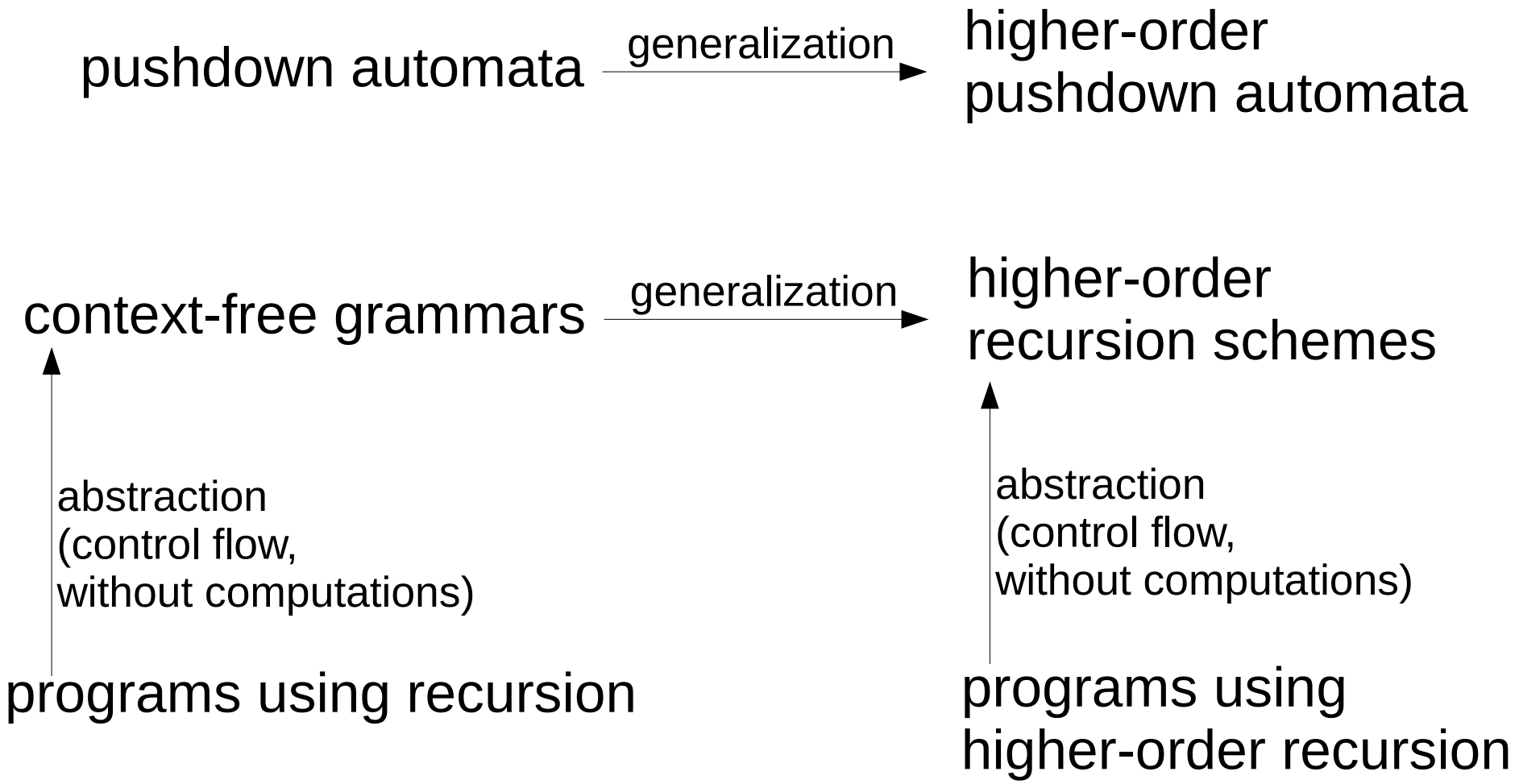
Automaty a rekurencja

Paweł Parys

Informacje ogólne

1)Materiały do wykładu będą na stronie:
http://www.mimuw.edu.pl/~parys/teaching/aar2016/
(slajdy / streszczenie / zeskanowane zapiski / linki)
2)Konsultacje: wtorek: 8:30-12:00, środa 8:30-12:00
(najlepiej dać znać na email: parys@mimuw.edu.pl)
3)Zachęcam do zadawania pytań, gdy coś nie jest jasne.
4)Ćwiczenia zaczynamy o 14:00.
5)Zaliczanie:
– zadania domowe (do 2 pkt)
– notatki do wykładu (do 2 pkt, po 1 pkt za wykład)
w Latexu, do 2 tyg. po wykładzie, proszę się zgłaszać
– egzamin z teorii (do 2 pkt)
ocena = 2+liczba_punktów

Higher-order recursion schemes



Higher-order pushdown automata - definition

A *1*-stack is an ordinary stack. A *2*-stack (resp. *(n+1)*-stack) is a stack of *1*-stacks (resp. *n*-stack).

[[*abc*][*de*][*fghij*]] – order 2
[[[*abc*][*de*][*xy*]][[*ab*][*fghij*]]] – order 3

Higher-order pushdown automata - definition

A 1-stack is an ordinary stack. A 2-stack (resp. $(n+1)$-stack) is a stack of 1-stacks (resp. n-stack).

Operations on 2-stacks: (top of stack is on right)

$push_1 x$: $[[abc][de][fghij]] \rightarrow [[abc][de][fghijx]]$

pop_1 : $[[abc][de][fghij]] \rightarrow [[abc][de][fghi]]$

Higher-order pushdown automata - definition

A 1-stack is an ordinary stack. A 2-stack (resp. $(n+1)$-stack) is a stack of 1-stacks (resp. n-stack).

Operations on 2-stacks: (top of stack is on right)

$push_1 x$: $[[abc][de][fghij]] \rightarrow [[abc][de][fghijx]]$

pop_1 : $[[abc][de][fghij]] \rightarrow [[abc][de][fghi]]$

$push_2$: $[[abc][de][fghij]] \rightarrow [[abc][de][fghij][fghij]]$

pop_2 : $[[abc][de][fghij]] \rightarrow [[abc][de]]$

Higher-order pushdown automata - definition

A 1-stack is an ordinary stack. A 2-stack (resp. $(n+1)$-stack) is a stack of 1-stacks (resp. n-stack).

Operations on 2-stacks: (top of stack is on right)

$push_1 x$: $[[abc][de][fghij]] \rightarrow [[abc][de][fghijx]]$

pop_1 : $[[abc][de][fghij]] \rightarrow [[abc][de][fghi]]$

$push_2$: $[[abc][de][fghij]] \rightarrow [[abc][de][fghij][fghij]]$

pop_2 : $[[abc][de][fghij]] \rightarrow [[abc][de]]$

An **order-n PDA** has an order-n stack, and has $push_i$ and pop_i for each $i \in \{1, ..., n\}$.

The next operation depends on the topmost stack symbol, the state, and the next letter on the input.

Higher-order pushdown automata - example

Language: $\{b^{2^k} : k \in \mathbb{N}\}$

- order 2
- 3 stack symbols: $\perp$, x, #

$(\_, q_1) \xrightarrow{\varepsilon} (q_1, \mathrm{push}_1(x))$

dowolny symbol (skrót)

q_1 $\boxed{\perp}$

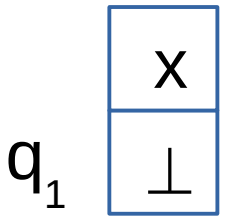
Higher-order pushdown automata - example

Language: $\{b^{2^k} : k \in \mathbb{N}\}$

- order 2
- 3 stack symbols: $\perp$, x, #

$(\_, q_1) \xrightarrow{\varepsilon} (q_1, \mathrm{push}_1(x))$

dowolny symbol (skrót)



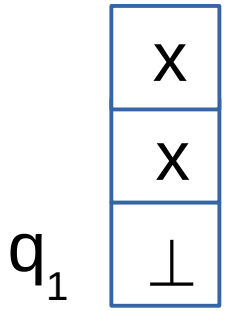
Higher-order pushdown automata - example

Language: $\{b^{2^k} : k \in \mathbb{N}\}$

- order 2
- 3 stack symbols: $\perp$, x, #

$(\_, q_1) \xrightarrow{\varepsilon} (q_1, push_1(x))$

dowolny symbol (skrót)



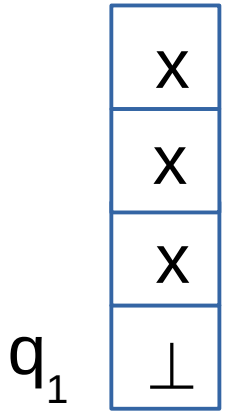
Higher-order pushdown automata - example

