Automaty ze stosem wyższego rzędu

Paweł Parys

Uniwersytet Warszawski

Automaty ze stosem wyższego rzędu (HOPDA) [Maslov 74, 76]

Stos rzędu 1 to zwykły stos.
Stos rzędu 2 (rzędu n + 1) to stos stosów rzędu 1 (rzędu n).

Operacje na stosach rzędu 2:
(s_i to stosy, szczyt stosu jest po prawej)

$push_2$: $[s_1 ... s_{i-1} s_i]$ -> $[s_1 ... s_{i-1} s_i s_i]$

pop_2 : $[s_1 ... s_{i-1} s_i]$ -> $[s_1 ... s_{i-1}]$

$push_1 x$: $[s_1 ... s_{i-1} [a_1 ... a_{j-1} a_j]]$ -> $[s_1 ... s_{i-1} [a_1 ... a_{j-1} a_j x]]$

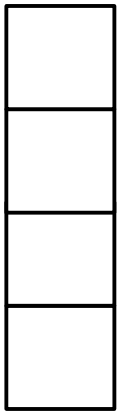
pop_1 : $[s_1 ... s_{i-1} [a_1 ... a_{j-1} a_j]]$ -> $[s_1 ... s_{i-1} [a_1 ... a_{j-1}]]$

Automat rzędu n ma stos rzędu n, dysponuje operacjami $push_i$ oraz pop_i dla każdego $1 \leq i \leq n$.

Automaty ze stosem wyższego rzędu (HOPDA)

Przykład: język $\{a^n b^n c^n\}$

- przy każdym “a” na wejściu włóż symbol na stos

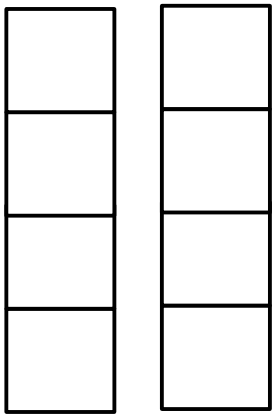


a a a a

Automaty ze stosem wyższego rzędu (HOPDA)

Przykład: język $\{a^n b^n c^n\}$

- przy każdym “a” na wejściu włóż symbol na stos
- skopiuj stos (operacja $push_2$)

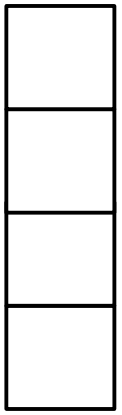


a a a a b

Automaty ze stosem wyższego rzędu (HOPDA)

Przykład: język $\{a^n b^n c^n\}$

- przy każdym “a” na wejściu włóż symbol na stos
- skopiuj stos (operacja $push_2$)
- przy każdym “b” na wejściu zdejmij symbol ze stosu



a a a a b b b b

Automaty ze stosem wyższego rzędu (HOPDA)

Przykład: język $\{a^n b^n c^n\}$

- przy każdym “a” na wejściu włóż symbol na stos
- skopiuj stos (operacja $push_2$)
- przy każdym “b” na wejściu zdejmij symbol ze stosu
- przy każdym “c” na wejściu zdejmij symbol ze stosu

a a a a b b b b c c c c

HOPDA z operacją paniki (“collapse”)

HOPDA z operacją paniki to rozszerzenie automatów ze stosem wyższego rzędu

Teraz elementy stosów rzędu 1 to krotki $(a,n_1,\ldots,n_k)$, dla $a \in \Sigma$, $n_i \in \mathbb{N}$.

(Możliwość wykonania operacji zależy tylko od *a*.)

$push_1 a$ - włóż $(a,n_1,\ldots,n_k)$ na szczyt najwyższego stosu rzędu 1, gdzie n_i to rozmiar najwyższego stosu rzędu i

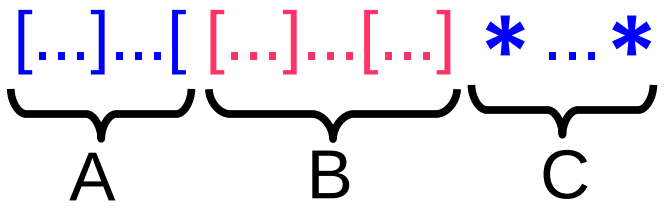
$collapse_i$ - pozostaw tylko n_i-1 elementów na najwyższym stosie rzędu i, (gdzie n_i odczytane z najwyższego symbolu)

