



# Verifying Finitely-Presentable Infinite Structures: A Game-Semantic Approach

Luke Ong

University of Oxford

[Acknowledgements: Jolie de Miranda, Klaus Aehlig, Andrzej Murawski,  
Matthew Hague + Olivier Serre]

10 August 2006

## Motivation

- Verification and game semantics
- Overview 1
- Overview
- Outline

A Tree Hierarchy  
Generated By  
Recursion Schemes

MSO Model-Checking  
Problem

Infinite Structures With  
Decidable MSO  
Theories

Deciding MSO Theories  
of Trees

Collapsible Pushdown  
Automata (with  
A. Murawski)

A Hierarchy of Graphs

# Motivation

## Verification and (Game) Semantics

### What is the Model Checking approach to Verification?

Given a **system** (e.g. lift controller, operating system) and a desired **property** (e.g. deadlock freedom, liveness) of the system:

1. Construct an abstract model  $M$  of the system.
2. Describe the property as a formula  $\varphi$  in some logic  $\mathcal{L}$ .
3. Exhaustively check the model  $M$  for violation of  $\varphi$ .

Extremely successful in verifying relatively “flat, unstructured” finite-state processes (e.g. protocols, circuits); less effective when applied to software.

### Key (interdependent) semantic and algorithmic questions:

- Does  $M$  model the system **accurately**?
- Is the problem “Does  $M$  satisfy  $\varphi$ ?” **decidable**?
- Is the violation check **efficient** (or better, optimal)?

**Our approach** is to analyse basic problems in Verification using (game-)semantic methods.

## Overview: Trees generated by recursion schemes

**A Basic Problem in Model Checking:** Find classes of finitely-presentable infinite structures with decidable monadic second-order (MSO) theories.

We study the infinite hierarchy of (possibly infinite) term-trees generated by **higher-order recursion schemes** (= simply-typed lambda calculus + uninterpreted 1st-order function symbols + fixpoints).

### Why?

- Natural case study of the game-semantic approach.
- Rich and unifying tree hierarchy - subsumes major classes.
- Robust framework - admits several different characterizations.

**Theorem.** For each  $n \geq 0$ , the modal mu-calculus model checking-problem for **RecSchTree<sub>n</sub>** (i.e. trees generated by order- $n$  recursion schemes) is  $n$ -EXPTIME complete. Thus these trees have decidable MSO theories.

## Overview

### Characterising expressiveness of higher-order recursion schemes:

#### Order- $n$ Collapsible Pushdown Automata (CPDA)

- Each stack symbol in  $n$ -stack “remembers” the stack content at the point it was first created (i.e. pushed).
- **collapse** (= **panic**) collapses the  $n$ -stack up to the point as remembered by the top element of the stack.

**Theorem** (Murawski + O.). For defining trees, order- $n$  recursion schemes = order- $n$  collapsible pushdown automata, for each  $n \geq 0$ .

### A new graph hierarchy

The same game-semantic approach is just as effective a basis for model-checking configuration graphs of CPDA.

**Theorem.** For each  $n \geq 0$ , parity games over the configuration graphs of CPDA are solvable.

Many further directions, and open problems.

# Outline

## Motivation

## A Tree Hierarchy Generated By Recursion Schemes

## MSO Model-Checking Problem

## Infinite Structures With Decidable MSO Theories

## Deciding MSO Theories of Trees

## Collapsible Pushdown Automata (with A. Murawski)

## A Hierarchy of Graphs

Motivation

---

**A Tree Hierarchy  
Generated By  
Recursion Schemes**

---

- Recursion schemes
- Value tree
- An order-2 example

MSO Model-Checking  
Problem

---

Infinite Structures With  
Decidable MSO  
Theories

---

Deciding MSO Theories  
of Trees

---

Collapsible Pushdown  
Automata (with  
A. Murawski)

---

A Hierarchy of Graphs

---

# A Tree Hierarchy Generated By Recursion Schemes

---

## Order- $n$ (deterministic) recursion scheme $G = (\mathcal{N}, \Sigma, \mathcal{R}, S)$

Fix a set of typed variables (written as  $\varphi, x, y$  etc).

- $\mathcal{N}$ : Typed **non-terminals** of order at most  $n$  (written as upper-case letters), including a distinguished **start symbol**  $S : o$ .
- $\Sigma$ : **Ranked alphabet of terminals**:  $\underline{f} \in \Sigma$  has **arity**  $ar(\underline{f}) \geq 0$  which determines a first-order type  $\underline{f} : \underbrace{o \rightarrow \cdots \rightarrow o}_{ar(\underline{f})} \rightarrow o$
- $\mathcal{R}$ : An **equation** for each non-terminal  $D : A_1 \rightarrow \cdots \rightarrow A_m \rightarrow o$  of shape

$$D \varphi_1 \cdots \varphi_m = e$$

where the term  $e : o$  is constructed from

- terminals  $\underline{f}, \underline{g}, \underline{a}$ , etc. from  $\Sigma$
- variables  $\varphi_1 : A_1, \cdots, \varphi_m : A_m$  from  $Var$ ,
- non-terminals  $D, F, G$ , etc. from  $\mathcal{N}$ .

using the **application rule**: If  $s : A \rightarrow B$  and  $t : A$  then  $(st) : B$ .

## Tree Generated by a Recursion Scheme $G$

The **value tree**  $\llbracket G \rrbracket$  of a recursion scheme  $G$  is a possibly infinite applicative term *constructed from the terminals*, obtained by rewriting the equations *ad infinitum*, replacing formal by actual parameters each time, starting from  $S$ .

**Example.**  $\Sigma = \{ \underline{f}, \underline{g}, \underline{a} \}$ . Take

$$G_1 : \begin{cases} S & = & F \underline{a} \\ F x & = & \underline{f} x (F (\underline{g} x)) \end{cases}$$

We have  $\llbracket G_1 \rrbracket = \underline{f} \underline{a} (\underline{f} (\underline{g} \underline{a}) (\underline{f} (\underline{g} (\underline{g} \underline{a})) (\dots)))$ .

We view the infinite term  $\llbracket G \rrbracket$  as a  **$\Sigma$ -labelled (ranked and ordered) tree** (generated by  $G$ ).

