# Model Checking Higher-Order Computation: I

Luke Ong

Computing Laboratory, University of Oxford

Marktoberdorf Summer School, 4-15 August 2009

## Model checking and computer-aided verification

Beginning in the 80s, computer-aided verification (notably model checking) of finite-state systems (e.g. hardware and communication protocols) has been a great success story in computer science.

**Clarke, Emerson and Sifakis won the 2007 ACM Turing Award**

"for their rôle in developing model checking into a highly effective verification technology, widely adopted in hardware and software industries".

Focus of past decade: transfer of these techniques to software verification.

# Model checking and computer-aided verification

Beginning in the 80s, computer-aided verification (notably model checking) of finite-state systems (e.g. hardware and communication protocols) has been a great success story in computer science.

**Clarke, Emerson and Sifakis won the 2007 ACM Turing Award**

"for their rôle in developing model checking into a highly effective verification technology, widely adopted in hardware and software industries".

Focus of past decade: transfer of these techniques to software verification.

Beginning in the 80s, computer-aided verification (notably model checking) of finite-state systems (e.g. hardware and communication protocols) has been a great success story in computer science.

**Clarke, Emerson and Sifakis won the 2007 ACM Turing Award**

"for their rôle in developing model checking into a highly effective verification technology, widely adopted in hardware and software industries".

Focus of past decade: transfer of these techniques to software verification.

# What is (software) model checking?

**Problem:** Given a system *Sys* (e.g. an OS), and given a desirable behavioural property *Spec* (e.g. deadlock freedom), does *Sys* satisfy *Spec*?

**The model checking approach:**

1. Find an abstract model $M$ of the system *Sys*.
2. Describe the property *Spec* as a formula $\varphi$ of a suitable logic.
3. Exhaustively check if $\varphi$ is violated by $M$.

Huge strides made in **verification of 1st-order imperative programs**.

Many tools: SLAM, Blast, Terminator, SatAbs, etc.

**Two key techniques:** State-of-the-art tools use

1. abstraction techniques, as exemplified by CEGAR (Counter-Example Guided Abstraction Refinement)
2. acceleration methods such as SAT- and SMT-solvers.

# What is (software) model checking?

**Problem:** Given a system *Sys* (e.g. an OS), and given a desirable behavioural property *Spec* (e.g. deadlock freedom), does *Sys* satisfy *Spec*?

**The model checking approach:**

1. Find an abstract model $M$ of the system *Sys*.
2. Describe the property *Spec* as a formula $\varphi$ of a suitable logic.
3. Exhaustively check if $\varphi$ is violated by $M$.

Huge strides made in **verification of 1st-order imperative programs**.

Many tools: SLAM, Blast, Terminator, SatAbs, etc.

**Two key techniques:** State-of-the-art tools use

1. abstraction techniques, as exemplified by CEGAR (Counter-Example Guided Abstraction Refinement)
2. acceleration methods such as SAT- and SMT-solvers.

## Verification of higher-order programs

**Examples**: OCaml, F#, Haskell, Lisp/Scheme, Ptalon, etc.
By comparison with 1st-order imperative program, the model checking of higher-order programs is in its infancy.

Some theoretical advances in recent years; very little tool development.

**Model-checking higher-order programs is hard**:

1. Infinite-state and extremely complex: Even without recursion, higher-order programs over a finite base type are infinite-state.

   (Other sources of infinity: data structures and manipulation, control structures (with recursion), asynchronous communication, real-time and embedded systems, systems with parameters etc.)

2. Models of higher-order features as studied in semantics – are typically too "abstract" to support any algorithmic analysis.

   (A notable exception is game semantics.)

**Examples**: OCaml, F#, Haskell, Lisp/Scheme, Ptalon, etc.
By comparison with 1st-order imperative program, the model checking of higher-order programs is in its infancy.

Some theoretical advances in recent years; very little tool development.

**Model-checking higher-order programs is hard**:

1. Infinite-state and extremely complex: Even without recursion, higher-order programs over a finite base type are infinite-state.

   (Other sources of infinity: data structures and manipulation, control structures (with recursion), asynchronous communication, real-time and embedded systems, systems with parameters etc.)

2. Models of higher-order features as studied in semantics – are typically too "abstract" to support any algorithmic analysis.

   (A notable exception is game semantics.)

# Verifying higher-order programs: a worthwhile challenge

1. **Widely used in diverse domains**. Succinct, less error-prone, easy to write and hence good for prototyping; performance (of e.g. F#) approaching $C{+}{+}$.

Traditional applications: theorem proving and reasoning assistance, computational linguistics, programming language processing.

More recently: databases, networking, internet search (Google's MapReduce), trading and investment banking.
See Wadler's page "Functional Programming in the Real World"[1]

2. **Many hard theoretical problems**: E.g. termination analysis, higher-order matching, and (contextual) reachability analysis.

**Our goal**: To use semantic methods, in conjunction with algorithmic ideas and techniques from Verification, to formally analyze programming situations in which higher-order features are important.

[1] http://homepages.inf.ed.ac.uk/wadler/realworld/

# Verifying higher-order programs: a worthwhile challenge

1. **Widely used in diverse domains**. Succinct, less error-prone, easy to write and hence good for prototyping; performance (of e.g. F#) approaching $C$++.

Traditional applications: theorem proving and reasoning assistance, computational linguistics, programming language processing.

More recently: databases, networking, internet search (Google's MapReduce), trading and investment banking.
See Wadler's page "Functional Programming in the Real World"[1]

2. **Many hard theoretical problems**: E.g. termination analysis, higher-order matching, and (contextual) reachability analysis.

**Our goal**: To use semantic methods, in conjunction with algorithmic ideas and techniques from Verification, to formally analyze programming situations in which higher-order features are important.

[1] http://homepages.inf.ed.ac.uk/wadler/realworld/

# Verifying higher-order programs: a worthwhile challenge

1. **Widely used in diverse domains**. Succinct, less error-prone, easy to write and hence good for prototyping; performance (of e.g. F#) approaching $C++$.

Traditional applications: theorem proving and reasoning assistance, computational linguistics, programming language processing.

More recently: databases, networking, internet search (Google's MapReduce), trading and investment banking.
See Wadler's page "Functional Programming in the Real World"[1]

2. **Many hard theoretical problems**: E.g. termination analysis, higher-order matching, and (contextual) reachability analysis.

**Our goal**: To use semantic methods, in conjunction with algorithmic ideas and techniques from Verification, to formally analyze programming situations in which higher-order features are important.

[1] `http://homepages.inf.ed.ac.uk/wadler/realworld/`

## Lecture Course: Aim and Overview

### Aim

To introduce a systematic approach to the algorithmics of infinite structures generated by families of higher-order generators, suitable as a basis for model checking a wide range of behavioural properties of higher-order functional programs.

4 lectures.

### Part 1: Background and Survey

1. Families of Generators of Higher-Order Infinite Structures
2. Survey of Algorithmic Model Theory

### Part 2: Some Theory and Application

1. Type Theory and Modal Mu-Calculus Model Checking
2. Application: Model Checking Functional Programs

# Outline I

1. Relating (Families of) Generators of Infinite Structures
   - Higher-Order Pushdown Automata
   - Higher-Order Recursion Schemes
   - Relating the Generator Families: Word Languages

2. Recursion Schemes and their Algorithmic Model Theory
   - Q1: Decidability of MSO / Modal Mu-Calculus Theories
   - Q2: Machine Characterisation by Collapsible Pushdown Automata
   - Q3: Expressivity: *The Safety Conjecture*
   - Q4: Infinite Graphs Generated by Recursion Schemes / CPDA

# Outline

### Order-2 pushdown automata

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. Top of stack is at the righthand end.

$$
\begin{array}{rcll}
push_2 & : & [s_1 \cdots s_{i-1} \underbrace{[a_1 \cdots a_n]}_{s_i}] & \mapsto & [s_1 \cdots s_{i-1}\, s_i\, s_i] \\
pop_2 & : & [s_1 \cdots s_{i-1} [a_1 \cdots a_n]] & \mapsto & [s_1 \cdots s_{i-1}] \\
push_1\, a & : & [s_1 \cdots s_{i-1} [a_1 \cdots a_n]] & \mapsto & [s_1 \cdots s_{i-1} [a_1 \cdots a_n\, a]] \\
pop_1 & : & [s_1 \cdots s_{i-1} [a_1 \cdots a_n\, a_{n+1}]] & \mapsto & [s_1 \cdots s_{i-1} [a_1 \cdots a_n]]
\end{array}
$$

Idea extends to all finite orders: an order-$n$ PDA has an order-$n$ stack, and has $push_i$ and $pop_i$ for each $1 \leq i \leq n$.

