Towards Static Analysis of Functional Programs using Tree Automata Completion

Thomas Genet

INRIA/IRISA/Université de Rennes 1

Background: Term Rewriting

- Set of ranked symbols $\mathcal{F} = \{cons : 2, nil : 0, app : 2, A : 0, B : 0\}$
- Set of ground terms $\mathcal{T}(\mathcal{F}) = \{app(cons(A, nil), nil), app(A, B), \ldots\}$
- Term Rewriting System $\mathcal{R} = \begin{cases} app(nil, x) \rightarrow x \\ app(cons(x, y), z) \rightarrow cons(x, app(y, z)) \end{cases}$
- $app(cons(A, nil), nil) \rightarrow_{\mathcal{R}} cons(A, app(nil, nil)) \rightarrow_{\mathcal{R}} cons(A, nil)$
- $app(cons(A, nil), nil) \rightarrow_{\mathcal{R}}{}^{*} cons(A, nil)$
- $app(x, y) \rightarrow app(x, x)$ is a left-linear rule, but not right-linear
- $app(x, y) \rightarrow app(y, x)$ is a linear rule

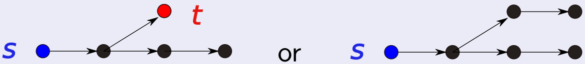
Set of reachable terms: $\mathcal{R}^*(\mathcal{L}) = \{u \in \mathcal{T}(\mathcal{F}) \mid s \in \mathcal{L} \wedge s \rightarrow_{\mathcal{R}}{}^{*} u\}$

e.g. $\mathcal{R}^*(\{app(cons(A, nil), nil)\}) =$
$\{app(cons(A, nil), nil), cons(A, app(nil, nil)), cons(A, nil)\}$

Motivation: Reachability analysis of rewriting

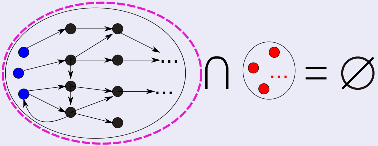
Given $\mathcal{R}$ and $s, t \in \mathcal{T}(\mathcal{F})$, prove that $s \rightarrow_{\mathcal{R}}^{*} t$ or $s \not\rightarrow_{\mathcal{R}}^{*} t$.

"Finite" Reachability: if $\mathcal{R}$ terminates, just rewrite!



"Infinite" Reachability: $\mathcal{R}$ does not terminate and infinite sets of s

- Using Abstractions and Narrowing [Bae, Escobar, Meseguer, 13]
- Using Induction and sufficient completeness [Rocha, Meseguer, 11]
- Using Tree Automata and basic set operations:



"Tree Automata Completion" techniques, developped since 1998
(Applied to cryptographic protocol and Java programs verification)

Motivation: Static Analysis of Functional Programs

OCaml type checking

```
let rec append l1 l2 = match l1 with
  | [] -> []
  | h::t -> h :: (append t l2);;
# val append: 'a list -> 'a list -> 'a list = <fun>

let rec rev l = match l with
  | [] -> []
  | h::t -> append (rev t) [h];;
# val rev: 'a list -> 'a list = <fun>
```

We would like to have... more precision than what offer simple types

```
# val rev: 'a list -> empty list
```

Motivation: Static Analysis of Functional Programs (II)

```
let rec append l1 l2 = match l1 with
  | [] -> l2
  | h::t -> h :: (append t l2);;

let rec rev l = match l with
  | [] -> []
  | h::t -> append (rev t) [h];;
```

OCaml interpreter

```
# append [1] [2;3];;
-:int list= [1;2;3]
```

We would like to have... an OCaml Abstract Interpreter

`# append [A*] [A*];;` `-:abst list= [A*]`	`# append [A*] [B*];;` `-:abst list= [A*;B*]`	`# rev [A*;B*];;` `...= [B*;A*]`

Motivation: Lightweight Formal Verification

```
let rec delete x l= match l with
  | [] -> []
  | h::t -> if h=x then t else h::(delete x t);;
```

OCaml interpreter

```
# delete 2 [1;2;3];;
-:int list= [1; 3]
```

Ocaml Abstract Interpreter for Lightweight Formal Verification

```
# delete A [(A|B)*];;
```

`-:abst list= [(A|B)*]` but expected `[B*]`... the function is bugged!