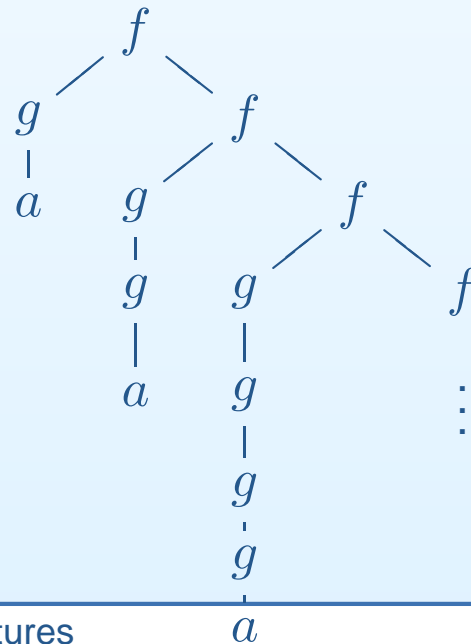
Formally a  **$\Sigma$ -labelled tree** is a function  $t : \text{dom}(t) \longrightarrow \Sigma$  such that  $\text{dom}(t) \subseteq \{1, \dots, m\}^*$  is prefix-closed, and for all nodes  $\alpha \in T$ , the  $\Sigma$ -symbol  $t(\alpha) \in \Sigma$  has arity  $k$  iff  $\alpha$  has  $k$  children, namely  $\alpha 1, \dots, \alpha k \in T$ .

## An order-2 example

$$\Sigma = \{ \underline{f}, \underline{g}, \underline{a} \}. \quad B : (o \rightarrow o) \rightarrow (o \rightarrow o) \rightarrow o \rightarrow o, \quad F : (o \rightarrow o) \rightarrow o$$

$$G_2 : \begin{cases} S & = & F g \\ B \varphi \psi x & = & \varphi (\psi x) \\ F \varphi & = & \underline{f} (\varphi \underline{a}) (F (B \varphi \varphi)) \end{cases}$$

The **value tree**,  $\llbracket G_2 \rrbracket : \{1, 2\}^* \longrightarrow \Sigma$ , is: 
$$\begin{cases} \epsilon & \mapsto & f & 11 & \mapsto & a \\ 1 & \mapsto & g & 21 & \mapsto & g \\ 2 & \mapsto & f & 22 & \mapsto & f \\ & & \dots & \dots & & \dots \end{cases}$$



## Motivation

A Tree Hierarchy  
Generated By  
Recursion Schemes

## **MSO Model-Checking Problem**

- Node-labelled trees
- MSO Logic
- Why MSO Logic?

Infinite Structures With  
Decidable MSO  
Theories

Deciding MSO Theories  
of Trees

Collapsible Pushdown  
Automata (with  
A. Murawski)

A Hierarchy of Graphs

# MSO Model-Checking Problem

## Representing $\Sigma$ -labelled trees as logical structures

Represent a  $\Sigma$ -labelled tree  $t : \text{dom}(t) \longrightarrow \Sigma$  by the tuple

$$\langle \text{dom}(t), \quad \langle \mathbf{d}_i : 1 \leq i \leq m \rangle, \quad \langle \mathbf{p}_f : f \in \Sigma \rangle \rangle$$

where

- $\text{dom}(t) \subseteq \{1, \dots, m\}^*$  with  $m = \max\{ar(f) : f \in \Sigma\}$
- **Parent-child relationship:**  $\mathbf{d}_i = \{(\alpha, \alpha i) : \alpha \in \text{dom}(t) \wedge \alpha i \in \text{dom}(t)\}$
- **Node labelling:**  $\mathbf{p}_f = \{\alpha \in \text{dom}(t) : t(\alpha) = f\}$ .

Hence given a ranked alphabet  $\Sigma$ , fix a **vocabulary** with binary predicate symbols  $\mathbf{d}_i$  where  $1 \leq i \leq$  maximum arity of  $\Sigma$ -symbols, and unary predicate symbols  $\mathbf{p}_f$ , one for each  $f \in \Sigma$ .

## Monadic Second-Order Logic (for $\Sigma$ -labelled trees)

First-order variables:  $x, y, z$ , etc. (ranging over *nodes*, which are finite words over  $\{1, \dots, m\}$ , for a fixed  $m$ )

Second-order variables:  $X, Y, Z$ , etc. (ranging over *sets* of nodes)

MSO formulas are built up from **atomic formulas**:

1. **Parent-child relationship between nodes:**  $\mathbf{d}_i(x, y) \equiv$  “ $y$  is  $i$ -child of  $x$ ”
2. **Node labelling:**  $\mathbf{p}_f(x) \equiv$  “ $x$  has label  $f$ ” where  $f$  is a  $\Sigma$ -symbol
3. **Set-membership:**  $x \in X$

and closed under boolean connectives, first-order quantification ( $\forall x. -$ ,  $\exists x. -$ ) and second-order quantifications:  $\forall X. -$ ,  $\exists X. -$ .

**RecSchTree<sub>n</sub>**:  $\Sigma$ -labelled trees generated by order- $n$  recursion schemes.

### MSO MODEL-CHECKING PROBLEM FOR **RecSchTree<sub>n</sub>**

- **INSTANCE:** An order- $n$  recursion scheme  $G$ , and an MSO formula  $\varphi$
- **QUESTION:** Does the  $\Sigma$ -labelled tree  $\llbracket G \rrbracket$  satisfy  $\varphi$ ?

## Why MSO Logic?

It is a kind of **gold standard!**

- **MSO is very expressive.** Over graphs, MSO is strictly more expressive than the modal mu-calculus, into which all standard temporal logics (e.g. LTL, CTL, CTL\*, etc.) can embed.  
Over trees, modal mu-calculus is as expressive as (but algorithmically more tractable than) MSO: For every MSO  $\varphi$ , there is a modal mu-calculus formula  $p_\varphi$  s.t. for every  $\Sigma$ -labelled tree  $t$ , we have  $t \models \varphi \iff t, \epsilon \models p_\varphi$ .
- **Any obvious extension of MSO would break decidability.** Either of the following would permit an encoding of a Turing machine:
  - Second-order quantification over binary relations.
  - Freely interpretable binary relations in the vocabulary.

E.g.  $T_a(i, t) =$  “ $i$ -th cell of the semi-infinite tape contains  $a \in \Sigma$  at time  $t$ ”.

Motivation

---

A Tree Hierarchy  
Generated By  
Recursion Schemes

---

MSO Model-Checking  
Problem

---

Infinite Structures With  
Decidable MSO  
Theories

---

- Some milestones
- What is the safety constraint?
- Two questions about safety

Deciding MSO Theories  
of Trees

---

Collapsible Pushdown  
Automata (with  
A. Murawski)

---

A Hierarchy of Graphs

---

# Infinite Structures With Decidable MSO Theories

## Structures with decidable MSO theories: some milestones

[In timeline below, each item subsumes developments in preceding items.]

