# Verifying Finitely-Presentable Infinite Structures:

# A Game-Semantic Approach

# (Lecture 2)

Luke Ong

University of Oxford

29 May - 2 June 2006

# Review

## Background

**Theorem** (Knapik, Niwiński + Urzyczyn FOSSACS02). The MSO model checking problem for trees generated by order-$n$ safe recursion schemes is decidable, for each $n \geq 0$.

**Recall:** Homogeneous types

- $o$ is homogeneous
- $(A_1 \to \cdots \to A_n \to o)$ is homogeneous just if $order(A_1) \geq order(A_2) \geq \cdots \geq order(A_n)$, and each $A_i$ is homogeneous.

Safety (which presupposes that all types are homogeneous) is a rather awkward syntactic constraint; but

- It has a clear algorithmic virtue: Safe lambda calculus is an "$\alpha$-conversion free lambda calculus".
- It has an elegant automata-theoretic characterization: Order-$n$ safe recursion schemes = order-$n$ pushdown automata.

## 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 decidabiliy? 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: Decidability result extends to all orders – main topic of this lecture.

## Safety is not necessary for MSO decidability

**Theorem**. (LICS 06) The modal mu-calculus model checking problem for trees generated by arbitrary order-$n$ recursion schemes is $n$-EXPTIME complete, for each $n \geq 0$.

We first consider the decidability argument and then discuss the complexity analysis.

## 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 $[\![\, G \,]\!]$ satisifes $\varphi$

$\Longleftrightarrow$ $\{$ Emerson + Jutla 1991$\}$

'Property' alternating parity tree automaton (APT) $\mathcal{B}_\varphi$

has an accepting run-tree over value tree $[\![\, G \,]\!]$

$\overset{\text{Part 1}}{\Longleftrightarrow}$ $\{$ Correspondence Theorem $\}$

$\mathcal{B}_\varphi$ has an accepting traversal-tree over computation tree $\lambda(G)$

$\overset{\text{Part 2}}{\Longleftrightarrow}$ $\{$ Simulation Theorem $\}$

'Traversal-simulating' APT $\mathcal{C}_\varphi$ 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 (Rabin, Emerson, Jutla, etc.).

## Outline

**Review**

**Alternating Parity Tree Automata (APT) and the Modal Mu-Calculus**

**Decidability Argument 1: Transference Principle**

**Decidability Argument 2: Simulating Traversals**

**Complexity Analysis**

**Characterising recursion schemes by collapsible PDA (with A. Murawski)**

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

# Alternating Parity Tree Automata (APT) and the Modal Mu-Calculus

## Modal mu-calculus and alternating parity tree automata (APT) are equivalent

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

Positive boolean formulas over a set $P$: $p$ ranges over $P$

$$\mathsf{B}^+(P) \ni \theta \ ::= \ \mathsf{true} \mid \mathsf{false} \mid p \mid \theta \wedge \theta \mid \theta \vee \theta$$

For $S \subseteq P$ and $\theta \in \mathsf{B}^+(P)$, we say $S$ satisfies $\theta$ if assigning true to elements in $S$ and false to elements in $P \setminus S$ makes $\theta$ true.

**A**lternating **P**arity **T**ree Automaton (APT)

$$\mathcal{B} \ = \ \langle \Sigma, \ Q, \ \delta, \ q_0 \in Q, \ \Omega : Q \longrightarrow \mathbb{N} \rangle$$

where
$\delta : Q \times \Sigma \longrightarrow \mathsf{B}^+([ar(\Sigma)] \times Q)$ is the transition function where, for each $f \in \Sigma$ and $q \in Q$, we have $\delta(q, f) \in \mathsf{B}^+([ar(f)] \times Q)$
*Notation:* $[m] = \{1, \cdots, m\}$; $ar(\Sigma) = \max\{ ar(f) : f \in \Sigma \}$.

## Acceptance of $\Sigma$-labelled tree $t : \mathrm{dom}(t) \longrightarrow \Sigma$ by an APT $\mathcal{B}$

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

1. '$\delta_{\mathcal{B}}$-respecting', and
2. such that the infinite paths among them satisfy the parity condition."

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

## Acceptance of $\Sigma$-labelled tree $t : \mathrm{dom}(t) \longrightarrow \Sigma$ by an APT $\mathcal{B}$

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

1. '$\delta_{\mathcal{B}}$-respecting', and
2. such that the infinite paths among them satisfy the parity condition."

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

**$\delta_{\mathcal{B}}$-respecting**:
Automaton reads root $\epsilon$ with initial state $q_0$.
Suppose automaton reads node $\alpha$ of $\mathrm{dom}(t)$ with state $q$.

- Recall: $\delta_{\mathcal{B}} : Q \times \Sigma \longrightarrow \mathsf{B}^+([ar(\Sigma)] \times Q)$.
  Guess a set $S \subseteq [ar(t(\alpha))] \times Q$ that satisfies the positive boolean formula $\delta_{\mathcal{B}}(q, t(\alpha))$.
- For each $(i, q') \in S$, spawn automaton to read $i$-child of $\alpha$ with state $q'$.

## Acceptance of $\Sigma$-labelled tree $t : \mathrm{dom}(t) \longrightarrow \Sigma$ by an APT $\mathcal{B}$

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

1. '$\delta_{\mathcal{B}}$-respecting', and
2. such that the infinite paths among them satisfy the parity condition."