Language: $\{b^{2^k} : k \in \mathbb{N}\}$

- order 2
- 3 stack symbols: $\perp$, x, #

$(\_, q_1) \xrightarrow{\varepsilon} (q_1, \mathrm{push}_1(x))$

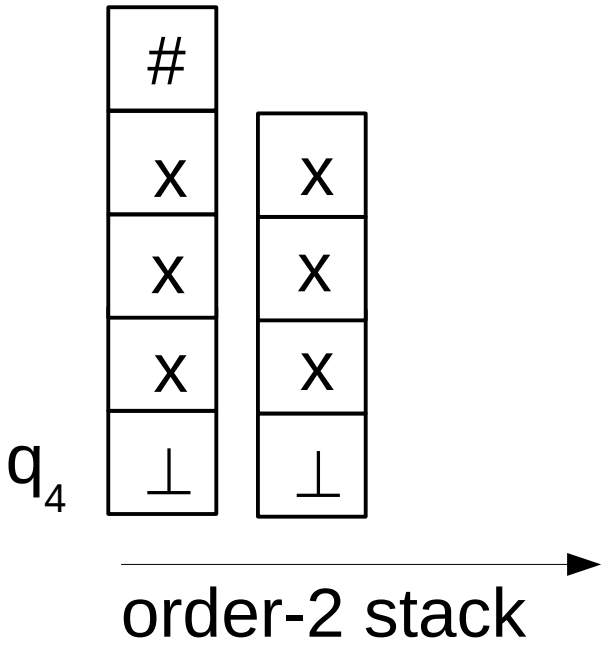
dowolny symbol (skrót)



Higher-order pushdown automata - example

Language: $\{b^{2^k} : k \in \mathbb{N}\}$

- order 2
- 3 stack symbols: $\perp$, x, #



$$(\_,q_1) \xrightarrow{\varepsilon} (q_1,\mathrm{push}_1(x))$$

$$(\_,q_1) \xrightarrow{\varepsilon} (q_2,\mathrm{push}_1(\#))$$

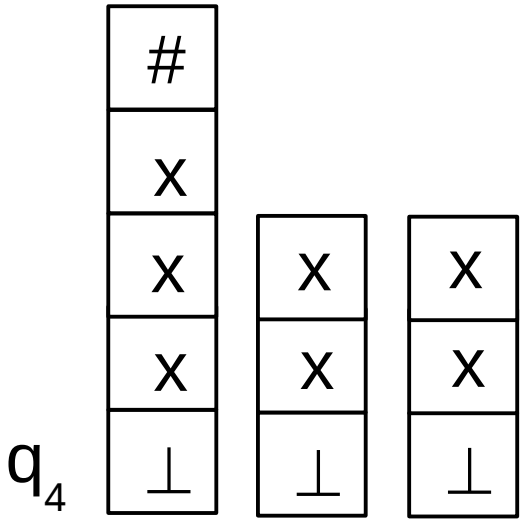
$$(\#,q_2) \xrightarrow{\varepsilon} (q_3,\mathrm{push}_2)$$

$$(\#,q_3) \xrightarrow{\varepsilon} (q_4,\mathrm{pop}_1)$$

Higher-order pushdown automata - example

Language: $\{b^{2^k} : k \in \mathbb{N}\}$

- order 2
- 3 stack symbols: $\perp$, x, #



$(\_,q_1) \xrightarrow{\varepsilon} (q_1,\text{push}_1(x))$

$(\_,q_1) \xrightarrow{\varepsilon} (q_2,\text{push}_1(\#))$

$(\#,q_2) \xrightarrow{\varepsilon} (q_3,\text{push}_2)$

$(\#,q_3) \xrightarrow{\varepsilon} (q_4,\text{pop}_1)$

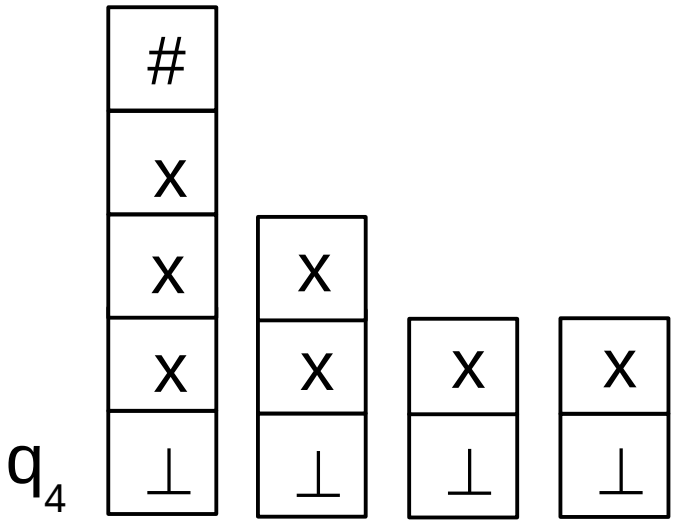
$(x,q_4) \xrightarrow{\varepsilon} (q_5,\text{pop}_1)$

$(\_,q_5) \xrightarrow{\varepsilon} (q_4,\text{push}_2)$

Higher-order pushdown automata - example

Language: $\{b^{2^k} : k \in \mathbb{N}\}$

- order 2
- 3 stack symbols: $\perp$, x, #



$(\_,q_1) \xrightarrow{\varepsilon} (q_1,push_1(x))$

$(\_,q_1) \xrightarrow{\varepsilon} (q_2,push_1(\#))$

$(\#,q_2) \xrightarrow{\varepsilon} (q_3,push_2)$

$(\#,q_3) \xrightarrow{\varepsilon} (q_4,pop_1)$

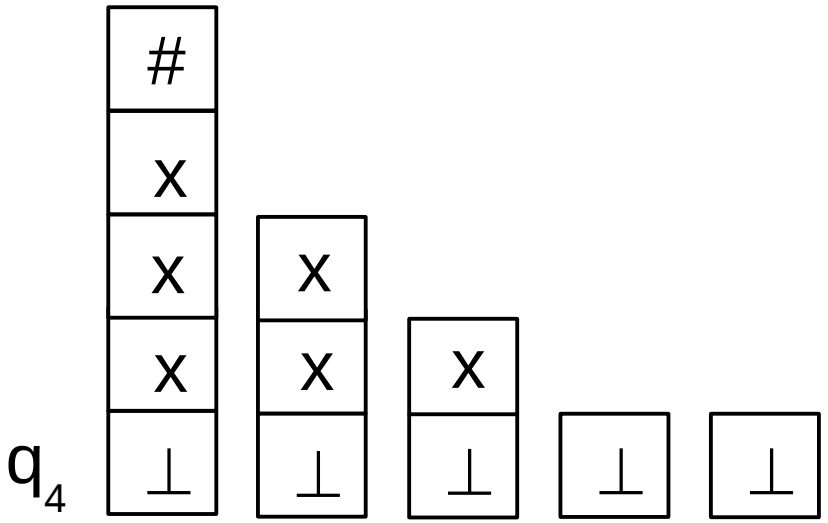
$(x,q_4) \xrightarrow{\varepsilon} (q_5,pop_1)$

$(\_,q_5) \xrightarrow{\varepsilon} (q_4,push_2)$

Higher-order pushdown automata - example

Language: $\{b^{2^k} : k \in \mathbb{N}\}$

- order 2
- 3 stack symbols: $\perp$, x, #



$(\_,q_1) \xrightarrow{\varepsilon} (q_1,\text{push}_1(x))$

$(\_,q_1) \xrightarrow{\varepsilon} (q_2,\text{push}_1(\#))$

$(\#,q_2) \xrightarrow{\varepsilon} (q_3,\text{push}_2)$

$(\#,q_3) \xrightarrow{\varepsilon} (q_4,\text{pop}_1)$

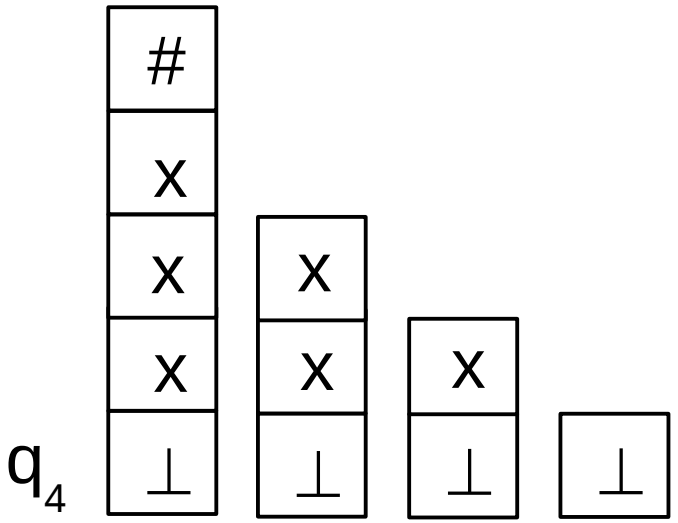
$(x,q_4) \xrightarrow{\varepsilon} (q_5,\text{pop}_1)$

$(\_,q_5) \xrightarrow{\varepsilon} (q_4,\text{push}_2)$

Higher-order pushdown automata - example

Language: $\{b^{2^k} : k \in \mathbb{N}\}$

- order 2
- 3 stack symbols: $\perp$, x, #



$(\_,q_1) \xrightarrow{\varepsilon} (q_1,push_1(x))$

$(\_,q_1) \xrightarrow{\varepsilon} (q_2,push_1(\#))$

$(\#,q_2) \xrightarrow{\varepsilon} (q_3,push_2)$

$(\#,q_3) \xrightarrow{\varepsilon} (q_4,pop_1)$

$(x,q_4) \xrightarrow{\varepsilon} (q_5,pop_1)$

$(\_,q_5) \xrightarrow{\varepsilon} (q_4,push_2)$

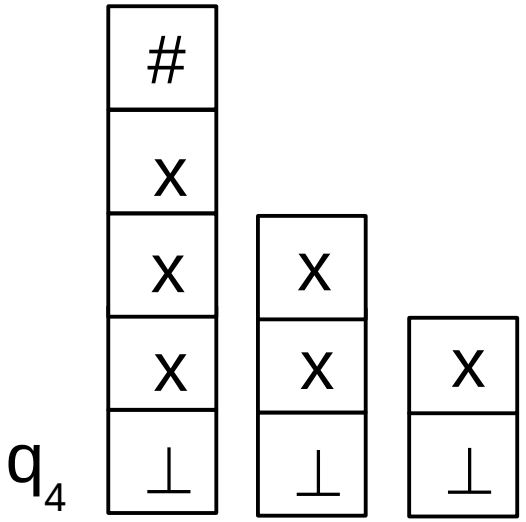
$(\perp,q_4) \xrightarrow{b} (q_4,pop_2)$

