

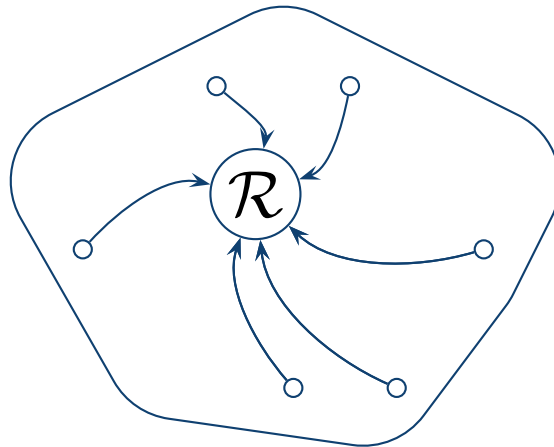
Symbolic Backwards Reachability Analysis of Higher Order Pushdown Systems

M. Hague and C.-H. L. Ong

August 11th 2009

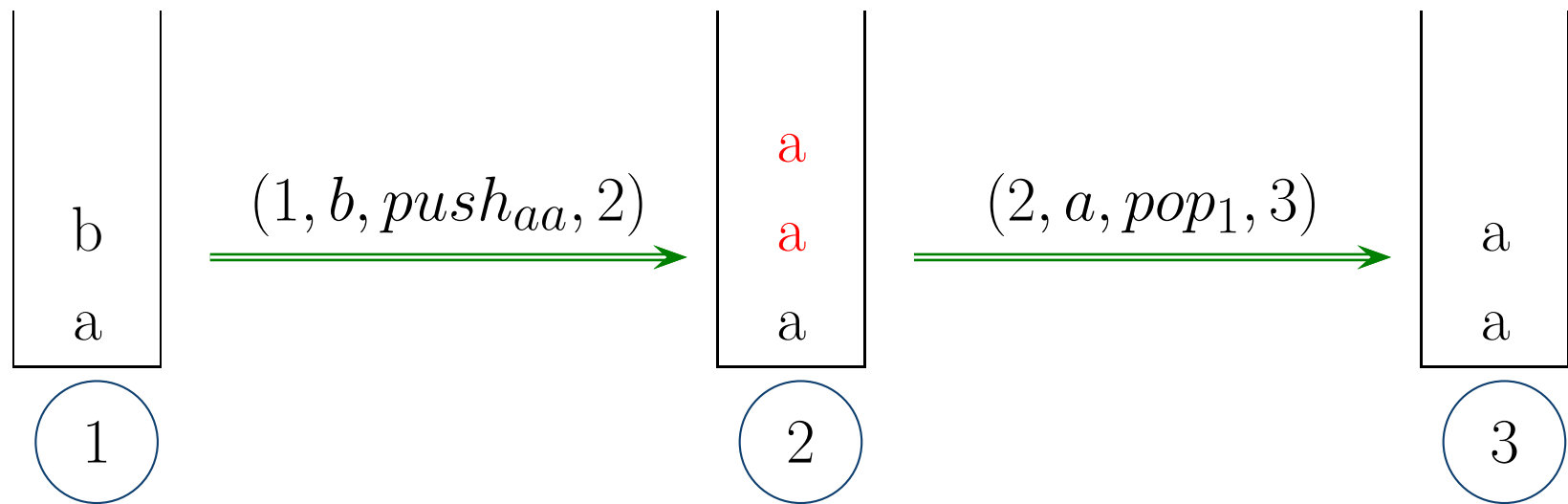
Abstract

- Pushdown systems — finite state + stack.
- Higher-order pushdown systems – stacks of stacks, &c.
- Backwards reachability analysis:



Pushdown Systems

Control states + order-one stack.



More behaviors than finite-state: $1^n 2^n \dots$

Note $pop_1 = push_\epsilon$.

Modelling Recursion

```
1: int fac(int n) {  
2:   if (n == 0)  
3:     return 1  
4:   else  
5:     return n * fac(n-1)  
6: }
```

▷ 7: fac(2)

(line = 7)

ret = ?

Modelling Recursion

```
1: int fac(int n) {  
▶ 2:   if (n == 0)  
3:     return 1  
4:   else  
5:     return n * fac(n-1)  
6: }
```

```
7: fac(2)
```

(line = 2, n = 2)
(line = 7)

ret = ?

Modelling Recursion

```
1: int fac(int n) {  
2:   if (n == 0)  
3:     return 1  
4:   else  
5:     return n * fac(n-1)  
6: }
```



7: fac(2)

(line = 5, n = 2)
(line = 7)

ret = ?

Modelling Recursion

```
1: int fac(int n) {  
▶ 2:   if (n == 0)  
3:     return 1  
4:   else  
5:     return n * fac(n-1)  
6: }
```

```
7: fac(2)
```

(line = 2, n = 1)
(line = 5, n = 2)
(line = 7)

ret = ?

Modelling Recursion

```
1: int fac(int n) {  
2:   if (n == 0)  
3:     return 1  
4:   else  
5:     return n * fac(n-1)  
6: }
```

7: fac(2)

(line = 5, n = 1)
(line = 5, n = 2)
(line = 7)

ret = ?

Modelling Recursion

```
1: int fac(int n) {  
▶ 2:   if (n == 0)  
3:     return 1  
4:   else  
5:     return n * fac(n-1)  
6: }
```

```
7: fac(2)
```

(line = 2, n = 0)

(line = 5, n = 1)

(line = 5, n = 2)

(line = 7)

ret = ?

Modelling Recursion

```
1: int fac(int n) {  
2:   if (n == 0)  
3:     return 1  
4:   else  
5:     return n * fac(n-1)  
6: }
```



7: fac(2)

(line = 3, n = 0)
(line = 5, n = 1)
(line = 5, n = 2)
(line = 7)

ret = ?