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

**$\delta_{\mathcal{B}}$-respecting**:
Automaton reads root $\epsilon$ with initial state $q_0$.
Suppose automaton reads node $\alpha$ of $\mathrm{dom}(t)$ with state $q$.

- Recall: $\delta_{\mathcal{B}} : Q \times \Sigma \longrightarrow \mathsf{B}^+([ar(\Sigma)] \times Q)$.
  Guess a set $S \subseteq [ar(t(\alpha))] \times Q$ that satisfies the positive boolean formula $\delta_{\mathcal{B}}(q, t(\alpha))$.
- For each $(i, q') \in S$, spawn automaton to read $i$-child of $\alpha$ with state $q'$.

**Parity condition**: largest priority that occurs infinitely often is even.

# Decidability Argument 1: Transference Principle

## Transference Principle: from value tree to computation tree

Direct algorithmic analysis of **value tree** $[\![\, G \,]\!]$ 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** $[\![ G ]\!]$ 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 - **traverals**.

## 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 $\boldsymbol{\lambda}.e$ ("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.

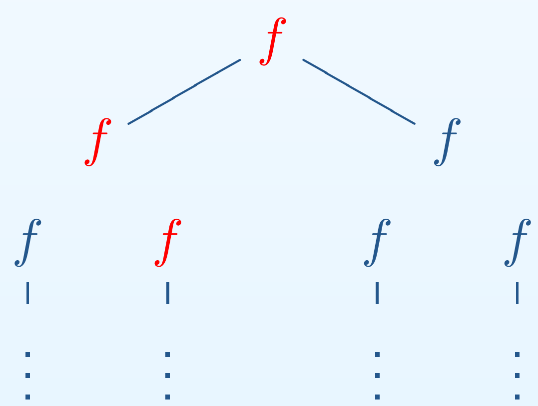
**Example**.

$$G : \left\{ \begin{array}{rcl} S & = & F\,H \\ F\,\varphi & = & \varphi\,(F\,\varphi) \\ H\,z & = & f\,z\,z \end{array} \right. \quad \mapsto \quad \overline{G} : \left\{ \begin{array}{rcl} S & = & \boldsymbol{\lambda}.@\,F\,(\boldsymbol{\lambda}x.@\,H\,\boldsymbol{\lambda}.x) \\ F & = & \boldsymbol{\lambda}\varphi.\varphi(\boldsymbol{\lambda}.@\,F\,(\boldsymbol{\lambda}y.\varphi(\boldsymbol{\lambda}.y)))) \\ H & = & \boldsymbol{\lambda}z.f\,(\boldsymbol{\lambda}.z)(\boldsymbol{\lambda}.z) \end{array} \right.$$

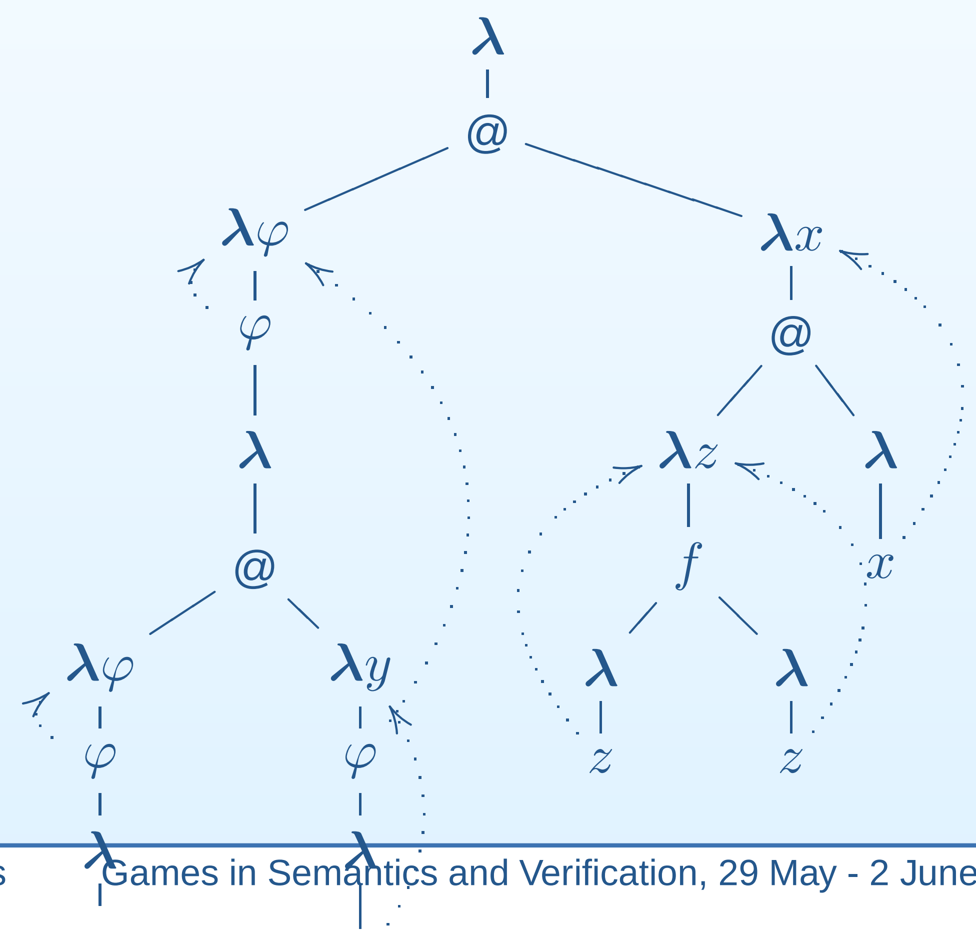
# Computation tree $\lambda(G)$ is obtained by infinitely unfolding $\overline{G}$:

$$G : \left\{ \begin{array}{rcl} S & = & F\, H \\ F\, \varphi & = & \varphi\, (F\, \varphi) \\ H\, z & = & f\, z\, z \end{array} \right. \qquad \mapsto \qquad \overline{G} : \left\{ \begin{array}{rcl} S & = & \boldsymbol{\lambda}.@\, F\, (\boldsymbol{\lambda} x.@\, H\, \boldsymbol{\lambda}.x) \\ F & = & \boldsymbol{\lambda}\varphi.\varphi(\boldsymbol{\lambda}.@\, F\, (\boldsymbol{\lambda} y.\varphi(\boldsymbol{\lambda}.y)))) \\ H & = & \boldsymbol{\lambda} z.f(\boldsymbol{\lambda}.z)(\boldsymbol{\lambda}.z) \end{array} \right.$$

$[\![ G ]\!]$



$\lambda(G)$



## 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 $t\ @\overset{0}{\frown}\boldsymbol{\lambda}\overline{\xi}$ .

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

**(Var)** If $t\,n\ \boldsymbol{\lambda}\overline{\xi}\overset{i}{\frown}\cdots\ \xi$ is a traversal, so is $t\ n\ \boldsymbol{\lambda}\overline{\xi}\overset{i}{\frown}\cdots\ \xi\ \boldsymbol{\lambda}\overline{\eta}$ .

**(Lam)** If $t\ \boldsymbol{\lambda}\overline{\xi}$ is a traversal, so is $t\ \boldsymbol{\lambda}\overline{\xi}\ n$, such that $\ulcorner t\ \boldsymbol{\lambda}\overline{\xi}\ n \urcorner$ 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 $[\![ G ]\!]$ and maximal traversals $t_p$ over computation tree $\lambda(G)$.

(ii) Further for each $p$, we have $p \upharpoonright \Sigma = t_p \upharpoonright \Sigma$.

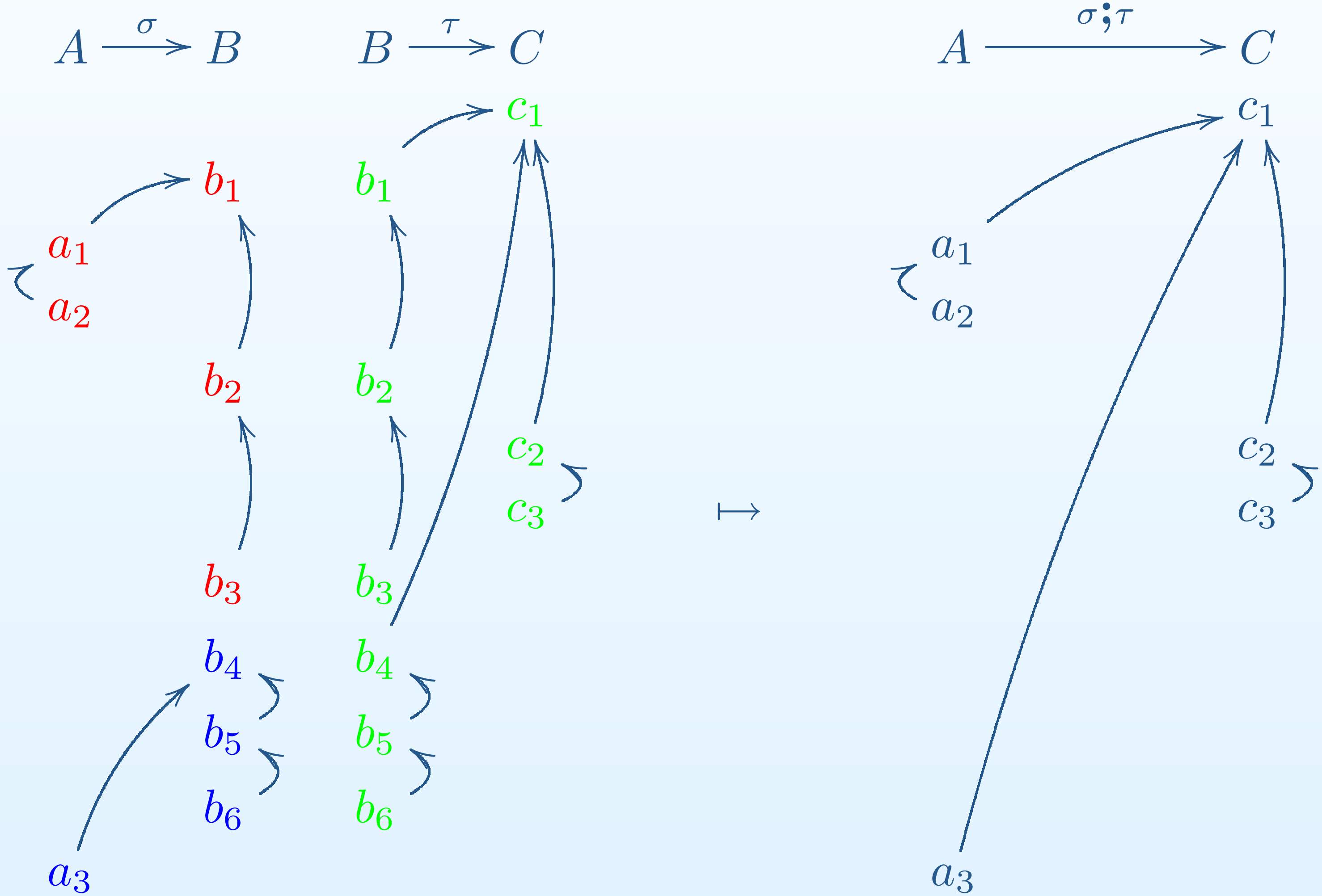
Proof is by game semantics.