Outline

1. Background on tree automata completion
2. Termination theorems for completion
3. Demos on "functional TRS"
4. Further and ongoing research

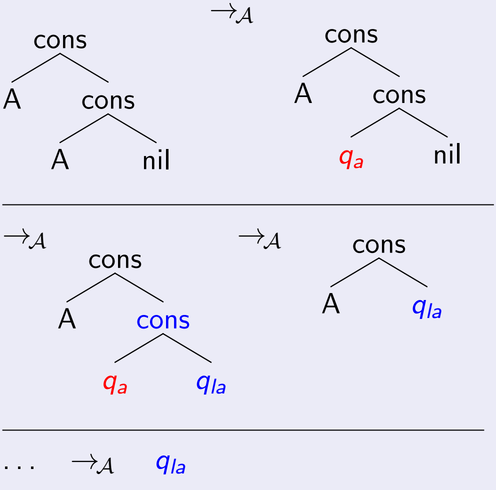
... Related work scattered in subsections

Background: Tree Automata (regular tree languages)

Recognized language $\mathcal{L}(\mathcal{A}, q) = \{s \in \mathcal{T}(\mathcal{F}) \mid s \rightarrow_{\mathcal{A}}^{*} q\}$

$\mathcal{A} = \langle \mathcal{F}, \mathcal{Q}, \mathcal{Q}_f, \Delta \rangle$
with $\mathcal{Q} = \{q_a, q_b, q_{la}, q_{lb}, q_f\}$
$\mathcal{Q}_f = \{q_f\}$ and

$$\Delta = \left\{ \begin{array}{l} A \rightarrow q_a \\ B \rightarrow q_b \\ \\ nil \rightarrow q_{la} \\ nil \rightarrow q_{lb} \\ \\ cons(q_a, q_{la}) \rightarrow q_{la} \\ cons(q_b, q_{lb}) \rightarrow q_{lb} \\ \\ app(q_{la}, q_{lb}) \rightarrow q_f \end{array} \right.$$



Background: Tree Automata (II)

Recognized language: $\mathcal{L}(\mathcal{A}, q) = \{s \in \mathcal{T}(\mathcal{F}) \mid s \rightarrow_{\mathcal{A}}{}^{*} q\}$

$\mathcal{A} = \langle \mathcal{F}, \mathcal{Q}, \mathcal{Q}_f, \Delta \rangle$
with $\mathcal{Q} = \{q_a, q_b, q_{la}, q_{lb}, q_f\}$
$\mathcal{Q}_f = \{q_f\}$ and

$$\Delta = \begin{cases} A \rightarrow q_a \\ B \rightarrow q_b \\ \\ nil \rightarrow q_{la} \\ nil \rightarrow q_{lb} \\ cons(q_a, q_{la}) \rightarrow q_{la} \\ cons(q_b, q_{lb}) \rightarrow q_{lb} \\ \\ app(q_{la}, q_{lb}) \rightarrow q_f \end{cases}$$

$\mathcal{L}(\mathcal{A}, q_{la}) = \{nil, cons(A, nil), cons(A, ...)\}$
$\mathcal{L}(\mathcal{A}, q_{la}) = \texttt{[A*]}$

$\mathcal{L}(\mathcal{A}, q_{lb}) = \{nil, cons(b, nil), cons(b, ...)\}$
$\mathcal{L}(\mathcal{A}, q_{lb}) = \texttt{[B*]}$

$\mathcal{L}(\mathcal{A}, q_f) = \{app(\texttt{[A*]}, \texttt{[B*]})\}$
$\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}, q_f)$

How to recognize $\mathcal{R}^*(\mathcal{L}(\mathcal{A}))$?
$\Longrightarrow$ complete $\mathcal{A}$!!

Background: Tree Automata Completion

Tree automata completion semi-algorithm

- Input: a left-linear TRS $\mathcal{R}$, a tree automaton $\mathcal{A}$
- Output:
  an automaton $\mathcal{A}^*_\mathcal{R}$ such that $\mathcal{L}(\mathcal{A}^*_\mathcal{R}) \supseteq \mathcal{R}^*(\mathcal{L}(\mathcal{A}))$
- Principle: complete $\mathcal{A}^0_\mathcal{R} = \mathcal{A}$ with new transitions into $\mathcal{A}^1_\mathcal{R}, \mathcal{A}^2_\mathcal{R}, \dots$ until reaching $\mathcal{A}^*_\mathcal{R}$, a fixpoint

One completion step: computing $\mathcal{A}^{i+1}_\mathcal{R}$ from $\mathcal{A}^i_\mathcal{R}$

Add new transitions: $\forall l \to r \in \mathcal{R}$, $\forall q \in \mathcal{Q}$, $\forall \sigma : \mathcal{X} \mapsto \mathcal{Q}$:

$$\begin{tikzcd} l\sigma \arrow[r, "\mathcal{R}"'] \arrow[d, "\mathcal{A}^i_\mathcal{R}"', "*"] & r\sigma \arrow[d, "*"', "\mathcal{A}^{i+1}_\mathcal{R}"] \\ q & q' \arrow[l, "\mathcal{A}^{i+1}_\mathcal{R}"] \end{tikzcd}$$