Modelling Recursion

```
1: int fac(int n) {  
2:   if (n == 0)  
3:     return 1  
4:   else  
5:     return n * fac(n-1)  
6: }
```



7: fac(2)

(line = 5, n = 1)
(line = 5, n = 2)
(line = 7)

ret = 1

Modelling Recursion

```
1: int fac(int n) {  
2:   if (n == 0)  
3:     return 1  
4:   else  
5:     return n * fac(n-1)  
6: }
```



7: fac(2)

(line = 5, n = 2)
(line = 7)

ret = 1

Modelling Recursion

```
1: int fac(int n) {  
2:   if (n == 0)  
3:     return 1  
4:   else  
5:     return n * fac(n-1)  
6: }
```

▷ 7: fac(2)

(line = 7)

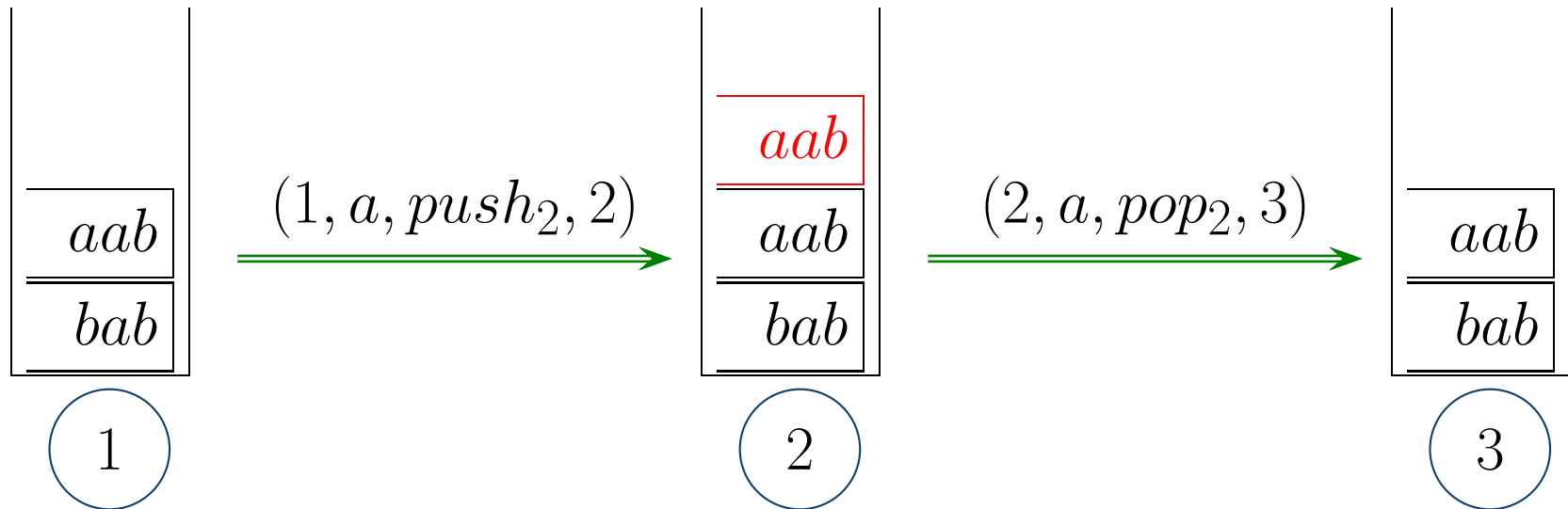
ret = 2

Model-Checking Pushdown Systems

	Local	Global
Reachability		Bouajjani <i>et al.</i> , 1997
Büchi		Cachat, 2003
Parity	Walukiewicz, 1996	Cachat, 2003 Serre, 2004 Vardi <i>et al.</i> , 2004 Hague <i>et al.</i> , 2009
Extensions	Many	Many

Higher-Order Pushdown Systems

Control state + order- n stack.

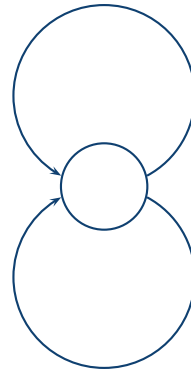


More behaviors than order-1: $1^n 2^n 3^n \dots$

Order- n PDSs form a strict hierarchy.

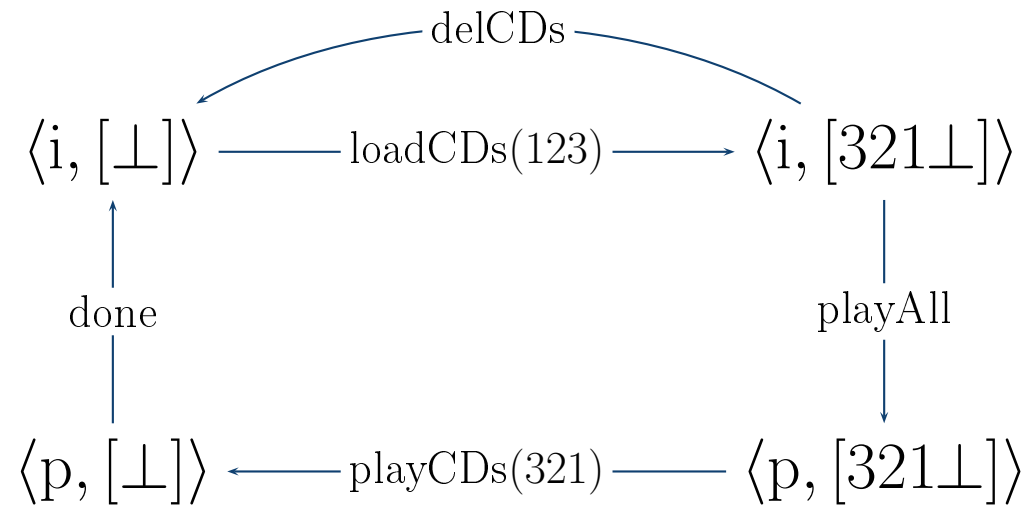
A Finite State CD Player

playCD(1)