Input: b

Higher-order pushdown automata - example

Language: $\{b^{2^k} : k \in \mathbb{N}\}$

- order 2
- 3 stack symbols: $\perp$, x, #



$(\_,q_1) \xrightarrow{\varepsilon} (q_1,push_1(x))$

$(\_,q_1) \xrightarrow{\varepsilon} (q_2,push_1(\#))$

$(\#,q_2) \xrightarrow{\varepsilon} (q_3,push_2)$

$(\#,q_3) \xrightarrow{\varepsilon} (q_4,pop_1)$

$(x,q_4) \xrightarrow{\varepsilon} (q_5,pop_1)$

$(\_,q_5) \xrightarrow{\varepsilon} (q_4,push_2)$

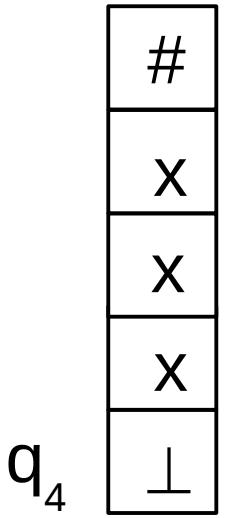
$(\perp,q_4) \xrightarrow{b} (q_4,pop_2)$

Input: b b

Higher-order pushdown automata - example

Language: $\{b^{2^k} : k \in \mathbb{N}\}$

- order 2
- 3 stack symbols: $\perp$, x, #



$(\_,q_1) \xrightarrow{\varepsilon} (q_1,push_1(x))$

$(\_,q_1) \xrightarrow{\varepsilon} (q_2,push_1(\#))$

$(\#,q_2) \xrightarrow{\varepsilon} (q_3,push_2)$

$(\#,q_3) \xrightarrow{\varepsilon} (q_4,pop_1)$

$(x,q_4) \xrightarrow{\varepsilon} (q_5,pop_1)$

$(\_,q_5) \xrightarrow{\varepsilon} (q_4,push_2)$

$(\perp,q_4) \xrightarrow{b} (q_4,pop_2)$

$(\#,q_4) \xrightarrow{\varepsilon} (q_{acc},id)$

Input: b b b b b b b b

1) Dlaczego to dobrze działa?
2) Ten język nie jest bezkontekstowy.

Higher-order pushdown automata

“Traditional” view:

- a nondeterministic HOPDA recognizing a language of words, as on previous slides

“Modern” view:

- a deterministic HOPDA generating a single tree (node-labeled, ranked, ordered, usually infinite)

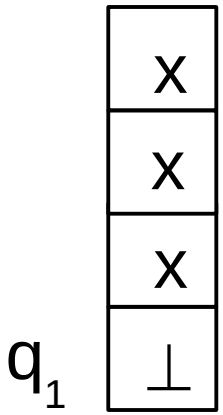
“Graph” view:

- configuration graph of a (deterministic) HOPDA; ε-contraction of this graph

Higher-order pushdown automata - example

nondeterminism – what to do next?

- order 2
- 3 stack symbols: ⊥, x, #



$(\_,q_1) \xrightarrow{\varepsilon} (q_1,\mathrm{push}_1(x))$

$(\_,q_1) \xrightarrow{\varepsilon} (q_2,\mathrm{push}_1(\#))$

$(\#,q_2) \xrightarrow{\varepsilon} (q_3,\mathrm{push}_2)$

$(\#,q_3) \xrightarrow{\varepsilon} (q_4,\mathrm{pop}_1)$

$(x,q_4) \xrightarrow{\varepsilon} (q_5,\mathrm{pop}_1)$

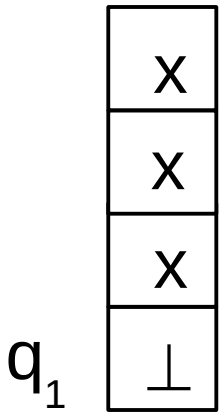
$(\_,q_5) \xrightarrow{\varepsilon} (q_4,\mathrm{push}_2)$

$(\bot,q_4) \xrightarrow{b} (q_4,\mathrm{pop}_2)$

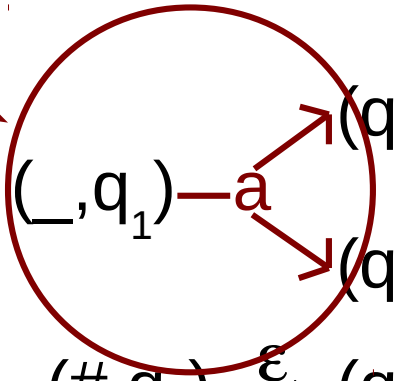
$(\#,q_4) \xrightarrow{\varepsilon} (q_{acc},\mathrm{id})$

Higher-order pushdown automata - example

- order 2
- 3 stack symbols: $\perp$, x, #



letter a of rank 2



$(\#,q_2) \xrightarrow{\varepsilon} (q_3,\text{push}_2)$

$(\#,q_3) \xrightarrow{\varepsilon} (q_4,\text{pop}_1)$

$(x,q_4) \xrightarrow{\varepsilon} (q_5,\text{pop}_1)$

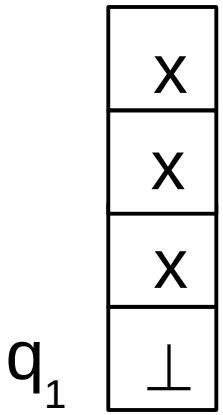
$(\_,q_5) \xrightarrow{\varepsilon} (q_4,\text{push}_2)$

$(\perp,q_4) \xrightarrow{b} (q_4,\text{pop}_2)$

$(\#,q_4) \xrightarrow{\varepsilon} (q_{acc},\text{id})$

Higher-order pushdown automata - example

- order 2
- 3 stack symbols: $\perp$, x, #



$(\_,q_1) \xrightarrow{a} (q_1,\mathrm{push}_1(x))$

$(\_,q_1) \xrightarrow{a} (q_2,\mathrm{push}_1(\#))$

$(\#,q_2) \xrightarrow{\varepsilon} (q_3,\mathrm{push}_2)$

$(\#,q_3) \xrightarrow{\varepsilon} (q_4,\mathrm{pop}_1)$

$(x,q_4) \xrightarrow{\varepsilon} (q_5,\mathrm{pop}_1)$

$(\_,q_5) \xrightarrow{\varepsilon} (q_4,\mathrm{push}_2)$

$(\perp,q_4) \xrightarrow{b} (q_4,\mathrm{pop}_2)$

$(\#,q_4) \xrightarrow{c}$

letter c of rank 0, instead of an accepting state

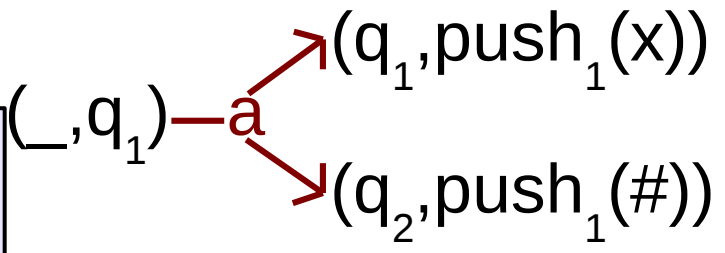
Higher-order pushdown automata - example

- order 2
- 3 stack symbols: ⊥, x, #

Tree-generating HOPDA - definition

From every pair of stack symbol & state there is either:

- one ε-transition
- one transition reading a letter of rank k, resulting in k (ordered) pairs of state & operation.



