Dependency Tree Automata

Colin Stirling
cps@inf.ed.ac.uk

LFCS
School of Informatics
University of Edinburgh

FOSSACS 2009, York, March 23rd

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]

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]
3. $\vdots \quad \vdots \quad \vdots$
4. Application of tree automata to higher-order matching
   [Comon + Jurski 1997, Stirling 2007, work described here]

Simply typed λ-calculus: fundamental exemplar of binding

- types $A ::= \mathbf{0} \mid A \to A$

Simply typed λ-calculus: fundamental exemplar of binding

- types $A ::= \mathbf{0} \mid A \rightarrow A$
- $A \rightarrow B$ type of functions from A to B
- $A_1 \rightarrow (\ldots (A_n \rightarrow \mathbf{0}) \ldots)$ written $(A_1, \ldots, A_n, \mathbf{0})$

Simply typed λ-calculus: fundamental exemplar of binding

- types $A ::= \mathbf{0} \mid A \to A$
- $A \to B$ type of functions from A to B
- $A_1 \to (\ldots(A_n \to \mathbf{0})\ldots)$ written $(A_1, \ldots, A_n, \mathbf{0})$
- order of $\mathbf{0}$ is 1;
- order of $(A_1, \ldots, A_n, \mathbf{0})$ is $1 + \max\{\text{order of } A_i\text{s}\}$

Terms T of simply typed λ-calculus

Variables and constants each have a unique type (Church style)

1. if x (f) has type A then $x : A \in T$ ($f : A \in T$)

Terms T of simply typed λ-calculus

Variables and constants each have a unique type (Church style)

1. if x (f) has type A then $x : A \in T$ ($f : A \in T$)
2. if $t : B \in T$ and $x : A \in T$ then $\lambda x.t : A \to B \in T$

Terms T of simply typed λ-calculus

Variables and constants each have a unique type (Church style)

1. if x (f) has type A then $x : A \in T$ ($f : A \in T$)
2. if $t : B \in T$ and $x : A \in T$ then $\lambda x.t : A \to B \in T$
3. if $t : A \to B \in T$ and $u : A \in T$ then $(tu) : B \in T$

Terms T of simply typed λ-calculus

Variables and constants each have a unique type (Church style)

1. if x (f) has type A then $x : A \in T$ $(f : A \in T)$
2. if $t : B \in T$ and $x : A \in T$ then $\lambda x.t : A \to B \in T$
3. if $t : A \to B \in T$ and $u : A \in T$ then $(tu) : B \in T$

- order of $t : A$ = order A

Terms T of simply typed λ-calculus

Variables and constants each have a unique type (Church style)

1. if x (f) has type A then $x : A \in T$ $(f : A \in T)$
2. if $t : B \in T$ and $x : A \in T$ then $\lambda x.t : A \rightarrow B \in T$
3. if $t : A \rightarrow B \in T$ and $u : A \in T$ then $(tu) : B \in T$

- order of $t : A$ $=$ order A
- closed $t : A$ no free variables

Terms T of simply typed λ-calculus

Variables and constants each have a unique type (Church style)

1. if x (f) has type A then $x : A \in T$ $(f : A \in T)$
2. if $t : B \in T$ and $x : A \in T$ then $\lambda x.t : A \to B \in T$
3. if $t : A \to B \in T$ and $u : A \in T$ then $(tu) : B \in T$

- order of $t : A$ $=$ order A
- closed $t : A$ no free variables
- $t, t' : A$ are α-equivalent renamings of each other

Reduction and all

$$
\begin{array}{lll}
(\beta) & (\lambda x.t)v \rightarrow_\beta t\{v/x\} & \{\cdot/\cdot\}\ \text{Substitution} \\
(\eta) & \lambda x.(tx) \rightarrow_\eta t & x \text{ not free in } t
\end{array}
$$

Reduction and all

$$\begin{array}{lll} (\beta) & (\lambda x.t)v \to_\beta t\{v/x\} & \{\cdot/\cdot\} \text{ Substitution} \\ (\eta) & \lambda x.(tx) \to_\eta t & x \text{ not free in } t \end{array}$$

- t in normal form = no β or η reductions in t
- Fact Each term has a unique $\beta\eta$-normal form

Reduction and all

$$\begin{array}{lll} (\beta) & (\lambda x.t)v \rightarrow_\beta t\{v/x\} & \{\cdot/\cdot\} \text{ Substitution} \\ (\eta) & \lambda x.(tx) \rightarrow_\eta t & x \text{ not free in } t \end{array}$$

- t in normal form $=$ no β or η reductions in t
- Fact Each term has a unique $\beta\eta$-normal form
- $=_{\beta\eta}$ means same normal form