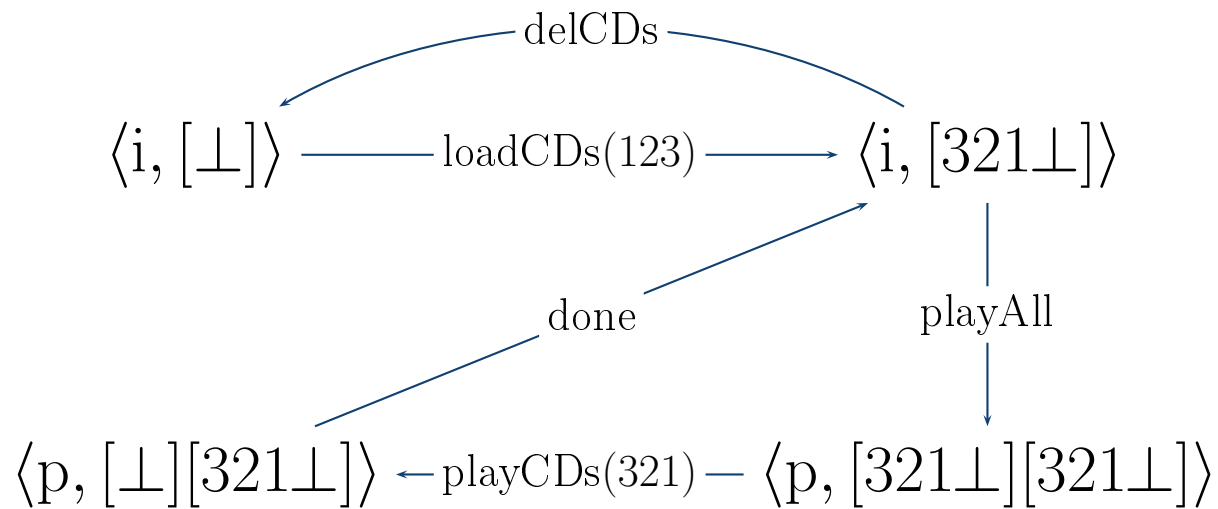


playCD(2)

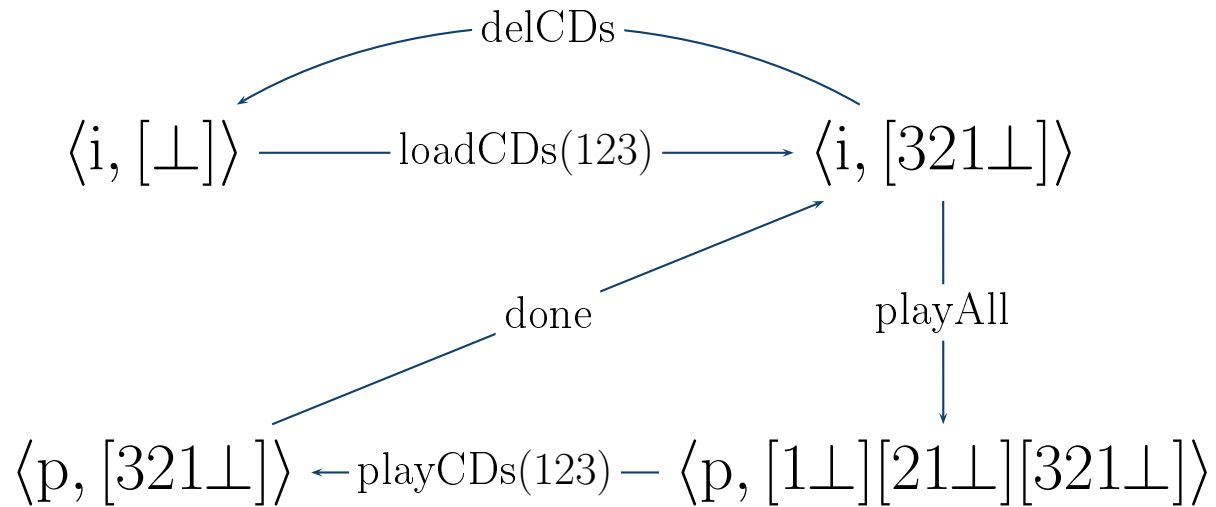
An Order-1 CD Player



An Order-2 CD Player



An Order-2 CD Player (Improved)



Higher-Order Pushdown Systems

Higher-Order Pushdown Systems are **robust**:

- The **Caucal hierarchy**. [Carayol, Wöhrle, 2003].
- **Higher-Order Recursion Schemes** satisfying the **safety** constraint [Knapik *et al.*, 2002].
- Decidable MSO theories [Knapik *et al.*, 2002].

Higher-order recursion schemes look like higher-order programs:

$$S = a(Fb) \quad Fx = x(GFa) \quad G\varphi x = x(\varphi a)$$

Evaluation:

$$\underline{S} \rightarrow$$

Higher-Order Pushdown Systems

Higher-Order Pushdown Systems are **robust**:

- The **Caucal hierarchy**. [Carayol, Wöhrle, 2003].
- **Higher-Order Recursion Schemes** satisfying the **safety** constraint [Knapik *et al.*, 2002].
- Decidable MSO theories [Knapik *et al.*, 2002].

Higher-order recursion schemes look like higher-order programs:

$$S = a(Fb) \quad Fx = x(GFa) \quad G\varphi x = x(\varphi a)$$

Evaluation:

$$S \rightarrow \underline{a(Fb)} \rightarrow$$

Higher-Order Pushdown Systems

Higher-Order Pushdown Systems are **robust**:

- The **Caucal hierarchy**. [Carayol, Wöhrle, 2003].
- **Higher-Order Recursion Schemes** satisfying the **safety** constraint [Knapik *et al.*, 2002].
- Decidable MSO theories [Knapik *et al.*, 2002].

