Weryfikacja wspomagana komputerowo

Wykład 4: Weryfikacja modelowa dla LTL

Algorytm

(i) $M \mapsto \mathcal{A}_M$

(ii) $\neg\phi \mapsto \mathcal{A}_{\neg\phi}$ (a nie $\phi \mapsto \mathcal{A}_\phi \mapsto \bar{\mathcal{A}}_\phi$)

(iii) $L_\omega(\mathcal{A}_M) \cap L_\omega(\mathcal{A}_{\neg\phi}) = \emptyset\,?$ (a nie $L_\omega(\mathcal{A}_M) \subseteq L_\omega(\mathcal{A}_\phi)$)

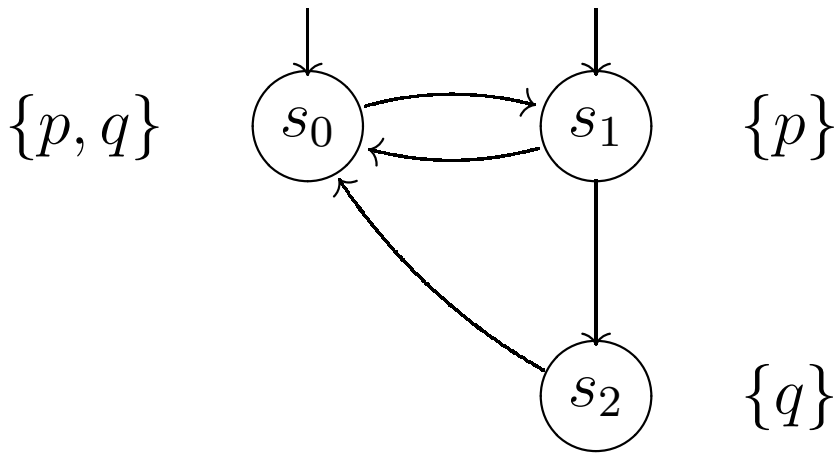
$L_\omega(\mathcal{A}_M \times \mathcal{A}_{\neg\phi}) = \emptyset\,?$

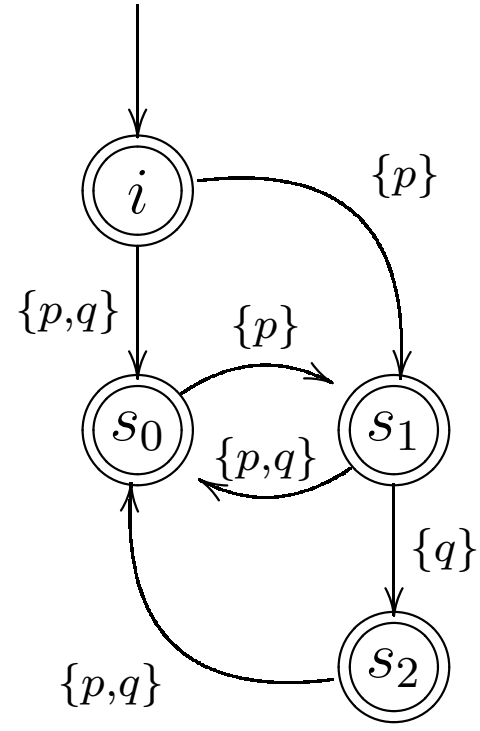
tak $\rightarrow$ $M \models \phi$

nie $\rightarrow$ $\neg(M \models \phi)$, kontrprzykład = ścieżka w M

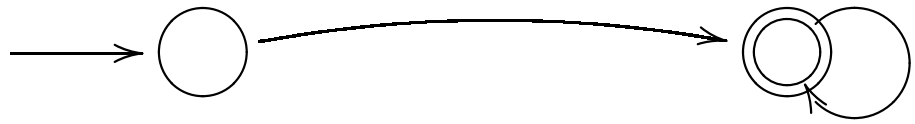
(i) $M \mapsto \mathcal{A}_M$

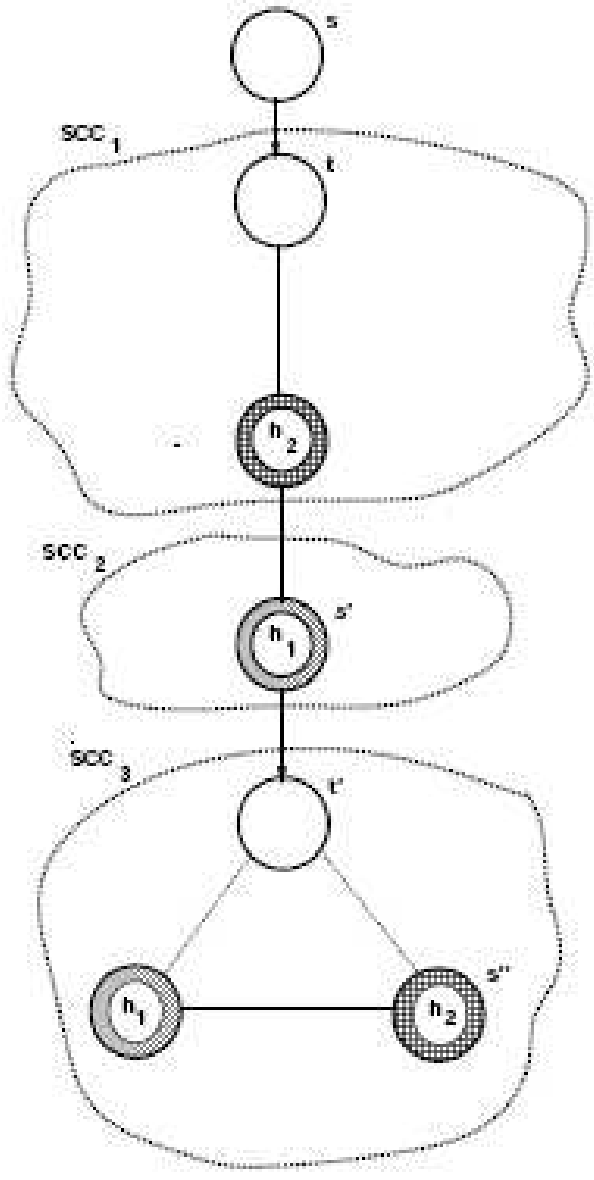
$M \mapsto \mathcal{A}_M$





(iii) $L_\omega(\mathcal{A}) \neq \emptyset$?



