Redukcja programów współbieżnych do sekwencyjnych przy ograniczonej liczbie zmian kontekstu

Marcin Sulikowski

MIMUW

17 kwietnia 2010

1. Problem osiągalności w programach boolowskich

2. Zamiana programu współbieżnego na program sekwencyjny
   - Podejście gorliwe
   - Podejście leniwe

3. Przykłady

Programy boolowskie (*Boolean programs*)

- Imperatywne
- Skończone dziedziny zmiennych
- Funkcje rekurencyjne
- Niedeterminizm
- Bez współbieżności

Programy boolowskie (*Boolean programs*)

Składnia

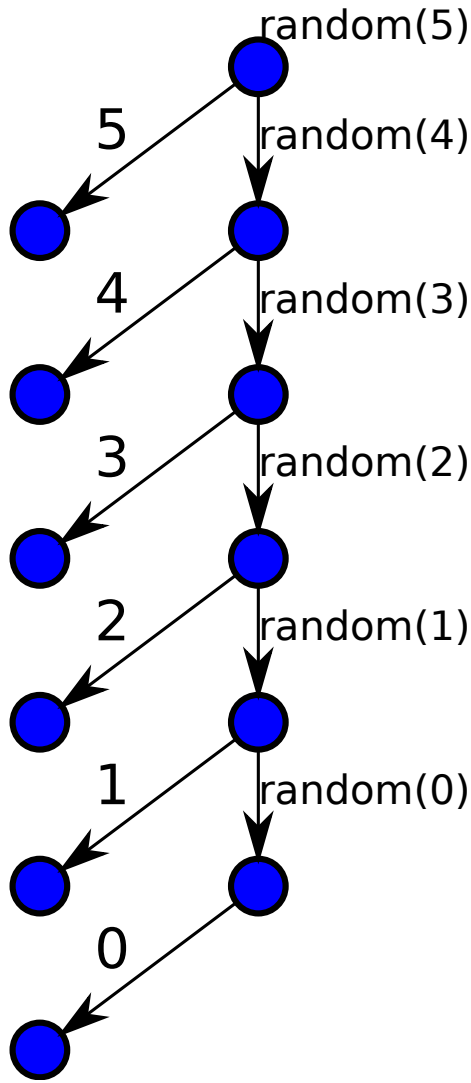
```
decl bool b;

int random(int limit)
begin
    decl int tmp;

    if (limit <= 0) then
      return 0;
    fi
    if (*) then
        return limit;
    else
        tmp := limit − 1;
        return random(tmp);
    fi
end
```

```
void main()
begin
    assume(b = F);
    if (random(100) = 0) then
        target: skip;
    fi
    while(T)
    do
        skip;
    od
    target2: skip;
end
```

Programy boolowskie (*Boolean programs*)

Niedeterminizm

```
decl bool b;

int random(int limit)
begin
    decl int tmp;

    if (limit <= 0) then
      return 0;
    fi
    if (*) then
        return limit;
    else
        tmp := limit − 1;
        return random(tmp);
    fi
end
```



Problem osiągalności

Sformułowanie dla programów boolowskich

Czy istnieje taki przebieg programu, że dana instrukcja programu jest wykonana?

Przykład

assert(ncrit <= 1); $\Longleftrightarrow$

if (!(ncrit <= 1)) **then**
    wybuch: **skip**;
fi

Problem osiągalności

Sformułowanie dla programów boolowskich

Czy istnieje taki przebieg programu, że dana instrukcja programu jest wykonana?

Przykład

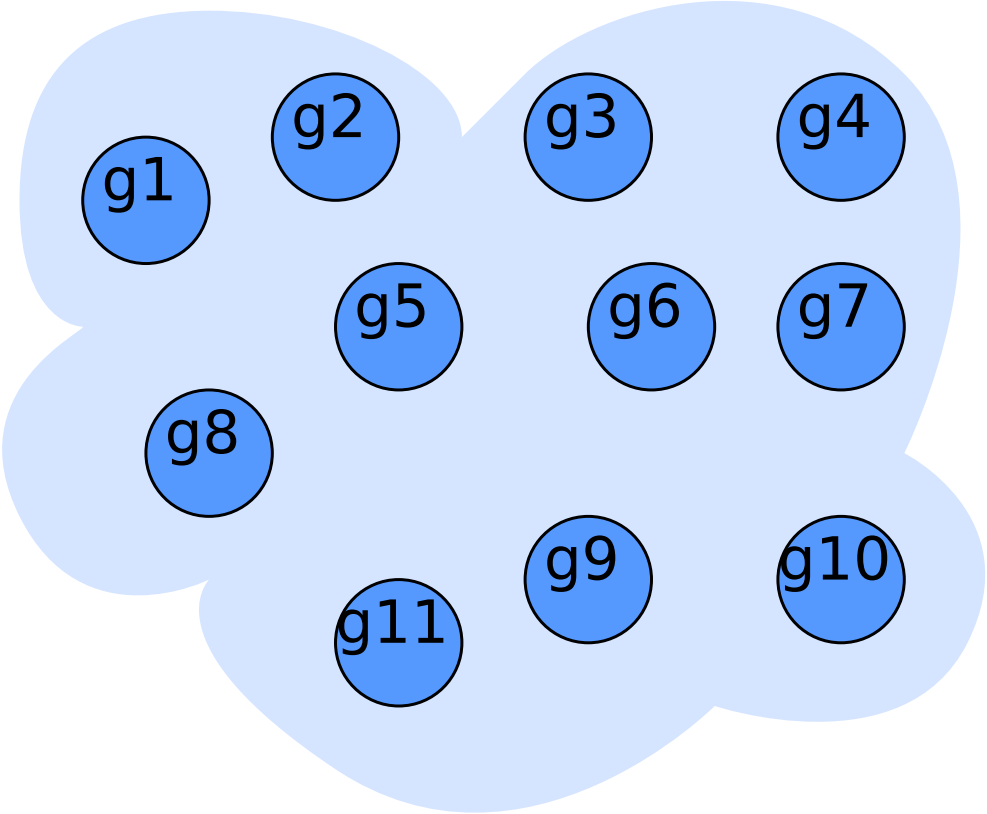
assert(ncrit <= 1); $\Longleftrightarrow$

