Sposób weryfikacji pamięci programów

Magda Zakrzewska

Wstęp

- Sprawdzanie jak zachowuje się pamięć programu: np. wycieki
- Trzeba zamodelować nieskończone sekwencje stanów
- Trzeba brać tylko podzbiór stanów
- Pomysł: skorzystanie z własności WQO
- Intuicja: tylko skończenie wiele kroków algorytmu

Co można sprawdzać?

- Tylko programy, które mają struktury danych z jednym następnikiem
- Listy jednokierunkowe; zacyklone. Mogą dzielić swoje części
- Można się zastanawiać nad uogólnieniem

Reprezentacja

- Pamięć to graf
- Wierzchołki – komórki pamięci
- Następnik wierzchołka(.next) – to wierzchołek wskazywany przez obecny wierzchołek. Co najwyżej jeden
- Skończony zbiór wskaźników – zmienne
- Wskazują na komórki

Reprezentacja cd.

- X – skończony zbiór zmiennych z nullem
- P = (Q,T)
- Q – stany
- T – tranzycje (q1,a,q2)

Graf

- *graf g* = (*V, succ, λ*)
- *V – skończony zbiór wierzchołków*
- *succ – funkcja częściowa z V do V*
- *λ – funkcja częściowa z X do V*
- *succ*(*λ*(*null*)) = ⊥

Akcje

- x = *y*
- *x != y*
- *x := y gdzie x != null*
- *x.next = y gdzie x != null*
- *x := y.next gdzie x, y != null*

Konfiguracje

- c = (*q, g*) gdzie $q \in Q$
- *Tranzycje na konfiguracjach*
- *t* = (*q, a, q1*)
- *c* = (*q, g*) *c1* = (*q1, g1*)

Tranzycje

- $x = y$, $\lambda(x) \mathrel{!}= \perp$, $\lambda(y) \mathrel{!}= \perp$, $\lambda(x) = \lambda(y)$, and $g = g1$.
- $x \mathrel{!}= y$, $\lambda(x) \mathrel{!}= \perp$, $\lambda(y) \mathrel{!}= \perp$, $\lambda(x) \mathrel{!}= \lambda(y)$, and $g = g1$.
- $x := y$, $\lambda(y) \mathrel{!}= \perp$, $succ = succ1$, and $\lambda 1 = \lambda\,[x \leftarrow \lambda(y)]$.
- $x := y.next$, $\lambda(y) \mathrel{!}= \perp$, $succ(\lambda(y)) \mathrel{!}= \perp$, $succ = succ1$, and $\lambda 1 = \lambda\,[x \leftarrow succ(\lambda(y))]$.
- $x.next := y$, $\lambda(x) \mathrel{!}= \perp$, $\lambda(y) \mathrel{!}= \perp$, $\lambda(x) \mathrel{!}= \lambda(null)$, $\lambda 1 = \lambda$, $succ1 = succ\,[\lambda(x) \leftarrow \lambda(y)]$.

Konfiguracje

- Definicja sumy, domknięcia

Porządek

- g1 <| g2 jeśli można uzyskać g1 z g2 poprzez:
  - usunięcie zmiennej
  - usunięcie wyizolowanego wierzchołka
  - usunięcie krawędzi
  - kontrakcja: usunięcie wierzchołka prostego
- g1 <= g2 jeśli można otrzymać g1 w skończonej liczbie ruchów
- c1 <= c2 – patrzymy na nierówność grafów;q1=q2

Porządek cd.

- Domknięcie górne
  - c↑={c1| c<=c1}
- Domknięcie i dolne
  - c↓= {c1| c1<=c}
- Abstrakcyjna tranzycja
  - *c*1→*tA c*2
  - Istnieje c3<=c1 takie, że *c*3→*t c*2

Własność bezpieczeństwa

Instance:

Sets C_{Init} and C_F of configurations.

Question: Is it the case $C_{Init} \xrightarrow{*} C_F\uparrow$?

Poprzednicy

- Definicja $c \rightsquigarrow c'$
- we write $c \overset{t}{\rightsquigarrow} c'$ to denote that $q = q_1$, $q' = q_2$ and $g \overset{a}{\rightsquigarrow} g'$
- Definicja Rank(C)c

Ważne wnioski

Lemma 1. *For configurations c_1, c_2 and c_3, if $c_1 \rightsquigarrow c_2$ and $c_3 \preceq c_1$ then there is a configuration c_4 such that $c_3 \rightsquigarrow c_4$ and $c_4 \preceq c_2$.*

Lemma 2. *Consider configurations c_1 and c_2. If $c_1 \rightsquigarrow c_2$ then $c_2 \longrightarrow c_1\uparrow$. If $c_1 \longrightarrow c_2\uparrow$ then $c_2 \rightsquigarrow c_1\downarrow$.*

- Dowody

Algorytm

Input: Two sets C_{Init} and C_F of configurations.

Output: $C_{Init} \xrightarrow{*}_A C_F\uparrow$?

ToExplore := C_F
Explored := $\emptyset$
while ToExplore $\neq \emptyset$ **do**
  remove some c from ToExplore
  if $\exists c' \in C_{Init}.\, c \preceq c'$ **then**
    return true
  else if $\exists c' \in$ Explored$.\, c' \preceq c$ **then**
    discard c
  else
    ToExplore := ToExplore $\bigcup \{c' \mid c \rightsquigarrow c'\}$
    Explored :=
    $\{c\} \bigcup \{c' \mid c' \in \text{Explored} \wedge (c \not\preceq c')\}$
  end if
end while
return false

Własność stopu

- WQO na słowach, wektorach, multizbiorach
- Wierzchołek prosty
- Kodowanie grafu: bloki, zbite grafy, zakodowanie

Własność stopu

- deg(g) – liczba niestrzeżonych wierzchołków, liści bez zmiennej
- Dla grafów zbitych o deg<k porządek na kodach to WQO
- Bo jest ograniczona liczba rodzajów bloków
- g[=g' => g<=g' (stosujemy kontrakcje)
- Przy cofaniu deg nie maleje – nie powstają niestrzeżone wierzchołki

Badania

Prog.	Prop.	Time	$\#C^{ons.}$	#Iter.	Prog.	Prop.	Time	$\#C^{ons.}$	#Iter.
Concat	Deref	0.4 s	7	3	Delete	Deref	0.4 s	8	4
Fumble	Deref	0.3 s	3	2	Reverse	Deref	0.3 s	2	1
Walk	Deref	0.4 s	9	3	Zip	Deref	1.9 s	206	12
Fumble	Garbage	0.7 s	38	14	Reverse	Garbage	0.8 s	55	24
Reverse	Well-form.	1.7 s	48	20					

The entry $\#C^{ons.}$ gives the total number of minimal configurations added to ToExplore in the analysis. The entry **#Iter.** is the number of iterations of the **while**-loop of the algorithm.

Podsumowanie

- Mamy metodę dla list
- W praktyce szybka
- Możliwe rozszerzenia:
  - Patrzeć na wartości
  - Rozważać więzy na długościach ścieżek
  - Integery jako sługości ścieżek

KONIEC