$\mathcal{R}$ classes enjoying ($\mathcal{L}$ regular $\Longrightarrow \mathcal{R}^*(\mathcal{L})$ regular)

$$
\begin{tikzcd}
 & \text{G} \arrow[d] \\
\text{RL-M} \arrow[dd] & \text{L-SM} \arrow[d] \\
 & \text{L-G}^{-1} \arrow[dl] \arrow[d] \\
\text{RL-G}^{-1} \arrow[dd] & \text{L-GSM} \arrow[d] \\
 & \text{L-FPO} \arrow[dl] \arrow[d] \\
\text{RL-FPO} & \text{L-GFPO}
\end{tikzcd}
$$

G Ground [Dauchet, Tison, 90], [Brainerd, 69]

RL-M Right-linear and Monadic [Salomaa, 88]

L-SM Linear and Semi-Monadic [Coquidé et al., 91]

L-G^{-1} Linear and inversely Growing [Jacquemard, 96]

RL-G^{-1} Right-linear and inversely Growing [Nagaya, Toyama, 99]

L-GSM Linear Generalized Semi-Monadic [Gyenizse, Vágvölgyi, 98]

L-FPO, **RL-FPO** (Right)-Linear Finite Path Overlapping [Takai et al. 00]

L-GFPO Linear Generalized Finite Path Overlapping [Takai 04]

$\mathcal{R}$ classes enjoying ($\mathcal{L}$ regular $\Longrightarrow \mathcal{R}^*(\mathcal{L})$ regular) (II)

Plus some classes incomparable with others:

L-IOSLT Linear I/O Separated Layered Transducing (a.k.a. Tree Transducers) [Seki et al. 02]

Constructor Constructor based + constraints on $\mathcal{L}$ [Réty 99]

WOS Well Oriented Systems [Bouajjani, Touili, 02]

$\mathcal{R}$ classes enjoying ($\mathcal{L}$ regular $\Longrightarrow \mathcal{R}^*(\mathcal{L})$ regular) (IV)

Constructor [Réty 99]

- Distiguishes between defined and constructor symbols
- Forbids dupplication (right linear TRSs)
- No nested defined symbols in the right-hand sides
- Terms of $\mathcal{L}$ have only of a finite number of defined symbols

$\left.\begin{array}{l} app(nil, x) \to x \\ app(cons(x, y), z) \to cons(x, app(y, z)) \end{array}\right\} = \mathcal{R}$ $\{app(app([A*], [B*]), [C*])\} = \mathcal{L}$	$\mathcal{R}^*(\mathcal{L})$ $\vert\vert$ $\{\, [A*, B*, C*] \,\}$

When do $\mathcal{A}^*_{\mathcal{R}}$ exists and $\mathcal{L}(\mathcal{A}^*_{\mathcal{R}}) = \mathcal{R}^*(\mathcal{L}(\mathcal{A}))$?

$$
\begin{tikzcd}
 & \text{G} \arrow[d] \\
\text{RL-M} \arrow[dd] & \text{L-SM} \arrow[d] \\
 & \text{L-G}^{-1} \arrow[d] \arrow[dl] \\
\text{RL-G}^{-1} \arrow[dd] & \text{L-GSM} \arrow[d] \\
 & \text{L-FPO} \arrow[d] \arrow[dl] \\
\text{RL-FPO} & \text{L-GFPO}
\end{tikzcd}
$$

Constructor

[with G. Feuillade, V. Viet Triem Tong, 04]
updated in 2009

L-IOSLT WOS

- with exact completion (default)
- with specific normalization strategies
- it also covers TRS and tree automata outside of those classes!

To deal with more TRS: equational abstraction

- Sets of equations $$E = \begin{cases} app(x, y) = app(y, x) \\ cons(x, y) = y \end{cases}$$

Congruence relation $=_E$

$$\begin{aligned} app(cons(A, cons(A, nil)), nil) &=_E \quad app(cons(A, nil), nil) \\ &=_E \quad app(nil, cons(A, nil)) \end{aligned}$$

Rewriting with $\mathcal{R}$ modulo E: $s \underset{\mathcal{R}/E}{\rightarrow} t \quad \Leftrightarrow \quad s =_E s' \rightarrow_{\mathcal{R}} t' =_E t$

$$\begin{aligned} app(cons(A, cons(A, nil)), nil) &=_E \quad app(nil, cons(A, nil)) \\ &\rightarrow_{\mathcal{R}} \quad cons(A, nil) \\ &=_E \quad cons(B, cons(B, cons(A, nil))) \end{aligned}$$

$\mathcal{T}(\mathcal{F})/_{=_E}$ is the set of equivalence classes of $\mathcal{T}(\mathcal{F})$ w.r.t. $=_E$

e.g. $\mathcal{T}(\mathcal{F})/_{=_E}$ is $\{[nil], [app(nil, nil)], [app(app(nil, nil), nil)], \ldots\}$

and the class $[nil]$ contains nil, $cons(A, nil)$, $cons(A, cons(B, nil))$, $\ldots$

