Introduction to Collapsible Pushdown Automata and Higher-Order Recursion Schemes

Paweł Parys

University of Warsaw

Workshop on Higher-Order Model Checking, Singapore, 20.09.2016

Higher-order pushdown automata [Maslov 1974] - definition

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

Higher-order pushdown automata [Maslov 1974] - 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: s_i are 1-stacks. Top of stack is on right.

$$push_2 : [s_1 \ldots s_{i-1} s_i] \rightarrow [s_1 \ldots s_{i-1} s_i s_i]$$

$$pop_2 : [s_1 \ldots s_{i-1} s_i] \rightarrow [s_1 \ldots s_{i-1}]$$

Higher-order pushdown automata [Maslov 1974] - 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: s_i are *1*-stacks. Top of stack is on right.

$$push_2 : [s_1 \dots s_{i-1} s_i] \rightarrow [s_1 \dots s_{i-1} s_i s_i]$$

$$pop_2 : [s_1 \dots s_{i-1} s_i] \rightarrow [s_1 \dots s_{i-1}]$$

$$push_1 x : [s_1 \dots s_{i-1} [a_1 \dots a_{j-1} a_j]] \rightarrow [s_1 \dots s_{i-1} [a_1 \dots a_{j-1} a_j x]]$$

$$pop_1 : [s_1 \dots s_{i-1} [a_1 \dots a_{j-1} a_j]] \rightarrow [s_1 \dots s_{i-1} [a_1 \dots a_{j-1}]]$$

Higher-order pushdown automata [Maslov 1974] - 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: s_i are *1*-stacks. Top of stack is on right.

$$push_2 : [s_1 \ldots s_{i-1} s_i] \rightarrow [s_1 \ldots s_{i-1} s_i s_i]$$
$$pop_2 : [s_1 \ldots s_{i-1} s_i] \rightarrow [s_1 \ldots s_{i-1}]$$

$$push_1 x : [s_1 \ldots s_{i-1} [a_1 \ldots a_{j-1} a_j]] \rightarrow [s_1 \ldots s_{i-1} [a_1 \ldots a_{j-1} a_j x]]$$
$$pop_1 : [s_1 \ldots s_{i-1} [a_1 \ldots a_{j-1} a_j]] \rightarrow [s_1 \ldots s_{i-1} [a_1 \ldots a_{j-1}]]$$

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

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: ⊥, x, #

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

any stack symbol

q_1 ⊥

Higher-order pushdown automata - example

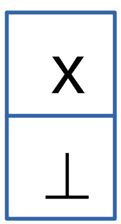
Language: $\{b^{2^k} : k \in \mathbb{N}\}$

- order 2
- 3 stack symbols: ⊥, x, #

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

any stack symbol

q_1



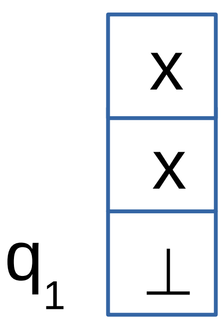
Higher-order pushdown automata - example

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

- order 2
- 3 stack symbols: ⊥, x, #

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

any stack symbol



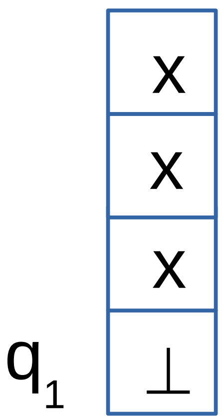
Higher-order pushdown automata - example

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

- order 2
- 3 stack symbols: ⊥, x, #

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

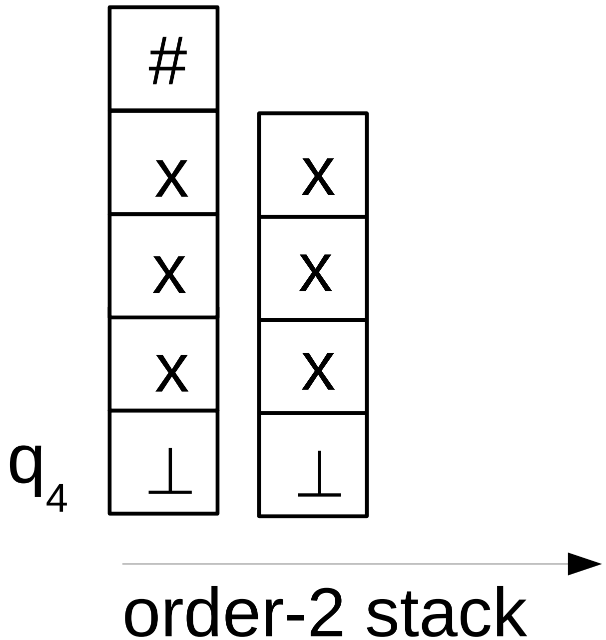
any stack symbol



Higher-order pushdown automata - example

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

- order 2
- 3 stack symbols: ⊥, x, #



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

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

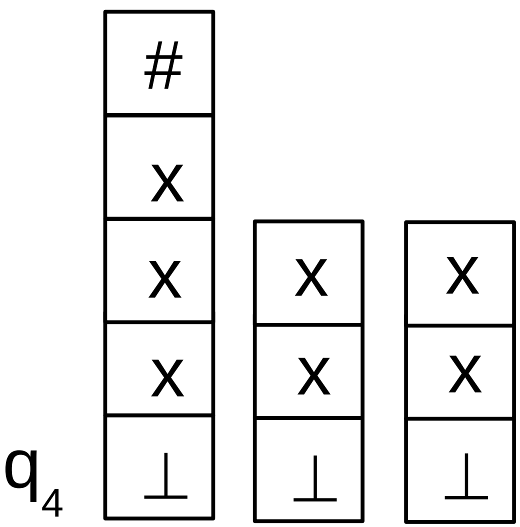
$$(\#,q_2) \xrightarrow{\varepsilon} (q_3,push_2)$$

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

Higher-order pushdown automata - example

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

- order 2
- 3 stack symbols: ⊥, 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: ⊥, x, #

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

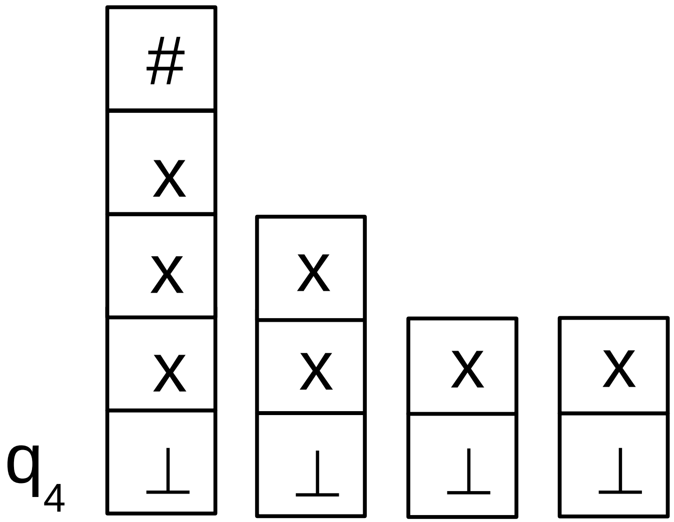
$$(\_,q_1) \xrightarrow{\varepsilon} (q_2,push_1(\#))$$

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

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

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

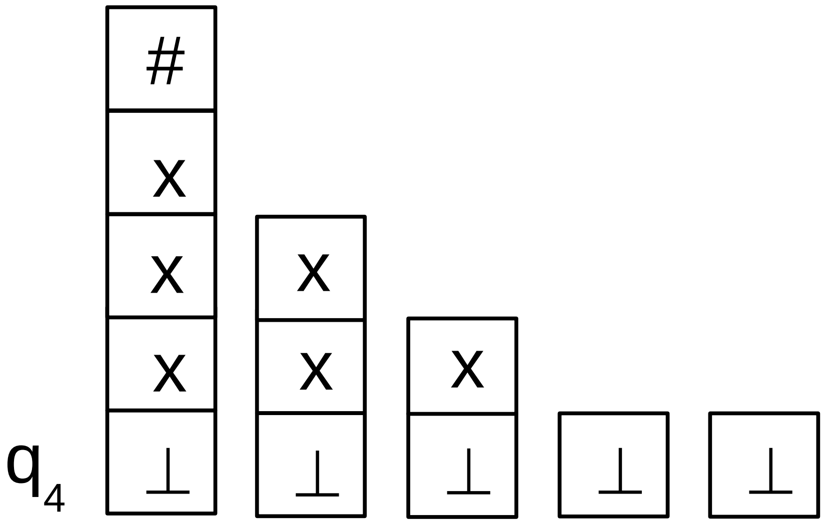
$$(\_,q_5) \xrightarrow{\varepsilon} (q_4,push_2)$$



Higher-order pushdown automata - example

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

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



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

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

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

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

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

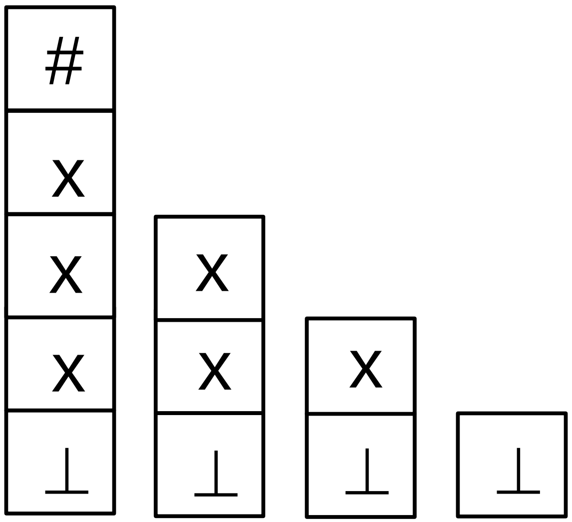
$(\_, q_5) \xrightarrow{\varepsilon} (q_4, push_2)$

Higher-order pushdown automata - example

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

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

q_4



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

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

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

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

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

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

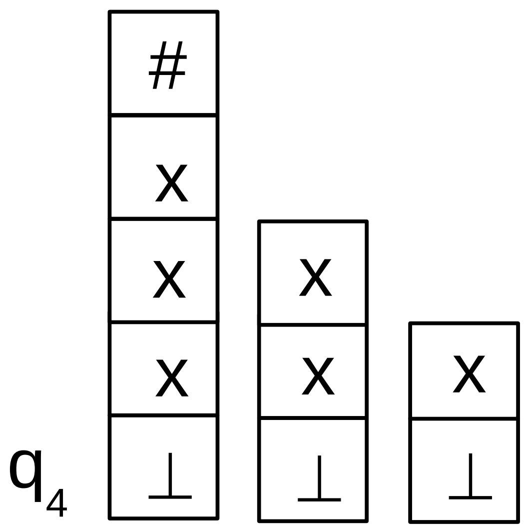
$(\bot, q_4) \xrightarrow{b} (q_4, pop_2)$

Input: b

Higher-order pushdown automata - example

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

- order 2
- 3 stack symbols: ⊥, x, #



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

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

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

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

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

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

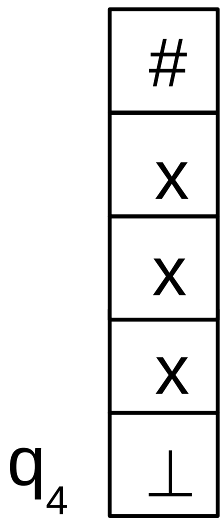
$$(\bot,q_4) \xrightarrow{b} (q_4,pop_2)$$

Input: b b

Higher-order pushdown automata - example

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

- order 2
- 3 stack symbols: ⊥, 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

Higher-order pushdown automata

“Traditional” view:

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

“Modern” view:

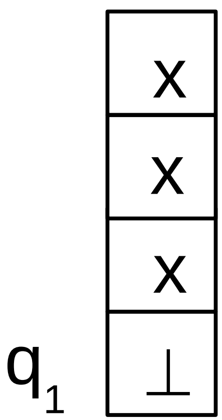
- a deterministic HOPDA generating a single tree (node-labeled, ranked, ordered, usually infinite)

One can also consider configuration graphs of HOPDA – not in this talk.

Higher-order pushdown automata - example

nondeterminism – what to do next?

- order 2
- 3 stack symbols: ⊥, x, #



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

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

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

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

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

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

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

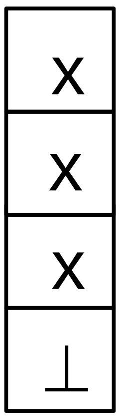
$(\#,q_4) \xrightarrow{\varepsilon} (q_{acc},id)$

Higher-order pushdown automata - example

letter a of rank 2

- order 2
- 3 stack symbols: ⊥, x, #

q_1



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

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

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

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

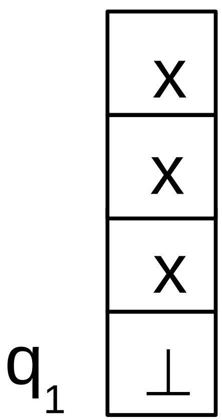
$(\_, q_5) \xrightarrow{\varepsilon} (q_4, push_2)$

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

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

Higher-order pushdown automata - example

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



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

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

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

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

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

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

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

letter c of rank 0, instead of an accepting state

Higher-order pushdown automata - example

- order 2
- 3 stack symbols: $\perp$, 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.