Higher-order recursion schemes look like higher-order programs:

$$S = a(Fb) \quad Fx = x(GFa) \quad G\varphi x = x(\varphi a)$$

Evaluation:

$$S \rightarrow a(Fb) \rightarrow \underline{ab(GFa)} \rightarrow$$

Higher-Order Pushdown Systems

Higher-Order Pushdown Systems are **robust**:

- The **Caucal hierarchy**. [Carayol, Wöhrle, 2003].
- **Higher-Order Recursion Schemes** satisfying the **safety** constraint [Knapik *et al.*, 2002].
- Decidable MSO theories [Knapik *et al.*, 2002].

Higher-order recursion schemes look like higher-order programs:

$$S = a(Fb) \quad Fx = x(GFa) \quad G\varphi x = x(\varphi a)$$

Evaluation:

$$S \rightarrow a(Fb) \rightarrow ab(GFa) \rightarrow aba(\underline{Fb}) \rightarrow$$

Higher-Order Pushdown Systems

Higher-Order Pushdown Systems are **robust**:

- The **Caucal hierarchy**. [Carayol, Wöhrle, 2003].
- **Higher-Order Recursion Schemes** satisfying the **safety** constraint [Knapik *et al.*, 2002].
- Decidable MSO theories [Knapik *et al.*, 2002].

Higher-order recursion schemes look like higher-order programs:

$$S = a(Fb) \quad Fx = x(GFa) \quad G\varphi x = x(\varphi a)$$

Evaluation:

$$S \rightarrow a(Fb) \rightarrow ab(GFa) \rightarrow aba(Fb) \rightarrow abab(\underline{GFa}) \rightarrow \dots$$

Higher-Order Model Checking

- Higher-order languages have applications in systems design:
 - Ideal for **concurrency** [Edward Lee].
 - Cataldo shows necessarily $2^{2^{\dots}}$ order-0 designs, with corresponding **linear** higher-order designs.
- Research at Microsoft verifying **C#** and security properties in **F#** [Müller, Ruskiewicz, 2007][Gordon *et al.*, 2006].
- Order-2 pushdown automata generate the **indexed languages**, which have applications in **NLP** [Aho, 1968][Gazdar, 1988].
- SPIN has been used to identify errors in NASA remote agents written in **LISP**.

Model-Checking HOPDS

	Local	Global
Reachability (One State)		Meyer <i>et al.</i> , 2005
Reachability		Hague, 2007
Parity	Cachat, 2003	Serre <i>et al.</i> , 2008 Seth

Reachability Over PDS

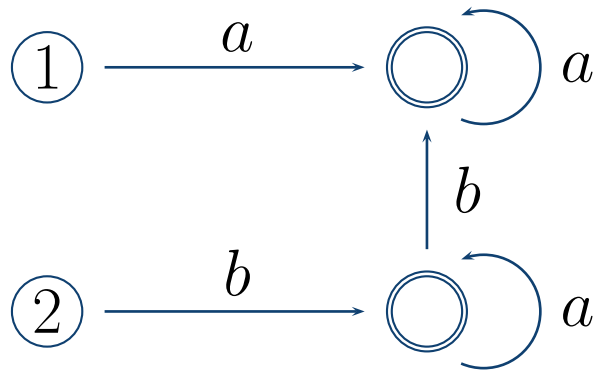
Saturation technique introduced by Bouajjani *et al*, 1997.

- Backwards reachability.
- Sets of configurations represented by a finite automaton.
 - An initial state for every control state.
 - Stack contents represented by runs.
- Transitions added according to the pushdown rules.
- Termination occurs when no new transitions are added.

* The technique is also due to Finkel *et al.*, 1997 and Book and Otto, 1993.

Reachability Over PDS

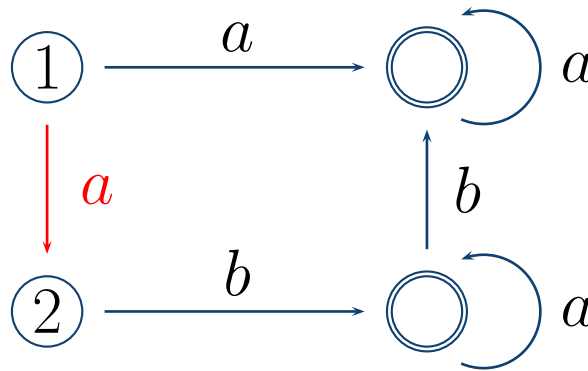
Determine the set of configurations that can reach \mathcal{R} .



$$\langle 1, aa^* \rangle, \langle 2, ba^* \rangle, \langle 2, ba^*ba^* \rangle$$

Reachability Over PDS

Determine the set of configurations that can reach \mathcal{R} .

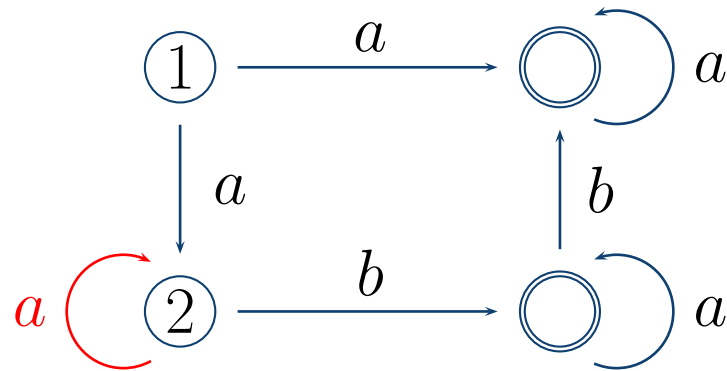


$(1, a, pop_1, 2)$

$\langle 1, a b a^* \rangle \hookrightarrow \langle 2, b a^* \rangle$

Reachability Over PDS

Determine the set of configurations that can reach \mathcal{R} .