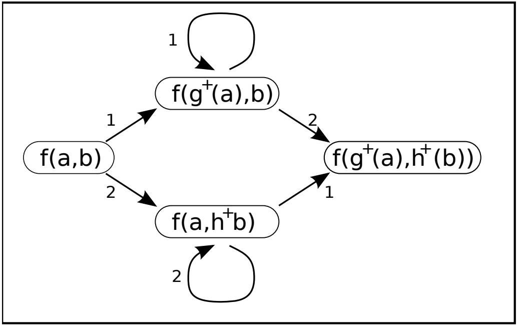
To deal with more TRS: equational abstraction (II)

[Meseguer, Palomino, Martí-Oliet, 03] [Takai, 04]

$$\mathcal{R} = \begin{cases} (1)\ f(x,y) \to f(g(x),y) \\ (2)\ f(x,y) \to f(x,h(y)) \end{cases}$$

prove that $f(a,b) \not\to_{\mathcal{R}}^{*} f(a,h(g(b)))$?

using $E = \{g(g(x)) = g(x), h(h(x)) = h(x)\}$



$$f(a,b) \not\to_{\mathcal{R}/E}^{*} f(a,h(g(b))) \quad \Longrightarrow \quad f(a,b) \not\to_{\mathcal{R}}^{*} f(a,h(g(b)))$$

Tree Automata Completion with Equational Abstraction

Tree automata completion semi-algorithm

- Input: a TRS $\mathcal{R}$, a tree automaton $\mathcal{A}$ and approximation equations E
- Output: an automaton $\mathcal{A}^*_{\mathcal{R},E}$
- Principle: complete $\mathcal{A}^0_{\mathcal{R},E} = \mathcal{A}$ with new transitions into $\mathcal{A}^1_{\mathcal{R},E}, \mathcal{A}^2_{\mathcal{R},E}, \ldots$ until reaching $\mathcal{A}^*_{\mathcal{R},E}$, a ($\mathcal{R}$-closed) fixpoint

Tree Automata Completion with Equational Abstraction

Tree automata completion semi-algorithm

- Input: a TRS $\mathcal{R}$, a tree automaton $\mathcal{A}$ and approximation equations E
- Output: an automaton $\mathcal{A}^*_{\mathcal{R},E}$
- Principle: complete $\mathcal{A}^0_{\mathcal{R},E} = \mathcal{A}$ with new transitions into $\mathcal{A}^1_{\mathcal{R},E}, \mathcal{A}^2_{\mathcal{R},E}, \ldots$ until reaching $\mathcal{A}^*_{\mathcal{R},E}$, a ($\mathcal{R}$-closed) fixpoint

One completion step: computing $\mathcal{A}^{i+1}_{\mathcal{R},E}$ from $\mathcal{A}^i_{\mathcal{R},E}$

Add new transitions
$\forall l \to r \in \mathcal{R}$, $q \in \mathcal{Q}$, $\sigma : \mathcal{X} \mapsto \mathcal{Q}$:

$$\begin{tikzcd}
l\sigma \arrow[r, "\mathcal{R}"'] \arrow[d, "\mathcal{A}^i_{\mathcal{R},E}"', "*"] & r\sigma \arrow[d, "*"', "{\mathcal{A}^{i+1}_{\mathcal{R}}, \not\in}"] \\
q & q' \arrow[l, "\mathcal{A}^{i+1}_{\mathcal{R}}"]
\end{tikzcd}$$

Tree Automata Completion with Equational Abstraction

Tree automata completion semi-algorithm

- Input: a TRS $\mathcal{R}$, a tree automaton $\mathcal{A}$ and approximation equations E
- Output: an automaton $\mathcal{A}^*_{\mathcal{R},E}$
- Principle: complete $\mathcal{A}^0_{\mathcal{R},E} = \mathcal{A}$ with new transitions into $\mathcal{A}^1_{\mathcal{R},E}, \mathcal{A}^2_{\mathcal{R},E}, \ldots$ until reaching $\mathcal{A}^*_{\mathcal{R},E}$, a ($\mathcal{R}$-closed) fixpoint

One completion step: computing $\mathcal{A}^{i+1}_{\mathcal{R},E}$ from $\mathcal{A}^i_{\mathcal{R},E}$

Add new transitions
$\forall l \to r \in \mathcal{R}$, $q \in \mathcal{Q}$, $\sigma : \mathcal{X} \mapsto \mathcal{Q}$:

$$\begin{tikzcd}
l\sigma \arrow[r, "\mathcal{R}"'] \arrow[d, "\mathcal{A}^i_{\mathcal{R},E}"', "*"] & r\sigma \arrow[d, "*"', "{\mathcal{A}^{i+1}_{\mathcal{R}}, \not\epsilon}"] \\
q & q' \arrow[l, "\mathcal{A}^{i+1}_{\mathcal{R}}"]
\end{tikzcd}$$