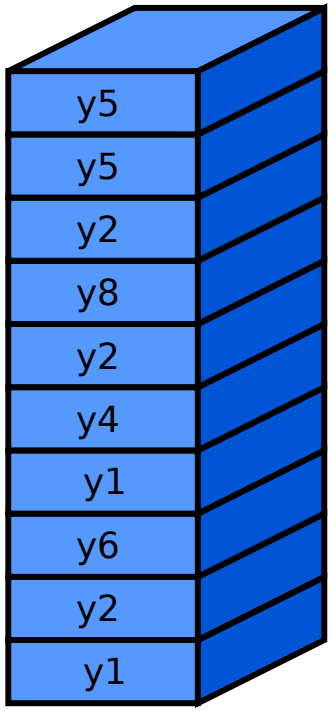
if (!(ncrit <= 1)) **then**
    wybuch: **skip**;
fi

Inny przykład?

procedura();
stop: **skip**;

Krok ku formalizacji

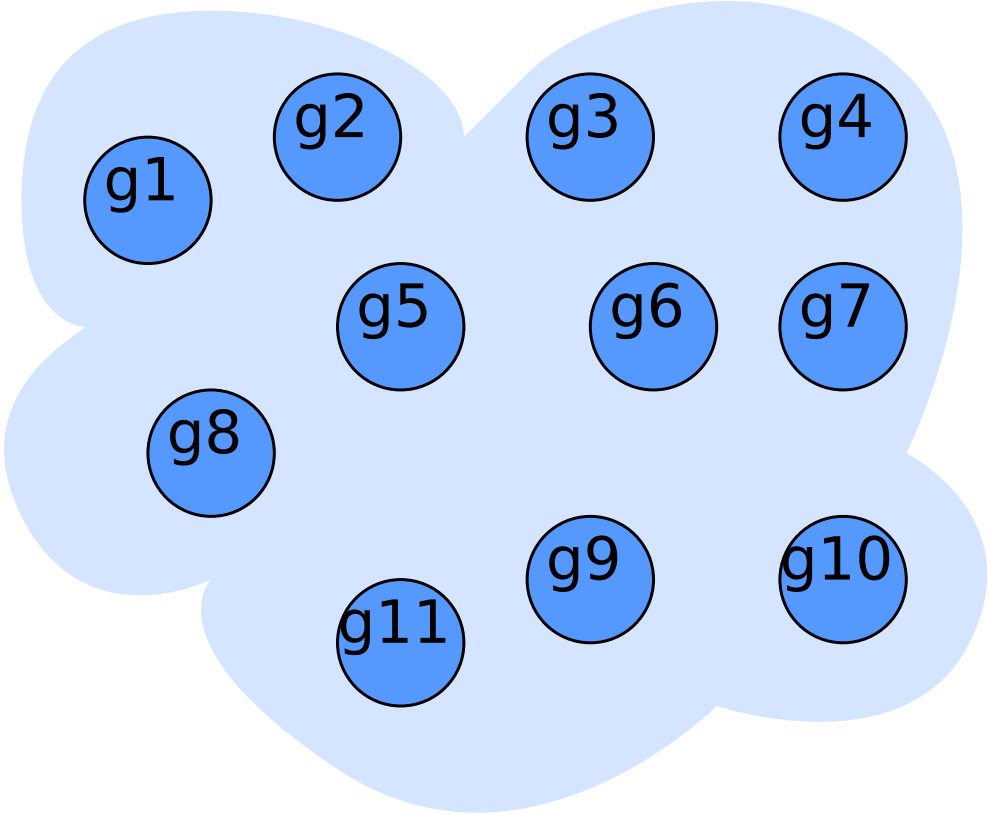


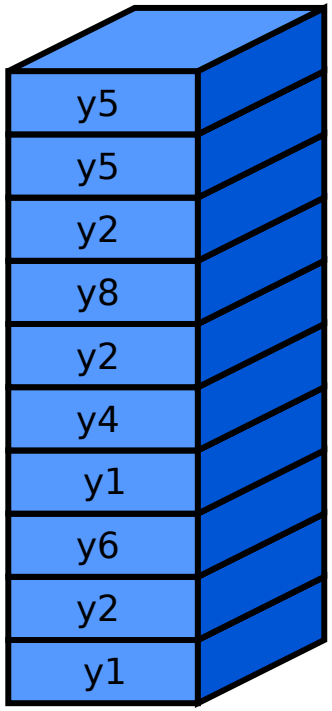


Krok ku formalizacji

Program boolowski odpowiada systemowi ze stosem.

- stany systemu $\sim$ zmienne globalne programu
- stos systemu $\sim$ stos wywołań programu i zmienne lokalne





System ze stosem

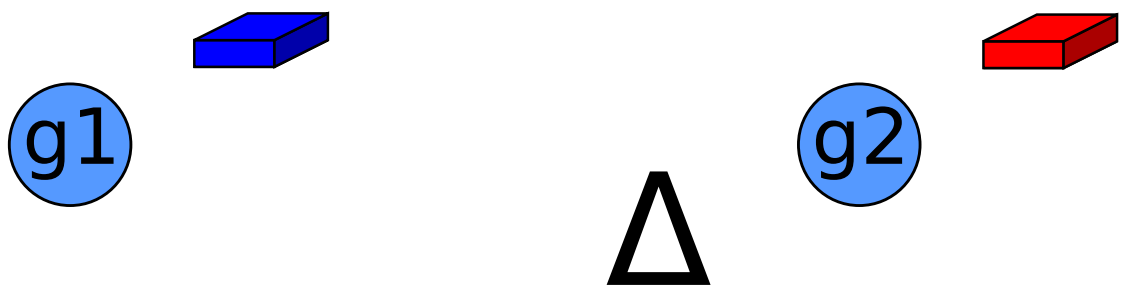
System ze stosem to krotka

$$(G, \Gamma, \Delta, g_{in}, w_{in})$$

G – zbiór stanów, Γ – alfabet stosowy, $g_{in} \in G$, $w_{in} \in \Gamma^*$,
$\Delta \subseteq G \times \Gamma \times G \times \Gamma^*$.
Konfiguracją nazwiemy parę $\langle g, w \rangle$, gdzie $g \in G$, $w \in \Gamma^*$.
Δ definiuje nam relację $\longrightarrow_\Delta$ na konfiguracjach:

$$\langle g_1, ww' \rangle \longrightarrow_\Delta \langle g_2, w''w' \rangle \Leftrightarrow (g_1, w, g_2, w'') \in \Delta$$

Jej domknięcie przechodnie oznaczymy $\longrightarrow_\Delta^*$.



