What Are Collapsible Pushdown Automata Good For?

Model-Checking, Logical Reflection & Selection

Olivier Serre

LIAFA (CNRS & Université Paris Diderot – Paris 7)

5th of Décembre 2011

based on joint works with C. Broadbent, A. Carayol, M. Hague, A. Meyer, A. Murawski & L. Ong [LICS08a, LICS08b, LICS10,xxx]

Model-Checking

Input:

- An infinite tree t described by a finite object (recursion scheme or a collapsible pushdown automaton in this talk).
- An MSO / μ-calculus formula φ.
- A node u in t.

Question: Is φ true in t at node u?

Global Model-Checking

Input:

- An infinite tree t described by a finite object (recursion scheme or a collapsible pushdown automaton in this talk).
- An MSO formula $\varphi(x)$ with a first-order free variable.

Question: Provide a description of the set of nodes u in t such that $\varphi[x \leftarrow u]$ is true?

Remark. More general than standard model-checking.

Selection

Input:

- An infinite tree t described by a finite object (recursion scheme or a collapsible pushdown automaton in this talk).
- An MSO formula $\varphi(X)$ with a second-order free variable.

Question: If exists, provide a description of a set U of nodes in t such that $\varphi[X \leftarrow U]$ is true at the root of t?

Remark. More general than global model-checking w.r.t some ψ. Indeed take

$$\varphi(X) = x \in X \Leftrightarrow \psi(x)$$

This talk

We will present two (equi-expressive) ways to define a very rich family of trees for which the model-checking / global model-checking / selection problems are decidable:

- Recursion schemes: rewriting systems for simply typed terms (*i.e.* simply typed λ-calculus with recursion).
- Collapsible pushdown automata: machines with a finite control and using a stack of stacks of stack... as an auxiliary storage.

This talk

We will present two (equi-expressive) ways to define a very rich family of trees for which the model-checking / global model-checking / selection problems are decidable:

- Recursion schemes
- Collapsible pushdown automata

Theorem

Given a recursion scheme $\mathcal{S}$ (generating a tree t) and an MSO formula $\varphi(X)$ with a second-order free variable, if $\exists X\ \varphi(X)$ holds at the root of t then one can construct a scheme $\mathcal{S}_\varphi$ generating a tree t_φ s.t.

- *t and t_φ have the same domain;*
- *$t_\varphi(u) = (t(u), x_u)$ with $x_u \in \{0,1\}$;*
- *$U = \{u \mid x_u = 1\}$ is such that $\varphi[X \leftarrow U]$ is true at the root of t.*

Higher-Order Recursion Schemes

Higher-Order Recursion Schemes: Example 1

$$
\begin{array}{rcl}
I & \to & F\,c \\
F\,x & \to & a\,(F\,(b\,x))\,(b\,x)
\end{array}
$$

- Terminals: a, b, c.
- Non-terminals: I, F.
- Variable: x.

Higher-Order Recursion Schemes: Example 1

$$\begin{array}{rcl} I & \rightarrow & F\,c \\ F\,x & \rightarrow & a\,(F\,(b\,x))\,(b\,x) \end{array}$$

$$I \quad \Rightarrow \quad \begin{array}{c} F \\ | \\ c \end{array}$$

Higher-Order Recursion Schemes: Example 1

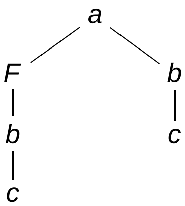
$$
\begin{array}{rcl}
I & \rightarrow & F\,c \\
F\,x & \rightarrow & a\,(F\,(b\,x))\,(b\,x)
\end{array}
$$

$$
\begin{array}{c}
F \\
| \\
c
\end{array}
\qquad \Rightarrow
$$

Higher-Order Recursion Schemes: Example 1

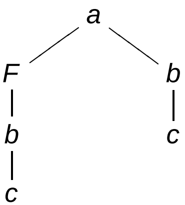
$$
\begin{array}{rcl}
I & \rightarrow & F\,c \\
F\,x & \rightarrow & a\,(F\,(b\,x))\,(b\,x)
\end{array}
$$

$\Rightarrow$



Higher-Order Recursion Schemes: Example 1

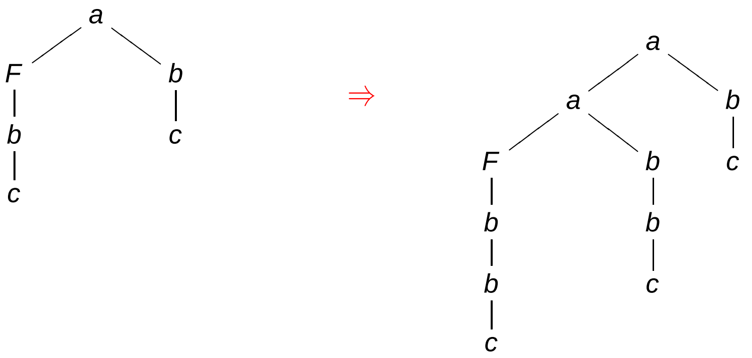
$$\begin{array}{rcl} I & \rightarrow & F\,c \\ F\,x & \rightarrow & a\,(F\,(b\,x))\,(b\,x) \end{array}$$



$\Rightarrow$

Higher-Order Recursion Schemes: Example 1

$$
\begin{aligned}
I &\rightarrow F\,c \\
F\,x &\rightarrow a\,(F\,(b\,x))\,(b\,x)
\end{aligned}
$$

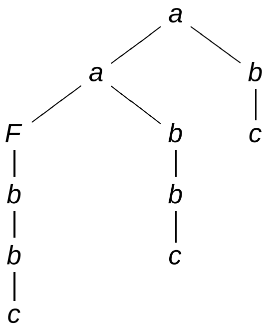


Higher-Order Recursion Schemes: Example 1

$$
\begin{array}{rcl}
I & \rightarrow & F\,c \\
F\,x & \rightarrow & a\,(F\,(b\,x))\,(b\,x)
\end{array}
$$

Applying the rules ad infinitum leads a tree whose branch language is the (context-free) language

$$\{a^n b^n c \mid n \geq 1\}$$



Higher-Order Recursion Schemes: Example 1

[Digression] Simple types

Built from the based (aka ground) type o using the $\rightarrow$ constructor (and $\rightarrow$ associate to the left).
Every type can uniquely be written as $\tau_1 \rightarrow \cdots \rightarrow \tau_n \rightarrow o$.
Examples:

- $\Delta : o \rightarrow o \rightarrow o \rightarrow o$ $\quad \Delta(a, b, c) = b^2 - 4ac$
- $\mathrm{Apply} : (o \rightarrow o) \rightarrow o \rightarrow o$ $\quad Apply(f, x) = f(x)$

Order of a type:
$Ord(o) = 0$ and $Ord(\tau_1 \rightarrow \cdots \rightarrow \tau_n \rightarrow o) = 1 + \max_i\{Ord(\tau_i)\}$.
Examples:

- $Ord(o \rightarrow o) = Ord(o \rightarrow o \rightarrow o \rightarrow o) = 1$
- $Ord((o \rightarrow o) \rightarrow o \rightarrow o) = 2$.

Higher-Order Recursion Schemes: Example 1

$$\begin{array}{rcl} I & \rightarrow & F\,c \\ F\,x & \rightarrow & a\,(F\,(b\,x))\,(b\,x) \end{array}$$

Everything in a scheme is (simply) typed!

- $ar(a) = 2$, $ar(b) = 1$, $ar(c) = 0$
  (*i.e.* $a : o \rightarrow o \rightarrow o$, $b : o \rightarrow o$, $c : o$)
- $I : o, F : o \rightarrow o$
- $x : o$
- Terms appearing on both sides of the rules are (applicative terms) of ground type o.

Order of a scheme = highest order of its non-terminals.

The previous example has order 1.

Formal Definition

Higher-order recursion scheme: $\mathcal{S} = \langle \Sigma, \mathcal{N}, \mathcal{R}, I \rangle$

- Σ: Ranked alphabet of terminals; $\{a : o \to o \to o, b : o \to o, c : o\}$
- $\mathcal{N}$: Set of typed non-terminals; $\{F : o \to o, I : o\}$
- $I \in \mathcal{N}$: Initial symbol of type o;
- $\mathcal{R}$: Set of rewriting rules, one for each non-terminal $F : \tau_1 \to \cdots \to \tau_n \to o$ of the form

$$F\,\xi_1 \cdots \xi_n \to e$$

ξ_i variable of type τ_i, and e is an applicative ground type term over $\Sigma \cup \mathcal{N} \cup \{\xi_1, \cdots, \xi_n\}$.

Example 1

$$\begin{array}{rcl} I & \to & F\,c \\ F\,x & \to & a\,(F\,(b\,x))\,(b\,x) \end{array}$$

Semantics: Associated Tree

Tree associated with $\mathcal{S} = \langle \Sigma, \mathcal{N}, \mathcal{R}, I \rangle$:

- $[|\mathcal{S}|]$: tree obtained by applying the rewriting rules from I *ad infinitum*. Rules are applied following an a Outermost-Innermost (OI) policy.
- More formally: supremum for the prefix ordering of trees derivable from I (which is well defined thanks to Church-Rosser property of schemes).

Semantics: Associated Tree

Tree associated with $\mathcal{S} = \langle \Sigma, \mathcal{N}, \mathcal{R}, I \rangle$:

- $[|\mathcal{S}|]$: tree obtained by applying the rewriting rules from I *ad infinitum*. Rules are applied following an a Outermost-Innermost (OI) policy.
- More formally: supremum for the prefix ordering of trees derivable from I (which is well defined thanks to Church-Rosser property of schemes).

Order of a recursion scheme: highest order of its non-terminals.

Semantics: Associated Tree

Tree associated with $\mathcal{S} = \langle \Sigma, \mathcal{N}, \mathcal{R}, I \rangle$:

- $[|\mathcal{S}|]$: tree obtained by applying the rewriting rules from I *ad infinitum*. Rules are applied following an a Outermost-Innermost (OI) policy.
- More formally: supremum for the prefix ordering of trees derivable from I (which is well defined thanks to Church-Rosser property of schemes).

Order of a recursion scheme: highest order of its non-terminals.

$$RecTrees_n(\Sigma) = \{[|\mathcal{S}|] \mid \mathcal{S} \text{ recursion scheme of ordrer } n\}$$

Higher-Order Recursion Schemes: Example 2

$$\begin{array}{rcl} I & \rightarrow & F\,b\,c \\ F\,\varphi\,\psi & \rightarrow & a\,(F\,(C_p\,b\,\varphi)\,(C_p\,c\,\psi))\,(\varphi\,(\psi\,d)) \\ C_p\,\varphi\,\psi\,x & \rightarrow & \varphi\,(\psi\,x) \end{array}$$

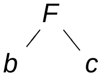
- Terminals:
  - $a : o \rightarrow o \rightarrow o$
  - $b, c : o \rightarrow o$
  - $d : o$
- Non-terminals:
  - $I : o$
  - $F : (o \rightarrow o) \rightarrow (o \rightarrow o) \rightarrow o$
  - $C_p : (o \rightarrow o) \rightarrow (o \rightarrow o) \rightarrow o \rightarrow o$ — Compose and apply
- Variable:
  - $x : o$
  - $\varphi, \psi : o \rightarrow o$

Higher-Order Recursion Schemes: Example 2

$$
\begin{array}{rcl}
I & \rightarrow & F\, b\, c \\
F\, \varphi\, \psi & \rightarrow & a\, (F\, (C_p\, b\, \varphi)\, (C_p\, c\, \psi))\, (\varphi\, (\psi\, d)) \\
C_p\, \varphi\, \psi\, x & \rightarrow & \varphi\, (\psi\, x)
\end{array}
$$

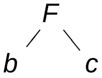
I

$\Rightarrow$



Higher-Order Recursion Schemes: Example 2

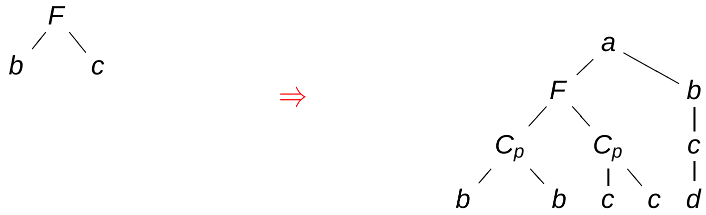
$$
\begin{array}{rcl}
I & \rightarrow & F\, b\, c \\
F\, \varphi\, \psi & \rightarrow & a\,(F\,(C_p\, b\, \varphi)\,(C_p\, c\, \psi))\,(\varphi\,(\psi\, d)) \\
C_p\, \varphi\, \psi\, x & \rightarrow & \varphi\,(\psi\, x)
\end{array}
$$



$$\Rightarrow$$

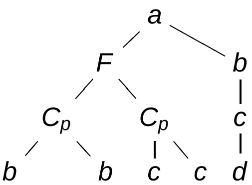
Higher-Order Recursion Schemes: Example 2

$$
\begin{array}{rcl}
I & \to & F\, b\, c \\
F\, \varphi\, \psi & \to & a\, (F\, (C_p\, b\, \varphi)\, (C_p\, c\, \psi))\, (\varphi\, (\psi\, d)) \\
C_p\, \varphi\, \psi\, x & \to & \varphi\, (\psi\, x)
\end{array}
$$



Higher-Order Recursion Schemes: Example 2

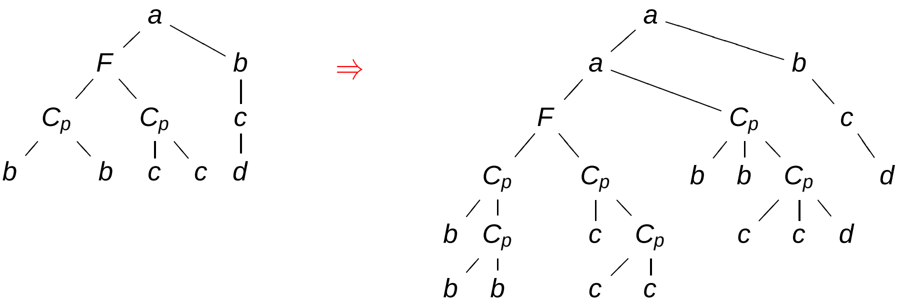
$$
\begin{array}{rcl}
I & \rightarrow & F\, b\, c \\
F\, \varphi\, \psi & \rightarrow & a\, (F\, (C_p\, b\, \varphi)\, (C_p\, c\, \psi))\, (\varphi\, (\psi\, d)) \\
C_p\, \varphi\, \psi\, x & \rightarrow & \varphi\, (\psi\, x)
\end{array}
$$



$\Rightarrow$

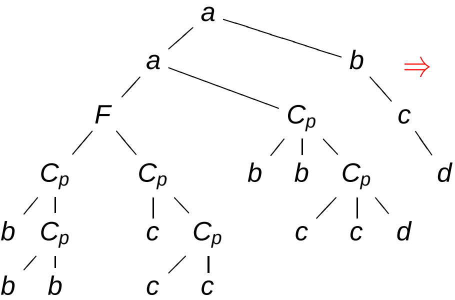
Higher-Order Recursion Schemes: Example 2

$$
\begin{array}{rcl}
I & \rightarrow & F\, b\, c \\
F\, \varphi\, \psi & \rightarrow & a\,(F\,(C_p\, b\, \varphi)\,(C_p\, c\, \psi))\,(\varphi\,(\psi\, d)) \\
C_p\, \varphi\, \psi\, x & \rightarrow & \varphi\,(\psi\, x)
\end{array}
$$



Higher-Order Recursion Schemes: Example 2

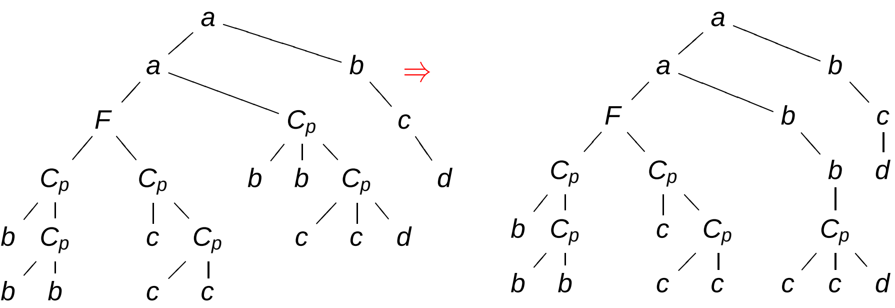
$$
\begin{array}{rcl}
I & \rightarrow & F\, b\, c \\
F\, \varphi\, \psi & \rightarrow & a\, (F\, (C_p\, b\, \varphi)\, (C_p\, c\, \psi))\, (\varphi\, (\psi\, d)) \\
C_p\, \varphi\, \psi\, x & \rightarrow & \varphi\, (\psi\, x)
\end{array}
$$



$\Rightarrow$

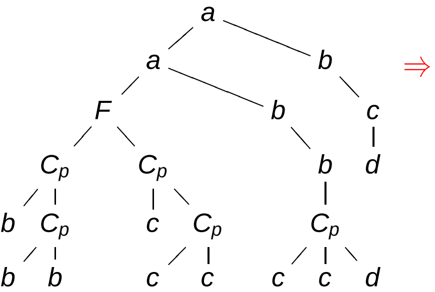
Higher-Order Recursion Schemes: Example 2

$$
\begin{array}{rcl}
I & \to & F\, b\, c \\
F\, \varphi\, \psi & \to & a\, (F\, (C_p\, b\, \varphi)\, (C_p\, c\, \psi))\, (\varphi\, (\psi\, d)) \\
C_p\, \varphi\, \psi\, x & \to & \varphi\, (\psi\, x)
\end{array}
$$



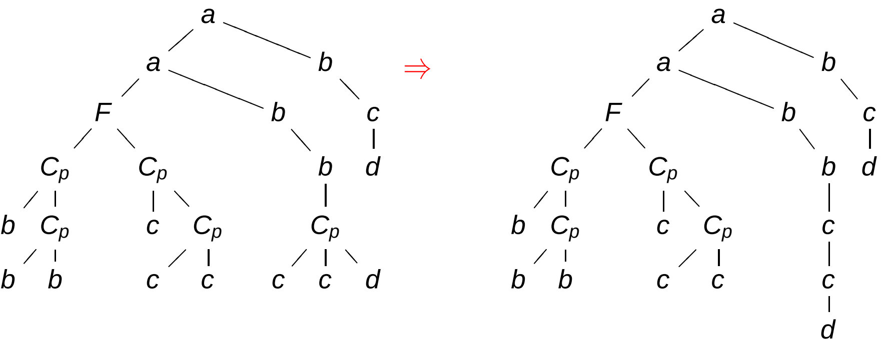
Higher-Order Recursion Schemes: Example 2

$$
\begin{array}{rcl}
I & \to & F\, b\, c \\
F\, \varphi\, \psi & \to & a\,(F\,(C_p\, b\, \varphi)\,(C_p\, c\, \psi))\,(\varphi\,(\psi\, d)) \\
C_p\, \varphi\, \psi\, x & \to & \varphi\,(\psi\, x)
\end{array}
$$