### Order-2 pushdown automata

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. Top of stack is at the righthand end.

$$
\begin{array}{rcllcl}
push_2 & : & [s_1 \cdots s_{i-1} \underbrace{[a_1 \cdots a_n]}_{s_i}] & \mapsto & [s_1 \cdots s_{i-1}\, s_i\, s_i] \\
pop_2 & : & [s_1 \cdots s_{i-1} [a_1 \cdots a_n]] & \mapsto & [s_1 \cdots s_{i-1}] \\
push_1\, a & : & [s_1 \cdots s_{i-1} [a_1 \cdots a_n]] & \mapsto & [s_1 \cdots s_{i-1} [a_1 \cdots a_n\, a]] \\
pop_1 & : & [s_1 \cdots s_{i-1} [a_1 \cdots a_n\, a_{n+1}]] & \mapsto & [s_1 \cdots s_{i-1} [a_1 \cdots a_n]]
\end{array}
$$

Idea extends to all finite orders: an order-$n$ PDA has an order-$n$ stack, and has $push_i$ and $pop_i$ for each $1 \leq i \leq n$.

### Order-2 pushdown automata

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. Top of stack is at the righthand end.

$$
\begin{array}{rcll}
push_2 & : & [s_1 \cdots s_{i-1} \underbrace{[a_1 \cdots a_n]}_{s_i}] & \mapsto \; [s_1 \cdots s_{i-1}\, s_i\, s_i] \\
pop_2 & : & [s_1 \cdots s_{i-1} [a_1 \cdots a_n]] & \mapsto \; [s_1 \cdots s_{i-1}] \\
push_1\, a & : & [s_1 \cdots s_{i-1} [a_1 \cdots a_n]] & \mapsto \; [s_1 \cdots s_{i-1} [a_1 \cdots a_n\, a]] \\
pop_1 & : & [s_1 \cdots s_{i-1} [a_1 \cdots a_n\, a_{n+1}]] & \mapsto \; [s_1 \cdots s_{i-1} [a_1 \cdots a_n]]
\end{array}
$$

Idea extends to all finite orders: an order-$n$ PDA has an order-$n$ stack, and has $push_i$ and $pop_i$ for each $1 \leq i \leq n$.

# HOPDA as recognizers of word languages

HOPDA can be used as recognizing/generating device for

1. finite-word languages (Maslov 74) (and $\omega$-word languages)

$$\langle\, \Sigma, Q, q_0, \Gamma, \Delta \subseteq (\Sigma \cup \{\, \epsilon \,\}) \times Q \times \Gamma \times Op_n \times Q, F \,\rangle$$

2. possibly-infinite (ranked) trees (KNU01), and tree languages
3. possibly infinite graphs (Muller+Schupp 86, Courcelle 95, Cachat 03)

Some basic facts (Maslov 74, 76):

1. HOPDA define an infinite hierarchy of word languages.
2. Low orders are well-known: orders 0, 1 and 2 are the regular, context free, and indexed languages (Aho 68). Higher-order languages are poorly understood.
3. For each $n \geq 0$, the order-$n$ languages form an abstract family of languages (closed under $+, \cdot, (-)^*$, intersection with regular languages, homomorphism and inverse homo.)
4. For each $n \geq 0$, the emptiness problem for order-$n$ PDA is decidable.

# HOPDA as recognizers of word languages

HOPDA can be used as recognizing/generating device for

1. finite-word languages (Maslov 74) (and $\omega$-word languages)

$$\langle \Sigma, Q, q_0, \Gamma, \Delta \subseteq (\Sigma \cup \{ \epsilon \}) \times Q \times \Gamma \times Op_n \times Q, F \rangle$$

2. possibly-infinite (ranked) trees (KNU01), and tree languages
3. possibly infinite graphs (Muller+Schupp 86, Courcelle 95, Cachat 03)

**Some basic facts (Maslov 74, 76):**

1. HOPDA define an infinite hierarchy of word languages.
2. Low orders are well-known: orders 0, 1 and 2 are the regular, context free, and indexed languages (Aho 68). Higher-order languages are poorly understood.
3. For each $n \geq 0$, the order-$n$ languages form an abstract family of languages (closed under $+, \cdot, (-)^*$, intersection with regular languages, homomorphism and inverse homo.)
4. For each $n \geq 0$, the emptiness problem for order-$n$ PDA is decidable.

$L$ is not context free. Use the "*uvwxy* Lemma".

**Idea**: Use top 1-stack to process $a^n b^n$, and height of 2-stack to remember $n$.

$$
\begin{tikzcd}
q_1\,[\,[\,]\,] \arrow[r, "a"] & q_1\,[\,[\,]\,[z]\,] \arrow[r, "a"] & q_1\,[\,[\,]\,[z]\,[zz]\,] \arrow[d, "b"] \\
 & & q_2\,[\,[\,]\,[z]\,[z]\,] \arrow[d, "b"] \\
q_3\,[\,[\,]\,] & q_3\,[\,[\,]\,[z]\,] \arrow[l, "c"'] & q_2\,[\,[\,]\,[z]\,[\,]\,] \arrow[l, "c"']
\end{tikzcd}
$$

---

$$
\begin{tikzcd}
q_1 \arrow[loop above, "- \xrightarrow{a} push_2\,;\, push_1 z"] \arrow[r, "z \xrightarrow{b} pop_1 z"'] & q_2 \arrow[loop above, "z \xrightarrow{b} pop_1"] \arrow[r, "\perp \xrightarrow{c} pop_2"'] & q_3 \arrow[loop above, "z \xrightarrow{c} pop_2"]
\end{tikzcd}
$$

'read $a$' 'read $b$' 'read $c$'

$L$ is not context free. Use the "*uvwxy* Lemma".

**Idea**: Use top 1-stack to process $a^n\, b^n$, and height of 2-stack to remember $n$.

$$
\begin{tikzcd}
q_1\,[[\,]] \arrow[r, "a"] & q_1\,[[\,]\,[z]] \arrow[r, "a"] & q_1\,[[\,]\,[z]\,[zz]] \arrow[d, "b"] \\
 & & q_2\,[[\,]\,[z]\,[z]] \arrow[d, "b"] \\
q_3\,[[\,]] & q_3\,[[\,]\,[z]] \arrow[l, "c"'] & q_2\,[[\,]\,[z]\,[\,]] \arrow[l, "c"']
\end{tikzcd}
$$

---

## Example: $L := \{\, a^n\, b^n\, c^n : n \geq 0 \,\}$ is recognizable by an order-2 PDA

$L$ is not context free. Use the "*uvwxy* Lemma".

**Idea**: Use top 1-stack to process $a^n\, b^n$, and height of 2-stack to remember $n$.

$$
\begin{tikzcd}
q_1\,{[\,[\,]\,]} \arrow[r, "a"] & q_1\,{[\,[\,]\,[z]\,]} \arrow[r, "a"] & q_1\,{[\,[\,]\,[z]\,[zz]\,]} \arrow[d, "b"] \\
 & & q_2\,{[\,[\,]\,[z]\,[z]\,]} \arrow[d, "b"] \\
q_3\,{[\,[\,]\,]} & q_3\,{[\,[\,]\,[z]\,]} \arrow[l, "c"'] & q_2\,{[\,[\,]\,[z]\,[\,]\,]} \arrow[l, "c"']
\end{tikzcd}
$$

$$
\begin{tikzcd}
q_1 \arrow[loop above, "- \xrightarrow{a} push_2\,;\, push_1 z"] \arrow[r, "z \xrightarrow{b} pop_1 z"'] & q_2 \arrow[loop above, "z \xrightarrow{b} pop_1"] \arrow[r, "\perp \xrightarrow{c} pop_2"'] & q_3 \arrow[loop above, "z \xrightarrow{c} pop_2"]
\end{tikzcd}
$$

'read $a$' 'read $b$' 'read $c$'

## Example: $L := \{ a^n b^n c^n : n \geq 0 \}$ is recognizable by an order-2 PDA

$L$ is not context free. Use the "*uvwxy* Lemma".