System ze stosem

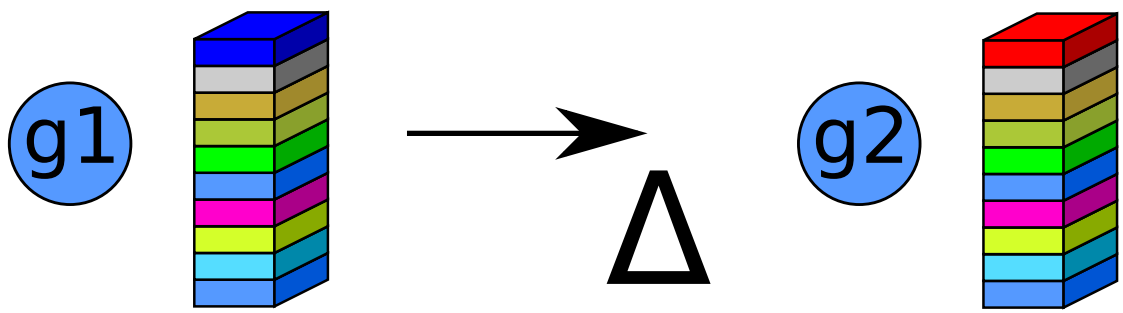
System ze stosem to krotka

$$(G, \Gamma, \Delta, g_{in}, w_{in})$$

G – zbiór stanów, Γ – alfabet stosowy, $g_{in} \in G$, $w_{in} \in \Gamma^*$,
$\Delta \subseteq G \times \Gamma \times G \times \Gamma^*$.
Konfiguracją nazwiemy parę $\langle g, w \rangle$, gdzie $g \in G$, $w \in \Gamma^*$.
Δ definiuje nam relację $\longrightarrow_\Delta$ na konfiguracjach:

$$\langle g_1, ww' \rangle \longrightarrow_\Delta \langle g_2, w''w' \rangle \Leftrightarrow (g_1, w, g_2, w'') \in \Delta$$

Jej domknięcie przechodnie oznaczymy $\longrightarrow_\Delta^*$.



System ze stosem

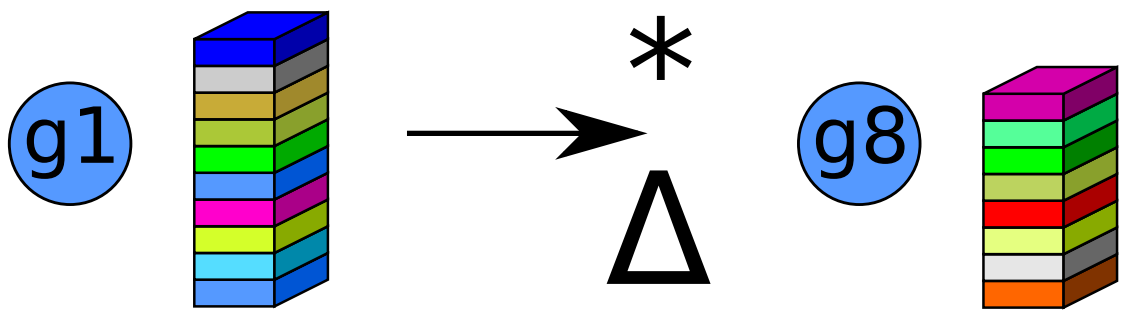
System ze stosem to krotka

$$(G, \Gamma, \Delta, g_{in}, w_{in})$$

G – zbiór stanów, Γ – alfabet stosowy, $g_{in} \in G$, $w_{in} \in \Gamma^*$,
$\Delta \subseteq G \times \Gamma \times G \times \Gamma^*$.
Konfiguracją nazwiemy parę $\langle g, w \rangle$, gdzie $g \in G$, $w \in \Gamma^*$.
Δ definiuje nam relację $\longrightarrow_\Delta$ na konfiguracjach:

$$\langle g_1, ww' \rangle \longrightarrow_\Delta \langle g_2, w''w' \rangle \Leftrightarrow (g_1, w, g_2, w'') \in \Delta$$

Jej domknięcie przechodnie oznaczymy $\longrightarrow_\Delta^*$.



Problem osiągalności

Sformułowanie dla systemów ze stosem

Czy dana konfiguracja $\langle g, w \rangle$ jest osiągalna? Formalnie – pytamy, czy dla systemu $(G, \Gamma, \Delta, g_{in}, w_{in})$ i danego $\langle g, w \rangle$ zachodzi:

$$\langle g_{in}, w_{in} \rangle \longrightarrow_{\Delta}^{*} \langle g, w \rangle$$

Problem osiągalności

Rozstrzygalność

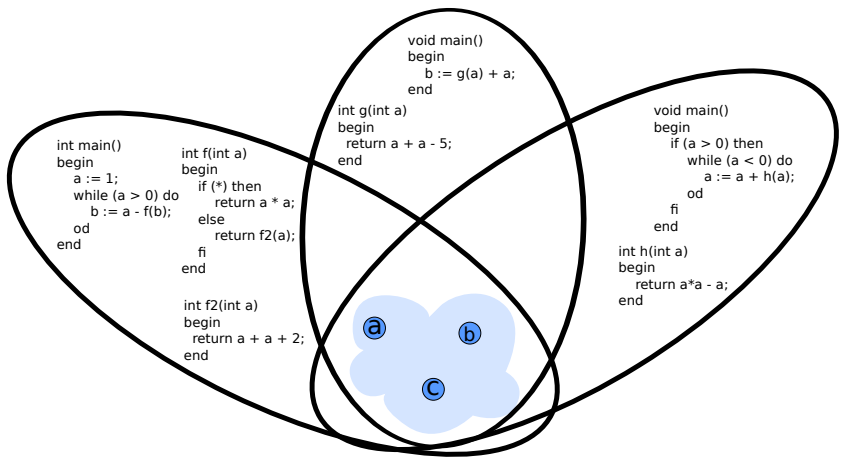
Problem osiągalności dla sekwencyjnych programów ze skończonymi dziedzinami zmiennych **jest rozstrzygalny**.

Dowód

Problem osiągalności dla systemów ze stosem jest rozstrzygalny, gdyż zbiór osiągalnych konfiguracji jest językiem regularnym.