**Idea**:

- Value tree $[\![ G ]\!]$ is a representation of the strategy-denotation of $G$ (in game semantics).
- Paths in $[\![ G ]\!]$ 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 $G$.

# Composition

Strategy composition is "parallel composition of two processes $\sigma : A \Rightarrow B$ and $\tau : B \Rightarrow C$ synchronizing on $B$, followed by hiding of $B$-moves."




The play in $\sigma ; \tau$ thus constructed is $c_1\ a_1\ a_2\ c_2\ c_3\ a_3$.

## From run-tree over $[\![\, G \,]\!]$ to traversal-tree over $\lambda(G)$

Thus: Property APT $\mathcal{B}$ has an accepting run-tree over $[\![\, G \,]\!]$

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

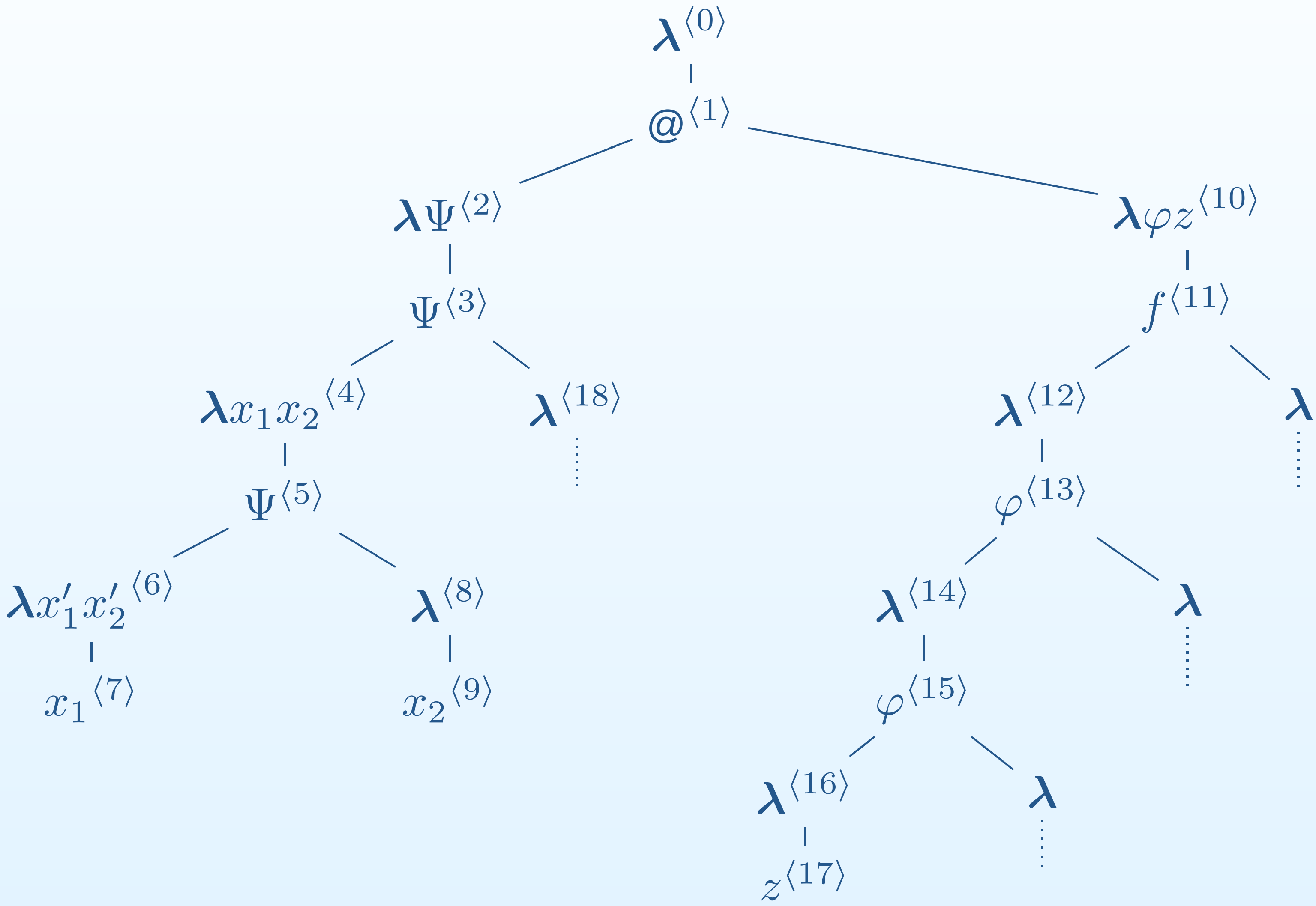
$\overset{\text{Thm (Corr)}}{\Longleftrightarrow}$ $\left\{ \begin{array}{l} \exists \text{ certain set of } \delta_{\mathcal{B}}\text{-respecting, state-annotated} \\ \text{traversals over } \lambda(G) \text{ satisfying parity condition} \end{array} \right.$

$\overset{\text{new def.}}{\Longleftrightarrow}$ 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!