(z najwyższego stosu rzędu i usuń wszystkie stosy rzędu i-1 na których jest symbol $(a,n_1,\ldots,n_k)$)

Spostrzeżenie: $collapse_1 = pop_1$

Przykład

alfabet: [,], ∗
Język U zawiera słowa postaci:



- część A jest prefiksem poprawnego wyrażenia nawiasowego, zakończonym [
- część B jest poprawnym wyrażeniem nawiasowym
- części A i C mają tą samą długość

na przykład:
[[ ] [[ ] [[ ]] ∗∗∗∗ ∈U

Jak rozpoznać U automatem z paniką?

- jednoelementowy alfabet stosowy
- na stosach rzędu 1 liczymy liczbę otwartych nawiasów
- wykonujemy kopię ($push_2$) po każdym nawiasie

te liczby to n_2 (rozmiar stosu rzędu 2
przy wstawianiu tego symbolu na stos)

Jak rozpoznać U automatem z paniką?

- jednoelementowy alfabet stosowy
- na stosach rzędu 1 liczymy liczbę otwartych nawiasów
- wykonujemy kopię ($push_2$) po każdym nawiasie

te liczby to n_2 (rozmiar stosu rzędu 2
przy wstawianiu tego symbolu na stos)

Jak rozpoznać U automatem z paniką?

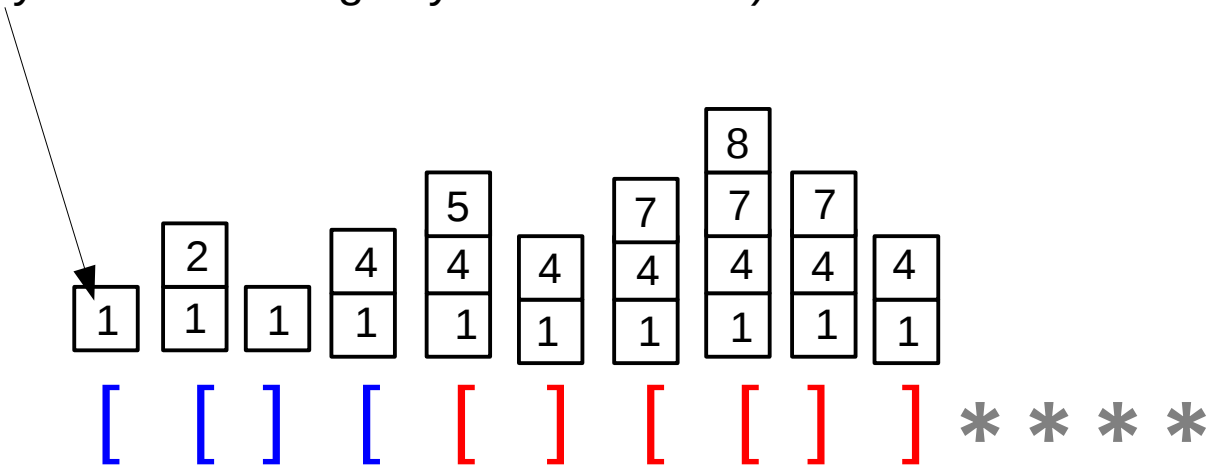
- jednoelementowy alfabet stosowy
- na stosach rzędu 1 liczymy liczbę otwartych nawiasów
- wykonujemy kopię ($push_2$) po każdym nawiasie

te liczby to n_2 (rozmiar stosu rzędu 2
przy wstawianiu tego symbolu na stos)

Jak rozpoznać U automatem z paniką?