1. **Rabin 1969**: Regular trees. “Mother of all decidability results”

## Structures with decidable MSO theories: some milestones

[In timeline below, each item subsumes developments in preceding items.]

1. **Rabin 1969**: Regular trees. “Mother of all decidability results”
2. **Muller and Schupp 1985**: Configuration graphs of pushdown automata.

## Structures with decidable MSO theories: some milestones

[In timeline below, each item subsumes developments in preceding items.]

1. **Rabin 1969**: Regular trees. “Mother of all decidability results”
2. **Muller and Schupp 1985**: Configuration graphs of pushdown automata.
3. **Caucal (ICALP 1996)**: Prefix-recognizable graphs ( =  $\epsilon$ -closures of configuration graphs of pushdown automata, **Stirling 2000**).

## Structures with decidable MSO theories: some milestones

[In timeline below, each item subsumes developments in preceding items.]

1. **Rabin 1969**: Regular trees. “Mother of all decidability results”
2. **Muller and Schupp 1985**: Configuration graphs of pushdown automata.
3. **Caucal (ICALP 1996)**: Prefix-recognizable graphs ( =  $\epsilon$ -closures of configuration graphs of pushdown automata, **Stirling 2000**).
4. **Knapik, Niwiński and Urzyczyn (TLCA 2001, FOSSACS 2002)**:  
***PushdownTree*** $_n \Sigma$  = Trees generated by order- $n$  pushdown automata.  
***SafeRecSchTree*** $_n \Sigma$  = Trees generated by order- $n$  **safe** recursion schemes.

## Structures with decidable MSO theories: some milestones

[In timeline below, each item subsumes developments in preceding items.]

1. **Rabin 1969**: Regular trees. “Mother of all decidability results”
2. **Muller and Schupp 1985**: Configuration graphs of pushdown automata.
3. **Caucal (ICALP 1996)**: Prefix-recognizable graphs (=  $\epsilon$ -closures of configuration graphs of pushdown automata, **Stirling 2000**).
4. **Knapik, Niwiński and Urzyczyn (TLCA 2001, FOSSACS 2002)**:  
***PushdownTree* <sub>$n$</sub>  $\Sigma$**  = Trees generated by order- $n$  pushdown automata.  
***SafeRecSchTree* <sub>$n$</sub>  $\Sigma$**  = Trees generated by order- $n$  **safe** recursion schemes.
5. **Caucal (MFCS 2002)**. ***CaucalTree* <sub>$n$</sub>  $\Sigma$**  and ***CaucalGraph* <sub>$n$</sub>  $\Sigma$** .  
**Theorem** (KNU-C). For every  $n \geq 0$ ,  
***PushdownTree* <sub>$n$</sub>  $\Sigma$  = *SafeRecSchTree* <sub>$n$</sub>  $\Sigma$  = *CaucalTree* <sub>$n$</sub>  $\Sigma$ .**

## Structures with decidable MSO theories: some milestones

[In timeline below, each item subsumes developments in preceding items.]

1. Rabin 1969: Regular trees. “Mother of all decidability results”
2. Muller and Schupp 1985: Configuration graphs of pushdown automata.
3. Caucal (ICALP 1996): Prefix-recognizable graphs (=  $\epsilon$ -closures of configuration graphs of pushdown automata, Stirling 2000).
4. Knapik, Niwiński and Urzyczyn (TLCA 2001, FOSSACS 2002):  
***PushdownTree*** $_n \Sigma$  = Trees generated by order- $n$  pushdown automata.  
***SafeRecSchTree*** $_n \Sigma$  = Trees generated by order- $n$  **safe** recursion schemes.
5. Caucal (MFCS 2002). ***CaucalTree*** $_n \Sigma$  and ***CaucalGraph*** $_n \Sigma$ .  
**Theorem** (KNU-C). For every  $n \geq 0$ ,  
***PushdownTree*** $_n \Sigma = \mathbf{SafeRecSchTree}_n \Sigma = \mathbf{CaucalTree}_n \Sigma$ .

**Question.** Do  $\Sigma$ -labelled trees generated by **unsafe** recursion schemes have decidable MSO theories? If so, at which orders?

## What is the safety constraint?

W. Damm: **Derived types** in “IO and OI Hierarchies”, TCS 1982.

**Definition** [KNU02]. An order-2 equation is **unsafe** if the RHS has a subterm  $P$  such that

1.  $P$  is order 1
2.  $P$  occurs in an **operand** position (i.e. as 2nd argument of the application operator)
3.  $P$  contains an order-0 parameter.

### Examples of unsafe equations:

$F : (o \rightarrow o) \rightarrow o \rightarrow o \rightarrow o$ ,  $G : o \rightarrow o$ ,  $H : (o \rightarrow o) \rightarrow o$ ,  $\underline{f} : o^2 \rightarrow o$ .

$$\begin{aligned} G x &= H(\underline{f} x) \\ F \varphi x y &= f(\underline{F}(\underline{F} \varphi y) y (\varphi x)) \underline{a} \end{aligned}$$

**Safety** (as presented above) seems syntactically awkward and semantically unnatural but (we shall see shortly) it has important algorithmic value.

## Is safety a genuine or spurious constraint for:

1. **Expressiveness.** Are there *inherently* unsafe  $\Sigma$ -labelled trees?

I.e. Is there an unsafe recursion scheme whose value tree is not the value tree of any safe recursion scheme? If so, at what order?

**Conjecture.** Yes, at order 2. But note:

**Theorem.** (A+deM+O FOSSACS 2005) There is no inherently unsafe word language at order 2.

2. **Decidability.** Is safety necessary for decidability? Two partial results:

**Theorem.** (A+deM+O 05)  $\Sigma$ -labelled trees generated by order-2 recursion schemes (*whether safe or not*) have decidable MSO theories.

**Theorem.** (KNUW 05) Modal  $\mu$ -calculus model checking problem for *homogeneously-typed* order-2 schemes (*whether safe or not*) is 2-EXPTIME complete.

