Bisimulation and Logic
Lecture 6

Colin Stirling

Laboratory for Foundations of Computer Science (LFCS)
School of Informatics
Edinburgh University

Summer School on Model Checking
Ziyu Hotel, Beijing
Oct 11–16 2010

Methods for verifying finite and infinite state systems

- Notable Success in Computer Science
- model checking + equivalence checking

Methods for verifying finite and infinite state systems

- Notable Success in Computer Science
- model checking + equivalence checking
- System = finite/infinite state transition graph

Methods for verifying finite and infinite state systems

- Notable Success in Computer Science
- model checking + equivalence checking
- System = finite/infinite state transition graph
- Model checking: does state $s \models \Phi$?
- apply automata/game theoretic techniques to solve it: mostly computing monadic fixed points, reachability sets by traversing graph (possibly repeatedly)

Methods for verifying finite and infinite state systems

- Notable Success in Computer Science
- model checking + equivalence checking
- System = finite/infinite state transition graph
- Model checking: does state $s \models \Phi$?
- apply automata/game theoretic techniques to solve it: mostly computing monadic fixed points, reachability sets by traversing graph (possibly repeatedly)
- Equivalence checking: is state s equivalent to t ?
- Mostly computing dyadic fixed points e.g. bisimulations to solve it. May need algebraic/combinatorial properties of reachability sets/generators of graph

Active research goal: transfer these techniques to

finite/infinite state systems with binding

1. Deciding observational equivalence for fragments of idealized Algol (w.r.t. finite value sets)
   [Ghica, McCusker 2000; Ong 2002, ...]

Active research goal: transfer these techniques to

finite/infinite state systems with binding

1. Deciding observational equivalence for fragments of idealized Algol (w.r.t. finite value sets)
   [Ghica, McCusker 2000; Ong 2002, ...]
2. Model checking higher-order trees
   [Knapik, Niwinski, Urzyczyn 2002; Caucal 2002; Ong 2006; Hague Murawski, Ong, Serre 2008; Kobayashi, Ong 2009; Broadbent, Ong 2010]

Active research goal: transfer these techniques to

finite/infinite state systems with binding

1. Deciding observational equivalence for fragments of idealized Algol (w.r.t. finite value sets)
   [Ghica, McCusker 2000; Ong 2002, ...]
2. Model checking higher-order trees
   [Knapik, Niwinski, Urzyczyn 2002; Caucal 2002; Ong 2006; Hague Murawski, Ong, Serre 2008; Kobayashi, Ong 2009; Broadbent, Ong 2010]
3. $\vdots \quad \vdots \quad \vdots$
4. Application of tree automata to higher-order matching
   [Comon + Jurski 1997, Stirling 2005-9]

Higher Order Schemes

Base type 0: finite/infinite trees with nodes labelled by elements of $\{f_1, \ldots, f_k\}$. Each f_i has an arity ≥ 0.
Scheme is a finite family

$$F_i\, x_1^i \ldots x_{n_i}^i \stackrel{\text{def}}{=} t_i \quad 1 \leq i \leq m$$

each F_i is typed and distinct

Higher Order Schemes

Base type 0: finite/infinite trees with nodes labelled by elements of $\{f_1, \ldots, f_k\}$. Each f_i has an arity ≥ 0.
Scheme is a finite family

$$\boxed{F_i\, x_1^i \ldots x_{n_i}^i \stackrel{\text{def}}{=} t_i \quad 1 \leq i \leq m}$$

each F_i is typed and distinct
each $t_i : 0$ built from the typed variables, $x_1^i, \ldots, x_{n_i}^i$, the f_js and the F_i's using application.
Also start (closed) expression $\boxed{S : 0}$ (Avoiding λ-terms)