Simplification: Merge E-equivalent states
$\forall u = v \in E$, $q_1, q_2 \in \mathcal{Q}$, $\sigma : \mathcal{X} \mapsto \mathcal{Q}$:

$$\begin{tikzcd}
u\sigma \arrow[r, equal, "E"'] \arrow[d, "{\mathcal{A}^{i+1}_{\mathcal{R}}, \not\epsilon}"', "*"] & v\sigma \arrow[d, "*"', "{\mathcal{A}^{i+1}_{\mathcal{R}}, \not\epsilon}"] \\
q_1 & q_2
\end{tikzcd}$$

Tree Automata Completion to approximate $\mathcal{R}^*(\mathcal{L}(\mathcal{A}))$

$\mathcal{R}^*(\mathcal{L}(\mathcal{A})) \subseteq \mathcal{L}(\mathcal{A}^*_{\mathcal{R},E}) \subseteq {\mathcal{R}_E}^*(\mathcal{L}(\mathcal{A}))$ [with V. Rusu, 2010]

Completion with Equational Abstraction: Demo

- `demo_basic.txt`

Theorem 1 (Upper bound)

Given a left-linear TRS $\mathcal{R}$, a tree automaton $\mathcal{A}$ and a set of equations E, if completion terminates on $\mathcal{A}^*_{\mathcal{R},E}$ then $\mathcal{L}(\mathcal{A}^*_{\mathcal{R},E}) \supseteq \mathcal{R}^*(\mathcal{L}(\mathcal{A}))$.

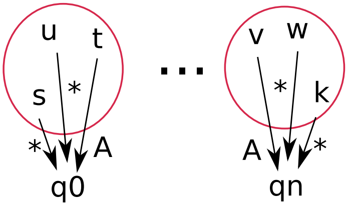
How to be sure that completion with equational abstraction terminates?

Equations guaranteeing termination of completion

Theorem 2 (Myhill-Nerode, [TATA])

Tree Automaton $\mathcal{A}$ $\Leftrightarrow$
- *There exists a set of equations* E *such that*
- $\mathcal{T}(\mathcal{F})/_{=E}$ *is a finite set of equivalence classes*
- $\mathcal{L}(\mathcal{A})$ *is the union of equivalence classes of* $\mathcal{T}(\mathcal{F})/_{=E}$

$\mathcal{T}(\mathcal{F})/_{=_E}$ is



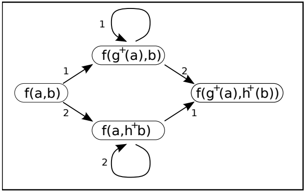
Equations guaranteeing termination of completion (II)

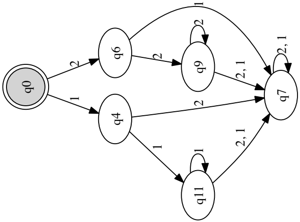
Intuition

$\mathcal{T}(\mathcal{F})/_{=_E}$ is finite $\Rightarrow$ finite set of states in $\mathcal{A}^*_{\mathcal{R},E}$ $\Rightarrow$ completion terminates

Two problems

- Determining wether $\mathcal{T}(\mathcal{F})/_{=_E}$ is finite is undecidable
  (private communication with Sophie Tison)
- Classes of $\mathcal{T}(\mathcal{F})/_{=_E}$ and states of $\mathcal{A}^*_{\mathcal{R},E}$ do not necessarily coincide!





Demo:
`demo_basic.txt`

Simplification of $\mathcal{A}$ with E does not implement $=_E$

Reflexivity of $=_E$ is not preserved by $\mathcal{A}^*_{\mathcal{R},E}$

We can have
$$\begin{array}{ccc} s & & s \\ \mathcal{A}^*_{\mathcal{R},E}\Big\downarrow * & & *\Big\downarrow \mathcal{A}^*_{\mathcal{R},E} \\ q_1 & & q_2 \end{array}$$
but q_1 and q_2 are not merged in $\mathcal{A}^*_{\mathcal{R},E}$

Simple fix: have $E \supseteq E^r$

With $E^r = \{f(x_1, \ldots, x_n) = f(x_1, \ldots, x_n) \mid f \in \mathcal{F}\}$

Transitivity of $=_E$ is not preserved by $\mathcal{A}^*_{\mathcal{R},E}$

We may have $s =_E t =_E u$ but $s \xrightarrow{*}_{\mathcal{A}^*_{\mathcal{R},E}} q_1$ and $u \xrightarrow{*}_{\mathcal{A}^*_{\mathcal{R},E}} q_2$

But no simple fix!