Higher-Order Recursion Schemes: Example 2

$$
\begin{array}{rcl}
I & \to & F\, b\, c \\
F\, \varphi\, \psi & \to & a\,(F\,(C_p\, b\, \varphi)\,(C_p\, c\, \psi))\,(\varphi\,(\psi\, d)) \\
C_p\, \varphi\, \psi\, x & \to & \varphi\,(\psi\, x)
\end{array}
$$

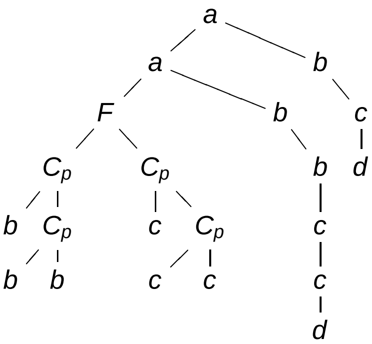


Higher-Order Recursion Schemes: Example 2

$$
\begin{array}{rcl}
I & \rightarrow & F\, b\, c \\
F\, \varphi\, \psi & \rightarrow & a\,(F\,(C_p\, b\, \varphi)\,(C_p\, c\, \psi))\,(\varphi\,(\psi\, d)) \\
C_p\, \varphi\, \psi\, x & \rightarrow & \varphi\,(\psi\, x)
\end{array}
$$

Applying the rules ad infinitum leads a tree whose branch language is the (indexed) language

$$\{a^n b^n c^n d \mid n \geq 1\}$$



Higher-Order Recursion Schemes: Example 3

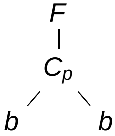
$$\begin{array}{rcl} I & \to & F\,(C_p\, b\, b) \\ F\,\varphi & \to & a\,(F\,(C_p\,\varphi\,\varphi))\,(\varphi\, d) \\ C_p\,\varphi\,\psi\, x & \to & \varphi\,(\psi\, x) \end{array}$$

- Terminals:
  - $a : o \to o \to o$
  - $b : o \to o$
  - $d : o$
- Non-terminals:
  - $I : o$
  - $F : (o \to o) \to o$
  - $C_p : (o \to o) \to (o \to o) \to o \to o$ Compose and apply
- Variable:
  - $x : o$
  - $\varphi, \psi : o \to o$

Higher-Order Recursion Schemes: Example 3

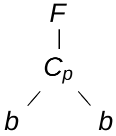
$$
\begin{array}{rcl}
I & \rightarrow & F\,(C_p\,b\,b) \\
F\,\varphi & \rightarrow & a\,(F\,(C_p\,\varphi\,\varphi))\,(\varphi\,d) \\
C_p\,\varphi\,\psi\,x & \rightarrow & \varphi\,(\psi\,x)
\end{array}
$$

$I \quad \Rightarrow$



Higher-Order Recursion Schemes: Example 3

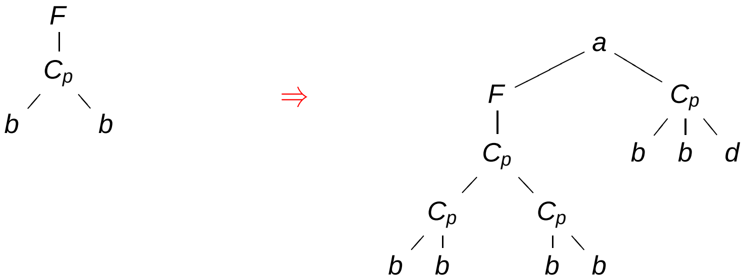
$$
\begin{array}{rcl}
I & \rightarrow & F\,(C_p\,b\,b) \\
F\,\varphi & \rightarrow & a\,(F\,(C_p\,\varphi\,\varphi))\,(\varphi\,d) \\
C_p\,\varphi\,\psi\,x & \rightarrow & \varphi\,(\psi\,x)
\end{array}
$$



$\Rightarrow$

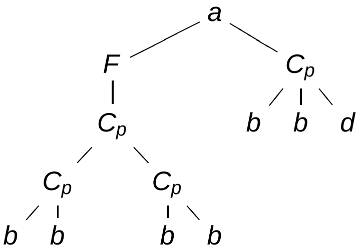
Higher-Order Recursion Schemes: Example 3

$$
\begin{array}{rcl}
I & \rightarrow & F\,(C_p\, b\, b) \\
F\,\varphi & \rightarrow & a\,(F\,(C_p\,\varphi\,\varphi))\,(\varphi\, d) \\
C_p\,\varphi\,\psi\, x & \rightarrow & \varphi\,(\psi\, x)
\end{array}
$$



Higher-Order Recursion Schemes: Example 3

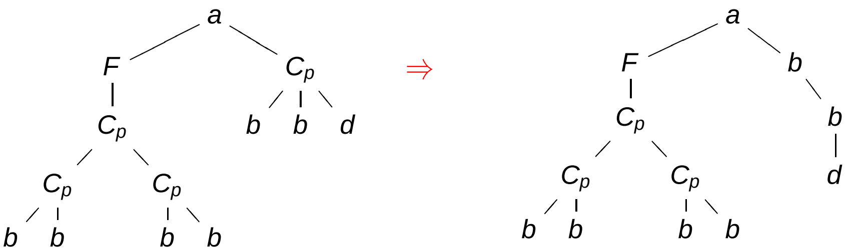
$$
\begin{array}{rcl}
I & \rightarrow & F\,(C_p\,b\,b) \\
F\,\varphi & \rightarrow & a\,(F\,(C_p\,\varphi\,\varphi))\,(\varphi\,d) \\
C_p\,\varphi\,\psi\,x & \rightarrow & \varphi\,(\psi\,x)
\end{array}
$$



$\Rightarrow$

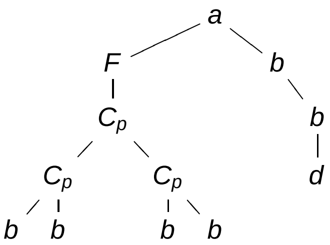
Higher-Order Recursion Schemes: Example 3

$$
\begin{array}{rcl}
I & \rightarrow & F\,(C_p\,b\,b) \\
F\,\varphi & \rightarrow & a\,(F\,(C_p\,\varphi\,\varphi))\,(\varphi\,d) \\
C_p\,\varphi\,\psi\,x & \rightarrow & \varphi\,(\psi\,x)
\end{array}
$$



Higher-Order Recursion Schemes: Example 3

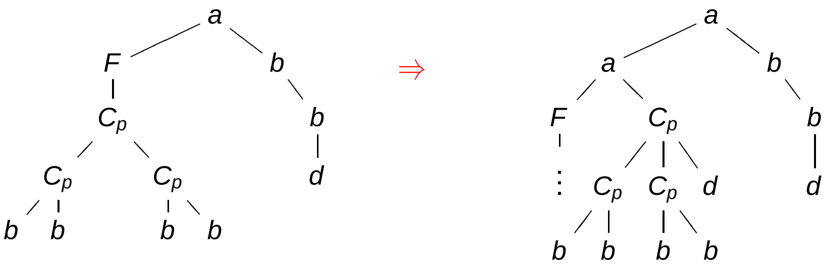
$$
\begin{array}{rcl}
I & \rightarrow & F\,(C_p\,b\,b) \\
F\,\varphi & \rightarrow & a\,(F\,(C_p\,\varphi\,\varphi))\,(\varphi\,d) \\
C_p\,\varphi\,\psi\,x & \rightarrow & \varphi\,(\psi\,x)
\end{array}
$$



$\Rightarrow$

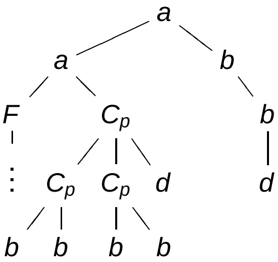
Higher-Order Recursion Schemes: Example 3

$$
\begin{array}{rcl}
I & \rightarrow & F\,(C_p\, b\, b) \\
F\,\varphi & \rightarrow & a\,(F\,(C_p\,\varphi\,\varphi))\,(\varphi\, d) \\
C_p\,\varphi\,\psi\, x & \rightarrow & \varphi\,(\psi\, x)
\end{array}
$$



Higher-Order Recursion Schemes: Example 3

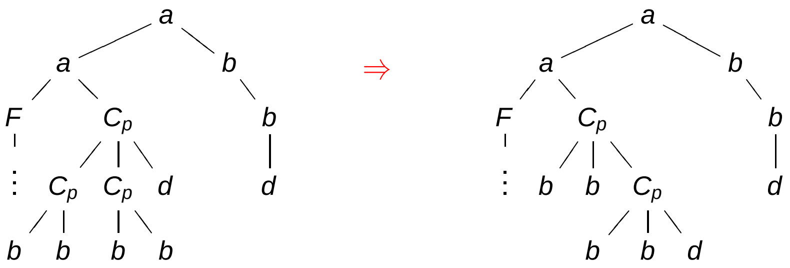
$$
\begin{aligned}
I &\rightarrow F\,(C_p\, b\, b) \\
F\,\varphi &\rightarrow a\,(F\,(C_p\,\varphi\,\varphi))\,(\varphi\, d) \\
C_p\,\varphi\,\psi\, x &\rightarrow \varphi\,(\psi\, x)
\end{aligned}
$$



$\Rightarrow$

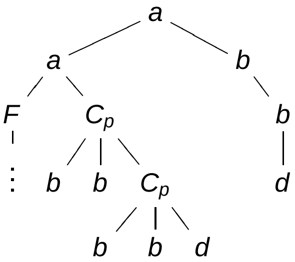
Higher-Order Recursion Schemes: Example 3

$$\begin{aligned} I &\rightarrow F\,(C_p\, b\, b) \\ F\,\varphi &\rightarrow a\,(F\,(C_p\,\varphi\,\varphi))\,(\varphi\, d) \\ C_p\,\varphi\,\psi\, x &\rightarrow \varphi\,(\psi\, x) \end{aligned}$$



Higher-Order Recursion Schemes: Example 3

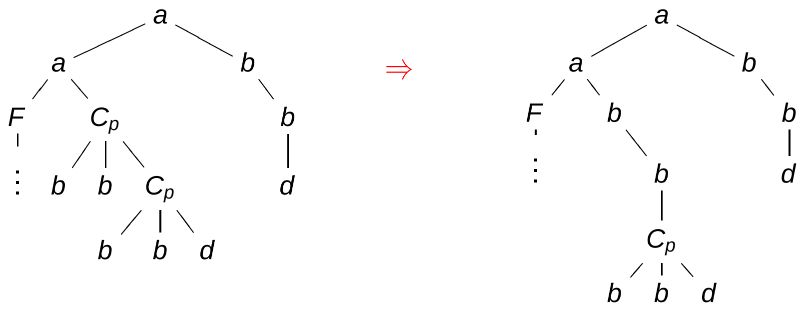
$$
\begin{array}{rcl}
I & \rightarrow & F\,(C_p\,b\,b) \\
F\,\varphi & \rightarrow & a\,(F\,(C_p\,\varphi\,\varphi))\,(\varphi\,d) \\
C_p\,\varphi\,\psi\,x & \rightarrow & \varphi\,(\psi\,x)
\end{array}
$$



$\Rightarrow$

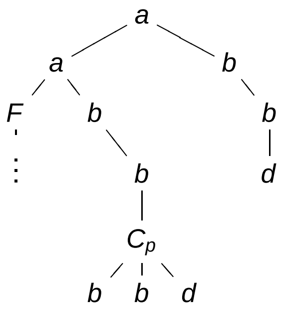
Higher-Order Recursion Schemes: Example 3

$$
\begin{array}{rcl}
I & \rightarrow & F\,(C_p\,b\,b) \\
F\,\varphi & \rightarrow & a\,(F\,(C_p\,\varphi\,\varphi))\,(\varphi\,d) \\
C_p\,\varphi\,\psi\,x & \rightarrow & \varphi\,(\psi\,x)
\end{array}
$$



Higher-Order Recursion Schemes: Example 3

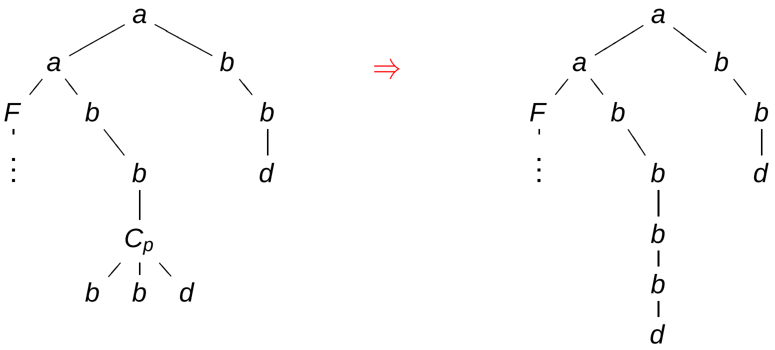
$$
\begin{array}{rcl}
I & \rightarrow & F\,(C_p\,b\,b) \\
F\,\varphi & \rightarrow & a\,(F\,(C_p\,\varphi\,\varphi))\,(\varphi\,d) \\
C_p\,\varphi\,\psi\,x & \rightarrow & \varphi\,(\psi\,x)
\end{array}
$$



$\Rightarrow$

Higher-Order Recursion Schemes: Example 3

$$\begin{aligned} I &\rightarrow F\,(C_p\,b\,b) \\ F\,\varphi &\rightarrow a\,(F\,(C_p\,\varphi\,\varphi))\,(\varphi\,d) \\ C_p\,\varphi\,\psi\,x &\rightarrow \varphi\,(\psi\,x) \end{aligned}$$

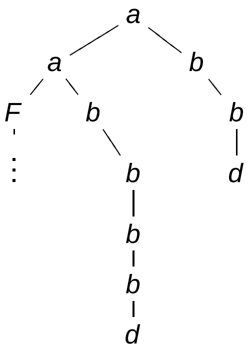


Higher-Order Recursion Schemes: Example 3

$$
\begin{aligned}
I &\rightarrow F\,(C_p\, b\, b) \\
F\,\varphi &\rightarrow a\,(F\,(C_p\,\varphi\,\varphi))\,(\varphi\, d) \\
C_p\,\varphi\,\psi\, x &\rightarrow \varphi\,(\psi\, x)
\end{aligned}
$$

Applying the rules ad infinitum leads a tree whose branch language is the language

$$\{a^n b^{2^n} d \mid n \geq 1\}$$



The Safety Constraint [Damm'82,KNU'01]

Quoting [KNU'02]:

> *A term of order k is unsafe if it contains an occurrence of a variable of order $< k$, otherwise the term is safe. An occurrence of an unsafe term t as a subexpression of a term t' is safe if it is in the context $\ldots(ts)\ldots$, otherwise the occurrence is unsafe. A scheme is safe if no unsafe term has an unsafe occurrence at a right-hand side of any rewriting rule.*

$$RecSafeTrees_n(\Sigma) = \{[\![G]\!] \mid S \textbf{ safe} \text{ recursion scheme of order } n\}$$

Related works:

- Derived type constraint [Damm'82].
- Safe-λ-calculus [AdMO'04,BO'07]: fragment of the λ-calculus in which there is no variable captured by some external λ when performing a substitution (hence no need for α-conversion when performing some β-reduction).

Some results

- [Damm'82] *level-n* tree grammars ($\approx$ safe grammars).
- [Greibach'69,Maslov'74] higher-order pushdown automata / generalized indexed languages.
- [Damm&Goerdt'86] OI-languages $\equiv$ higher-order pushdown automata
- [Knapik,Niwiński,Urzyczyn'01&'02] *RecSafeTrees* $\equiv$ languages accepted by hopda ($\Rightarrow$ internal repr.) + MSO decidability.
- [Caucal'02] transformational representation + MSO decidability.
- [Knapik,Niwiński,Urzyczyn,Walukiewicz'05] [Aehlig, de Miranda,Ong'05] MSO decidability for $\mathit{RecTrees}_2(\Sigma)$ + internal repr.
- [Ong'06] MSO decidability for $\mathit{RecTrees}(\Sigma)$.
- [Hague, Murawski, Ong, Serre'08] Internal repr. for $\mathit{RecTrees}(\Sigma)$, parity games + corollaries (in particular [Ong'06]).
- [Carayol, Hague, Meyer, Ong, Serre'08] Global Model-checking for *RecSafeTrees*.
- [Broadbent, Carayol, Ong, Serre'10] Global Model-checking *RecTrees*.

Collapsible Pushdown Automata

Pushdown Automata

Pushdown automaton: finite control + stack.

$\mathcal{A} = (Q, A \cup \{\varepsilon\}, \Gamma, q_0, \Delta)$

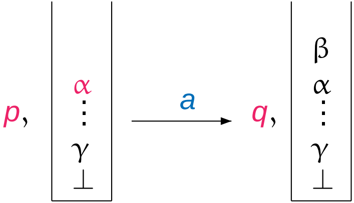
$$p, \quad \begin{array}{|c|} \alpha \\ \vdots \\ \gamma \\ \perp \\ \hline \end{array}$$

$$\Delta : Q \times \Gamma \times (A \cup \{\varepsilon\}) \longrightarrow 2^{Q \times \{rew(\alpha), pop, push(\alpha) \mid \alpha \in \Gamma\}}$$

Pushdown automaton: finite control + stack.

$\mathcal{A} = (Q, A \cup \{\varepsilon\}, \Gamma, q_0, \Delta)$

$(q, push(\beta))$:

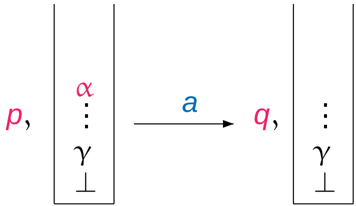


$$\Delta : Q \times \Gamma \times (A \cup \{\varepsilon\}) \longrightarrow 2^{Q \times \{rew(\alpha), pop, push(\alpha) | \alpha \in \Gamma\}}$$

Pushdown automaton: finite control + stack.

$\mathcal{A} = (Q, A \cup \{\varepsilon\}, \Gamma, q_0, \Delta)$

(q, pop):



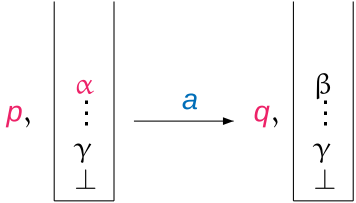
$$\Delta : Q \times \Gamma \times (A \cup \{\varepsilon\}) \rightarrow 2^{Q \times \{rew(\alpha), pop, push(\alpha) \mid \alpha \in \Gamma\}}$$

Pushdown Automata

Pushdown automaton: finite control + stack.

$\mathcal{A} = (Q, A \cup \{\varepsilon\}, \Gamma, q_0, \Delta)$

$(q, rew(\beta))$:



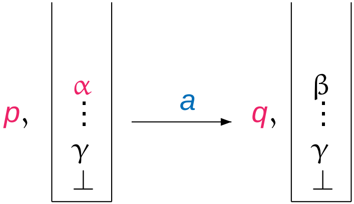
$$\Delta : Q \times \Gamma \times (A \cup \{\varepsilon\}) \rightarrow 2^{Q \times \{rew(\alpha), pop, push(\alpha) \mid \alpha \in \Gamma\}}$$

Pushdown Automata

Pushdown automaton: finite control + stack.