Higher Order Schemes

Base type 0: finite/infinite trees with nodes labelled by elements of $\{f_1, \ldots, f_k\}$. Each f_i has an arity ≥ 0.
Scheme is a finite family

$$\boxed{F_i\, x_1^i \ldots x_{n_i}^i \stackrel{\text{def}}{=} t_i \quad 1 \leq i \leq m}$$

each F_i is typed and distinct
each $t_i : 0$ built from the typed variables, $x_1^i, \ldots, x_{n_i}^i$, the f_js and the F_i's using application.
Also start (closed) expression $\boxed{S : 0}$ (Avoiding λ-terms)
Interpretation of a scheme: tree generated by S

Example: first-order

$Fx_1x_2 \stackrel{\mathrm{def}}{=} f(Fx_1h(x_2))x_2$ with start Fbb

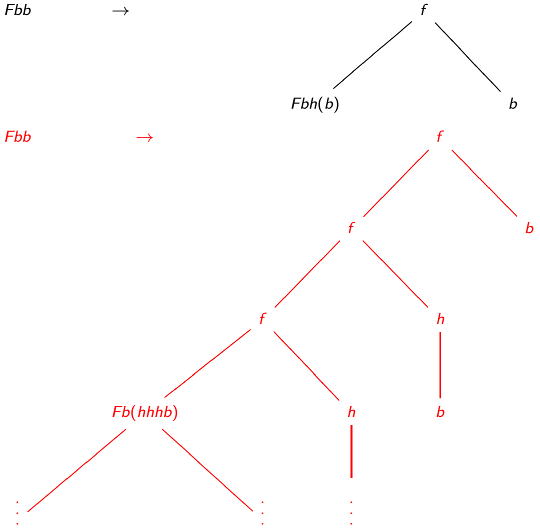
Example: first-order

$Fx_1x_2 \stackrel{\text{def}}{=} f(Fx_1h(x_2))x_2$ with start Fbb

$$Fbb \quad \rightarrow \quad \begin{tikzcd} & f \arrow[dl, no head] \arrow[dr, no head] & \\ Fbh(b) & & b \end{tikzcd}$$

Example: first-order

$Fx_1x_2 \stackrel{\mathrm{def}}{=} f(Fx_1h(x_2))x_2$ with start Fbb

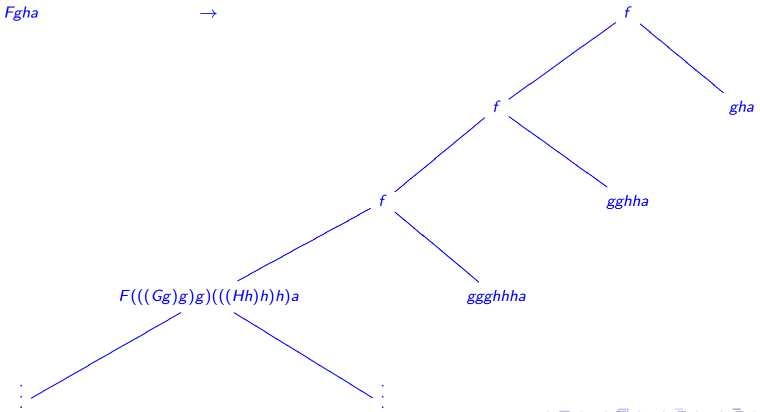


Example: second-order

$$\begin{array}{lcl} F x_1 x_2 x_3 & \stackrel{\text{def}}{=} & f\,(F(G x_1)(H x_2) x_3)\, x_1 (x_2 x_3) \\ G y_1 y_2 & \stackrel{\text{def}}{=} & g(y_1(y_2)) \\ H z_1 z_2 & \stackrel{\text{def}}{=} & h(z_1(z_2)) \\ F g h a & & \text{Start} \end{array}$$

Example: second-order

$$
\begin{array}{lcl}
F x_1 x_2 x_3 & \stackrel{\mathrm{def}}{=} & f\,(F(Gx_1)(Hx_2)x_3)\,x_1(x_2x_3) \\
G y_1 y_2 & \stackrel{\mathrm{def}}{=} & g(y_1(y_2)) \\
H z_1 z_2 & \stackrel{\mathrm{def}}{=} & h(z_1(z_2)) \\
Fgha & & \text{Start}
\end{array}
$$



Model checking problem

Given S, does its tree have a decidable monadic 2nd-order theory?