Współbieżne programy boolowskie

- Zbiór zmiennych współdzielonych
- Kilka wykonujących się współbieżnie programów boolowskich



Współbieżne programy boolowskie

Składnia

```
decl bool b1, b2, b3;

void thread1()
begin
    while(T) do b1 := *; od;
end
void thread2()
begin
    while(T) do b2 := *; od;
end
void thread3()
begin
    while(T) do b3 := *; od;
end
```

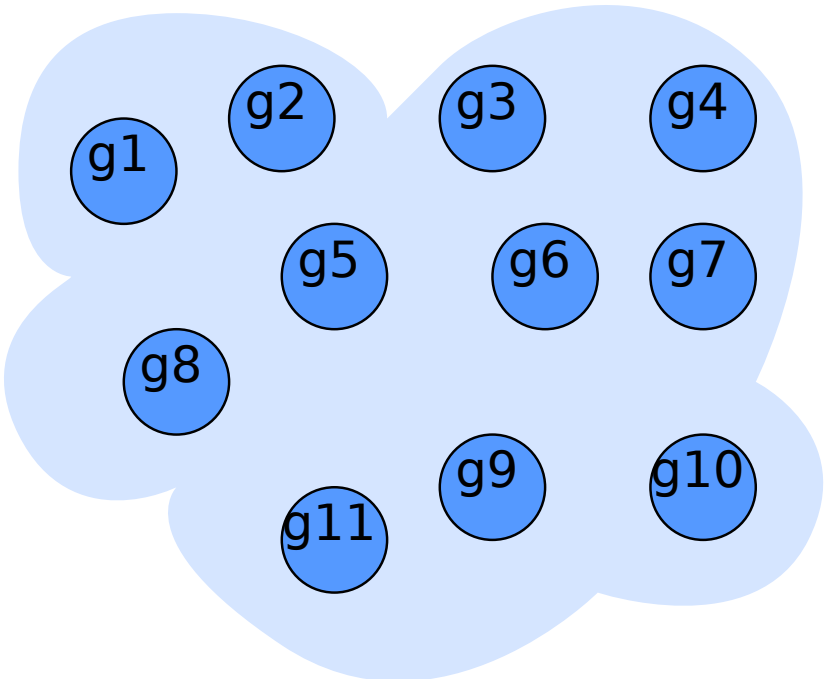
```
void thread4()
begin
    assume(b1 & b2 & b3);
    target: skip;
end

void init()
begin
    b1, b2, b3 := F, F, F.
end
```

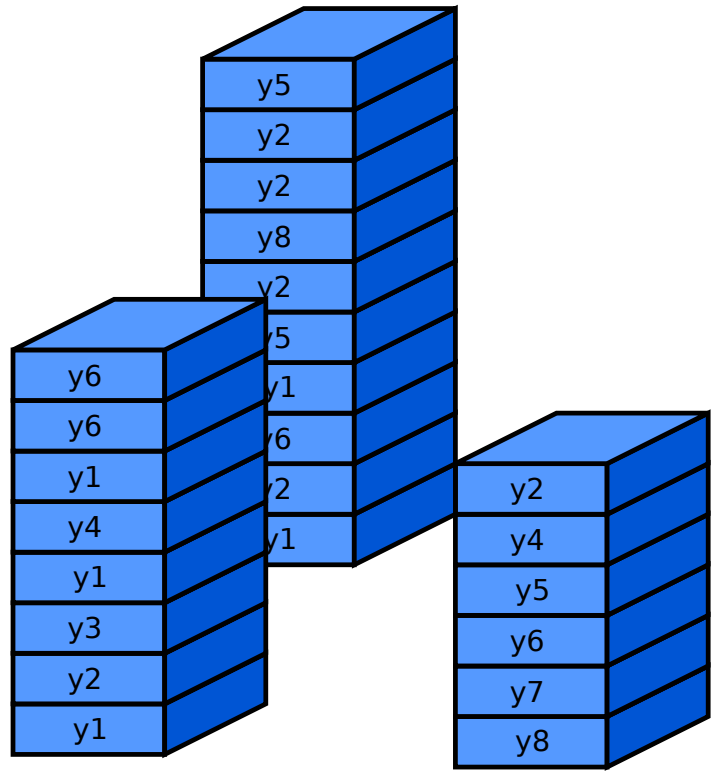
System z wieloma stosami

System z wieloma stosami to krotka:

$$(G, \Gamma, \Delta_0, \ldots, \Delta_N, g_{in}, w_{in})$$

Konfiguracja to krotka $\langle g, w_0, \ldots, w_N \rangle$, $g \in G$, $w_i \in \Gamma^*$.





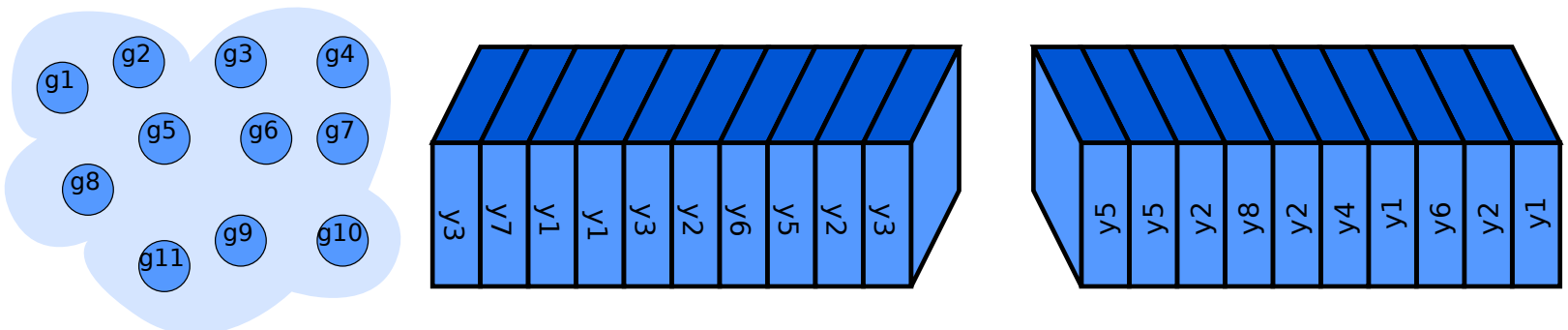
Problem osiągalności

Rozstrzygalność

Problem osiągalności dla **współbieżnych** programów ze skończonymi dziedzinami zmiennych **nie jest rozstrzygalny**.