$\mathcal{A} = (Q, A \cup \{\varepsilon\}, \Gamma, q_0, \Delta)$

$(q, rew(\beta))$:



$$\Delta : Q \times \Gamma \times (A \cup \{\varepsilon\}) \rightarrow 2^{Q \times \{rew(\alpha), pop, push(\alpha) \mid \alpha \in \Gamma\}}$$

$\mathcal{A}$ is deterministic iff

Δ takes value in singletons + empty set.

ε moves are possible only if no letter can be read.

Transition Graph & Tree Generated. Back to Ex. 1

$\mathcal{A}$	a	b	c
$q_0, \_$	$q_0, push(\alpha)$	q_b, pop	$\_$
q_b, α	$\_$	q_b, pop	$\_$
$q_b, \bot$	$\_$	$\_$	$q_c, rew(\bot)$

Transition graph:

$$\begin{tikzcd}
(q_0, \bot) \arrow[r, "a"] & (q_0, \alpha\bot) \arrow[r, "a"] \arrow[d, "b"'] & (q_0, \alpha\alpha\bot) \arrow[r, "a"] \arrow[d, "b"'] & (q_0, \alpha\alpha\alpha\bot) \arrow[r, "a"] \arrow[d, "b"'] & \cdots \\
(q_c, \bot) & (q_b, \bot) \arrow[l, "c"'] & (q_b, \alpha\bot) \arrow[l, "b"'] & (q_b, \alpha\alpha\bot) \arrow[l, "b"'] & \cdots \arrow[l, "b"']
\end{tikzcd}$$

Transition Graph & Tree Generated. Back to Ex. 1

$$
\begin{tikzcd}
(q_0, \perp) \arrow[r, "a"] & (q_0, \alpha\perp) \arrow[r, "a"] \arrow[d, "b"'] & (q_0, \alpha\alpha\perp) \arrow[r, "a"] \arrow[d, "b"'] & (q_0, \alpha\alpha\alpha\perp) \arrow[r, "a"] \arrow[d, "b"'] & \cdots \\
(q_c, \perp) & (q_b, \perp) \arrow[l, "c"'] & (q_b, \alpha\perp) \arrow[l, "b"'] & (q_b, \alpha\alpha\perp) \arrow[l, "b"'] & \cdots \arrow[l, "b"']
\end{tikzcd}
$$

Unfolding:

$$
\begin{tikzcd}
(q_0, \perp) \arrow[r, "a"] & (q_0, \alpha\perp) \arrow[r, "a"] \arrow[d, "b"'] & (q_0, \alpha\alpha\perp) \arrow[r, "a"] \arrow[d, "b"'] & (q_0, \alpha\alpha\alpha\perp) \arrow[r, "a"] \arrow[d, "b"'] & \cdots \\
 & (q_b, \perp) \arrow[d, "c"'] & (q_b, \alpha\perp) \arrow[d, "b"'] & (q_b, \alpha\alpha\perp) \arrow[d, "b"'] & \\
 & (q_c, \perp) & (q_b, \perp) \arrow[d, "c"'] & (q_b, \alpha\perp) \arrow[d, "b"'] & \\
 & & (q_c, \perp) & (q_b, \alpha\perp) \arrow[d, "c"'] & \\
 & & & (q_c, \perp) &
\end{tikzcd}
$$

Transition Graph & Tree Generated. Back to Ex. 1

$$\begin{tikzcd}
(q_0,\perp) \arrow[r, "a"] & (q_0,\alpha\perp) \arrow[r, "a"] \arrow[d, "b"'] & (q_0,\alpha\alpha\perp) \arrow[r, "a"] \arrow[d, "b"'] & (q_0,\alpha\alpha\alpha\perp) \arrow[r, "a"] \arrow[d, "b"'] & \cdots \\
(q_c,\perp) & (q_b,\perp) \arrow[l, "c"'] & (q_b,\alpha\perp) \arrow[l, "b"'] & (q_b,\alpha\alpha\perp) \arrow[l, "b"'] & \cdots \arrow[l, "b"']
\end{tikzcd}$$

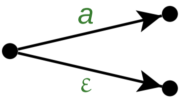
Unfolding + Labelling:

$$\begin{tikzcd}
r \arrow[r, "0"] & a \arrow[r, "0"] \arrow[d, "1"'] & a \arrow[r, "0"] \arrow[d, "1"'] & a \arrow[r, "0"] \arrow[d, "1"'] & \cdots \\
 & b \arrow[d, "0"'] & b \arrow[d, "0"'] & b \arrow[d, "0"'] & \\
 & c & b \arrow[d, "0"'] & b \arrow[d, "0"'] & \\
 & & c & b \arrow[d, "0"'] & \\
 & & & c &
\end{tikzcd}$$

Tree Associated with a Deterministic Pushdown Automaton

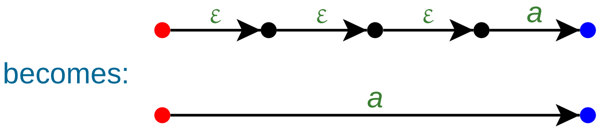
- Fix a deterministic pushdown $\mathcal{A} = \langle Q, A \cup \{\varepsilon\}, \Gamma, \delta, q_0, F \rangle$

  Forbidden pattern:



Tree Associated with a Deterministic Pushdown Automaton

- Fix a deterministic pushdown $\mathcal{A} = \langle Q, A \cup \{\varepsilon\}, \Gamma, \delta, q_0, F \rangle$
- Let $G(\mathcal{A})$ be its configuration graph and $G_\varepsilon(\mathcal{A})$ its ε-closure.



Tree Associated with a Deterministic Pushdown Automaton

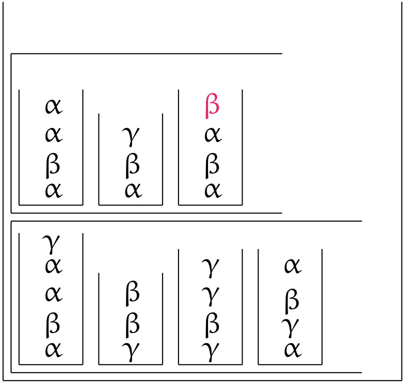
- Fix a deterministic pushdown $\mathcal{A} = \langle Q, A \cup \{\varepsilon\}, \Gamma, \delta, q_0, F \rangle$
- Let $G(\mathcal{A})$ be its configuration graph and $G_\varepsilon(\mathcal{A})$ its ε-closure.
- Unfold $G_\varepsilon(\mathcal{A})$ from the initial configuration.

Tree Associated with a Deterministic Pushdown Automaton

- Fix a deterministic pushdown $\mathcal{A} = \langle Q, A \cup \{\varepsilon\}, \Gamma, \delta, q_0, F \rangle$
- Let $G(\mathcal{A})$ be its configuration graph and $G_\varepsilon(\mathcal{A})$ its ε-closure.
- Unfold $G_\varepsilon(\mathcal{A})$ from the initial configuration.
- Pick some $\tau : Q \times \Gamma \to \Sigma$ and label the vertices of the unfolding using τ.
- Use the edge labels to define directions.

Order-*n* pushdown automaton: finite control + order-*n* stack.

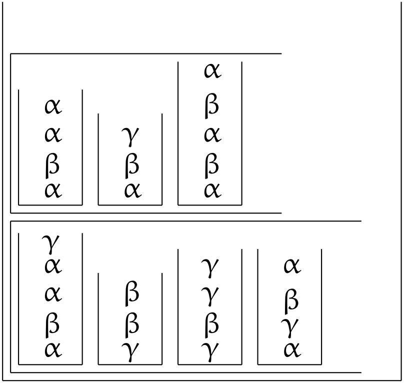
q,



Order-*n* pushdown automaton: finite control + order-*n* stack.

$push_1(\alpha)$:

q,

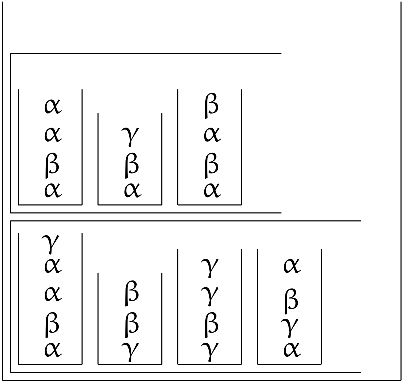


Higher-Order Pushdown Automata

Order-*n* pushdown automaton: finite control + order-*n* stack.

pop_1:

q,

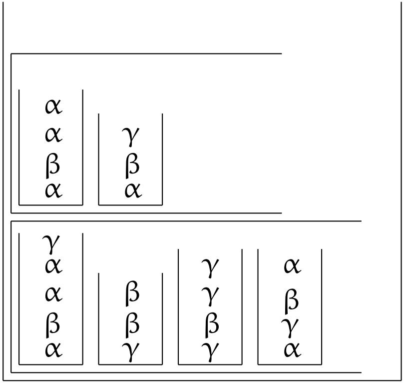


Higher-Order Pushdown Automata

Order-*n* pushdown automaton: finite control + order-*n* stack.

pop$_2$:

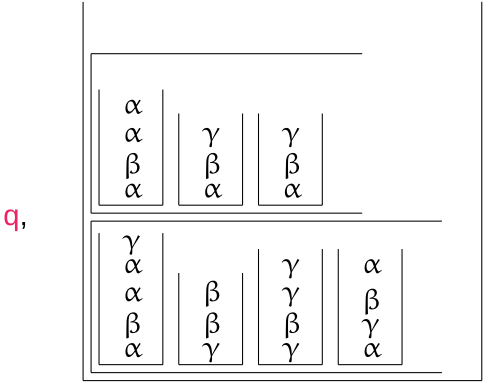
q,



Higher-Order Pushdown Automata

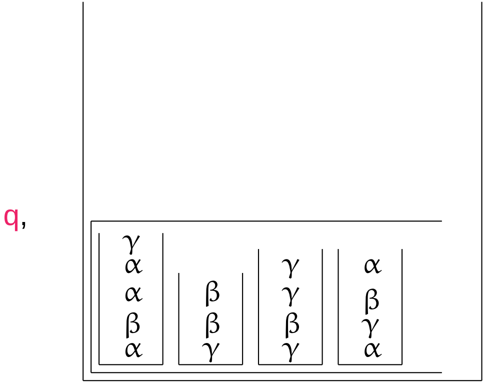
Order-n pushdown automaton: finite control + order-n stack.

push$_2$:



Order-n pushdown automaton: finite control + order-n stack.

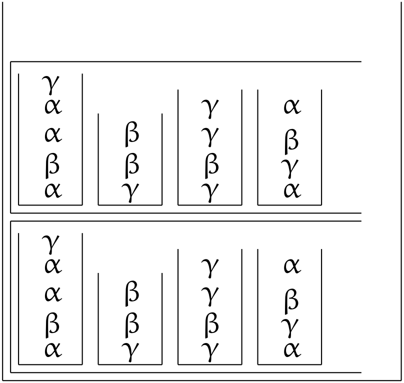
pop_3:



Order-*n* pushdown automaton: finite control + order-*n* stack.

push$_3$:

q,



Higher-order pushdown automata. Back to Ex. 2

$a : push_1(\alpha)$, $b : push_2; pop_1/pop_1$, $c : pop_2; pop_1/pop_1$

$$
\begin{tikzcd}
q_0, \big[\ [\bot]\ \big] \arrow[r, "a"] & q_a, \big[\ [\alpha\bot]\ \big] \arrow[r, "a"] \arrow[d, "b"] & q_a, \big[\ [\alpha\alpha\bot]\ \big] \arrow[r, "a"] \arrow[d, "b"] & q_a, \big[\ [\alpha\alpha\alpha\bot]\ \big] \arrow[r] \arrow[d, "b"] & {} \\
 & q_b, \begin{bmatrix} [\bot] \\ [\alpha\bot] \end{bmatrix} \arrow[d, "c"] & q_b, \begin{bmatrix} [\alpha\bot] \\ [\alpha\alpha\bot] \end{bmatrix} \arrow[d, "b"] & q_b, \begin{bmatrix} [\alpha\alpha\bot] \\ [\alpha\alpha\alpha\bot] \end{bmatrix} \arrow[d, "b"] & \\
q_d, \big[\ [\bot]\ \big] & q_c, \big[\ [\bot]\ \big] \arrow[l, "d"'] & q_b, \begin{bmatrix} [\bot] \\ [\alpha\alpha\bot] \end{bmatrix} \arrow[d, "c"] & q_b, \begin{bmatrix} [\alpha\bot] \\ [\alpha\alpha\alpha\bot] \end{bmatrix} \arrow[d, "b"] & \\
 & & q_c, \big[\ [\alpha\bot]\ \big] \arrow[ul, "c"] & q_b, \begin{bmatrix} [\bot] \\ [\alpha\alpha\alpha\bot] \end{bmatrix} \arrow[d, "c"] & \\
 & & & q_c, \big[\ [\alpha\alpha\bot]\ \big] \arrow[ul, "c"] &
\end{tikzcd}
$$

$$
\begin{tikzcd}
q_0, \left[\ [\bot]\ \right] \arrow[r, "a"] & q_a, \left[\ [\alpha\bot]\ \right] \arrow[r, "a"] \arrow[d, "b"] & q_a, \left[\ [\alpha\alpha\bot]\ \right] \arrow[r, "a"] \arrow[d, "b"] & q_a, \left[\ [\alpha\alpha\alpha\bot]\ \right] \arrow[r] \arrow[d, "b"] & {} \\
& q_b, \begin{bmatrix} [\bot] \\ [\alpha\bot] \end{bmatrix} \arrow[d, "c"] & q_b, \begin{bmatrix} [\alpha\bot] \\ [\alpha\alpha\bot] \end{bmatrix} \arrow[d, "b"] & q_b, \begin{bmatrix} [\alpha\alpha\bot] \\ [\alpha\alpha\alpha\bot] \end{bmatrix} \arrow[d, "b"] & \\
q_d, \left[\ [\bot]\ \right] & q_c, \left[\ [\bot]\ \right] \arrow[l, "d"] & q_b, \begin{bmatrix} [\bot] \\ [\alpha\alpha\bot] \end{bmatrix} \arrow[d, "c"] & q_b, \begin{bmatrix} [\alpha\bot] \\ [\alpha\alpha\alpha\bot] \end{bmatrix} \arrow[d, "b"] & \\
& & q_c, \left[\ [\alpha\bot]\ \right] \arrow[ul, "c"] & q_b, \begin{bmatrix} [\bot] \\ [\alpha\alpha\alpha\bot] \end{bmatrix} \arrow[d, "c"] & \\
& & & q_c, \left[\ [\alpha\alpha\bot]\ \right] \arrow[ul, "c"] &
\end{tikzcd}
$$

Unfolding + ad-hoc labeling: one gets the tree of Example 2.

Can You Get Example 3?

Follows the idea in [Woehrle'05]

$$q_0, [[\varepsilon]] \overset{aaa}{\rightsquigarrow} q_1, [[000]] \overset{\varepsilon}{\rightsquigarrow} q_2 \begin{bmatrix} [0] \\ [00] \\ [000] \end{bmatrix}$$

$$\overset{b}{\rightsquigarrow} q_2 \begin{bmatrix} [1] \\ [00] \\ [000] \end{bmatrix} \overset{b}{\rightsquigarrow} q_2 \begin{bmatrix} [0] \\ [10] \\ [000] \end{bmatrix} \overset{b}{\rightsquigarrow} q_2 \begin{bmatrix} [1] \\ [10] \\ [000] \end{bmatrix}$$

$$\overset{b}{\rightsquigarrow} q_2 \begin{bmatrix} [0] \\ [00] \\ [100] \end{bmatrix} \overset{b^4}{\rightsquigarrow} q_2 \begin{bmatrix} [1] \\ [10] \\ [100] \end{bmatrix} \overset{\varepsilon}{\rightsquigarrow} q_f, [[\varepsilon]]$$

Do we need more?

Theorem (Equi-expressivity, KNU'02 Caucal'03)

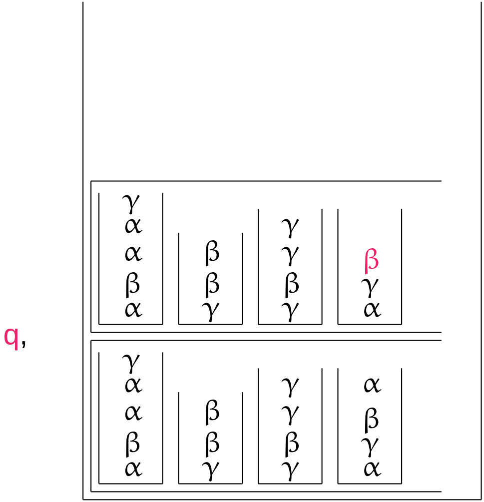
- ⋆ *Let $\mathcal{S}$ be an order-n* safe *recursion scheme over Σ and let t be its value tree. Then there is an order-n PDA $\mathcal{A} = \langle Q, A \cup \{\varepsilon\}, \Gamma, \delta, q_0, F \rangle$, and $\tau : Q \times \Gamma \to \Sigma$ such that t is the tree generated by $\mathcal{A}$ and τ.*
- ⋆ *Let $\mathcal{A} = \langle Q, A \cup \{\varepsilon\}, \Gamma, \delta, q_0, F \rangle$ be an order-n PDA, and let t be the Σ-labelled tree generated by $\mathcal{A}$ and a given map $\tau : Q \times \Gamma \to \Sigma$. Then there is an order-n* safe *recursion scheme over Σ whose value tree is t.*

Moreover, the transformations from scheme to PDA and vice versa are computable in polynomial time.