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

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

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

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

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

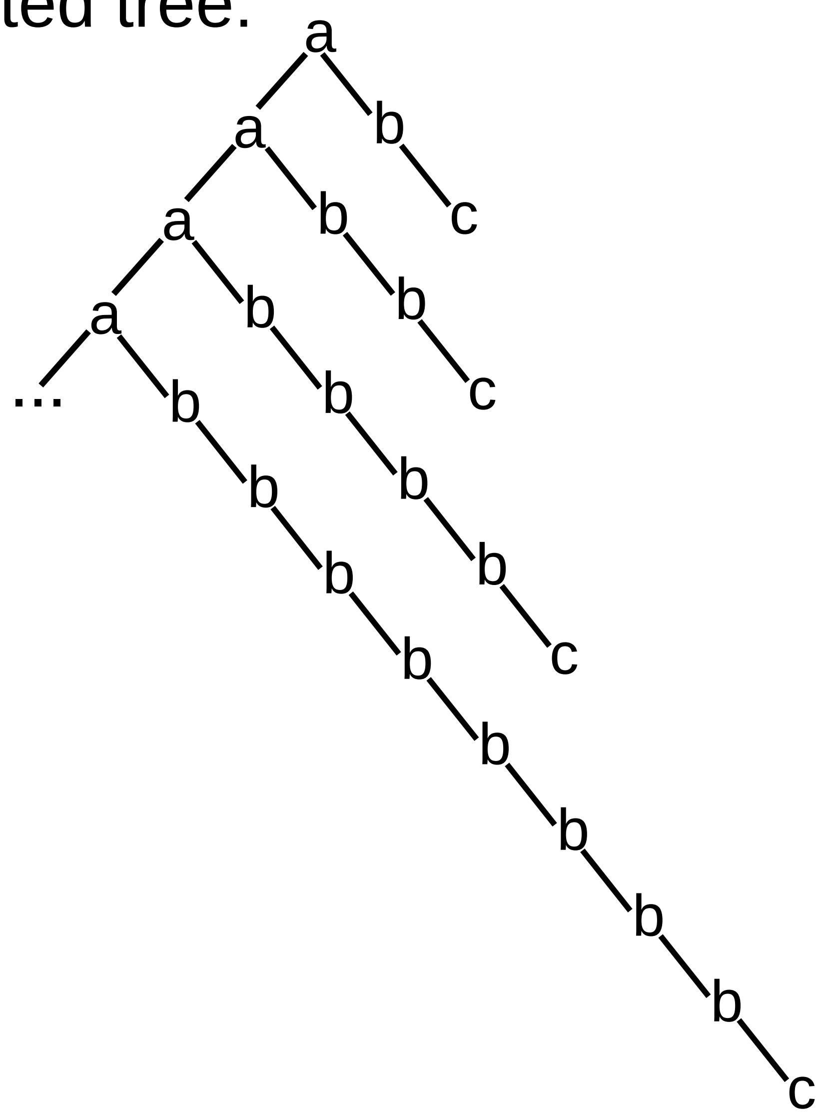
$(\perp, q_4) \xrightarrow{b} (q_4, pop_2)$

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

Higher-order pushdown automata - example

- order 2
- 3 stack symbols: ⊥, x, #

Generated tree:



$(\_,q_1)$ — a → $(q_1,push_1(x))$ and $(q_2,push_1(\#))$

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

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

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

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

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

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

Higher-order recursion schemes

pushdown automata —generalization→ higher-order pushdown automata

context-free grammars —generalization→ higher-order recursion schemes

Higher-order recursion schemes - definition

Nonterminals may take arguments, that can be then used on the right side of productions.

Higher-order recursion schemes - definition

Nonterminals may take arguments, that can be then used on the right side of productions.

Every nonterminal (every argument) has assigned some type.

Types:

$$\alpha ::= o \mid \alpha \to \beta$$

- o – type of a tree
- $o \to o$ – type of a function that takes a tree, and produces a tree
- $o \to (o \to o) \to o$ – type of a function that takes a tree and a function of type $o \to o$, and produces a tree

abbreviation of $o \to ((o \to o) \to o)$

Higher-order recursion schemes - definition

Nonterminals may take arguments, that can be then used on the right side of productions.

Every nonterminal (every argument) has assigned some type.

Types:

$$\alpha ::= o \mid \alpha \to \beta$$

Order:

$$\mathrm{ord}(o) = 0$$

$$\mathrm{ord}(\alpha_1 \to \ldots \to \alpha_k \to o) = 1 + \max(\mathrm{ord}(\alpha_1), \ldots, \mathrm{ord}(\alpha_k))$$

- $\mathrm{ord}(o) = 0$,
- $\mathrm{ord}(o \to o) = \mathrm{ord}(o \to o \to o) = 1$,
- $\mathrm{ord}(o \to (o \to o) \to o) = 2$

Higher-order recursion schemes – example

Ranked alphabet:

$a^{o\to o\to o}$ of rank 2, $b^{o\to o}$ of rank 1, c^{o} of rank 0

Nonterminals:

S^{o} (starting), $A^{(o\to o)\to o}$, $D^{(o\to o)\to o\to o}$

Higher-order recursion schemes – example (of order 2)

Ranked alphabet:

$a^{o\to o\to o}$ of rank 2, $b^{o\to o}$ of rank 1, c^{o} of rank 0

Nonterminals:

S^{o} (starting), $A^{(o\to o)\to o}$, $D^{(o\to o)\to o\to o}$

order 0 order 2 order 2

Order of a HORS = maximal order of (a type of) its nonterminal

Higher-order recursion schemes – example (of order 2)

Ranked alphabet:

$a^{o\to o\to o}$ of rank 2, $b^{o\to o}$ of rank 1, c^{o} of rank 0

Nonterminals:

S^{o} (starting), $A^{(o\to o)\to o}$, $D^{(o\to o)\to o\to o}$

Rules:

$S \to A\ b$

$A\ f \to a\ (A\ (D\ f))\ (f\ c)$

$D\ f\ x \to f\ (f\ x)$

It is required that:

1) types are respected

e.g. D of type $(o\to o)\to o\to o$ is applied to f of type $o\to o$,

A of type $(o\to o)\to o$ is applied to $D\ f$ of type $o\to o$, etc.

2) right side of every rule is of type o

Higher-order recursion schemes – example (of order 2)

Ranked alphabet:

$a^{o\to o\to o}$ of rank 2, $b^{o\to o}$ of rank 1, c^{o} of rank 0

Nonterminals:

S^{o} (starting), $A^{(o\to o)\to o}$, $D^{(o\to o)\to o\to o}$

Rules:

$S \to A\ b$

$A\ f \to a\ (A\ (D\ f))\ (f\ c)$

$D\ f\ x \to f\ (f\ x)$

$S \to A\ b \to a\ (A\ (D\ b))\ (b\ c)$

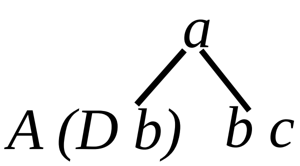
Higher-order recursion schemes – example (of order 2)

Ranked alphabet:

$a^{o\to o\to o}$ of rank 2, $b^{o\to o}$ of rank 1, c^{o} of rank 0

Nonterminals:

S^{o} (starting), $A^{(o\to o)\to o}$, $D^{(o\to o)\to o\to o}$



Rules:

$$S \to A\ b$$
$$A\ f \to a\ (A\ (D\ f))\ (f\ c)$$
$$D\ f\ x \to f\ (f\ x)$$

$$S \to A\ b \to a\ (A\ (D\ b))\ (b\ c)$$

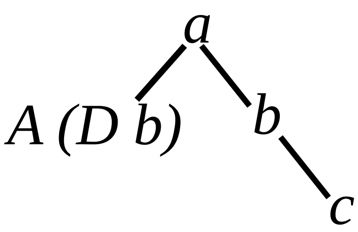
Higher-order recursion schemes – example (of order 2)

Ranked alphabet:

$a^{o \to o \to o}$ of rank 2, $b^{o \to o}$ of rank 1, c^{o} of rank 0

Nonterminals:

S^{o} (starting), $A^{(o \to o) \to o}$, $D^{(o \to o) \to o \to o}$



Rules:

$S \to A\ b$

$A\ f \to a\ (A\ (D\ f))\ (f\ c)$

$D\ f\ x \to f\ (f\ x)$

$S \to A\ b \to a\ (A\ (D\ b))\ (b\ c)$

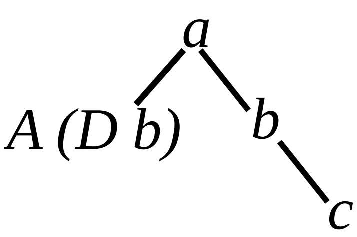
Higher-order recursion schemes – example (of order 2)

Ranked alphabet:

$a^{o\to o\to o}$ of rank 2, $b^{o\to o}$ of rank 1, c^{o} of rank 0

Nonterminals:

S^{o} (starting), $A^{(o\to o)\to o}$, $D^{(o\to o)\to o\to o}$



Rules:

$S \quad\;\; \to A\ b$

$A\ f \quad \to a\ (A\ (D\ f))\ (f\ c)$

$D\ f\ x \to f\ (f\ x)$

$S \to A\ b \to a\ (A\ (D\ b))\ (b\ c)$

$A\ (D\ b) \to a\ (A\ (D\ (D\ b)))\ (D\ b\ c)$

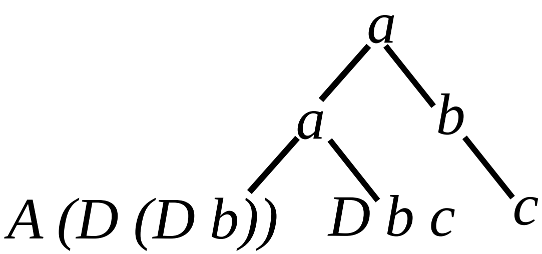
Higher-order recursion schemes – example (of order 2)

Ranked alphabet:

$a^{o\to o\to o}$ of rank 2, $b^{o\to o}$ of rank 1, c^{o} of rank 0

Nonterminals:

S^{o} (starting), $A^{(o\to o)\to o}$, $D^{(o\to o)\to o\to o}$



Rules:

$S \quad \to A\; b$

$A\; f \quad \to a\; (A\; (D\; f))\; (f\; c)$

$D\; f\; x \to f\; (f\; x)$

$S \to A\; b \to a\; (A\; (D\; b))\; (b\; c)$

$A\; (D\; b) \to a\; (A\; (D\; (D\; b)))\; (D\; b\; c)$

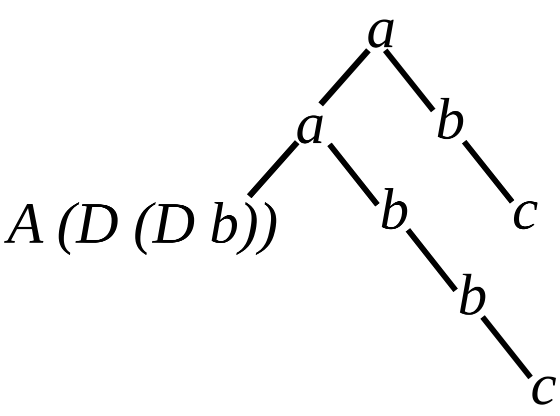
Higher-order recursion schemes – example (of order 2)

Ranked alphabet:

$a^{o \to o \to o}$ of rank 2, $b^{o \to o}$ of rank 1, c^{o} of rank 0

Nonterminals:

S^{o} (starting), $A^{(o \to o) \to o}$, $D^{(o \to o) \to o \to o}$



Rules:

$S \quad\quad \to A\ b$

$A\ f \quad \to a\ (A\ (D\ f))\ (f\ c)$

$D\ f\ x \to f\ (f\ x)$

$S \to A\ b \to a\ (A\ (D\ b))\ (b\ c)$

$A\ (D\ b) \to a\ (A\ (D\ (D\ b)))\ (D\ b\ c)$

$D\ b\ c \to b\ (b\ c)$

Higher-order recursion schemes – example (of order 2)

Ranked alphabet:

$a^{o\to o\to o}$ of rank 2, $b^{o\to o}$ of rank 1, c^{o} of rank 0

Nonterminals:

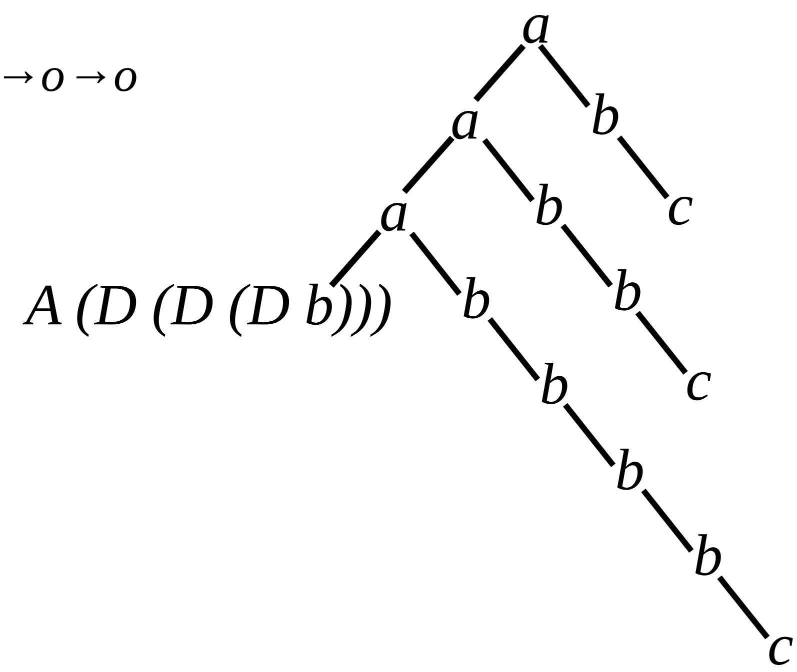
S^{o} (starting), $A^{(o\to o)\to o}$, $D^{(o\to o)\to o\to o}$

Rules:

$S \to A\ b$

$A\ f \to a\ (A\ (D\ f))\ (f\ c)$

$D\ f\ x \to f\ (f\ x)$



$S \to A\ b \to a\ (A\ (D\ b))\ (b\ c)$

$A\ (D\ b) \to a\ (A\ (D\ (D\ b)))\ (D\ b\ c)$

$D\ b\ c \to b\ (b\ c)$

$A\ (D\ (D\ b)) \to a\ (A\ (D\ (D\ (D\ b))))\ (D\ (D\ b)\ c)$

$D\ (D\ b)\ c \to D\ b\ (D\ b\ c) \to b\ (b\ (D\ b\ c))$

Higher-order recursion schemes – example (of order 2)

Ranked alphabet:

$a^{o \to o \to o}$ of rank 2, $b^{o \to o}$ of rank 1, c^{o} of rank 0

Nonterminals:

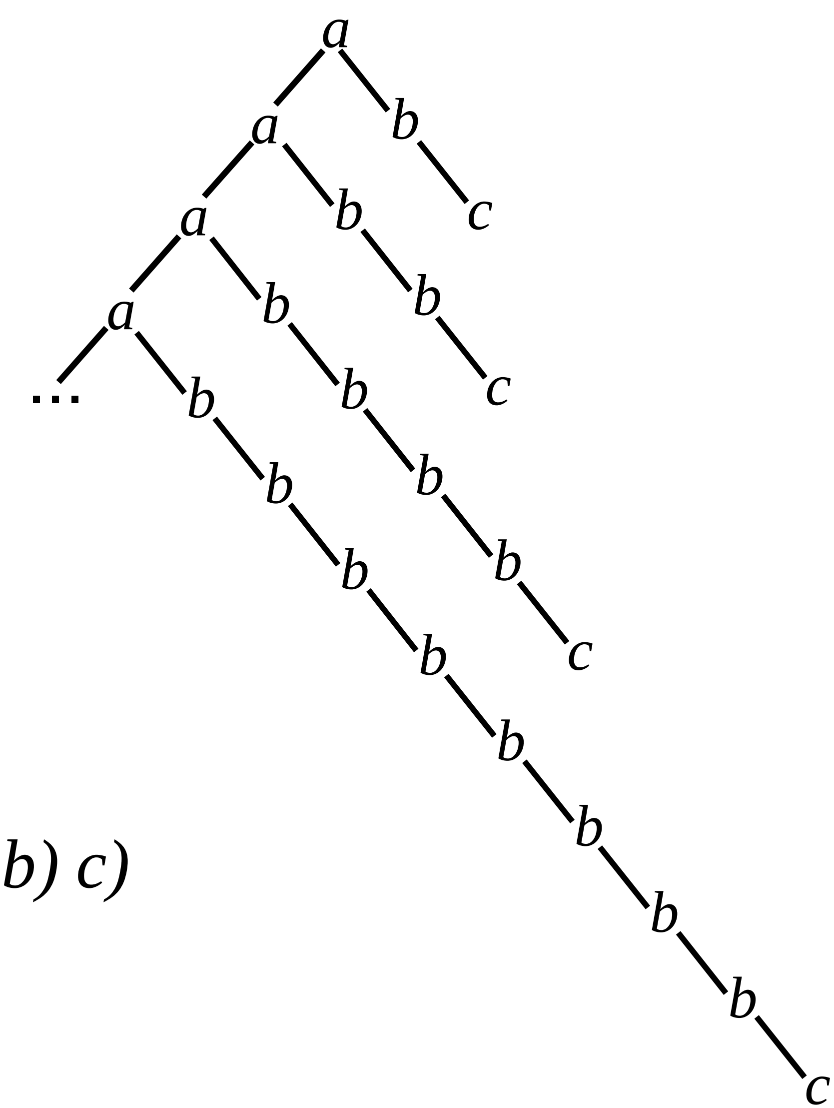
S^{o} (starting), $A^{(o \to o) \to o}$, $D^{(o \to o) \to o \to o}$

Rules:

$S \to A\ b$

$A\ f \to a\ (A\ (D\ f))\ (f\ c)$

$D\ f\ x \to f\ (f\ x)$



$S \to A\ b \to a\ (A\ (D\ b))\ (b\ c)$

$A\ (D\ b) \to a\ (A\ (D\ (D\ b)))\ (D\ b\ c)$

$D\ b\ c \to b\ (b\ c)$

$A\ (D\ (D\ b)) \to a\ (A\ (D\ (D\ (D\ b))))\ (D\ (D\ b)\ c)$

$D\ (D\ b)\ c \to D\ b\ (D\ b\ c) \to b\ (b\ (D\ b\ c))$

Higher-order recursion schemes

- Previous slides: a deterministic HORS generating a single tree.