Reduction and all

$$\begin{array}{lll}(\beta) & (\lambda x.t)v \rightarrow_\beta t\{v/x\} & \{\cdot/\cdot\}\ \text{Substitution} \\ (\eta) & \lambda x.(tx) \rightarrow_\eta t & x \text{ not free in } t\end{array}$$

- t in normal form $=$ no β or η reductions in t
- Fact Each term has a unique $\beta\eta$-normal form
- $=_{\beta\eta}$ means same normal form
- η-long form for $t : A$ defined by cases on A

Reduction and all

$$
\begin{array}{lll}
(\beta) & (\lambda x.t)v \to_\beta t\{v/x\} & \{\cdot/\cdot\}\ \text{Substitution} \\
(\eta) & \lambda x.(tx) \to_\eta t & x \text{ not free in } t
\end{array}
$$

- t in normal form = no β or η reductions in t
- Fact Each term has a unique $\beta\eta$-normal form
- $=_{\beta\eta}$ means same normal form
- η-long form for $t : A$ defined by cases on A
    1. $\mathbf{0}$ t is $u : \mathbf{0}$ or $u\, t_1 \ldots t_k$ where $u : (B_1, \ldots, B_k, \mathbf{0})$ constant/variable and each $t_i : B_i$ is in η-long form

Reduction and all

$$\begin{array}{lll}(\beta) & (\lambda x.t)v \rightarrow_\beta t\{v/x\} & \{\cdot/\cdot\} \text{ Substitution} \\ (\eta) & \lambda x.(tx) \rightarrow_\eta t & x \text{ not free in } t\end{array}$$

- t in normal form = no β or η reductions in t
- Fact Each term has a unique $\beta\eta$-normal form
- $=_{\beta\eta}$ means same normal form
- η-long form for $t : A$ defined by cases on A
  1. $\mathbf{0}$ t is $u : \mathbf{0}$ or $u\, t_1 \ldots t_k$ where $u : (B_1, \ldots, B_k, \mathbf{0})$ constant/variable and each $t_i : B_i$ is in η-long form
  2. $(A_1, \ldots, A_n, \mathbf{0})$ t is $\lambda y_1 \ldots y_n.t'$ where each $y_i : A_i$ and $t' : \mathbf{0}$ is in η-long form ($\lambda y_1 \ldots y_n.t'$ abbreviates $\lambda y_1. \ldots \lambda y_n.t'$)

Reduction and all

$$
\begin{array}{lll}
(\beta) & (\lambda x.t)v \to_\beta t\{v/x\} & \{\cdot/\cdot\} \text{ Substitution} \\
(\eta) & \lambda x.(tx) \to_\eta t & x \text{ not free in } t
\end{array}
$$

- t in normal form = no β or η reductions in t
- Fact Each term has a unique $\beta\eta$-normal form
- $=_{\beta\eta}$ means same normal form
- η-long form for $t : A$ defined by cases on A
  1. $\mathbf{0}$ t is $u : \mathbf{0}$ or $u\, t_1 \dots t_k$ where $u : (B_1, \dots, B_k, \mathbf{0})$ constant/variable and each $t_i : B_i$ is in η-long form
  2. $(A_1, \dots, A_n, \mathbf{0})$ t is $\lambda y_1 \dots y_n.t'$ where each $y_i : A_i$ and $t' : \mathbf{0}$ is in η-long form ($\lambda y_1 \dots y_n.t'$ abbreviates $\lambda y_1. \dots \lambda y_n.t'$)
- Fact Each term in normal form has a unique η-long form
- When η-long forms $=_\beta$ is $=_{\beta\eta}$

Monadic decision questions

- Given a type A and property ϕ, is the following set non-empty?

$$\{t : A \mid t \text{ in normal form and } t \models \phi\}$$

Monadic decision questions

- Given a type A and property ϕ, is the following set non-empty?

$$\{t : A \mid t \text{ in normal form and } t \models \phi\}$$

- Is this set recognisable by an automaton?

Monadic decision questions

- Given a type A and property ϕ, is the following set non-empty?

$$\{t : A \mid t \text{ in normal form and } t \models \phi\}$$

- Is this set recognisable by an automaton?
- Standard representation of terms is as trees: therefore,

Monadic decision questions

- Given a type A and property ϕ, is the following set non-empty?

$$\{t : A \mid t \text{ in normal form and } t \models \phi\}$$

- Is this set recognisable by an automaton?
- Standard representation of terms is as trees: therefore,
- Is this set recognisable by a tree automaton?