Something more is needed for the unsafe case

Collapsible Pushdown Automata

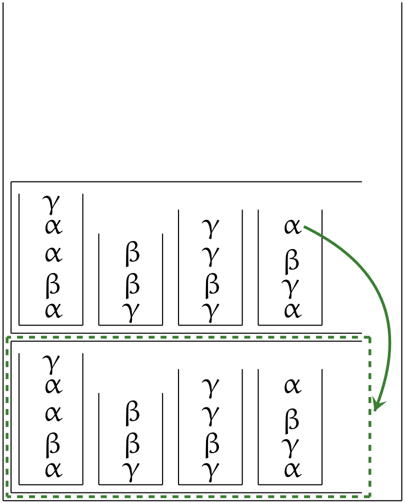
Collapsible Pushdown Automaton: finite control + stack with links



Collapsible Pushdown Automaton: finite control + stack with links

$push_1^3(\alpha)$:

q,

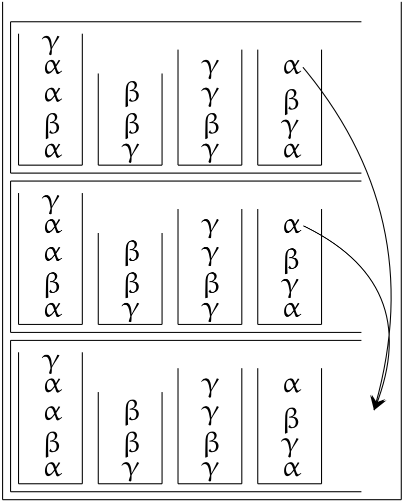


Collapsible Pushdown Automata

Collapsible Pushdown Automaton: finite control + stack with links

push$_3$:

q,

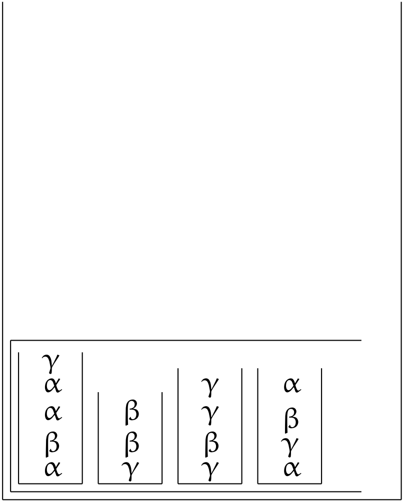


Collapsible Pushdown Automata

Collapsible Pushdown Automaton: finite control + stack with links

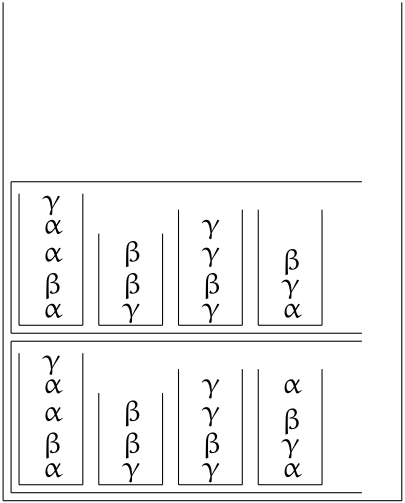
collapse:

q,



Collapsible Pushdown Automaton: finite control + stack with links

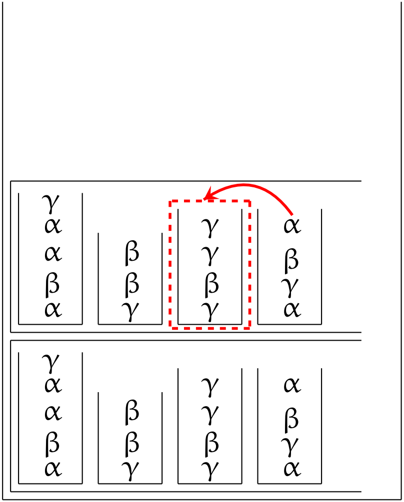
q,



Collapsible Pushdown Automaton: finite control + stack with links

$push_1^2(\alpha)$:

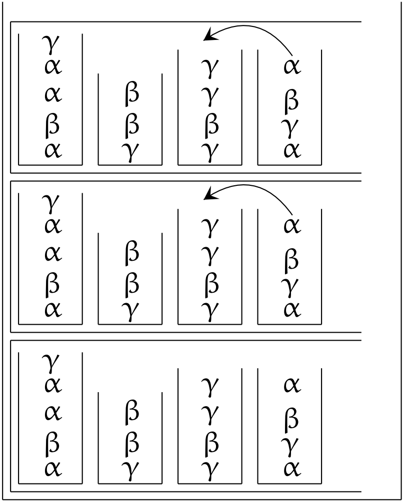
q,



Collapsible Pushdown Automaton: finite control + stack with links

push$_3$:

q,

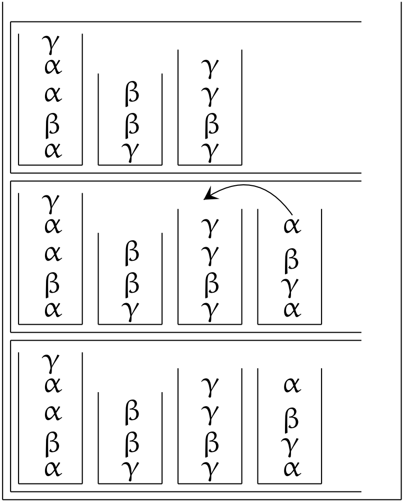


Collapsible Pushdown Automata

Collapsible Pushdown Automaton: finite control + stack with links

collapse:

q,



Equi-Expressivity Theorem

Theorem (Equi-expressivity, HMOS'08, CS)

- ⋆ *Let $\mathcal{S}$ be an order-n recursion scheme over Σ and let t be its value tree. Then there is an order-n CPDA $\mathcal{A} = \langle Q, A \cup \{\varepsilon\}, \Gamma, \delta, q_0, F \rangle$, and $\tau : Q \times \Gamma \to \Sigma$ such that t is the tree generated by $\mathcal{A}$ and τ.*
- ⋆ *Let $\mathcal{A} = \langle Q, A \cup \{\varepsilon\}, \Gamma, \delta, q_0, F \rangle$ be an order-n CPDA, and let t be the Σ-labelled tree generated by $\mathcal{A}$ and a given map $\tau : Q \times \Gamma \to \Sigma$. Then there is an order-n recursion scheme over Σ whose value tree is t.*

Moreover, the transformations from scheme to CPDA and vice versa are computable in polynomial time.

Do You Really Need the Collapse? safe *vs* unsafe

The Urzyczyn language *U*

One considers finite words over the alphabet $\{(,),\star\}$. A word in U must we of the following form:

$$(\cdots(\cdots(\cdots((\cdots)\cdots(\cdots)\star\cdots\star$$

- The blue part is the prefix of a well-bracketed word and ends by an unmatched $($.
- The red part is a well-bracketed word.
- The green part consists of k stars, where k is the number of $($ in the blue segment.

For instance $(()(()(()(()())\star\star\star\star\star \in U$

A 2-CPDA recognizing U:

- reading $($: $push_2; push_1^2(a)$
- reading $)$: pop_1
- reading $\star$: *collapse* (first occurence) / pop_2 (others)

Run over $(()(()\star\star\star \in U$

$$\begin{bmatrix}[\bot]\end{bmatrix} \xrightarrow{(} \begin{bmatrix}[a^1\bot]\\ [\bot]\end{bmatrix} \xrightarrow{(} \begin{bmatrix}[a^2a^1\bot]\\ [a^1\bot]\\ [\bot]\end{bmatrix} \xrightarrow{)} \begin{bmatrix}[a^1\bot]\\ [a^1\bot]\\ [\bot]\end{bmatrix} \xrightarrow{(} \begin{bmatrix}[a^3a^1\bot]\\ [a^1\bot]\\ [a^1\bot]\\ [\bot]\end{bmatrix}$$

$$\xrightarrow{(} \begin{bmatrix}[a^4a^3a^1\bot]\\ [a^3a^1\bot]\\ [a^1\bot]\\ [a^1\bot]\\ [\bot]\end{bmatrix} \xrightarrow{)} \begin{bmatrix}[a^3a^1\bot]\\ [a^3a^1\bot]\\ [a^1\bot]\\ [a^1\bot]\\ [\bot]\end{bmatrix} \xrightarrow{\star} \begin{bmatrix}[a^1\bot]\\ [a^1\bot]\\ [\bot]\end{bmatrix} \xrightarrow{\star} \begin{bmatrix}[a^1\bot]\\ [\bot]\end{bmatrix} \xrightarrow{\star} \begin{bmatrix}[\bot]\end{bmatrix}$$

Do You Really Need the Collapse? safe *vs* unsafe

Theorem (Parys'11)

There is no 2-DPDA accepting U. Hence $\mathit{RecSafeTree}_2 \subsetneq \mathit{RecTree}_2$.

Logical Consequences

Black Boxing the Equi-Expressivity Theorem

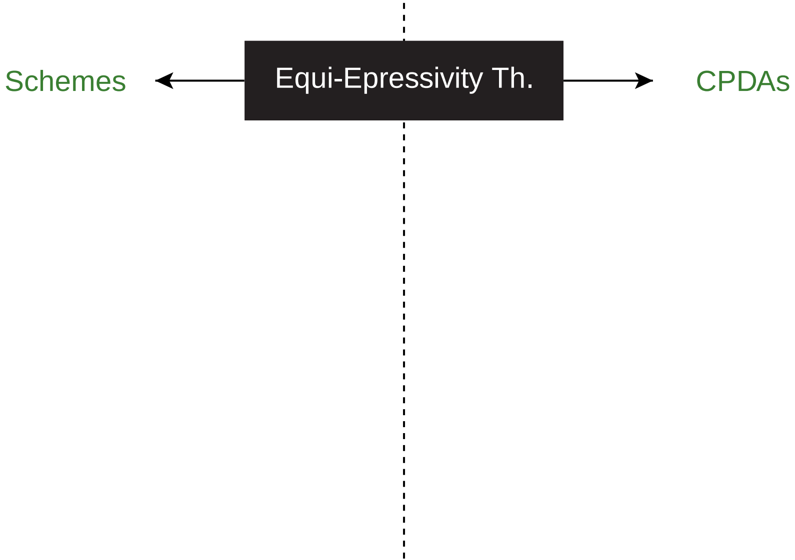
Schemes

CPDAs

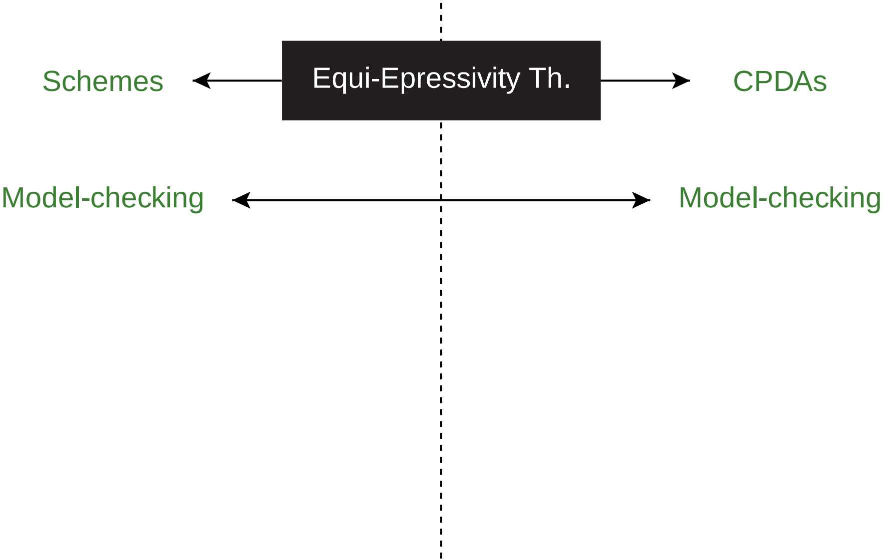
Black Boxing the Equi-Expressivity Theorem

Schemes $\longleftarrow$ Equi-Epressivity Th. $\longrightarrow$ CPDAs

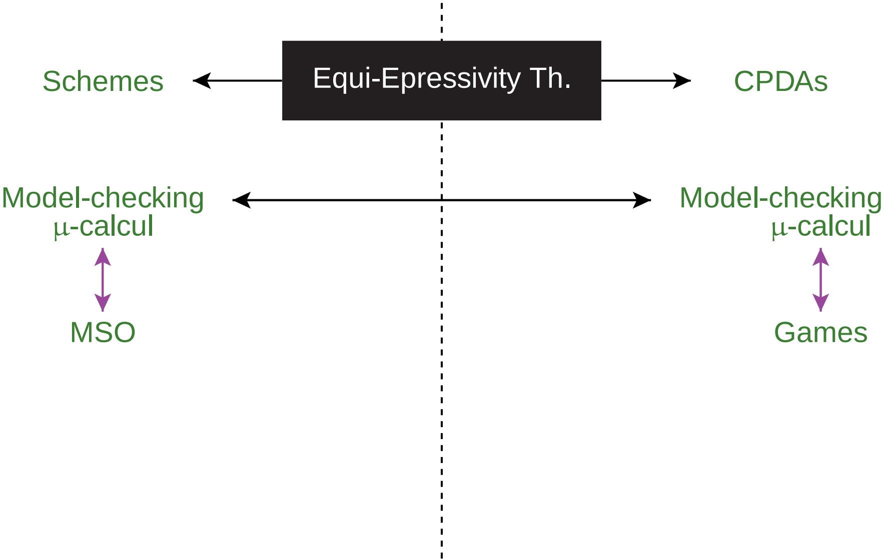
Black Boxing the Equi-Expressivity Theorem



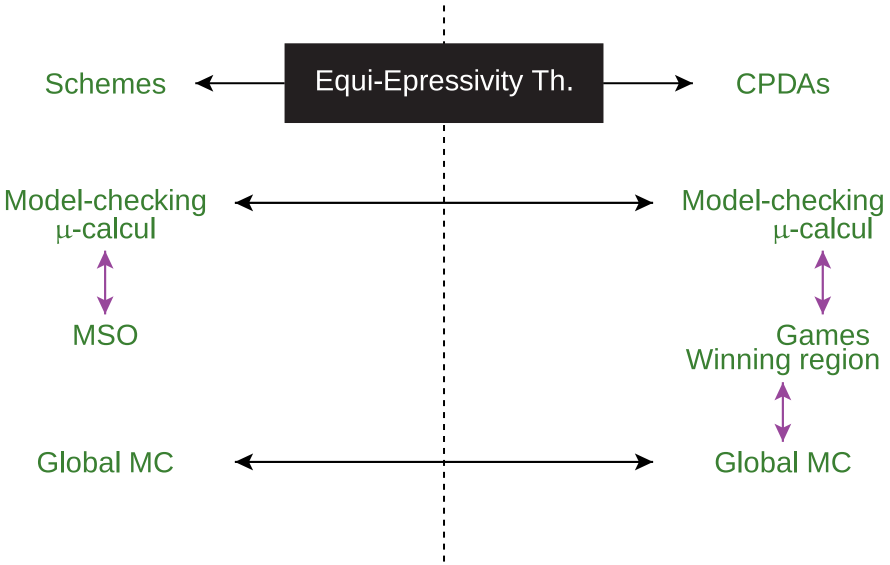
Black Boxing the Equi-Expressivity Theorem



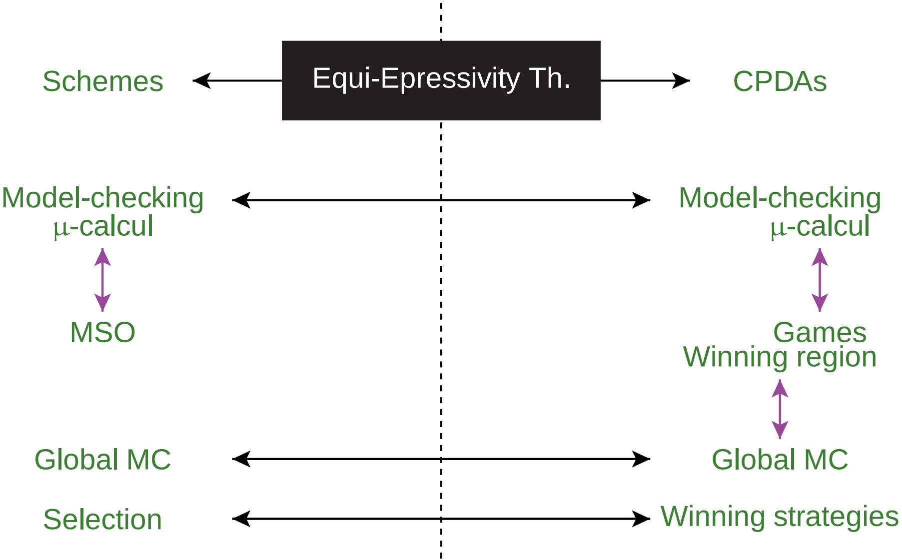
Black Boxing the Equi-Expressivity Theorem



Black Boxing the Equi-Expressivity Theorem



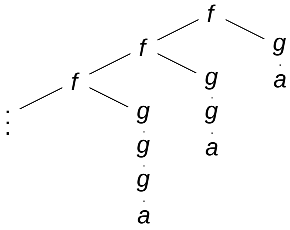
Black Boxing the Equi-Expressivity Theorem



Global Model-Checking

$$\begin{cases} I & \rightarrow \quad F\, g\, (g a) \\ F\, \varphi\, x & \rightarrow \quad f\, (F\, \varphi\, (\varphi\, x))\, x \end{cases}$$

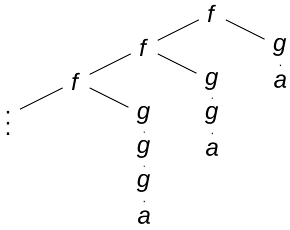
$$\varphi = p_g \wedge \mu X.(\diamond_1 p_a \vee \diamond_1 \diamond_1 X)$$



Global Model-Checking

$$\begin{cases} I & \rightarrow \quad F\, g\,(g a) \\ F\,\varphi\,x & \rightarrow \quad f\,(F\,\varphi\,(\varphi\,x))\,x \end{cases}$$

$$\varphi = p_g \wedge \mu X.(\diamond_1 p_a \vee \diamond_1 \diamond_1 X)$$

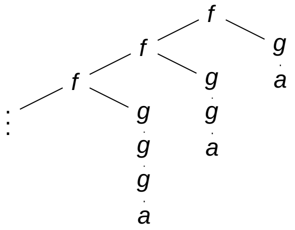


- ⋆ Exogeneous approach : $|t|_\varphi = \{1^n 2 1^k \mid n+k \text{ is odd}\}$

Global Model-Checking

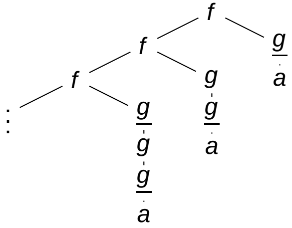
$$\begin{cases} I & \rightarrow & F\,g\,(ga) \\ F\,\varphi\,x & \rightarrow & f\,(F\,\varphi\,(\varphi\,x))\,x \end{cases}$$

$$\varphi = p_g \wedge \mu X.(\diamond_1 p_a \vee \diamond_1 \diamond_1 X)$$



- ⋆ Exogeneous approach : $|t|_\varphi = \{1^n 2 1^k \mid n+k \text{ is odd}\}$
- ⋆ Endogeneous approach : t_φ

$$\begin{cases} I & \rightarrow & H\,\underline{g}\,a \\ H\,z & \rightarrow & f\,(\underline{H}\,g\,z)\,z \\ \underline{H}\,z & \rightarrow & f\,(H\,\underline{g}\,z)\,z \end{cases}$$



Reflection (Global Model-Checking)

Theorem (μ-Calculus Reflection)

Let t be a Σ-labelled tree generated by an order-n recursion scheme $\mathcal{S}$ and φ be a μ-calculus formula.

- ⋆ *There is an algorithm that transforms $(\mathcal{S}, \varphi)$ to an order-n CPDA $\mathcal{A}$ such that $L(\mathcal{A}) = \|t\|_\varphi$.*
- ⋆ *There is an algorithm that transforms $(\mathcal{S}, \varphi)$ to an order-n recursion scheme that generates t_φ.*

Reflection (Global Model-Checking)

Theorem (μ-Calculus Reflection)

Let t be a Σ-labelled tree generated by an order-n recursion scheme $\mathcal{S}$ and φ be a μ-calculus formula.

- *There is an algorithm that transforms $(\mathcal{S}, \varphi)$ to an order-n CPDA $\mathcal{A}$ such that $L(\mathcal{A}) = \|t\|_\varphi$.*
- *There is an algorithm that transforms $(\mathcal{S}, \varphi)$ to an order-n recursion scheme that generates t_φ.*

Corollary (MSO Reflection)

Recursion schemes are reflective with respect to monadic second order logic.