**Idea**: Use top 1-stack to process $a^n b^n$, and height of 2-stack to remember $n$.

$$
\begin{tikzcd}
q_1\,[[\,]] \arrow[r, "a"] & q_1\,[[\,]\,[z]] \arrow[r, "a"] & q_1\,[[\,]\,[z]\,[zz]] \arrow[d, "b"] \\
& & q_2\,[[\,]\,[z]\,[z]] \arrow[d, "b"] \\
q_3\,[[\,]] & q_3\,[[\,]\,[z]] \arrow[l, "c"'] & q_2\,[[\,]\,[z]\,[\,]] \arrow[l, "c"']
\end{tikzcd}
$$

---

$$
\begin{tikzcd}
q_1 \arrow[loop above, "- \xrightarrow{a} push_2\,;\, push_1 z"] \arrow[r, "z \xrightarrow{b} pop_1 z"'] & q_2 \arrow[loop above, "z \xrightarrow{b} pop_1"] \arrow[r, "\bot \xrightarrow{c} pop_2"'] & q_3 \arrow[loop above, "z \xrightarrow{c} pop_2"]
\end{tikzcd}
$$

'read $a$'   'read $b$'   'read $c$'

# Pumping Lemma for Context-Free Languages

**Theorem ($uvwxy$)**

*Let $L$ be an infinite CFL. Every word in $L$ longer then $p$ can be written as a concatenation of subwords, $u\,v\,w\,x\,y$, such that $|v\,w\,x| \leq p$, $|v\,x| \geq 1$, and for every $i \geq 0$, $u\,v^i\,w\,x^i\,y$ is in $L$.*

# A reminder: simple types

**Types** $\quad A \quad ::= \quad \mathsf{o} \quad | \quad (A \to B)$

Every type can be written uniquely as

$$A_1 \to (A_2 \cdots \to (A_n \to \mathsf{o}) \cdots), \quad n \geq 0$$

often abbreviated to $A_1 \to A_2 \cdots \to A_n \to \mathsf{o}$.

**Order** of a type: measures "nestedness" on LHS of $\to$.

$$\begin{aligned} \mathrm{order}(\mathsf{o}) &= 0 \\ \mathrm{order}(A \to B) &= \max(\mathrm{order}(A) + 1, \mathrm{order}(B)) \end{aligned}$$

**Examples.** $\mathbb{N} \to \mathbb{N}$ and $\mathbb{N} \to (\mathbb{N} \to \mathbb{N})$ both have order 1; $(\mathbb{N} \to \mathbb{N}) \to \mathbb{N}$ has order 2.

**Notation.** $\quad e : A \quad$ means "expression $e$ has type $A$".

# A reminder: simple types

**Types** $\quad A \quad ::= \quad o \quad | \quad (A \to B)$

Every type can be written uniquely as

$$A_1 \to (A_2 \cdots \to (A_n \to o) \cdots ), \quad n \geq 0$$

often abbreviated to $A_1 \to A_2 \cdots \to A_n \to o$.

**Order** of a type: measures "nestedness" on LHS of $\to$.

$$\begin{array}{rcl} \mathrm{order}(o) & = & 0 \\ \mathrm{order}(A \to B) & = & \max(\mathrm{order}(A) + 1, \mathrm{order}(B)) \end{array}$$

**Examples.** $\mathbb{N} \to \mathbb{N}$ and $\mathbb{N} \to (\mathbb{N} \to \mathbb{N})$ both have order 1;
$(\mathbb{N} \to \mathbb{N}) \to \mathbb{N}$ has order 2.

**Notation.** $\quad e : A \quad$ means "expression $e$ has type $A$".

## Higher-order recursion schemes [Par68, Niv72, NC78, Dam82,...]

An order-$n$ recursion scheme = closed ground-type term definable in order-$n$ fragment of simply-typed $\lambda$-calculus with recursion and uninterpreted order-1 constant symbols.

**Example: An order-1 recursion scheme**. Fix ranked alphabet $\Sigma = \{ f : 2, g : 1, a : 0 \}$.

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

Unfolding from the start symbol $S$:

$$\begin{array}{rcl} S & \to & F\,a \\ & \to & f\,a\,(F\,(g\,a)) \\ & \to & f\,a\,(f\,(g\,a)\,(F\,(g\,(g\,a)))) \\ & \to & \cdots \end{array}$$

The (term-)tree thus generated, $[\![ G ]\!]$, is $f\,a\,(f\,(g\,a)\,(f\,(g\,(g\,a))(\cdots)))$.

An order-$n$ recursion scheme = closed ground-type term definable in order-$n$ fragment of simply-typed $\lambda$-calculus with recursion and uninterpreted order-1 constant symbols.

**Example: An order-1 recursion scheme**. Fix ranked alphabet $\Sigma = \{ f : 2, g : 1, a : 0 \}$.

$$G \ : \ \left\{ \begin{array}{rcl} S & = & F\,a \\ F\,x & = & f\,x\,(F\,(g\,x)) \end{array} \right.$$

Unfolding from the start symbol $S$:

$$\begin{array}{rcl} S & \rightarrow & F\,a \\ & \rightarrow & f\,a\,(F\,(g\,a)) \\ & \rightarrow & f\,a\,(f\,(g\,a)\,(F\,(g\,(g\,a)))) \\ & \rightarrow & \cdots \end{array}$$

The (term-)tree thus generated, $[\![\, G \,]\!]$, is $f\,a\,(f\,(g\,a)\,(f\,(g\,(g\,a))\,(\cdots)))$.

## Higher-order recursion schemes [Par68, Niv72, NC78, Dam82,...]

An order-$n$ recursion scheme = closed ground-type term definable in order-$n$ fragment of simply-typed $\lambda$-calculus with recursion and uninterpreted order-1 constant symbols.

**Example: An order-1 recursion scheme**. Fix ranked alphabet $\Sigma = \{ f : 2, g : 1, a : 0 \}$.

$$G \; : \; \begin{cases} S & = & F\,a \\ F\,x & = & f\,x\,(F\,(g\,x)) \end{cases}$$

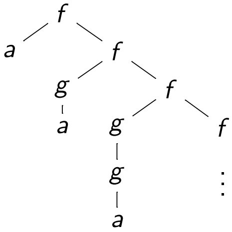
Unfolding from the start symbol $S$:

$$\begin{array}{lll} S & \rightarrow & F\,a \\ & \rightarrow & f\,a\,(F\,(g\,a)) \\ & \rightarrow & f\,a\,(f\,(g\,a)\,(F\,(g\,(g\,a)))) \\ & \rightarrow & \cdots \end{array}$$

The (term-)tree thus generated, $[\![ G ]\!]$, is $f\,a\,(f\,(g\,a)\,(f\,(g\,(g\,a))(\cdots)))$.

# Representing the term-tree $[\![ G ]\!]$ as a $\Sigma$-labelled tree

$[\![ G ]\!] = f\, a\, (f\, (g\, a)\, (f\, (g\, (g\, a))(\cdots)))$ is the (term-)tree

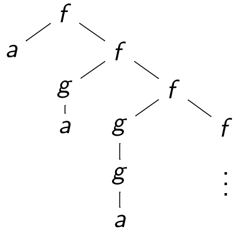



We view the infinite term $[\![ G ]\!]$ as a $\Sigma$-labelled tree, formally, a map $T \longrightarrow \Sigma$, where $T$ is a prefix-closed subset of $Dir^*$, with $Dir$ a set of edge labels.

Formally term-trees such as $[\![ G ]\!]$ are ranked and ordered.

## Representing the term-tree $[\![ G ]\!]$ as a $\Sigma$-labelled tree

$[\![ G ]\!] = f\,a\,(f\,(g\,a)\,(f\,(g\,(g\,a))(\cdots)))$ is the (term-)tree



We view the infinite term $[\![ G ]\!]$ as a $\Sigma$-labelled tree, formally, a map $T \longrightarrow \Sigma$, where $T$ is a prefix-closed subset of $Dir^*$, with $Dir$ a set of edge labels.

Formally term-trees such as $[\![ G ]\!]$ are ranked and ordered.

## Definition: 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: $f \in \Sigma$ has arity $\mathrm{ar}(f) \geq 0$
- $\mathcal{R}$: An equation for each non-terminal $D : A_1 \to \cdots \to A_m \to o$ of shape

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

  where the term $e : o$ is constructed from
  - terminals $f, g, a$, etc. from $\Sigma$
  - variables $\varphi_1 : A_1, \cdots, \varphi_m : A_m$ from $\mathit{Var}$,
  - non-terminals $D, F, G$, etc. from $\mathcal{N}$.

  using the application rule: If $s : A \to B$ and $t : A$ then $(s\,t) : B$.

Given a term $t$, define a (finite) tree $t^\perp$ by

$$t^\perp \;:=\; \begin{cases} f & \text{if } t \text{ is a terminal } f \\ t_1^\perp \, t_2^\perp & \text{if } t = t_1 \, t_2 \text{ and } t_1^\perp \neq \perp \\ \perp & \text{otherwise} \end{cases}$$

We extend the flat partial order on $\Sigma$ (i.e. $\perp \leq a$ for all $a \in \Sigma$) to trees by:

$$s \leq t \;:=\; \forall \alpha \in \mathrm{dom}(s) \,.\, \alpha \in \mathrm{dom}(t) \wedge s(\alpha) \leq t(\alpha)$$

E.g. $\perp \leq f \perp \perp \leq f \perp b \leq f a b$.

For a directed set $T$ of trees, we write $\bigsqcup T$ for the lub of $T$ w.r.t. $\leq$.