Monadic decision questions

- Given a type A and property ϕ, is the following set non-empty?

$$\{t : A \mid t \text{ in normal form and } t \models \phi\}$$

- Is this set recognisable by an automaton?
- Standard representation of terms is as trees: therefore,
- Is this set recognisable by a tree automaton?
- BUT: PROBLEM

Monster type $M = ((((\mathbf{0}, \mathbf{0}), \mathbf{0}), \mathbf{0}), \mathbf{0}, \mathbf{0})$, order 5

- Let $f : (((\mathbf{0}, \mathbf{0}), \mathbf{0}), \mathbf{0})$, $g_i : (\mathbf{0}, \mathbf{0})$, $x : \mathbf{0}$

Monster type $M = ((((\mathbf{0}, \mathbf{0}), \mathbf{0}), \mathbf{0}), \mathbf{0}, \mathbf{0})$, order 5

- Let $f : (((\mathbf{0}, \mathbf{0}), \mathbf{0}), \mathbf{0})$, $g_i : (\mathbf{0}, \mathbf{0})$, $x : \mathbf{0}$
- for all $n \geq 0$ the folowing terms belong to M
- $\lambda f, x.f(\lambda g_1.f(\lambda g_2 \dots f(\lambda g_n.g_n(g_{n-1}(\dots g_1(x)) \dots)) \dots))$

Monster type $M = ((((\mathbf{0}, \mathbf{0}), \mathbf{0}), \mathbf{0}), \mathbf{0}, \mathbf{0})$, order 5

- Let $f : (((\mathbf{0}, \mathbf{0}), \mathbf{0}), \mathbf{0})$, $g_i : (\mathbf{0}, \mathbf{0})$, $x : \mathbf{0}$
- for all $n \geq 0$ the folowing terms belong to M
- $\lambda f, x.f(\lambda g_1.f(\lambda g_2 \dots f(\lambda g_n.g_n(g_{n-1}(\dots g_1(x)) \dots)) \dots))$
- To write down this subset of terms up to α-equivalence requires an alphabet of unbounded size

Monster type $M = ((((\mathbf{0},\mathbf{0}),\mathbf{0}),\mathbf{0}),\mathbf{0},\mathbf{0})$, order 5

- Let $f : (((\mathbf{0},\mathbf{0}),\mathbf{0}),\mathbf{0})$, $g_i : (\mathbf{0},\mathbf{0})$, $x : \mathbf{0}$
- for all $n \geq 0$ the folowing terms belong to M
- $\lambda f, x.f(\lambda g_1.f(\lambda g_2 \ldots f(\lambda g_n.g_n(g_{n-1}(\ldots g_1(x))\ldots))\ldots))$
- To write down this subset of terms up to α-equivalence requires an alphabet of unbounded size
- There are automata with infinite alphabets but none applicable e.g.[Segoufin 2006 survey: "Automata and logics for words and trees over an infinite alphabet"]

Binding Trees

- $\Sigma = \Sigma_1 \cup \Sigma_2 \cup \Sigma_3$ is alphabet, each $s \in \Sigma$ has arity $\mathrm{ar}(s) \geq 0$. Σ_1 are binders (with arity 1); Σ_2 are (bound) variables; Σ_3 other symbols (such as constants)

Binding Trees

- $\Sigma = \Sigma_1 \cup \Sigma_2 \cup \Sigma_3$ is alphabet, each $s \in \Sigma$ has arity $\mathsf{ar}(s) \geq 0$.
  Σ_1 are binders (with arity 1); Σ_2 are (bound) variables; Σ_3 other symbols (such as constants)
- A binding Σ-tree is a finite tree

Binding Trees

- $\Sigma = \Sigma_1 \cup \Sigma_2 \cup \Sigma_3$ is alphabet, each $s \in \Sigma$ has arity $\mathrm{ar}(s) \geq 0$. Σ_1 are binders (with arity 1); Σ_2 are (bound) variables; Σ_3 other symbols (such as constants)
- A binding Σ-tree is a finite tree
  1. each node labelled with an element of Σ
  2. extra binary relation $\downarrow$ (representing binding)

Binding Trees

- $\Sigma = \Sigma_1 \cup \Sigma_2 \cup \Sigma_3$ is alphabet, each $s \in \Sigma$ has arity $\mathrm{ar}(s) \geq 0$. Σ_1 are binders (with arity 1); Σ_2 are (bound) variables; Σ_3 other symbols (such as constants)
- A binding Σ-tree is a finite tree
  1. each node labelled with an element of Σ
  2. extra binary relation $\downarrow$ (representing binding)
  3. if node n labelled with s, $\mathrm{ar}(s) = k$ then n has k successors

Binding Trees

- $\Sigma = \Sigma_1 \cup \Sigma_2 \cup \Sigma_3$ is alphabet, each $s \in \Sigma$ has arity $\mathrm{ar}(s) \geq 0$. Σ_1 are binders (with arity 1); Σ_2 are (bound) variables; Σ_3 other symbols (such as constants)
- A binding Σ-tree is a finite tree
  1. each node labelled with an element of Σ
  2. extra binary relation $\downarrow$ (representing binding)
  3. if node n labelled with s, $\mathrm{ar}(s) = k$ then n has k successors
  4. if node n labelled $s \in \Sigma_2$ then a unique b above n labelled with $s' \in \Sigma_1$ and $b \downarrow n$