Dowód

System z kilkoma stosami za bardzo przypomina maszynę Turinga.



Problem osiągalności

Rozstrzygalność

Problem osiągalności dla **współbieżnych** programów ze skończonymi dziedzinami zmiennych **jest rozstrzygalny**, jeśli **ograniczymy liczbę zmian kontekstu** przez stałą.

Zamiana programu współbieżnego na program sekwencyjny

Mamy:

Program współbieżny P_w z instrukcją I_w oraz stałą k

Zadanie:

Stworzyć program sekwencyjny P_s z instrukcją I_s taki, że:

Instrukcja I_s jest osiągalna w P_s

$$\Longleftrightarrow$$

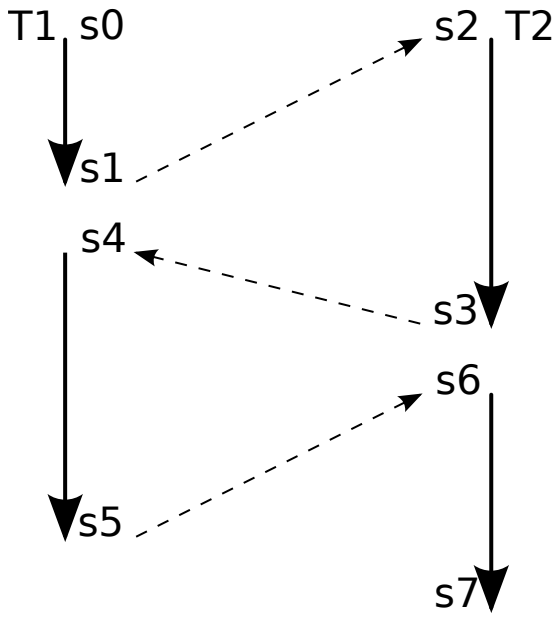
Instrukcja I_w jest osiągalna w P_w
po co najwyżej k zmianach kontekstu

Zamiana programu współbieżnego na program sekwencyjny

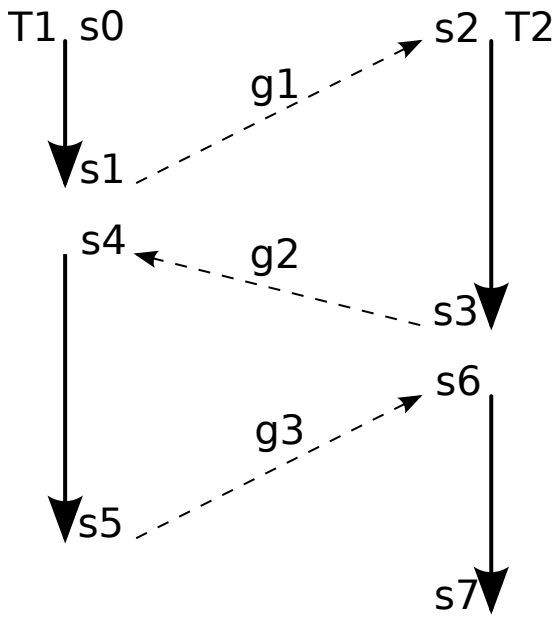
Dwa podejścia

- Podejście gorliwe
- Podejście leniwe

Podejście gorliwe

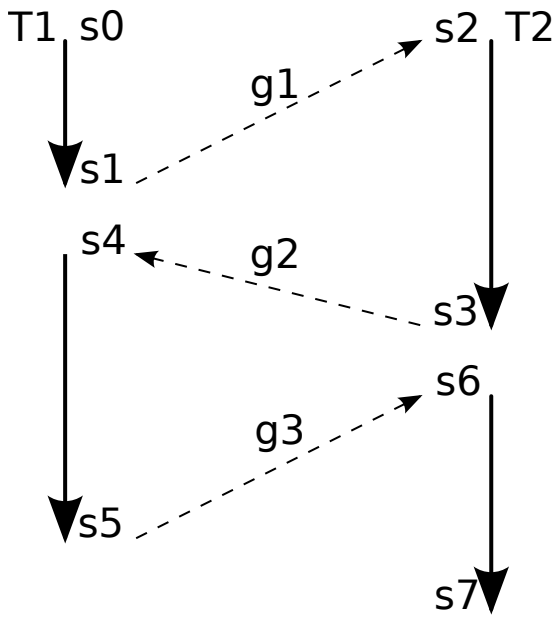


Podejście gorliwe



- Tworzymy zmienne $t_0, t_1, t_2, t_3 - t_i = j$ jeśli w i-tym kontekście wykonuje się wątek T_j
- Tworzymy 3 dodatkowe zestawy zmiennych współdzielonych: g_1, g_2, g_3.
- Tworzymy zmienną goal $=$ **F**. Ustawimy ją na **T**, gdy wykonamy instrukcję, której osiągalność sprawdzamy.

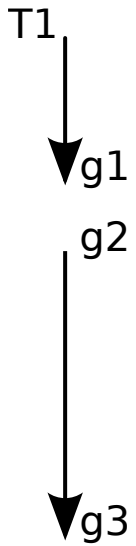
Podejście gorliwe



- Tworzymy zmienne $t_0, t_1, t_2, t_3 - t_i = j$ jeśli w i-tym kontekście wykonuje się wątek T_j
- Tworzymy 3 dodatkowe zestawy zmiennych współdzielonych: g_1, g_2, g_3.
- Tworzymy zmienną goal $=$ **F**. Ustawimy ją na **T**, gdy wykonamy instrukcję, której osiągalność sprawdzamy.

- Zgadujemy wartości t_i, np. $t_0 = 1, t_1 = 2, t_2 = 1, t_3 = 2$.
- Zgadujemy wartości g_i.

Podejście gorliwe



- Tworzymy zmienne $t_0, t_1, t_2, t_3 - t_i = j$ jeśli w i-tym kontekście wykonuje się wątek T_j
- Tworzymy 3 dodatkowe zestawy zmiennych współdzielonych: g_1, g_2, g_3.
- Tworzymy zmienną goal $=$ **F**. Ustawimy ją na **T**, gdy wykonamy instrukcję, której osiągalność sprawdzamy.