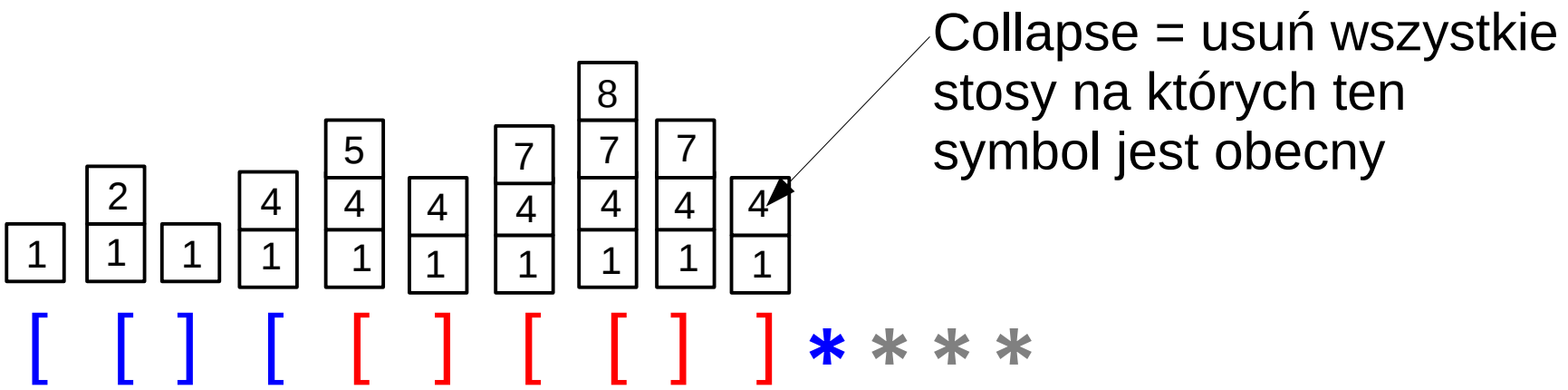
- jednoelementowy alfabet stosowy
- na stosach rzędu 1 liczymy liczbę otwartych nawiasów
- wykonujemy kopię ($push_2$) po każdym nawiasie

te liczby to n_2 (rozmiar stosu rzędu 2
przy wstawianiu tego symbolu na stos)



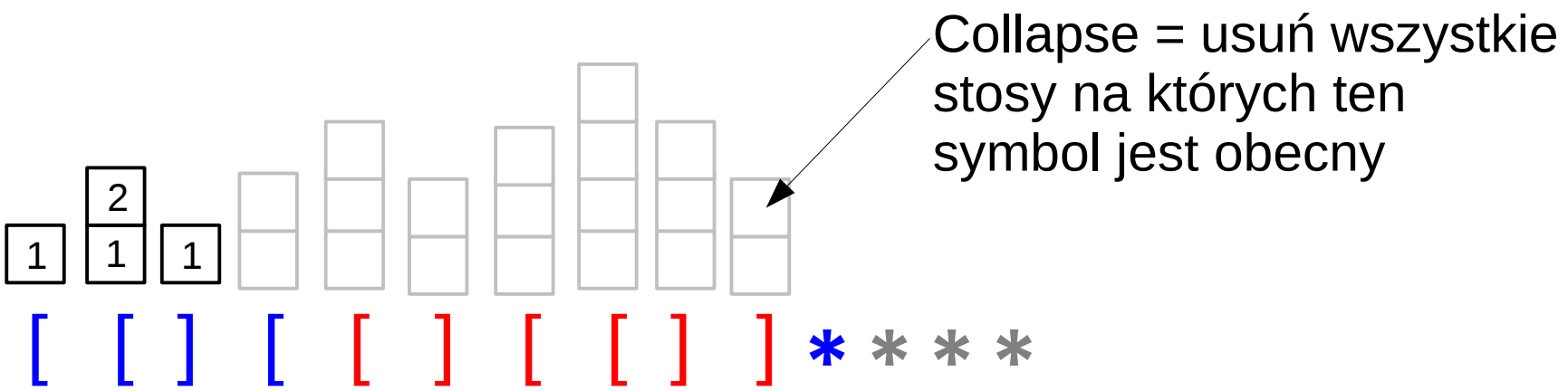
Jak rozpoznać U automatem z paniką?

- jednoelementowy alfabet stosowy
- na stosach rzędu 1 liczymy liczbę otwartych nawiasów
- wykonujemy kopię ($push_2$) po każdym nawiasie
- przy pierwszej gwiazdce robimy “collapse”
- liczymy ile stosów pozostało



Jak rozpoznać U automatem z paniką?

- jednoelementowy alfabet stosowy
- na stosach rzędu 1 liczymy liczbę otwartych nawiasów
- wykonujemy kopię ($push_2$) po każdym nawiasie
- przy pierwszej gwiazdce robimy “collapse”
- liczymy ile stosów pozostało



Motywacja: od weryfikacji programów do automatów ze stosem

Przykład

```
open(x, “foo”)
a := 0
while a<100 do
    read(x)
    a := a+1
close(x)
```

czy dostęp do pliku “foo” wygląda tak: open,read*,close?