Binding Trees

- $\Sigma = \Sigma_1 \cup \Sigma_2 \cup \Sigma_3$ is alphabet, each $s \in \Sigma$ has arity $\mathrm{ar}(s) \geq 0$. Σ_1 are binders (with arity 1); Σ_2 are (bound) variables; Σ_3 other symbols (such as constants)
- A binding Σ-tree is a finite tree
  1. each node labelled with an element of Σ
  2. extra binary relation $\downarrow$ (representing binding)
  3. if node n labelled with s, $\mathrm{ar}(s) = k$ then n has k successors
  4. if node n labelled $s \in \Sigma_2$ then a unique b above n labelled with $s' \in \Sigma_1$ and $b \downarrow n$
- Compare nested words/trees [Alur + Chaudhuri + Madhusudan, 2006]

Terms of monster type M as binding trees

- Let $f : (((\mathbf{0}, \mathbf{0}), \mathbf{0}), \mathbf{0})$, $g : (\mathbf{0}, \mathbf{0})$, $x : \mathbf{0}$

Terms of monster type M as binding trees

- Let $f : (((\mathbf{0}, \mathbf{0}), \mathbf{0}), \mathbf{0})$, $g : (\mathbf{0}, \mathbf{0})$, $x : \mathbf{0}$
- Now we have:

$$\lambda f, x \qquad f(\lambda g \qquad f(\lambda g \qquad \ldots f(\lambda g(\qquad g(\ldots \qquad g(\qquad x \ldots$$

Terms of monster type M as binding trees

- Let $f : (((\mathbf{0},\mathbf{0}),\mathbf{0}),\mathbf{0})$, $g : (\mathbf{0},\mathbf{0})$, $x : \mathbf{0}$
- Now we have:

$$\lambda f, x \quad f(\lambda g \quad f(\lambda g \quad \ldots f(\lambda g(\quad g(\ldots \quad g(\quad x\ldots$$

- Finite alphabet $+$ edge relation $\downarrow$

Dependency tree automata for binding trees

- $\mathsf{A} = (Q, \Sigma, q_0, \Delta)$ where Q finite set of states, Σ finite alphabet, $q_0 \in Q$ initial state and Δ finite set of transitions

Dependency tree automata for binding trees

- A $= (Q, \Sigma, q_0, \Delta)$ where Q finite set of states, Σ finite alphabet, $q_0 \in Q$ initial state and Δ finite set of transitions
  1. $qs \Rightarrow (q_1, \ldots, q_k)$ where $s \in \Sigma_2 \cup \Sigma_3$, $\mathrm{ar}(s) = k$, $q, q_1, \ldots, q_k \in Q$

Dependency tree automata for binding trees

- A $= (Q, \Sigma, q_0, \Delta)$ where Q finite set of states, Σ finite alphabet, $q_0 \in Q$ initial state and Δ finite set of transitions
  1. $qs \Rightarrow (q_1, \ldots, q_k)$ where $s \in \Sigma_2 \cup \Sigma_3$, $\mathrm{ar}(s) = k$, $q, q_1, \ldots, q_k \in Q$
  2. $qs \Rightarrow q's'$ where $s \in \Sigma_1$, $s' \in \Sigma_3$ and $q, q' \in Q$

Dependency tree automata for binding trees

- A $= (Q, \Sigma, q_0, \Delta)$ where Q finite set of states, Σ finite alphabet, $q_0 \in Q$ initial state and Δ finite set of transitions
  1. $qs \Rightarrow (q_1, \ldots, q_k)$ where $s \in \Sigma_2 \cup \Sigma_3$, $\mathrm{ar}(s) = k$, $q, q_1, \ldots, q_k \in Q$
  2. $qs \Rightarrow q's'$ where $s \in \Sigma_1$, $s' \in \Sigma_3$ and $q, q' \in Q$
  3. $(q', q)s \Rightarrow q_1 x$ where $s \in \Sigma_1$, $x \in \Sigma_2$ and $q', q, q_1 \in Q$

Dependency tree automata for binding trees

- A $= (Q, \Sigma, q_0, \Delta)$ where Q finite set of states, Σ finite alphabet, $q_0 \in Q$ initial state and Δ finite set of transitions
  1. $qs \Rightarrow (q_1, \ldots, q_k)$ where $s \in \Sigma_2 \cup \Sigma_3$, $\mathrm{ar}(s) = k$, $q, q_1, \ldots, q_k \in Q$
  2. $qs \Rightarrow q's'$ where $s \in \Sigma_1$, $s' \in \Sigma_3$ and $q, q' \in Q$
  3. $(q', q)s \Rightarrow q_1 x$ where $s \in \Sigma_1$, $x \in \Sigma_2$ and $q', q, q_1 \in Q$
- Alternating dependency tree; change transitions 1. to