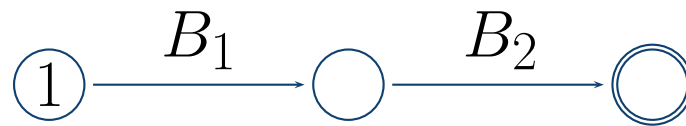


$$(2, a, pop_1, 2)$$

$$\langle 2, a^*ba^* \rangle \hookrightarrow^* \langle 2, aba^* \rangle \hookrightarrow \langle 2, ba^* \rangle$$

Order-2 PDSs (Single Control State)

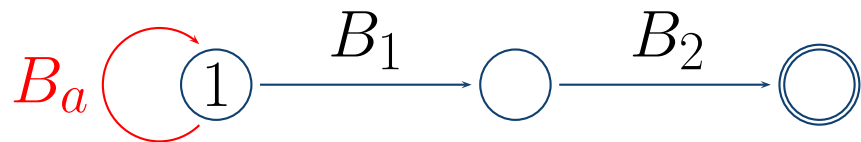
Nested store automata — transitions labelled by automata accepting order-1 stacks [Bouajjani, Meyer, 2004].



$$\langle 1, [u][v] \rangle \quad u \in \mathcal{L}(B_1), v \in \mathcal{L}(B_2)$$

Order-2 PDSs (Single Control State)

Nested store automata — transitions labelled by automata accepting order-1 stacks [Bouajjani, Meyer, 2004].

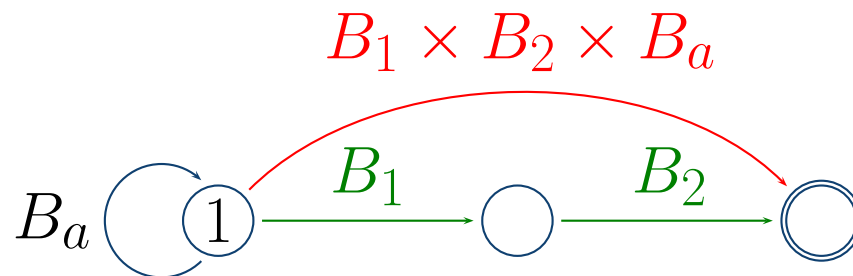


(a, pop_2)

$$\langle 1, [aw][u][v] \rangle \hookrightarrow \langle 1, [u][v] \rangle$$

Order-2 PDSs (Single Control State)

Nested store automata — transitions labelled by automata accepting order-1 stacks [Bouajjani, Meyer, 2004].

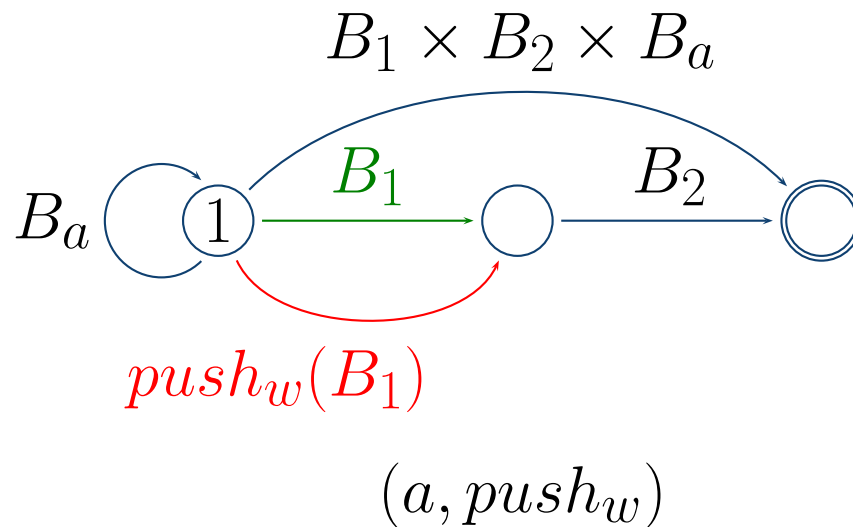


$(a, push_2)$

$\langle 1, [w] \rangle \hookrightarrow \langle 1, [w][w] \rangle$

Order-2 PDSs (Single Control State)

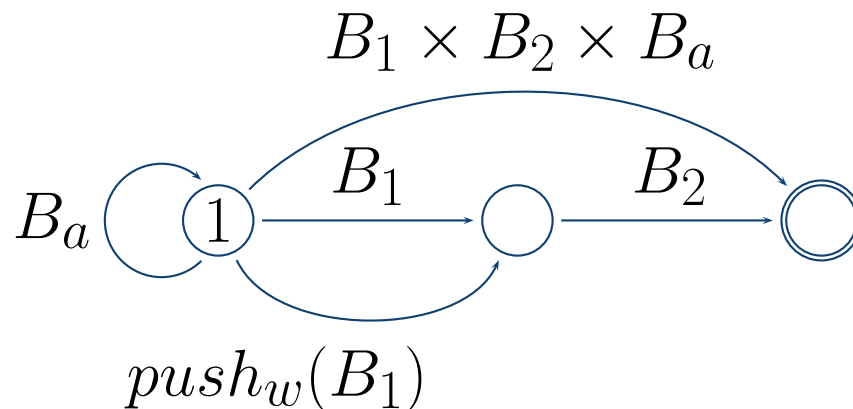
Nested store automata — transitions labelled by automata accepting order-1 stacks [Bouajjani, Meyer, 2004].



$$\langle 1, [au][v] \rangle \hookrightarrow \langle 1, [wu][v] \rangle$$

Order-2 PDSs (Single Control State)

Nested store automata — transitions labelled by automata accepting order-1 stacks [Bouajjani, Meyer, 2004].