Motywacja: od weryfikacji programów do automatów ze stosem

Przykład

Krok 1: przybliż, abstrahując od wartości zmiennych.

```
open(x, “foo”)
a := 0
while a<100 do
    read(x)
    a := a+1
close(x)
```

```
open(x, “foo”)

while * do
    read(x)

close(x)
```

czy dostęp do pliku “foo” wygląda tak: open,read*,close?

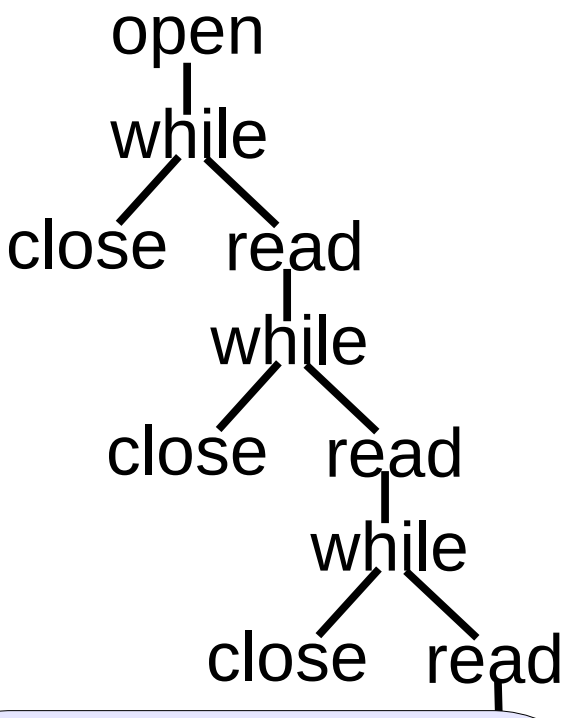
czy dostęp do pliku “foo” wygląda tak: open,read*,close?

Motywacja: od weryfikacji programów do automatów ze stosem

Przykład

Krok 2: rozważ drzewo możliwych przebiegów programu.

```
open(x, “foo”)
while * do
    read(x)
close(x)
```



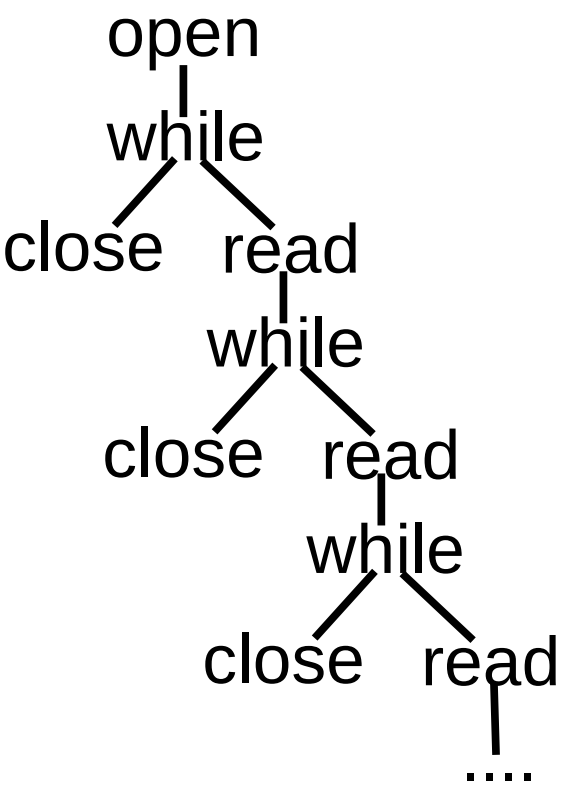
czy dostęp do pliku “foo” wygląda tak: open,read*,close?

czy każda ścieżka jest etykietowana open,read*,close?