Let $G$ be a recursion scheme. We define the tree generated by $G$ by

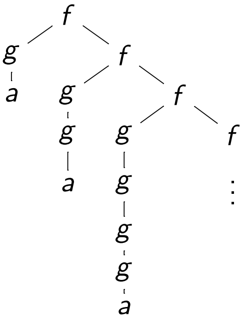
$$[\![\, G \,]\!] \;:=\; \bigsqcup \{\, t^\perp \mid S \to^* t \,\}$$

# An order-2 example

$\Sigma = \{ f : 2, g : 1, a : 0 \}$.
$S : o, \quad B : (o \to o) \to (o \to o) \to o \to o, \quad F : (o \to o) \to o$

$$G_2 \; : \; \left\{ \begin{array}{rcl} S & = & F\, g \\ B\,\varphi\,\psi\,x & = & \varphi\,(\psi\,x) \\ F\,\varphi & = & f\,(\varphi\,a)\,(F\,(B\,\varphi\,\varphi)) \end{array} \right.$$

The generated tree, $[\![ G_2 ]\!] : \{ 1, 2 \}^* \longrightarrow \Sigma$, is:



## Using recursion schemes as generators of word languages

**Idea:** A word is just a linear tree.

Represent a finite word "$a\,b\,c$" (say) as the applicative term $a\,(b\,(c\,e))$, viewing $a, b$ and $c$ as symbols of arity 1, where $e$ is the arity-0 end-of-word marker.

Fix an input alphabet $\Sigma$. We can use a (non-deterministic) recursion scheme to generate finite-word languages, with ranked alphabet $\overline{\Sigma} := \{\, a : 1 \mid a \in \Sigma \,\} \cup \{\, e : 0 \,\}$.

**Example**. $\{\, a^n\, b^n \mid n \geq 0 \,\}$ is generated by order-1 recursion scheme:

$$\left\{ \begin{array}{rcl} S & \rightarrow & F\,e \\ F\,x & \rightarrow & a\,(F\,(b\,x)) \quad | \quad x \end{array} \right.$$

# Exercises

1. Find an order-2 (word-language) recursion scheme that generates $L = \{ a^i b^i c^i \mid i \geq 0 \}$.
2. Prove that context-free languages are equivalent to languages generated by order-1 (word-language) recursion schemes.

**Answer to 1.**

$$\left\{ \begin{array}{rcl} S & \rightarrow & F\,I\,e \\ F\,\varphi\,x & \rightarrow & \varphi\,x \quad \mid \quad F\,(H\,\varphi)\,(c\,x) \\ H\,\varphi\,y & \rightarrow & a\,(\varphi\,(b\,y)) \\ I\,x & \rightarrow & x \end{array} \right.$$

# Relating the two generator-families: word-language case

**Theorem (Equi-expressivity)**

*For each $n \geq 0$, the three formalisms*

1. *order-n pushdown automata (Maslov 76)*
2. *order-n safe recursion schemes (Damm 82, Damm + Goerdt 86)*
3. *order-n indexed grammars (Maslov 76)*

*generate the same class of word languages.*

What is safety? (See later.)

# Maslov Hierarchy: Many Open Problems

1. Pumping Lemma, Myhill-Nerode, and Parikh Theorems.
   Weak "pumping lemmas" for levels 1 and 2 (Hayashi 73, Gilman 96).
   *Pace* (Blumensath 04) for Maslov Hierarchy – but runs (*not* plays) are pumpable, conditions given as lengths of runs and configuration size.
2. Logical characterisations.
   E.g. MSOL for regular languages (Büchi 60). Characterisation of CFL using quantification over matchings (LST 94).
3. Complexity-theoretic characterisations.
   *Pace* (Engelfriet 83, 91): characterisations of languages accepted by alternating / two-way / multi-head / space-auxiliary order-$n$ PDA as time-complexity classes (but no result for Maslov Hierarchy itself)
4. Relationship with Chomsky Hierachy. E.g. is level 3 context-sensitive?

## Why study the two families of generators?

They are relevant to both semantics and verification:

1. Recursion schemes are an old and influential formalism for the semantical analysis of imperative and functional programs (Nivat 75, Damm 82). They are a compelling model of computation for higher-order functional programs.
2. Pushdown automata characterize the control flow of 1st-order (recursive) procedural programs.
   Pushdown checkers (e.g. MOPED) are essential back-end engines of state-of-the-art software model checkers (e.g. SLAM, Terminator).
3. Higher-order (collapsible) pushdown automata are highly accurate models of computation of higher-order procedural programs.

## Why study the two families of generators?

They are relevant to both semantics and verification:

1. Recursion schemes are an old and influential formalism for the semantical analysis of imperative and functional programs (Nivat 75, Damm 82). They are a compelling model of computation for higher-order functional programs.
2. Pushdown automata characterize the control flow of 1st-order (recursive) procedural programs.
   Pushdown checkers (e.g. MOPED) are essential back-end engines of state-of-the-art software model checkers (e.g. SLAM, Terminator).
3. Higher-order (collapsible) pushdown automata are highly accurate models of computation of higher-order procedural programs.

## Why study the two families of generators?

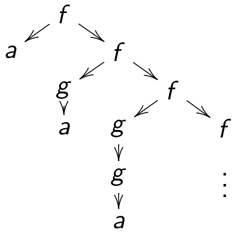
They are relevant to both semantics and verification:

1. Recursion schemes are an old and influential formalism for the semantical analysis of imperative and functional programs (Nivat 75, Damm 82). They are a compelling model of computation for higher-order functional programs.
2. Pushdown automata characterize the control flow of 1st-order (recursive) procedural programs.
   Pushdown checkers (e.g. MOPED) are essential back-end engines of state-of-the-art software model checkers (e.g. SLAM, Terminator).
3. Higher-order (collapsible) pushdown automata are highly accurate models of computation of higher-order procedural programs.

# Outline

# A challenge problem in higher-order verification

**Example**: Consider $[\![ G ]\!]$ on the right




- $\varphi_1 =$ "Infinitely many $f$-nodes are reachable".
- $\varphi_2 =$ "Only finitely many $g$-nodes are reachable".

Every node on the tree satisfies $\varphi_1 \vee \varphi_2$.

Let **RecSchTree**$_n$ be the class of $\Sigma$-labelled trees generated by order-$n$ recursion schemes.

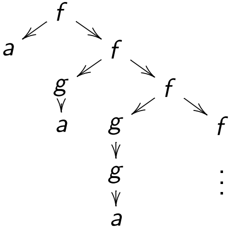
Is the "MSO Model-Checking Problem for **RecSchTree**$_n$" decidable?

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

# A challenge problem in higher-order verification

**Example**: Consider $[\![ G ]\!]$ on the right

- $\varphi_1 =$ "Infinitely many $f$-nodes are reachable".
- $\varphi_2 =$ "Only finitely many $g$-nodes are reachable".




Every node on the tree satisfies $\varphi_1 \vee \varphi_2$.

Let $\mathbf{RecSchTree}_n$ be the class of $\Sigma$-labelled trees generated by order-$n$ recursion schemes.

**Is the "MSO Model-Checking Problem for $\mathbf{RecSchTree}_n$" decidable?**

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

Because it is the gold standard of logics for describing model-checking properties.

- MSO is *very* expressive. Over graphs, MSO is more expressive than the modal mu-calculus, into which all standard temporal logics (e.g. LTL, CTL, CTL$*$, etc.) can embed.
- It is hard to extend MSO meaningfully without sacrificing decidability where it holds.

## What is MSO logic?

## Why study MSO logic?

Because it is the gold standard of logics for describing model-checking properties.

- MSO is *very* expressive. Over graphs, MSO is more expressive than the modal mu-calculus, into which all standard temporal logics (e.g. LTL, CTL, CTL$*$, etc.) can embed.
- It is hard to extend MSO meaningfully without sacrificing decidability where it holds.