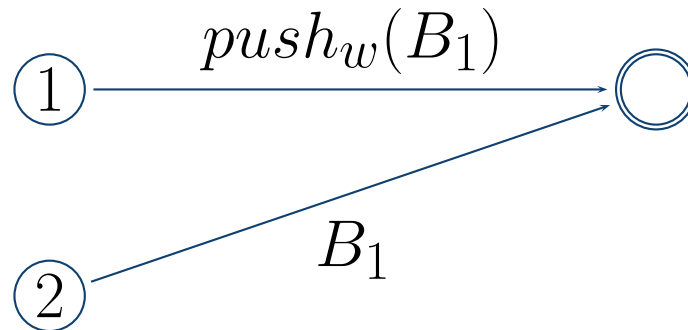


Finite number of order-1 automata wrt state-space,

$$Q_a \times Q_1 \times Q_2$$

Multiple Control States

A command $(1, a, push_w, 2)$.



Let $\mathcal{L}(B_1) = w$ $\langle 2, [w] \rangle$

We require $\mathcal{L}(push_w(B_1)) = a$ $\langle 1, [a] \rangle \hookrightarrow \langle 2, [w] \rangle$

But $\mathcal{L}(push_w(B_1)) = a \cup w$ $\langle 1, [w] \rangle \hookrightarrow ???$

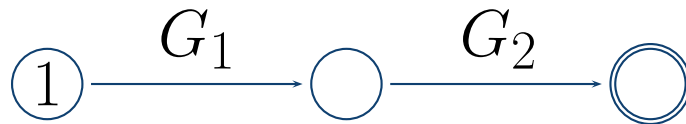
Adding new states breaks the termination argument.

Multiple Control States

Commands $(1, a, pop_1, 1), (1, a, push_2, 1)$.

$$G_1 = B_1$$

$$G_2 = B_2$$



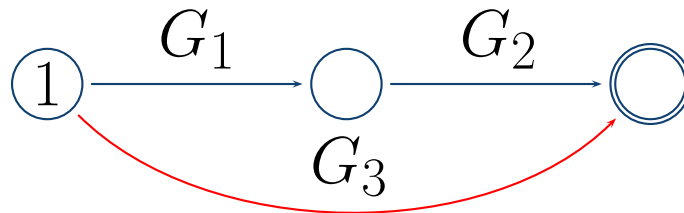
Multiple Control States

Commands $(1, a, pop_1, 1), (1, a, push_2, 1)$.

$$G_1 = B_1 \mid \text{pop}_1(G_1), B_1$$

$$G_2 = B_2 \mid B_2$$

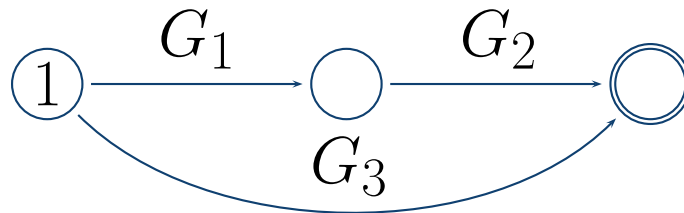
$$G_3 = \mid G_1 \times G_2$$



Multiple Control States

Commands $(1, a, pop_1, 1), (1, a, push_2, 1)$.

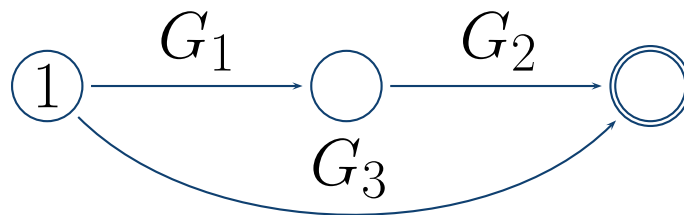
$$\begin{array}{l}
 G_1 = B_1 \mid pop_1(G_1), B_1 \mid pop_1(G_1), B_1 \\
 G_2 = B_2 \mid B_2 \qquad \qquad \qquad B_2 \\
 G_3 = \qquad \mid G_1 \times G_2 \qquad \qquad \mid pop_1(G_3), G_1 \times G_2
 \end{array}$$



Multiple Control States

Commands $(1, a, pop_1, 1), (1, a, push_2, 1)$.

$$\begin{array}{l}
 G_1 = \left[\begin{array}{c|c} pop_1(G_1), B_1 & pop_1(G_1), B_1 \\ \hline \dots & \dots \end{array} \right. \\
 G_2 = \left[\begin{array}{c|c} B_2 & B_2 \\ \hline \dots & \dots \end{array} \right. \\
 G_3 = \left[\begin{array}{c|c} pop_1(G_3), G_1 \times G_2 & pop_1(G_3), G_1 \times G_2 \\ \hline \dots & \dots \end{array} \right.
 \end{array}$$



Constructing G_1, G_2 and G_3

$$\begin{array}{l} G_1 = B_1 \mid \text{pop}_1(G_1), B_1 \mid \text{pop}_1(G_1), B_1 \\ G_2 = B_2 \mid B_2 \qquad \qquad \qquad \mid B_2 \\ G_3 = \qquad \mid G_1 \times G_2 \qquad \qquad \mid \text{pop}_1(G_3), G_1 \times G_2 \end{array}$$

These instructions can be seen as pushdown systems.

$$\begin{array}{l} \alpha_1 : G_1 = B_1 \\ \qquad G_2 = B_2 \\ \qquad G_3 = \end{array}$$

Constructing G_1, G_2 and G_3

$$\begin{array}{l}
 G_1 = B_1 \mid \text{pop}_1(G_1), B_1 \mid \text{pop}_1(G_1), B_1 \\
 G_2 = B_2 \mid B_2 \qquad \qquad \qquad \mid B_2 \\
 G_3 = \qquad \mid G_1 \times G_2 \qquad \qquad \mid \text{pop}_1(G_3), G_1 \times G_2
 \end{array}$$