**Question.** What about higher orders?

**Yes: MSO Decidability extends to all orders.**

## Motivation

A Tree Hierarchy  
Generated By  
Recursion Schemes

MSO Model-Checking  
Problem

Infinite Structures With  
Decidable MSO  
Theories

## Deciding MSO Theories of Trees

- Mu-calculus and APT
- Transference principle
- Long transform
- Traversals
- Path-Traversal Correspondence
- Traversal tree
- Example
- Example
- Simulation
- Variable profiles
- Traversal-simulating APT
- Main Technical Lemma
- Key Steps of the Decidability Proof

# Deciding MSO Theories of Trees

## MSO model-checking $\iff$ APT acceptance

**Aim:** Construct an algorithm to decide “Given order- $n$  recursion scheme  $G$ , and modal mu-calculus formula  $\varphi$ , does  $\llbracket G \rrbracket \models \varphi$ ? “

## MSO model-checking $\iff$ APT acceptance

**Aim:** Construct an algorithm to decide “Given order- $n$  recursion scheme  $G$ , and modal mu-calculus formula  $\varphi$ , does  $\llbracket G \rrbracket \models \varphi$ ? “

### Modal Mu-Calculus = Alternating Parity Tree Automaton (APT)

**Theorem** [EM91]. There is a transformation from mu-calculus formulas to APT,  $\varphi \mapsto \mathcal{B}_\varphi$ , such that for any  $\Sigma$ -labelled tree  $t$ ,  $t \models \varphi$  iff the APT  $\mathcal{B}_\varphi$  accepts  $t$ .

## MSO model-checking $\iff$ APT acceptance

**Aim:** Construct an algorithm to decide “Given order- $n$  recursion scheme  $G$ , and modal mu-calculus formula  $\varphi$ , does  $\llbracket G \rrbracket \models \varphi$ ? “

### Modal Mu-Calculus = Alternating Parity Tree Automaton (APT)

**Theorem** [EM91]. There is a transformation from mu-calculus formulas to APT,  $\varphi \mapsto \mathcal{B}_\varphi$ , such that for any  $\Sigma$ -labelled tree  $t$ ,  $t \models \varphi$  iff the APT  $\mathcal{B}_\varphi$  accepts  $t$ .

An APT  $\mathcal{B}$  **accepts a  $\Sigma$ -labelled tree**  $t$  if it has an **accepting run-tree** over  $t$  i.e. “there is a set of state-annotated **paths** in  $t$  that

1. respects the transition map  $\delta_{\mathcal{B}}$ , and
2. every infinite path satisfies the **parity condition**: largest priority that occurs infinitely often is even.”

Think of these (state-annotated) paths as footprints of the automata descending the tree.

## MSO model-checking $\iff$ APT acceptance

**Aim:** Construct an algorithm to decide “Given order- $n$  recursion scheme  $G$ , and modal mu-calculus formula  $\varphi$ , does  $\llbracket G \rrbracket \models \varphi$ ? “

### Modal Mu-Calculus = Alternating Parity Tree Automaton (APT)

**Theorem** [EM91]. There is a transformation from mu-calculus formulas to APT,  $\varphi \mapsto \mathcal{B}_\varphi$ , such that for any  $\Sigma$ -labelled tree  $t$ ,  $t \models \varphi$  iff the APT  $\mathcal{B}_\varphi$  accepts  $t$ .

An APT  $\mathcal{B}$  **accepts a  $\Sigma$ -labelled tree**  $t$  if it has an **accepting run-tree** over  $t$  i.e. “there is a set of state-annotated **paths** in  $t$  that

1. respects the transition map  $\delta_{\mathcal{B}}$ , and
2. every infinite path satisfies the **parity condition**: largest priority that occurs infinitely often is even.”

Think of these (state-annotated) paths as footprints of the automata descending the tree.

Hence our task is reduced to: **Given recursion scheme  $G$  and APT  $\mathcal{B}$ , check if  $\llbracket G \rrbracket$  has such a set of (state-annotated) paths.**

## Transference Principle: from value tree to computation tree

Direct algorithmic analysis of **value tree**  $\llbracket G \rrbracket$  is futile:

Value tree has no useful structure for our purpose: It is the “extensional” outcome of a (potentially infinite) computational process comprising two kinds of **intertwined** basic actions

1. unfolding
2.  $\beta$ -reduction

It is the algorithmics of this process that we *should* analyse.  
[KNU2002 did, hence their restriction to the safe case!]

## Transference Principle: from **value tree** to **computation tree**

Direct algorithmic analysis of **value tree**  $\llbracket G \rrbracket$  is futile:

Value tree has no useful structure for our purpose: It is the “extensional” outcome of a (potentially infinite) computational process comprising two kinds of **intertwined** basic actions

1. unfolding
2.  $\beta$ -reduction

It is the algorithmics of this process that we *should* analyse.

[KNU2002 did, hence their restriction to the safe case!]

**Our approach:** By considering rewrite-rule in **long form** (= curried, eta-long form), unfolding and  $\beta$ -reduction can be analysed separately (Aehlig).

- Build an auxiliary **computation tree**  $\lambda(G)$  which is the outcome of performing all of the unfolding, but none of the  $\beta$ -reduction (thus no substitution and hence no renaming needed!).
- Analyse the  $\beta$ -reduction **locally** (i.e. without the **global** operation of substitution) using game semantics - **traversals**.

## The Long Transform: from (order- $n$ ) $G$ to (order-0) $\overline{G}$

$\overline{G}$ -rules are obtained by: For each  $G$ -rule

1. Expand RHS to its  **$\eta$ -long form**, including ground-type subterm in *operand* position. Thus  $e : o$   $\eta$ -expands to  $\lambda.e$  (“dummy lambdas”).
2. Insert **long-apply symbol @**: Replace every ground-type subterm  $D e_1 \cdots e_n$  by  $@ D e_1 \cdots e_n$ , where  $D$  ranges over non-terminals.
3. Curry each equation.
4. Rename (bound) variables afresh. Only finitely many new names.