**What is MSO logic?**

## Review: Representing trees as logical structures

Represent a $\Sigma$-labelled tree $t : \mathrm{dom}(t) \longrightarrow \Sigma$ as a logical structure

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

- Parent-child relationship between nodes: $\mathbf{d}_i(x, y) \equiv$ "$y$ is $i$-child of $x$"
- Node labelling: $\mathbf{p}_f(x) \equiv$ "$x$ has label $f$" where $f$ is a $\Sigma$-symbol

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

First-order variables: $x, y, z$, etc. (ranging over nodes)

Second-order variables: $X, Y, Z$, etc. (ranging over sets of nodes i.e. monadic relations)

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

- Parent-child relationship between nodes: $\mathbf{d}_i(x, y)$
- Node labelling: $\mathbf{p}_f(x)$
- Set-membership: $x \in X$

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

## Examples of MSO-definable properties

Several useful relations are definable:

1. Set inclusion (and hence equality): $X \subseteq Y \;\equiv\; \forall x \,.\, x \in X \to x \in Y$.
2. "Is-an-ancestor-of" or prefix ordering $x \leq y$ (and hence $x = y$):

$$\begin{array}{rcl} \mathrm{PrefCl}(X) & \equiv & \forall x, y \,.\, y \in X \wedge \bigvee_{i=1}^{m} \mathbf{d}_i(x,y) \;\to\; x \in X \\ x \leq y & \equiv & \forall X \,.\, \mathrm{PrefCl}(X) \wedge y \in X \;\to\; x \in X \end{array}$$

**Reachability property:** "$X$ is a path"

$$\begin{array}{rcl} \mathrm{Path}(X) & \equiv & \forall x, y \in X \,.\, x \leq y \vee y \leq x \\ & \wedge & \forall x, y, z \,.\, x \in X \wedge z \in X \wedge x \leq y \leq z \;\to\; y \in X \end{array}$$

$\mathrm{MaxPath}(X) \equiv \mathrm{Path}(X) \wedge \forall Y \,.\, \mathrm{Path}(Y) \wedge X \subseteq Y \;\to\; Y \subseteq X$.

## E.g. MSO can expresss "$\exists$ infinitely many $f$-labelled nodes"

A set of nodes is a cut if no two nodes in it are $\leq$-compatible, and it has a non-empty intersection with every maximal path.

$$\begin{array}{rcl} \mathrm{Cut}(X) & \equiv & \forall x, y \in X \,.\, \neg(x \leq y \ \vee \ y \leq x) \\ & \wedge & \forall Z \,.\, \mathrm{MaxPath}(Z) \ \rightarrow \ \exists z \in Z \,.\, z \in X \end{array}$$

**Lemma**

*A set $X$ of nodes in a finitely-branching tree is finite iff there is a cut $C$ such that every $X$-node is a prefix of some $C$-node.*

$$\mathrm{Finite}(X) \ \equiv \ \exists Y \,.\, \mathrm{Cut}(Y) \ \wedge \ \forall x \in X \,.\, \exists y \in Y \,.\, x \leq y$$

Hence "there are finitely many nodes labelled by $f$" is expressible in MSO by $\exists X \,.\, \mathrm{Finite}(X) \ \wedge \ \forall x \,.\, \mathbf{p}_f(x) \rightarrow x \in X$.

**But** "MSO cannot count": E.g. "$X$ has twice as many elements as $Y$".

# A (selective) survey of MSO-decidable structures: up to 2002

- Rabin 1969: Regular trees. "Mother of all decidability results in Verification."
- Muller and Schupp 1985: Configuration graphs of PDA.
- Caucal 1996 Prefix-recognizable graphs ($\epsilon$-closures of configuration graphs of pushdown automata, Stirling 2000).
- Knapik, Niwiński and Urzyczyn (TLCA 2001, FOSSACS 2002):
  $\mathbf{PushdownTree}_n\Sigma$ = Trees generated by order-$n$ pushdown automata.
  $\mathbf{SafeRecSchTree}_n\Sigma$ = Trees generated by order-$n$ safe rec. schemes.
- Subsuming all the above:
  Caucal (MFCS 2002). $\mathbf{CaucalTree}_n\Sigma$ and $\mathbf{CaucalGraph}_n\Sigma$.

**Theorem (KNU-Caucal 2002)**

*For $n \geq 0$, $\mathbf{PushdownTree}_n\Sigma = \mathbf{SafeRecSchTree}_n\Sigma = \mathbf{CaucalTree}_n\Sigma$; and they have decidable MSO theories.*

## A (selective) survey of MSO-decidable structures: up to 2002

- Rabin 1969: Regular trees. "Mother of all decidability results in Verification."
- Muller and Schupp 1985: Configuration graphs of PDA.
- Caucal 1996 Prefix-recognizable graphs ($\epsilon$-closures of configuration graphs of pushdown automata, Stirling 2000).
- Knapik, Niwiński and Urzyczyn (TLCA 2001, FOSSACS 2002):
  $\mathbf{PushdownTree}_n\Sigma$ = Trees generated by order-$n$ pushdown automata.
  $\mathbf{SafeRecSchTree}_n\Sigma$ = Trees generated by order-$n$ safe rec. schemes.
- Subsuming all the above:
  Caucal (MFCS 2002). $\mathbf{CaucalTree}_n\Sigma$ and $\mathbf{CaucalGraph}_n\Sigma$.

**Theorem (KNU-Caucal 2002)**

*For $n \geq 0$, $\mathbf{PushdownTree}_n\Sigma = \mathbf{SafeRecSchTree}_n\Sigma = \mathbf{CaucalTree}_n\Sigma$; and they have decidable MSO theories.*

# A (selective) survey of MSO-decidable structures: up to 2002

- Rabin 1969: Regular trees. "Mother of all decidability results in Verification."
- Muller and Schupp 1985: Configuration graphs of PDA.
- Caucal 1996 Prefix-recognizable graphs ($\epsilon$-closures of configuration graphs of pushdown automata, Stirling 2000).
- Knapik, Niwiński and Urzyczyn (TLCA 2001, FOSSACS 2002):
  $\mathbf{PushdownTree}_n\Sigma$ = Trees generated by order-$n$ pushdown automata.
  $\mathbf{SafeRecSchTree}_n\Sigma$ = Trees generated by order-$n$ safe rec. schemes.
- Subsuming all the above:
  Caucal (MFCS 2002). $\mathbf{CaucalTree}_n\Sigma$ and $\mathbf{CaucalGraph}_n\Sigma$.

**Theorem (KNU-Caucal 2002)**

*For $n \geq 0$, $\mathbf{PushdownTree}_n\Sigma = \mathbf{SafeRecSchTree}_n\Sigma = \mathbf{CaucalTree}_n\Sigma$; and they have decidable MSO theories.*

## A (selective) survey of MSO-decidable structures: up to 2002

- Rabin 1969: Regular trees. "Mother of all decidability results in Verification."
- Muller and Schupp 1985: Configuration graphs of PDA.
- Caucal 1996 Prefix-recognizable graphs ($\epsilon$-closures of configuration graphs of pushdown automata, Stirling 2000).
- Knapik, Niwiński and Urzyczyn (TLCA 2001, FOSSACS 2002): $\mathbf{PushdownTree}_n\Sigma$ = Trees generated by order-$n$ pushdown automata. $\mathbf{SafeRecSchTree}_n\Sigma$ = Trees generated by order-$n$ safe rec. schemes.
- Subsuming all the above: Caucal (MFCS 2002). $\mathbf{CaucalTree}_n\Sigma$ and $\mathbf{CaucalGraph}_n\Sigma$.

**Theorem (KNU-Caucal 2002)**

*For $n \geq 0$, $\mathbf{PushdownTree}_n\Sigma = \mathbf{SafeRecSchTree}_n\Sigma = \mathbf{CaucalTree}_n\Sigma$; and they have decidable MSO theories.*

# A (selective) survey of MSO-decidable structures: up to 2002

- Rabin 1969: Regular trees. "Mother of all decidability results in Verification."
- Muller and Schupp 1985: Configuration graphs of PDA.
- Caucal 1996 Prefix-recognizable graphs ($\epsilon$-closures of configuration graphs of pushdown automata, Stirling 2000).
- Knapik, Niwiński and Urzyczyn (TLCA 2001, FOSSACS 2002): $\mathbf{PushdownTree}_n\Sigma$ = Trees generated by order-$n$ pushdown automata. $\mathbf{SafeRecSchTree}_n\Sigma$ = Trees generated by order-$n$ safe rec. schemes.
- Subsuming all the above: Caucal (MFCS 2002). $\mathbf{CaucalTree}_n\Sigma$ and $\mathbf{CaucalGraph}_n\Sigma$.

**Theorem (KNU-Caucal 2002)**

*For $n \geq 0$, $\mathbf{PushdownTree}_n\Sigma = \mathbf{SafeRecSchTree}_n\Sigma = \mathbf{CaucalTree}_n\Sigma$; and they have decidable MSO theories.*

# A (selective) survey of MSO-decidable structures: up to 2002

- Rabin 1969: Regular trees. "Mother of all decidability results in Verification."
- Muller and Schupp 1985: Configuration graphs of PDA.
- Caucal 1996 Prefix-recognizable graphs ($\epsilon$-closures of configuration graphs of pushdown automata, Stirling 2000).
- Knapik, Niwiński and Urzyczyn (TLCA 2001, FOSSACS 2002):
  $\mathbf{PushdownTree}_n\Sigma$ = Trees generated by order-$n$ pushdown automata.
  $\mathbf{SafeRecSchTree}_n\Sigma$ = Trees generated by order-$n$ safe rec. schemes.
- Subsuming all the above:
  Caucal (MFCS 2002). $\mathbf{CaucalTree}_n\Sigma$ and $\mathbf{CaucalGraph}_n\Sigma$.