Motywacja: od weryfikacji programów do automatów ze stosem

Przykład

```
open(x, “foo”)
while * do
    read(x)
close(x)
```



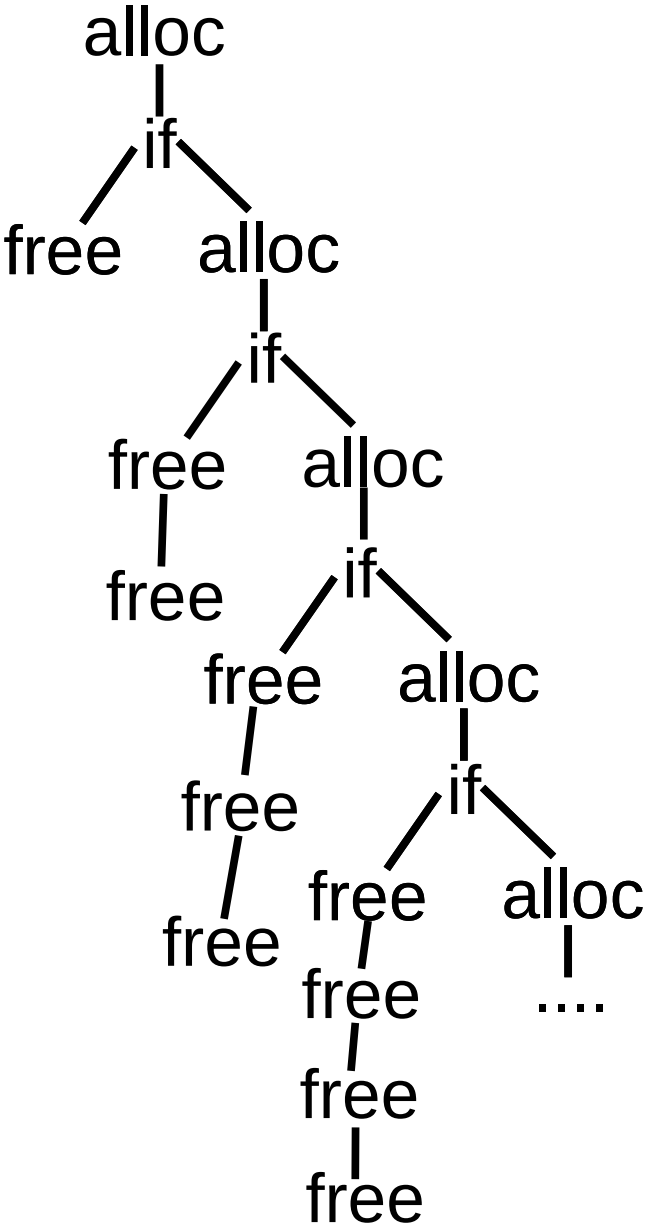
Obserwacja: dla programów bez rekurencji, każda ścieżka takiego drzewa to język regularny.
(program to deterministyczny automat skończony)

Rabin 1969: Drzewa regularne mają rozstrzygalną logikę MSO.

Motywacja: od weryfikacji programów do automatów ze stosem

Przykład 2 - program rekurencyjny

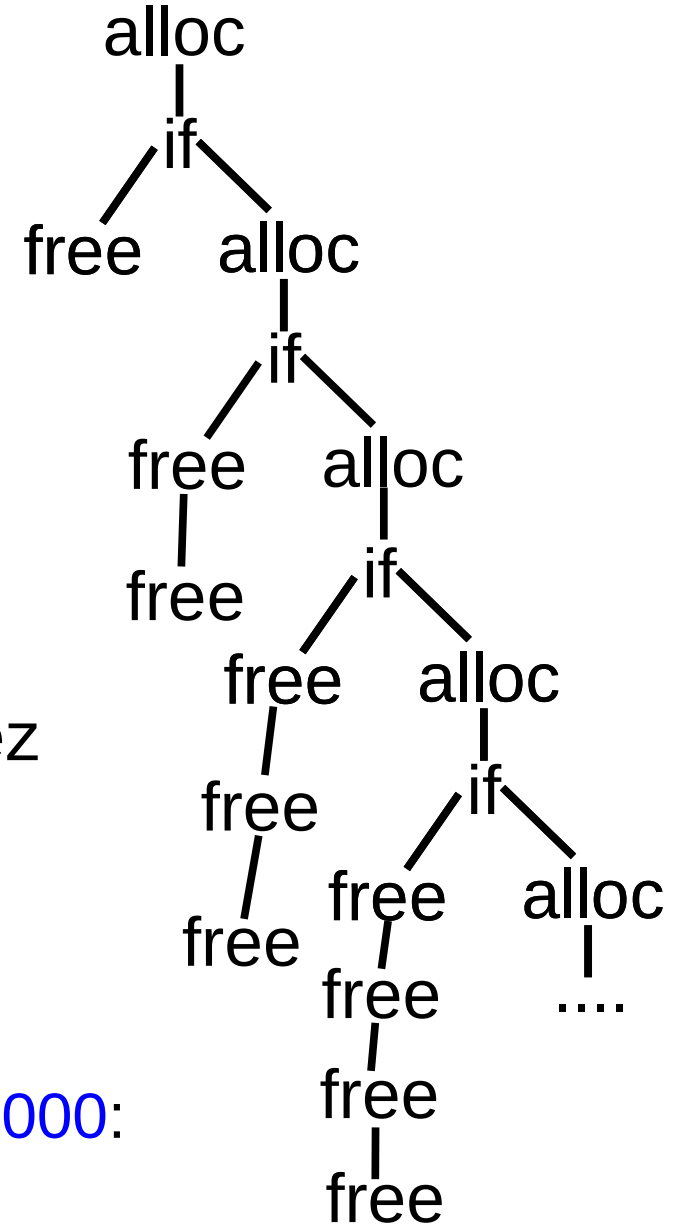
```
let f(x) =
    alloc(x)
    if * then f(x)
    free(x)
f(x)
```