Model checking problem

Given S, does its tree have a decidable monadic 2nd-order theory?
Solved +vely for a subset of schemes (safe schemes)
[Knapik + Niwinski + Urzyczyn 2002]
Proof uses geometry of interaction on infinite λ-terms +
higher-order pushdown automata

Model checking problem

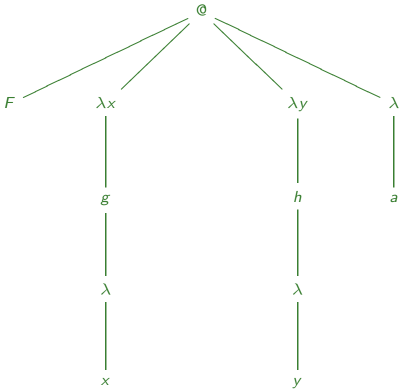
Given S, does its tree have a decidable monadic 2nd-order theory?
Solved +vely for a subset of schemes (safe schemes)
[Knapik + Niwinski + Urzyczyn 2002]
Proof uses geometry of interaction on infinite λ-terms + higher-order pushdown automata
Extended to all schemes
[Ong 2006, Hague + Murawski + Ong +Serre, 2008]
Proof uses game semantics on infinite lambda terms. Later paper also uses extended higher-order automata

$$
\begin{array}{lcl}
F x_1 x_2 x_3 & \stackrel{\text{def}}{=} & f\,(F(G x_1)(H x_2) x_3)\, x_1 (x_2 x_3) \\
G y_1 y_2 & \stackrel{\text{def}}{=} & g(y_1(y_2)) \\
H z_1 z_2 & \stackrel{\text{def}}{=} & h(z_1(z_2)) \\
Fgha & & \text{Start}
\end{array}
$$

Follow Ong's transformation into normal form

$$\begin{array}{lcl}
F x_1 x_2 x_3 & \stackrel{\text{def}}{=} & f\,(F(G x_1)(H x_2) x_3)\,x_1(x_2 x_3) \\
G y_1 y_2 & \stackrel{\text{def}}{=} & g(y_1(y_2)) \\
H z_1 z_2 & \stackrel{\text{def}}{=} & h(z_1(z_2)) \\
Fgha & & \text{Start}
\end{array}$$

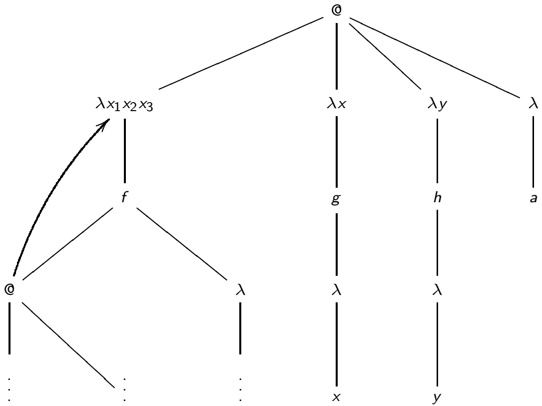
Follow Ong's transformation into normal form



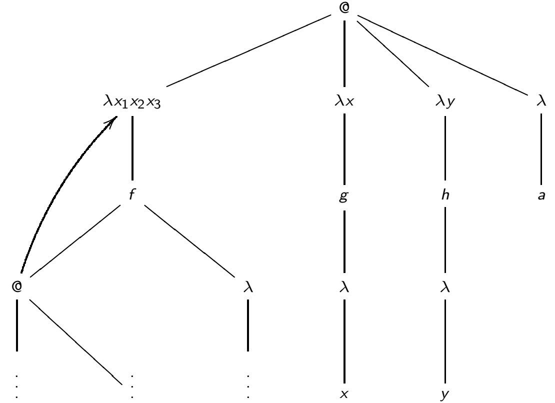
$$F = \lambda x_1 x_2 x_3 . f(@ \ldots)\lambda . x_1(\lambda . x_2 . (\lambda . x_3))$$