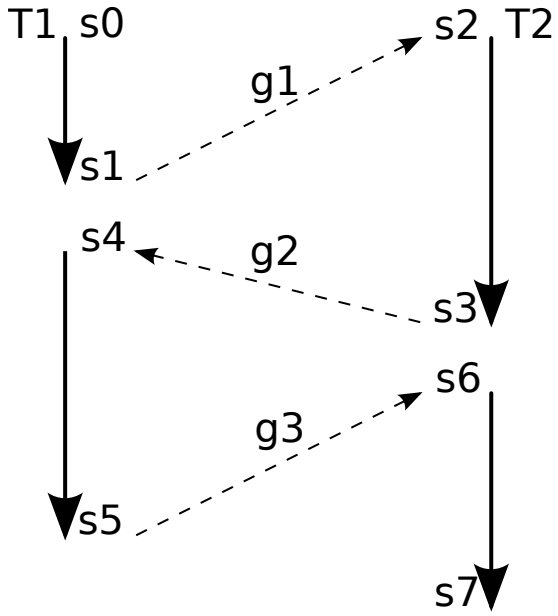
- Zgadujemy wartości t_i, np. $t_0 = 1, t_1 = 2, t_2 = 1, t_3 = 2$.
- Zgadujemy wartości g_i.
- Wykonujemy wątek T_1 i sprawdzamy, czy wartości zestawów g_1, g_2, g_3 są dobrze zgadnięte.

Podejście gorliwe

- Tworzymy zmienne $t_0, t_1, t_2, t_3 - t_i = j$ jeśli w i-tym kontekście wykonuje się wątek T_j
- Tworzymy 3 dodatkowe zestawy zmiennych współdzielonych: g_1, g_2, g_3.
- Tworzymy zmienną goal $=$ **F**. Ustawimy ją na **T**, gdy wykonamy instrukcję, której osiągalność sprawdzamy.

- Zgadujemy wartości t_i, np. $t_0 = 1, t_1 = 2, t_2 = 1, t_3 = 2$.
- Zgadujemy wartości g_i.
- Wykonujemy wątek T_1 i sprawdzamy, czy wartości zestawów g_1, g_2, g_3 są dobrze zgadnięte.
- Wykonujemy wątek T_2 i sprawdzamy, czy wartości zestawów g_1, g_2, g_3 są dobrze zgadnięte.

Podejście gorliwe



- Tworzymy zmienne t_0, t_1, t_2, t_3 – $t_i = j$ jeśli w i-tym kontekście wykonuje się wątek T_j
- Tworzymy 3 dodatkowe zestawy zmiennych współdzielonych: g_1, g_2, g_3.
- Tworzymy zmienną goal = **F**. Ustawimy ją na **T**, gdy wykonamy instrukcję, której osiągalność sprawdzamy.

- Zgadujemy wartości t_i, np. $t_0 = 1, t_1 = 2, t_2 = 1, t_3 = 2$.
- Zgadujemy wartości g_i.
- Wykonujemy wątek T_1 i sprawdzamy, czy wartości zestawów g_1, g_2, g_3 są dobrze zgadnięte.
- Wykonujemy wątek T_2 i sprawdzamy, czy wartości zestawów g_1, g_2, g_3 są dobrze zgadnięte.
- Sprawdzamy, czy goal = **T**.

Podejście gorliwe

Dodatkowe zmienne globalne

- $g_1, \ldots, g_k$ – kopie oryginalnych zmiennych globalnych
- $t_0, \ldots, t_k$ – t_i = numer wątku wykonującego się w kontekście i
- *goal* – czy wykonaliśmy sprawdzaną instrukcję
- t – numer aktualnie wykonywanego wątku
- cx – numer aktualnie wykonywanego kontekstu
- *terminate* – ustawiamy na **T**, gdy chcemy by wątek przestał się wykonywać

Podejście gorliwe

Funkcja main

```
void main()
begin
    goal := F;
    assume(0 < t_0 <= n) ... assume(0 < t_k <= n);
    for (t := 1; t <= n; t++) do
        cx := firstContext();
        if (cx <= k) then
            if (cx = 0) then init();
            else g := g_cx; fi
            terminate := F;
            thread_t();
        fi
    od
    assume(goal);
    target: skip;
end
```

Podejście gorliwe

Zmiany w oryginalnym kodzie

- **assume**(**F**); przed **return** w funkcjach thread_i
- goal := **T**; przed instrukcją, której osiągalność sprawdzamy
- Kod kontrolny w każdym potencjalnym miejscu zmiany kontekstu

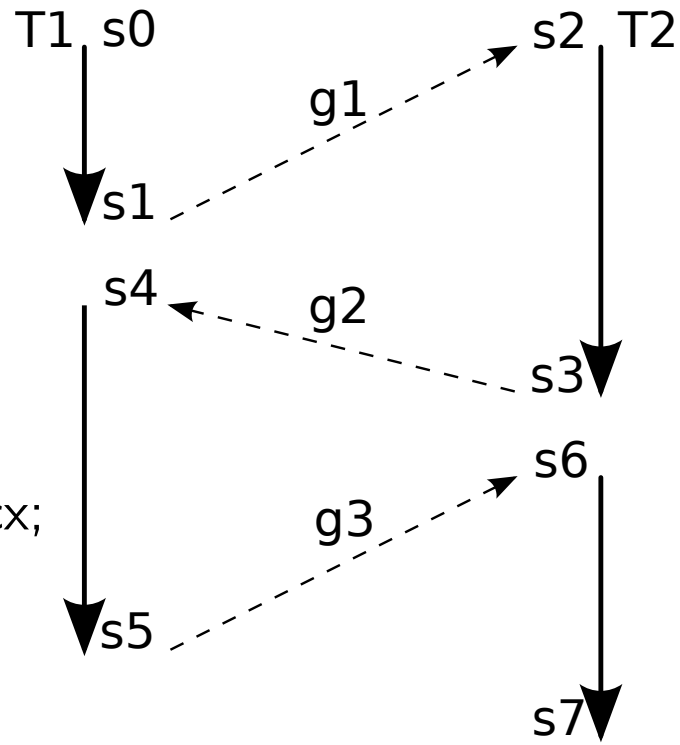
Kod kontrolny

if (terminate) **then return**; **fi**
if (∗) **then** contextSwitch(); **fi**
if (terminate) **then return**; **fi**