Dependency tree automata for binding trees

- A $= (Q, \Sigma, q_0, \Delta)$ where Q finite set of states, Σ finite alphabet, $q_0 \in Q$ initial state and Δ finite set of transitions
  1. $qs \Rightarrow (q_1, \ldots, q_k)$ where $s \in \Sigma_2 \cup \Sigma_3$, $\mathrm{ar}(s) = k$, $q, q_1, \ldots, q_k \in Q$
  2. $qs \Rightarrow q's'$ where $s \in \Sigma_1$, $s' \in \Sigma_3$ and $q, q' \in Q$
  3. $(q', q)s \Rightarrow q_1 x$ where $s \in \Sigma_1$, $x \in \Sigma_2$ and $q', q, q_1 \in Q$
- Alternating dependency tree; change transitions 1. to
  1. $qs \Rightarrow (Q_1, \ldots, Q_k)$ where $s \in \Sigma_2 \cup \Sigma_3$, $\mathrm{ar}(s) = k$, $q \in Q$ and $Q_1, \ldots, Q_k \subseteq Q$.

Run of automaton A on $t \in \mathrm{T}_\Sigma$

- An additional labelling of t with states Q defined top-down.
  The root is labelled with initial state q_0

Run of automaton A on $t \in T_\Sigma$

- An additional labelling of t with states Q defined top-down.
  The root is labelled with initial state q_0
- Consider a node n labelled $s \in \Sigma$ and with state q

Run of automaton A on $t \in T_\Sigma$

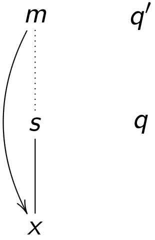
- An additional labelling of t with states Q defined top-down. The root is labelled with initial state q_0
- Consider a node n labelled $s \in \Sigma$ and with state q
- If $s \in \Sigma_2 \cup \Sigma_3$ and $qs \Rightarrow (q_1, \ldots, q_k) \in \Delta$ then each successor ni is labelled q_i

Run of automaton A on $t \in T_\Sigma$

- An additional labelling of t with states Q defined top-down. The root is labelled with initial state q_0
- Consider a node n labelled $s \in \Sigma$ and with state q
- If $s \in \Sigma_2 \cup \Sigma_3$ and $qs \Rightarrow (q_1, \ldots, q_k) \in \Delta$ then each successor ni is labelled q_i
- If $s \in \Sigma_1$, $n1$ is labelled $s' \in \Sigma_3$ and $qs \Rightarrow q's' \in \Delta$ then $n1$ is labelled q'.

Run of automaton A on $t \in T_\Sigma$

- An additional labelling of t with states Q defined top-down. The root is labelled with initial state q_0
- Consider a node n labelled $s \in \Sigma$ and with state q
- If $s \in \Sigma_2 \cup \Sigma_3$ and $qs \Rightarrow (q_1, \ldots, q_k) \in \Delta$ then each successor ni is labelled q_i
- If $s \in \Sigma_1$, $n1$ is labelled $s' \in \Sigma_3$ and $qs \Rightarrow q's' \in \Delta$ then $n1$ is labelled q'.
- If $s \in \Sigma_1$, $n1$ is labelled $x \in \Sigma_2$, $m \downarrow n1$ in t, m labelled q' and $(q', q)s \Rightarrow q_1 x \in \Delta$ then $n1$ is labelled q_1



Run of automaton A on $t \in T_\Sigma$

- An additional labelling of t with states Q defined top-down.
  The root is labelled with initial state q_0
- Consider a node n labelled $s \in \Sigma$ and with state q
- If $s \in \Sigma_2 \cup \Sigma_3$ and $qs \Rightarrow (q_1, \ldots, q_k) \in \Delta$ then each successor ni is labelled q_i
- If $s \in \Sigma_1$, $n1$ is labelled $s' \in \Sigma_3$ and $qs \Rightarrow q's' \in \Delta$ then $n1$ is labelled q'.
- If $s \in \Sigma_1$, $n1$ is labelled $x \in \Sigma_2$, $m \downarrow n1$ in t, m labelled q' and $(q', q)s \Rightarrow q_1 x \in \Delta$ then $n1$ is labelled q_1

$$
\begin{tikzcd}
m \arrow[d, dotted, no head] \arrow[dd, bend right] & q' & \\
s \arrow[d, no head] & q & (q', q)s \Rightarrow q_1 x \\
x & q_1 &
\end{tikzcd}
$$

Run of automaton A on $t \in T_\Sigma$

- A accepts the Σ-tree t iff there is a run of A on t such that every node of t is labelled with a state in Q.