Infinite λ-term (can be folded into a finite tree with backedges)

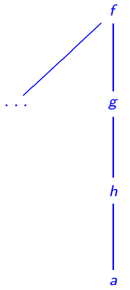
infinite λ-tree



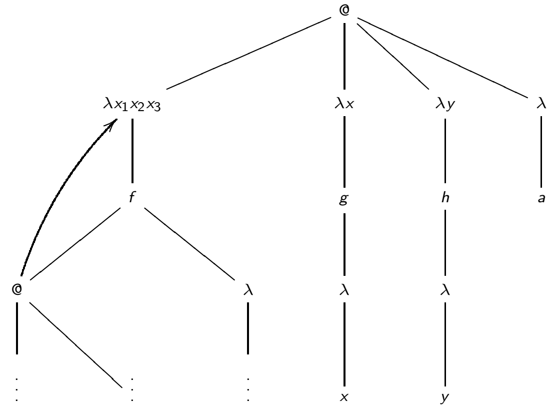
infinite λ-tree



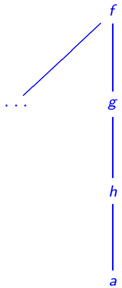
using game semantics, extract the following infinite tree



infinite λ-tree



using game semantics, extract the following infinite tree



transform parity tree automaton on above tree into one on the infinite λ-tree

Equivalence problem (from 1970s)

Given two schemes S_1, S_2 do they generate same tree ?

$S_1 \sim S_2$ bisimulation equivalence

Equivalence problem (from 1970s)

Given two schemes S_1, S_2 do they generate same tree ?

$S_1 \sim S_2$ bisimulation equivalence

For order 1, $S_1 \sim S_2 \equiv$ DPDA equivalence problem

[Courcelle 1978]

Do two configurations of a deterministic pushdown automaton generate the same language ?

Equivalence problem (from 1970s)

Given two schemes S_1, S_2 do they generate same tree ?

$S_1 \sim S_2$ bisimulation equivalence

For order 1, $S_1 \sim S_2 \equiv$ DPDA equivalence problem

[Courcelle 1978]

Do two configurations of a deterministic pushdown automaton generate the same language ?

Solved positively [Sénizergues 2001]

(Simpler proofs [Stirling 2001, 2002])

Equivalence problem (from 1970s)

Given two schemes S_1, S_2 do they generate same tree ?

$S_1 \sim S_2$ bisimulation equivalence

For order 1, $S_1 \sim S_2 \equiv$ DPDA equivalence problem

[Courcelle 1978]

Do two configurations of a deterministic pushdown automaton generate the same language ?

Solved positively [Sénizergues 2001]

(Simpler proofs [Stirling 2001, 2002])

Order > 1 ?

Equivalence problem (from 1970s)

Given two schemes S_1, S_2 do they generate same tree ?

$S_1 \sim S_2$ bisimulation equivalence

For order 1, $S_1 \sim S_2 \equiv$ DPDA equivalence problem

[Courcelle 1978]

Do two configurations of a deterministic pushdown automaton generate the same language ?

Solved positively [Sénizergues 2001]

(Simpler proofs [Stirling 2001, 2002])

Order > 1 ?

Open $+$ Hard

(Decidable for a small subset of 2nd-order case [Stirling 2006])

Equivalence problem (from 1970s)

Given two schemes S_1, S_2 do they generate same tree ?

$S_1 \sim S_2$ bisimulation equivalence

For order 1, $S_1 \sim S_2 \equiv$ DPDA equivalence problem

[Courcelle 1978]

Do two configurations of a deterministic pushdown automaton generate the same language ?

Solved positively [Sénizergues 2001]

(Simpler proofs [Stirling 2001, 2002])

Order > 1 ?

Open $+$ Hard

(Decidable for a small subset of 2nd-order case [Stirling 2006])

INFINITELY MANY OPEN PROBLEMS