Higher-order recursion schemes

- Previous slides: a deterministic HORS generating a single tree.
- One can also consider a nondeterministic HORS, recognizing a language of finite trees.

Higher-order recursion schemes

- Previous slides: a deterministic HORS generating a single tree.
- One can also consider a nondeterministic HORS, recognizing a language of finite trees.
- If every letter is of rank 1, except a single letter of rank 0, then these trees, consisting of a single branch, can be seen as words → the HORS recognizes a set of words.

Higher-order recursion schemes

- Previous slides: a deterministic HORS generating a single tree.
- One can also consider a nondeterministic HORS, recognizing a language of finite trees.
- If every letter is of rank 1, except a single letter of rank 0, then these trees, consisting of a single branch, can be seen as words → the HORS recognizes a set of words.

end of word marker

Example:

Alphabet: ~~a of rank 2~~, b of rank 1, c of rank 0

Nonterminals: S^{o} (starting), $A^{(o\to o)\to o}$, $D^{(o\to o)\to o\to o}$

Rules: $S \to A\, b$

~~$A\, f \to a\, (A\, (D\, f))\, (f\, c)$~~

$D\, f\, x \to f\, (f\, x)$

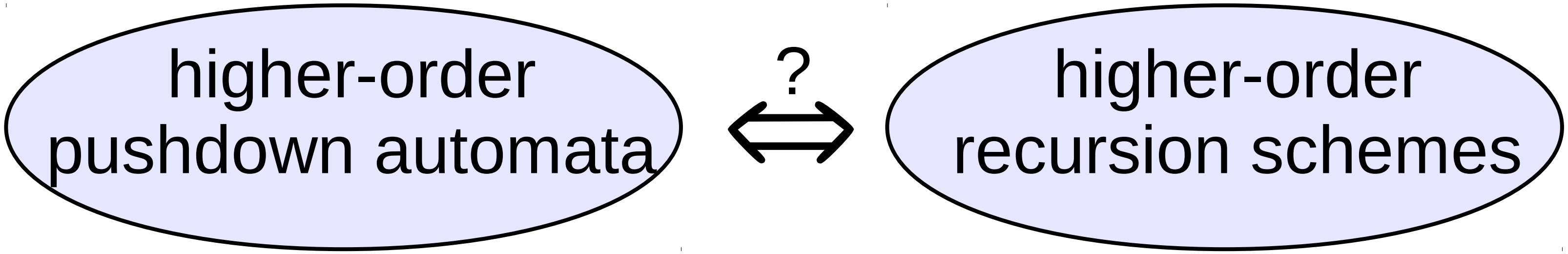
$A\, f \to A\, (D\, f)$

$A\, f \to f\, c$

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

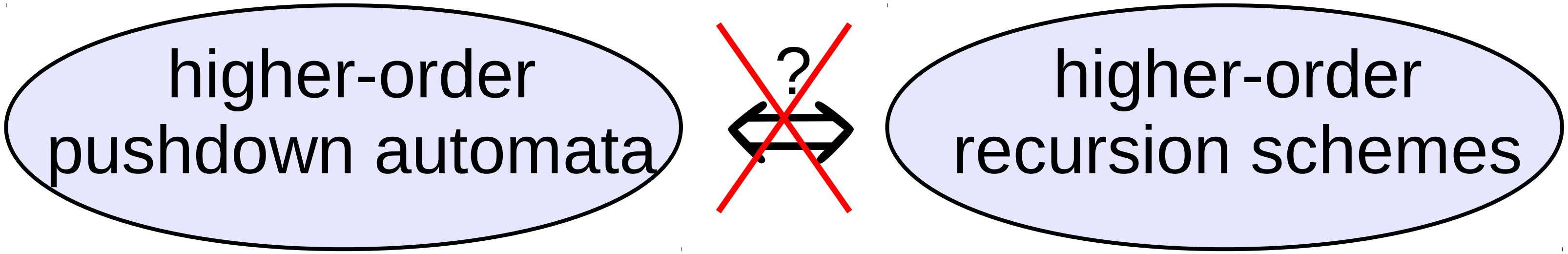
HOPDA vs HORS

Are these two formalisms equivalent?



HOPDA vs HORS

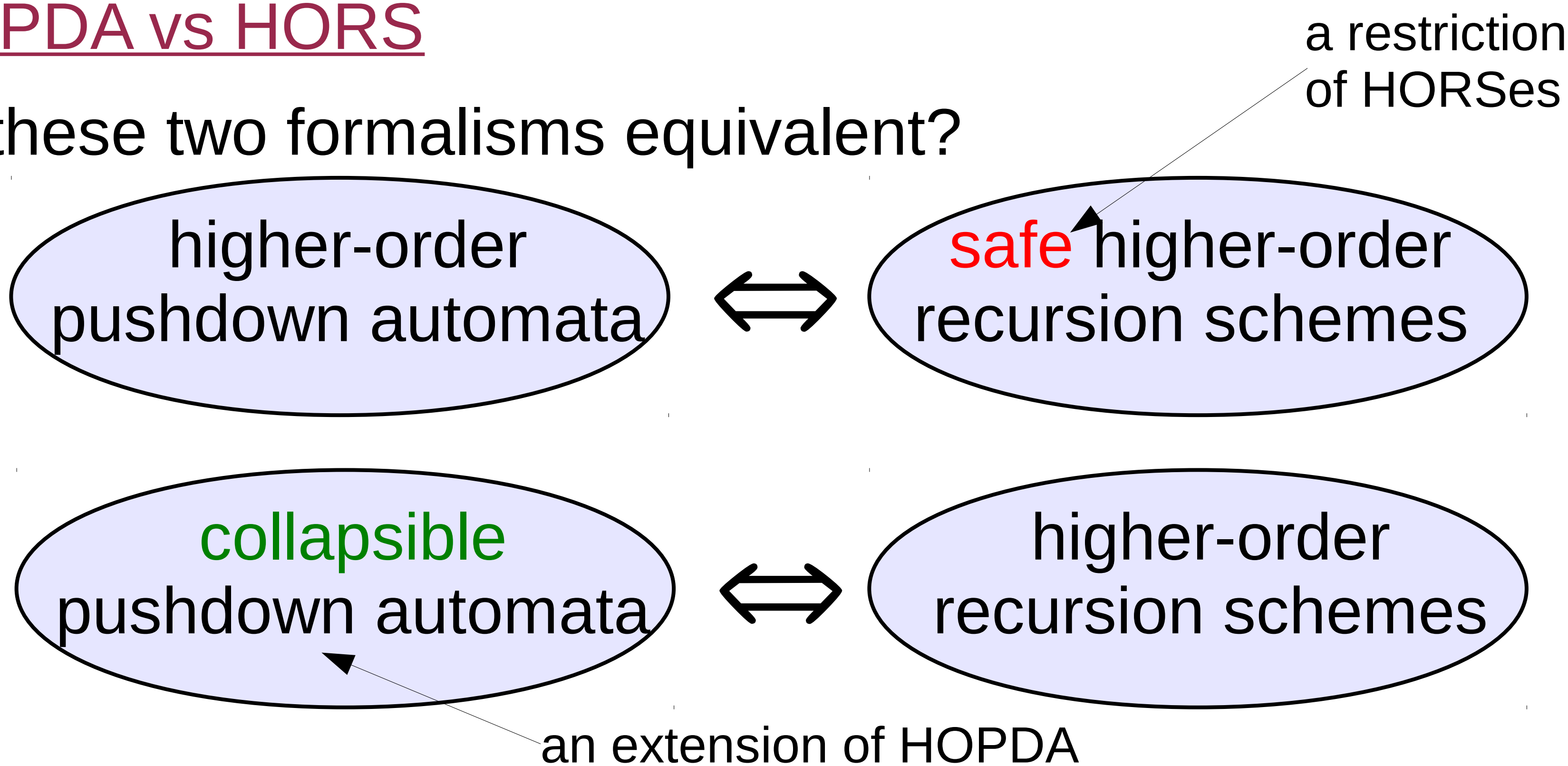
Are these two formalisms equivalent?



Not exactly!

HOPDA vs HORS

Are these two formalisms equivalent?



Theorem [Knapik, Niwiński, Urzyczyn 2002 & earlier results]
For every n, HOPDA of order n and safe HORSes of order n generate the same trees (recognize the same word languages);
[Caucal 2002] these are trees from the Caucal hierarchy, defined by iterating MSO interpretations and unfolding of graphs into trees.

Theorem [Hague, Murawski, Ong, Serre 2008]
For every n, collapsible HOPDA of order n and HORSes of order n generate the same trees (recognize the same word languages).

What is safety?

Restriction on terms appearing on right sides of rules:

- unrestricted terms:

$$M ::= a \mid x \mid A \mid M\, N$$

- safe terms:

$$M ::= a \mid x \mid A \mid M\, N_1 \dots N_k$$

only if $ord(M\, N_1 \dots N_k) \leq ord(N_i)$ for all i

In other words: if we apply an argument of some order k, then we have to apply also all arguments of order $\geq k$

What is safety?

Restriction on terms appearing on right sides of rules:

- unrestricted terms:

$$M ::= a \mid x \mid A \mid M\ N$$

- safe terms:

$$M ::= a \mid x \mid A \mid M\ N_1 \ldots N_k$$

only if $ord(M\ N_1 \ldots N_k) \leq ord(N_i)$ for all i

In other words: if we apply an argument of some order k, then we have to apply also all arguments of order $\geq k$

Let's check for our example HORS:

$$S \to A\ b$$
$$A\ f \to a\ (A\ (D\ f))\ (f\ c)$$
$$D\ f\ x \to f\ (f\ x)$$

What is safety?

Restriction on terms appearing on right sides of rules:

- unrestricted terms:

$$M ::= a \mid x \mid A \mid M\ N$$

- safe terms:

$$M ::= a \mid x \mid A \mid M\ N_1 \dots N_k$$

only if $ord(M\ N_1 \dots N_k) \leq ord(N_i)$ for all i

In other words: if we apply an argument of some order k, then we have to apply also all arguments of order $\geq k$

Let's check for our example HORS:

$$S \to A\ b$$
$$A\ f \to a\ (A\ (D\ f))\ (f\ c)$$
$$D\ f\ x \to f\ (f\ x)$$

$ord(D\ f) = 1 \leq 1 = ord(f)$ → OK

What is safety?

Restriction on terms appearing on right sides of rules:

- unrestricted terms:

$$M ::= a \mid x \mid A \mid M\, N$$

- safe terms:

$$M ::= a \mid x \mid A \mid M\, N_1 \ldots N_k$$

only if $ord(M\, N_1 \ldots N_k) \leq ord(N_i)$ for all i

In other words: if we apply an argument of some order k, then we have to apply also all arguments of order $\geq k$

Let's check for our example HORS:

$$S \to A\, b$$
$$A\, f \to a\, (A\, (D\, f))\, (f\, c)$$
$$D\, f\, x \to f\, (f\, x)$$

✓safe

$ord(D\, f) = 1 \leq 1 = ord(f)$ → OK

All other subterms are of order 0 → OK

What is safety?

Restriction on terms appearing on right sides of rules:

- unrestricted terms:

$$M ::= a \mid x \mid A \mid M\,N$$

- safe terms:

$$M ::= a \mid x \mid A \mid M\,N_1 \ldots N_k$$

only if $ord(M\,N_1 \ldots N_k) \le ord(N_i)$ for all i

In other words: if we apply an argument of some order k, then we have to apply also all arguments of order $\geq k$

Example: Unsafe HORS (generating "Urzyczyn's tree" U):

Types: $a^{o\to o\to o}$, $b^{o\to o}$, $c^{o\to o}$, d^{o}, e^{o}, S^{o}, $F^{(o\to o)\to o\to o\to o}$

Rules: $S \to F\ b\ d\ e$

$F\ f\ x\ y \to a\ (F\ (F\ f\ x)\ y\ (c\ y))\ (a\ (f\ y)\ x)$

✘ unsafe

(and not equivalent to any safe HORS)

$ord(F\ f\ x) = 1 > 0 = ord(x)$

(F expects two order-0 arguments; we have applied one (x), but not the other)

Why safety helps?

Theorem [Knapik, Niwiński, Urzyczyn 2002; Blum, Ong 2007]
Substitution (hence β-reduction) in safe λ-calculus can be implemented without renaming bound variables.

Bad example: when you substitute $(\lambda x.y\ x)\ [a\ x\ x\ /\ y]$, it is neccessary to change the first two x to some other variable name

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

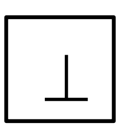
- Every stack symbol has an identifier.
- $push_1 x$ pushes symbol x with a fresh identifier.
- $push_k$ for $k \geq 2$ copy symbols with their identifiers.
- New operation $collapse_k$:

  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-$(k\text{-}1)$ stacks containing a copy of the topmost stack symbol.

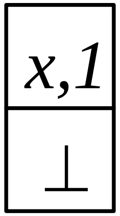


Collapsible pushdown automata

- 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.

$push_1 x$



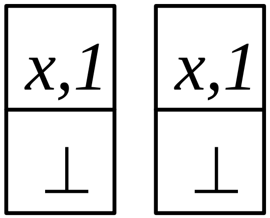
Collapsible pushdown automata

- 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-$(k\text{-}1)$ stacks containing a copy of the topmost stack symbol.