**Theorem (KNU-Caucal 2002)**

*For $n \geq 0$, $\mathbf{PushdownTree}_n\Sigma = \mathbf{SafeRecSchTree}_n\Sigma = \mathbf{CaucalTree}_n\Sigma$; and they have decidable MSO theories.*

Safety is a set of constraints on where variables may occur in a term.

**Definition (Damm TCS 82, KNU FoSSaCS'02)**

An order-2 equation is unsafe if the RHS has a subterm $P$ s.t.

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

**Consequence:** An order-$i$ subterm of a safe term can only have free variables of order at least $i$.

**Example (unsafe eqn)**: $F : (o \to o) \to o \to o \to o$, $f : o^2 \to o$, $x, y : o$.

$$F\,\varphi\,x\,y \quad = \quad f\,(F\,\underline{(F\,\varphi\,y)}\,y\,(\varphi\,x))\,\underline{a}$$

## What is the safety constraint on recursion schemes?

Safety is a set of constraints on where variables may occur in a term.

**Definition (Damm TCS 82, KNU FoSSaCS'02)**

An order-2 equation is unsafe if the RHS has a subterm $P$ s.t.

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

**Consequence:** An order-$i$ subterm of a safe term can only have free variables of order at least $i$.

**Example (unsafe eqn)**: $F : (o \to o) \to o \to o \to o,\ f : o^2 \to o,\ x, y : o.$

$$F\,\varphi\,x\,y \quad = \quad f\,(F\,\underline{(F\,\varphi\,y)}\,y\,(\varphi\,x))\,\underline{a}$$

## What is the point of safety?

Safety does have an important algorithmic advantage!

**Theorem (Blum + O. TLCA 07, LMCS 09)**

*Substitution (hence $\beta$-red.) in safe $\lambda$-calculus can be safely implemented without renaming bound variables! Hence no fresh names needed.*

**Theorem**

1. *(Schwichtenberg 76) The numeric functions representable by simply-typed $\lambda$-terms are multivariate polynomials with conditional.*
2. *(Blum + O. LMCS 09) The numeric functions representable by simply-typed safe $\lambda$-terms are the multivariate polynomials.*

(See (Blum + O. LMCS 09) for a study on the safe lambda calculus.)

# What is the point of safety?

Safety does have an important algorithmic advantage!

**Theorem (Blum + O. TLCA 07, LMCS 09)**

*Substitution (hence $\beta$-red.) in safe $\lambda$-calculus can be safely implemented without renaming bound variables! Hence no fresh names needed.*

**Theorem**

1. *(Schwichtenberg 76) The numeric functions representable by simply-typed $\lambda$-terms are multivariate polynomials with conditional.*
2. *(Blum + O. LMCS 09) The numeric functions representable by simply-typed safe $\lambda$-terms are the multivariate polynomials.*

(See (Blum + O. LMCS 09) for a study on the safe lambda calculus.)

# Infinite structures generated by recursion schemes: key questions

1. **MSO decidability**: Is safety a genuine constraint for decidability?
   I.e. do trees generated by (arbitrary) recursion schemes have decidable MSO theories?
2. **Machine characterisation**: Find a hierarchy of automata that characterise the expressive power of recursion schemes.
   I.e. how should the power of higher-order pushdown automata be augmented to achieve equi-expressivity with (arbitrary) recursion schemes?
3. **Expressivity**: Is safety a genuine constraint for expressivity?
   I.e. are there inherently unsafe word languages / trees / graphs?

## Infinite structures generated by recursion schemes: key questions

1. **MSO decidability**: Is safety a genuine constraint for decidability?
   I.e. do trees generated by (arbitrary) recursion schemes have decidable MSO theories?
2. **Machine characterisation**: Find a hierarchy of automata that characterise the expressive power of recursion schemes.
   I.e. how should the power of higher-order pushdown automata be augmented to achieve equi-expressivity with (arbitrary) recursion schemes?
3. **Expressivity**: Is safety a genuine constraint for expressivity?
   I.e. are there inherently unsafe word languages / trees / graphs?

1. **MSO decidability**: Is safety a genuine constraint for decidability?
   I.e. do trees generated by (arbitrary) recursion schemes have decidable MSO theories?
2. **Machine characterisation**: Find a hierarchy of automata that characterise the expressive power of recursion schemes.
   I.e. how should the power of higher-order pushdown automata be augmented to achieve equi-expressivity with (arbitrary) recursion schemes?
3. **Expressivity**: Is safety a genuine constraint for expressivity?
   I.e. are there inherently unsafe word languages / trees / graphs?

4 **Graph families**:

1. **Definition**: What is a good definition of "graphs generated by recursion schemes"?
2. **Model-checking properties**: What are the decidable (modal-) logical theories of the graph families?

## Q1. Do trees in $\text{RecSchTree}_n\Sigma$ have decidable MSO theories?

**Recent Progress:**

**Theorem (Aehlig, de Miranda + O. TLCA 2005)**

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

**Theorem (Knapik, Niwinski, Urczyczn + Walukiewicz, ICALP 2005)**

*Modal mu-calculus model checking problem for homogenously-typed order-2 schemes (whether safe or not) is 2-EXPTIME complete.*

What about higher orders?

Yes: MSO decidability extends to all orders (O. LICS06).

# Q1. Do trees in $\mathrm{RecSchTree}_n\Sigma$ have decidable MSO theories?

**Recent Progress:**

**Theorem (Aehlig, de Miranda + O. TLCA 2005)**

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

**Theorem (Knapik, Niwinski, Urczyczn + Walukiewicz, ICALP 2005)**

*Modal mu-calculus model checking problem for homogenously-typed order-2 schemes (whether safe or not) is 2-EXPTIME complete.*

What about higher orders?

Yes: MSO decidability extends to all orders (O. LICS06).

# Q1. Do trees in $\mathsf{RecSchTree}_n\Sigma$ have decidable MSO theories?

**Recent Progress:**

**Theorem (Aehlig, de Miranda + O. TLCA 2005)**

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

**Theorem (Knapik, Niwinski, Urczyczn + Walukiewicz, ICALP 2005)**

*Modal mu-calculus model checking problem for homogenously-typed order-2 schemes (whether safe or not) is 2-EXPTIME complete.*

What about higher orders?

Yes: MSO decidability extends to all orders (O. LICS06).

# Q1. Do trees in $\mathbf{RecSchTree}_n\Sigma$ have decidable MSO theories? Yes

**Theorem (O. LICS 2006)**

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

[This is the largest generically-defined MSO-decidable class of ranked trees (*cf.* Montanari + Puppis, LICS 2007).]

**Two key ingredients**:

$$\begin{array}{ll} & [\![ G ]\!] \text{ satisifes modal mu-calculus formula } \varphi \\ \iff & \{ \text{ Emerson + Jutla 1991} \} \\ & \text{APT } \mathcal{B}_\varphi \text{ has accepting run-tree over generated tree } [\![ G ]\!] \\ \iff & \{ \textbf{ I. Transference Principle: Traversal-Path Correspondence} \} \\ & \text{APT } \mathcal{B}_\varphi \text{ has accepting traversal-tree over computation tree } \lambda(G) \\ \iff & \{ \textbf{ II. Simulation of traversals by paths } \} \\ & \text{APT } \mathcal{C}_\varphi \text{ has an accepting run-tree over computation tree } \lambda(G) \end{array}$$

which is decidable.

## Q1. Do trees in $\mathbf{RecSchTree}_n\Sigma$ have decidable MSO theories? Yes

**Theorem (O. LICS 2006)**

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

[This is the largest generically-defined MSO-decidable class of ranked trees (*cf.* Montanari + Puppis, LICS 2007).]

**Two key ingredients**:

$[\![ G ]\!]$ satisifes modal mu-calculus formula $\varphi$

$\iff$ { Emerson + Jutla 1991}

APT $\mathcal{B}_\varphi$ has accepting run-tree over generated tree $[\![ G ]\!]$

$\iff$ { **I. Transference Principle: Traversal-Path Correspondence**}

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

$\iff$ { **II. Simulation of traversals by paths** }

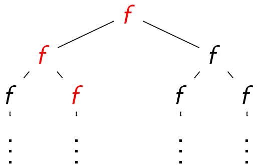
APT $\mathcal{C}_\varphi$ has an accepting run-tree over computation tree $\lambda(G)$

which is decidable.

