Higher-Order Model Checking

I: Relating Families of Generators of Infinite Structures

Luke Ong

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

Estonia Winter School in Computer Science, 3-8 Mar 2013

Model checking and computer-aided verification

Beginning in the 80s, computer-aided algorithmic 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?

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 1st-order imperative programs**.

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

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

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

Higher-Order Functional Programming

Examples: OCaml, F#, Haskell, Lisp/Scheme, JavaScript, and Erlang; even C++.

Why higher-order functional languages?

1. Functional programs are succinct, less error-prone, easy to write and maintain, good for prototyping.
2. λ-expressions and closures now basic in Javascript, Perl5, Python, C# and C++0x, which are standard in web programming, hardware and embedded systems design. [TIOBE index]
3. FL support domain-specific languages and organise data parallelism well; increasingly prevalent in scientific applications and financial modelling
4. Absence of mutable variables and use of monadic structuring principles make FL attractive for concurrent programming, thanks to growth of multi-core, GPGPU processing and cloud computing.

Verifying functional programs

Two standard approaches

1. Program analysis, often type-based
   - sound, scalable but often imprecise
   E.g. control flow analysis (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.

Model checking higher-order functional programs

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.

   Many 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.

Aims of the lecture course

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.

References for the course

http://www.cs.ox.ac.uk/people/luke.ong/personal/EWSCS13

A reminder: simple types

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

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. $e : A$ 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 ranked alphabet $\Sigma = \{ f : 2, g : 1, a : 0 \}$.

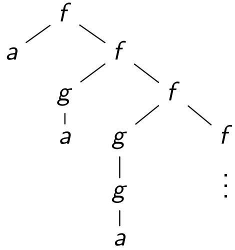
$$G \; : \; \begin{cases} S & \to \; F\,a \\ F\,x & \to \; f\,x\,(F\,(g\,x)) \end{cases}$$

Unfolding from the start symbol S:

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

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

$[\![G]\!] = f\, a\, (f\, (g\, a)\, (f\, (g\, (g\, a))(\cdots)))$ 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.

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

Fix a set of typed variables (written as φ, x, y etc).

- $\mathcal{N}$: Typed non-terminals of order at most n (written as upper-case letters), including a distinguished start symbol $S : o$.
- Σ: Ranked alphabet of terminals: $f \in \Sigma$ has arity $\mathrm{ar}(f) \geq 0$
- $\mathcal{R}$: An equation for each non-terminal $F : A_1 \to \cdots \to A_m \to o$ of shape

$$F\,\varphi_1 \cdots \varphi_m \quad \to \quad e$$

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

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

The tree generated by a recursion scheme: value tree

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 Σ (i.e. $\perp \leq a$ for all $a \in \Sigma$) to trees by:

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

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

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

Infinite full binary trees

1. $\Sigma \to \{ a : 2 \}$

$$S \to a\ S\ S$$

2. $\{ a : 2, b : 2 \}$

$$\begin{cases} S & \to & b\,(b\ A\ A)\,(a\ A\ B) \\ A & \to & a\ A\ A \\ B & \to & b\ B\ B \end{cases}$$

Is it true that "every path has only finitely many b"?
No. There is a path $b\ a\ b^{\omega}$.

3. $\{ a : 2, b : 2 \}$

$$\begin{cases} S & \to & b\,(b\ A\ A)\,(a\ A\ A) \\ A & \to & a\ A\ A \\ B & \to & b\ B\ B \end{cases}$$

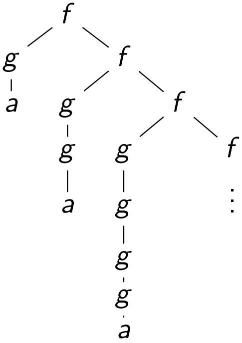
Is it true that "every path has only finitely many b"?
Yes. Every path matches $b\,(b + a)\ a^{\omega}$.

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:



An Order-3 Example: Fibonacci Numbers

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

Non-terminals: Write Ch as a shorthand for $(o \to o) \to o \to o$

$$
\begin{array}{lcl}
S & : & o \\
Z & : & Ch \\
U & : & Ch \\
F & : & Ch \to Ch \to o \\
P & : & Ch \to Ch \to (o \to o) \to o \to o
\end{array}
$$

$$
\texttt{fib} \left\{
\begin{array}{rcl}
S & \to & F\ Z\ U \\
Z\ \varphi\ x & \to & x \\
U\ \varphi\ x & \to & \varphi\ x \\
F\ n_1\ n_2 & \to & c\ (n_1\ s\ z)\ (F\ n_2\ (P\ n_1\ n_2)) \\
P\ n_1\ n_2\ \varphi\ x & \to & n_1\ \varphi\ (n_2\ \varphi\ x)
\end{array}
\right.
$$

Recapitulation

- Introduction
- HORS (Higher-Order Recursion Schemes) as generators of Σ-labelled trees

Synopsis of today's lecture: 5 March 13

- HORS as generators of word languages
- Higher-order Pushdown Automata (HOPDA) as generators of word languages (and trees). Maslov Hierarchy.
- Relating the two families of generators. Safe Lambda Calculus.
- Monadic second-order (MSO) logic of Σ-labelled trees
- Model checking trees against MSO formulas

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

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

Recall: in word-generating recursion schemes, letters $a, b : 1$ (i.e. of arity 1) and $e : 0$ is the end-of-word.