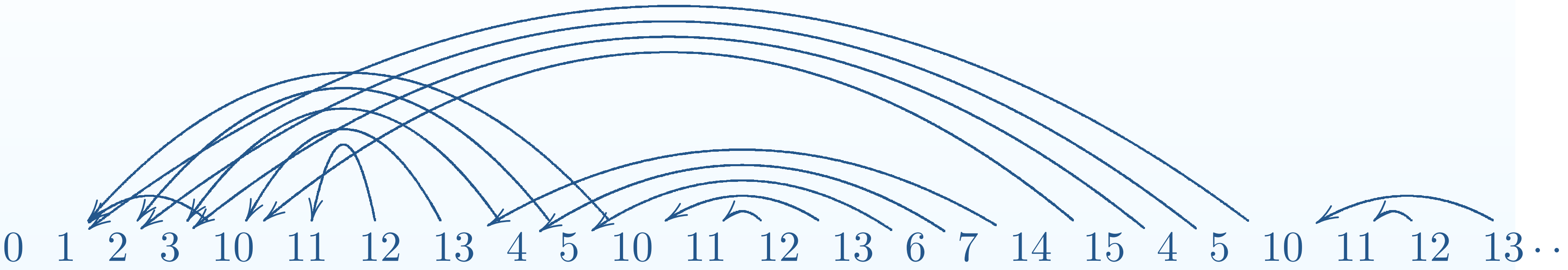
**Problem**: Find a device to recognise an accepting traversal-tree.

## Example



$\boldsymbol{\lambda}^{\langle 0\rangle}$
$@^{\langle 1\rangle}$
$\boldsymbol{\lambda}\Psi^{\langle 2\rangle}$
$\boldsymbol{\lambda}\varphi z^{\langle 10\rangle}$
$\Psi^{\langle 3\rangle}$
$f^{\langle 11\rangle}$
$\boldsymbol{\lambda} x_1 x_2^{\langle 4\rangle}$
$\boldsymbol{\lambda}^{\langle 18\rangle}$
$\boldsymbol{\lambda}^{\langle 12\rangle}$
$\boldsymbol{\lambda}$
$\Psi^{\langle 5\rangle}$
$\varphi^{\langle 13\rangle}$
$\boldsymbol{\lambda} x_1' x_2'^{\langle 6\rangle}$
$\boldsymbol{\lambda}^{\langle 8\rangle}$
$\boldsymbol{\lambda}^{\langle 14\rangle}$
$\boldsymbol{\lambda}$
$x_1^{\langle 7\rangle}$
$x_2^{\langle 9\rangle}$
$\varphi^{\langle 15\rangle}$
$\boldsymbol{\lambda}^{\langle 16\rangle}$
$\boldsymbol{\lambda}$
$z^{\langle 17\rangle}$

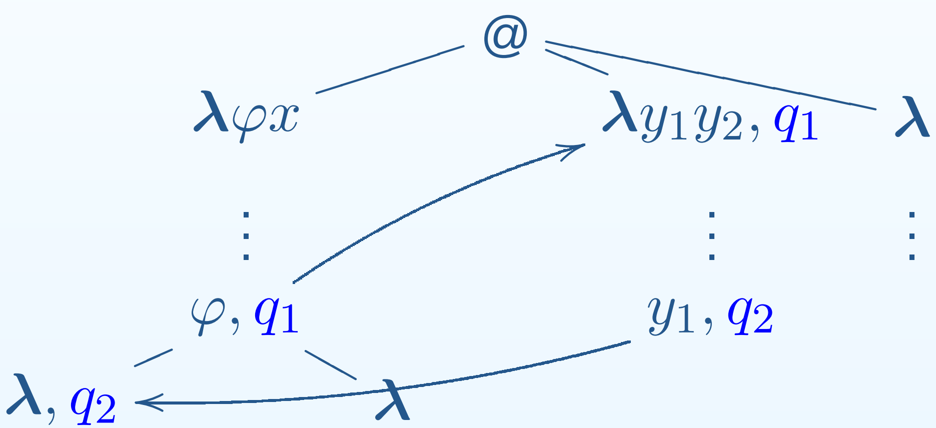
## Example



# Decidability Argument 2: Simulating Traversals

## 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 $\boldsymbol{\lambda}$-child with state $q_2$
- Spawn an automaton at $\boldsymbol{\lambda} y_1 y_2$ 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 $[p] = \{ 1, \cdots, p \}$.

$$\begin{array}{rcl} \mathbf{VP}_G^{\mathcal{B}}(o) & = & Var_G^o \times Q \times [p] \times 2^{\varnothing} \\ \mathbf{VP}_G^{\mathcal{B}}(A_1 \to \cdots \to A_n \to o) & = & Var_G^A \times Q \times [p] \times 2^{(\bigcup_{i=1}^n \mathbf{VP}_G^{\mathcal{B}}(A_i))} \end{array}$$

