Higher-Order Model Checking: Part 1 or 2

Luke Ong

University of Oxford
http://www.cs.ox.ac.uk/people/luke.ong/personal/
http://mjolnir.cs.ox.ac.uk

LATA, Bilbao, 3 April 2013

Model checking and computer-aided verification

Beginning in the 80s, the computer-aided algorithmic verification (especially model checking) of finite-state systems has been a great success story in computer science.

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

What is (software) model checking?

A Verification Problem: Given a system *Sys* (e.g. an OS), and a correctness property *Spec* (e.g. deadlock freedom), does *Sys* satisfy *Spec*?

The model checking approach:

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

Huge strides made in verification of first-order imperative programs (e.g. C).

Verification of higher-order programs

Two standard methods

1. Static analysis, often type-based: sound, scalable but often imprecise
   E.g. kCFA, type and effect systems (region-based memory management), refinement types, resource usage (sized types), etc.
2. Theorem proving and dependent types: accurate, typically requires human intervention; does not scale well
   E.g. Coq, Agda, etc.

A relatively recent approach:

Higher-Order Model Checking (HOMC) is the model checking of infinite structures (such as trees) that are defined by recursion schemes and related families of "higher-order" generators, with a view to formally analysing higher-order computation.

Aims of the lectures

1. We introduce a systematic approach to the algorithmics of infinite structures generated by families of higher-order generators.
2. We present an approach to verifying higher-order functional programs by reduction to the model checking of recursion schemes.

Outline: four parts

1. Relating families of generators of infinite structures
2. Recursion schemes (and collapsible pushdown automata) and their algorithmics
3. Reducing model checking to type inference
4. Application: verification of higher-order functional programs

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 λ-calculus with recursion and uninterpreted order-1 constant symbols.

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

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

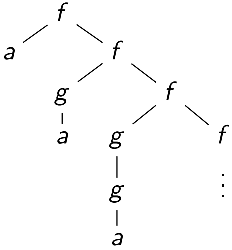
Unfolding from the start symbol S:

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

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

Representing the term-tree $[\![G]\!]$ as a Σ-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 Σ-labelled tree, formally, a map $T \longrightarrow \Sigma$, where T is a prefix-closed subset of $\{ 1, \cdots, m \}^*$, and m is the maximal arity of symbols in Σ.

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

Think of $[\![G]\!]$ as the Böhm tree of G.

An Order-3 Example: Fibonacci Numbers

`fib` generates an infinite spine, with each member (encoded in unary) of the Fibonacci sequence appearing in turn as a left branch from the spine.

Terminals: $b : 2,\ u : 1,\ z : 0$
Non-terminals: Write Ch as a shorthand for $(o \to o) \to o \to o$

$$
\begin{array}{rcl}
S & : & o \\
Zero & : & Ch \\
Unit & : & Ch \\
Show & : & Ch \to Ch \to o \\
Add & : & Ch \to Ch \to (o \to o) \to o \to o
\end{array}
$$

$$
\mathtt{fib} \left\{
\begin{array}{rcl}
S & \to & Show\ Zero\ Unit \\
Zero\ \varphi\ x & \to & x \\
Unit\ \varphi\ x & \to & \varphi\ x \\
Show\ n_1\ n_2 & \to & b\ (n_1\ s\ z)\ (Show\ n_2\ (Add\ n_1\ n_2)) \\
Add\ n_1\ n_2\ \varphi\ x & \to & n_1\ \varphi\ (n_2\ \varphi\ x)
\end{array}
\right.
$$

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 Σ. We can use a (non-deterministic) recursion scheme to generate finite-word languages, with ranked alphabet

$$\overline{\Sigma} := \{\, a : 1 \mid a \in \Sigma \,\} \cup \{\, e : 0 \,\}.$$

1. A word language is regular iff it is generated by an order-0 recursion scheme.
2. A word language is context-free iff it is generated by an order-1 recursion scheme.

What class of word languages do order-2 recursion schemes define?

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.

$$
\begin{array}{rcll}
push_2 & : & [s_1 \cdots s_{i-1} \underbrace{[\gamma_1 \cdots \gamma_n]}_{s_i}] & \mapsto & [s_1 \cdots s_{i-1}\, s_i\, s_i] \\
pop_2 & : & [s_1 \cdots s_{i-1} [\gamma_1 \cdots \gamma_n]] & \mapsto & [s_1 \cdots s_{i-1}] \\
push_1\, \gamma & : & [s_1 \cdots s_{i-1} [\gamma_1 \cdots \gamma_n]] & \mapsto & [s_1 \cdots s_{i-1} [\gamma_1 \cdots \gamma_n\, \gamma]] \\
pop_1 & : & [s_1 \cdots s_{i-1} [\gamma_1 \cdots \gamma_n\, \gamma_{n+1}]] & \mapsto & [s_1 \cdots s_{i-1} [\gamma_1 \cdots \gamma_n]]
\end{array}
$$

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

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

L is not context free—thanks to the "*uvwxy* Lemma".

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

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

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

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

Some Properties of the Maslov Hierarchy of Word Languages

(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.

A recent breakthrough

Theorem (Inaba + Maneth FSTTCS08)

All languages of the Maslov Hierarchy are context-sensitive.

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? (More anon.)

Two Families of Generators of Infinite Structures

HOPDA can be used as recognising/generating device for

1. finite-word languages (Maslov 74) and ω-word languages
2. possibly-infinite ranked trees (KNU01) and, more generally, languages of such trees
3. possibly infinite graphs (Muller+Schupp 86, Courcelle 95, Cachat 03), *qua* configuration graphs of these pushdown systems

HORS (higher-order recursion schemes) can also be used to generate word languages, potentially-infinite trees (and languages there of) and graphs.