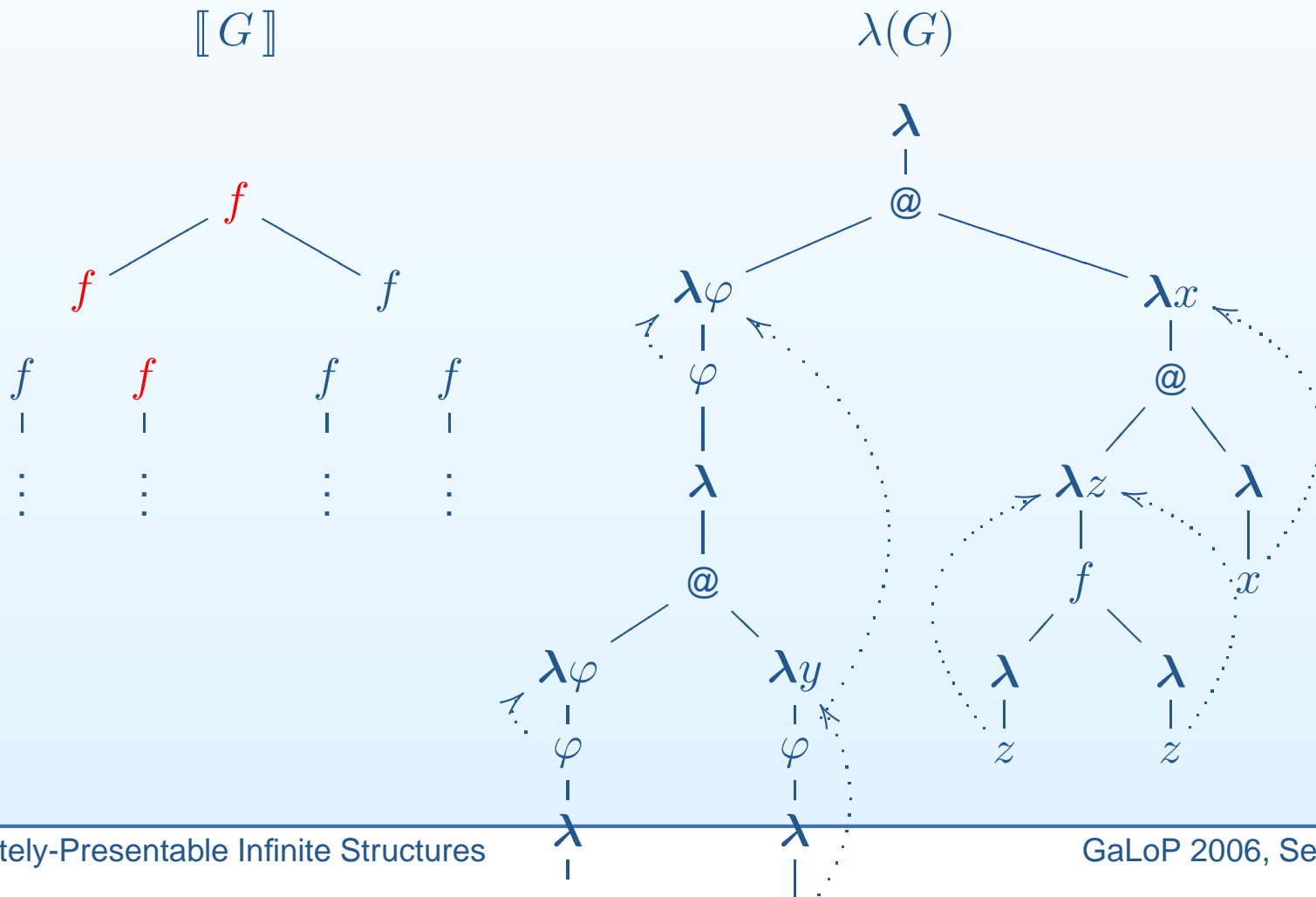
Note: This transform is **canonical** for innocent game semantics.

**Example.**

$$G : \begin{cases} S = F H \\ F \varphi = \varphi (F \varphi) \\ H z = f z z \end{cases} \mapsto \overline{G} : \begin{cases} S = \lambda.@ F (\lambda x.@ H \lambda.x) \\ F = \lambda\varphi.\varphi(\lambda.@ F (\lambda y.\varphi(\lambda.y))) \\ H = \lambda z.f(\lambda.z)(\lambda.z) \end{cases}$$

# Computation tree $\lambda(G)$ is obtained by infinitely unfolding $\overline{G}$ :

$$G : \begin{cases} S = FH \\ F\varphi = \varphi(F\varphi) \\ Hz = fzz \end{cases} \quad \mapsto \quad \overline{G} : \begin{cases} S = \lambda.@ F (\lambda x.@ H \lambda.x) \\ F = \lambda\varphi.\varphi(\lambda.@ F (\lambda y.\varphi(\lambda.y))) \\ H = \lambda z.f(\lambda.z)(\lambda.z) \end{cases}$$



# Traversals

**Definition.** *Traversals* over  $\lambda(G)$  are justified sequences defined by induction:

**(Root)** The singleton sequence (comprising  $\epsilon$ ) is a traversal.

**(App)** If  $t @$  is a traversal, so is  $t @ \overset{\leftarrow 0}{\lambda_{\xi}^{\bar{\xi}}}$ .

**(Sig)** If  $t f$  is a traversal, so is  $t f \overset{\leftarrow i}{\lambda}$  where  $1 \leq i \leq \text{arity}(f)$ .

**(Var)** If  $t n \overset{\leftarrow i}{\lambda_{\xi}^{\bar{\xi}}} \dots \xi$  is a traversal, so is  $t n \overset{\leftarrow i}{\lambda_{\xi}^{\bar{\xi}}} \dots \xi \overset{\leftarrow i}{\lambda_{\eta}^{\bar{\eta}}}$ .

**(Lam)** If  $t \lambda_{\xi}^{\bar{\xi}}$  is a traversal, so is  $t \lambda_{\xi}^{\bar{\xi}} n$ , such that  $\lceil t \lambda_{\xi}^{\bar{\xi}} n \rceil$  is a path in  $\lambda(G)$ .

## Key lemma:

- (i) Traversals are justified sequences that satisfy Visibility.
- (ii) **P-views of traversals are paths in the computation tree.**

## Path-Traversal Correspondence

**Theorem. (Correspondence)** Let  $G$  be an order- $n$  recursion scheme.

- (i) There is a 1-1 correspondence between **maximal paths**  $p$  in ( $\Sigma$ -labelled) value tree  $\llbracket G \rrbracket$  and **maximal traversals**  $t_p$  over computation tree  $\lambda(G)$ .
- (ii) Further for each  $p$ , we have  $p \upharpoonright \Sigma = t_p \upharpoonright \Sigma$ .

Proof is by game semantics.

### Idea:

- Value tree  $\llbracket G \rrbracket$  is a representation of the strategy-denotation of  $G$  (in game semantics).
- **Paths** in  $\llbracket G \rrbracket$  correspond to **plays** in the strategy-denotation.
- Nodes of the computation trees are representations of move-occurrences of the constituent arenas.
- **Traversals**  $t_p$  over computation tree  $\lambda(G)$  are just (representations of) the **uncoverings** of the plays (= path)  $p$  in the strategy-denotation of  $G$ .