$push_1 x$

$push_2$



Collapsible pushdown automata

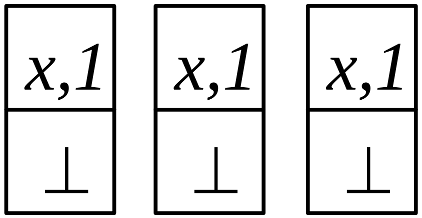
- Every stack symbol has an identifier.
- $push_1 x$ pushes symbol x with a fresh identifier.
- $push_k$ for $k \geq 2$ copy symbols with their identifiers.
- New operation $collapse_k$:

  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

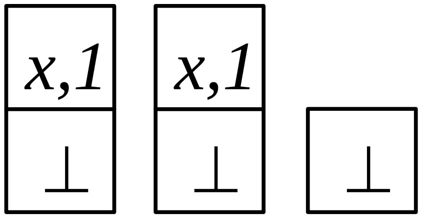
- Every stack symbol has an identifier.
- $push_1 x$ pushes symbol x with a fresh identifier.
- $push_k$ for $k \geq 2$ copy symbols with their identifiers.
- New operation $collapse_k$:
  remove from the topmost order-k stack all order-$(k-1)$ stacks containing a copy of the topmost stack symbol.

$push_1 x$

$push_2$

$push_2$

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.

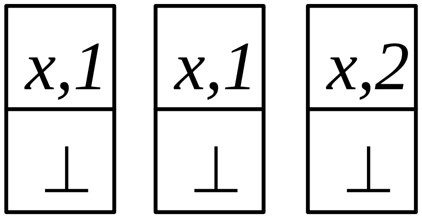
$push_1 x$

$push_2$

$push_2$

pop_1

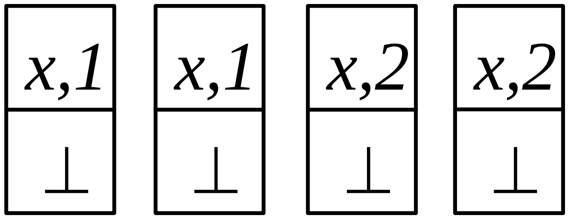
$push_1 x$



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.



$push_1 x$

$push_2$

$push_2$

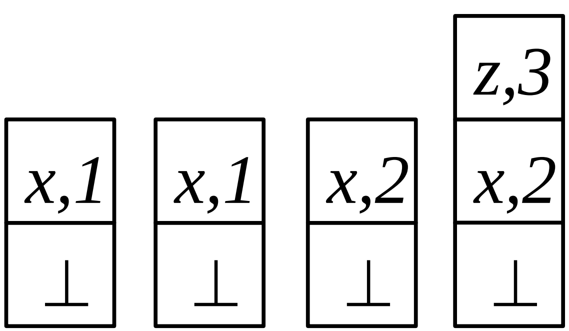
pop_1

$push_1 x$

$push_2$

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

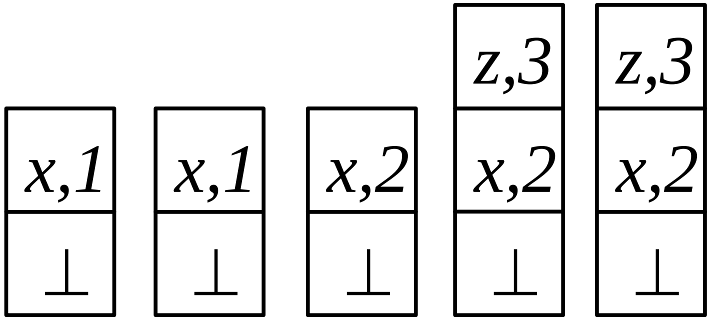
$push_1 x$

$push_2$

$push_1 z$

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.

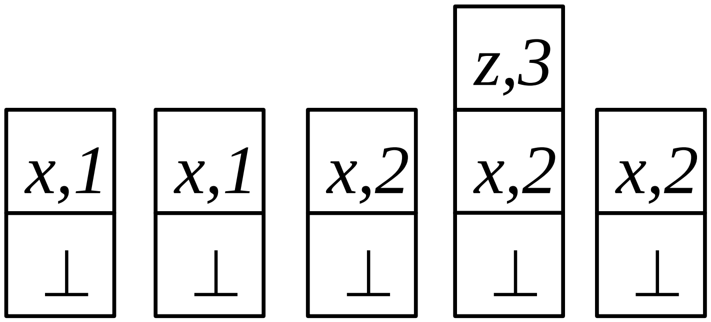


$push_1 x$
$push_2$
$push_2$
pop_1
$push_1 x$
$push_2$
$push_1 z$
$push_2$

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-$(k\text{-}1)$ stacks containing a copy of the topmost stack symbol.



$push_1 x$

$push_2$

$push_2$

pop_1

$push_1 x$

$push_2$

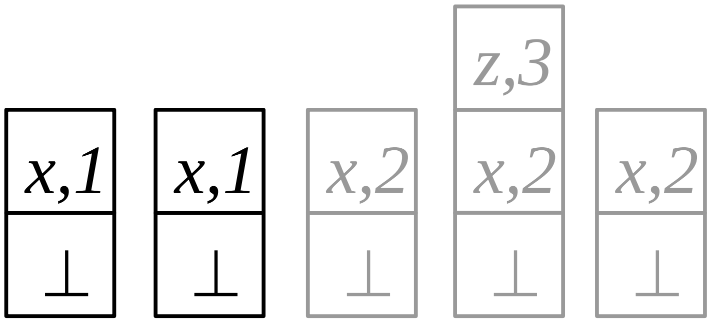
$push_1 z$

$push_2$

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
$collapse_2$

Collapsible pushdown automata

How collapse can be useful? – Urzyczyn's language U
(≈ branches in the Urzyczyn's tree)

alphabet: [,], *
U contains words of the form:

[[ ] [[ ] [[ ]] ****

A B C

- segment A forms a prefix of a well-bracketed word that ends in [not matched in the entire word
- segment B forms a well-bracketed word
- the number of stars in C equals the number of brackets in A

Collapsible pushdown automata

How collapse can be useful? – Urzyczyn's language U
(≈ branches in the Urzyczyn's tree)

Words in U:
A) a prefix of a well-bracketed word
B) a well-bracketed word
C) as many stars as brackets in part A

- one stack symbol
- first-order stack counts the number of currently open brackets
- a copy is done after each bracket

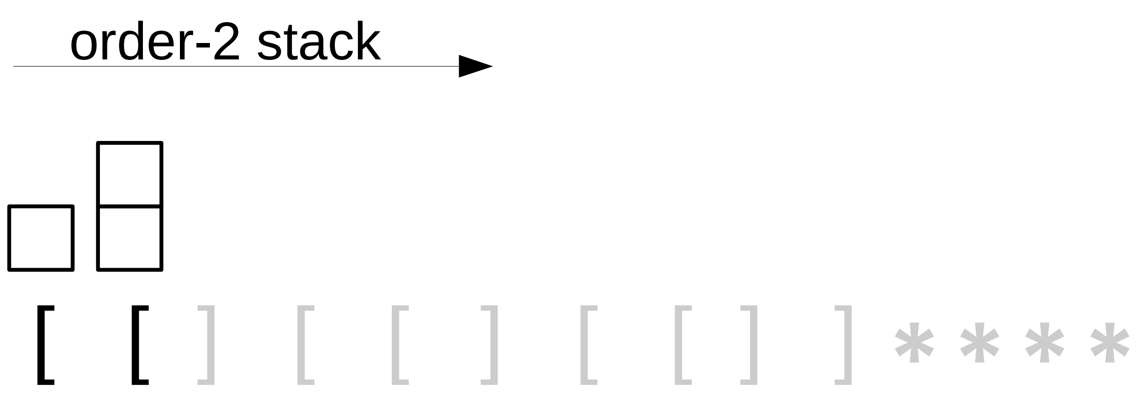
[[ ] [[ ] [[ ]] * * * *

Collapsible pushdown automata

How collapse can be useful? – Urzyczyn's language U
(≈ branches in the Urzyczyn's tree)

Words in U:
A) a prefix of a well-bracketed word
B) a well-bracketed word
C) as many stars as brackets in part A

- one stack symbol
- first-order stack counts the number of currently open brackets
- a copy is done after each bracket



Collapsible pushdown automata

How collapse can be useful? – Urzyczyn's language U

(≈ branches in the Urzyczyn's tree)

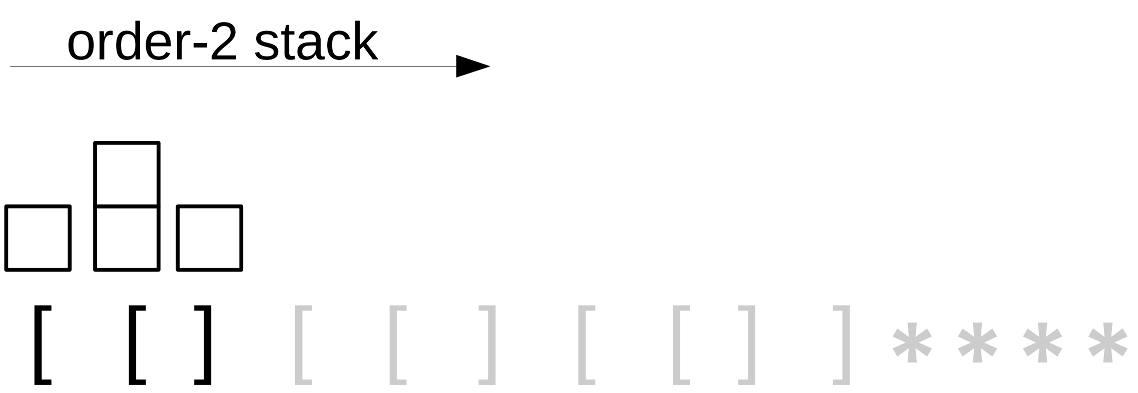
Words in U:

A) a prefix of a well-bracketed word

B) a well-bracketed word

C) as many stars as brackets in part A

- one stack symbol
- first-order stack counts the number of currently open brackets
- a copy is done after each bracket

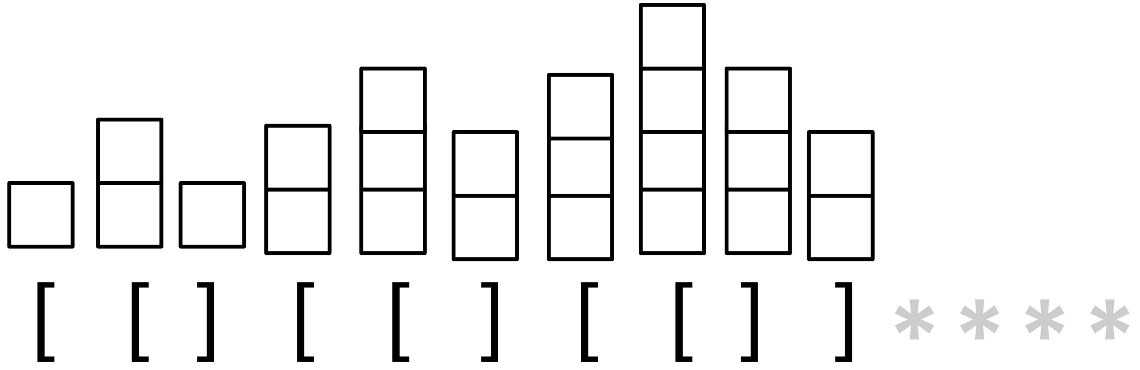


Collapsible pushdown automata

How collapse can be useful? – Urzyczyn's language U
(≈ branches in the Urzyczyn's tree)

Words in U:
A) a prefix of a well-bracketed word
B) a well-bracketed word
C) as many stars as brackets in part A

- one stack symbol
- first-order stack counts the number of currently open brackets
- a copy is done after each bracket



Collapsible pushdown automata

How collapse can be useful? – Urzyczyn's language U

(≈ branches in the Urzyczyn's tree)

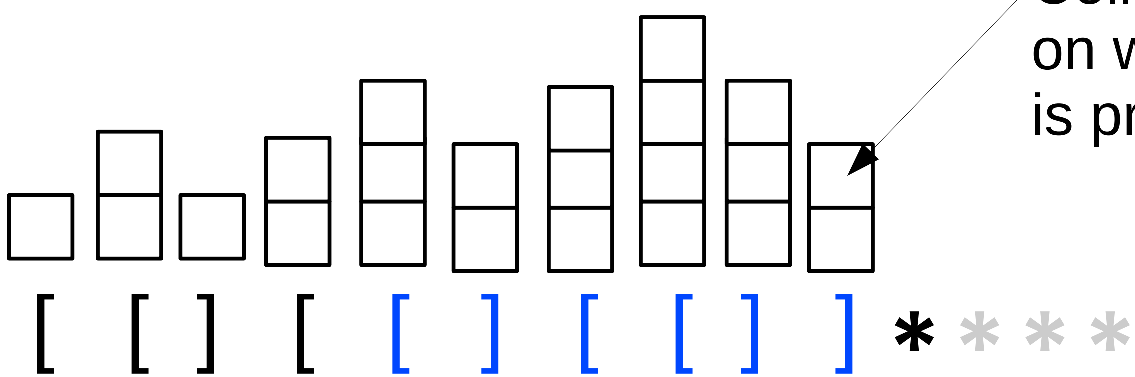
Words in U:

A) a prefix of a well-bracketed word

B) a well-bracketed word

C) as many stars as brackets in part A

- one stack symbol
- first-order stack counts the number of currently open brackets
- a copy is done after each bracket
- on the first star make the collapse
- count the number of stacks



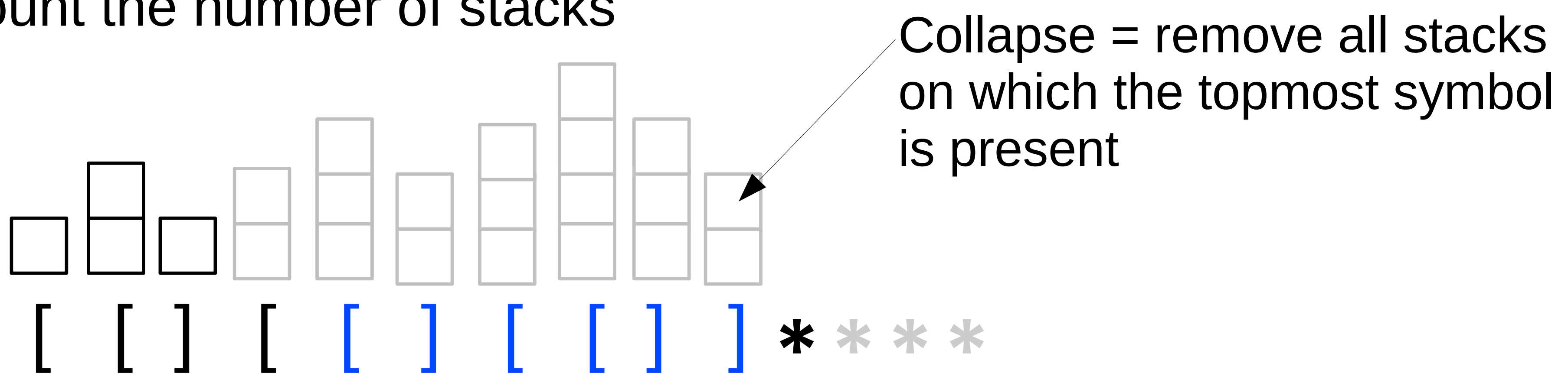
Collapse = remove all stacks on which the topmost symbol is present