Motywacja: od weryfikacji programów do automatów ze stosem

Przykład 2 - program rekurencyjny

```
let f(x) =
    alloc(x)
    if * then f(x)
    free(x)
f(x)
```



To drzewo nie jest już regularne!!

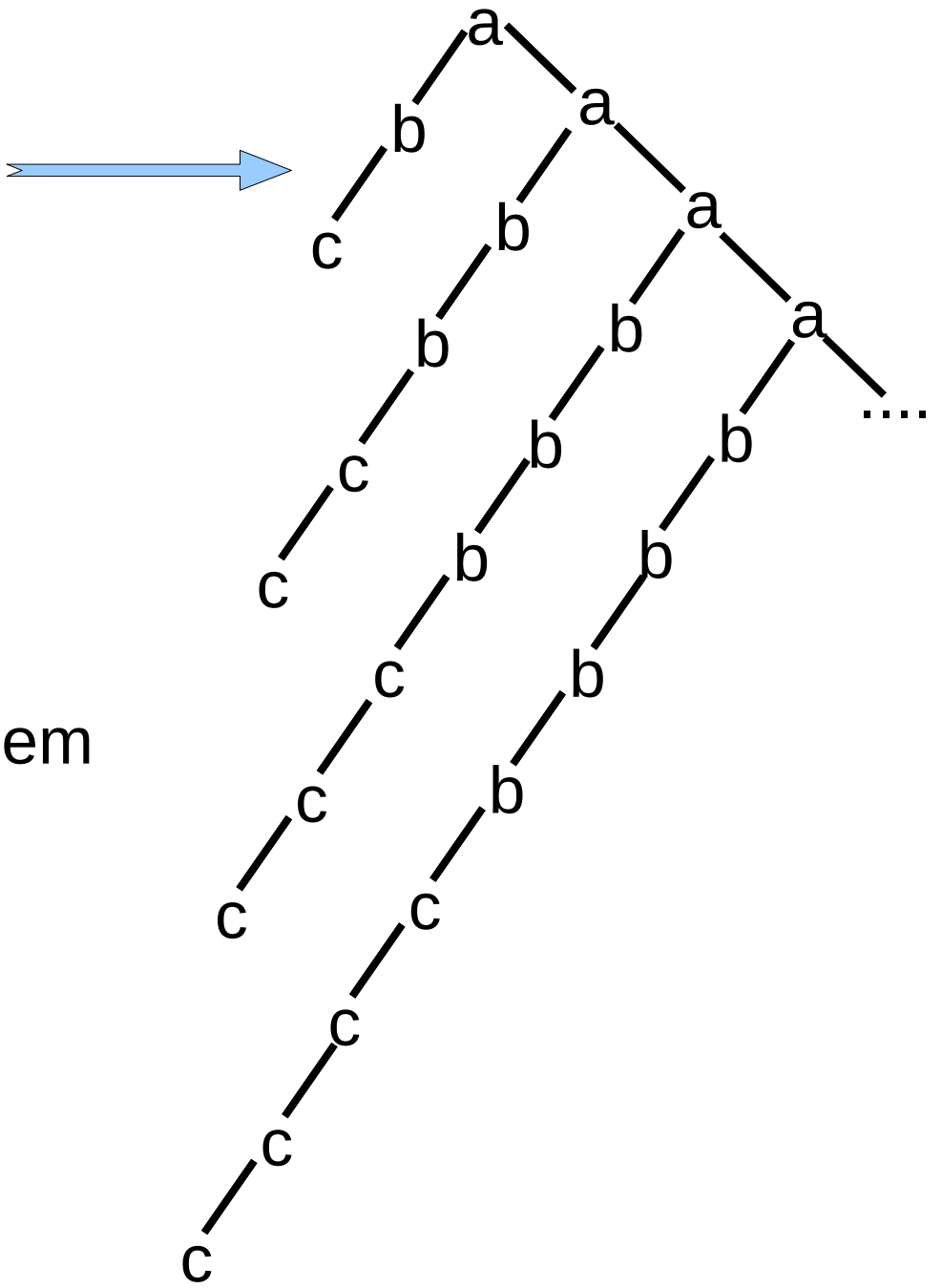
Ale każda ścieżka jest rozpoznawana przez **deterministyczny** automat ze stosem.