## From run-tree over $\llbracket G \rrbracket$ to traversal-tree over $\lambda(G)$

Thus: Property APT  $\mathcal{B}$  has an **accepting run-tree** over  $\llbracket G \rrbracket$

by def.  $\iff$   $\left\{ \begin{array}{l} \exists \text{ certain set of } \delta_{\mathcal{B}}\text{-respecting, state-annotated} \\ \text{paths in } \llbracket G \rrbracket \text{ satisfying parity condition} \end{array} \right.$

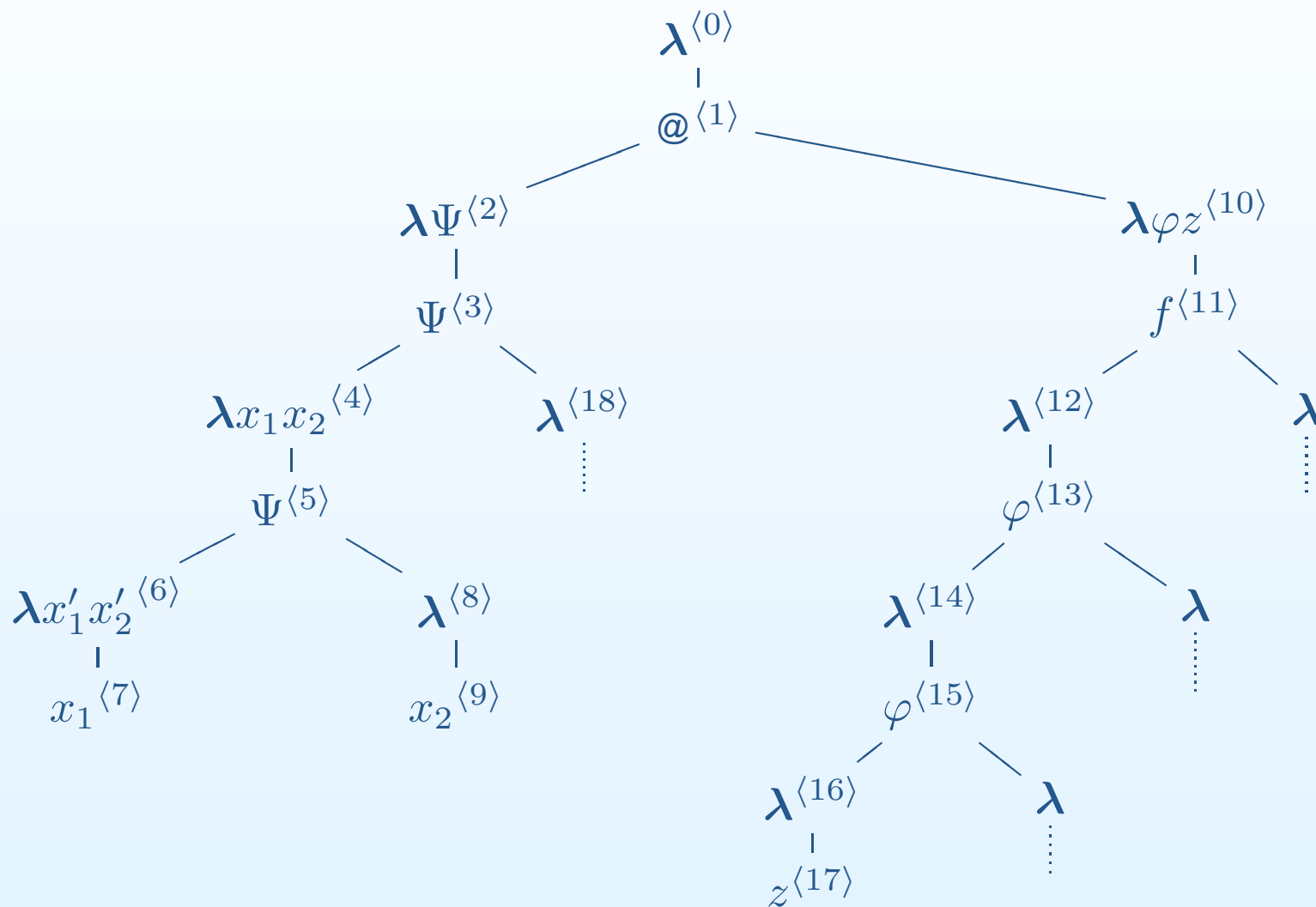
Thm (Corr)  $\iff$   $\left\{ \begin{array}{l} \exists \text{ certain set of } \delta_{\mathcal{B}}\text{-respecting, state-annotated} \\ \text{traversals over } \lambda(G) \text{ satisfying parity condition} \end{array} \right.$

**new def.**  $\iff$  Property APT  $\mathcal{B}$  has an **accepting traversal-tree** over  $\lambda(G)$ .

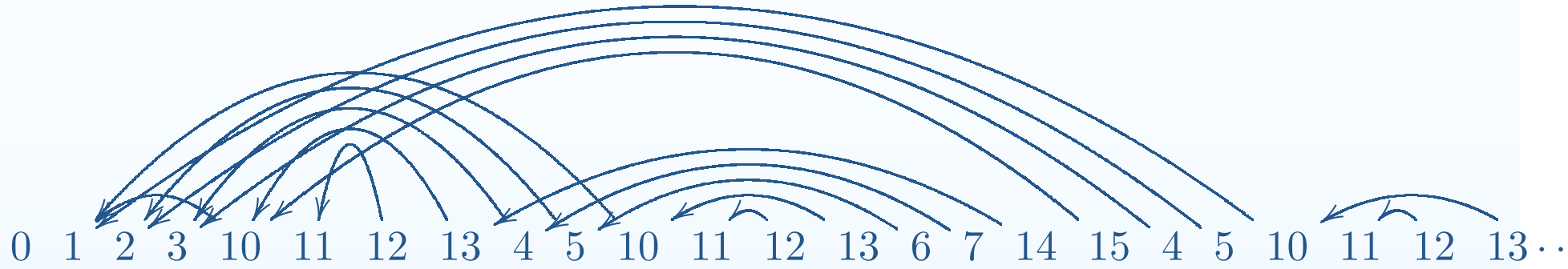
Higher-order traversals can be very complex - they jump all over the tree, and can visit certain nodes infinitely often. See order-3 example!

