операції: порівняння, перетину базисів і ефект ef, які використовуються в методі, відповідно до алгебри даних U-Y програми, можна отримати більш складні інваріанти. В наступних працях планується розширити алгебру до алгебри лінійних многочленів і дослідити паралелізацію ітераційних методів [7].

1. *Hoare T.* The Verifying Compiler: A Grand Challenge for Computing Research // Journal of the ACM.– 2003.– N. 50(1). – P. 63–69.
2. *Kovacs L., Voronkov A.* Finding loop invariants for programs over arrays using a theorem prover. Proceedings of the 12th International Conference on Fundamental Approaches to Software Engineering of Lecture Notes in Computer Science, Springer.– 2009. – Vol. 5503.– P. 470.
3. *Кривий С.Л.* Проблеми розробки програмного забезпечення. Формалізація, модель, алгоритми, програма, верифікація.– К., 2010.
4. *Maksymets O.M.* Application of minimization algorithm for finite acyclic automata in finding condition's basis for program invariant search // Proceedings of International Conference on Theoretical and Applied Aspects of Cybernetics. Informatics and Computer Science Section.– 2011.– P. 35–37.
5. *Годлевский А.Б., Капитонова Ю.В., Кривой С.Л., Летичевский А.А.* Итеративные методы анализа программ // Кибернетика.– 1989.– № 2.– С. 9–19.
6. *Годлевский А.Б., Кривой С.Л.* Трансформационный синтез эффективных алгоритмов с учетом дополнительной информации // Кибернетика.– 1989.– № 2.– С. 9–19.
7. *Кривой С.Л., Ракша С.Г.* Поиск инвариантных линейных зависимостей в программах // Кибернетика.– 1984.– № 6.– С. 23–28.