Muller, Schupp 1985 / Caucal 1986 / Stirling 2000: takie drzewa mają rozstrzygalną logikę MSO.

Motywacja: od weryfikacji programów do automatów ze stosem

A co z rekurencją wyższego rzędu?

```
let f(x, g) =
    a(x)
    if * then g(x)
    else f(x, fun h x -> h(x); b(x))
    c(x)
f(x, b)
```



Potrzebne są automaty ze stosem wyższego rzędu!!!

Relacja między programami a automatami

Pomijamy ścisłą definicję

Dla każdego rzędu mamy dwie klasy (drzew):

$\mathbf{PdaTree}_n\,\Sigma$ = drzewa generowane przez automaty ze stosem rzędu n

$\mathbf{RecSchTree}_n\,\Sigma$ = drzewa generowane przez schematy rekurencyjne (programy) rzędu n

Czy te klasy są takie same?

Relacja między programami a automatami

Dla każdego rzędu mamy dwie klasy (drzew):

$\mathbf{PdaTree}_n\Sigma$ = drzewa generowane przez automaty ze stosem rzędu n

$\mathbf{SafeRecSchTree}_n\Sigma$ = drzewa generowane przez bezpieczne schematy rekurencyjne (programy) rzędu n

Czy te klasy są takie same?

Knapik, Niwiński, Urzyczyn 2002:

$\mathbf{PdaTree}_n\Sigma = \mathbf{SafeRecSchTree}_n\Sigma$ (dla każdego n)

oraz te drzewa mają rozstrzygalną logikę MSO.

co to znaczy "bezpieczne"?

To pewne ograniczenie na składnię schematów rekurencyjnych.

(wynik przekazanai parametrów rzędu k musi być funkcją rzędu mniejszego niż k)

Dla rzędu 1 każdy program jest "bezpieczny".

Relacja między programami a automatami

Dla każdego rzędu mamy dwie klasy (drzew):

$\mathbf{ColPdaTree}_n\,\Sigma$ = drzewa generowane przez automaty z paniką rzędu n

$\mathbf{RecSchTree}_n\,\Sigma$ = drzewa generowane przez schematy rekurencyjne (programy) rzędu n

Czy te klasy są takie same?

Hague, Murawski, Ong, Serre 2008:

$\mathbf{ColPdaTree}_n\,\Sigma = \mathbf{RecSchTree}_n\,\Sigma$ (dla każdego n)

dlatego warto rozważać automaty z paniką

Ong 2006:

Drzewa z $\mathbf{RecSchTree}_n\,\Sigma$ mają rozstrzygalną logikę MSO.

Relacja między programami a automatami

$$\mathbf{PdaTree}_n\Sigma \quad = \quad \mathbf{SafeRecSchTree}_n\Sigma$$

$$= ? \qquad\qquad = ?$$

$$\mathbf{ColPdaTree}_n\Sigma \quad = \quad \mathbf{RecSchTree}_n\Sigma$$

P. 2010:

$\mathbf{PdaTree}_2\Sigma \neq \mathbf{ColPdaTree}_2\Sigma$

P. 2011:

$\mathbf{PdaTree}_2\Sigma \not\subseteq \mathbf{ColPdaTree}_n\Sigma$ (dla każdego n)

Przykład: język U

Dwie hierarchie (drzew / języków słów):