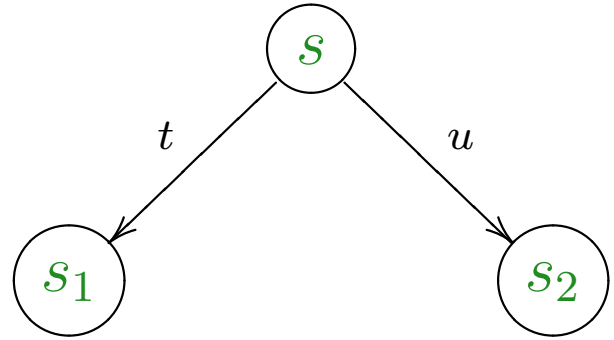


[Clarke, Grumberg, Peled 2000]

Ograniczenia

(1) Weryfikcja w locie

for each successsor s_i of s do . . .



. . .

Osiągalność: F źle

bezpieczeństwo: DFS lub BFS

```
proc dfs(s)
    if error(s) then report error fi
    add s to Statespace
    for each successor t of s do
      if t not in Statespace then dfs(t) fi
    od
end
```

[Holzmann,Peled,Yannakakis 1996]

Podwójny DFS

```
proc dfs(s)
    if error(s) then report error fi
    add {s,0} to Statespace
    for each successor t of s do
      if {t,0} not in Statespace then dfs(t) fi
    od
    if accepting(s) then seed:=s; ndfs(s) fi
end
proc ndfs(s) /* the nested search */
    add {s,1} to Statespace
    for each successor t of s do
      if {t,1} not in Statespace then ndfs(t) fi
      else if t==seed then report cycle fi
    od
end
```

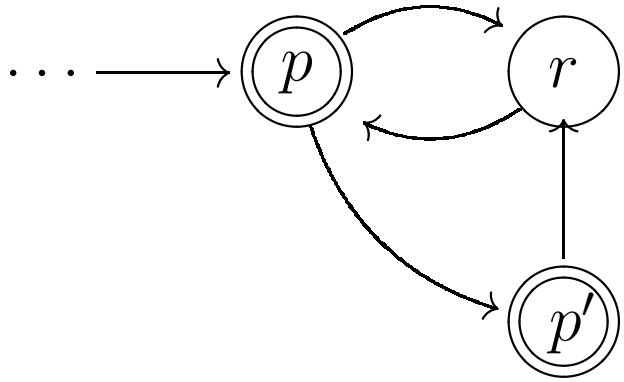
[Holzmann,Peled,Yannakakis 1996]

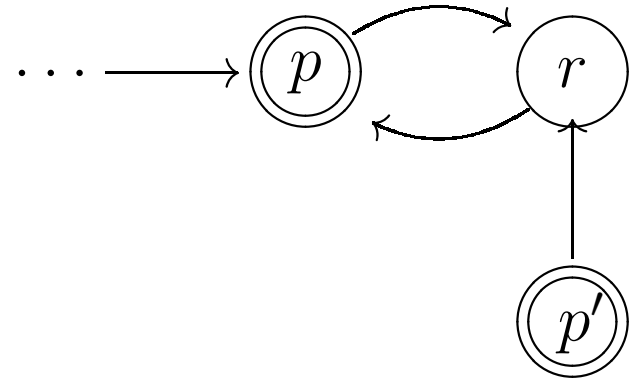
Dowód

Załóżmy, że jest stan akceptujący `p`, który ma cykl niewykryty w `ndfs(p)`. Niech `p` – pierwszy taki stan.

Niech `r` – pierwszy stan odwiedzony w `ndfs(p)` t.że `r` jest na cyklu zawierającym `p` i `{r,1} in Statespace`.

Niech `p'` – stan akceptujący t.że `r` osiągnięto w `ndfs(p')`.





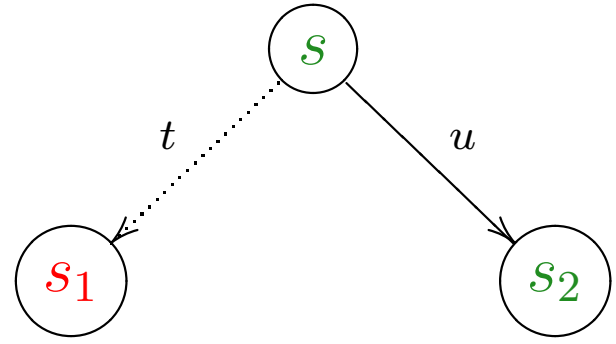
Ograniczenia (cd)

(1) Weryfikcja w locie

for each successsor s_i of s do . . .

(2) Redukcje częściowo-porządkowe

for each selected successsor s_i of s do . . .



selected – zależy od dotychczasowej ścieżki !

Rozwiązanie

```
proc dfs(s)
    if error(s) then report error fi
    add {s,0} to Statespace
    add s to Stack
    for each (selected) successor t of s do
      if {t,0} not in Statespace then dfs(t) fi
    od
    if accepting(s) then ndfs(s) fi
    delete s from Stack
end
proc ndfs(s) /* the nested search */
    add {s,1} to Statespace
    for each (selected) successor t of s do
      if {t,1} not in Statespace then ndfs(t) fi
      else if t in Stack then report cycle fi
    od
end
```

[Holzmann,Peled,Yannakakis 1996]

np-cycles: FG ¬ progress

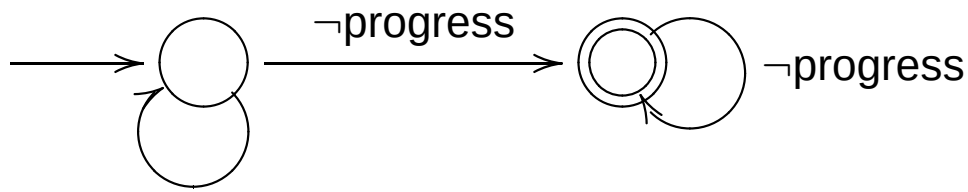
```
proc dfs(s)
    if error(s) then report error fi
    add {s,0} to Statespace
    for each successor t of s do
      if {t,0} not in Statespace then dfs(t) fi
    od
    ndfs(s) /* different */
end
proc ndfs(s) /* the nested search */
    if s is Progress State then return fi /* new */
    add {s,1} to Statespace
    add s to Stack /* new */
    for each successor t of s do
      if {t,1} not in Statespace then ndfs(t) fi
      else if t is in Stack then report cycle fi /* different */
    od
    delete s from Stack /* new */
end
```