Road Map to Logical Reflection of Schemes

- Input: Order-n recursion scheme $\mathcal{S}$ + a μ-calculus formula φ

Road Map to Logical Reflection of Schemes

- $\star$ Input: Order-*n* recursion scheme $\mathcal{S}$ + a μ-calculus formula φ
- $\star$ Build $\mathcal{A}$ and τ s.t. $t_{\mathcal{S}} = \tau(\mathit{Unfold}(G_{\varepsilon}(\mathcal{A})))$ [Equi-expr. Th]

Road Map to Logical Reflection of Schemes

- ⋆ Input: Order-*n* recursion scheme $\mathcal{S}$ + a μ-calculus formula φ
- ⋆ Build $\mathcal{A}$ and τ s.t. $t_{\mathcal{S}} = \tau(\mathit{Unfold}(G_\varepsilon(\mathcal{A})))$ [Equi-expr. Th]
- ⋆ Build the μ-calculus model-checking parity game $\mathbb{G}$ for φ *vs* $G_\varepsilon(\mathcal{A})$ [Cartesian product with a finite structure]

Road Map to Logical Reflection of Schemes

- $\star$ Input: Order-*n* recursion scheme $\mathcal{S}$ + a μ-calculus formula φ
- $\star$ Build $\mathcal{A}$ and τ s.t. $t_{\mathcal{S}} = \tau(\mathit{Unfold}(G_\varepsilon(\mathcal{A})))$ [Equi-expr. Th]
- $\star$ Build the μ-calculus model-checking parity game $\mathbb{G}$ for φ *vs* $G_\varepsilon(\mathcal{A})$ [Cartesian product with a finite structure]
- $\star$ Compute a finite representation of Eve's winning region in $\mathbb{G}$

Road Map to Logical Reflection of Schemes

- ⋆ Input: Order-n recursion scheme $\mathcal{S}$ + a μ-calculus formula φ
- ⋆ Build $\mathcal{A}$ and τ s.t. $t_{\mathcal{S}} = \tau(\mathit{Unfold}(G_\varepsilon(\mathcal{A})))$ [Equi-expr. Th]
- ⋆ Build the μ-calculus model-checking parity game $\mathbb{G}$ for φ *vs* $G_\varepsilon(\mathcal{A})$ [Cartesian product with a finite structure]
- ⋆ Compute a finite representation of Eve's winning region in $\mathbb{G}$
- ⋆ Incorporate this representation into $\mathcal{A}$ and get $\mathcal{A}'$

Road Map to Logical Reflection of Schemes

- ⋆ Input: Order-n recursion scheme $\mathcal{S}$ + a μ-calculus formula φ
- ⋆ Build $\mathcal{A}$ and τ s.t. $t_{\mathcal{S}} = \tau(\mathit{Unfold}(G_\varepsilon(\mathcal{A})))$ [Equi-expr. Th]
- ⋆ Build the μ-calculus model-checking parity game $\mathbb{G}$ for φ *vs* $G_\varepsilon(\mathcal{A})$ [Cartesian product with a finite structure]
- ⋆ Compute a finite representation of Eve's winning region in $\mathbb{G}$
- ⋆ Incorporate this representation into $\mathcal{A}$ and get $\mathcal{A}'$
- ⋆ Output: $\mathcal{S}'$ s.t. $\tau'(\mathit{Unfold}(G_\varepsilon(\mathcal{A}'))) = t_{\mathcal{S}'} = t_{\mathcal{S},\varphi}$ [Equi-expr. Th]

Road Map to Logical Reflection of Schemes

- ⋆ Input: Order-n recursion scheme $\mathcal{S}$ + a μ-calculus formula φ
- ⋆ Build $\mathcal{A}$ and τ s.t. $t_{\mathcal{S}} = \tau(\mathit{Unfold}(G_\varepsilon(\mathcal{A})))$ [Equi-expr. Th]
- ⋆ Build the μ-calculus model-checking parity game $\mathbb{G}$ for φ *vs* $G_\varepsilon(\mathcal{A})$ [Cartesian product with a finite structure]
- ⋆ Compute a finite representation of Eve's winning region in $\mathbb{G}$
- ⋆ Incorporate this representation into $\mathcal{A}$ and get $\mathcal{A}'$
- ⋆ Output: $\mathcal{S}'$ s.t. $\tau'(\mathit{Unfold}(G_\varepsilon(\mathcal{A}'))) = t_{\mathcal{S}'} = t_{\mathcal{S},\varphi}$ [Equi-expr. Th]

Hard work is on CPDA parity games

A stack with link is a well bracketed word with a binary relation:

$$[\,[\,[\,\perp\,\alpha\,]\,]\;[\,[\,\perp\,]\,[\,\perp\,\alpha\,\beta\,\gamma\,]\,]\,]$$

Regular Sets of Stacks with Links

A stack with link is a well bracketed word with a binary relation:

$$[[[\perp\alpha]][[\perp][\perp\alpha\beta\gamma]]]$$

Automaton

A (deterministic) automaton is a tuple $\langle Q, A, q_{in}, F, \delta \rangle$ where $\delta : (Q \times A) \cup (Q \times A \times Q) \to Q$:

- ⋆ On reading a position without a link the state is updated wrt the current state and current symbol;
- ⋆ On reading a position with a link the state is updated wrt the current state, the current symbol and the state the automaton was after the targeted position;

Regular Sets of Stacks with Links

A stack with link is a well bracketed word with a binary relation:

$$[\,[\,[\perp\alpha\,]\,]\,[\,[\perp]\,[\perp\alpha\,\beta\,\gamma\,]\,]\,]$$

Automaton

A (deterministic) automaton is a tuple $\langle Q, A, q_{in}, F, \delta \rangle$ where $\delta : (Q \times A) \cup (Q \times A \times Q) \to Q$:

- On reading a position without a link the state is updated wrt the current state and current symbol;
- On reading a position with a link the state is updated wrt the current state, the current symbol and the state the automaton was after the targeted position;

A set of stacks L is regular iff there is an automaton that decide for any stack in input whether it is in L.

Winning Regions in CPDA Games and Logical Reflection

Theorem

The winning regions in CPDA pushdown parity games are regular sets of configurations.

Theorem

CPDA are closed under regular test.

Corollary

Recursion Schemes are logically reflective w.r.t μ-calculus.

A Nice Consequence of Logical Reflection

Theorem

Let t be a Σ-labelled tree given by some order-n recursion scheme $\mathcal{S}$ and let $\mathcal{I}$ be an MSO-interpretation. The unfolding of $\mathcal{I}(t)$ from any vertex u can be generated by an order-$(n+1)$ recursion scheme.

Road Map to Selection for Schemes

- Input: Order-n recursion scheme $\mathcal{S}$ + an *MSO* formula $\varphi(X)$

Road Map to Selection for Schemes

- $\star$ Input: Order-n recursion scheme $\mathcal{S}$ + an *MSO* formula $\varphi(X)$
- $\star$ Build $\mathcal{A}$ and τ s.t. $t_{\mathcal{S}} = \tau(\mathit{Unfold}(G_\varepsilon(\mathcal{A})))$ [Equi-expr. Th]

Road Map to Selection for Schemes

- Input: Order-n recursion scheme $\mathcal{S}$ + an *MSO* formula $\varphi(X)$
- Build $\mathcal{A}$ and τ s.t. $t_{\mathcal{S}} = \tau(\mathit{Unfold}(G_{\varepsilon}(\mathcal{A})))$ [Equi-expr. Th]
- Build an nondeterministic parity tree automaton $\mathcal{A}_{\varphi}$ that accepts marked trees satisfying $\varphi[X \leftarrow \text{marked}]$

Road Map to Selection for Schemes

- $\star$ Input: Order-*n* recursion scheme $\mathcal{S}$ + an *MSO* formula $\varphi(X)$
- $\star$ Build $\mathcal{A}$ and τ s.t. $t_{\mathcal{S}} = \tau(\mathit{Unfold}(G_\varepsilon(\mathcal{A})))$ [Equi-expr. Th]
- $\star$ Build an nondeterministic parity tree automaton $\mathcal{A}_\varphi$ that accepts marked trees satisfying $\varphi[X \leftarrow \text{marked}]$
- $\star$ Consider the acceptance game of $t_{\mathcal{S}}$ by $\mathcal{A}_\varphi$ where Eve is also in charge of giving the marking
- $\star$ Compute a winning strategy in the game that is synchronised with the underlying CPDA defining the game

Road Map to Selection for Schemes

- Input: Order-n recursion scheme $\mathcal{S}$ + an *MSO* formula $\varphi(X)$
- Build $\mathcal{A}$ and τ s.t. $t_{\mathcal{S}} = \tau(\mathit{Unfold}(G_\varepsilon(\mathcal{A})))$ [Equi-expr. Th]
- Build an nondeterministic parity tree automaton $\mathcal{A}_\varphi$ that accepts marked trees satisfying $\varphi[X \leftarrow \text{marked}]$
- Consider the acceptance game of $t_{\mathcal{S}}$ by $\mathcal{A}_\varphi$ where Eve is also in charge of giving the marking
- Compute a winning strategy in the game that is synchronised with the underlying CPDA defining the game
- Incorporate this strategy into the game and get a new CPDA $\mathcal{A}'$ (forget about the run component)

Road Map to Selection for Schemes

- Input: Order-*n* recursion scheme $\mathcal{S}$ + an *MSO* formula $\varphi(X)$
- Build $\mathcal{A}$ and τ s.t. $t_{\mathcal{S}} = \tau(\mathit{Unfold}(G_\varepsilon(\mathcal{A})))$ [Equi-expr. Th]
- Build an nondeterministic parity tree automaton $\mathcal{A}_\varphi$ that accepts marked trees satisfying $\varphi[X \leftarrow \text{marked}]$
- Consider the acceptance game of $t_{\mathcal{S}}$ by $\mathcal{A}_\varphi$ where Eve is also in charge of giving the marking
- Compute a winning strategy in the game that is synchronised with the underlying CPDA defining the game
- Incorporate this strategy into the game and get a new CPDA $\mathcal{A}'$ (forget about the run component)
- Output: $\mathcal{S}'$ s.t. $\tau'(\mathit{Unfold}(G_\varepsilon(\mathcal{A}'))) = t_{\mathcal{S}'}$ [Equi-expr. Th]

Road Map to Selection for Schemes

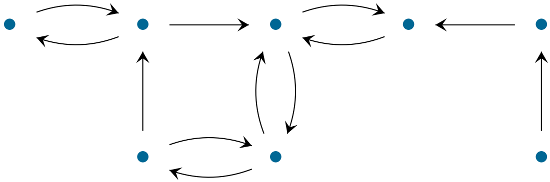
- $\star$ Input: Order-n recursion scheme $\mathcal{S}$ + an *MSO* formula $\varphi(X)$
- $\star$ Build $\mathcal{A}$ and τ s.t. $t_{\mathcal{S}} = \tau(\mathit{Unfold}(G_\varepsilon(\mathcal{A})))$ [Equi-expr. Th]
- $\star$ Build an nondeterministic parity tree automaton $\mathcal{A}_\varphi$ that accepts marked trees satisfying $\varphi[X \leftarrow \text{marked}]$
- $\star$ Consider the acceptance game of $t_{\mathcal{S}}$ by $\mathcal{A}_\varphi$ where Eve is also in charge of giving the marking
- $\star$ Compute a winning strategy in the game that is synchronised with the underlying CPDA defining the game
- $\star$ Incorporate this strategy into the game and get a new CPDA $\mathcal{A}'$ (forget about the run component)
- $\star$ Output: $\mathcal{S}'$ s.t. $\tau'(\mathit{Unfold}(G_\varepsilon(\mathcal{A}'))) = t_{\mathcal{S}'}$ [Equi-expr. Th]

Hard work is to build a synchronised strategy in a CPDA parity games

CPDA Parity Games

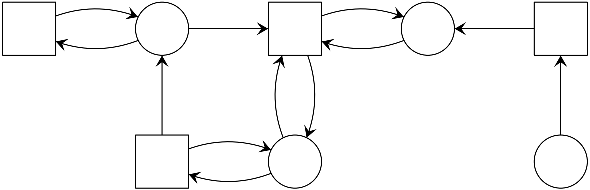
Ingredients :

- ⋆ A graph.



Ingredients :

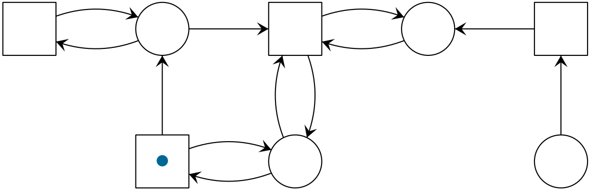
- ⋆ A graph.
- ⋆ Two players : Eve (◯) and Adam (□).



Games in a Nutshell

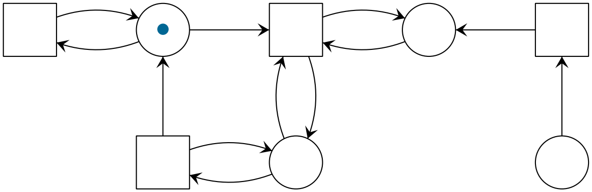
Ingredients :

- ⋆ A graph.
- ⋆ Two players : Eve (◯) and Adam (□).
- ⋆ Play: moving a token.



Ingredients :

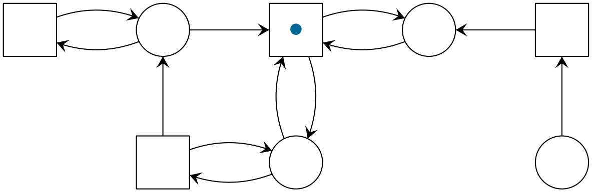
- ⋆ A graph.
- ⋆ Two players : Eve (◯) and Adam (□).
- ⋆ Play: moving a token.



Games in a Nutshell

Ingredients :

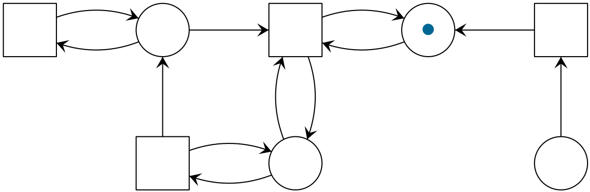
- ⋆ A graph.
- ⋆ Two players : Eve (◯) and Adam (□).
- ⋆ Play: moving a token.



Games in a Nutshell

Ingredients :

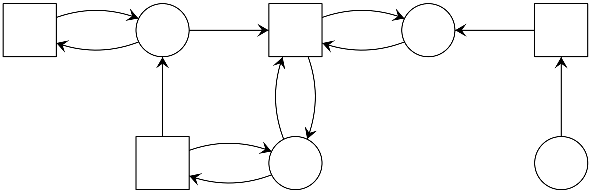
- ⋆ A graph.
- ⋆ Two players : Eve (◯) and Adam (□).
- ⋆ Play: moving a token.



Games in a Nutshell

Ingredients :

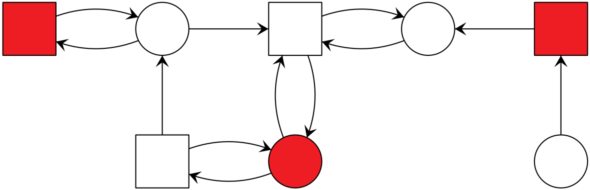
- ⋆ A graph.
- ⋆ Two players : Eve (◯) and Adam (□).
- ⋆ Play: moving a token.



Games in a Nutshell

Ingredients :

- ⋆ A graph.
- ⋆ Two players : Eve (◯) and Adam (□).
- ⋆ Play: moving a token.
- ⋆ A winning condition: reachability.

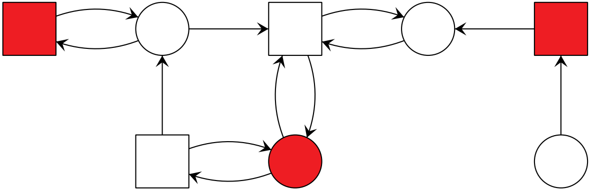


Eve wins iff a final vertex is visited.

Games in a Nutshell

Ingredients :

- ⋆ A graph.
- ⋆ Two players : Eve (◯) and Adam (□).
- ⋆ Play: moving a token.
- ⋆ A winning condition: Büchi.

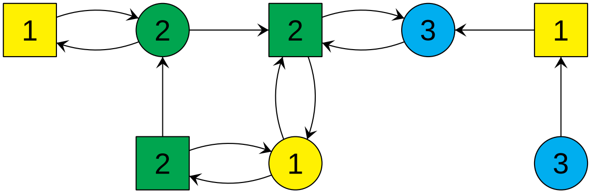


Eve wins iff final vertices are visited infinitely often.

Games in a Nutshell

Ingredients :

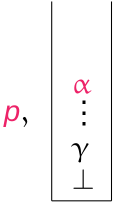
- ⋆ A graph.
- ⋆ Two players : Eve (◯) and Adam (□).
- ⋆ Play: moving a token.
- ⋆ A winning condition: parity.



Eve wins iff the smallest infinitely repeated colour is even.

A General Model: Abstract Pushdown Automata

Abstract Pushdown Automata: finite control + stack over a (possibly) infinite alphabet.

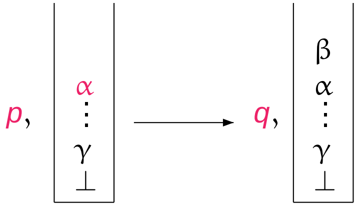


$$\Delta : Q \times \Gamma \rightarrow 2^{Q \times \{rew(\alpha), pop, push(\alpha) \mid \alpha \in \Gamma\}}$$

A General Model: Abstract Pushdown Automata

Abstract Pushdown Automata: finite control + stack over a (possibly) **infinite** alphabet.

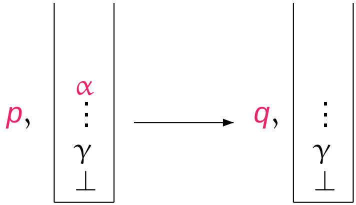
$(q, push(\beta))$:



$$\Delta : Q \times \Gamma \rightarrow 2^{Q \times \{rew(\alpha), pop, push(\alpha) \mid \alpha \in \Gamma\}}$$

Abstract Pushdown Automata: finite control + stack over a (possibly) infinite alphabet.

(q, pop):

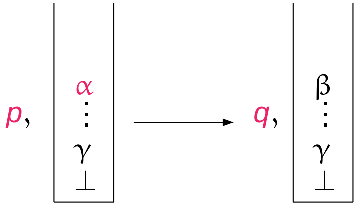


$$\Delta : Q \times \Gamma \rightarrow 2^{Q \times \{rew(\alpha), pop, push(\alpha) \mid \alpha \in \Gamma\}}$$

A General Model: Abstract Pushdown Automata

Abstract Pushdown Automata: finite control + stack over a (possibly) infinite alphabet.