$(\#,q_2) \xrightarrow{\varepsilon} (q_3,push_2)$

$(\#,q_3) \xrightarrow{\varepsilon} (q_4,pop_1)$

$(x,q_4) \xrightarrow{\varepsilon} (q_5,pop_1)$

$(\_,q_5) \xrightarrow{\varepsilon} (q_4,push_2)$

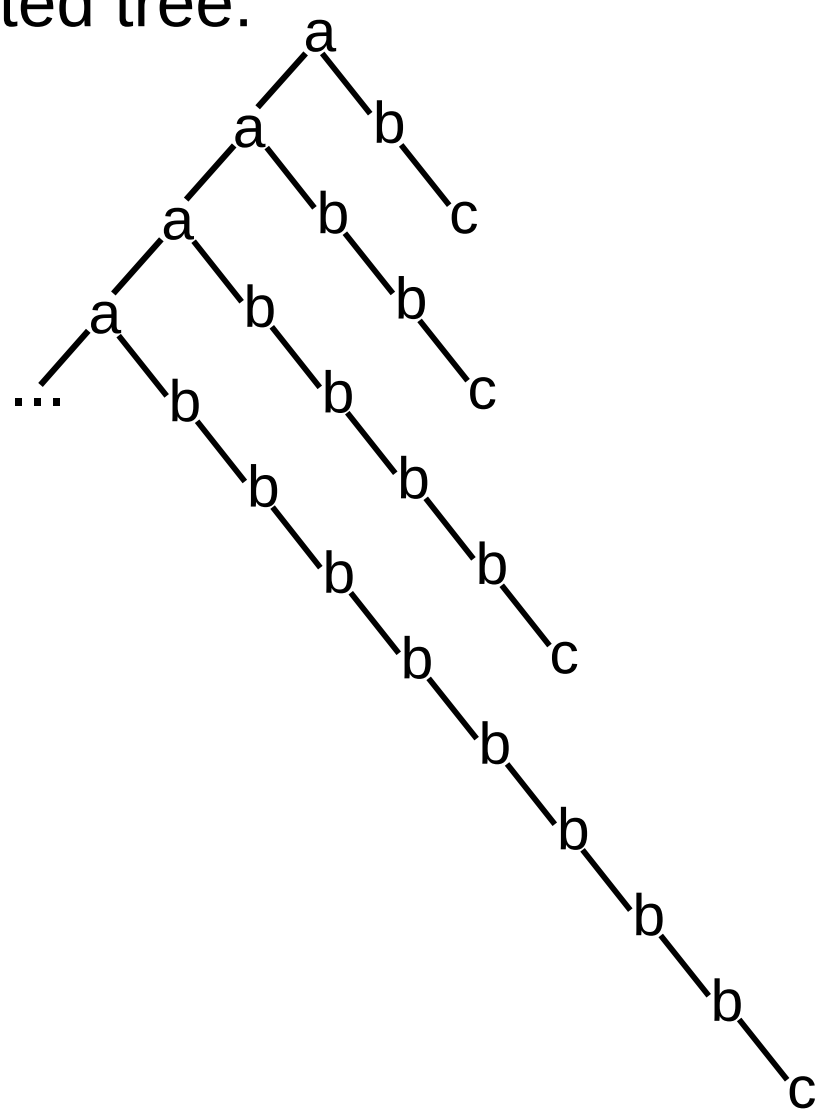
$(\perp,q_4) \xrightarrow{b} (q_4,pop_2)$

$(\#,q_4) \xrightarrow{c}$

Higher-order pushdown automata - example

- order 2
- 3 stack symbols: ⊥, x, #

Generated tree:



$(\_,q_1) \xrightarrow{a} (q_1,push_1(x))$

$(\_,q_1) \xrightarrow{a} (q_2,push_1(\#))$

$(\#,q_2) \xrightarrow{\varepsilon} (q_3,push_2)$

$(\#,q_3) \xrightarrow{\varepsilon} (q_4,pop_1)$

$(x,q_4) \xrightarrow{\varepsilon} (q_5,pop_1)$

$(\_,q_5) \xrightarrow{\varepsilon} (q_4,push_2)$

$(\perp,q_4) \xrightarrow{b} (q_4,pop_2)$

$(\#,q_4) \xrightarrow{c}$

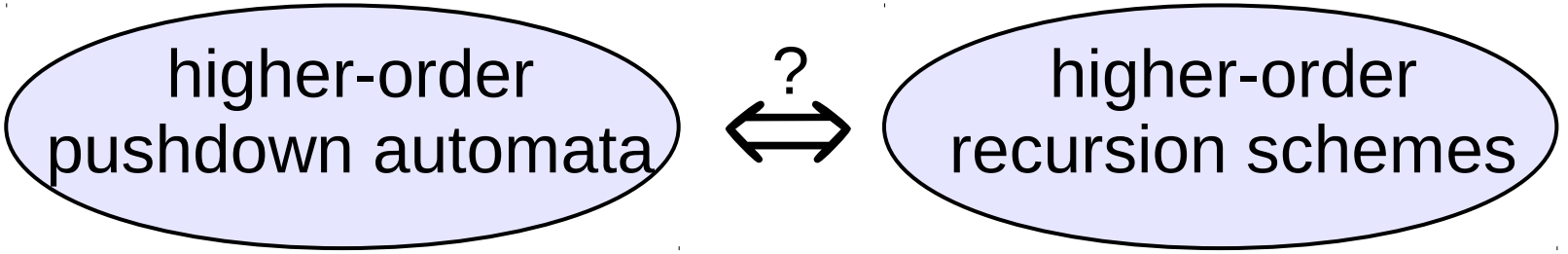
Higher-order recursion schemes - definition

Trochę jak gramatyka bezkontekstowa,
ale nieterminale mogą brać argumenty
i używać ich po prawej stronie produkcji.
Dokładna definicja później.

Inna definicja: otypowany rachunek λ + rekurencja

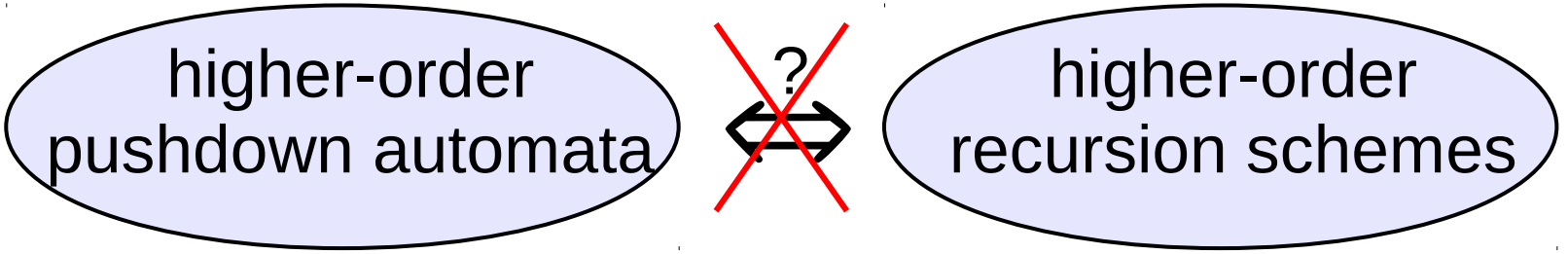
HOPDA vs HORS

Are these two formalisms equivalent?



HOPDA vs HORS

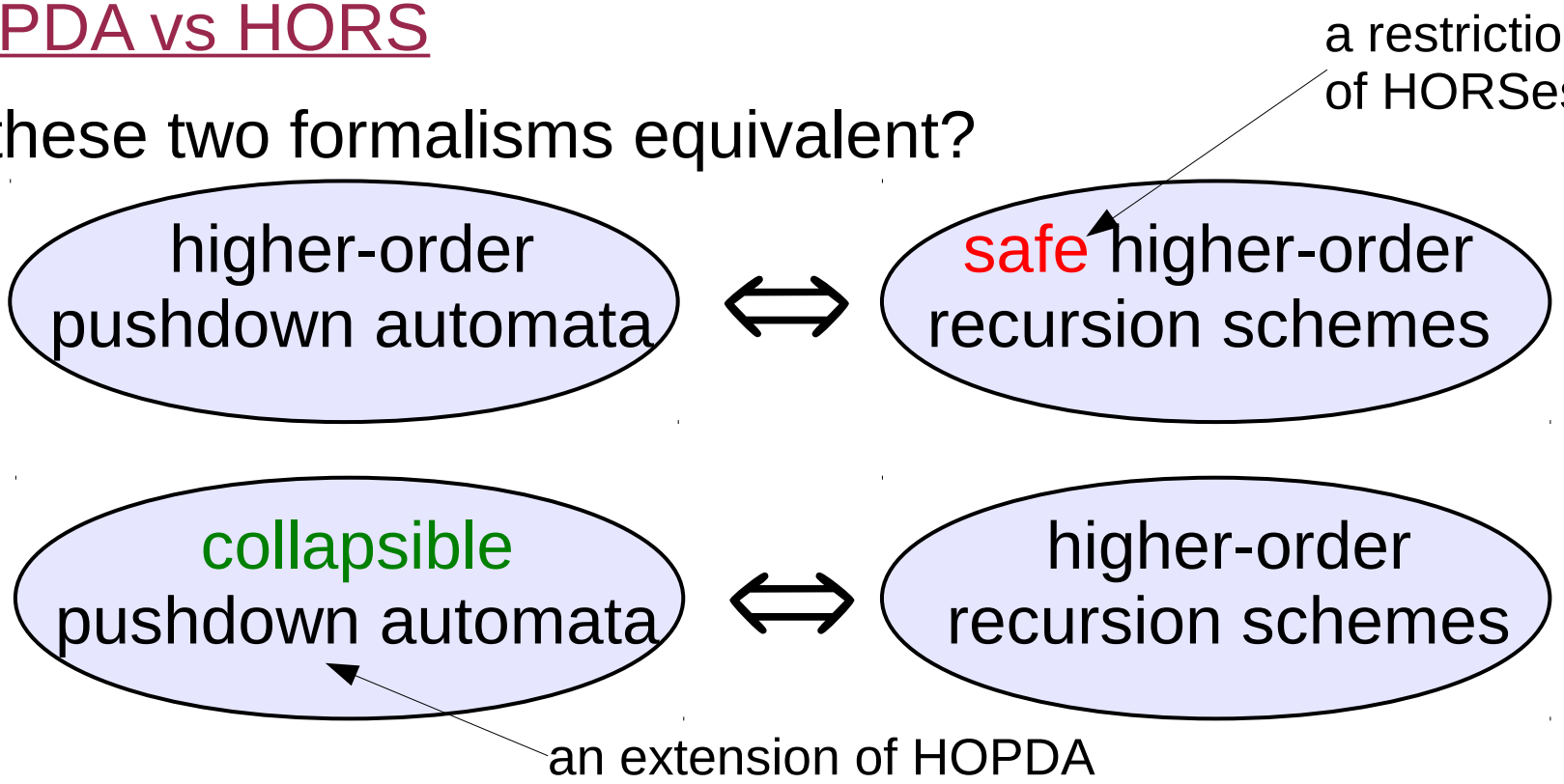
Are these two formalisms equivalent?