[Holzmann,Peled,Yannakakis 1996]

np-cycles: automat

```
never { /* non-progress: ◇□¬progress */
    do
    :: skip
    :: !progress -> break
    od;
accept: do
    :: !progress
    od
}
```

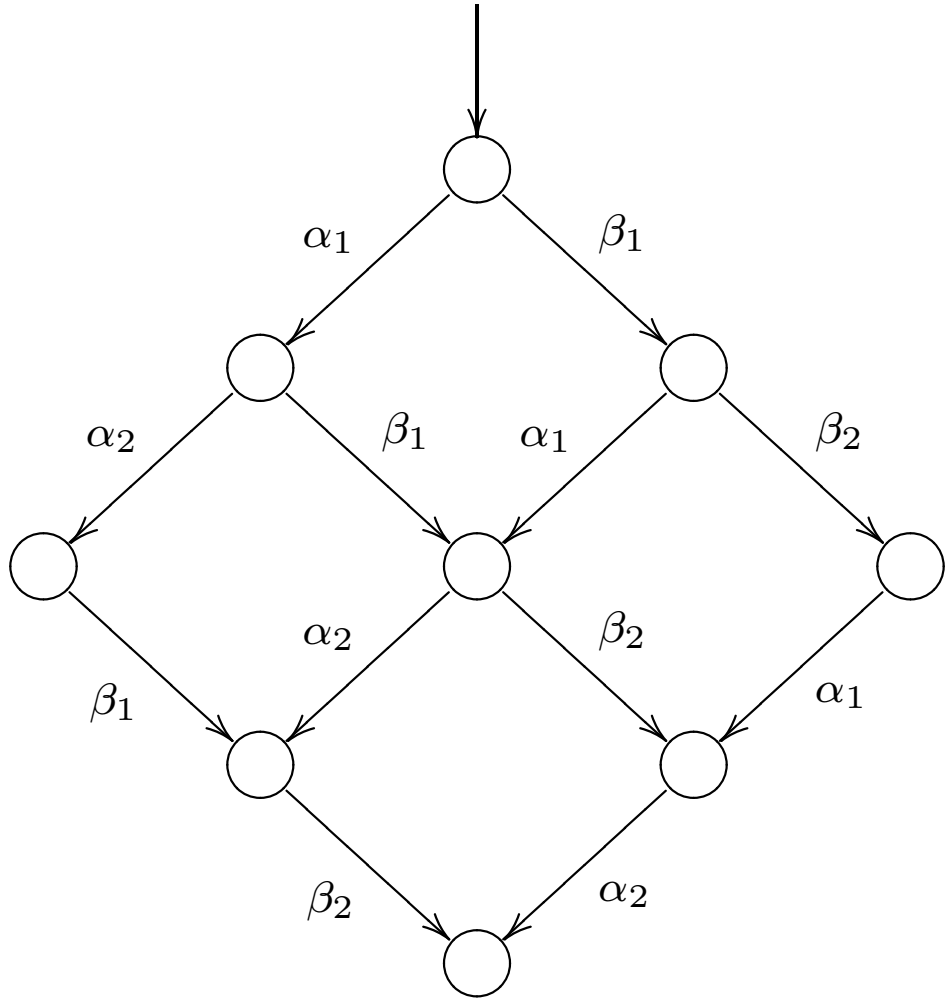
[Holzmann,Peled,Yannakakis 1996]



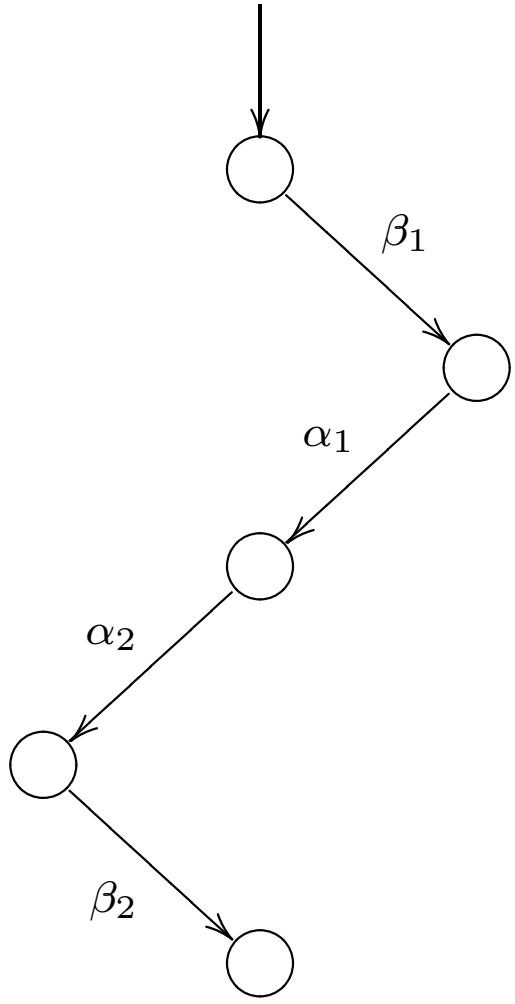
(co-Büchi $\subseteq$ Büchi)