Run of automaton A on $t \in T_\Sigma$

- A accepts the Σ-tree t iff there is a run of A on t such that every node of t is labelled with a state in Q.
- Let $T_\Sigma(A)$ be the set of Σ-trees accepted by A

Run of automaton A on $t \in \mathrm{T}_\Sigma$

- A accepts the Σ-tree t iff there is a run of A on t such that every node of t is labelled with a state in Q.
- Let $\mathrm{T}_\Sigma(\mathrm{A})$ be the set of Σ-trees accepted by A
- Fact The nonemptiness problem, given A is $\mathrm{T}_\Sigma(\mathrm{A}) \neq \emptyset$?, is decidable.

Run of automaton A on $t \in \mathrm{T}_\Sigma$

- A accepts the Σ-tree t iff there is a run of A on t such that every node of t is labelled with a state in Q.
- Let $\mathrm{T}_\Sigma(\mathrm{A})$ be the set of Σ-trees accepted by A
- Fact The nonemptiness problem, given A is $\mathrm{T}_\Sigma(\mathrm{A}) \neq \emptyset$?, is decidable.
- Fact Dependency tree automata closed under intersection and union

Run of automaton A on $t \in \mathrm{T}_\Sigma$

- A accepts the Σ-tree t iff there is a run of A on t such that every node of t is labelled with a state in Q.
- Let $\mathrm{T}_\Sigma(\mathrm{A})$ be the set of Σ-trees accepted by A
- Fact The nonemptiness problem, given A is $\mathrm{T}_\Sigma(\mathrm{A}) \neq \emptyset$?, is decidable.
- Fact Dependency tree automata closed under intersection and union
- OPEN PROBLEM IN PAPER: is non-emptiness decidable for the alternating automata? (Undecidable shown in [Ong + Tzevelekos 2009], to appear at LICS 09)

Decision questions

- Higher-order unification
- $v = u$ contains free variables $x_1, \ldots, x_n$

Decision questions

- Higher-order unification
- $v = u$ contains free variables $x_1, \ldots, x_n$
- Solution $\theta = \{t_1/x_1, \ldots, t_n/x_n\}$ such that $v\theta =_{\beta\eta} u\theta$
  Simultaneous substitution

Decision questions

- Higher-order unification
- $v = u$ contains free variables $x_1, \ldots, x_n$
- Solution $\theta = \{t_1/x_1, \ldots, t_n/x_n\}$ such that $v\theta =_{\beta\eta} u\theta$
  Simultaneous substitution
- Decision question: given $v = u$, does it have a solution ?
- Order is max order of the x_is

Decision questions

- Higher-order unification
- $v = u$ contains free variables $x_1, \ldots, x_n$
- Solution $\theta = \{t_1/x_1, \ldots, t_n/x_n\}$ such that $v\theta =_{\beta\eta} u\theta$
  Simultaneous substitution
- Decision question: given $v = u$, does it have a solution ?
- Order is max order of the x_is
- Undecidable (even at order 2) [Huet 1972,Goldfarb 1981]

Decision questions

- Higher-order unification
- $v = u$ contains free variables $x_1, \ldots, x_n$
- Solution $\theta = \{t_1/x_1, \ldots, t_n/x_n\}$ such that $v\theta =_{\beta\eta} u\theta$
  Simultaneous substitution
- Decision question: given $v = u$, does it have a solution ?
- Order is max order of the x_is
- Undecidable (even at order 2) [Huet 1972,Goldfarb 1981]

Decision questions (cont)

- Higher-order matching
- $v = u$ contains free variables $x_1, \ldots, x_n$ BUT u closed

Decision questions (cont)

- Higher-order matching
- $v = u$ contains free variables $x_1, \ldots, x_n$ BUT u closed
- Solution $\theta = \{t_1/x_1, \ldots, t_n/x_n\}$ such that $v\theta =_{\beta\eta} u$
  W.l.o.g assume $v, u : \mathbf{0}$

Decision questions (cont)

- Higher-order matching
- $v = u$ contains free variables $x_1, \ldots, x_n$ BUT u closed
- Solution $\theta = \{t_1/x_1, \ldots, t_n/x_n\}$ such that $v\theta =_{\beta\eta} u$
  W.l.o.g assume $v, u : \mathbf{0}$
- Decision question: given $v = u$, does it have a solution ?
- Order is max order of the x_is

Decision questions (cont)

- Higher-order matching
- $v = u$ contains free variables $x_1, \ldots, x_n$ BUT u closed
- Solution $\theta = \{t_1/x_1, \ldots, t_n/x_n\}$ such that $v\theta =_{\beta\eta} u$
  W.l.o.g assume $v, u : \mathbf{0}$
- Decision question: given $v = u$, does it have a solution ?
- Order is max order of the x_is
- Huet conjecture decidable [Huet 1976]

Decision questions (cont)

- Higher-order matching
- $v = u$ contains free variables $x_1, \ldots, x_n$ BUT u closed
- Solution $\theta = \{t_1/x_1, \ldots, t_n/x_n\}$ such that $v\theta =_{\beta\eta} u$
  W.l.o.g assume $v, u : \mathbf{0}$
- Decision question: given $v = u$, does it have a solution ?
- Order is max order of the x_is
- Huet conjecture decidable [Huet 1976]
- Up to order 4 decidable + special cases [Huet 1976, Dowek 1993, Padovani 2000, ...]