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 $f : o^k \to o$.
  Guess a set $\{ (i_1, q_1), \cdots, (i_l, q_l) \}$ satisfying $\delta_{\mathcal{B}}(q, f)$ (abort, if impossible), and guess environments $\rho_1, \cdots, \rho_l$ such that $\bigcup_{j=1}^{l} \rho_j = \rho$.
  For each $j$, spawn automata with state $q_j\ \rho_j$ in direciton $i_j$.
- $l$ is an @ with children labelled by $\boldsymbol{\lambda}\overline{\varphi}$ and $\boldsymbol{\lambda}\overline{\eta_1}, \cdots, \boldsymbol{\lambda}\overline{\eta_k}$.
  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 $\rho_1, \cdots, \rho_l$ 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:

$$
\begin{array}{ll}
 & \text{Value tree } [\![ G ]\!] \text{ satisifes } \varphi \\
\Longleftrightarrow & \{ \text{ Emerson + Jutla 1991} \} \\
 & \text{Property APT } \mathcal{B} \text{ has accepting run-tree over } [\![ G ]\!] \\
\Longleftrightarrow & \{ \text{ Correspondence Theorem } \} \\
 & \mathcal{B} \text{ has an accepting traversal-tree over computation tree } \lambda(G) \\
\Longleftrightarrow & \{ \text{Simulation Theorem } \} \\
 & \text{Traversal-simulating APT } \mathcal{C} \text{ has an accepting run-tree over } \lambda(G)
\end{array}
$$

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

# Complexity Analysis

## Equivalence of decision problems

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

The question of whether

Value tree $[\![\, G \,]\!]$ satisifes $\varphi$

$\iff$ $\{$ Emerson + Jutla 1991$\}$

Property APT $\mathcal{B}$ has accepting run-tree over $[\![\, G \,]\!]$

$\iff$ $\{$ 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 $\lambda(G)$

$\iff$ $\{$ Emerson + Jutla, Stirling, etc. $\}$

Eloise has winning strategy in acceptance parity game $\mathbf{G}(\mathrm{Gr}(G), \mathcal{C})$ (from root) for finite graph $\mathrm{Gr}(G)$ which unravels to $\lambda(G)$

## Complexity of Modal Mu-Calculus Model Checking

Model-checking safe trees is already $n$-EXPTIME hard. (Cachat ICALP'04 + Walukiewicz)

**Use parity game to show problem is decidable in $n$-EXPTIME.**

**Theorem**. (Jurdzinski 2000) Eloise's winning regions and strategy in a parity game over $(V, E)$ with $p$ $(\geq 2)$ priorities is computable in time

$$O\left(p \cdot |E| \cdot \left(\frac{|V|}{\lfloor p/2 \rfloor}\right)^{\lfloor p/2 \rfloor}\right)$$

**Note:** Actually a coarse bound $|V|^{O(|p|)}$ suffices (Emerson + Lei 86).

Fix an order-$n$ $G$, a property APT $\mathcal{B}$ with traversal-simulating APT $\mathcal{C}$. Construct **acceptance parity game G**$(\mathrm{Gr}(G), \mathcal{C})$ such that the finite deterministic $\Lambda_G$-labelled graph $\mathrm{Gr}(G)$ unfolds to $\lambda(G)$.

**Fact**. Eloise has a winning strategy in **G**$(\mathrm{Gr}(G), \mathcal{C})$ iff the APT $\mathcal{C}$ accepts $\lambda(G)$ (iff $\mathcal{B}$ accepts $[\![ G ]\!]$).

## Complexity

Recall $\textbf{VP}^{\mathcal{B}}_G(A_1 \to \cdots \to A_n \to o) = Var^A_G \times Q \times [p] \times 2^{(\bigcup_{i=1}^n \textbf{VP}^{\mathcal{B}}_G(A_i))}$
Write $\textbf{VP}^{\mathcal{B}}_G(i)$ for the set of profiles of variables of order at most $i$. We have

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

**Theorem**. (Succinctness). If the traversal-simulating APT $\mathcal{C}$ has an accepting run-tree, it has one with a small branching factor.

As a corollary, $|V| = \exp_n O(|G| \cdot |Q| \cdot p)$. Since $|E|$ is at most $|V|^2$, time complexity is

$$O\left(p \cdot (|V|)^{\lfloor p/2 \rfloor + 2}\right) \ = \ \exp_n O(|G| \cdot |Q| \cdot p)$$

**Theorem**. The modal mu-calculus model checking problem for trees generated by order-$n$ recursion schemes is $n$-EXPTIME complete.