Podejście gorliwe

Funkcja contextSwitch()

```
void contextSwitch()
begin
    if (cx = k) then
        terminate := T;
    else
        assume(g = g_(cx+1));
        cx := nextContext();
        if (cx <= k) then g := g_cx;
        else terminate := T; fi
    fi
end
```



Podejście gorliwe

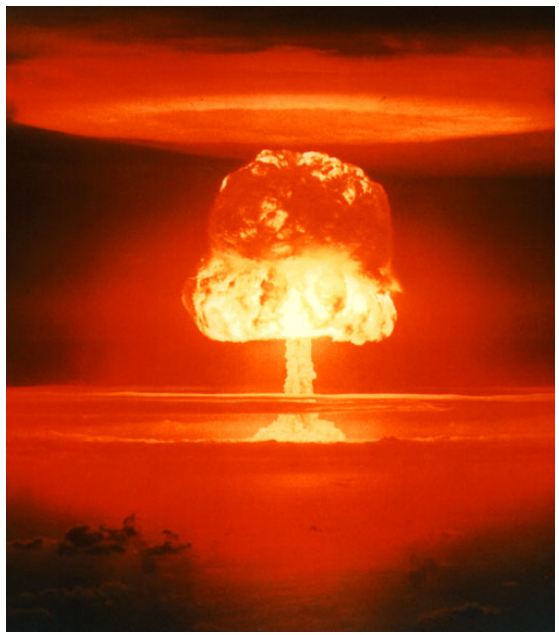
Co uzyskaliśmy?

Program sekwencyjny, w którym osiągalność jest równoważna ograniczonej osiągalności w wyjściowym programie współbieżnym

Podejście gorliwe

Wada

Przeszukujemy ogromną przestrzeń nieosiągalnych stanów!



Podejście leniwe

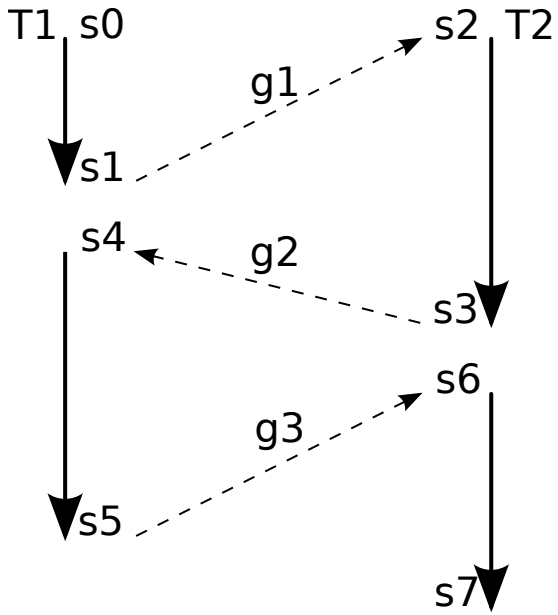
Cel

Przeszukujmy tylko osiągalne stany!

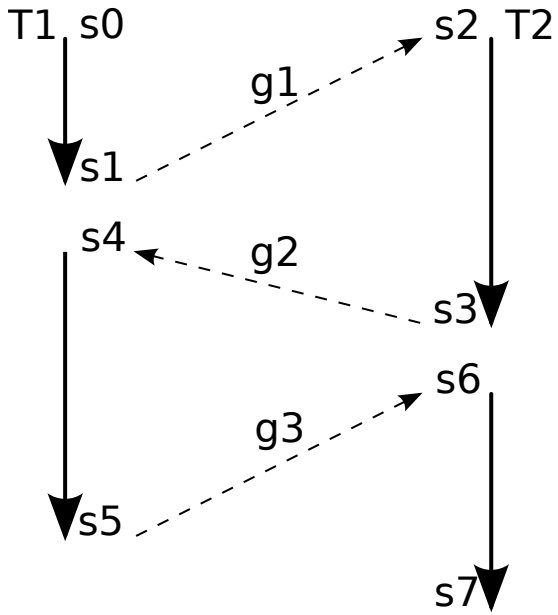
Pomysł

Zamiast zgadywać wartości zmiennych globalnych – obliczmy je

Podejście leniwe

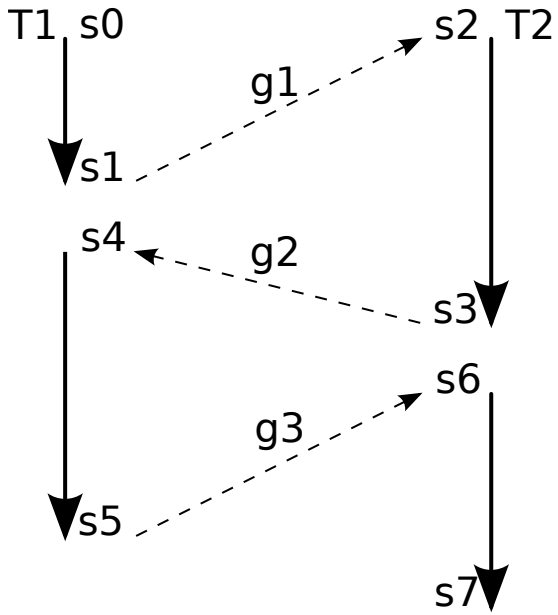


Podejście leniwe



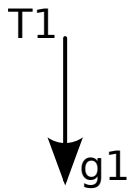
- Tworzymy zmienne $t_0, t_1, t_2, t_3 - t_i = j$ jeśli w i-tym kontekście wykonuje się wątek T_j
- Tworzymy 3 dodatkowe zestawy zmiennych współdzielonych: g_1, g_2, g_3.