Not exactly!

HOPDA vs HORS

Are these two formalisms equivalent?



Theorem

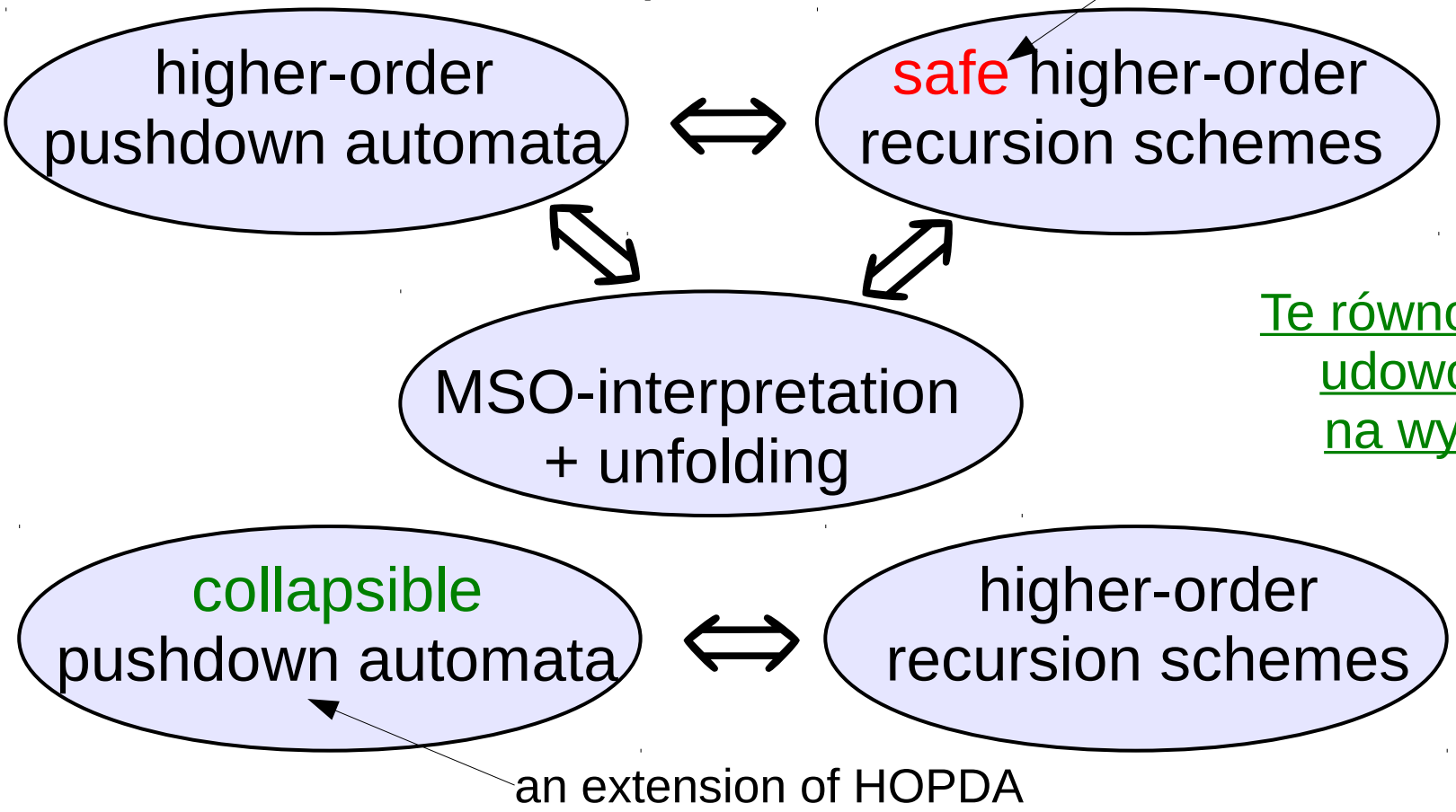
For every n, HOPDA of order n and safe HORSes of order n generate the same trees (recognize the same word languages);

Theorem

For every n, collapsible HOPDA of order n and HORSes of order n generate the same trees (recognize the same word languages).

HOPDA vs HORS

Are these two formalisms equivalent?



Te równoważności udowodnimy na wykładzie

Theorem

For every n, HOPDA of order n and safe HORSes of order n generate the same trees (recognize the same word languages).

These are trees from the Caucal hierarchy, defined by iterating MSO interpretations and unfolding of graphs into trees.

Expressivity questions

Tree(n)= trees generated by HORSes (CPDA) of order n
SafeTree(n) = trees generated by safe HORSes (HOPDA) of order n

$$\begin{array}{ccccccccc} \text{SafeTree}(0) & \subseteq & \text{SafeTree}(1) & \subseteq & \text{SafeTree}(2) & \subseteq & \text{SafeTree}(3) & \subseteq & \ldots \\ \cap & & \cap & & \cap & & \cap & & \\ \text{Tree}(0) & \subseteq & \text{Tree}(1) & \subseteq & \text{Tree}(2) & \subseteq & \text{Tree}(3) & \subseteq & \ldots \end{array}$$

Lang(n) = word languages recogn. by HORSes (CPDA) of order n
SafeLang(n) = word lang. rec. by safe HORSes (HOPDA) of order n

$$\begin{array}{ccccccccc} \text{SafeLang}(0) & \subseteq & \text{SafeLang}(1) & \subseteq & \text{SafeLang}(2) & \subseteq & \text{SafeLang}(3) & \subseteq & \ldots \\ \cap & & \cap & & \cap & & \cap & & \\ \text{Lang}(0) & \subseteq & \text{Lang}(1) & \subseteq & \text{Lang}(2) & \subseteq & \text{Lang}(3) & \subseteq & \ldots \end{array}$$

Expressivity questions

Tree(n)= trees generated by HORSes (CPDA) of order n
SafeTree(n) = trees generated by safe HORSes (HOPDA) of order n

$$\begin{array}{ccccccccc}
\text{SafeTree}(0) & \subseteq & \text{SafeTree}(1) & \subseteq & \text{SafeTree}(2) & \subseteq & \text{SafeTree}(3) & \subseteq & \ldots \\
\| & & \cap & & \cap & & \cap & & \\
\text{Tree}(0) & \subseteq & \text{Tree}(1) & \subseteq & \text{Tree}(2) & \subseteq & \text{Tree}(3) & \subseteq & \ldots \\
\| & & & & & & & & \\
\text{regular} & & & & & & & & \\
\text{trees} & & & & & & & &
\end{array}$$

Lang(n) = word languages recogn. by HORSes (CPDA) of order n
SafeLang(n) = word lang. rec. by safe HORSes (HOPDA) of order n

$$\begin{array}{ccccccccc}
\text{SafeLang}(0) & \subseteq & \text{SafeLang}(1) & \subseteq & \text{SafeLang}(2) & \subseteq & \text{SafeLang}(3) & \subseteq & \ldots \\
\| & & \cap & & \cap & & \cap & & \\
\text{Lang}(0) & \subseteq & \text{Lang}(1) & \subseteq & \text{Lang}(2) & \subseteq & \text{Lang}(3) & \subseteq & \ldots \\
\| & & & & & & & & \\
\text{regular} & & & & & & & & \\
\text{languages} & & & & & & & &
\end{array}$$

Expressivity questions

Tree(n)= trees generated by HORSes (CPDA) of order n
SafeTree(n) = trees generated by safe HORSes (HOPDA) of order n

$$\begin{array}{ccccccccc}
\text{SafeTree}(0) & \subseteq & \text{SafeTree}(1) & \subseteq & \text{SafeTree}(2) & \subseteq & \text{SafeTree}(3) & \subseteq & \ldots \\
= & & = & & \cap & & \cap & & \\
\text{Tree}(0) & \subseteq & \text{Tree}(1) & \subseteq & \text{Tree}(2) & \subseteq & \text{Tree}(3) & \subseteq & \ldots \\
= & & = & & & & & & \\
\text{regular trees} & & \text{context-free trees} & & & & & &
\end{array}$$

Lang(n) = word languages recogn. by HORSes (CPDA) of order n
SafeLang(n) = word lang. rec. by safe HORSes (HOPDA) of order n

$$\begin{array}{ccccccccc}
\text{SafeLang}(0) & \subseteq & \text{SafeLang}(1) & \subseteq & \text{SafeLang}(2) & \subseteq & \text{SafeLang}(3) & \subseteq & \ldots \\
= & & = & & \cap & & \cap & & \\
\text{Lang}(0) & \subseteq & \text{Lang}(1) & \subseteq & \text{Lang}(2) & \subseteq & \text{Lang}(3) & \subseteq & \ldots \\
= & & = & & & & & & \\
\text{regular languages} & & \text{context-free languages} & & & & & &
\end{array}$$