# Characterising recursion schemes by collapsible PDA (with A. Murawski)

# Order-2 pushdown automata (Maslov 74)

A 1-stack is just 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

$$\begin{array}{rcll}
\mathsf{push}_2 & : & s_1 \cdots s_{i-1} \underbrace{[a_1 \cdots a_n]}_{s_i} & \mapsto \quad s_1 \cdots s_{i-1}\, s_i\, s_i \\
\mathsf{pop}_2 & : & s_1 \cdots s_{i-1} \underbrace{[a_1 \cdots a_n]}_{s_i} & \mapsto \quad s_1 \cdots s_{i-1} \\
\mathsf{push}_1\, a & : & s_1 \cdots s_{i-1} \underbrace{[a_1 \cdots a_n]}_{s_i} & \mapsto \quad s_1 \cdots s_{i-1} \underbrace{[a_1 \cdots a_n\, a]} \\
\mathsf{pop}_1 & : & s_1 \cdots s_{i-1} \underbrace{[a_1 \cdots a_n]}_{s_i}\,] & \mapsto \quad s_1 \cdots s_{i-1} \underbrace{[a_1 \cdots a_{n-1}]}
\end{array}$$

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

**Order-2 CPDA**:
[KNUW ICALP05] "panic automata"; [AdMO FOSSACS05] "2PDA with links"

Each stack symbol in $2$-stack "remembers" the stack content at the point it was first created (i.e. push$_1$-ed), by way of a pointer to some 1-stack buried underneath it (if there is one such).
Two new operations:

- 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 push$_1$, namely, push$_1^j$ a, with $1 \leq j \leq n - 1$:
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.

## Example: Urzyczyn's Language $U$ over alphabet $\{\,\mathbf{(}, \mathbf{)}, *\,\}$

$U$-words are uniquely composed of 3 segments:

$$\underbrace{\mathbf{(}\cdots\mathbf{(}\cdots\mathbf{(}}_{A}\ \underbrace{\mathbf{(}\cdots\mathbf{)}\cdots\mathbf{(}\cdots\mathbf{)}}_{B}\ \underbrace{*\cdots*}_{C}$$

- Segment $A$ is a prefix of a well-bracketed word that ends in **(**, such that none of its prefixes is a well-bracketed word.
- Segment $B$ is a well-bracketed word.
- Segment $C$ has length equal to the number of **(** in $A$.

E.g. $\mathbf{(\,(\,)\,(\,(\,)\,(\,(\,)\,)}\ *\ *\ * \quad \in \quad U$.

## Recognising $U$ by a 2CPDA. E.g. **( ( ) ( ( )** $* \; * * \; \in \; U$

push$_1 a$ upon reading the first symbol, which must be **(**. Thereafter:

- push$_2$ **;** push$_1 a$ upon reading **(**
- pop$_1$ upon reading **)**
- collapse upon reading first $*$, thereafter pop$_2$ for each subsequent $*$.

**(** $\quad$ `[[` $a$ `]]`

## Recognising $U$ by a 2CPDA. E.g. **( ( ) ( ( )** $* ** \in U$

push$_1 a$ upon reading the first symbol, which must be **(**. Thereafter:

- push$_2$ **;** push$_1 a$ upon reading **(**
- pop$_1$ upon reading **)**
- collapse upon reading first $*$, thereafter pop$_2$ for each subsequent $*$.

| | |
|---|---|
| **(** | $[\,[\ a\ ]\,]$ |
| **(** | $[\,[\ a\ ]\,[\ a\ \ a\ ]\,]$ |

## Recognising $U$ by a 2CPDA. E.g. **( ( ) ( ( )** $* \; * * \; \in \; U$

$\mathsf{push}_1 a$ upon reading the first symbol, which must be **(**. Thereafter:

- $\mathsf{push}_2$ **;** $\mathsf{push}_1 a$ upon reading **(**
- $\mathsf{pop}_1$ upon reading **)**
- collapse upon reading first $*$, thereafter $\mathsf{pop}_2$ for each subsequent $*$.

| | |
|---|---|
| **(** | $[\,[\;a\;]\,]$ |
| **(** | $[\,[\;a\;]\,[\;a\;\;a\;]\,]$ |
| **)** | $[\,[\;a\;]\,[\;a\;]\,]$ |

## Recognising $U$ by a 2CPDA. E.g. **( ( ) ( ( )** $* \;\; * * \;\in\; U$

push$_1 a$ upon reading the first symbol, which must be **(**. Thereafter:

- push$_2$ **;** push$_1 a$ upon reading **(**
- pop$_1$ upon reading **)**
- collapse upon reading first $*$, thereafter pop$_2$ for each subsequent $*$.

| | |
|---|---|
| **(** | $[[\; a \;]]$ |
| **(** | $[[\; a \;][\; a \;\; a \;]]$ |
| **)** | $[[\; a \;][\; a \;]]$ |
| **(** | $[[\; a \;][\; a \;][\; a \;\; a \;]]$ |

## Recognising $U$ by a 2CPDA. E.g. **( ( ) ( ( )** $* \; * * \; \in \; U$

push$_1 a$ upon reading the first symbol, which must be **(**. Thereafter:

- push$_2$ **;** push$_1 a$ upon reading **(**
- pop$_1$ upon reading **)**
- collapse upon reading first $*$, thereafter pop$_2$ for each subsequent $*$.

| | |
|---|---|
| **(** | $[[\, a \,]]$ |
| **(** | $[[\, a \,][\, a \;\; a \,]]$ |
| **)** | $[[\, a \,][\, a \,]]$ |
| **(** | $[[\, a \,][\, a \,][\, a \;\; a \,]]$ |
| **(** | $[[\, a \,][\, a \,][\, a \;\; a \,][\, a \;\; a \;\; a \,]]$ |

## Recognising $U$ by a 2CPDA. E.g. **( ( ) ( ( )** $* \; * * \; \in \; U$

push$_1 a$ upon reading the first symbol, which must be **(**. Thereafter:

- push$_2$ **;** push$_1 a$ upon reading **(**
- pop$_1$ upon reading **)**
- collapse upon reading first $*$, thereafter pop$_2$ for each subsequent $*$.