Decision questions (cont)

- Higher-order matching
- $v = u$ contains free variables $x_1, \ldots, x_n$ BUT u closed
- Solution $\theta = \{t_1/x_1, \ldots, t_n/x_n\}$ such that $v\theta =_{\beta\eta} u$
  W.l.o.g assume $v, u : \mathbf{0}$
- Decision question: given $v = u$, does it have a solution ?
- Order is max order of the x_is
- Huet conjecture decidable [Huet 1976]
- Up to order 4 decidable + special cases [Huet 1976, Dowek 1993, Padovani 2000, ...]
- Undecidable for $=_\beta$ [Loader 2003]

Matching is essentially monadic

- Given $v = u$ with $x_1 : A_1, \ldots, x_n : A_n$ in v

Matching is essentially monadic

- Given $v = u$ with $x_1 : A_1, \ldots, x_n : A_n$ in v
- $\boxed{x(\lambda x_1 \ldots x_n.v) = u}$ where $x : ((A_1, \ldots, A_n, \mathbf{0}), \mathbf{0})$
- Conceptually simpler problem: just one free variable

Matching is essentially monadic

- Given $v = u$ with $x_1 : A_1, \ldots, x_n : A_n$ in v
- $x(\lambda x_1 \ldots x_n.v) = u$ where $x : ((A_1, \ldots, A_n, \mathbf{0}), \mathbf{0})$
- Conceptually simpler problem: just one free variable
- Called "interpolation": $x\,w = u$ where $x : (B, \mathbf{0})$, $w : B$, $u : \mathbf{0}$ in normal form.

Matching is essentially monadic

- Given $v = u$ with $x_1 : A_1, \ldots, x_n : A_n$ in v
- $x(\lambda x_1 \ldots x_n.v) = u$ where $x : ((A_1, \ldots, A_n, \mathbf{0}), \mathbf{0})$
- Conceptually simpler problem: just one free variable
- Called "interpolation": $x\, w = u$ where $x : (B, \mathbf{0})$, $w : B$, $u : \mathbf{0}$ in normal form.
- Solution t in normal form such that $tw \to_\beta u$

Matching is essentially monadic

- Given $v = u$ with $x_1 : A_1, \ldots, x_n : A_n$ in v
- $x(\lambda x_1 \ldots x_n.v) = u$ where $x : ((A_1, \ldots, A_n, \mathbf{0}), \mathbf{0})$

- Canonical solution $x = \lambda z.z\, t_1 \ldots t_n$ where t_is are closed
$\Rightarrow$
$v\{t_1/x_1, \ldots, t_n/x_n\} \rightarrow_\beta u$ Solves $v = u$

Matching is essentially monadic

- Given $v = u$ with $x_1 : A_1, \ldots, x_n : A_n$ in v
- $x(\lambda x_1 \ldots x_n.v) = u$ where $x : ((A_1, \ldots, A_n, \mathbf{0}), \mathbf{0})$
- Canonical solution $x = \lambda z.z\, t_1 \ldots t_n$ where t_is are closed
  $\Rightarrow$
  $v\{t_1/x_1, \ldots, t_n/x_n\} \to_\beta u$ Solves $v = u$
- Restrict constants in solution terms to be those in u plus fresh $c : \mathbf{0}$: finite alphabet

Example

- $x_1(\lambda z.x_1(\lambda z'.za)) = a$
  4th-order when $z, z' : (\mathbf{0}, \mathbf{0})$ and $x_1 : (((\mathbf{0}, \mathbf{0}), \mathbf{0}), \mathbf{0})$

Example

- $x_1(\lambda z.x_1(\lambda z'.za)) = a$
  4th-order when $z, z' : (\mathbf{0}, \mathbf{0})$ and $x_1 : (((\mathbf{0}, \mathbf{0}), \mathbf{0}), \mathbf{0})$
- Associated interpolation problem
  $x(\lambda x_1.x_1(\lambda z.x_1(\lambda z'.za))) = a$
  $x : (((((\mathbf{0}, \mathbf{0}), \mathbf{0}), \mathbf{0}), \mathbf{0}), \mathbf{0})$.