Collapsible pushdown automata

How collapse can be useful? – Urzyczyn's language U
(≈ branches in the Urzyczyn's tree)

Words in U:
A) a prefix of a well-bracketed word
B) a well-bracketed word
C) as many stars as brackets in part A

- one stack symbol
- first-order stack counts the number of currently open brackets
- a copy is done after each bracket
- on the first star make the collapse
- count the number of stacks



How collapse can be useful? – Urzyczyn's language U

(≈ branches in the Urzyczyn's tree)

Words in U:
A) a prefix of a well-bracketed word
B) a well-bracketed word
C) as many stars as brackets in part A

- one stack symbol
- first-order stack counts the number of currently open brackets
- a copy is done after each bracket
- on the first star make the collapse
- count the number of stacks

Remark:
A nondeterministic order-2 PDA without collapse can recognize U,
as it can guess when is the beginning of the “B” part.
But not a deterministic HOPDA without collapse, of any order!
(This means that the Urzyczyn's tree cannot be generated by a HOPDA)

Expressivity questions

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

$$
\begin{array}{ccccccccc}
\mathrm{SafeTree}(0) & \subseteq & \mathrm{SafeTree}(1) & \subseteq & \mathrm{SafeTree}(2) & \subseteq & \mathrm{SafeTree}(3) & \subseteq & \dots \\
\rotatebox[origin=c]{-90}{$\subseteq$} & & \rotatebox[origin=c]{-90}{$\subseteq$} & & \rotatebox[origin=c]{-90}{$\subseteq$} & & \rotatebox[origin=c]{-90}{$\subseteq$} & & \\
\mathrm{Tree}(0) & \subseteq & \mathrm{Tree}(1) & \subseteq & \mathrm{Tree}(2) & \subseteq & \mathrm{Tree}(3) & \subseteq & \dots
\end{array}
$$

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

$$
\begin{array}{ccccccccc}
\mathrm{SafeLang}(0) & \subseteq & \mathrm{SafeLang}(1) & \subseteq & \mathrm{SafeLang}(2) & \subseteq & \mathrm{SafeLang}(3) & \subseteq & \dots \\
\rotatebox[origin=c]{-90}{$\subseteq$} & & \rotatebox[origin=c]{-90}{$\subseteq$} & & \rotatebox[origin=c]{-90}{$\subseteq$} & & \rotatebox[origin=c]{-90}{$\subseteq$} & & \\
\mathrm{Lang}(0) & \subseteq & \mathrm{Lang}(1) & \subseteq & \mathrm{Lang}(2) & \subseteq & \mathrm{Lang}(3) & \subseteq & \dots
\end{array}
$$

Expressivity questions

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

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

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

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

Expressivity questions

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

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

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

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

Expressivity questions

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

SafeTree(0)	$\subseteq$	SafeTree(1)	$\subseteq$	SafeTree(2)	$\subseteq$	SafeTree(3)	$\subseteq$	...
$\parallel$		$\parallel$		$\cap\vert$		$\cap\vert$		
Tree(0)	$\subseteq$	Tree(1)	$\subseteq$	Tree(2)	$\subseteq$	Tree(3)	$\subseteq$	...
$\parallel$		$\parallel$						
regular trees		context-free trees						

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

SafeLang(0)	$\subseteq$	SafeLang(1)	$\subseteq$	SafeLang(2)	$\subseteq$	SafeLang(3)	$\subseteq$	...
$\parallel$		$\parallel$		$\parallel$		$\cap\vert$		
Lang(0)	$\subseteq$	Lang(1)	$\subseteq$	Lang(2)	$\subseteq$	Lang(3)	$\subseteq$	...
$\parallel$		$\parallel$		$\parallel$				
regular languages		context-free languages		indexed languages				

Expressivity questions

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

SafeTree(0)	$\subseteq$	SafeTree(1)	$\subseteq$	SafeTree(2)	$\subseteq$	SafeTree(3)	$\subseteq$	...
$\parallel$		$\parallel$		$\cap\!\mid$		$\cap\!\mid$		
Tree(0)	$\subseteq$	Tree(1)	$\subseteq$	Tree(2)	$\subseteq$	Tree(3)	$\subseteq$	...
$\parallel$		$\parallel$						
regular trees		context-free trees						

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

SafeLang(0)	$\subseteq$	SafeLang(1)	$\subseteq$	SafeLang(2)	$\subseteq$	SafeLang(3)	$\subseteq$	...
$\parallel$		$\parallel$		$\parallel$		$\cap\!\mid$		
Lang(0)	$\subseteq$	Lang(1)	$\subseteq$	Lang(2)	$\subseteq$	Lang(3)	$\subseteq$	...
$\parallel$		$\parallel$		$\parallel$				
regular languages		context-free languages		indexed languages				

Are these hierarchies strict?

Expressivity questions

SafeTree(0)	$\not\subsetneq$	SafeTree(1)	$\not\subsetneq$	SafeTree(2)	$\not\subsetneq$	SafeTree(3)	$\not\subsetneq$	...
$=$		$=$		$\subseteq$		$\subseteq$		
Tree(0)	$\subseteq$	Tree(1)	$\subseteq$	Tree(2)	$\subseteq$	Tree(3)	$\subseteq$	...

SafeLang(0)	$\not\subsetneq$	SafeLang(1)	$\not\subsetneq$	SafeLang(2)	$\not\subsetneq$	SafeLang(3)	$\not\subsetneq$	...
$=$		$=$		$=$		$\subseteq$		
Lang(0)	$\subseteq$	Lang(1)	$\subseteq$	Lang(2)	$\subseteq$	Lang(3)	$\subseteq$	...

Are these hierarchies strict?

Theorem [Engelfriet 1991]
For every n, SafeLang(n) ≠ SafeLang($n+1$),
and thus also SafeTree(n) ≠ SafeTree($n+1$).

Separating language: correct sequences of operations of order-($n+1$) HOPDA (including the topmost stack symbol after every step).
Proof: “Simple trick” using the fact that reachability for order-n HOPDA is in (n-1)-EXPTIME, while reachability for order-($n+1$) HOPDA is n-EXPTIME-hard.

Expressivity questions

SafeTree(0)	$\not\subsetneq$	SafeTree(1)	$\not\subsetneq$	SafeTree(2)	$\not\subsetneq$	SafeTree(3)	$\not\subsetneq$	...
$=$		$=$		$\subseteq$		$\subseteq$		
Tree(0)	$\not\subsetneq$	Tree(1)	$\not\subsetneq$	Tree(2)	$\not\subsetneq$	Tree(3)	$\not\subsetneq$	...

SafeLang(0)	$\not\subsetneq$	SafeLang(1)	$\not\subsetneq$	SafeLang(2)	$\not\subsetneq$	SafeLang(3)	$\not\subsetneq$	...
$=$		$=$		$=$		$\subseteq$		
Lang(0)	$\not\subsetneq$	Lang(1)	$\not\subsetneq$	Lang(2)	$\not\subsetneq$	Lang(3)	$\not\subsetneq$	...

Are these hierarchies strict?

Theorem [Engelfriet 1991]
For every n, SafeLang(n) ≠ SafeLang($n+1$),
and thus also SafeTree(n) ≠ SafeTree($n+1$).

Separating language: correct sequences of operations of order-($n+1$) HOPDA
(including the topmost stack symbol after every step).
Proof: "Simple trick" using the fact that reachability for order-n HOPDA is in (n-1)-EXPTIME, while reachability for order-($n+1$) HOPDA is n-EXPTIME-hard.

The same proof works for CPDA.
Thus Tree(n) ≠ Tree($n+1$) & Lang(n) ≠ Lang($n+1$).

Expressivity questions

$$
\begin{array}{ccccccccc}
\text{SafeTree}(0) & \not\subsetneq & \text{SafeTree}(1) & \not\subsetneq & \text{SafeTree}(2) & \not\subsetneq & \text{SafeTree}(3) & \not\subsetneq & \dots \\
\| & & \| & & \rotatebox{90}{$\subseteq$} & & \rotatebox{90}{$\subseteq$} & & \\
\text{Tree}(0) & \not\subsetneq & \text{Tree}(1) & \not\subsetneq & \text{Tree}(2) & \not\subsetneq & \text{Tree}(3) & \not\subsetneq & \dots
\end{array}
$$

$$
\begin{array}{ccccccccc}
\text{SafeLang}(0) & \not\subsetneq & \text{SafeLang}(1) & \not\subsetneq & \text{SafeLang}(2) & \not\subsetneq & \text{SafeLang}(3) & \not\subsetneq & \dots \\
\| & & \| & & \| & & \rotatebox{90}{$\subseteq$} & & \\
\text{Lang}(0) & \not\subsetneq & \text{Lang}(1) & \not\subsetneq & \text{Lang}(2) & \not\subsetneq & \text{Lang}(3) & \not\subsetneq & \dots
\end{array}
$$

Are these hierarchies strict?

Another separator:

T_n = tree with branches $a^k b^{exp_n(k)} c$, where $exp_n(k)=\underbrace{2^{2^{\cdot^{\cdot^{\cdot^{2^k}}}}}}_{n}$

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

pumping lemma [Kartzow, P. 2012]

Expressivity questions

SafeTree(0)	$\subsetneq$	SafeTree(1)	$\subsetneq$	SafeTree(2)	$\subsetneq$	SafeTree(3)	$\subsetneq$	...
$\parallel$		$\parallel$		$\cap$		$\cap$		
Tree(0)	$\subsetneq$	Tree(1)	$\subsetneq$	Tree(2)	$\subsetneq$	Tree(3)	$\subsetneq$	...

SafeLang(0)	$\subsetneq$	SafeLang(1)	$\subsetneq$	SafeLang(2)	$\subsetneq$	SafeLang(3)	$\subsetneq$	...
$\parallel$		$\parallel$		$\parallel$		$\cap$		
Lang(0)	$\subsetneq$	Lang(1)	$\subsetneq$	Lang(2)	$\subsetneq$	Lang(3)	$\subsetneq$	...

Are these hierarchies strict?

Another separator:

T_n = tree with branches $a^k b^{exp_n(k)} c$, where $exp_n(k)=\underbrace{2^{2^{\cdot^{\cdot^{2^k}}}}}_{n}$

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

pumping lemma [Kartzow, P. 2012]

For languages we do not know:

SafeLang($n+1$) $\ni \{b^{exp_n(k)} : k \in \mathbb{N}\} \overset{?}{\notin}$ Lang(n).

Open problem: a pumping lemma for nondeterministic HORSes.

Expressivity questions

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

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

<u>Is safety really a restriction?</u>

Expressivity questions

$$
\begin{array}{ccccccccc}
\text{SafeTree}(0) & \subsetneq & \text{SafeTree}(1) & \subsetneq & \text{SafeTree}(2) & \subsetneq & \text{SafeTree}(3) & \subsetneq & \cdots \\
\| & & \| & & \rotatebox{90}{$\subsetneq$} & & \rotatebox{90}{$\subsetneq$} & & \\
\text{Tree}(0) & \subsetneq & \text{Tree}(1) & \subsetneq & \text{Tree}(2) & \subsetneq & \text{Tree}(3) & \subsetneq & \cdots
\end{array}
$$

$$
\begin{array}{ccccccccc}
\text{SafeLang}(0) & \subsetneq & \text{SafeLang}(1) & \subsetneq & \text{SafeLang}(2) & \subsetneq & \text{SafeLang}(3) & \subsetneq & \cdots \\
\| & & \| & & \| & & \rotatebox{90}{$\subseteq$} & & \\
\text{Lang}(0) & \subsetneq & \text{Lang}(1) & \subsetneq & \text{Lang}(2) & \subsetneq & \text{Lang}(3) & \subsetneq & \cdots
\end{array}
$$

Is safety really a restriction?

For trees – yes.

Example: Urzyczyn's tree U

$\text{Tree}(2) \ni U \notin \text{SafeTree}(n)$ for every n [P. 2012]

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

Expressivity questions

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

Are these languages context-sensitive?

Expressivity questions

$$\begin{array}{ccccccccc}
\text{SafeLang}(0) & \subsetneq & \text{SafeLang}(1) & \subsetneq & \text{SafeLang}(2) & \subsetneq & \text{SafeLang}(3) & \subsetneq & \cdots \subseteq \text{CSens} \\
\| & & \| & & \| & & \rotatebox[origin=c]{-90}{$\subseteq$} & & \\
\text{Lang}(0) & \subsetneq & \text{Lang}(1) & \subsetneq & \text{Lang}(2) & \subsetneq & \text{Lang}(3) & \subsetneq & \cdots
\end{array}$$

Are these languages context-sensitive?

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

SafeLang(n) $\subseteq$ CSens, for every n [Inaba, Maneth 2008]

Expressivity questions

$$
\begin{array}{ccccccccccc}
\text{SafeLang}(0) & \subsetneq & \text{SafeLang}(1) & \subsetneq & \text{SafeLang}(2) & \subsetneq & \text{SafeLang}(3) & \subsetneq & \cdots & \subseteq & \text{CSens} \\
\| & & \| & & \| & & \rotatebox{-90}{$\subseteq$} & & & & \\
\text{Lang}(0) & \subsetneq & \text{Lang}(1) & \subsetneq & \text{Lang}(2) & \subsetneq & \text{Lang}(3) & \subsetneq & \cdots & & \\
 & & & & & & \rotatebox{-90}{$\subseteq$} & & & & \\
 & & & & & & \text{CSens} & & & &
\end{array}
$$

Are these languages context-sensitive?

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

SafeLang(n) ⊆ CSens, for every n [Inaba, Maneth 2008]

Lang(3) ⊆ CSens [Kobayashi, Inaba, Tsukada 2014]

Expressivity questions

$$
\begin{array}{ccccccccccc}
\text{SafeLang}(0) & \subsetneq & \text{SafeLang}(1) & \subsetneq & \text{SafeLang}(2) & \subsetneq & \text{SafeLang}(3) & \subsetneq & \cdots & \subseteq & \text{CSens} \\
\parallel & & \parallel & & \parallel & & \rotatebox{90}{$\subseteq$} & & & & \\
\text{Lang}(0) & \subsetneq & \text{Lang}(1) & \subsetneq & \text{Lang}(2) & \subsetneq & \text{Lang}(3) & \subsetneq & \cdots & & \\
 & & & & & & \rotatebox{90}{$\subseteq$} & & & & \\
 & & & & & & \text{CSens} & & & &
\end{array}
$$

Are these languages context-sensitive?