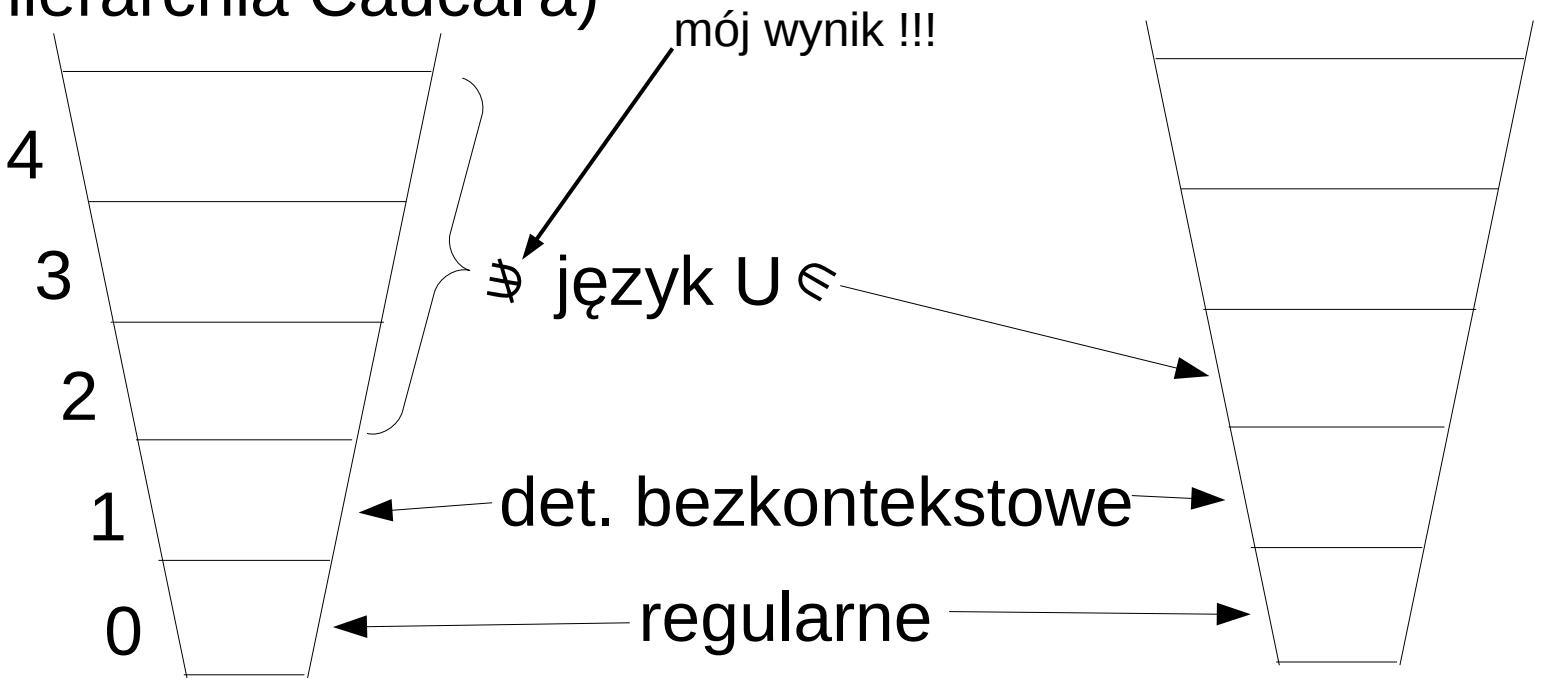
deterministyczne
automaty wyższego rzędu

bezpieczne schematy
rekurencyjne

(hierarchia Caucal'a)

deterministyczne automaty
wyższego rzędu z paniką

schematy rekurencyjne



To są różne hierarchie!!!

Co dalej?

1) Lemat o pompowaniu?

2) Czy operacja “collapse” zwiększa siłę wyrazu **niedeterministycznych** automatów wyższego rzędu?

Aehlig, Miranda, Ong 2005: dla poziomu 2 – NIE
(“collapse” może być symulowana przez niedeterminizm)

ale:
- automaty niedeterministyczne nie mają naturalnych powiązań z weryfikacją programów
- większosć problemów jest nierozstrzygalna, (nawet “czy automat rzędu 1 akceptuje wszystkie słowa?”)