1. The regular language $(a\,(a+b)^*\;b)^*$ is generated by the order-0 recursion scheme:

$$\left\{ \begin{array}{lcl} S & \to & e \quad | \quad a\,F \\ F & \to & a\,F \quad | \quad b\,F \quad | \quad b\,S \end{array} \right.$$

2. The context-free language $\{\, a^n\, b^n \mid n \geq 0 \,\}$ is generated by the order-1 recursion scheme:

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

Regular languages are exactly order-0

Lemma

A word language is regular iff it is generated by an order-0 (non-deterministic) recursion scheme.

Take a NFA $(Q, \Delta \subseteq Q \times (\Sigma \cup \{ \epsilon \}) \times Q, q_I, F \subseteq Q)$. Define an order-0 RS $(\Sigma, \{ F_q \mid q \in Q \}, F_{q_I}, \mathcal{R})$ where $\mathcal{R}$ has following rules:

- For each $(q, a, q') \in \Delta$, introduce a rewrite rule:

$$F_q \to a \; F_{q'}$$

- For each $(q, \epsilon, q') \in \Delta$, introduce a rewrite rule:

$$F_q \to F_{q'}$$

- For each $q_f \in F$, introduce

$$F_{q_f} \to e$$

Exercise

1. Prove the following:

Lemma

A word language is context-free (equivalently, recognisable by a non-deterministic pushdown automata) iff it is generated by an order-1 (word-language) recursion scheme.

2. Find an order-2 (word-language) recursion scheme that generates

$$\{ a^i b^i c^i \mid i \geq 0 \}.$$

A PDA is a finite-state machine equipped with a pushdown (LIFO) stack.
Transition

$$(q, a, \gamma, q', \theta) \in Q \times \Sigma \times \Gamma \times Q \times Op_1$$

where $Op_1 = \{ push\,\gamma \mid \gamma \in \Gamma \} \cup \{ pop \}$.

$$\begin{array}{rcl} push_1\,\gamma & : & [\gamma_1 \cdots \gamma_n] \quad \mapsto \quad [\gamma_1 \cdots \gamma_n\,\gamma] \\ pop_1 & : & [\gamma_1 \cdots \gamma_n\,\gamma_{n+1}] \quad \mapsto \quad [\gamma_1 \cdots \gamma_n] \end{array}$$

(Top of stack is the righthand end.)

Example. $\{ a^i\ b^i \mid i \geq 0 \}$ is recognisable by a PDA.
Idea: use the depth of stack to remember number of a already read.

$$q_0\ [] \xrightarrow{a} q_0\ [\gamma] \xrightarrow{a} q_0\ [\gamma\,\gamma] \xrightarrow{b} q_0\ [\gamma] \xrightarrow{b} q_0\ []$$

Higher-order pushdown automata (HOPDA) [Maslov 74]

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.

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

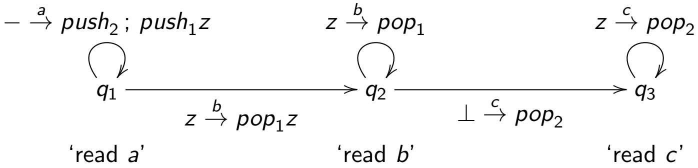
Idea extends to all finite orders: an order-n PDA has an order-n stack, and has $push_i$ and pop_i for each $1 \leq i \leq n$.

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.

$$q_1 \, [[\,]] \xrightarrow{a} q_1 \, [[\,][z]] \xrightarrow{a} q_1 \, [[\,][z][zz]] \xrightarrow{b} q_2 \, [[\,][z][z]] \xrightarrow{b} q_2 \, [[\,][z][\,]] \xrightarrow{c} q_3 \, [[\,][z]] \xrightarrow{c} q_3 \, [[\,]]$$



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.)

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 result.

Theorem (Inaba + Maneth FSTTCS08)

All languages of the Maslov Hierarchy are context-sensitive.

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 generally languages of such trees
3. possibly infinite graphs (Muller+Schupp 86, Courcelle 95, Cachat 03)

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

Why study the two families of generators?

They are relevant to 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 characterise 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 functional programs.