Expressivity questions

Tree(n)= trees generated by HORSes (CPDA) of order n
SafeTree(n) = trees generated by safe HORSes (HOPDA) of order n

$$\begin{array}{ccccccccc}
\text{SafeTree}(0) & \subseteq & \text{SafeTree}(1) & \subseteq & \text{SafeTree}(2) & \subseteq & \text{SafeTree}(3) & \subseteq & \ldots \\
\| & & \| & & \cap\!\| & & \cap\!\| & & \\
\text{Tree}(0) & \subseteq & \text{Tree}(1) & \subseteq & \text{Tree}(2) & \subseteq & \text{Tree}(3) & \subseteq & \ldots \\
\| & & \| & & & & & & \\
\text{regular trees} & & \text{context-free trees} & & & & & &
\end{array}$$

Lang(n) = word languages recogn. by HORSes (CPDA) of order n
SafeLang(n) = word lang. rec. by safe HORSes (HOPDA) of order n

$$\begin{array}{ccccccccc}
\text{SafeLang}(0) & \subseteq & \text{SafeLang}(1) & \subseteq & \text{SafeLang}(2) & \subseteq & \text{SafeLang}(3) & \subseteq & \ldots \\
\| & & \| & & \| & & \cap\!\| & & \\
\text{Lang}(0) & \subseteq & \text{Lang}(1) & \subseteq & \text{Lang}(2) & \subseteq & \text{Lang}(3) & \subseteq & \ldots \\
\| & & \| & & \| & & & & \\
\text{regular languages} & & \text{context-free languages} & & \text{indexed languages} & & & &
\end{array}$$

Expressivity questions

Tree(n)= trees generated by HORSes (CPDA) of order n
SafeTree(n) = trees generated by safe HORSes (HOPDA) of order n

$$\begin{array}{ccccccccc}
\text{SafeTree}(0) & \subseteq & \text{SafeTree}(1) & \subseteq & \text{SafeTree}(2) & \subseteq & \text{SafeTree}(3) & \subseteq & \ldots \\
\| & & \| & & \| & & \| & & \\
\text{Tree}(0) & \subseteq & \text{Tree}(1) & \subseteq & \text{Tree}(2) & \subseteq & \text{Tree}(3) & \subseteq & \ldots \\
\| & & \| & & & & & & \\
\text{regular trees} & & \text{context-free trees} & & & & & &
\end{array}$$

Lang(n) = word languages recogn. by HORSes (CPDA) of order n
SafeLang(n) = word lang. rec. by safe HORSes (HOPDA) of order n

$$\begin{array}{ccccccccc}
\text{SafeLang}(0) & \subseteq & \text{SafeLang}(1) & \subseteq & \text{SafeLang}(2) & \subseteq & \text{SafeLang}(3) & \subseteq & \ldots \\
\| & & \| & & \| & & \| & & \\
\text{Lang}(0) & \subseteq & \text{Lang}(1) & \subseteq & \text{Lang}(2) & \subseteq & \text{Lang}(3) & \subseteq & \ldots \\
\| & & \| & & \| & & & & \\
\text{regular languages} & & \text{context-free languages} & & \text{indexed languages} & & & &
\end{array}$$

Are these hierarchies strict?

Expressivity questions

$$
\begin{array}{ccccccccc}
\mathrm{SafeTree}(0) & \subsetneq & \mathrm{SafeTree}(1) & \subsetneq & \mathrm{SafeTree}(2) & \subsetneq & \mathrm{SafeTree}(3) & \subsetneq & \cdots \\
\| & & \| & & \cap| & & \cap| & & \\
\mathrm{Tree}(0) & \subsetneq & \mathrm{Tree}(1) & \subsetneq & \mathrm{Tree}(2) & \subsetneq & \mathrm{Tree}(3) & \subsetneq & \cdots \\
 & & & & & & & & \\
\mathrm{SafeLang}(0) & \subsetneq & \mathrm{SafeLang}(1) & \subsetneq & \mathrm{SafeLang}(2) & \subsetneq & \mathrm{SafeLang}(3) & \subsetneq & \cdots \\
\| & & \| & & \| & & \cap| & & \\
\mathrm{Lang}(0) & \subsetneq & \mathrm{Lang}(1) & \subsetneq & \mathrm{Lang}(2) & \subsetneq & \mathrm{Lang}(3) & \subsetneq & \cdots
\end{array}
$$

Are these hierarchies strict?

Theorem

For every n, $\mathrm{SafeLang}(n) \neq \mathrm{SafeLang}(n+1)$ & $\mathrm{Lang}(n) \neq \mathrm{Lang}(n+1)$, and thus also $\mathrm{SafeTree}(n) \neq \mathrm{SafeTree}(n+1)$ & $\mathrm{Tree}(n) \neq \mathrm{Tree}(n+1)$.

Separating language: correct sequences of operations of order-$(n+1)$ HOPDA (including the topmost stack symbol after every step).

Będzie na wykładzie.

Expressivity questions

$$\begin{array}{ccccccccc}
\mathrm{SafeTree}(0) & \subsetneq & \mathrm{SafeTree}(1) & \subsetneq & \mathrm{SafeTree}(2) & \subsetneq & \mathrm{SafeTree}(3) & \subsetneq & \cdots \\
\| & & \| & & \cap| & & \cap| & & \\
\mathrm{Tree}(0) & \subsetneq & \mathrm{Tree}(1) & \subsetneq & \mathrm{Tree}(2) & \subsetneq & \mathrm{Tree}(3) & \subsetneq & \cdots
\end{array}$$

$$\begin{array}{ccccccccc}
\mathrm{SafeLang}(0) & \subsetneq & \mathrm{SafeLang}(1) & \subsetneq & \mathrm{SafeLang}(2) & \subsetneq & \mathrm{SafeLang}(3) & \subsetneq & \cdots \\
\| & & \| & & \| & & \cap| & & \\
\mathrm{Lang}(0) & \subsetneq & \mathrm{Lang}(1) & \subsetneq & \mathrm{Lang}(2) & \subsetneq & \mathrm{Lang}(3) & \subsetneq & \cdots
\end{array}$$

Are these hierarchies strict?

Another separator:

T_n = tree with branches $a^k b^{exp_n(k)} c$, where $exp_n(k) = 2^{2^{\cdots^{2^k}}}$ (a tower of n twos)

We have SafeTree($n+1$) $\ni T_n \notin$ Tree(n).

Expressivity questions

$$
\begin{array}{ccccccccc}
\mathrm{SafeTree}(0) & \subsetneq & \mathrm{SafeTree}(1) & \subsetneq & \mathrm{SafeTree}(2) & \subsetneq & \mathrm{SafeTree}(3) & \subsetneq & \cdots \\
\| & & \| & & \cap & & \cap & & \\
\mathrm{Tree}(0) & \subsetneq & \mathrm{Tree}(1) & \subsetneq & \mathrm{Tree}(2) & \subsetneq & \mathrm{Tree}(3) & \subsetneq & \cdots \\
 & & & & & & & & \\
\mathrm{SafeLang}(0) & \subsetneq & \mathrm{SafeLang}(1) & \subsetneq & \mathrm{SafeLang}(2) & \subsetneq & \mathrm{SafeLang}(3) & \subsetneq & \cdots \\
\| & & \| & & \| & & \cap & & \\
\mathrm{Lang}(0) & \subsetneq & \mathrm{Lang}(1) & \subsetneq & \mathrm{Lang}(2) & \subsetneq & \mathrm{Lang}(3) & \subsetneq & \cdots
\end{array}
$$

Is safety really a restriction?

Expressivity questions

$$\begin{array}{ccccccccc}
\mathrm{SafeTree}(0) & \subsetneq & \mathrm{SafeTree}(1) & \subsetneq & \mathrm{SafeTree}(2) & \subsetneq & \mathrm{SafeTree}(3) & \subsetneq & \dots \\
\| & & \| & & \not\| & & \not\| & & \\
\mathrm{Tree}(0) & \subsetneq & \mathrm{Tree}(1) & \subsetneq & \mathrm{Tree}(2) & \subsetneq & \mathrm{Tree}(3) & \subsetneq & \dots \\
 & & & & & & & & \\
\mathrm{SafeLang}(0) & \subsetneq & \mathrm{SafeLang}(1) & \subsetneq & \mathrm{SafeLang}(2) & \subsetneq & \mathrm{SafeLang}(3) & \subsetneq & \dots \\
\| & & \| & & \| & & \not\| & & \\
\mathrm{Lang}(0) & \subsetneq & \mathrm{Lang}(1) & \subsetneq & \mathrm{Lang}(2) & \subsetneq & \mathrm{Lang}(3) & \subsetneq & \dots
\end{array}$$

Is safety really a restriction?

For trees – yes.
Example: Urzyczyn's tree U

$\mathrm{Tree}(2) \ni U \notin \mathrm{SafeTree}(n)$ for every n

For word languages – open problem (e.g. $\mathrm{SafeLang}(3) \stackrel{?}{\neq} \mathrm{Lang}(3)$)

Nie będzie na wykładzie.