Redukcja częściowo-porządkowa

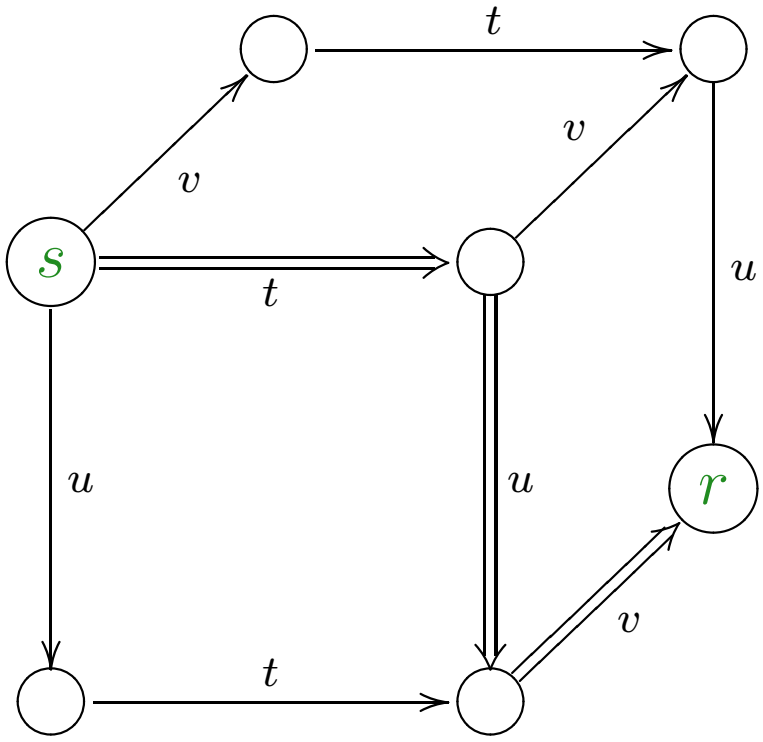
Motywacja


α_1
β_1
α_2
β_1
α_1
β_2
α_2
β_2
β_1
α_1
β_2
α_2

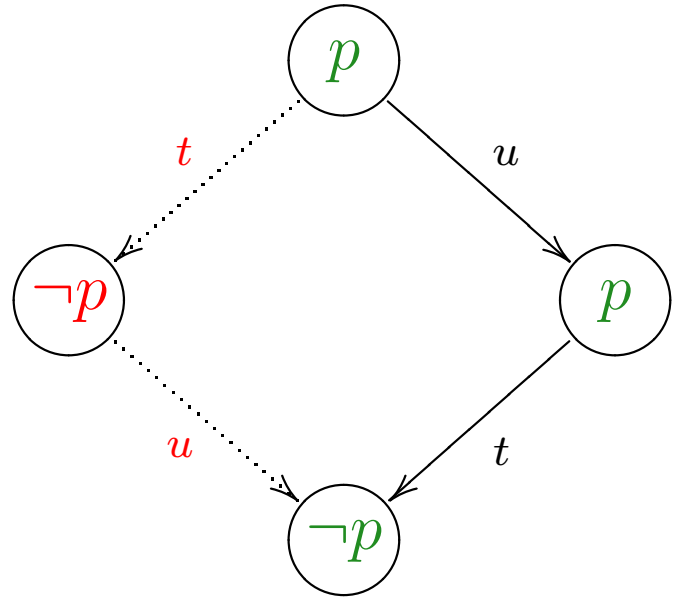
Motywacja



Motywacja



F $\neg p$



t, u niezależne

Model

Def.: $M = \langle S, S_{\text{pocz}}, T, L \rangle$ T – operacje (tranzycje)

dla $\alpha \in T$: $\text{en}_\alpha \subseteq S$, $\alpha : \text{en}_\alpha \to S$ (determinizm)

ścieżka: $\Pi = s_0 \xrightarrow{\alpha_0} s_1 \xrightarrow{\alpha_1} s_2 \xrightarrow{\alpha_2} \ldots$ $s_0 = s_{\text{pocz}}$

$\alpha_i(s_i) = s_{i+1}$

$\text{en}_s := \{\alpha \mid s \in \text{en}_\alpha\}$ $(\alpha \in \text{en}_s \iff s \in \text{en}_\alpha)$

Pomysł: $\text{ample}_s \subseteq \text{en}_s$ zamiast en_s w podwójnym DFS'ie ?

Opłacalność

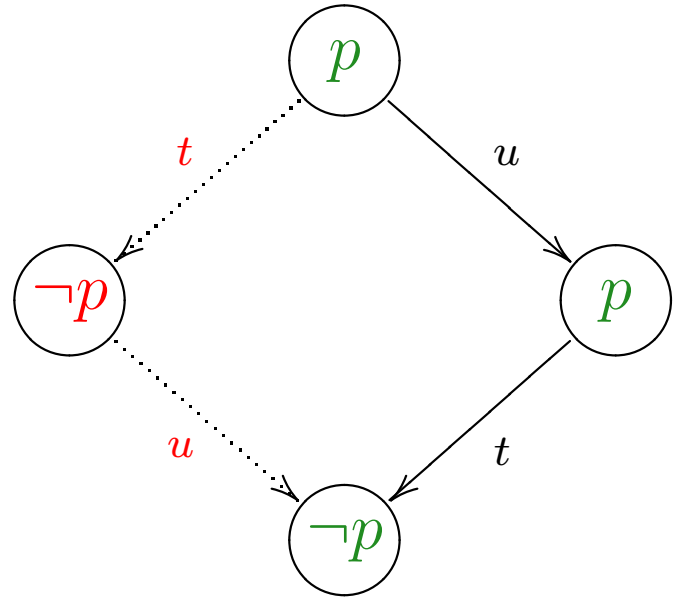
Pomysł: $\mathrm{ample}_s \subseteq \mathrm{en}_s$ zamiast en_s w podwójnym DFS'ie ?

To ma sens, gdy:

- weryfikacja daje ten sam wynik (poprawność)
- znacząco maleje liczba stanów
- narzut czasowy jest rozsądny (opłacalność)

Problemy?

Kiedy można ignorować t ?



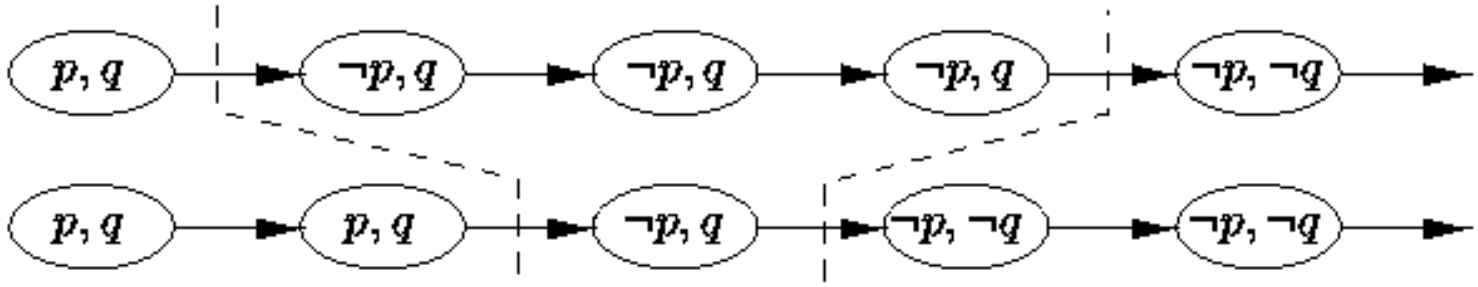
Problem 1: Własność może zależeć od stanu $\neg p$.