$(q, rew(\beta))$:

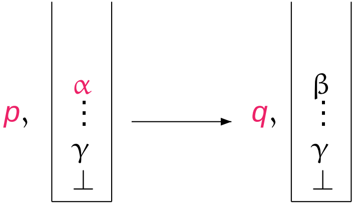


$$\Delta : Q \times \Gamma \rightarrow 2^{Q \times \{rew(\alpha), pop, push(\alpha) \mid \alpha \in \Gamma\}}$$

A General Model: Abstract Pushdown Automata

Abstract Pushdown Automata: finite control + stack over a (possibly) infinite alphabet.

$(q, rew(\beta))$:



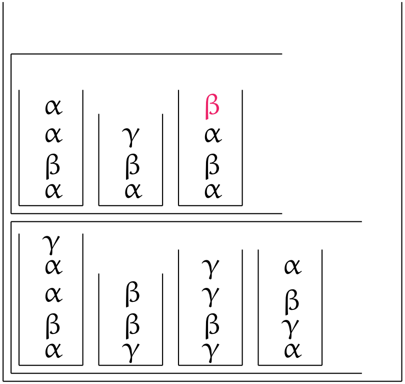
$$\Delta : Q \times \Gamma \to 2^{Q \times \{rew(\alpha), pop, push(\alpha) \mid \alpha \in \Gamma\}}$$

Pushdown automata: abstract pushdown automata whose stack alphabet is finite

Order-*n* Pushdown Automata As Abstract Pushdown Automata

Order-*n* pushdown automata: finite control + order-*n* stack.

q,

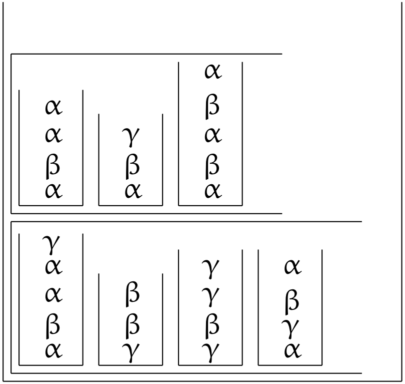


Order-*n* Pushdown Automata As Abstract Pushdown Automata

Order-*n* pushdown automata: finite control + order-*n* stack.

push$_1(\alpha)$:

q,

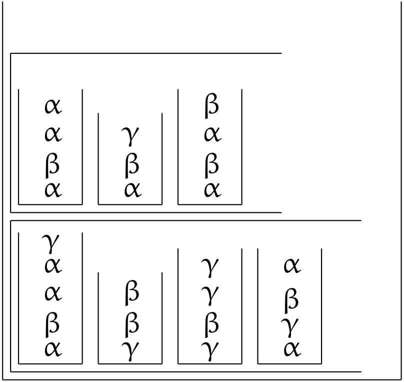


Order-n Pushdown Automata As Abstract Pushdown Automata

Order-n pushdown automata: finite control + order-n stack.

pop_1:

q,

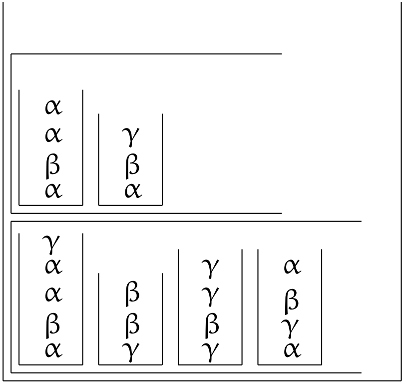


Order-n Pushdown Automata As Abstract Pushdown Automata

Order-n pushdown automata: finite control + order-n stack.

pop_2:

q,

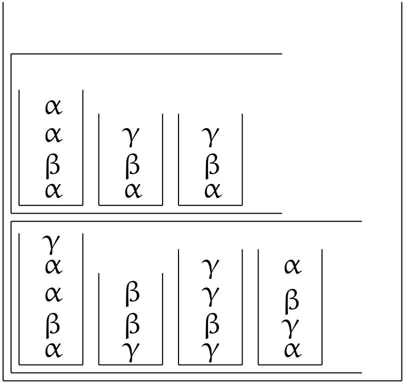


Order-n Pushdown Automata As Abstract Pushdown Automata

Order-n pushdown automata: finite control + order-n stack.

push$_2$:

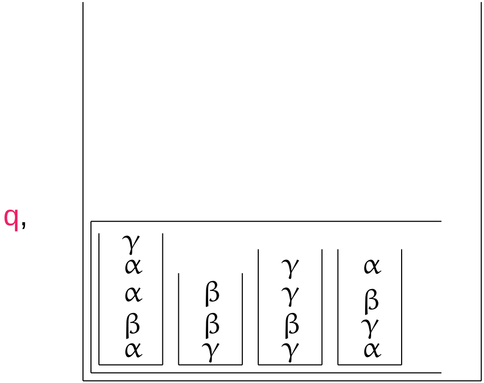
q,



Order-n Pushdown Automata As Abstract Pushdown Automata

Order-n pushdown automata: finite control + order-n stack.

pop_3:

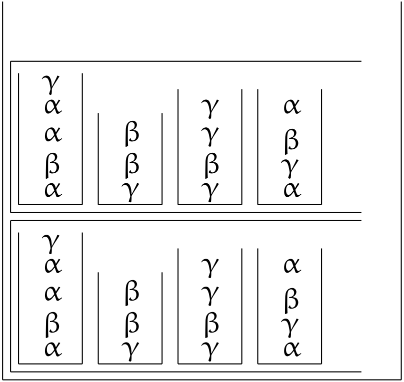


Order-n Pushdown Automata As Abstract Pushdown Automata

Order-n pushdown automata: finite control + order-n stack.

push$_3$:

q,



Order-n Pushdown Automata As Abstract Pushdown Automata

Order-n pushdown automata: finite control + order-n stack.

Order-n pushdown automata $\mathcal{P} = (Q, \Sigma, \Delta)$

Define $\mathcal{P}' = (Q, \Gamma, \Delta')$ as:

- $\Gamma =$ set of all order-$(n-1)$ stacks

Order-n Pushdown Automata As Abstract Pushdown Automata

Order-n pushdown automata: finite control + order-n stack.

Order-n pushdown automata $\mathcal{P} = (Q, \Sigma, \Delta)$

Define $\mathcal{P}' = (Q, \Gamma, \Delta')$ as:

- ⋆ $\Gamma =$ set of all order-$(n-1)$ stacks
- ⋆ $(q, pop) \in \Delta'(p, \gamma)$ iff $(q, pop_n) \in \Delta(q, top_1(\gamma))$
- ⋆ $(q, push(\gamma)) \in \Delta'(p, \gamma)$ iff $(q, push_n) \in \Delta(q, top_1(\gamma))$
- ⋆ $(q, rew(op(\gamma))) \in \Delta'(p, \gamma)$ iff $(q, op) \in \Delta(q, top_1(\gamma))$ with op order $k < n$ action.

Order-n Pushdown Automata As Abstract Pushdown Automata

Order-n pushdown automata: finite control + order-n stack.

Order-n pushdown automata $\mathcal{P} = (Q, \Sigma, \Delta)$

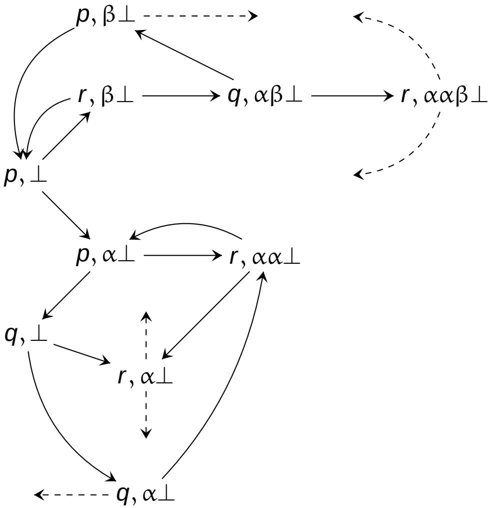
Define $\mathcal{P}' = (Q, \Gamma, \Delta')$ as:

- $\star$ $\Gamma =$ set of all order-$(n-1)$ stacks
- $\star$ $(q, pop) \in \Delta'(p, \gamma)$ iff $(q, pop_n) \in \Delta(q, top_1(\gamma))$
- $\star$ $(q, push(\gamma)) \in \Delta'(p, \gamma)$ iff $(q, push_n) \in \Delta(q, top_1(\gamma))$
- $\star$ $(q, rew(op(\gamma))) \in \Delta'(p, \gamma)$ iff $(q, op) \in \Delta(q, top_1(\gamma))$ with op order $k < n$ action.

Remark. The same works for order-n CPDA without n-links.

Playing with Abstract Pushdown Automata

$$
\begin{aligned}
Q &= \{p, q, r\}\\
\Gamma &= \{\alpha, \beta, \perp\}\\
\Delta(p,\alpha) &= \{pop(q), push(r,\alpha)\}\\
\Delta(q,\alpha) &= \{pop(p), push(r,\alpha)\}\\
\Delta(q,\alpha) &= \{pop(p), push(r,\alpha)\}\\
\Delta(r,\alpha) &= \{pop(p), pop(r)\}\\
\Delta(p,\beta) &= \{pop(p), push(p,\alpha)\}\\
\Delta(q,\beta) &= \{pop(p), push(r,\alpha)\}\\
\Delta(r,\beta) &= \{pop(p), push(q,\alpha)\}\\
\Delta(p,\perp) &= \{push(p,\alpha), push(r,\beta)\}\\
\Delta(q,\perp) &= \{push(q,\alpha), push(r,\alpha)\}\\
\Delta(r,\perp) &= \{push(q,\beta), push(r,\alpha)\}
\end{aligned}
$$

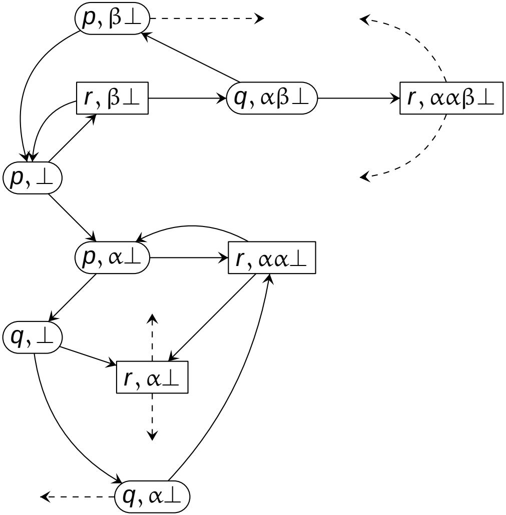


Playing with Abstract Pushdown Automata

$$
\begin{aligned}
Q &= \{p, q, r\}\\
\Gamma &= \{\alpha, \beta, \perp\}\\
\Delta(p, \alpha) &= \{pop(q), push(r, \alpha)\}\\
\Delta(q, \alpha) &= \{pop(p), push(r, \alpha)\}\\
\Delta(q, \alpha) &= \{pop(p), push(r, \alpha)\}\\
\Delta(r, \alpha) &= \{pop(p), pop(r)\}\\
\Delta(p, \beta) &= \{pop(p), push(p, \alpha)\}\\
\Delta(q, \beta) &= \{pop(p), push(r, \alpha)\}\\
\Delta(r, \beta) &= \{pop(p), push(q, \alpha)\}\\
\Delta(p, \perp) &= \{push(p, \alpha), push(r, \beta)\}\\
\Delta(q, \perp) &= \{push(q, \alpha), push(r, \alpha)\}\\
\Delta(r, \perp) &= \{push(q, \beta), push(r, \alpha)\}
\end{aligned}
$$

Associated parity game:

$Q_E = \{p, q\}$ and $Q_A = \{r\}$



Playing with Abstract Pushdown Automata

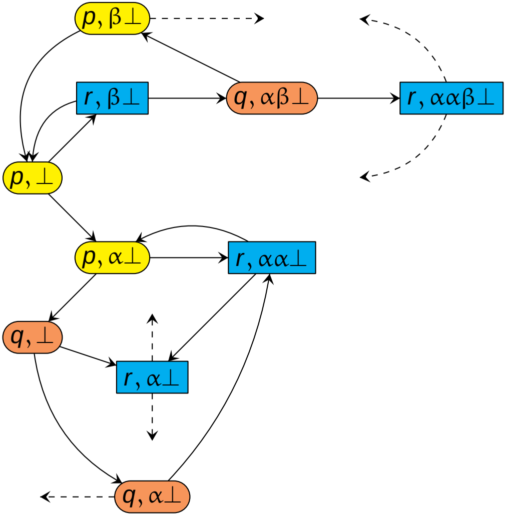
$$
\begin{aligned}
Q &= \{p, q, r\} \\
\Gamma &= \{\alpha, \beta, \perp\} \\
\Delta(p, \alpha) &= \{pop(q), push(r, \alpha)\} \\
\Delta(q, \alpha) &= \{pop(p), push(r, \alpha)\} \\
\Delta(q, \alpha) &= \{pop(p), push(r, \alpha)\} \\
\Delta(r, \alpha) &= \{pop(p), pop(r)\} \\
\Delta(p, \beta) &= \{pop(p), push(p, \alpha)\} \\
\Delta(q, \beta) &= \{pop(p), push(r, \alpha)\} \\
\Delta(r, \beta) &= \{pop(p), push(q, \alpha)\} \\
\Delta(p, \perp) &= \{push(p, \alpha), push(r, \beta)\} \\
\Delta(q, \perp) &= \{push(q, \alpha), push(r, \alpha)\} \\
\Delta(r, \perp) &= \{push(q, \beta), push(r, \alpha)\}
\end{aligned}
$$

Associated parity game:

$Q_E = \{p, q\}$ and $Q_A = \{r\}$

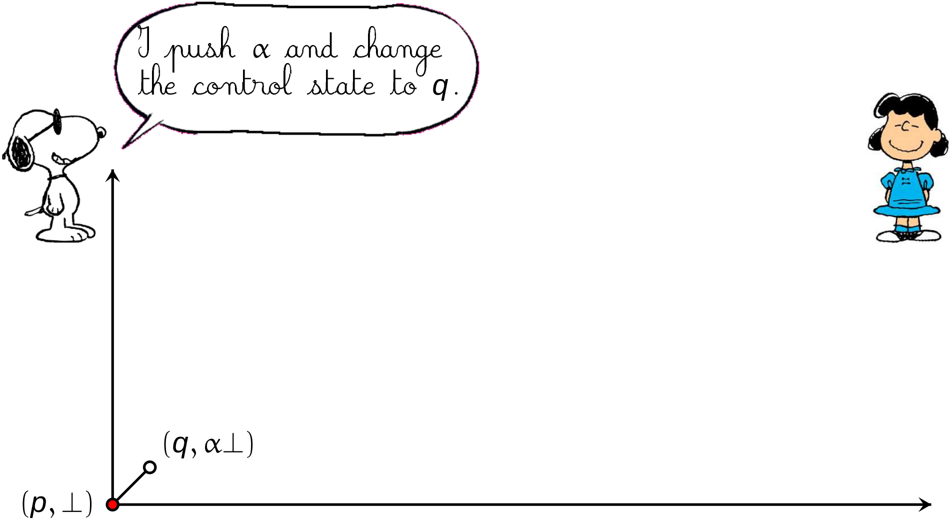
$\rho(p) = 0, \ \rho(q) = 2, \ \rho(r) = 1$

Eve wins iff the smallest infinitely often visited color is even

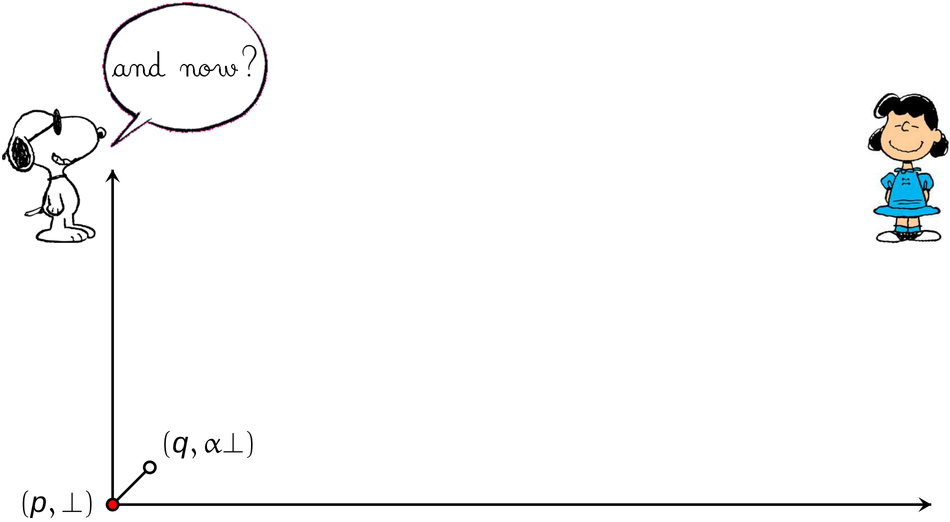


Simulation Game for Abstract Pushdown Games

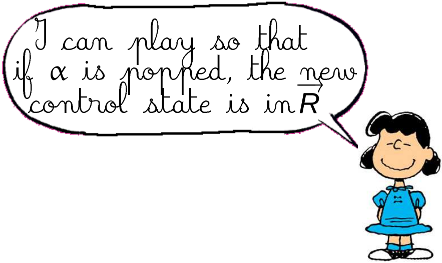
Simulation Game for Abstract Pushdown Games



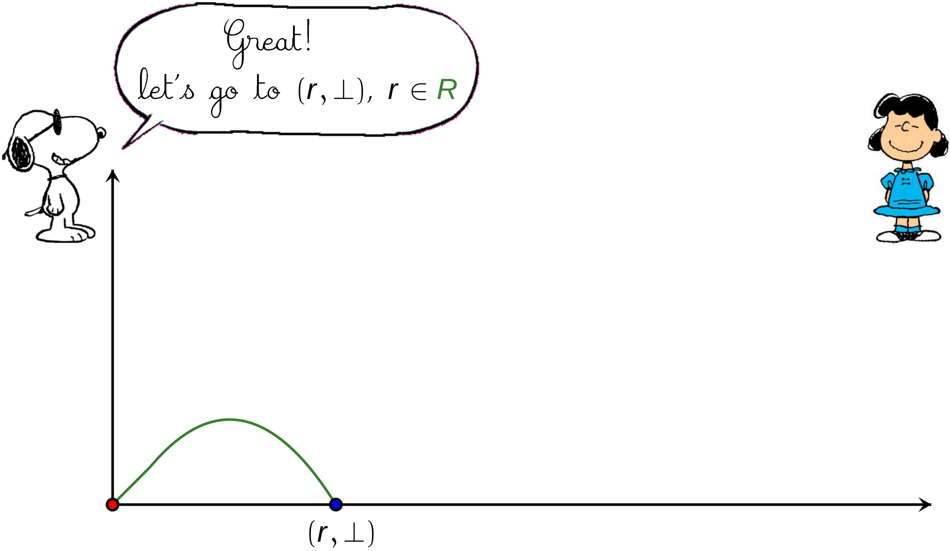
Simulation Game for Abstract Pushdown Games