Example

- $x_1(\lambda z.x_1(\lambda z'.za)) = a$
  4th-order when $z, z' : (\mathbf{0}, \mathbf{0})$ and $x_1 : (((\mathbf{0}, \mathbf{0}), \mathbf{0}), \mathbf{0})$
- Associated interpolation problem
  $x(\lambda x_1.x_1(\lambda z.x_1(\lambda z'.za))) = a$
  $x : (((((\mathbf{0}, \mathbf{0}), \mathbf{0}), \mathbf{0}), \mathbf{0}), \mathbf{0})$.
- A canonical solution has the form
  $\lambda x.x(\lambda y.y(\lambda y_1^1 \dots y(\lambda y_1^k.s) \dots))$
  where s is the constant a or one of the variables y_1^j, $1 \leq j \leq k$.

Padovani's decidability proof for order 4 and tree automata

- $t \equiv_A^u t'$ if $t, t' : A$ solve same interpolation problems with u on right hand side
  $t \equiv_A^u t'$ "observational equivalence"

Padovani's decidability proof for order 4 and tree automata

- $t \equiv_A^u t'$ if $t, t' : A$ solve same interpolation problems with u on right hand side
  $t \equiv_A^u t'$ "observational equivalence"
- Set of normal forms $T_A / \equiv_A^u$ is finite
  Up to order 4, representatives for each equivalence class can be decided

Padovani's decidability proof for order 4 and tree automata

- $t \equiv_A^u t'$ if $t, t' : A$ solve same interpolation problems with u on right hand side
  $t \equiv_A^u t'$ "observational equivalence"
- Set of normal forms $T_A / \equiv_A^u$ is finite
  Up to order 4, representatives for each equivalence class can be decided
- [Comon + Jurski 1997]

Theorem
Set of solutions to a 4th-order problem is regular: recognizable by a tree automaton

Padovani's decidability proof for order 4 and tree automata

- $t \equiv_A^u t'$ if $t, t' : A$ solve same interpolation problems with u on right hand side
  $t \equiv_A^u t'$ "observational equivalence"
- Set of normal forms $T_A / \equiv_A^u$ is finite
  Up to order 4, representatives for each equivalence class can be decided
- [Comon + Jurski 1997]

Theorem

Set of solutions to a 4th-order problem is regular: recognizable by a tree automaton

- In the proof states of automaton built from $\equiv_{A'}^{u'}$ representatives where u' subterm of u, A' subtype of A.

Open Question

- Given $xw = u$, is set of solutions automaton recognisable ?
  Two problems extending Comon+ Jurski's result

Open Question

- Given $xw = u$, is set of solutions automaton recognisable ?

  Two problems extending Comon+ Jurski's result

  1. Ensuring finitely many states in automaton

     $\leq$ 4th-order: only need to examine finitely many terms to build $\equiv^{u'}_{A'}$ classes

     $>$ 4th-order, need to examine infinitely many terms to build $\equiv^{u'}_{A'}$ classes

     equivalent to solving matching problem! [Padovani 2001]

Open Question

- Given $xw = u$, is set of solutions automaton recognisable ?

  Two problems extending Comon+ Jurski's result

  1. Ensuring finitely many states in automaton

     $\leq$ 4th-order: only need to examine finitely many terms to build $\equiv_{A'}^{u'}$ classes

     $>$ 4th-order, need to examine infinitely many terms to build $\equiv_{A'}^{u'}$ classes

     equivalent to solving matching problem! [Padovani 2001]
  2. Ensuring finite alphabet in automaton

Overcoming one of the problems

- Given $xw = u$, is set of solutions automaton recognisable ?

Overcoming one of the problems

- Given $xw = u$, is set of solutions automaton recognisable ?

- Theorem
  The set of solutions built out of a fixed finite alphabet of variables is regular (recognised by a classical tree automaton)

  [Stirling 2007]

Overcoming one of the problems

- Given $xw = u$, is set of solutions automaton recognisable ?
- Theorem
  The set of solutions built out of a fixed finite alphabet of variables is regular (recognised by a classical tree automaton)

  [Stirling 2007]
  - States of automaton built from variable profiles (based on [Ong 2006] schema paper). Proof uses game-theoretic characterisation of interpolation from [Stirling 2005,2006]

Overcoming both problems

- Given $xw = u$, is set of solutions automaton recognisable ?

Overcoming both problems

  - Given $xw = u$, is set of solutions automaton recognisable ?

- Theorem
  The set of solutions is recognised by a alternating dependency tree automaton

Overcoming both problems

- Given $xw = u$, is set of solutions automaton recognisable ?

- Theorem
  The set of solutions is recognised by a alternating dependency tree automaton
  - Proof uses games

Overcoming both problems

  - Given $xw = u$, is set of solutions automaton recognisable ?
- Theorem
  The set of solutions is recognised by a alternating dependency tree automaton
  - Proof uses games
  - OPEN QUESTION: is there a class of tree automata between dependency tree automata and alternating tree automata that can recognise solutions to matching.