Problem 2: Stan $\neg p$ może mieć następniki nieosiągalne inaczej.

Def.: $\Pi = s_0 \to s_1 \to s_2 \to \ldots$ i $\Pi' = s'_0 \to s'_1 \to s'_2 \to \ldots$ są równoważne (ang. stuttering equivalence), $\Pi \equiv \Pi'$, jeśli ciągi

$$L(s_0), L(s_1), L(s_2), \ldots \qquad L(s'_0), L(s'_1), L(s'_2), \ldots$$

stają się identyczne po pogrupowaniu:



Def.: $M \equiv M'$ wtw. gdy – $\forall \Pi\ w\ M \quad \exists \Pi'\ w\ M' \quad \Pi \equiv \Pi'$

– $\forall \Pi'\ w\ M' \quad \exists \Pi\ w\ M \quad \Pi \equiv \Pi'$

LTL$_{-X}$ = LTL bez X

Tw.: Gdy $\phi \in$ LTL$_{-X}$ i $\Pi \equiv \Pi'$, to $\Pi \models \phi \iff \Pi' \models \phi$

Tw.: Gdy $\phi \in$ LTL$_{-X}$ i $M \equiv M'$, to $M \models \phi \iff M' \models \phi$

Tw.: LTL$_{-X}$ $=$ FO$_{\equiv}$

Poprawność

$$M \xrightarrow{\text{redukcja}} M'$$

$$M \equiv M'$$

Warunki wystarczające dla poprawności

(C0) $\mathrm{ample}_s = \emptyset \iff \mathrm{en}_s = \emptyset$

(C1) . . .

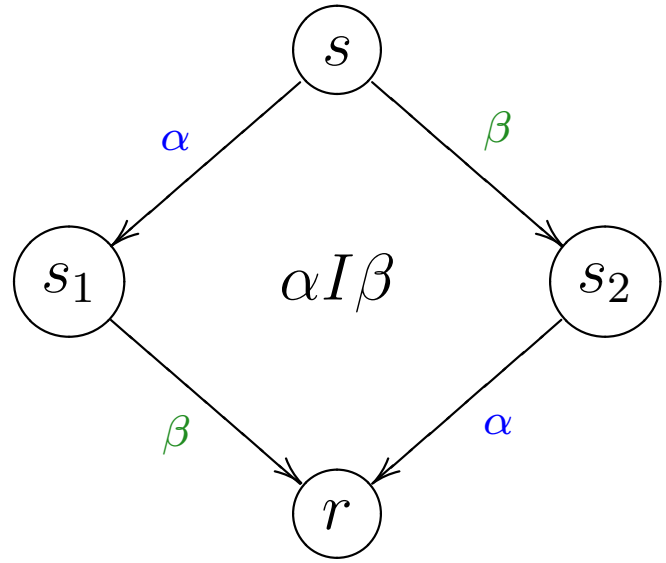
(C2) . . .

(C3) . . .

Def.: α jest niewidoczna gdy $L(s) = L(\alpha(s)), \forall\ s \in \mathrm{en}_\alpha$.

Przykład: Jeśli α niewidoczna, to

$$ss_1r \equiv ss_2r$$



Warunki wystarczające dla poprawności

(C0) $\mathrm{ample}_s = \emptyset \iff \mathrm{en}_s = \emptyset$

(C1) jeśli $\mathrm{ample}_s \neq \mathrm{en}_s$, to każda $\alpha \in \mathrm{ample}_s$ jest

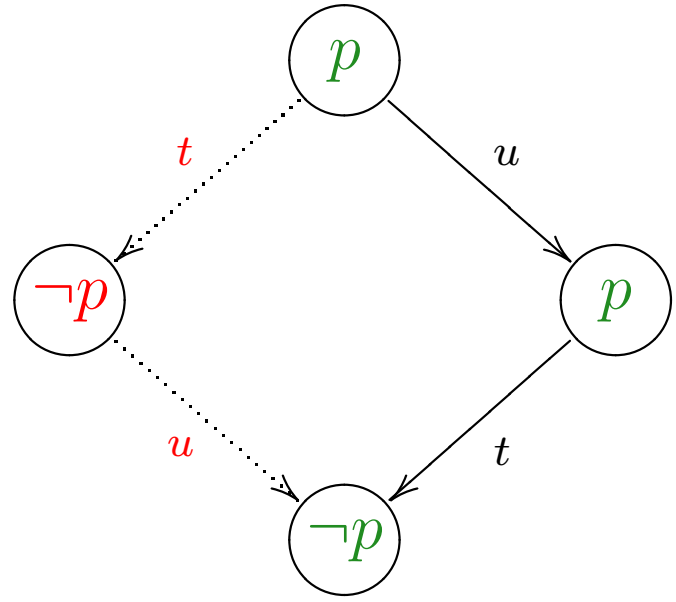
niewidoczna

(C2) . . .

(C3) . . .

Intuicja: Zamiast zrobić dziś, zrób w przyszłości!

Problem 1: Własność może zależeć od stanu $\neg p$.



Rozwiązany dzięki **(C1)** !

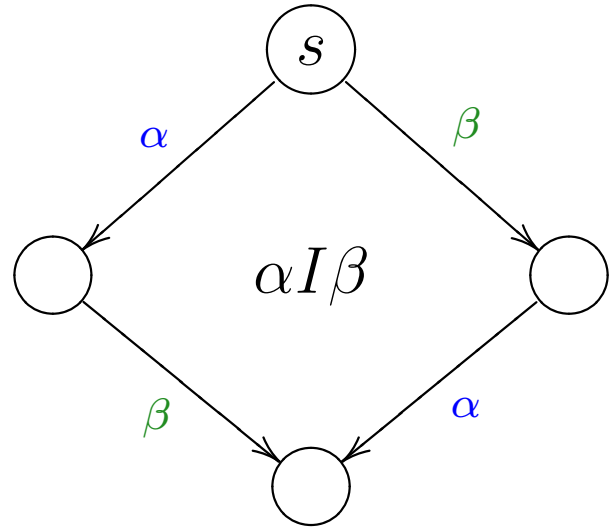
(C1) jeśli $\mathrm{ample}_s \neq \mathrm{en}_s$, to każda $\alpha \in \mathrm{ample}_s$ jest

niewidoczna

Def.: Relacja niezależności $I \subseteq T \times T$:

- relacja przeciwzwrotna i symetryczna
- jeśli $\alpha I \beta$, $\alpha \in \mathrm{en}_s$, $\beta \in \mathrm{en}_s$, to $(s \in \mathrm{en}_\alpha \cap \mathrm{en}_\beta)$
  - $\beta(s) \in \mathrm{en}_\alpha$, $\alpha(s) \in \mathrm{en}_b$
  - $\beta(\alpha(s)) = \alpha(\beta(s))$

$D = T \times T \setminus I$ (relacja zależności)



Przykład: Niezależne mogą być:

- 2 instrukcje różnych procesów operujące na zmiennych lokalnych
- 2 instrukcje różnych procesów inkrementujące tę samą zmienną globalną
- 2 instrukcje różnych procesów piszące lub czytające z różnych buforów

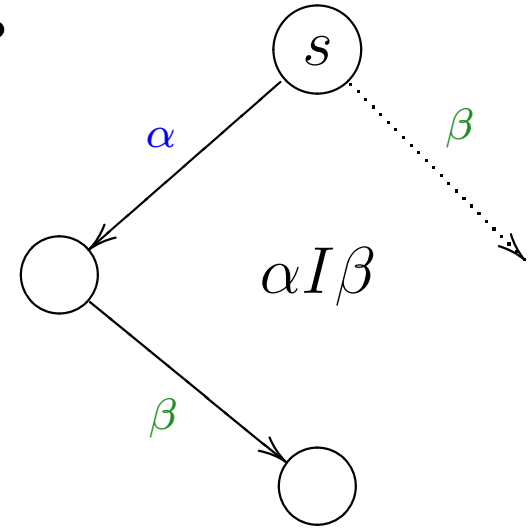
Przykład: Niezależne mogą być:

- 2 instrukcje różnych procesów operujące na zmiennych lokalnych
- 2 instrukcje różnych procesów inkrementujące tę samą zmienną globalną
- 2 instrukcje różnych procesów piszące lub czytające z różnych buforów

- 2 instrukcje tego samego procesu ?

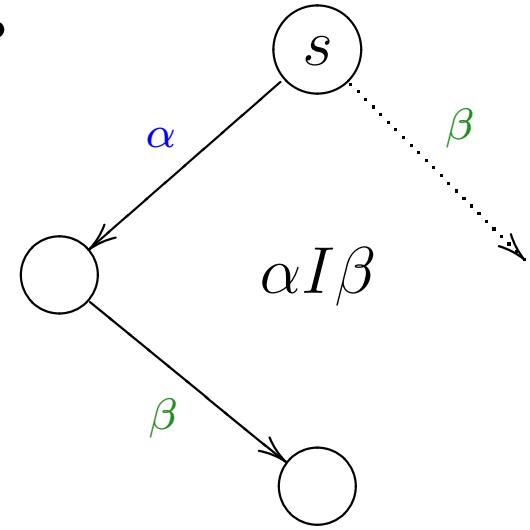
Pytanie: Niech $\alpha I \beta$. Czy możliwe jest

$$s \in \mathrm{en}_\alpha \setminus \mathrm{en}_\beta \qquad \alpha(s) \in \mathrm{en}_\beta \; ?$$



Pytanie: Niech $\alpha I \beta$. Czy możliwe jest

$$s \in \mathrm{en}_\alpha \setminus \mathrm{en}_\beta \qquad \alpha(s) \in \mathrm{en}_\beta \; ?$$



Tak! Np. asynchroniczne czytanie i pisanie do tego samego bufora.

Warunki wystarczające dla poprawności

(C0) $\mathrm{ample}_s = \emptyset \iff \mathrm{en}_s = \emptyset$

(C1) jeśli $\mathrm{ample}_s \neq \mathrm{en}_s$, to każda $\alpha \in \mathrm{ample}_s$ jest

niewidoczna

(C2) ? $\left(\mathrm{en}_s \setminus \mathrm{ample}_s\right) \; I \; \mathrm{ample}_s$

(C3) ...

Intuicja: Zamiast zrobić dziś, zrób w przyszłości!

(C2) tranzycja zależna od pewnej tranzycji z ample_s

nie może być wykonana zanim nie wykona się

tranzycja z ample_s

(C2) tranzycja zależna od pewnej tranzycji z ample_s

nie może być wykonana zanim nie wykona się

tranzycja z ample_s

(C2) dla każdej ścieżki Π rozpoczynającej się z s:

jeśli $\alpha \in \mathrm{ample}_s,\ \beta \notin \mathrm{ample}_s,\ \alpha D \beta$

to β nie może być wykonane w Π

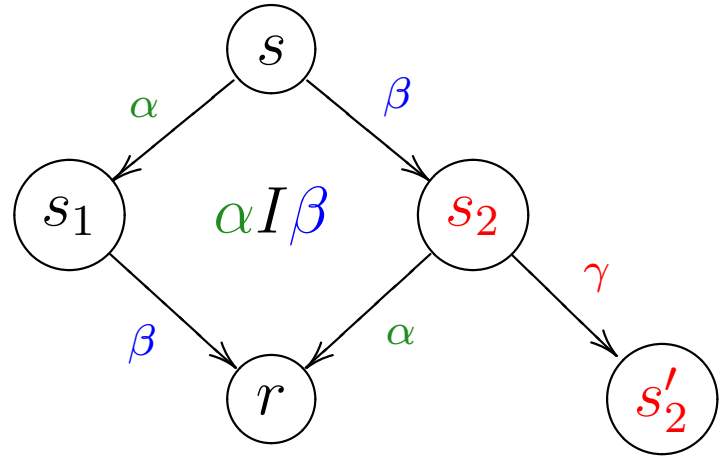
zanim nie wykona się pewna tranzycja z ample_s

Lemat: Jeśli (C2) , to $\left(\mathrm{en}_s \setminus \mathrm{ample}_s\right)\ I\ \mathrm{ample}_s$.

Dowód: Niech $\beta \in \mathrm{en}_s \setminus \mathrm{ample}_s$, $\alpha \in \mathrm{ample}_s$, $\alpha D \beta$.

$s \xrightarrow{\beta} \beta(s) \rightarrow \ldots$ sprzeczność z (C2) .