**Problem:** Find a device to recognise an accepting traversal-tree.

# Example

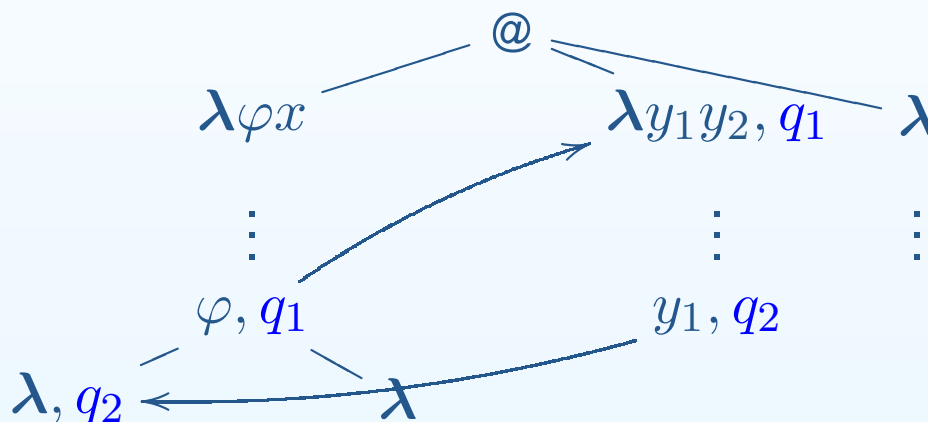


# Example



## Simulate traversals by *paths* – an order-2 illustration

**Idea.** Simulate an annotated traversal by the respective **P-views** of all its prefixes, which are a set of annotated paths in the computation tree.



Simulate the traversal above (indicated by arrows) by **paths**:

- At  $\varphi$  with  $q_1$ , **guess** that the detour will return at first  $\lambda$ -child with state  $q_2$
- **Spawn** an automaton at  $\lambda y_1 y_2$  to **verify the guess**.

## Formalising the guesses as **Variable Profiles** $\mathbf{VP}_G^{\mathcal{B}}(A)$

Fix a recursion scheme  $G$ , and a **property APT**  $\mathcal{B} = \langle \Sigma, Q, \delta, q_0, \Omega \rangle$  with  $p$  priorities. Write  $[p] = \{1, \dots, p\}$ .

$$\begin{aligned}\mathbf{VP}_G^{\mathcal{B}}(o) &= \text{Var}_G^o \times Q \times [p] \times 2^\emptyset \\ \mathbf{VP}_G^{\mathcal{B}}(A_1 \rightarrow \dots \rightarrow A_n \rightarrow o) &= \text{Var}_G^A \times Q \times [p] \times 2^{(\cup_{i=1}^n \mathbf{VP}_G^{\mathcal{B}}(A_i))}\end{aligned}$$

Asserting  $(\varphi, q, m, c) \in \mathbf{VP}_G^{\mathcal{B}}(A)$  at node  $\alpha$  of computation tree means: the traversal being simulated will reach some descendant-node that is labelled  $\varphi$

1. with state  $q$ , such that
2.  $m$  is the highest priority that will have been encountered up to that point
3. further, the traversal (which will then jump to the root of a subtree that denotes the *actual* argument of  $\varphi$ ) will eventually return to the children of the node labelled  $\varphi$  “in accord with  $c$ ”.

Note:  $|\mathbf{VP}_G^{\mathcal{B}}(i)| = \exp_i O(|G| \cdot |Q| \cdot p)$ .

## Traversal-simulating APT

**Aim:** Simulate  $\mathcal{B}$ -states + verify guesses (= variable profiles).

$\mathcal{C}$ -states:  $q \rho$  where  $q$  is  $\mathcal{B}$ -state being simulated, and environment  $\rho$  is the set of profiles of variable (within current scope) to be verified.

**Suppose automaton with state  $q \rho$  reading node with label  $l$ : Some cases**  
(verification of priorities omitted)

- $l$  is a  $\Sigma$ -symbol  $f : o^k \rightarrow o$ .  
Guess a set  $\{ (i_1, q_1), \dots, (i_l, q_l) \}$  satisfying  $\delta_{\mathcal{B}}(q, f)$  (abort, if impossible), and guess environments  $\rho_1, \dots, \rho_l$  such that  $\bigcup_{j=1}^l \rho_j = \rho$ .  
For each  $j$ , spawn automata with state  $q_j \rho_j$  in direction  $i_j$ .
- $l$  is an @ with children labelled by  $\lambda \bar{\varphi}$  and  $\lambda \bar{\eta}_1, \dots, \lambda \bar{\eta}_k$ .  
Guess  $\rho' = \{ (\varphi_{i_j}, q_j, m_j, c_j) : 1 \leq j \leq l \}$ , and spawn automaton with state  $q \rho'$  in direction 0.  
Guess  $\rho_1, \dots, \rho_l$  with  $\bigcup_{j=1}^l \rho_j = \rho$ . For each  $j$ , spawn automaton with state  $q_j (\rho_j \cup c_j)$  in direction  $i_j$ .

## Main Technical Lemma

**Theorem (Simulation).** The following are equivalent:

- (i) Property APT  $\mathcal{B}$  has an accepting traversal-tree over the computation tree  $\lambda(G)$ .
- (ii) Traversal-simulating APT  $\mathcal{C}$  has an accepting run-tree over the computation tree  $\lambda(G)$ .

“(i)  $\Rightarrow$  (ii)”: From the traversal-tree annotated only by  $\mathcal{B}$ -states, we perform a succession of annotation operations, transforming it to a traversal-tree annotated by  $\mathcal{C}$ -states.

The set of P-views of all such  $\mathcal{C}$ -state-annotated traversals is precisely an accepting run-tree of  $\mathcal{C}$ .

“(ii)  $\Rightarrow$  (i)”: Reconstruct each traversal (of the putative traversal-tree) as a sequence of segments of paths (=P-views) in the accepting run-tree, thus inheriting an accepting state-annotation.

Satisfaction of parity condition tricky to show!

## Key Steps of the Decidability Proof

Let  $G$  be any order- $n$  recursion scheme, and  $\varphi$  a modal mu-calculus formula.