These instructions can be seen as pushdown systems.

$$\alpha_1 : \left. \begin{array}{l} G_1 = B_1 \\ G_2 = B_2 \\ G_3 = \end{array} \right\} \begin{array}{l} (G_1, \text{skip}, B_1) \\ (G_2, \text{skip}, B_2) \end{array}$$

Constructing G_1, G_2 and G_3

$$\begin{array}{l}
 G_1 = B_1 \mid \text{pop}_1(G_1), B_1 \mid \text{pop}_1(G_1), B_1 \\
 G_2 = B_2 \mid B_2 \qquad \qquad \qquad \mid B_2 \\
 G_3 = \qquad \mid G_1 \times G_2 \qquad \qquad \mid \text{pop}_1(G_3), G_1 \times G_2
 \end{array}$$

These instructions can be seen as pushdown systems.

$$\alpha_1 : \left. \begin{array}{l} G_1 = B_1 \\ G_2 = B_2 \\ G_3 = \end{array} \right\} \begin{array}{l} (G_1, \text{skip}, B_1) \\ (G_2, \text{skip}, B_2) \end{array}$$

From which we compute

$$\boxed{Pre_{\alpha_1}(B_1 \uplus B_2)}$$

that has states for G_1, G_2 and G_3 .

Constructing G_1, G_2 and G_3

$$\begin{aligned} G_1 &= B_1 \mid \text{pop}_1(G_1), B_1 \mid \text{pop}_1(G_1), B_1 \\ G_2 &= B_2 \mid B_2 \qquad \qquad \qquad \mid B_2 \\ G_3 &= \qquad \mid G_1 \times G_2 \qquad \qquad \mid \text{pop}_1(G_3), G_1 \times G_2 \end{aligned}$$

These instructions can be seen as pushdown systems.

$$Pre_{\alpha_1}(B_1 \uplus B_2)$$

Constructing G_1, G_2 and G_3

$$\begin{array}{l} G_1 = B_1 \mid \text{pop}_1(G_1), B_1 \mid \text{pop}_1(G_1), B_1 \\ G_2 = B_2 \mid B_2 \mid B_2 \\ G_3 = \mid G_1 \times G_2 \mid \text{pop}_1(G_3), G_1 \times G_2 \end{array}$$

These instructions can be seen as pushdown systems.

$$\begin{array}{l} \alpha_2 : G_1 = \text{pop}_1(G_1) \\ \quad B_1 \\ G_2 = B_2 \\ G_3 = G_1 \times G_2 \end{array}$$

$$\text{Pre}_{\alpha_1}(B_1 \uplus B_2)$$

Constructing G_1, G_2 and G_3

$$\begin{array}{l}
 G_1 = B_1 \mid \text{pop}_1(G_1), B_1 \mid \text{pop}_1(G_1), B_1 \\
 G_2 = B_2 \mid B_2 \mid B_2 \\
 G_3 = \mid G_1 \times G_2 \mid \text{pop}_1(G_3), G_1 \times G_2
 \end{array}$$

These instructions can be seen as pushdown systems.

$$\alpha_2 : \left. \begin{array}{l}
 G_1 = \text{pop}_1(G_1) \\
 B_1 \\
 G_2 = B_2 \\
 G_3 = G_1 \times G_2
 \end{array} \right\} \begin{array}{l}
 (G_1, \text{pop}_1, G_1) \\
 (G_1, \text{skip}, B_1) \\
 (G_2, \text{skip}, B_2) \\
 (G_3, \text{skip}, \{G_1, G_2\})
 \end{array}$$

$$\text{Pre}_{\alpha_1}(B_1 \uplus B_2)$$

Constructing G_1, G_2 and G_3

$$\begin{array}{l}
 G_1 = B_1 \mid \text{pop}_1(G_1), B_1 \mid \text{pop}_1(G_1), B_1 \\
 G_2 = B_2 \mid B_2 \mid B_2 \\
 G_3 = \mid G_1 \times G_2 \mid \text{pop}_1(G_3), G_1 \times G_2
 \end{array}$$

These instructions can be seen as pushdown systems.

$$\alpha_2 : \left. \begin{array}{l}
 G_1 = \text{pop}_1(G_1) \\
 B_1 \\
 G_2 = B_2 \\
 G_3 = G_1 \times G_2
 \end{array} \right\} \begin{array}{l}
 (G_1, \text{pop}_1, G_1) \\
 (G_1, \text{skip}, B_1) \\
 (G_2, \text{skip}, B_2) \\
 (G_3, \text{skip}, \{G_1, G_2\})
 \end{array}$$

$$\text{Pre}_{\alpha_2}(\text{Pre}_{\alpha_1}(B_1 \uplus B_2))$$

Constructing G_1, G_2 and G_3

$$\begin{array}{l}
 G_1 = B_1 \mid \text{pop}_1(G_1), B_1 \mid \text{pop}_1(G_1), B_1 \\
 G_2 = B_2 \mid B_2 \qquad \qquad \qquad \mid B_2 \\
 G_3 = \qquad \mid G_1 \times G_2 \qquad \qquad \mid \text{pop}_1(G_3), G_1 \times G_2
 \end{array}$$

These instructions can be seen as pushdown systems.

$$\alpha_3 : \quad \dots \quad \} \quad \dots$$

$$\text{Pre}_{\alpha_3} (\text{Pre}_{\alpha_2} (\text{Pre}_{\alpha_1} (B_1 \uplus B_2)))$$