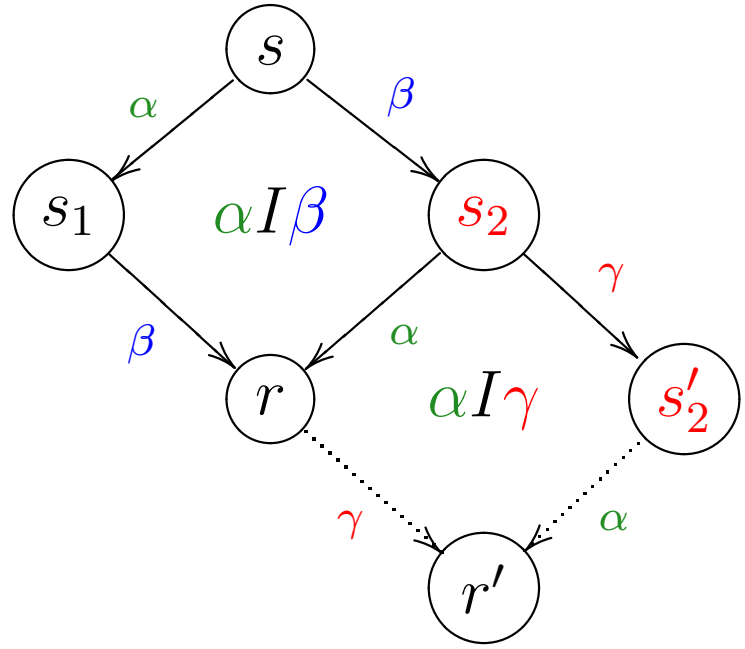
Problem 2: stan s_2 może mieć następniki nieosiągalne inaczej.



np. niech $\alpha \in \text{ample}_s, \beta \notin \text{ample}_s$

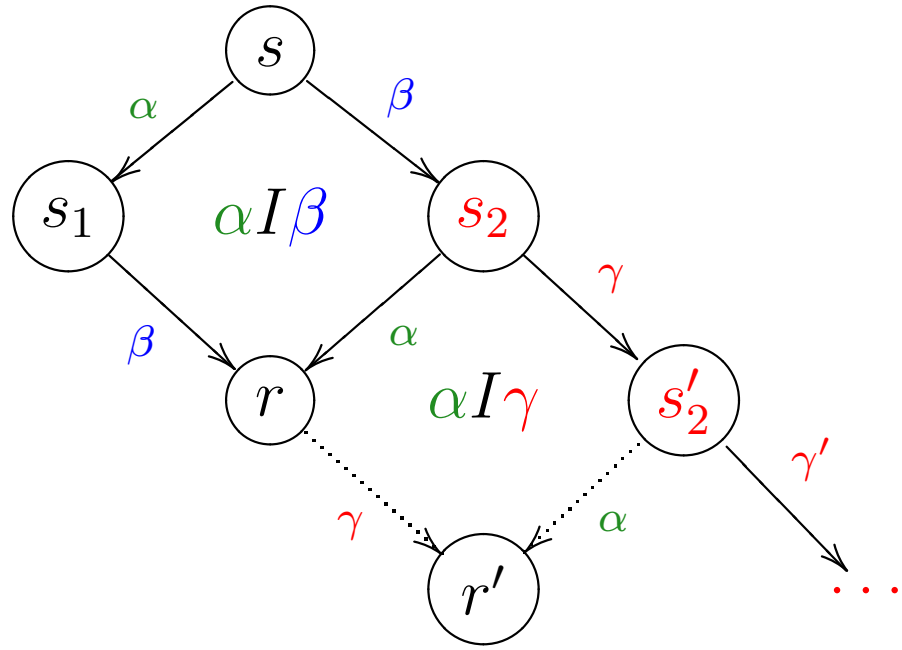
dzięki **(C2)** dla $\beta\gamma \ldots$, wnioskujemy $\gamma I \alpha$

Problem 2: stan s_2 może mieć następniki nieosiągalne inaczej.



α niewidoczne, więc $ss_1rr' \equiv ss_2s_2'$

Problem 2': stan s_2 może mieć nieskończoną ścieżkę nieosiągalną inaczej.

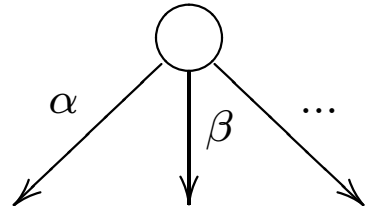


dzięki **(C2)** wnioskujemy $\gamma I \alpha$, $\gamma' I \alpha$, ...

α niewidoczne, więc $s s_1 r r' \ldots \equiv s s_2 s_2' \ldots$

Def. (sprawiedliwość): jeśli $\alpha \in \mathrm{en}_s$ prawie zawsze, to α w końcu się wykona.

Wniosek: dla każdego osiagalnego stanu s, jeśli $\alpha \in \mathrm{en}_s$ to kiedyś musi zostać wykonana β t. że $\alpha D \beta$.



Problem 2' niemożliwy, gdy założymy sprawiedliwość

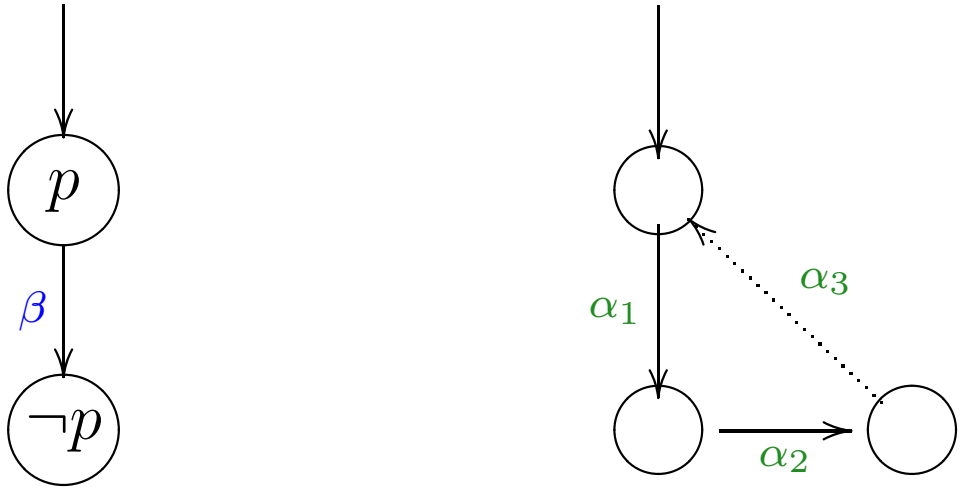
Wystarczy?