The question of whether:

- Value tree  $\llbracket G \rrbracket$  satisfies  $\varphi$
- $\iff$  { Emerson + Jutla 1991 }
- Property APT  $\mathcal{B}$  has accepting run-tree over  $\llbracket G \rrbracket$
- $\iff$  { Correspondence Theorem }
- $\mathcal{B}$  has an accepting traversal-tree over computation tree  $\lambda(G)$
- $\iff$  { Simulation Theorem }
- Traversal-simulating APT  $\mathcal{C}$  has an accepting run-tree over  $\lambda(G)$

which is decidable, since the computation tree  $\lambda(G)$  is regular, and the APT acceptance problem of regular trees is decidable.

**Theorem.** For each  $n \geq 0$ , the modal mu-calculus model checking-problem for **RecSchTree** $_n$  (i.e. trees generated by order- $n$  recursion schemes) is  $n$ -EXPTIME complete. Thus these trees have decidable MSO theories.

Motivation

---

A Tree Hierarchy  
Generated By  
Recursion Schemes

---

MSO Model-Checking  
Problem

---

Infinite Structures With  
Decidable MSO  
Theories

---

Deciding MSO Theories  
of Trees

---

**Collapsible Pushdown  
Automata (with  
A. Murawski)**

---

- Order-2 PDA
- Collapsible PDA
- Schemes and Automata

A Hierarchy of Graphs

---

---

# Collapsible Pushdown Automata (with A. Murawski)

---

## Order-2 pushdown automata (Maslov 74)

A **1-stack** is an ordinary stack.

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

**Operations on 2-stacks:**  $s_i$  ranges over 1-stacks

$$\text{push}_2 : [s_1 \cdots s_{i-1} \underbrace{[a_1 \cdots a_n]}_{s_i}] \mapsto [s_1 \cdots s_{i-1} s_i s_i]$$

$$\text{pop}_2 : [s_1 \cdots s_{i-1} [a_1 \cdots a_n]] \mapsto [s_1 \cdots s_{i-1}]$$

$$\text{push}_1 a : [s_1 \cdots s_{i-1} [a_1 \cdots a_n]] \mapsto [s_1 \cdots s_{i-1} [a_1 \cdots a_n a]]$$

$$\text{pop}_1 : [s_1 \cdots s_{i-1} [a_1 \cdots a_n a_{n+1}]] \mapsto [s_1 \cdots s_{i-1} [a_1 \cdots a_n]]$$

## Order- $n$ collapsible pushdown automata (CPDA)

**Order-2 CPDA:** [KNUW05] “panic automata”; [AdMO05] “2PDA with links”

Each stack symbol in 2-stack “remembers” the stack content at the point it was first created (i.e.  $\text{push}_1$ -ed), by way of a pointer to some 1-stack buried underneath it (if there is one such).

Two new operations:

- $\text{push}_1 a$ : Whenever a symbol is pushed onto the top of stack, it has a pointer to the 1-stack immediately below the top 1-stack.
- “collapse” (= panic) collapses the 2-stack up to the point as remembered by (i.e. pointed to) by the top element of the 2-stack.

In order- $n$  CPDA, there are  $n - 1$  versions of  $\text{push}_1$ , namely,  $\text{push}_1^j a$ , with  $1 \leq j \leq n - 1$ :

$\text{push}_1^j a$ : Whenever a symbol is pushed onto the top of stack, it has a pointer to the  $j$ -stack immediately below the top  $j$ -stack.

## Order- $n$ recursion schemes = order- $n$ CPDA

$U$  is recognised by a **deterministic** 2CPDA and a **non-deterministic** 2PDA.

**Conjecture.**  $U$  is not recognisable by a deterministic 2PDA.

As a corollary, there is an associated tree that is generated by an order-2 recursion scheme, but not by any order-2 **safe** scheme.

**Theorem.** For each  $n \geq 0$ , order- $n$  collapsible PDA and order- $n$  recursion schemes are equi-expressive for  $\Sigma$ -labelled trees.

### Proof idea

- From recursion scheme  $G$  to CPDA  $\mathcal{A}_G$ : Use game semantics.  
Code traversals as  $n$ -stacks.  
Invariant: The top 1-stack is the P-view of the encoded traversal.
- From CPDA  $\mathcal{A}$  to recursion scheme  $G_{\mathcal{A}}$ :  
Code configurations  $c$  as  $\Sigma$ -term  $M_c$ , so that  $c \rightarrow c'$  implies  $M_c$  rewrites (in 1-step) to  $M_{c'}$ .

Motivation

A Tree Hierarchy  
Generated By  
Recursion Schemes

MSO Model-Checking  
Problem

Infinite Structures With  
Decidable MSO  
Theories

Deciding MSO Theories  
of Trees

Collapsible Pushdown  
Automata (with  
A. Murawski)

**A Hierarchy of Graphs**

- CPDA graphs
- Many Further  
Directions

# A Hierarchy of Graphs

## Parity games over collapsible $n$ -pushdown graphs

The same approach applies to solving parity games over graphs.

There is a **transformation** from  $n$ -collapsible pushdown system (CPDS)  $A$  to an equivalent order- $n$  (non-deterministic) recursion scheme  $G_A$ .

**Transference Principle:** Paths in the configuration graph of the CPDS  $A$ ,  $\mathcal{CG}_A$ , correspond exactly to traversals over the computation tree  $\lambda(G_A)$  (or equivalently over the finite computation graph  $\text{Gr}(A)$  that unfolds to  $\lambda(G)$ ).

**Simulating Traversals:** For any parity game over  $\mathcal{G}_A$ , accepting traversal-trees over  $\text{Gr}(A)$  can be recognised by a traversal-simulating APT  $\mathcal{C}$ .

Thus, for any parity game over  $\mathcal{CG}_A$ , there is an equivalent **finite** acceptance parity game, which is a product of  $\text{Gr}(A)$  and  $\mathcal{C}$ .

Hence parity games over collapsible  $n$ -pushdown graphs are solvable.

Many open problems. E.g. Is winning region of 2-collapsible pushdown game regular? What about higher-order games?

## Many Further Directions

1. Is safety a genuine constraint on expressiveness? Equivalently, are order- $n$  collapsible PDA more expressive than order- $n$  PDA?

**Conjecture.**  $\mathbf{SafeRecSch}_2\Sigma \subset \mathbf{RecSchTree}_2\Sigma$  I.e. There are *inherently* unsafe trees (at order 2).

Candidate: Urzyczyn's tree.

2. “Mixing semantic and verification games”: Denotational semantics of  $\lambda$ -calculus “relative to an alternating parity tree automaton (APT)”.

**Problem.** Construct a cartesian closed category (= model of the lambda calculus), parameterized by an APT, whose maps are witnessed by profiles (“guesses”).