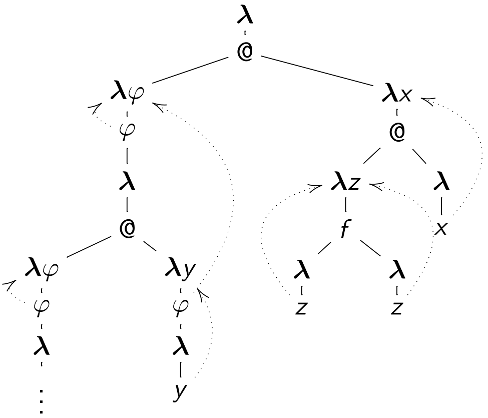
## Transference principle, based on a theory of traversals

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

$[\![\, G \,]\!]$




$\lambda(G)$



**Idea**: $\beta$-reduction is global (i.e. substitution changes the term being evaluated); game semantics gives an equivalent but local view.

A traversal (over the computation tree $\lambda(G)$) is a trace of the local computation that produces a path (over $[\![ G ]\!]$).

### Theorem (Path-traversal correspondence)

*Let $G$ be an order-$n$ recursion scheme.*

(i) *There is a 1-1 correspondence between maximal paths $p$ in ($\Sigma$-labelled) generated 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.

**Explanation (for game semanticists)**:

- Term-tree $[\![ G ]\!]$ is (a representation of) the game semantics of $G$.
- Paths in $[\![ G ]\!]$ correspond to plays in the strategy-denotation.
- Traversals $t_p$ over computation tree $\lambda(G)$ are just (representations of) the uncoverings of the plays (= path) $p$ in the game semantics of $G$.

**Idea**: $\beta$-reduction is global (i.e. substitution changes the term being evaluated); game semantics gives an equivalent but local view.
A traversal (over the computation tree $\lambda(G)$) is a trace of the local computation that produces a path (over $[\![ G ]\!]$).

### Theorem (Path-traversal correspondence)

*Let $G$ be an order-$n$ recursion scheme.*

(i) *There is a 1-1 correspondence between maximal paths $p$ in ($\Sigma$-labelled) generated 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.

**Explanation (for game semanticists)**:

- Term-tree $[\![ G ]\!]$ is (a representation of) the game semantics of $G$.
- Paths in $[\![ G ]\!]$ correspond to plays in the strategy-denotation.
- Traversals $t_p$ over computation tree $\lambda(G)$ are just (representations of) the uncoverings of the plays (= path) $p$ in the game semantics of $G$.

**Idea**: $\beta$-reduction is global (i.e. substitution changes the term being evaluated); game semantics gives an equivalent but local view.
A traversal (over the computation tree $\lambda(G)$) is a trace of the local computation that produces a path (over $[\![ G ]\!]$).

### Theorem (Path-traversal correspondence)

*Let $G$ be an order-$n$ recursion scheme.*

(i) *There is a 1-1 correspondence between maximal paths $p$ in ($\Sigma$-labelled) generated 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.

**Explanation (for game semanticists)**:

- Term-tree $[\![ G ]\!]$ is (a representation of) the game semantics of $G$.
- Paths in $[\![ G ]\!]$ correspond to plays in the strategy-denotation.
- Traversals $t_p$ over computation tree $\lambda(G)$ are just (representations of) the uncoverings of the plays (= path) $p$ in the game semantics of $G$.

# Q2: Machine characterization: collapsible pushdown automata

Order-2 collapsible pushdown automata [HOMS, LiCS 08a] are essentially the same as 2PDA with links [AdMO 05], and panic automata [KNUW 05].

**Idea**: Each stack symbol in 2-stack "remembers" the stack content at the point it was first created (i.e. $push_1$ed onto the stack), by way of a pointer to some 1-stack underneath it (if there is one such).

**Two new stack operations:** $a \in \Gamma$ (stack alphabet)

- $push_1\ a$: pushes $a$ onto the top of the top 1-stack, together with a pointer to the 1-stack immediately below the top 1-stack.
- *collapse* (= panic) collapses the 2-stack down to the prefix pointed to by the $top_1$-element of the 2-stack.

Note that the pointer-relation is preserved by $push_2$.

# Q2: Machine characterization: collapsible pushdown automata

Order-2 collapsible pushdown automata [HOMS, LiCS 08a] are essentially the same as 2PDA with links [AdMO 05], and panic automata [KNUW 05].

**Idea**: Each stack symbol in 2-stack "remembers" the stack content at the point it was first created (i.e. $push_1$ed onto the stack), by way of a pointer to some 1-stack underneath it (if there is one such).

**Two new stack operations:** $a \in \Gamma$ (stack alphabet)

- $push_1\ a$: pushes $a$ onto the top of the top 1-stack, together with a pointer to the 1-stack immediately below the top 1-stack.
- *collapse* (= panic) collapses the 2-stack down to the prefix pointed to by the $top_1$-element of the 2-stack.

Note that the pointer-relation is preserved by $push_2$.

## Q2: Machine characterization: collapsible pushdown automata

Order-2 collapsible pushdown automata [HOMS, LiCS 08a] are essentially the same as 2PDA with links [AdMO 05], and panic automata [KNUW 05].

**Idea**: Each stack symbol in 2-stack "remembers" the stack content at the point it was first created (i.e. $push_1$ed onto the stack), by way of a pointer to some 1-stack underneath it (if there is one such).

**Two new stack operations:** $a \in \Gamma$ (stack alphabet)

- $push_1\ a$: pushes $a$ onto the top of the top 1-stack, together with a pointer to the 1-stack immediately below the top 1-stack.
- *collapse* (= panic) collapses the 2-stack down to the prefix pointed to by the $top_1$-element of the 2-stack.

Note that the pointer-relation is preserved by $push_2$.

**Order-2 Collapsible Pushdown Automata** (for word languages):

$$\langle \Sigma, Q, q_0, \Gamma, \Delta \subseteq (\Sigma \cup \{ \epsilon \}) \times Q \times \Gamma \times Q \times Op_2, F \rangle$$

where $Op_2 := \{ \mathrm{push}_2, \mathrm{pop}_2, \mathrm{pop}_1, \mathrm{collapse} \} \cup \{ \mathrm{push}_1 a \mid a \in \Gamma \}$.

**Example**. Starting from the empty 2-stack $[[\,]]$, what is the top-of-stack symbol after the following sequence of actions?

1. $\mathrm{push}_2$
2. $\mathrm{push}_1 a$
3. $\mathrm{push}_2$
4. $\mathrm{push}_1 b$
5. $\mathrm{push}_2$
6. $\mathrm{pop}_1$
7. collapse

**Order-2 Collapsible Pushdown Automata** (for word languages):

$$\langle \Sigma, Q, q_0, \Gamma, \Delta \subseteq (\Sigma \cup \{ \epsilon \}) \times Q \times \Gamma \times Q \times Op_2, F \rangle$$

where $Op_2 \ := \ \{ \mathrm{push}_2, \mathrm{pop}_2, \mathrm{pop}_1, \mathrm{collapse} \} \ \cup \ \{ \mathrm{push}_1 a \mid a \in \Gamma \}$.

**Example**. Starting from the empty 2-stack $[[\,]]$, what is the top-of-stack symbol after the following sequence of actions?

1. $\mathrm{push}_2$
2. $\mathrm{push}_1 a$
3. $\mathrm{push}_2$
4. $\mathrm{push}_1 b$
5. $\mathrm{push}_2$
6. $\mathrm{pop}_1$
7. collapse

# Collapsible pushdown automata: extending to all finite orders

In **order-$n$ CPDA**, there are $n - 1$ versions of $push_1$, namely, $push_1^j\ a$, with $1 \leq j \leq n - 1$:

> *$push_1^j\ a$: pushes $a$ onto the top of the top 1-stack, together with a pointer to the $j$-stack immediately below the top $j$-stack.*

**Definition** (Aehlig, de Miranda + O. FoSSaCS 05) A ***U*-word** has 3 segments:

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

- Segment $A$ is a prefix of a well-bracketed word that ends in **(**, and the opening **(** is not matched in the entire word.
- Segment $B$ is a well-bracketed word.
- Segment $C$ has length equal to the number of **(** in segment $A$.

**Examples**

1. $(\,(\,)\,(\,(\,)\,(\,(\,)\,)\,*\ *\ *\ \in\ U$
2. For each $n \geq 0$, we have $(\,(^n\,)^n\,(\ *^n\ **\ \in\ U$. (Hence by "*uvwxy* Lemma", $U$ is not context-free.)

**Definition** (Aehlig, de Miranda + O. FoSSaCS 05) A ***U*-word** has 3 segments:

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

- Segment $A$ is a prefix of a well-bracketed word that ends in **(**, and the opening **(** is not matched in the entire word.
- Segment $B$ is a well-bracketed word.
- Segment $C$ has length equal to the number of **(** in segment $A$.

### Examples

1. $(\,(\,)\,(\,(\,)\,(\,(\,)\,)\,*\,*\,*\,\in\,U$
2. For each $n \geq 0$, we have $(\,(^n\,)^n\,(\,*^n\,**\,\in\,U$. (Hence by "*uvwxy* Lemma", $U$ is not context-free.)