Podejście leniwe



- Tworzymy zmienne $t_0, t_1, t_2, t_3 - t_i = j$ jeśli w i-tym kontekście wykonuje się wątek T_j
- Tworzymy 3 dodatkowe zestawy zmiennych współdzielonych: g_1, g_2, g_3.
- Zgadujemy wartości t_0, t_1, t_2, t_3

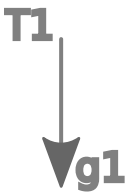
Podejście leniwe

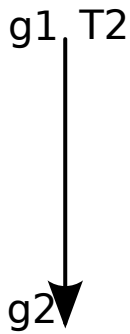


- Tworzymy zmienne $t_0, t_1, t_2, t_3 - t_i = j$ jeśli w i-tym kontekście wykonuje się wątek T_j
- Tworzymy 3 dodatkowe zestawy zmiennych współdzielonych: g_1, g_2, g_3.
- Zgadujemy wartości t_0, t_1, t_2, t_3

- Wątek o numerze t_0 oblicza g_1

Podejście leniwe

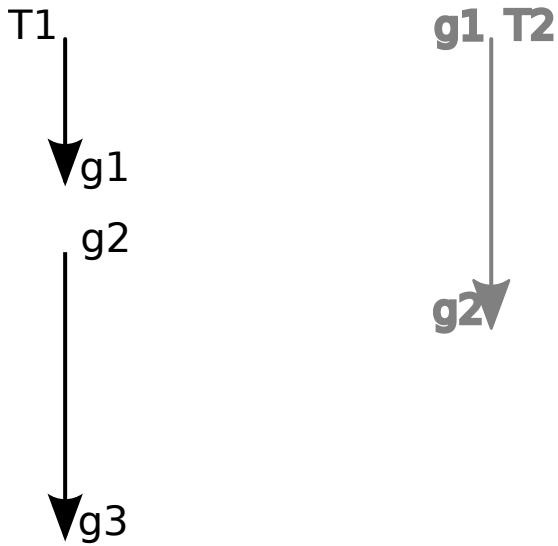




- Tworzymy zmienne $t_0, t_1, t_2, t_3 - t_i = j$ jeśli w i-tym kontekście wykonuje się wątek T_j
- Tworzymy 3 dodatkowe zestawy zmiennych współdzielonych: g_1, g_2, g_3.
- Zgadujemy wartości t_0, t_1, t_2, t_3

- Wątek o numerze t_0 oblicza g_1
- Wątek o numerze t_1 oblicza g_2

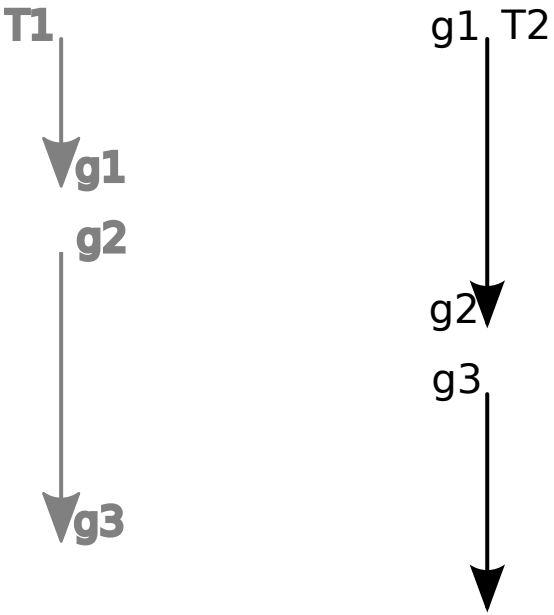
Podejście leniwe



- Tworzymy zmienne $t_0, t_1, t_2, t_3 - t_i = j$ jeśli w i-tym kontekście wykonuje się wątek T_j
- Tworzymy 3 dodatkowe zestawy zmiennych współdzielonych: g_1, g_2, g_3.
- Zgadujemy wartości t_0, t_1, t_2, t_3

- Wątek o numerze t_0 oblicza g_1
- Wątek o numerze t_1 oblicza g_2
- Wątek o numerze t_2 oblicza g_3

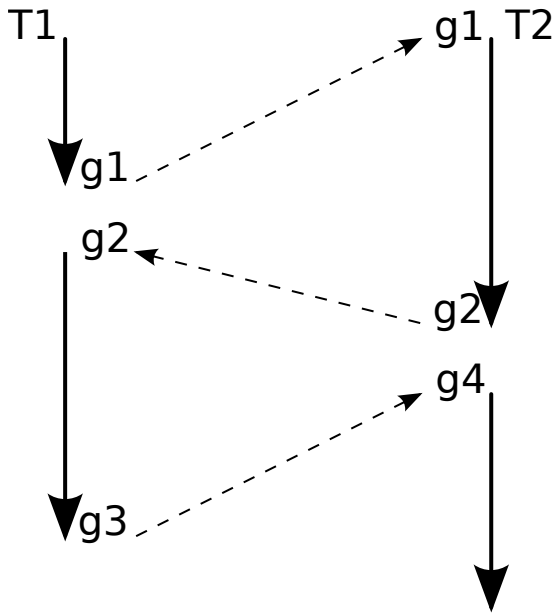
Podejście leniwe



- Tworzymy zmienne $t_0, t_1, t_2, t_3 - t_i = j$ jeśli w i-tym kontekście wykonuje się wątek T_j
- Tworzymy 3 dodatkowe zestawy zmiennych współdzielonych: g_1, g_2, g_3.
- Zgadujemy wartości t_0, t_1, t_2, t_3

- Wątek o numerze t_0 oblicza g_1
- Wątek o numerze t_1 oblicza g_2
- Wątek o numerze t_2 oblicza g_3
- Wątek o numerze t_3 wykonuje się

Podejście leniwe



- Tworzymy zmienne $t_0, t_1, t_2, t_3 - t_i = j$ jeśli w i-tym kontekście wykonuje się wątek T_j
- Tworzymy 3 dodatkowe zestawy zmiennych współdzielonych: g_1, g_2, g_3.
- Zgadujemy wartości t_0, t_1, t_2, t_3

- Wątek o numerze t_0 oblicza g_1
- Wątek o numerze t_1 oblicza g_2
- Wątek o numerze t_2 oblicza g_3
- Wątek o numerze t_3 wykonuje się
- Symulacja kończy się

Podejście leniwe

Co uzyskaliśmy?

Przeglądamy tylko osiągalne stany oryginalnego programu.

Przykłady

Bibliografia


S. La Torre, P. Madhusudan, G. Parlato
Reducing Context-bounded Concurrent Reachability to Sequential Reachability
CAV, tom 5643 LNCS, strony 477-492. Springer, 2009.


S. Qadeer, J. Rehof
Context-bounded model checking of concurrent software.
TACAS, tom 3440 LNCS, strony 93-107. Springer, 2005.


A. Lal, T. W. Reps
Reducing Concurrent Analysis Under a Context Bound to Sequential Analysis
CAV, tom 5123 LNCS, strony 37-51. Springer, 2008.