Czy **(C0)** – **(C2)** są wystarczające?

Wystarczy?

Czy **(C0)** – **(C2)** są wystarczające?

Nie!



(C3) zabraniamy cykli t. że $\exists \beta$ t. że

$\forall s$ w cyklu, $\beta \in \mathrm{en}_s \setminus \mathrm{ample}_s$

Warunki wystarczające dla poprawności

(C1) jeśli $\mathrm{ample}_s \neq \mathrm{en}_s$, to każda $\alpha \in \mathrm{ample}_s$ jest niewidoczna

(C2) dla każdej ścieżki Π rozpoczynającej się z s:

jeśli $\alpha \in \mathrm{ample}_s,\ \beta \notin \mathrm{ample}_s,\ \alpha D \beta$

to β nie może być wykonane w Π

zanim nie wykona się pewna tranzycja z ample_s

(C3) zabraniamy cykli t. że $\exists \beta$ t. że

$\forall s$ w cyklu, $\beta \in \mathrm{en}_s \setminus \mathrm{ample}_s$

Jak to zaimplementować?

Warunki wystarczające dla poprawności

(C1) łatwy

(C2) trudny, policzymy jego przybliżenie

- relacja D będzie policzona w sposób przybliżony
- warunek **(C2)** jest monotoniczny
- analiza statyczna zamiast dynamicznej

(C3) zastąpimy przez łatwiejszy ale mocniejszy:

(C3') jeśli $\mathrm{ample}_s \neq \mathrm{en}_s$ to $\forall \alpha \in \mathrm{ample}_s \ \alpha(s) \notin$ stos

Def.:

- $\mathrm{pc}_i(s)$ punkt sterowania (*ang. program counter*)

  w procesie p_i

- $T_i \subseteq T$ operacje (tranzycje) procesu i

- $T_i(s) := T_i \cap \mathrm{en}_s$ $T_i(s)$ to kandydat na ample_s

- $\mathrm{current}_i(s) := \bigcup\{T_i(s') \mid \mathrm{pc}_i(s') = \mathrm{pc}_i(s)\}$

$T_i(s) \subseteq \mathrm{current}_i(s)$

Decyzja:

$$\text{ample}_s = T_i(s)$$

Decyzja:

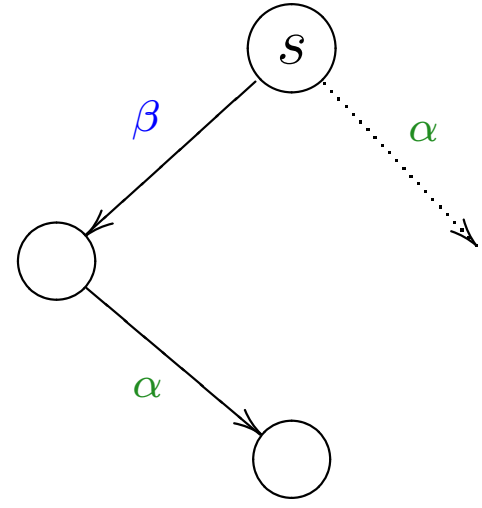
$\mathrm{ample}_s = T_i(s)$

o ile

- $T_i(s)$ niezależne od wszystkich T_j
- żadna operacja $\in \mathrm{current}_i(s) \setminus T_i(s)$ nie może zostać umożliwiona przez procesy p_j

Def.:

- $\mathrm{pre}(\alpha) \supseteq \{\beta \mid \exists s.\ \beta \in \mathrm{en}_s,\ \alpha \notin \mathrm{en}_s,\ \alpha \in \mathrm{en}_{\beta(s)}\}$



- $\mathrm{dep}(\alpha) := \{\beta \mid \alpha D \beta\}$

Uwaga: $\mathrm{pre}(\alpha)$ i $\mathrm{dep}(\alpha)$ (a właściwie D) są obliczane w sposób przybliżony

$\mathrm{pre}(\alpha)$ (przybliżenie z góry)

– $\mathrm{pre}(\alpha)$ zawiera wszystkie operacje β, które zmieniają pc tak, że można wykonać α

– jeśli warunek umożliwiający dla α zależy od zmiennych globalnych, to $\mathrm{pre}(\alpha)$ zawiera wszystkie operacje modyfikujące te zmienne

– jeśli α to pisanie/czytanie z bufora, to $\mathrm{pre}(\alpha)$ zawiera wszystkie operacje czytania/pisania do tego bufora

$\alpha D \beta$ (przybliżenie z góry)

– α i β używają tej samej zmiennej dzielonej,

  a przynajmniej jedna z nich ją modyfikuje (z góry)

– α i β są w tym samym procesie; komunikacja synchroniczna jest uważana za należącą do obydwu procesów

– α i β piszą/czytają do tego samego bufora

Ale pisanie i czytanie z tego samego bufora jest niezależne!

Co pozostaje niezależne?

Przykład:

Instrukcje niezależne od wszytkich instrukcji innych procesów:

- operacje na zmiennych lokalnych
- czytanie z bufora gdy xr
- pisanie do bufora gdy xs
- test `nempty(q)` bufora gdy xr dla q
- test `nfull(q)` bufora gdy xs dla q

Implementacja

function ample(s)

for all P_i **such that** $T_i(s) \neq \emptyset$

if C1($T_i(s)$) **and** C2(s, i) **and** C3(s, $T_i(s)$) **then**

return $T_i(s)$;

return en_s

Implementacja

function `C1`(X) ...

function `C2`(s, i)

for all $P_j \neq P_i$

if $\mathrm{dep}(T_i(s)) \cap T_j \neq \emptyset$ **or**

$\mathrm{pre}(\mathrm{current}_i(s) \setminus T_i(s)) \cap T_j \neq \emptyset$ **then**

return `false`;

return `true`

β zabronione w **(C2)** jest:

w $T_j \neq T_i$

w T_i

function `C3`(s, X) ...

Redukcja cz.-p. a weryfikacja w locie

– w każdym z DFS'ów zbiór ample_s musi być ten sam

– warunek **(C3')** stosujemy do $M \times \mathcal{A}_{\neg\phi}$ zamiast do M

czy to jest poprawne?