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

SafeLang(n) $\subseteq$ CSens, for every n [Inaba, Maneth 2008]

Lang(3) $\subseteq$ CSens [Kobayashi, Inaba, Tsukada 2014]

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

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?

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

This inclusion is "almost obvious":

- Recall that CSens = languages recognized by a nondeterministic Turing machine in linear space.

Expressivity questions

$$\begin{array}{ccccccccc}
\text{SafeLang}(0) & \subsetneq & \text{SafeLang}(1) & \subsetneq & \text{SafeLang}(2) & \subsetneq & \text{SafeLang}(3) & \subsetneq & \dots \subseteq \text{CSens} \\
\| & & \| & & \| & & \rotatebox{90}{$\subseteq$} & & \\
\text{Lang}(0) & \subsetneq & \text{Lang}(1) & \subsetneq & \text{Lang}(2) & \subsetneq & \text{Lang}(3) & \subsetneq & \dots \\
 & & & & & & \rotatebox{90}{$\subseteq$} & & \\
 & & & & & & \text{CSens} & &
\end{array}$$

Are these languages context-sensitive?

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

This inclusion is "almost obvious":

- Recall that CSens = languages recognized by a nondeterministic Turing machine in linear space.
- Consider the following algorithm: starting from the initial nonterminal, follow nondeterministically rules of the HORS, trying to derive the input word.

Expressivity questions

$$\begin{array}{ccccccccccc}
\text{SafeLang}(0) & \subsetneq & \text{SafeLang}(1) & \subsetneq & \text{SafeLang}(2) & \subsetneq & \text{SafeLang}(3) & \subsetneq & \ldots & \subseteq & \text{CSens} \\
\| & & \| & & \| & & \rotatebox{90}{$\subseteq$} & & & & \\
\text{Lang}(0) & \subsetneq & \text{Lang}(1) & \subsetneq & \text{Lang}(2) & \subsetneq & \text{Lang}(3) & \subsetneq & \ldots & & \\
 & & & & & & \rotatebox{90}{$\subseteq$} & & & & \\
 & & & & & & \text{CSens} & & & &
\end{array}$$

Are these languages context-sensitive?

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

This inclusion is "almost obvious":

- Recall that CSens = languages recognized by a nondeterministic Turing machine in linear space.
- Consider the following algorithm: starting from the initial nonterminal, follow nondeterministically rules of the HORS, trying to derive the input word.
- It works well if all intermediate terms are smaller than the derived word (= input word).
- The "only difficulty": describe/eliminate nonterminals that are "not productive", i.e., that do not increase the size of the derived word.

Algorithmic questions

Problem: MSO model-checking

Input: MSO formula ϕ, HORS S

Output: does ϕ hold in the tree generated by S?

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.

[Knapik, Niwiński, Urzyczyn 2002] – safe schemes only
[Knapik, Niwiński, Urzyczyn, Walukiewicz 2005] – order-2 only
[Ong 2006] – via game semantics
[Hague, Murawski, Ong, Serre 2008] – via collapsible pushdown automata
[Broadbent, Ong 2009] – global model-checking
[Kobayashi, Ong 2009] – via a type system
[Broadbent, Carayol, Ong, Serre 2010] – MSO reflection
[Salvati, Walukiewicz 2011] – via Krivine machine
[Carayol, Serre 2012] – MSO selection
[Salvati, Walukiewicz 2015] – model for λY-calculus
...

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

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

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)

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

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 *(see next talk)*

Algorithmic questions

Theorem: MSO model-checking is decidable.

Idea of a proof

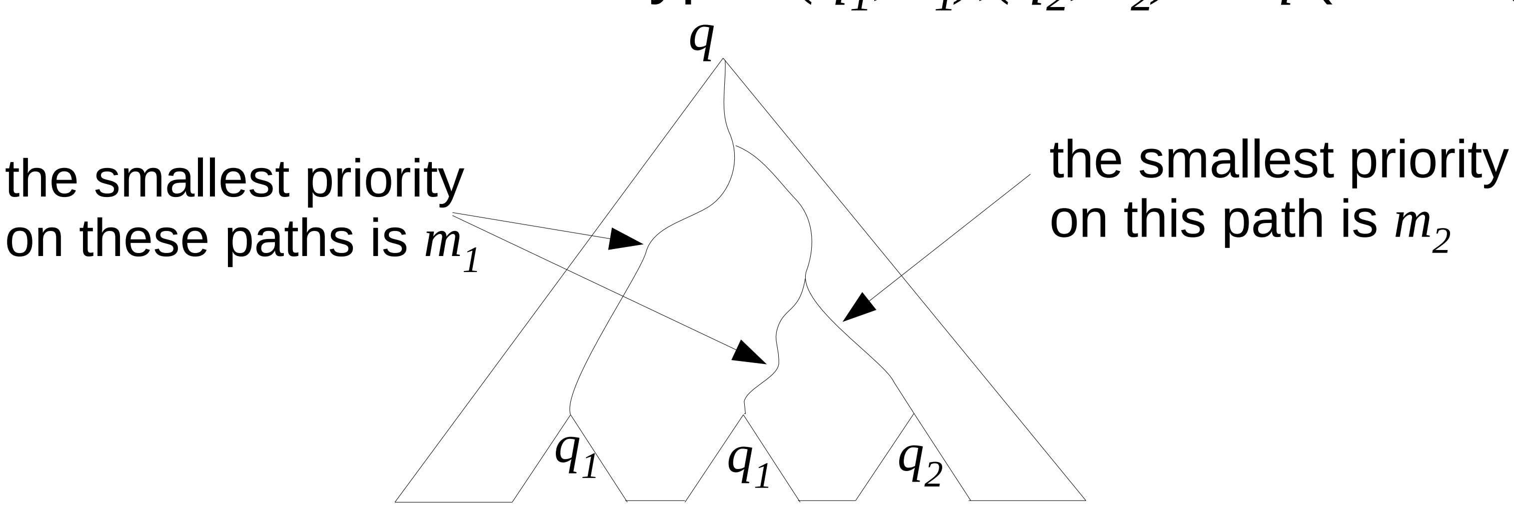
Input: alternating parity automaton A, HORS S
Question: does A accept the tree generated by S?

We refine simple types into intersection types of the form:

o is refined to $q \in Q$ (a state)

$\alpha \to \beta$ is refined to $\{(\tau_1, m_1), \ldots, (\tau_k, m_k)\} \to \tau$ where τ_i refines α, τ refines β, m_i is a priority

Intuition: a function with type $\{(q_1, m_1), (q_2, m_2)\} \to q$ (refining $o \to o$):



Algorithmic questions

Theorem: MSO model-checking is decidable.

[Knapik, Niwiński, Urzyczyn 2002] – safe schemes only
[Knapik, Niwiński, Urzyczyn, Walukiewicz 2005] – order-2 only
[Ong 2006] – via game semantics
[Hague, Murawski, Ong, Serre 2008] – via collapsible pushdown automata
[Broadbent, Ong 2009] – global model-checking
[Kobayashi, Ong 2009] – via a type system
[Broadbent, Carayol, Ong, Serre 2010] – MSO reflection ??
[Salvati, Walukiewicz 2011] – via Krivine machine
[Carayol, Serre 2012] – MSO selection
[Salvati, Walukiewicz 2015] – model for λY-calculus
...

MSO reflection

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

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

Algorithmic questions

Theorem: MSO model-checking is decidable.

[Knapik, Niwiński, Urzyczyn 2002] – safe schemes only
[Knapik, Niwiński, Urzyczyn, Walukiewicz 2005] – order-2 only
[Ong 2006] – via game semantics
[Hague, Murawski, Ong, Serre 2008] – via collapsible pushdown automata
[Broadbent, Ong 2009] – global model-checking
[Kobayashi, Ong 2009] – via a type system
[Broadbent, Carayol, Ong, Serre 2010] – MSO reflection ??
[Salvati, Walukiewicz 2011] – via Krivine machine
[Carayol, Serre 2012] – MSO selection ??
[Salvati, Walukiewicz 2015] – model for λY-calculus
...

MSO reflection

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

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

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.

Algorithmic questions

Theorem: MSO model-checking is decidable.

[Knapik, Niwiński, Urzyczyn 2002] – safe schemes only
[Knapik, Niwiński, Urzyczyn, Walukiewicz 2005] – order-2 only
[Ong 2006] – via game semantics
[Hague, Murawski, Ong, Serre 2008] – via collapsible pushdown automata
[Broadbent, Ong 2009] – global model-checking
[Kobayashi, Ong 2009] – via a type system
[Broadbent, Carayol, Ong, Serre 2010] – MSO reflection ??
[Salvati, Walukiewicz 2011] – via Krivine machine
[Carayol, Serre 2012] – MSO selection ??
[Salvati, Walukiewicz 2015] – model for λY-calculus ??
...

MSO reflection

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

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

MSO selection

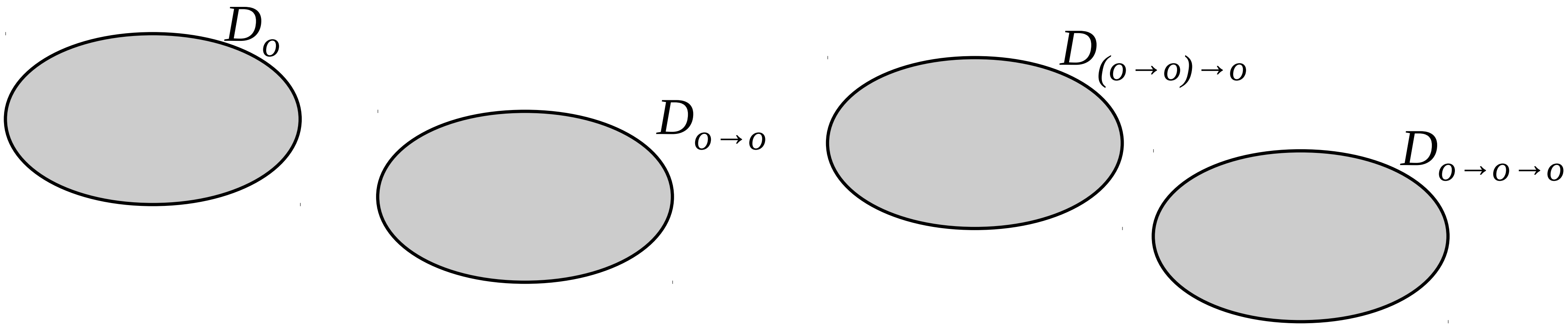
Input: MSO formula $\phi(X)$, HORS S

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.

What is a model?

Input: MSO formula ϕ, HORS S

Output: a finite set D_α for every sort α



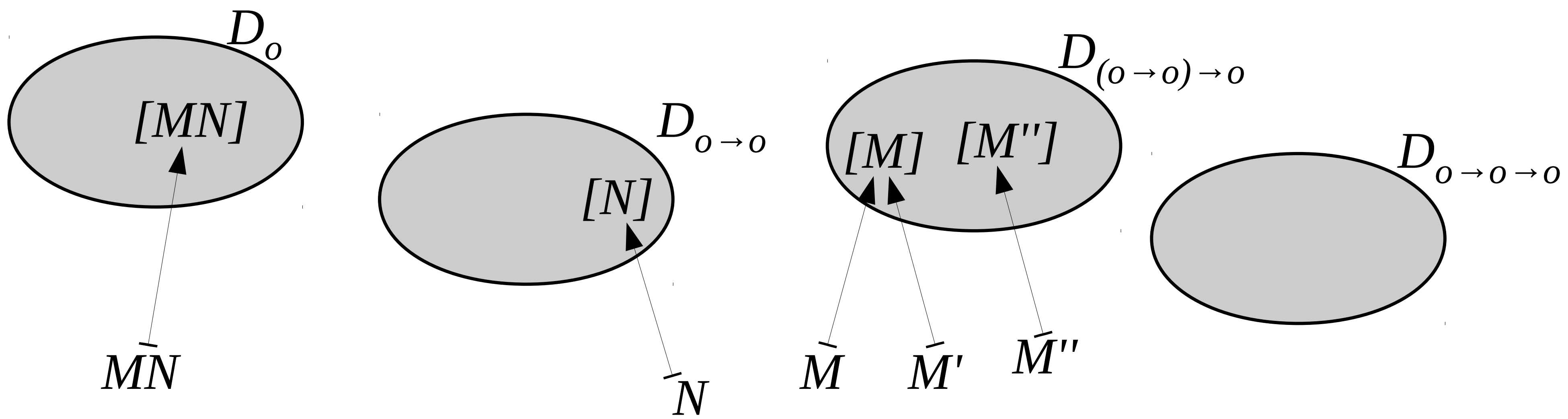
What is a model?

$[M]$ depends on valuation of free variables of M

Input: MSO formula ϕ, HORS S

Output: a finite set D_α for every sort α,

a value $[M]\in D_\alpha$ for every term M sort α



What is a model?

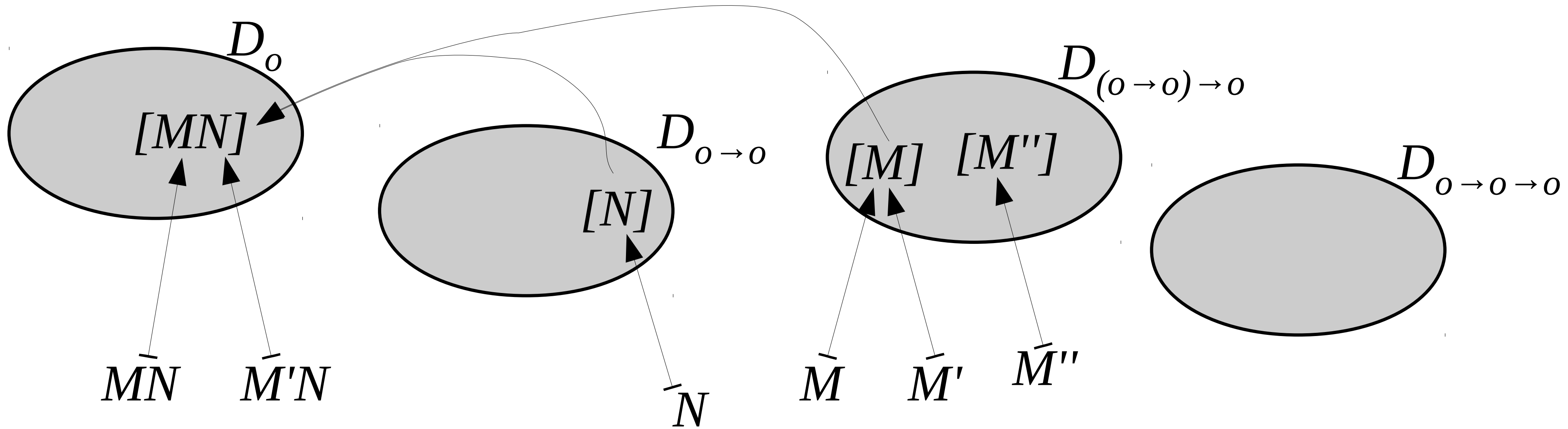
$[M]$ depends on valuation of free variables of M

Input: MSO formula ϕ, HORS S

Output: a finite set D_α for every sort α,

a value $[M] \in D_\alpha$ for every term M sort α

has to be compositional: $[MN]$ determined by $[M]$ and $[N]$.