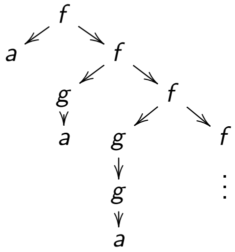
The two families are closely related.

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 $\varphi_1 \vee \varphi_2$.



Monadic second-order (MSO) logic can describe properties such as $\Phi_1 \vee \Phi_2$.

Is the "MSO Model-Checking Problem for Recursion Schemes" decidable?

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

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

- Rabin 1969: Infinite binary trees and regular trees. "Mother of all decidability results in algorithmic verification."
- Muller and Schupp 1985: Configuration graphs of PDA.
- Caucal 1996 Prefix-recognisable graphs (ϵ-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.

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 rule).

$$F : (o \to o) \to o \to o \to o, \; f : o^2 \to o, \; x, y : o.$$

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

The subterm $F \; \varphi \; y$ has order 1, but the free variable y has order 0.

What is the point of safety?

Safety does have an important algorithmic advantage!

Theorem (KNU 02, Blum + O. TLCA 07, LMCS 09)

Substitution (hence β-red.) in safe λ-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 λ-terms are multivariate polynomials with conditional.*
2. *(Blum + O. LMCS 09) The numeric functions representable by simply-typed safe λ-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?
4. **Graph families**:
   1. **Definition**: What is a good definition of "graphs generated by recursion schemes"?
   2. **Model-checking properties**: What are the decidable theories of the graph families?

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.*

Proof Idea. Two key ingredients:

Generated tree $[\![G]\!]$ satisfies MSO formula φ

$\iff$ { 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 because $\lambda(G)$ is regular.

Four different proofs of the MSO decidability result

1. Game semantics and traversals (O. LICS06)
   - variable profiles. E.g. a profile of $(o \to o) \to o$ is $(\{\,(\{\,q\,\}, q), (\{\,q, q'\,\}, q')\,\}, q)$
2. Collapsible pushdown automata (Hague, Murawski, O. & Serre LICS08)
   - equi-expressivity theorem + rank aware automata
3. Type-theoretic characterisation of APT (Kobayashi & O. LICS09)
   - intersection types. E.g. $(q \to q) \wedge (q \wedge q' \to q') \to q$
4. Krivine machines (Salvati & Walukiewicz ICALP11)
   - residuals

A common pattern

1. Decision problem equivalent to solving an infinite parity game.
2. Simulate the infinite parity game by a finite parity game.
3. Key ingredient of the game: variable profiles / automaton control-states / intersection types / residuals.

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 $push_2$.

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

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. $(\,(\,)\,(\,(\,)\,(\,(\,)\,)\,*\;*\;*$ is a U-word
2. For each $n \geq 0$, we have $(\,(^n\,)^n\,(\,*^{n+2}$ is a U-word.
   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?

Is order-n CPDA strictly more expressive than order-n PDA?

Equivalently, does the *collapse* operation add any expressive power?

Lemma (AdMO FoSSaCS05): Urzyczyn's language U is quite telling!

1. U is *not* recognised by any 1PDA.
2. U is recognised by a non-deterministic 2PDA.
3. U is recognised by a deterministic 2CPDA.

Question

1. *Is U recognisable by a deterministic 2PDA?*
2. *More generally, is U recognisable by a deterministic nPDA for any n?*

(If true, there is an associated tree that is generated by an order-2 recursion scheme, but not by any order-2 safe recursion scheme.)

Theorem (Equi-expressivity [Hague, Murawski, O. & Serre LICS08])

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

Proof idea

- From recursion scheme to CPDA: Use game semantics.
  Code traversals as n-stacks.
  Invariant: The top 1-stack is the P-view of the encoded traversal.
  For a direct proof (without game semantics) see [Carayol & Serre LICS12].
- From CPDA to recursion scheme:
  Code configuration c as Σ-term M_c, so that $c \to c'$ implies M_c rewrites to $M_{c'}$.

CPDA are a machine characterization of simply-typed lambda calculus with recursions.

Q3: Is safety a genuine constraint on expressivity?

Question (Safety, KNW FoSSaCS02)

Are there inherently unsafe word languages / trees / graphs?

Word languages? Yes

Theorem (Parys STACS11, LICS12)

There is a language (similar to U) recognised by a deterministic 2CPDA but not by any deterministic nPDA for all $n \geq 0$.

Proof uses a powerful pumping lemma for HOPDA.

(Another pumping lemma for nCPDA is used to prove a hierarchy theorem for collapsible graphs and trees [Kartzow & Parys, MFCS12])

Trees? Yes

Theorem (Parys STACS11, LICS12)

There is a tree generated by an order-2 recursion scheme but not by any safe HORS.

Graphs? Yes.

Theorem (Hague, Murawski, O and Serre LICS08)

1. *Solvability of parity games over order-n CPDA graphs is n-EXPTIME complete.*
2. *There is an 2CPDA configuration graph with an undecidable MSO theory.*

Corollary

There is a 2CPDA whose configuration graph (semi-infinite grid) is not that of any nPDA, for any n.

A safety question for non-determinacy

Question (Safety non-determinacy)

Is there a word language recognised by a order-n CPDA which is not recognisable by any non-deterministic higher-order PDA?

For order 2, the answer is no.

Theorem (Aehlig, de Miranda and O. FoSSaCS 2005)

For every order-2 recursion scheme, there is a safe non-deterministic order-2 recursion scheme that generates the same word language.