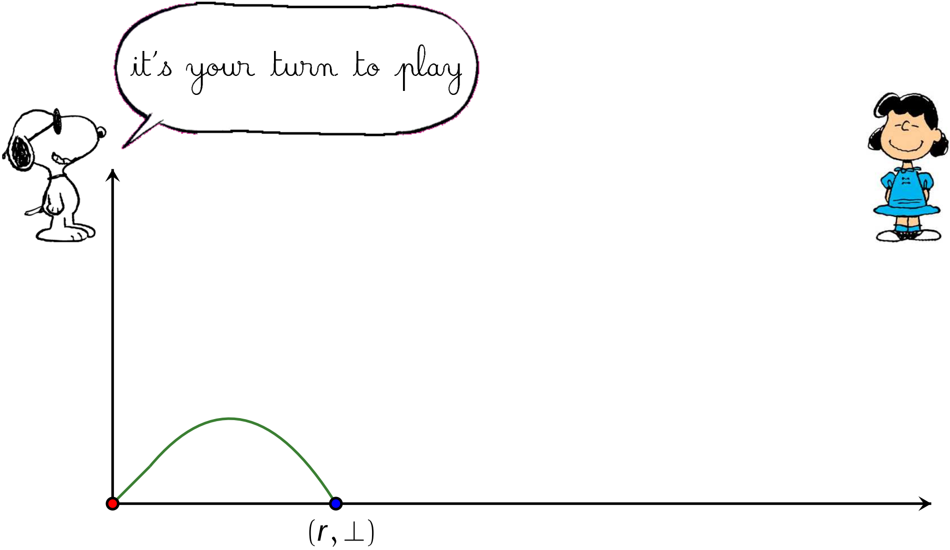
Simulation Game for Abstract Pushdown Games



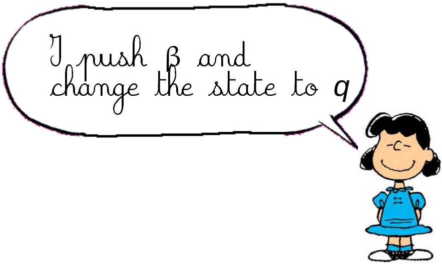
Simulation Game for Abstract Pushdown Games

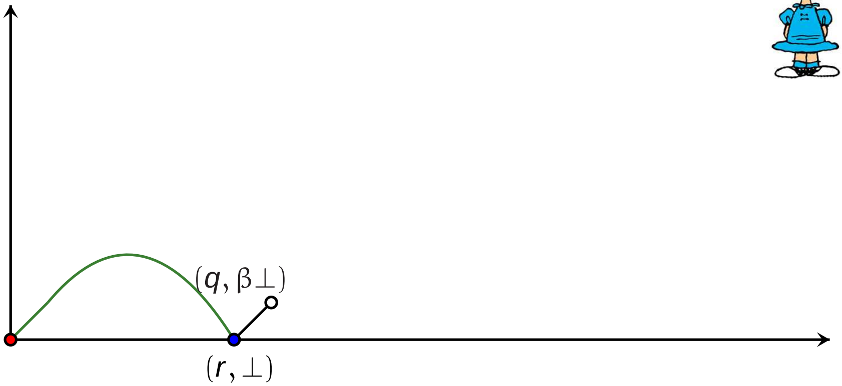


Simulation Game for Abstract Pushdown Games

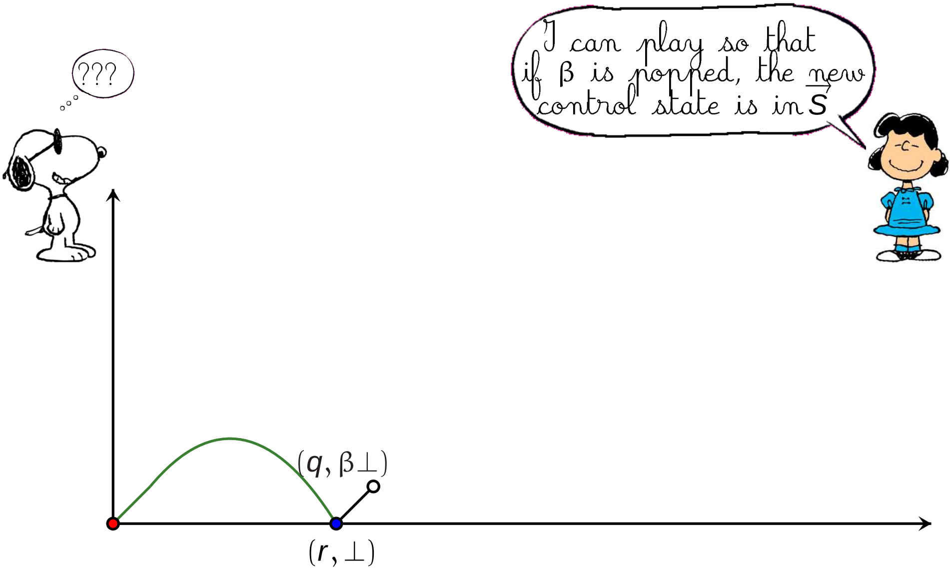


Simulation Game for Abstract Pushdown Games

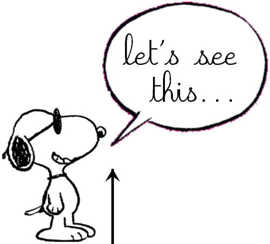


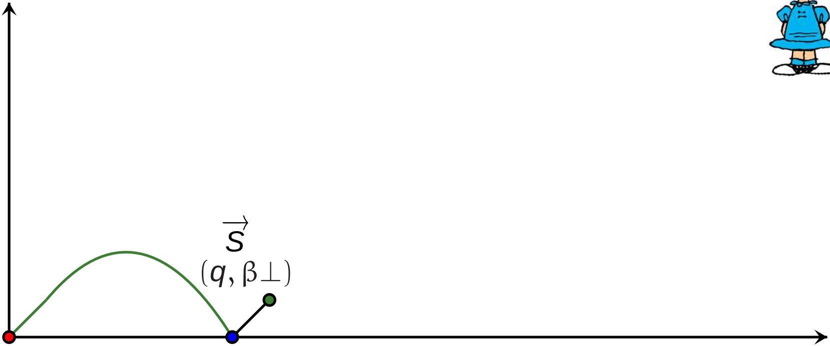


Simulation Game for Abstract Pushdown Games



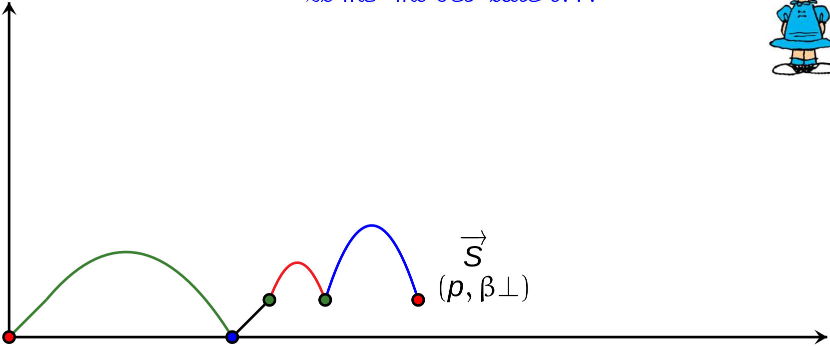
Simulation Game for Abstract Pushdown Games





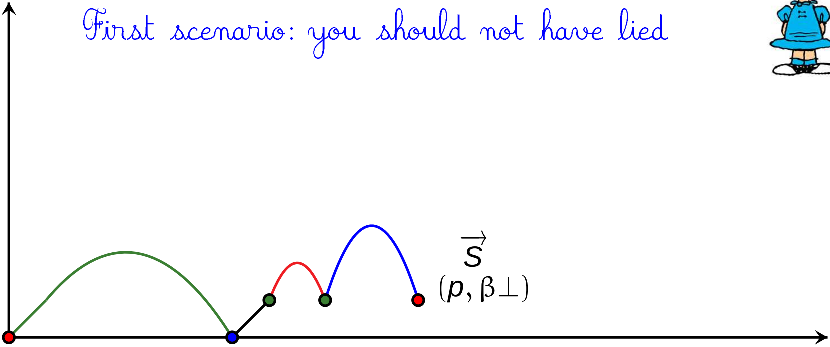
Simulation Game for Abstract Pushdown Games

Some moves later...

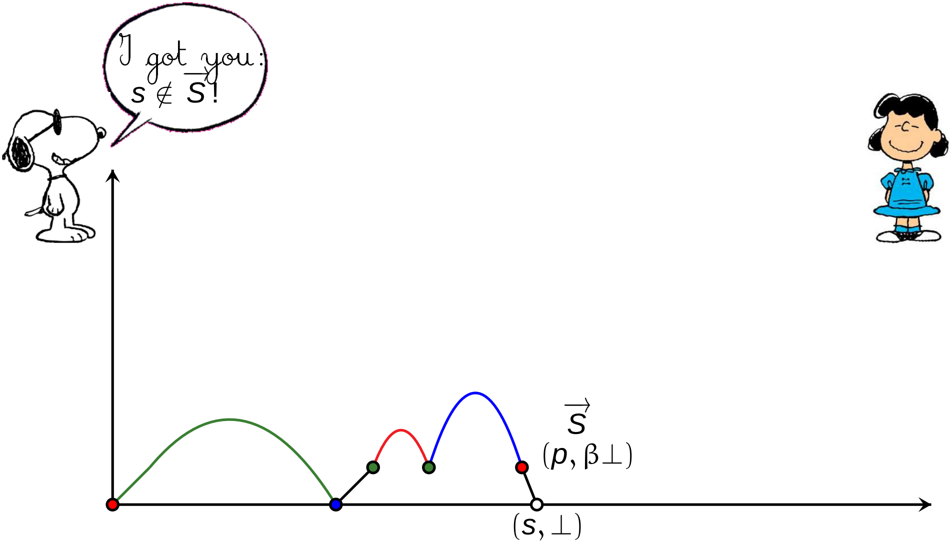


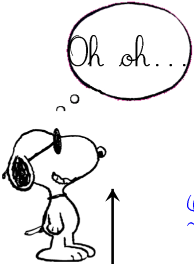
Simulation Game for Abstract Pushdown Games

First scenario: you should not have lied

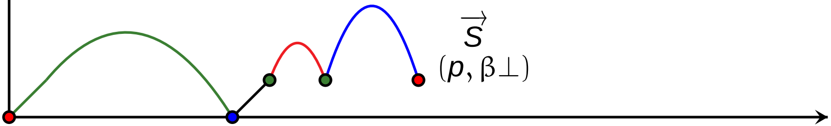


Simulation Game for Abstract Pushdown Games

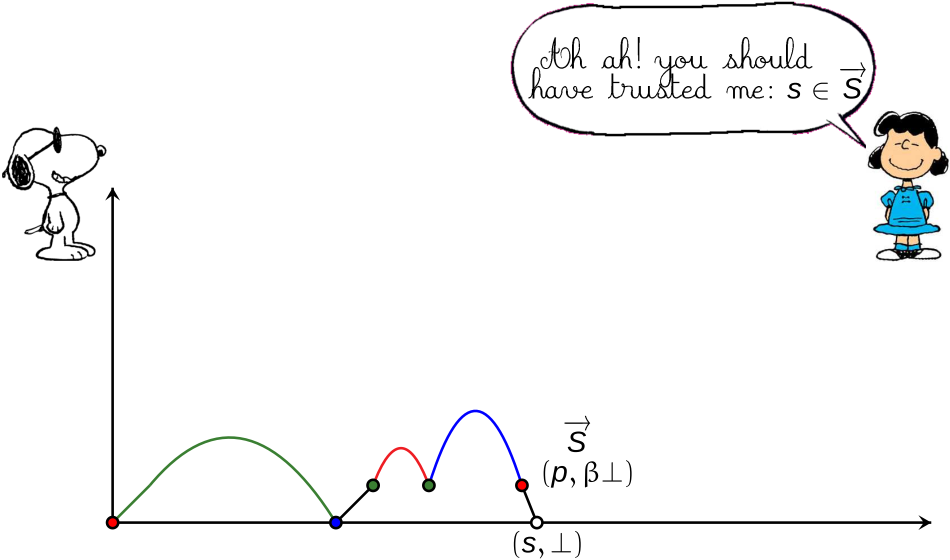




Second scenario: you should have trusted me

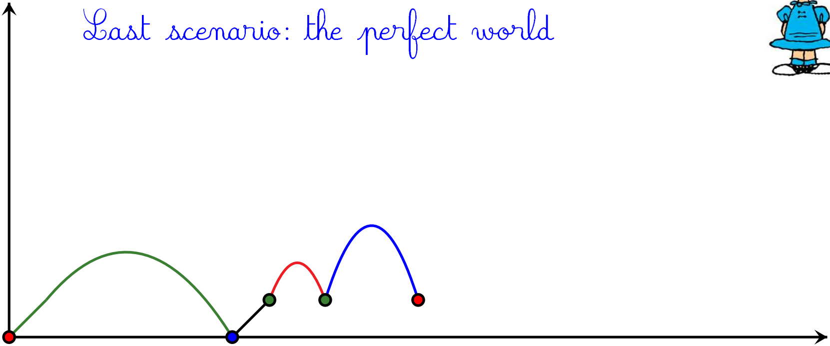


Simulation Game for Abstract Pushdown Games

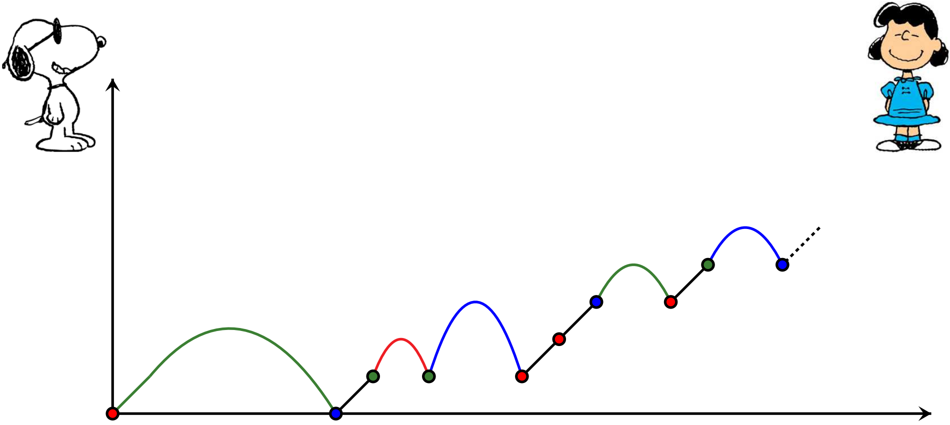


Simulation Game for Abstract Pushdown Games

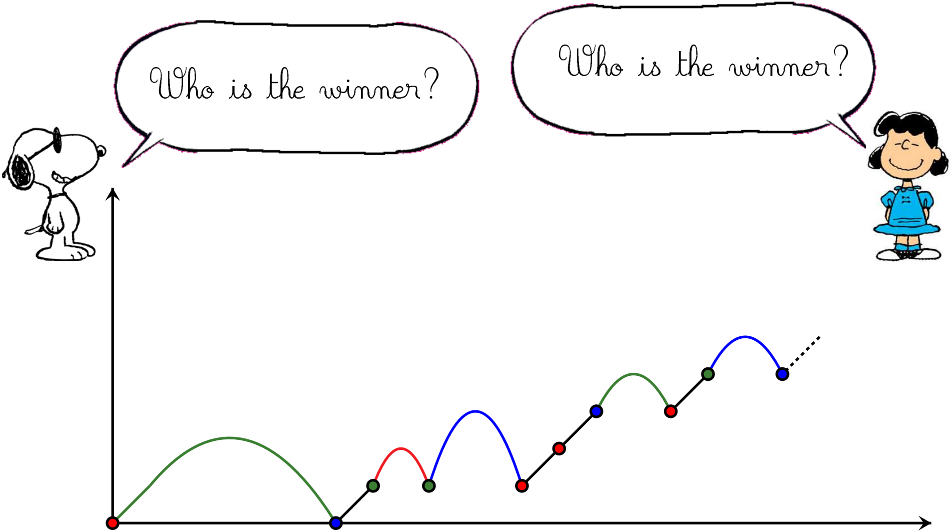
Last scenario: the perfect world



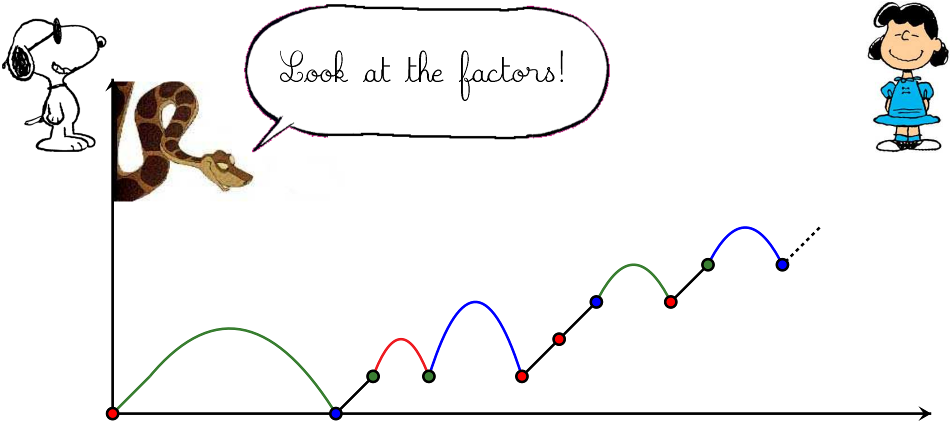
Simulation Game for Abstract Pushdown Games



Simulation Game for Abstract Pushdown Games



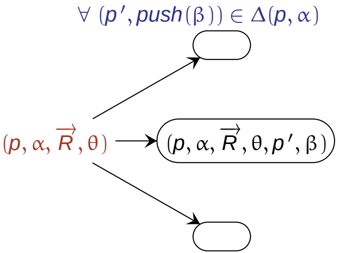
Simulation Game for Abstract Pushdown Games