What is a model?

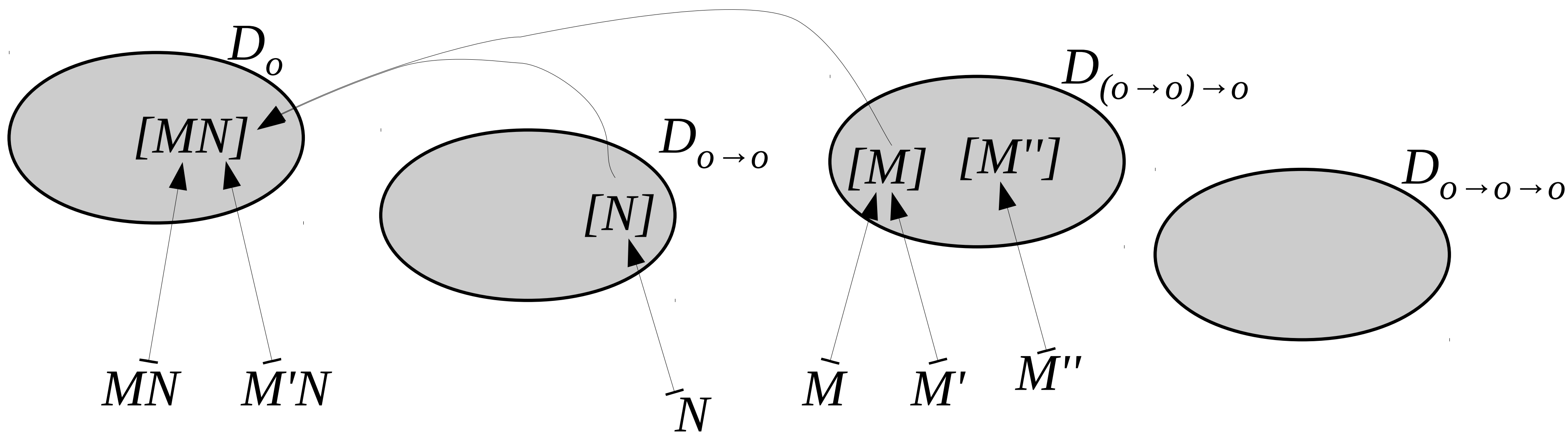
$[M]$ depends on valuation of free variables of M

Input: MSO formula ϕ, HORS S

Output: a finite set D_α for every sort α,

a value $[M] \in D_\alpha$ for every term M sort α

has to be compositional: $[MN]$ determined by $[M]$ and $[N]$.



model ⇒ reflection

(we enrich the scheme so that every term “knows” its value in the model)

What is a model?

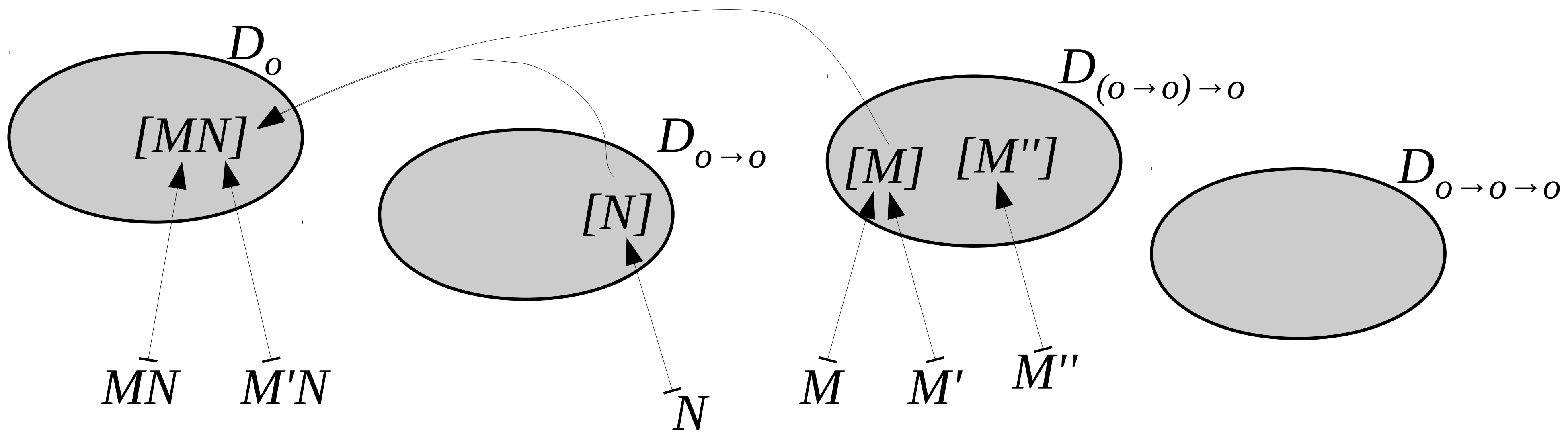
Input: MSO formula ϕ, HORS S

Output: a finite set D_α for every sort α,

a value $[M]\in D_\alpha$ for every term M sort α

has to be compositional: $[MN]$ determined by $[M]$ and $[N]$.

[M] depends on valuation of free variables of M



model ⇒ reflection

⇓ transfer theorem

(we enrich the scheme so that every term "knows" its value in the model)

What is a model?

[M] depends on valuation of free variables of M

Input: MSO formula ϕ, HORS S

Output: a finite set D_α for every sort α,

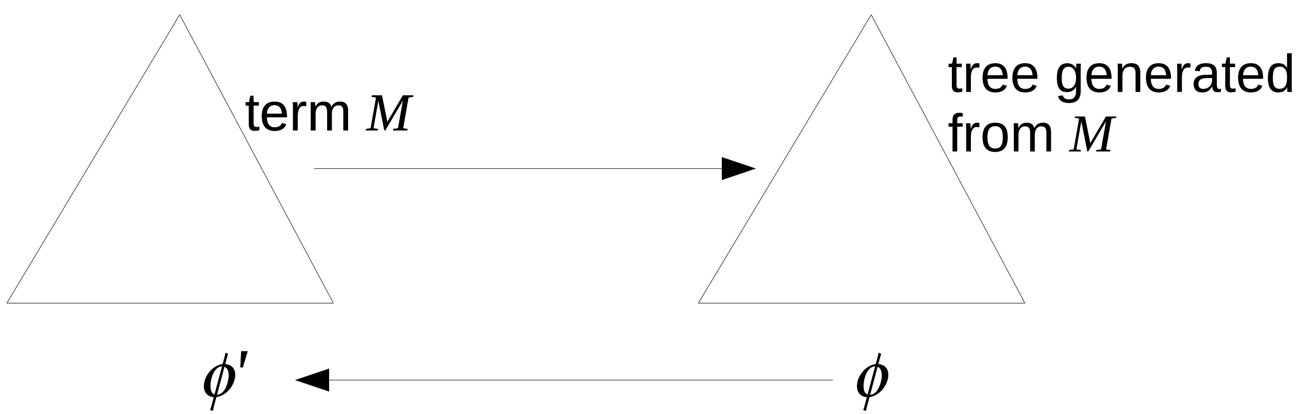
a value $[M] \in D_\alpha$ for every term M sort α

has to be compositional: $[MN]$ determined by $[M]$ and $[N]$.

model ⟹ reflection

⇘ transfer theorem

Basing on ϕ one can construct ϕ' such that ϕ' holds in a closed term M of sort o iff ϕ holds in the tree generated from M.



(special case: M = starting nonterminal)

Beyond MSO?

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

Ongoing work: WMSO+U model-checking is decidable.

MSO+U = Weak MSO (set quantifiers range over finite sets only)
+ new quantifier U

where: U$X.\phi$ means that ϕ holds for some arbitrarily large finite sets X

Downward closure

Let L be a set of words. Its downward closure $L\downarrow$ contains all words that can be obtained from words in L by removing some letters.

E.g. $L=\{abc\}$, $L\downarrow=\{e,a,b,c,ab,bc,ac,abc\}$

Downward closure

Let L be a set of words. Its downward closure $L\downarrow$ contains all words that can be obtained from words in L by removing some letters.

E.g. $L=\{abc\}$, $L\downarrow=\{e,a,b,c,ab,bc,ac,abc\}$

Higman's lemma: the downward closure of any set L is a regular language.

Downward closure

Let L be a set of words. Its downward closure $L\downarrow$ contains all words that can be obtained from words in L by removing some letters.

E.g. $L=\{abc\}$, $L\downarrow=\{e,a,b,c,ab,bc,ac,abc\}$

Higman's lemma: the downward closure of any set L is a regular language.

Quest: Given a scheme S recognizing L, compute $L\downarrow$.

Downward closure

Let L be a set of words. Its downward closure $L\downarrow$ contains all words that can be obtained from words in L by removing some letters.

E.g. $L=\{abc\}$, $L\downarrow=\{e,a,b,c,ab,bc,ac,abc\}$

Higman's lemma: the downward closure of any set L is a regular language.

Quest: Given a scheme S recognizing L, compute $L\downarrow$.

- Trivial but useless: Compute a scheme S' recognizing $L\downarrow$.

Downward closure

Let L be a set of words. Its downward closure $L\downarrow$ contains all words that can be obtained from words in L by removing some letters.

E.g. $L=\{abc\}$, $L\downarrow=\{e,a,b,c,ab,bc,ac,abc\}$

Higman's lemma: the downward closure of any set L is a regular language.

Quest: Given a scheme S recognizing L, compute $L\downarrow$.

- Trivial but useless: Compute a scheme S' recognizing $L\downarrow$.
- Real quest: Compute an NFA A recognizing $L\downarrow$.

Downward closure

Let L be a set of words. Its downward closure $L\downarrow$ contains all words that can be obtained from words in L by removing some letters.

E.g. $L=\{abc\}$, $L\downarrow=\{e,a,b,c,ab,bc,ac,abc\}$

Higman's lemma: the downward closure of any set L is a regular language.

Quest: Given a scheme S recognizing L, compute $L\downarrow$.

- Trivial but useless: Compute a scheme S' recognizing $L\downarrow$.
- Real quest: Compute an NFA A recognizing $L\downarrow$.

Theorem [Zetzsche 2015, Hague, Kochems, Ong 2016, Clemente, P., Salvati, Walukiewicz 2016]
Given a scheme S recognizing L, one can compute an NFA A recognizing $L\downarrow$.

Downward closure

Theorem

Given a scheme S recognizing L, one can compute an NFA A recognizing $L\downarrow$.

Some ideas:

- For every regular language K we check whether $L\downarrow = K$.

Downward closure

Theorem

Given a scheme S recognizing L, one can compute an NFA A recognizing $L\downarrow$.

Some ideas:

- For every regular language K we check whether $L\downarrow = K$.
- Easy to test whether $L\downarrow \subseteq K$, i.e. $L\downarrow \cap \overline{K} = \emptyset$.

Downward closure

Theorem

Given a scheme S recognizing L, one can compute an NFA A recognizing $L\downarrow$.

Some ideas:

- For every regular language K we check whether $L\downarrow = K$.
- Easy to test whether $L\downarrow \subseteq K$, i.e. $L\downarrow \cap \overline{K} = \emptyset$.
- $L\downarrow$ (so K as well) is necessarily a finite union of languages of the form $S_i = A_0^* a_1^? A_1^* a_2^? \ldots A_{k-1}^* a_k^? A_k^*$. It remains to check whether $S_i \subseteq L\downarrow$ for all i.

Downward closure

Theorem

Given a scheme S recognizing L, one can compute an NFA A recognizing $L\downarrow$.

Some ideas:

- For every regular language K we check whether $L\downarrow = K$.
- Easy to test whether $L\downarrow \subseteq K$, i.e. $L\downarrow \cap \overline{K} = \emptyset$.
- $L\downarrow$ (so K as well) is necessarily a finite union of languages of the form $S_i = A_0^* a_1^? A_1^* a_2^? \dots A_{k-1}^* a_k^? A_k^*$. It remains to check whether $S_i \subseteq L\downarrow$ for all i.
- By transforming the scheme, this reduces to the diagonal problem:

Input: a scheme S recognizing $L \subseteq a_1^* a_2^* \dots a_k^*$ (with different letters)

Question: does $L\downarrow = a_1^* a_2^* \dots a_k^*$?

(in other words: is it the case that for every n we have in L words with more than n appearances of every letter?)

This is the actual problem to be solved.

The diagonal problem

Input: a scheme S recognizing $L \subseteq a_1^* a_2^* \ldots a_k^*$ (with different letters)

Question: does $L\downarrow = a_1^* a_2^* \ldots a_k^*$?

How to solve it?

a scheme S of order n with a word written on a branch —step 1→ a scheme S of order n-1 with this word written in leaves

The diagonal problem

Input: a scheme S recognizing $L \subseteq a_1^* a_2^* \ldots a_k^*$ (with different letters)

Question: does $L{\downarrow} = a_1^* a_2^* \ldots a_k^*$?

How to solve it?

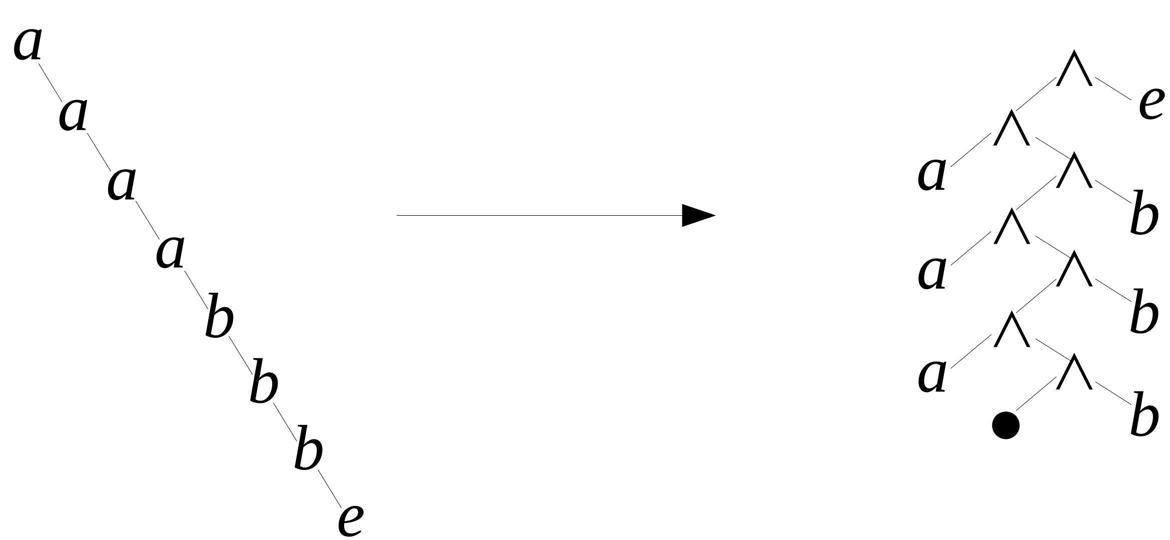
a scheme S of order n with a word written on a branch —step 1→ a scheme S of order n-1 with this word written in leaves

Example:

$S \to A\ e$
$A\ x \to a\ (A\ (b\ x))$
$A\ x \to x$
(rank 1: a, b; rank 0: e)

→

$S \to \wedge\ A\ e$
$A \to \wedge\ a\ (\wedge\ A\ b))$
$A \to \bullet$
(rank 2: $\wedge$; rank 0: a, b, e, $\bullet$)



The diagonal problem

Input: a scheme S recognizing $L \subseteq a_1^* a_2^* \ldots a_k^*$ (with different letters)

Question: does $L{\downarrow} = a_1^* a_2^* \ldots a_k^*$?