Simplification of $\mathcal{A}$ with E does not implement $=_E$ (II)

Proposed solution: set of contracting equations E^c

- Simplification with E^c preserves transitivity of $=_{E^c}$
- $\mathcal{T}(\mathcal{F})/_{=_{E^c}}$ is trivially finite

Definition 3 (Sets of contracting equations, E^c)

A set of equations is contracting denoted by E^c, if all equations of E^c are of the form $u = u|_p$ with $u \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ a linear term, $p \neq \lambda$, and if the set of normal forms of $\overrightarrow{E^c} = \{u \to u|_p \mid u = u|_p \in E^c\}$ is finite.

Example 4 (Contracting equations for $\mathcal{F} = \{f : 1, g : 2, a : 0\}$)

$E^c = \{f(f(x)) = f(x), g(x, y) = y\}$ is contracting.

$\overrightarrow{E^c} = \{f(f(x)) \to f(x), g(x, y) \to y\}$

normal forms of $\overrightarrow{E^c}$ are $\{a, f(a)\}$ and $\mathcal{T}(\mathcal{F})/_{=E^c} = \{[a]_{E^c}, [f(a)]_{E^c}\}$

Equations guaranteeing termination of completion (III)

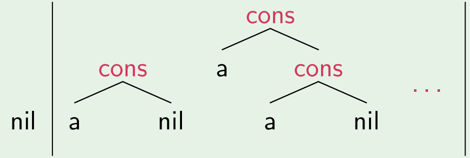
Theorem 5 (Termination of completion)

Let $\mathcal{A}_0$ be a tree automaton, $\mathcal{R}$ a left linear TRS and E a set of equations. If $E \supseteq E^r \cup E^c$, then completion of $\mathcal{A}_0$ by $\mathcal{R}$ and E terminates.

Good but still not well adapted to static analysis of functional programs

Example 6 (Equations for the `rev` function)

Let $\mathcal{F} = \{rev : 1, app : 2, cons : 2, nil : 0, a : 0\}$.



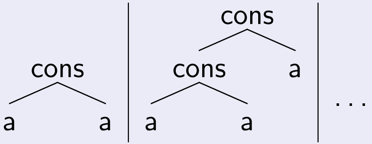
With $E^c =$ $\{cons(x, cons(y, z)) = cons(x, z)\}$

But $\mathcal{T}(\mathcal{F})/_{=E^c}$ is not finite!

Equations guaranteeing termination of completion (IV)

With $E^c = \{cons(x, cons(y, z)) = cons(x, z)\}$, $\mathcal{T}(\mathcal{F})/_{=_E}$ is not finite!

Infinitely many classes of ill-typed terms



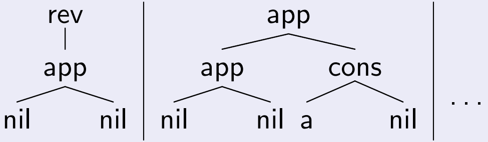
are all in different classes!

Ill-typed terms incompatible with `cons:`α `->` α `list ->` α `list`

We restrict to well-typed terms $\mathcal{T}(\mathcal{F})^{\mathcal{S}}$

With $E = \{cons(x, cons(y, z)) = cons(x, z)\}$, $\mathcal{T}(\mathcal{F})^{\mathcal{S}}/_{=_E}$ is not finite!

Infinitely many classes of partially evaluated terms



are all in different classes!

Partially evaluated terms

Equations guaranteeing termination of completion (V)

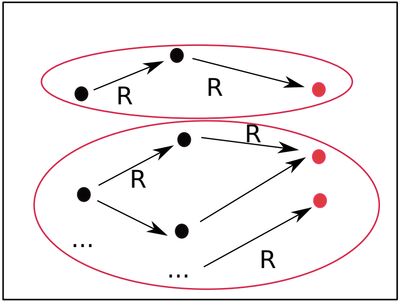
Proposed solution: separate $\mathcal{F}$ into $\mathcal{C} \uplus \mathcal{D}$ and $E \supseteq E^r \cup E^c_{\mathcal{C}} \cup E_{\mathcal{R}}$

- $\mathcal{D}$efined and $\mathcal{C}$onstructor *e.g.* $\mathcal{D} = \{app, rev\}$ and $\mathcal{C} = \{a, cons, nil\}$
- Define $E^c_{\mathcal{C}}$ a set of contracting equations on $\mathcal{T}(\mathcal{C})^{\mathcal{S}}$, such that $\mathcal{T}(\mathcal{C})^{\mathcal{S}}/_{=E^c_{\mathcal{C}}}$ is finite, *e.g.* $cons(x, cons(y, z)) = cons(x, z)$
- Define $E_{\mathcal{R}} = \{l = r \mid l \rightarrow r \in \mathcal{R}\}$ and $E \supseteq E^r \cup E^c_{\mathcal{C}} \cup E_{\mathcal{R}}$