Expressivity questions

$$
\begin{array}{ccccccccc}
\text{SafeLang}(0) & \subsetneq & \text{SafeLang}(1) & \subsetneq & \text{SafeLang}(2) & \subsetneq & \text{SafeLang}(3) & \subsetneq & \ldots \\
= & & = & & = & & \cap & & \\
\text{Lang}(0) & \subsetneq & \text{Lang}(1) & \subsetneq & \text{Lang}(2) & \subsetneq & \text{Lang}(3) & \subsetneq & \ldots
\end{array}
$$

Are these languages context-sensitive?

Expressivity questions

$$\begin{array}{ccccccccc}
\text{SafeLang}(0) & \subsetneq & \text{SafeLang}(1) & \subsetneq & \text{SafeLang}(2) & \subsetneq & \text{SafeLang}(3) & \subsetneq & \cdots \subseteq \text{CSens} \\
\| & & \| & & \| & & \cap & & \\
\text{Lang}(0) & \subsetneq & \text{Lang}(1) & \subsetneq & \text{Lang}(2) & \subsetneq & \text{Lang}(3) & \subsetneq & \cdots \\
 & & & & & & \cap & & \\
 & & & & & & \text{CSens} & &
\end{array}$$

Are these languages context-sensitive?

CSens = context-sensitive languages (type-1 in the Chomsky hierarchy)

SafeLang(n) $\subseteq$ CSens, for every n

Lang(3) $\subseteq$ CSens

Lang(n) $\overset{?}{\subseteq}$ CSens for $n \geq 4$ – open problem

Nie będzie na wykładzie.

Algorithmic questions

Problem: MSO model-checking
Input: MSO formula ϕ, HORS S
Output: does ϕ hold in the tree generated by S?

lub równoważnie:

Problem: MSO model-checking
Input: nondeterministic (alternating) parity automaton A,
HORS S
Output: does A accept the tree generated by S?

Można też rozważać słabsze logiki / modele automatów.

Algorithmic questions

Problem: MSO model-checking
Input: MSO formula ϕ, HORS S
Output: does ϕ hold in the tree generated by S?

Theorem: MSO model-checking is decidable.

Coś o tym będzie na wykładzie.

Algorithmic questions

Problem: MSO model-checking
Input: MSO formula ϕ, HORS S
Output: does ϕ hold in the tree generated by S?

Theorem: MSO model-checking is decidable.

Complexity:

- nonelementary when $\phi \in$ MSO

Coś o tym będzie na wykładzie.

Algorithmic questions

Problem: MSO model-checking
Input: MSO formula ϕ, HORS S
Output: does ϕ hold in the tree generated by S?

Theorem: MSO model-checking is decidable.

Complexity:

- nonelementary when $\phi \in$ MSO
- n-EXPTIME-complete when ϕ is given as a μ-calculus formula or a parity automaton, and the scheme is of order n

Coś o tym będzie na wykładzie.

Algorithmic questions

Problem: MSO model-checking
Input: MSO formula ϕ, HORS S
Output: does ϕ hold in the tree generated by S?

Theorem: MSO model-checking is decidable.

Complexity:

- nonelementary when $\phi \in$ MSO
- n-EXPTIME-complete when ϕ is given as a μ-calculus formula or a parity automaton, and the scheme is of order n
- $(n\text{-}1)$-EXPTIME-complete for reachability properties (is "a" present in the tree)

Coś o tym będzie na wykładzie.

Algorithmic questions

Problem: MSO model-checking
Input: MSO formula ϕ, HORS S
Output: does ϕ hold in the tree generated by S?

Theorem: MSO model-checking is decidable.

Complexity:

- nonelementary when $\phi \in$ MSO
- n-EXPTIME-complete when ϕ is given as a μ-calculus formula or a parity automaton, and the scheme is of order n
- (n-1)-EXPTIME-complete for reachability properties (is "a" present in the tree)
- polynomial when n, ϕ, and maximal arity of a nonterminal fixed

Coś o tym będzie na wykładzie.

Algorithmic questions

Problem: MSO model-checking
Input: MSO formula ϕ, HORS S
Output: does ϕ hold in the tree generated by S?

Theorem: MSO model-checking is decidable.

Complexity:

- nonelementary when $\phi \in$ MSO
- n-EXPTIME-complete when ϕ is given as a μ-calculus formula or a parity automaton, and the scheme is of order n
- (n-1)-EXPTIME-complete for reachability properties (is "a" present in the tree)
- polynomial when n, ϕ, and maximal arity of a nonterminal fixed
- despite high complexity, solvable in practice

Coś o tym będzie na wykładzie.

Algorithmic questions

Trochę silniejsze wyniki:

MSO reflection

Input: MSO formula $\phi(x)$, HORS S (x to wierzchołek)

Output: HORS S' generating the same tree as S, where the nodes x in which $\phi(x)$ holds are marked.

MSO selection

Input: MSO formula $\phi(X)$, HORS S (X to zbiór wierzchołków)

Output: HORS S' generating the same tree as S with some nodes marked so that $\phi(X)$ holds for the set X of marked nodes.

Model-cheking (of programs with higher-order recursion)

Model-checking (weryfikacja modelowa) = automatyczne sprawdzanie, czy program spełnia zadaną specyfikację

NIE polega na uruchamianiu / testowaniu.

Polega na sprawdzeniu wszystkich możliwych przebiegów programu i "udowodnieniu", że specyfikacja jest spełniona – na podstawie kodu źródłowego.

Model-cheking (of programs with higher-order recursion)

Model-checking (weryfikacja modelowa) = automatyczne sprawdzanie, czy program spełnia zadaną specyfikację

NIE polega na uruchamianiu / testowaniu.

Polega na sprawdzeniu wszystkich możliwych przebiegów programu i "udowodnieniu", że specyfikacja jest spełniona – na podstawie kodu źródłowego.

Teoretycznie problem nierozstrzygalny (problem stopu dla maszyny Turinga), w praktyce jakoś działa.

Stosowane z powodzeniem przy tworzeniu:
- urządzeń sprzętowych
- sterowników (Static Driver Verifier w Windowsie)

Sprawdzane własności:
- brak zakleszczeń, procedura zawsze się kończy
- funkcje systemowe są prawidłowo wywoływane
- asercje są zawsze prawdziwe
- ...

Model-cheking of programs with higher-order recursion

Model-checking dla programów z rekurencją wyższego rzędu.

Obecnie niezbyt rozwinięty,
aczkolwiek istnieją narzędzia umożliwiające zweryfikowanie prostych procedur

Bardzo ogólna zasada działania:

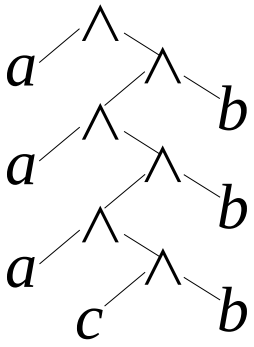
Tłumaczymy program na HORS, pomijając różne szczegóły
[główna trudność – jak to dobrze zrobić]

program błędny ⇒ błąd widoczny w HORS
błąd w HORS ⇒ błąd w programie, lub niedokładność tłumaczenia

Potem sprawdzamy, czy HORS spełnia zadaną własność (w pełni rozstrzygalne)

Coś o tym będzie na wykładzie.

Between trees and words

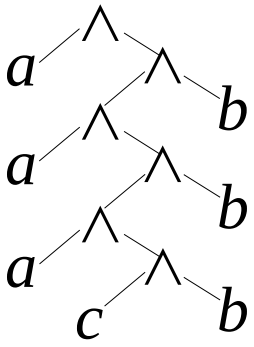


aaacbbb

regularny język drzew → język w liściach - bezkontekstowy

istnieje regularny język drzew taki, że w liściach jest L ← język L bezkontekstowy

Between trees and words



$aaacbbb$

regularny język drzew → język w liściach - bezkontekstowy

istnieje regularny język drzew taki, że w liściach jest L ← język L bezkontekstowy

język drzew rzędu n → język w liściach – rzędu $n+1$

istnieje język drzew rzędu n taki, że w liściach jest L ← język L rzędu $n+1$

Będzie na wykładzie.

Collapsible pushdown automata

- Every stack symbol has an identifier.
- $push_1 x$ pushes symbol x with a fresh identifier.
- $push_k$ for $k \geq 2$ copy symbols with their identifiers.

Collapsible pushdown automata

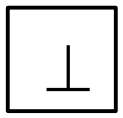
- Every stack symbol has an identifier.
- $push_1 x$ pushes symbol x with a fresh identifier.
- $push_k$ for $k \geq 2$ copy symbols with their identifiers.
- New operation $collapse_k$:

  remove from the topmost order-k stack all order-(k-1) stacks containing a copy of the topmost stack symbol.

Collapsible pushdown automata

- Every stack symbol has an identifier.
- $push_1 x$ pushes symbol x with a fresh identifier.
- $push_k$ for $k \geq 2$ copy symbols with their identifiers.
- New operation $collapse_k$:

  remove from the topmost order-k stack all order-(k-1) stacks containing a copy of the topmost stack symbol.