How to solve it?

a scheme S of order n with a word written on a branch —step 1→ a scheme S of order n-1 with this word written in leaves

Example:

$$S \to A\, e \qquad\qquad S \to \wedge\, A\, e$$
$$A\, x \to a\, (A\, (b\, x)) \quad\longrightarrow\quad A \to \wedge\, a\, (\wedge\, A\, b))$$
$$A\, x \to x \qquad\qquad A \to \bullet$$

(rank 1: a, b; rank 0: e) → (rank 2: $\wedge$; rank 0: a, b, e, $\bullet$)

Idea: 1) Observe that an argument of type o can be used at most once.

The diagonal problem

Input: a scheme S recognizing $L \subseteq a_1^* a_2^* \dots a_k^*$ (with different letters)

Question: does $L\!\downarrow = a_1^* a_2^* \dots a_k^*$?

How to solve it?

a scheme S of order n with a word written on a branch —step 1→ a scheme S of order n-1 with this word written in leaves

Example:

$$S \to A\, e \qquad\qquad S \to \wedge\, A\, e$$
$$A\, x \to a\, (A\, (b\, x)) \qquad \longrightarrow \qquad A \to \wedge\, a\, (\wedge\, A\, b))$$
$$A\, x \to x \qquad\qquad A \to \bullet$$

(rank 1: a, b; rank 0: e) (rank 2: $\wedge$; rank 0: a, b, e, $\bullet$)

Idea: 1) Observe that an argument of type o can be used at most once.
2) All arguments of type o are dropped ($\Rightarrow$ order decreases).
3) Every subterm $M\, N$ with N of type o can be replaced
a) either by $\wedge\, M\, N$ (when the argument is used in M),
b) or by M (when the argument is ignored in M).

The diagonal problem

Input: a scheme S recognizing $L \subseteq a_1^* a_2^* \ldots a_k^*$ (with different letters)

Question: does $L{\downarrow} = a_1^* a_2^* \ldots a_k^*$?

How to solve it?

a scheme S of order n with a word written on a branch —step 1→ a scheme S of order n-1 with this word written in leaves

Example:

$S \to A\ e$
$A\ x \to a\ (A\ (b\ x))$
$A\ x \to x$
(rank 1: a, b; rank 0: e)

→

$S \to \wedge\ A\ e$
$A \to \wedge\ a\ (\wedge\ A\ b))$
$A \to \bullet$
(rank 2: $\wedge$; rank 0: a, b, e, $\bullet$)

Idea: 1) Observe that an argument of type o can be used at most once.
2) All arguments of type o are dropped ($\Rightarrow$ order decreases).
3) Every subterm $M\ N$ with N of type o can be replaced
a) either by $\wedge\ M\ N$ (when the argument is used in M),
b) or by M (when the argument is ignored in M).
4) Additional work is required to choose correctly a) or b).

The diagonal problem

Input: a scheme S recognizing $L \subseteq a_1^* a_2^* ... a_k^*$ (with different letters)

Question: does $L\!\downarrow = a_1^* a_2^* ... a_k^*$?

How to solve it?

a scheme S of order n with a word written on a branch

step 1 →

a scheme S of order n-1 with this word written in leaves

step 2 →

a scheme S of order n-1 with a *similar* word written on a branch

The diagonal problem

Input: a scheme S recognizing $L \subseteq a_1^* a_2^* \ldots a_k^*$ (with different letters)

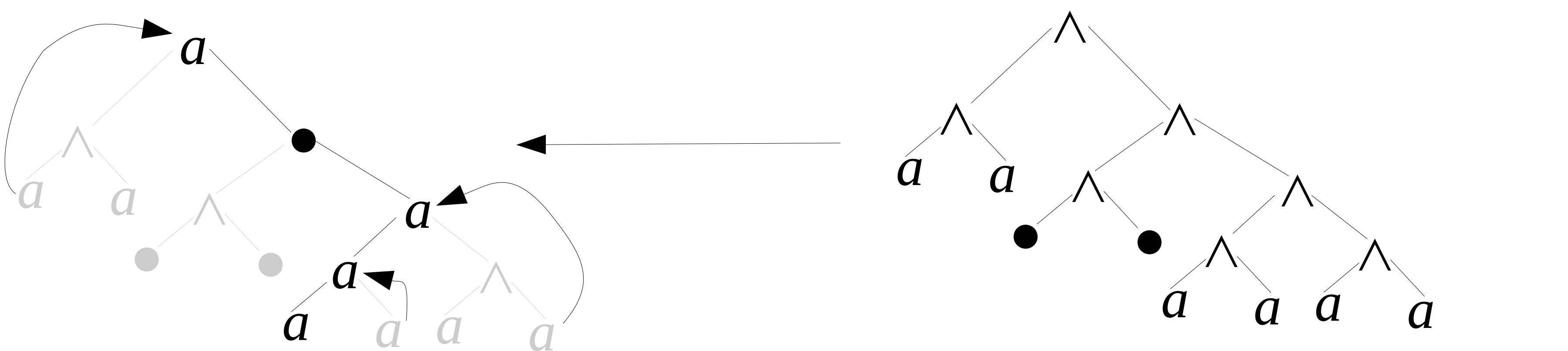
Question: does $L\!\downarrow = a_1^* a_2^* \ldots a_k^*$?

How to solve it?

a scheme S of order n with a word written on a branch —step 1→ a scheme S of order n-1 with this word written in leaves

—step 2→ a scheme S of order n-1 with a similar word written on a branch

Example:



Idea:

1) Choose (nondeterministically) only one branch.

2) For every removed subtree with a, write a new a just above.

The diagonal problem

Input: a scheme S recognizing $L \subseteq a_1^* a_2^* \ldots a_k^*$ (with different letters)

Question: does $L\!\downarrow = a_1^* a_2^* \ldots a_k^*$?

How to solve it?

a scheme S of order n with a word written on a branch —step 1→ a scheme S of order n-1 with this word written in leaves

—step 2→ a scheme S of order n-1 with a similar word written on a branch

Example:

Idea:

1) Choose (nondeterministically) only one branch.
2) For every removed subtree with a, write a new a just above.
3) The number of a's decreases at most logarithmically,
   if the branch is chosen correctly (always go to the subtree with more a's).

The diagonal problem

Input: a scheme S recognizing $L \subseteq a_1^* a_2^* \dots a_k^*$ (with different letters)

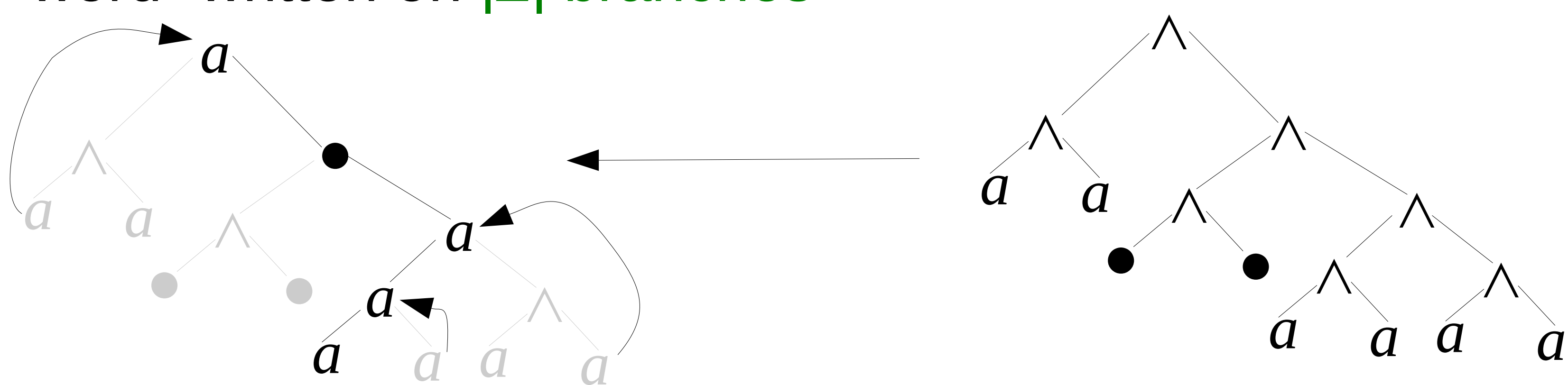
Question: does $L{\downarrow} = a_1^* a_2^* \dots a_k^*$?

How to solve it?

a scheme S of order n with a word written on a branch —step 1→ a scheme S of order n-1 with this word written in leaves

—step 2→ a scheme S of order n-1 with a similar "word" written on |Σ| branches

Example:



Idea:

1) Choose (nondeterministically) only one branch.

2) For every removed subtree with a, write a new a just above.

3) The number of a's decreases at most logarithmically,
   if the branch is chosen correctly (always go to the subtree with more a's).

We have to this for every letter ⇒ |Σ| branches

The diagonal problem

Input: a scheme S recognizing $L \subseteq a_1^* a_2^* \ldots a_k^*$ (with different letters)

Question: does $L{\downarrow} = a_1^* a_2^* \ldots a_k^*$?

How to solve it?

a scheme S of order n with a "word" written on |Σ| branches —step 1→ a scheme S of order n-1 with this "word" written in leaves

—step 2→ a scheme S of order n-1 with a similar "word" written on |Σ| branches

Example:

Idea:

1) Choose (nondeterministically) only one branch.

2) For every removed subtree with a, write a new a just above.

3) The number of a's decreases at most logarithmically,
   if the branch is chosen correctly (always go to the subtree with more a's).

We have to this for every letter ⇒ |Σ| branches

The diagonal problem

Input: a scheme S recognizing $L \subseteq a_1^* a_2^* \ldots a_k^*$ (with different letters)

Question: does $L\downarrow = a_1^* a_2^* \ldots a_k^*$?

How to solve it?

a scheme S of order n with a "word" written on $|\Sigma|$ branches

step 1 →

a scheme S of order n-1 with this "word" written in leaves

step 2 →

a scheme S of order n-1 with a similar "word" written on $|\Sigma|$ branches

Repeat these steps until the order drops down to 0,
and solve the diagonal problem for a regular language.

Downward closure

Theorem

Given a scheme S recognizing L, one can compute an NFA A recognizing $L\downarrow$.

Motivation?

0. It gives a simple abstraction of the language recognized by a scheme.

Downward closure

Theorem

Given a scheme S recognizing L, one can compute an NFA A recognizing $L\downarrow$.

Motivation?

0. It gives a simple abstraction of the language recognized by a scheme.
1. It is undecidable whether $L=A^*$, $L_1=L_2$, etc.

   But we can check this approximately, by checking whether $L\downarrow=A^*$, $L_1\downarrow=L_2\downarrow$, etc.

Downward closure

Theorem

Given a scheme S recognizing L, one can compute an NFA A recognizing $L{\downarrow}$.

Motivation?

0. It gives a simple abstraction of the language recognized by a scheme.
1. It is undecidable whether $L=A^*$, $L_1=L_2$, etc.

   But we can check this approximately, by checking whether $L{\downarrow}=A^*$, $L_1{\downarrow}=L_2{\downarrow}$, etc.
2. The problem "is there a piecewise testable language (i.e., boolean combination of downward closed languages) containing L_1 and not intersecting with L_2" reduces to the diagonal problem [Czerwiński, Martens, van Rooijen, Zeitoun 2015]. This gives a more refined approximation for disjointness of L_1 and L_2 than the test $L_1{\downarrow}\cap L_2{\downarrow}=\emptyset$.

Downward closure

Theorem

Given a scheme S recognizing L, one can compute an NFA A recognizing $L\downarrow$.

Motivation?

0. It gives a simple abstraction of the language recognized by a scheme.
1. It is undecidable whether $L=A^*$, $L_1=L_2$, etc.

   But we can check this approximately, by checking whether $L\downarrow=A^*$, $L_1\downarrow=L_2\downarrow$, etc.
2. The problem "is there a piecewise testable language (i.e., boolean combination of downward closed languages) containing L_1 and not intersecting with L_2" reduces to the diagonal problem [Czerwiński, Martens, van Rooijen, Zeitoun 2015]. This gives a more refined approximation for disjointness of L_1 and L_2 than the test $L_1\downarrow\cap L_2\downarrow=\emptyset$.
3. ~~Consider a system with one leader and some (unspecified) number of contributors, that communicate via common register (read or write, without any locks). The reachability problem in such system reduces to computation of the downward closure [La Torre, Muscholl, Walukiewicz 2015].~~ *(Yesterday's talk – downward closure no longer needed)*

Downward closure

Theorem

Given a scheme S recognizing L, one can compute an NFA A recognizing $L\downarrow$.

Complexity?

Downward closure

Theorem

Given a scheme S recognizing L, one can compute an NFA A recognizing $L{\downarrow}$.

Complexity?

- The diagonal problem ⇒ probably (n-1)-EXPTIME for schemes of order n (ongoing work)

Downward closure

Theorem

Given a scheme S recognizing L, one can compute an NFA A recognizing $L\downarrow$.

Complexity?

- The diagonal problem ⇒ probably (n-1)-EXPTIME for schemes of order n (ongoing work)
- Computation of downward closure ⇒ open problem
  We need to bound the maximal size of the downward closure (a pumping lemma is needed).
- Lower bound: checking whether $L_1\downarrow = L_2\downarrow$ or $L_1\downarrow \subseteq L_2\downarrow$ is co-n-NEXPTIME-hard [Zetzsche 2016]

Downward closure

Theorem

Given a scheme S recognizing L, one can compute an NFA A recognizing $L{\downarrow}$.

Complexity?

- The diagonal problem ⇒ probably (n-1)-EXPTIME for schemes of order n (ongoing work)
- Computation of downward closure ⇒ open problem
  We need to bound the maximal size of the downward closure (a pumping lemma is needed).
- Lower bound: checking whether $L_1{\downarrow}=L_2{\downarrow}$ or $L_1{\downarrow}\subseteq L_2{\downarrow}$ is co-n-NEXPTIME-hard [Zetzsche 2016]

Another open problem: computation of downward closure for schemes recognizing languages of trees.

(By Kruskal's tree theorem the downward closure of any language of trees is a regular language.)

Thank you!