If $\mathcal{R}$ is sufficiently complete then $\mathcal{T}(\mathcal{F})^{\mathcal{S}}/_{=E}$ is finite

$\mathcal{R}$ Sufficiently complete:

$\forall s \in \mathcal{T}(\mathcal{F})^{\mathcal{S}}. \exists t \in \mathcal{T}(\mathcal{C})^{\mathcal{S}}. s \rightarrow_{\mathcal{R}}^* t$



Equations guaranteeing termination of completion (VI)

Theorem 7 (Termination of completion for "functional" TRS)

Let $\mathcal{A}_0$ be a tree automaton recognizing well-sorted terms, $\mathcal{R}$ a sufficiently complete sort-preserving left-linear TRS and E a sort-preserving set of equations. If $E \supseteq E^r \cup E^c_{\mathcal{C},\mathcal{S}} \cup E_{\mathcal{R}}$ with $E^c_{\mathcal{C},\mathcal{S}}$ contracting and $\mathcal{A}_0$ is $\mathcal{R}/E$-coherent then completion of $\mathcal{A}_0$ by $\mathcal{R}$ and E terminates.

- `demo_reverse.txt` `reverse [A+;B+]= [B+;A+]`
- `demo_fact.txt` `fact s*(0)= s+(0)`
- `demo_deleteBug2.txt` `delete A [(A|B)*]= [(A|B)*]`
- `demo_delete.txt` `delete A [(A|B)*]= [B*]`

What is missing for a decent static analyzer for functional languages?

- Have a terminating analysis! ✓
- Deal with higher order functions
- Take evaluation strategies into account
  - call by value (*e.g.* Ocaml) $\approx$ innermost rewrite strategy
  - call by need (*e.g.* Haskell) $\approx$ outermost rewrite strategy $+$ sharing
  - order in pattern matching $\approx$ priority rewrite strategy
- Deal with built-in types
- Have a modular analysis
- User friendly way to display/define language annotations ...

A word about Higher-Order functions

Static analysis of higher-order functional programs

- use higher-order formalisms: *e.g.*
  HORS [L. Ong, 2006], [Broadbent, Carayol, Hague, Serre, 2013]
  PMRS [L. Ong, S. Ramsay, 2011]
- use first-order formalisms (*e.g* tree automata and TRS) with an encoding of higher-order into first-order *e.g.* [N. Jones, 1987]

Example 8 (Encoding of H.O. functions into TRS)

Use an explicit function application operator '@' representing closures.

```
let rec map f l1 = match l1 with
  | [] -> []
  | h::t -> (f h) :: (map f t);;
```

becomes

$@(@(map, f), nil) \rightarrow nil$

$@(@(map, f), cons(h, t)) \rightarrow cons(@(f, h), @(@(map, f), t))$

A word about Higher-Order functions (II)

Is the @-encoding enough?

- Auhors of H.O. formalisms claim that the @-encoding + regular approximation (of Jones) is too imprecise
- On H.O. examples of [L. Ong, S. Ramsay, 2011], we obtained similar results with the @-encoding, TRSs, and tree automata completion

Example 9 (`filter` `nz` on any `nat` `list`, results in a list without 0)

```
let if2 c t e = match c with
  | true -> t
  | false -> e;;
```

```
let nz i= match i with
  | 0 -> false
  | S(x) -> true;;
```

```
let rec filter p l= match l with
  | [] -> []
  | h::t -> if2 (p h) (h::(filter p t)) (filter p t);;
```

Successful on some examples but needs to be investigated further!

A word about evaluation strategies

Example 10 (Terminating with call-by-need but not for call-by-value)

```
let rec sumList(x,y)= (x+y)::sumList(x+y,y+1);;
let rec nth i (x::l)= if i<=0 then x else nth (i-1) l;;
let sum x= nth x (sumList(0,0));;
```

$(sum\,4) = 10$ with call by need and diverges with call-by-value

Completion covers all reachable terms (for all strategies)

$\mathcal{R}^*((sum\,4)) \subseteq \mathcal{L}(\mathcal{A}^*_{\mathcal{R},E})$ contains 10 (and intermediate computations)

Call-by-value $\leftrightarrow$ innermost strategy for TRSs

Adapted tree automata completion for innermost strategy [with Y. Salmon]

$\mathcal{R}^*_{in}((sum\,x)) \subseteq \mathcal{L}(\mathcal{A}^*_{\mathcal{R}_{in},E})$ contains no normal form (no result)

A word about built-in types

Recall this example:

Example 11 (`filter nz` on any `nat list`, results in a list without 0)

```
let if2 c t e = match c with
  | true -> t
  | false -> e;;
```

```
let nz i= match i with
  | 0 -> false
  | S(x) -> true;;
```

Programs usually use machine integers instead of Peano numbers

Lattice Tree Automata completion [with Legay, Le Gall, Murat, 2013]

LTA completion permits to seamlessly plug abstract domains in ARTMC

e.g. integer lists with no zero:

$cons(q_i, q_l) \rightarrow q_l$	$[-\infty; -1] \rightarrow q_i$
$nil \rightarrow q_l$	$[1; +\infty] \rightarrow q_i$

More demos!