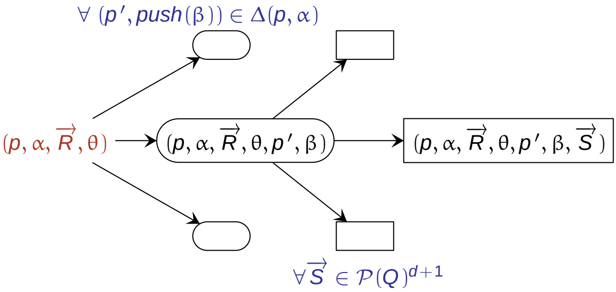
More Formally: Simulation Game

$(p, \alpha, \overrightarrow{R}, \theta)$

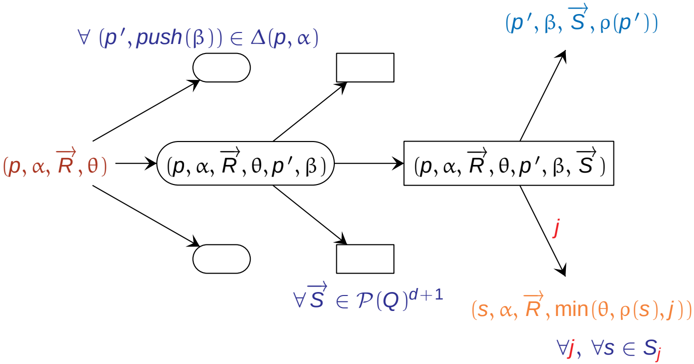
More Formally: Simulation Game



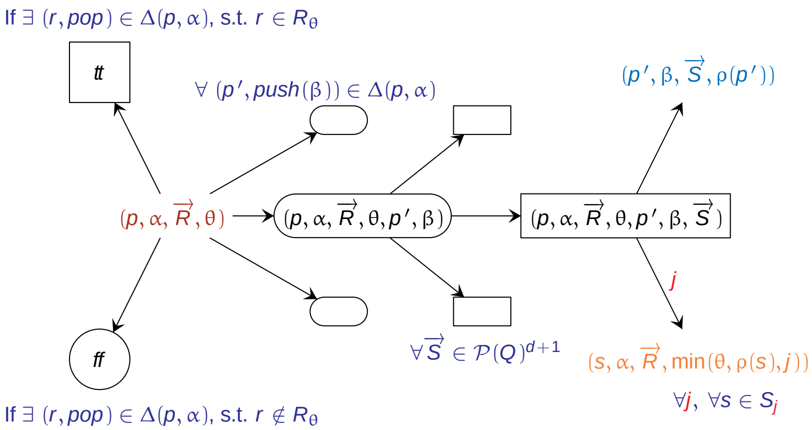
More Formally: Simulation Game



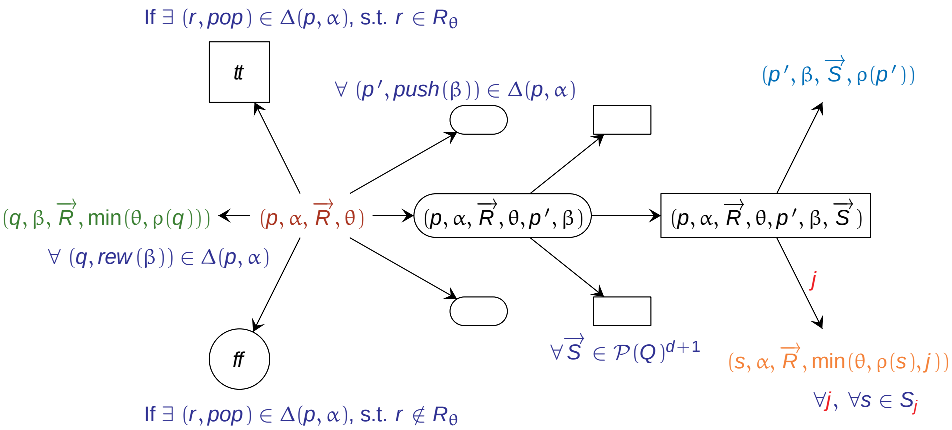
More Formally: Simulation Game



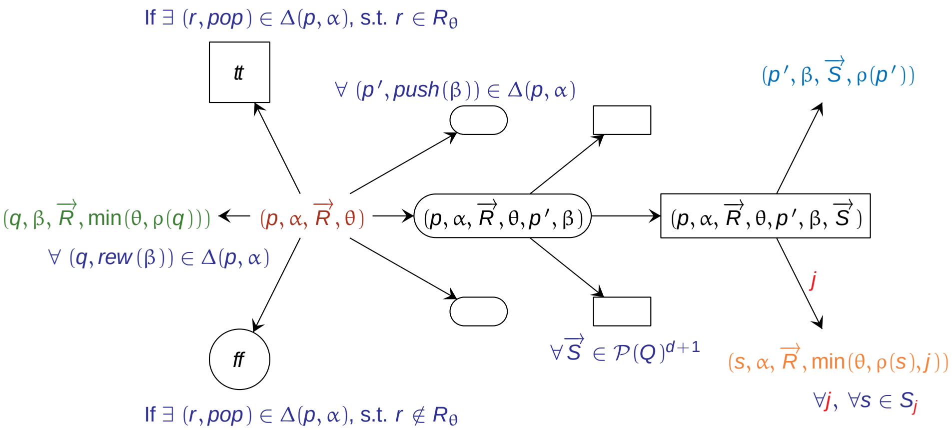
More Formally: Simulation Game



More Formally: Simulation Game



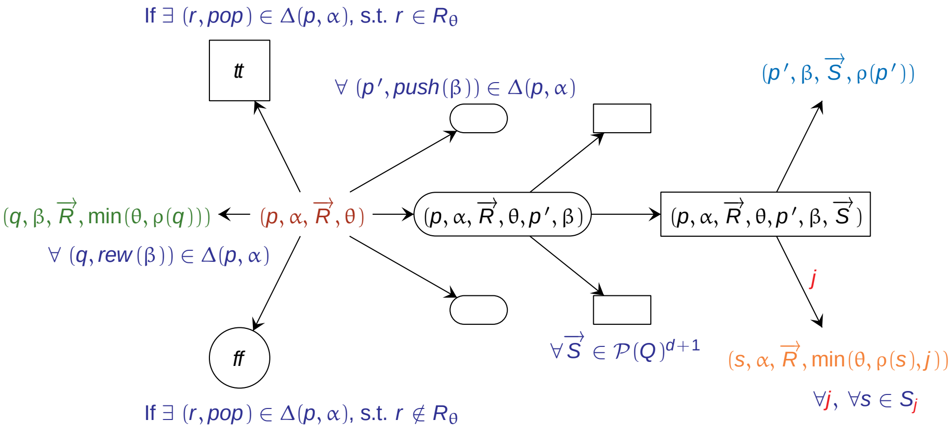
More Formally: Simulation Game



Theorem

Eve wins from $(p, \perp)$ in $\mathbb{G}$ iff she wins from $(p, \perp, (\emptyset, \ldots, \emptyset), \rho(p))$ in $\widetilde{\mathbb{G}}$.

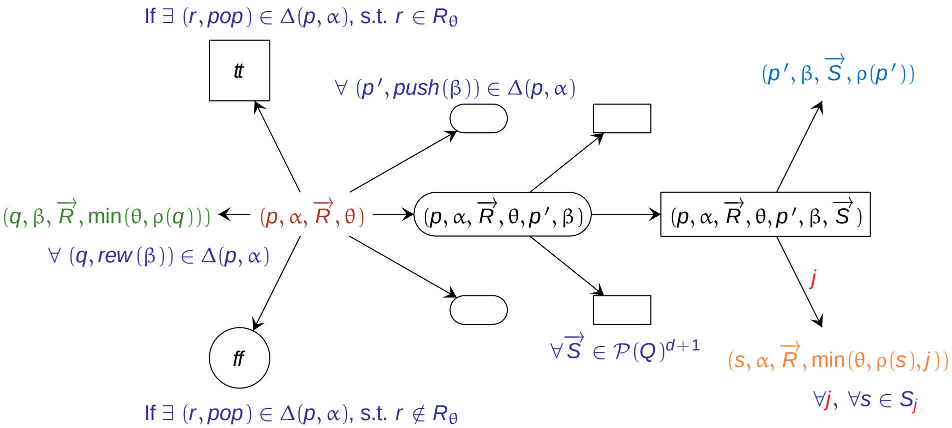
Special Case of Higher-Order Pushdown Games



Remark

If the abstract pushdown automaton is an order-n pushdown automaton then one always has $\alpha = \beta$. Hence, the new game is an order-$(n-1)$ pushdown game.

Special Case of Higher-Order Pushdown Games



Theorem

*Deciding whether Eve wins in an order-n pushdown parity game is n-*EXPTIME*-complete.*

How to Solve a CPDA Game?

Input: an order-*n* CPDA game

```
k=n
Do
```

(1) Get rid of the k-links
(2) Build the simulation game using the reduction for abstract pushdown games

```
k=k-1
Until k=0
```

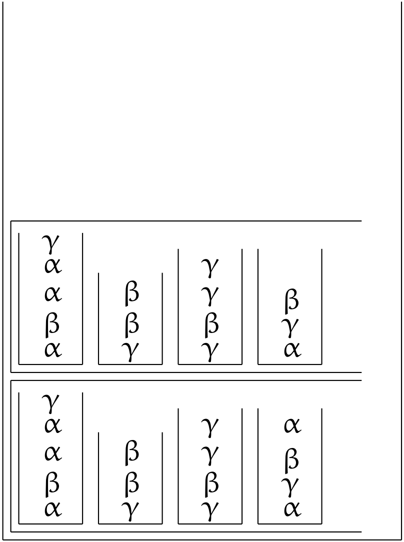
A Simulation Game to Get Rid of the n-links

The plays goes as in the original game...

q_{in},

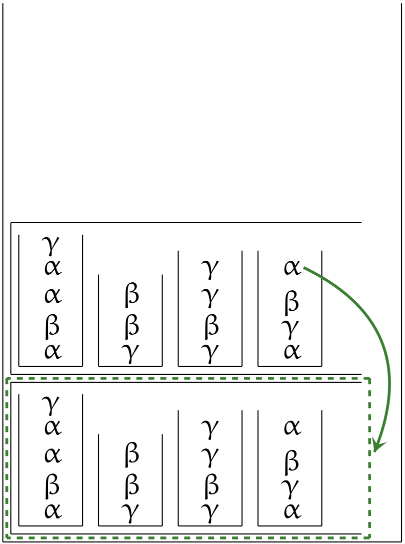
A Simulation Game to Get Rid of the n-links

The plays goes as in the original game...



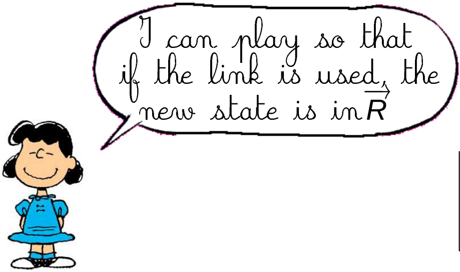
A Simulation Game to Get Rid of the n-links

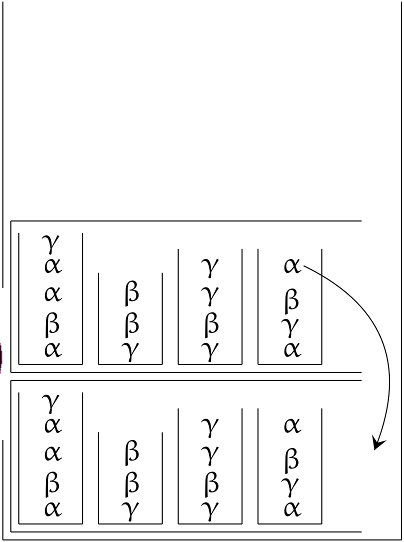
The plays goes as in the original game...except when a $push_1^n(\alpha)$ occurs



A Simulation Game to Get Rid of the n-links

The plays goes as in the original game. . . except when a $push_1^n(\alpha)$ occurs

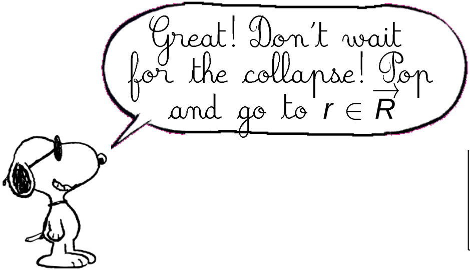


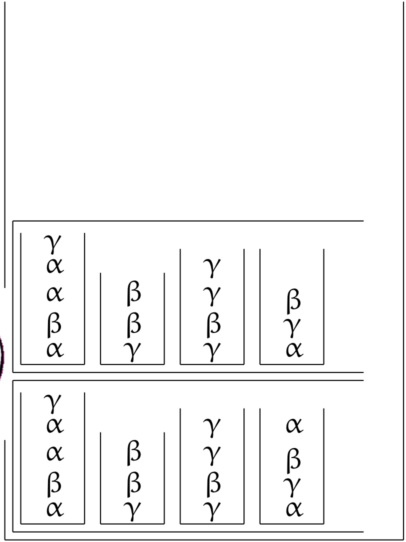


A Simulation Game to Get Rid of the n-links

The plays goes as in the original game. . .except when a $push_1^n(\alpha)$ occurs

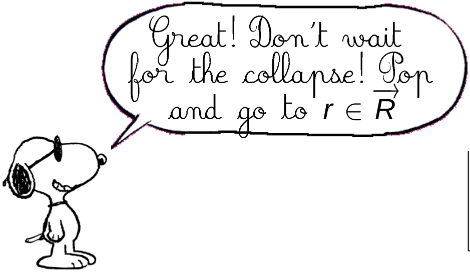
First case:

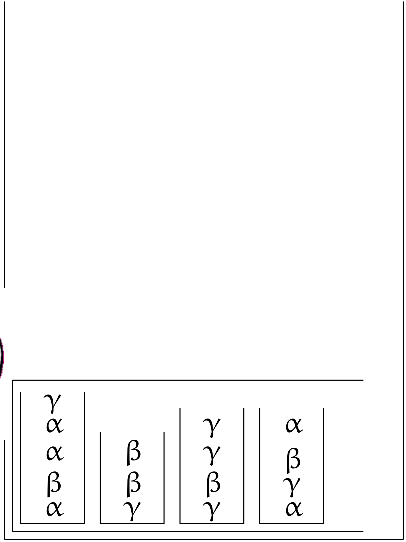




The plays goes as in the original game. . . except when a $push_1^n(\alpha)$ occurs

First case:

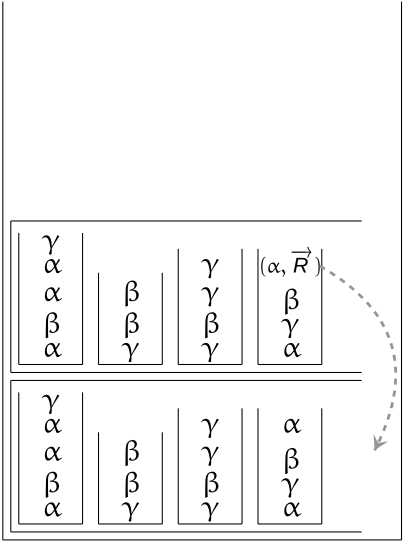




A Simulation Game to Get Rid of the n-links

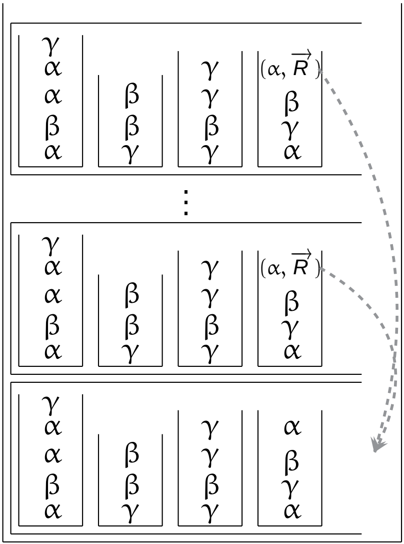
The plays goes as in the original game. . . except when a $push_1^n(\alpha)$ occurs

Second case:



A Simulation Game to Get Rid of the n-links

The plays goes as in the original game. . . except when a $push_1^n(\alpha)$ occurs

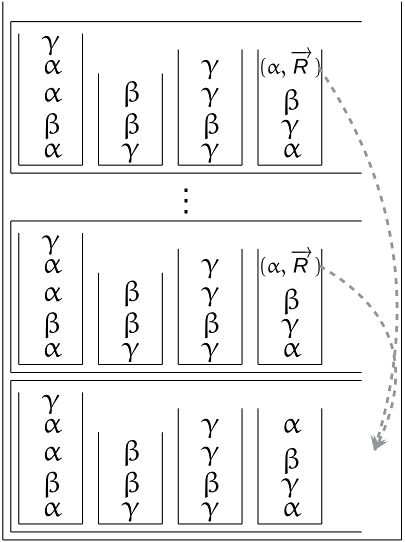


A Simulation Game to Get Rid of the n-links

The plays goes as in the original game. . . except when a $push_1^n(\alpha)$ occurs

When simulating a collapse (involving an n-link).

- ⋆ If the state reached is consistent with $\overrightarrow{R}$, Eve wins.
- ⋆ Otherwise Adam wins.

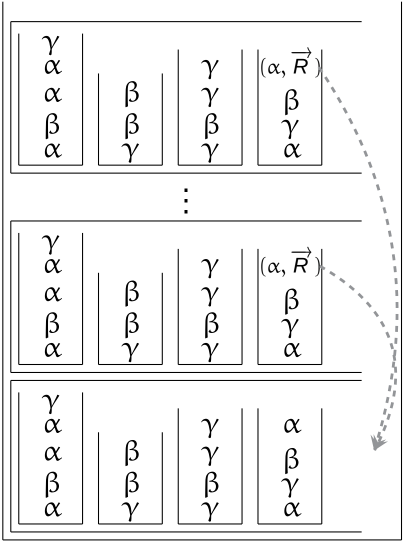


The plays goes as in the original game. . .except when a $push_1^n(\alpha)$ occurs

When simulating a collapse (involving an n-link).

- ⋆ If the state reached is consistent with $\overrightarrow{R}$, Eve wins.
- ⋆ Otherwise Adam wins.

How to check consistency???



A Simulation Game to Get Rid of the n-links

Definition

An n CPDA is rank-aware if it stores in its control state the smallest colour visited since the creation of the n-link (if exists) in the top_1 element.

Lemma

For every CPDA one can construct an "equivalent" rank-aware CPDA.

How to Solve a CPDA Game?

Input: an order-n CPDA game

```
k=n
Do
```

(0) Make the underlying CPDA rank-aware
(1) Get rid of the k-links
(2) Build the simulation game using the reduction for abstract pushdown games

```
    k=k-1
Until k=0
```

Theorem

Deciding whether Eve wins in an order-n CPDA parity game is n-EXPTIME-complete.

Complexity Issues

Theorem

The overall complexity of deciding the winner in an n-CPDA parity game is:

- *n-times exponential in the number of states of the CPDA;*
- $(n+1)$*-times exponential in the number of colours;*
- *polynomial in the stack alphabet of the CPDA.*

Complexity Issues

Theorem

The overall complexity of deciding the winner in an n-CPDA parity game is:

- *n-times exponential in the number of states of the CPDA;*
- $(n+1)$*-times exponential in the number of colours;*
- *polynomial in the stack alphabet of the CPDA.*

Corollary

Model-checking μ*-calculus against a scheme is polynomial if one bounds the alternation depth of the formula, the order of the scheme and the arity of the tree generated by it.*

Conclusion

Summary

- Recursion schemes and CPDA are equi-expressive for generating trees.
- Algorithmic on CPDA (in particular games) are useful to answer questions on schemes. In particular because CPDA come with a lot of structure that allows precise analysis.
- Correspondence between logic and games:
  - Model-checking $\Leftrightarrow$ Deciding the winner;
  - Global Model-checking + Reflection $\Leftrightarrow$ Computing the winning region + embedding a description of it into the model of CPDA;
  - Selection $\Leftrightarrow$ Computing a winning strategy + embedding a description of it into the model of CPDA.

Extra, Perspectives

Not discussed here:

- On transition graphs of CPDA, the logical story is not that nice…
- CPDA are useful to understand safety betterly

Perspectives (work in progress)

- What can you do directly on schemes?
- Move to a practice with a toy language
- And much more…