Constructing G_1, G_2 and G_3

$$\begin{array}{l} G_1 = \quad \left. \begin{array}{l} \text{pop}_1(G_1), B_1 \\ \text{pop}_1(G_1), B_1 \end{array} \right\} \\ G_2 = \dots \left. \begin{array}{l} B_2 \\ B_2 \end{array} \right\} \dots \\ G_3 = \quad \left. \begin{array}{l} \text{pop}_1(G_3), G_1 \times G_2 \\ \text{pop}_1(G_3), G_1 \times G_2 \end{array} \right\} \end{array}$$

These instructions can be seen as pushdown systems.

$$\alpha_3 = \alpha_4 = \alpha_5 = \alpha_6 = \dots$$

$$Pre_{\alpha_3} (Pre_{\alpha_2} (Pre_{\alpha_1} (B_1 \uplus B_2)))$$

Constructing G_1, G_2 and G_3

$$\begin{array}{l} G_1 = \quad \left. \begin{array}{l} \text{pop}_1(G_1), B_1 \\ \text{pop}_1(G_1), B_1 \end{array} \right\} \\ G_2 = \dots \left. \begin{array}{l} B_2 \\ B_2 \end{array} \right\} \dots \\ G_3 = \quad \left. \begin{array}{l} \text{pop}_1(G_3), G_1 \times G_2 \\ \text{pop}_1(G_3), G_1 \times G_2 \end{array} \right\} \end{array}$$

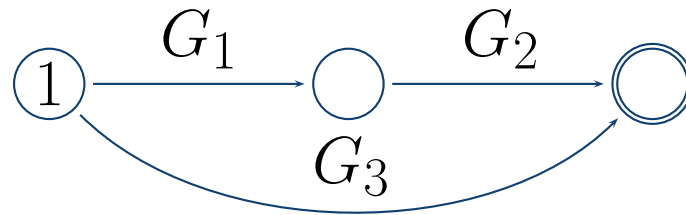
These instructions can be seen as pushdown systems.

$$\alpha_3 = \alpha_4 = \alpha_5 = \alpha_6 = \dots$$

$$Pre_{\alpha_3}^* (Pre_{\alpha_2} (Pre_{\alpha_1} (B_1 \uplus B_2)))$$

Completing the Construction

We construct



with the values of G_1 , G_2 and G_3 taken from

$$Pre_{\alpha_3}^* (Pre_{\alpha_2} (Pre_{\alpha_1} (B_1 \uplus B_2)))$$

Applications

Higher-order pushdown systems:

- Model checking linear time properties.
- Model checking alternation-free μ -Calculus.

Games over higher-order systems:

- Reachability games.

Higher-order pushdown automata:

- Non-emptiness
 - Shows n -EXPTIME-completeness of reachability.

Recent Developments

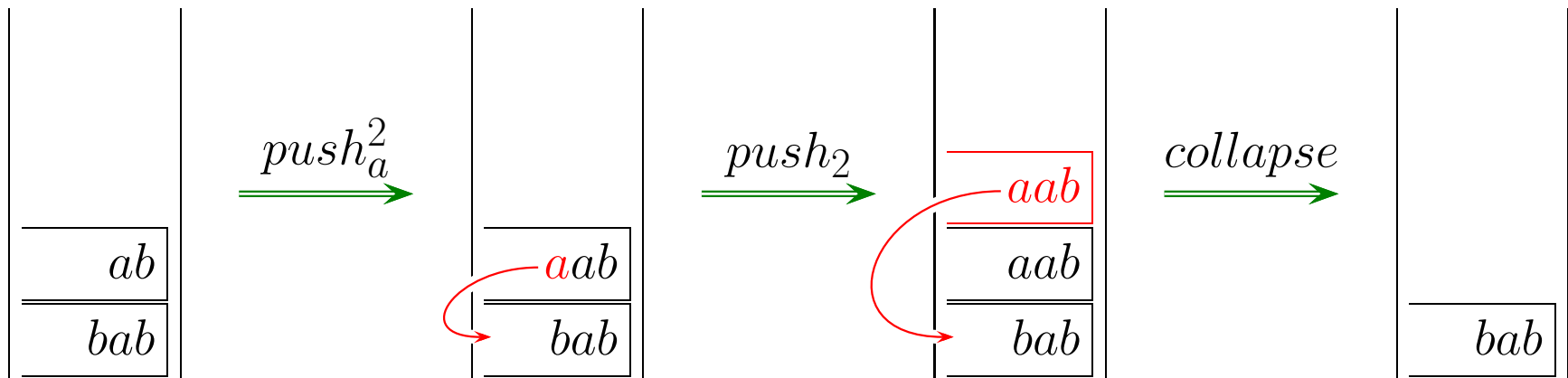
Parity games over order-1 PDS:

- An extension of the saturation method [Hague, Ong, 2009].

Parity games over higher-order PDS:

- [Carayol *et al.*, 2008]
 - An extension of Serre's algorithm.
 - An extension of Vardi's algorithm.
- Seth extended Walukiewicz's approach using higher-order functions.

Collapsible Pushdown Systems



[Ong *et al.*, 2008]

- Recursion schemes without safety.
- Parity games (local).

Parity games (global) [Broadbent, 2009].

Type theoretic proof [Kobayashi, Ong, 2009].

Conclusions and Future Work

Conclusions:

- An (optimal) n -EXPTIME algorithm for reachability games over higher-order pushdown systems.
- Applications to model-checking.

Future Work:

- Parity games.
- Implementation / alternative techniques.
 - Other notions of regularity [Carayol, 2004].
 - Grammars.
- Extensions of pushdown systems.

Current Work

Parity games **implementation**.

- extension to second-order reachability.

MSO characterisation of higher-order pushdown languages.

- Connection with **nested trees**.
- Difficulties with complementation.

Stochastic higher-order pushdown systems.

- Polynomials?

Lower bounds for temporal logics (LTL, CTL, &c.).

- $(n - 1)$ -EXPTIME or n -EXPTIME?