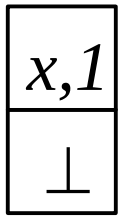


Collapsible pushdown automata

- Every stack symbol has an identifier.
- $push_1 x$ pushes symbol x with a fresh identifier.
- $push_k$ for $k \geq 2$ copy symbols with their identifiers.
- New operation $collapse_k$:

  remove from the topmost order-k stack all order-(k-1) stacks containing a copy of the topmost stack symbol.

$$push_1 x$$



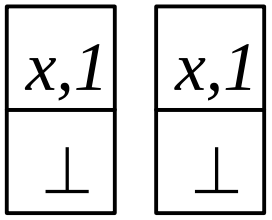
Collapsible pushdown automata

- Every stack symbol has an identifier.
- $push_1 x$ pushes symbol x with a fresh identifier.
- $push_k$ for $k \geq 2$ copy symbols with their identifiers.
- New operation $collapse_k$:

  remove from the topmost order-k stack all order-(k-1) stacks containing a copy of the topmost stack symbol.

$push_1 x$

$push_2$

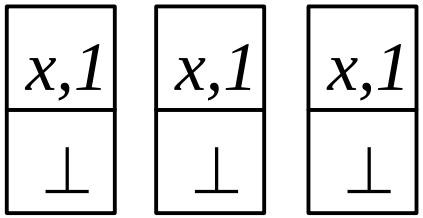


Collapsible pushdown automata

- Every stack symbol has an identifier.
- $push_1 x$ pushes symbol x with a fresh identifier.
- $push_k$ for $k \geq 2$ copy symbols with their identifiers.
- New operation $collapse_k$:

  remove from the topmost order-k stack all order-(k-1) stacks containing a copy of the topmost stack symbol.

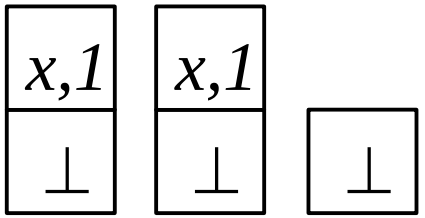
$$push_1 x$$
$$push_2$$
$$push_2$$



Collapsible pushdown automata

- Every stack symbol has an identifier.
- $push_1 x$ pushes symbol x with a fresh identifier.
- $push_k$ for $k \geq 2$ copy symbols with their identifiers.
- New operation $collapse_k$:
  remove from the topmost order-k stack all order-(k-1) stacks containing a copy of the topmost stack symbol.

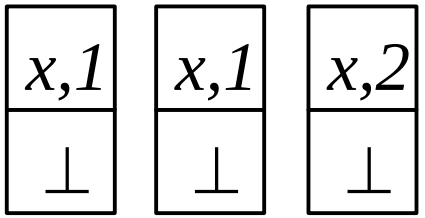
$push_1 x$
$push_2$
$push_2$
pop_1



Collapsible pushdown automata

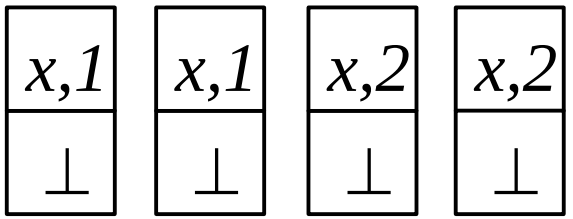
- Every stack symbol has an identifier.
- $push_1 x$ pushes symbol x with a fresh identifier.
- $push_k$ for $k \geq 2$ copy symbols with their identifiers.
- New operation $collapse_k$:
  remove from the topmost order-k stack all order-(k-1) stacks containing a copy of the topmost stack symbol.

$push_1 x$
$push_2$
$push_2$
pop_1
$push_1 x$



Collapsible pushdown automata

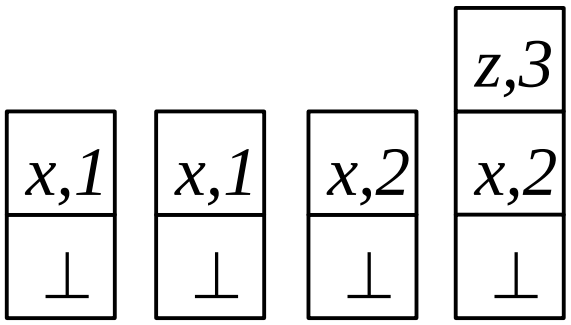
- Every stack symbol has an identifier.
- $push_1 x$ pushes symbol x with a fresh identifier.
- $push_k$ for $k \geq 2$ copy symbols with their identifiers.
- New operation $collapse_k$:
  remove from the topmost order-k stack all order-(k-1) stacks containing a copy of the topmost stack symbol.



$push_1 x$
$push_2$
$push_2$
pop_1
$push_1 x$
$push_2$

Collapsible pushdown automata

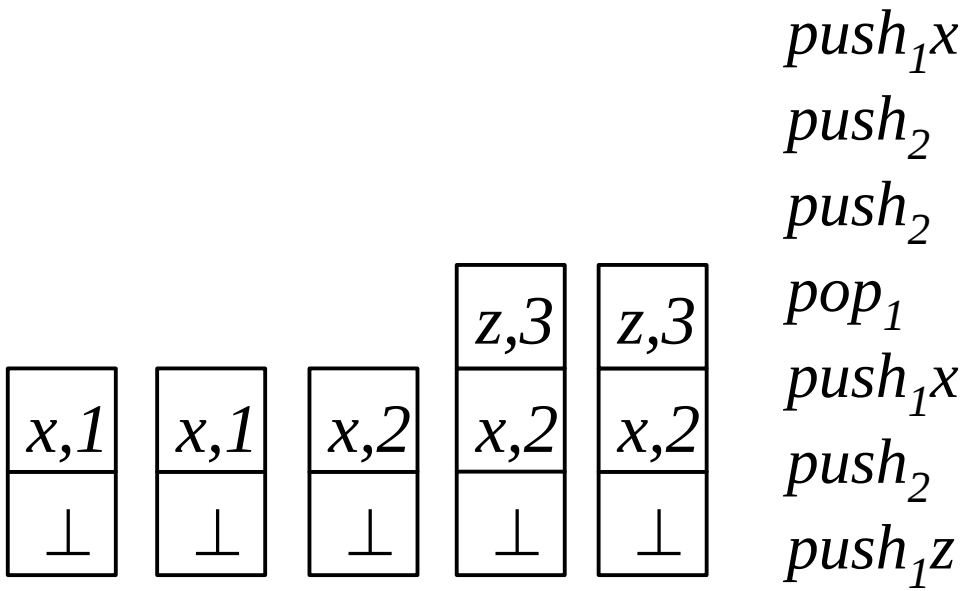
- Every stack symbol has an identifier.
- $push_1 x$ pushes symbol x with a fresh identifier.
- $push_k$ for $k \geq 2$ copy symbols with their identifiers.
- New operation $collapse_k$:
  remove from the topmost order-k stack all order-(k-1) stacks containing a copy of the topmost stack symbol.



$push_1 x$
$push_2$
$push_2$
pop_1
$push_1 x$
$push_2$
$push_1 z$

Collapsible pushdown automata

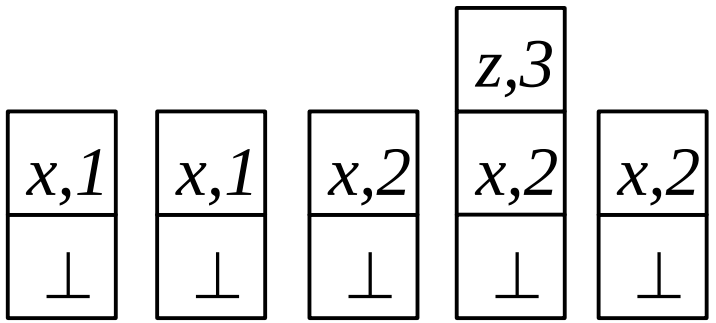
- Every stack symbol has an identifier.
- $push_1 x$ pushes symbol x with a fresh identifier.
- $push_k$ for $k \geq 2$ copy symbols with their identifiers.
- New operation $collapse_k$:
  remove from the topmost order-k stack all order-(k-1) stacks containing a copy of the topmost stack symbol.



$push_1 x$
$push_2$
$push_2$
pop_1
$push_1 x$
$push_2$
$push_1 z$
$push_2$

Collapsible pushdown automata

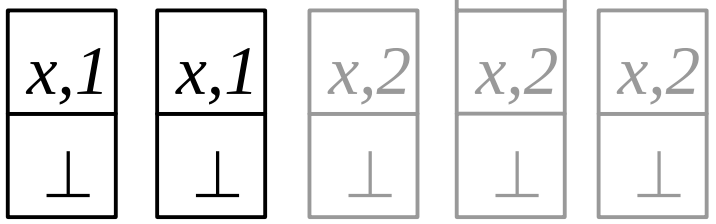
- Every stack symbol has an identifier.
- $push_1 x$ pushes symbol x with a fresh identifier.
- $push_k$ for $k \geq 2$ copy symbols with their identifiers.
- New operation $collapse_k$:
  remove from the topmost order-k stack all order-(k-1) stacks containing a copy of the topmost stack symbol.



$push_1 x$
$push_2$
$push_2$
pop_1
$push_1 x$
$push_2$
$push_1 z$
$push_2$
pop_1

Collapsible pushdown automata

- Every stack symbol has an identifier.
- $push_1 x$ pushes symbol x with a fresh identifier.
- $push_k$ for $k \geq 2$ copy symbols with their identifiers.
- New operation $collapse_k$:
  remove from the topmost order-k stack all order-(k-1) stacks containing a copy of the topmost stack symbol.



$push_1 x$
$push_2$
$push_2$
pop_1
$push_1 x$
$push_2$
$push_1 z$
$push_2$
pop_1
$collapse_2$