Definition (Aehlig, de Miranda + O. FoSSaCS 05) A ***U*-word** has 3 segments:

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

- Segment $A$ is a prefix of a well-bracketed word that ends in **(**, and the opening **(** is not matched in the entire word.
- Segment $B$ is a well-bracketed word.
- Segment $C$ has length equal to the number of **(** in segment $A$.

### Examples

1. $(\,(\,)\,(\,(\,)\,(\,(\,)\,)\,*\ *\ *\ \in\ U$
2. For each $n \geq 0$, we have $(\,(^n\,)^n\,(\ *^n\ *\,*\ \in\ U$. (Hence by "*uvwxy* Lemma", $U$ is not context-free.)

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

(Ignoring control states for simplicity)

| Upon reading | Do |
|---|---|
| ( | $push_2$ ; $push_1 a$ |
| ) | $pop_1$ |
| first $*$ | $collapse$ |
| subsequent $*$ | $pop_2$ |

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

What does the depth of the top 1-stack mean?

### Observation

1. $U$ is recognisable by a deterministic order-2 CPDA.
2. Equivalently (thanks to [AdMO 05]) $U$ is recognisable by a non-deterministic order-2 PDA — because of the need to guess the transition from segment A to segment B.

### Conjecture

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

(Related to the Safety Conjecture - more anon.)

**Exercise** (moderately hard). Give an order-2 recursion scheme that generates $U$.

## Q2: Recursion schemes are equi-expressive with CPDA

**Theorem (Equi-Expressivity, Hague, Murawski, O. + Serre LICS'08)**

*For each $n \geq 0$, order-$n$ recursion schemes and order-$n$ collapsible PDA are equi-expressive for $\Sigma$-labelled trees. I.e.* $\mathbf{RecSchTree}_n\Sigma = \mathbf{CPDATree}_n\Sigma$

(Proof uses theory of traversals, based on game semantics.)

**Consequences**:

1. **Kleene's Problem:** What computing power is required to compute order-$n$ lambda-definable functionals?

   The Theorem gives a syntax-independent characterisation of pure simply-typed lambda-calculus with recursion.

2. A **new proof** of the MSO decidability of trees generated by order-$n$ recursion schemes.

**Open Problem**. Find a new proof of "RS $\rightarrow$ CPDA" without using game semantics.

# Q2: Recursion schemes are equi-expressive with CPDA

**Theorem (Equi-Expressivity, Hague, Murawski, O. + Serre LICS'08)**

*For each $n \geq 0$, order-$n$ recursion schemes and order-$n$ collapsible PDA are equi-expressive for $\Sigma$-labelled trees. I.e.* $\mathbf{RecSchTree}_n\Sigma = \mathbf{CPDATree}_n\Sigma$

(Proof uses theory of traversals, based on game semantics.)

**Consequences**:

1. **Kleene's Problem:** What computing power is required to compute order-$n$ lambda-definable functionals?

   The Theorem gives a syntax-independent characterisation of pure simply-typed lambda-calculus with recursion.

2. A **new proof** of the MSO decidability of trees generated by order-$n$ recursion schemes.

**Open Problem**. Find a new proof of "RS $\longrightarrow$ CPDA" without using game semantics.

## Q2: Recursion schemes are equi-expressive with CPDA

**Theorem (Equi-Expressivity, Hague, Murawski, O. + Serre LICS'08)**

*For each $n \geq 0$, order-$n$ recursion schemes and order-$n$ collapsible PDA are equi-expressive for $\Sigma$-labelled trees. I.e.* $\mathbf{RecSchTree}_n\Sigma = \mathbf{CPDATree}_n\Sigma$

(Proof uses theory of traversals, based on game semantics.)

**Consequences**:

1. **Kleene's Problem:** What computing power is required to compute order-$n$ lambda-definable functionals?

   The Theorem gives a syntax-independent characterisation of pure simply-typed lambda-calculus with recursion.

2. A **new proof** of the MSO decidability of trees generated by order-$n$ recursion schemes.

**Open Problem**. Find a new proof of "RS $\longrightarrow$ CPDA" without using game semantics.

# Q2: Recursion schemes are equi-expressive with CPDA

**Theorem (Equi-Expressivity, Hague, Murawski, O. + Serre LICS'08)**

*For each $n \geq 0$, order-$n$ recursion schemes and order-$n$ collapsible PDA are equi-expressive for $\Sigma$-labelled trees. I.e.* $\mathbf{RecSchTree}_n\Sigma = \mathbf{CPDATree}_n\Sigma$

(Proof uses theory of traversals, based on game semantics.)

**Consequences**:

1. **Kleene's Problem:** What computing power is required to compute order-$n$ lambda-definable functionals?

   The Theorem gives a syntax-independent characterisation of pure simply-typed lambda-calculus with recursion.

2. A **new proof** of the MSO decidability of trees generated by order-$n$ recursion schemes.

**Open Problem**. Find a new proof of "RS $\to$ CPDA" without using game semantics.

# Q3: Does safety constrain expressivity?

**Case 1: Word languages.** Conjecture: Yes; but note

**Theorem (Aehlig, de Miranda + O., FoSSaCS 2005)**

*At order 2, there are no inherently unsafe word languages. I.e. for every unsafe order-2 recursion scheme, there is a safe (non-deterministic) order-2 recursion scheme that generates the same language.*

**Case 2: Trees.** Conjecture: Yes.

**The Safety Conjecture**

For each $n \geq 2$, there is a tree generated by an unsafe order-$n$ recursion scheme but not by any safe order-$n$ recursion scheme.

## Q3: Does safety constrain expressivity?

**Case 1: Word languages.** Conjecture: Yes; but note

**Theorem (Aehlig, de Miranda + O., FoSSaCS 2005)**

*At order 2, there are no inherently unsafe word languages. I.e. for every unsafe order-2 recursion scheme, there is a safe (non-deterministic) order-2 recursion scheme that generates the same language.*

**Case 2: Trees.** Conjecture: Yes.

**The Safety Conjecture**

For each $n \geq 2$, there is a tree generated by an unsafe order-$n$ recursion scheme but not by any safe order-$n$ recursion scheme.

**Case 3: Graphs.** Yes.

**Theorem (Hague, Murawski, O. + Serre LICS 2008a)**

*There is an order-2 CPDA graph that is not generated by any order-2 PDA.*

(See example graph later.)

## A survey of graph families with model-checking properties

| | Decidable? | | | |
|---|---|---|---|---|
| | MSO | $\mu$ | FO(R) | FO |
| Caucal Graph Hierarchy | yes | yes | yes | yes |
| Ground-term tree rewriting (Löding 02) | no | no | yes | yes |
| Automatic graphs (Hodgson 76, KN 94) | no | no | no | yes |
| Rational graphs | no | no | no | no |

| | Decidable? | | | |
|---|---|---|---|---|
| | MSO | $\mu$ | FO(R) | FO |
| Caucal's Graph Hierarchy | yes | yes | yes | yes |
| **C** | no | yes | ? | ? |
| Ground-term tree rewriting (Löding 02) | no | no | yes | yes |
| Automatic graphs (Hodgson 76, KN 94) | no | no | no | yes |
| Rational graphs | no | no | no | no |

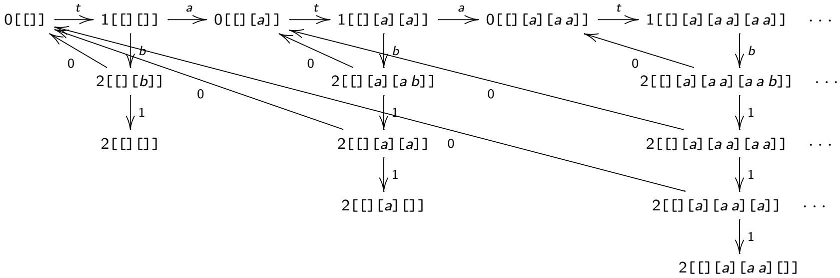
**Question**

Is there a generically-defined family **C** of graphs that have decidable modal-mu calculus theories but undecidable MSO theories?

Yes. See construction on next slide (HMOS, LiCS 08a).

# Configuration graphs of (order-2) CPDA is not MSO-decidable

An order-2 CPDA graph: MSO-interpretable into the infinite half-grid.




To our knowledge CPDA graphs are the first "natural" generically-defined graph families that have decidable modal mu-calculus theories but undecidable MSO theories.

## Q4: Model-checking properties of CPDA graphs

**Theorem (Hague, Murawski, O and Serre, LiCS 2008a)**

1. *For each $n \geq 0$, the decidability of modal mu-calculus model-checking problem for configuration graphs of order-n CPDA is n-EXPTIME complete.*
2. *Equivalently solvability of parity games over order-n CPDA graphs is n-EXPTIME complete.*