| | |
|---|---|
| **(** | $[[\, a \,]]$ |
| **(** | $[[\, a \,][\, a \;\; a \,]]$ |
| **)** | $[[\, a \,][\, a \,]]$ |
| **(** | $[[\, a \,][\, a \,][\, a \;\; a \,]]$ |
| **(** | $[[\, a \,][\, a \,][\, a \;\; a \,][\, a \;\; a \;\; a \,]]$ |
| **)** | $[[\, a \,][\, a \,][\, a \;\; a \,][\, a \;\; a \,]]$ |

# Recognising $U$ by a 2CPDA. E.g. **( ( ) ( ( )** $* \; * * \; \in \; U$

$\mathrm{push}_1 a$ upon reading the first symbol, which must be **(**. Thereafter:

- $\mathrm{push}_2$ **;** $\mathrm{push}_1 a$ upon reading **(**
- $\mathrm{pop}_1$ upon reading **)**
- collapse upon reading first $*$, thereafter $\mathrm{pop}_2$ for each subsequent $*$.

| | | |
|---|---|---|
| **(** | $[[\ a\ ]]$ | |
| **(** | $[[\ a\ ][\ a\ \ a\ ]]$ | |
| **)** | $[[\ a\ ][\ a\ ]]$ | |
| **(** | $[[\ a\ ][\ a\ ][\ a\ \ a\ ]]$ | |
| **(** | $[[\ a\ ][\ a\ ][\ a\ \ a\ ][\ a\ \ a\ \ a\ ]]$ | |
| **)** | $[[\ a\ ][\ a\ ][\ a\ \ a\ ][\ a\ \ a\ ]]$ | Collapse! |
| $*$ | $[[\ a\ ][\ a\ ]]$ | |

## Recognising $U$ by a 2CPDA. E.g. **( ( ) ( ( )** $* \; * * \in U$

$\mathsf{push}_1 a$ upon reading the first symbol, which must be **(**. Thereafter:

- $\mathsf{push}_2$ **;** $\mathsf{push}_1 a$ upon reading **(**
- $\mathsf{pop}_1$ upon reading **)**
- collapse upon reading first $*$, thereafter $\mathsf{pop}_2$ for each subsequent $*$.

| | | |
|---|---|---|
| **(** | $[[\, a \,]]$ | |
| **(** | $[[\, a \,][\, a \;\; a \,]]$ | |
| **)** | $[[\, a \,][\, a \,]]$ | |
| **(** | $[[\, a \,][\, a \,][\, a \;\; a \,]]$ | |
| **(** | $[[\, a \,][\, a \,][\, a \;\; a \,][\, a \;\; a \;\; a \,]]$ | |
| **)** | $[[\, a \,][\, a \,][\, a \;\; a \,][\, a \;\; a \,]]$ | Collapse! |
| $*$ | $[[\, a \,][\, a \,]]$ | |
| $*$ | $[[\, a \,]]$ | |

## Recognising $U$ by a 2CPDA. E.g. **( ( ) ( ( )** $* \ * * \ \in \ U$

push$_1 a$ upon reading the first symbol, which must be **(**. Thereafter:

- push$_2$ **;** push$_1 a$ upon reading **(**
- pop$_1$ upon reading **)**
- collapse upon reading first $*$, thereafter pop$_2$ for each subsequent $*$.

| | | |
|---|---|---|
| **(** | $[\,[\ a\ ]\,]$ | |
| **(** | $[\,[\ a\ ]\,[\ a\ \ a\ ]\,]$ | |
| **)** | $[\,[\ a\ ]\,[\ a\ ]\,]$ | |
| **(** | $[\,[\ a\ ]\,[\ a\ ]\,[\ a\ \ a\ ]\,]$ | |
| **(** | $[\,[\ a\ ]\,[\ a\ ]\,[\ a\ \ a\ ]\,[\ a\ \ a\ \ a\ ]\,]$ | |
| **)** | $[\,[\ a\ ]\,[\ a\ ]\,[\ a\ \ a\ ]\,[\ a\ \ a\ ]\,]$ | Collapse! |
| $*$ | $[\,[\ a\ ]\,[\ a\ ]\,]$ | |
| $*$ | $[\,[\ a\ ]\,]$ | |
| $*$ | $[\,[\ ]\,]$ | |

## 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 $M_{c'}$.

# Parity games over collapsible $n$-pushdown 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 systems (CPDS) $A$ to an equivalent order-$n$ (non-deterministic) recursion schemes $G_A$.

**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 $\mathrm{Gr}(A)$ that unfolds to $\lambda(G_A)$).

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

Thus, for any parity game over a collapsible $n$-pushdown graph $\mathcal{CG}_A$, there is an equivalent finite acceptance parity game, which is an appropriate product of $\mathrm{Gr}(A)$ 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**. ***SafeRecSch***$_2\Sigma \subset$ ***RecSchTree***$_2\Sigma$ I.e. There are *inherently* unsafe trees (at order 2).

   Candidate: Urzyczyn's tree.

2. Define graphs generated by order-$n$ recursion schemes to be $\epsilon$-closures of configuration graphs of order-$n$ collapsible PDA? Are their MSO theories decidable?

   Matthew Hague's work.

   Notions of graphs definable by order-$n$ recursion schemes.

3. "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").