- demo_filter.txt (with builtins)

  filter [$]-\infty;+\infty[$*]= [($]-\infty;-1]$ | $[1;+\infty[$)*]

- demo_map.txt (with builtins and higher order)

  map fact [$]-\infty;+\infty[$*]= [$[1;+\infty[$*]

Conclusion & Further research

- Have a terminating analysis!
- Deal with higher order functions [with Y. Salmon]
  Competitive with state of the art techniques [L. Ong, S. Ramsay, 11]
- Take evaluation strategies into account
  - call by value (*e.g.* Ocaml) $\approx$ innermost rewrite strategy [with Y. Salmon, 2012]
  - call by need (*e.g.* Haskell) $\approx$ outermost rewrite strategy $+$ sharing
  - order in pattern matching $\approx$ priority rewrite strategy
- Deal with built-in types [with T. Le Gall, A. Legay, V. Murat, 2013]
- Have a modular analysis
- User friendly way to display/define language annotations ...

Further research

- Completion dealing with the priority strategy on rewrite rules
- Completion termination theorem for: innermost+priority+'@' closures
- Define a translation from a reasonable subset of OCaml to TRS s.t.
  - Typing is preserved
  - Higher-order functions can be encoded
  - OCaml pattern matching exhaustivity $\Longrightarrow$ TRS sufficient completeness
- ... or discard the "sufficient completeness" requirement

Example 12 (sumList is not sufficiently complete)

```
let rec sumList(x,y)= (x+y)::sumList(x+y,y+1);;
let rec nth i (x::l)= if i<=0 then x else nth (i-1) l;;
let sum x= nth x (sumList(0,0));;
```

What about the presentation of the results/annotations?

A simple automaton for `[A+;B+]`

```
Automaton A0
States qA, qB, qnil, qlB, qlAB
Final States qlAB
Transitions
  A -> qA
  B -> qB
  nil -> qnil
  cons(qB,qnil) -> qlB
  cons(qB,qlB) -> qlB
  cons(qA,qlB) -> qlAB
  cons(qA, qlAB) -> qlAB
```

Any suggestion for a short textual/graphical format is welcome!

What about the presentation of the results/annotations?

Contracts [D. Xu, 2009]

```
contract rev = {l | ab l} -> {l | ba l};;
```

where `ab` and `ba` are user defined functions discriminating the ≪A then B lists≫ etc. Contracts can be dynamically or statically checked.

Liquid Types (and variants) [N. Vazou, P. Rondon, R. Jhala, 2013]

```
rev :: [a]<{\h v -> h <= v}> -> [a]<{\h v -> h >= v}
```

Liquid types are statically checked.

Two remarks and one question

+ Those techniques prove stronger properties (*e.g.* `quicksort` sorts)
− (Co)-Domains annotations are given by the user (we infer them)
- Can we define user friendly "language annotations" close to types?

Tree Automata Completion to approximate $\mathcal{R}^*(\mathcal{L}(\mathcal{A}))$

$\mathcal{R}^*(\mathcal{L}(\mathcal{A})) \subseteq \mathcal{L}(\mathcal{A}^*_{\mathcal{R},E}) \subseteq \mathcal{R}^*_E(\mathcal{L}(\mathcal{A}))$ [with V. Rusu, 2010]

Theorem 13 (Upper bound)

Given a left-linear TRS $\mathcal{R}$, a tree automaton $\mathcal{A}$ and a set of equations E, if completion terminates on $\mathcal{A}^*_{\mathcal{R},E}$ then $\mathcal{L}(\mathcal{A}^*_{\mathcal{R},E}) \supseteq \mathcal{R}^*(\mathcal{L}(\mathcal{A}))$.

Theorem 14 (Lower bound)

Given a left-linear TRS $\mathcal{R}$, a tree automaton $\mathcal{A}$ and a set of equations E, if $\mathcal{A}$ is R/E-coherent then $\mathcal{L}(\mathcal{A}^i_{\mathcal{R},E}) \subseteq \mathcal{R}^*_E(\mathcal{L}(\mathcal{A}))$ and $\mathcal{A}^i_{\mathcal{R},E}$ is R/E-coherent.

Background: Tree Automata (II)

Recognized language: $\mathcal{L}(\mathcal{A}, q) = \{s \in \mathcal{T}(\mathcal{F}) \mid s \rightarrow_{\mathcal{A}}^{*} q\}$

$\mathcal{A} = \langle \mathcal{F}, \mathcal{Q}, \mathcal{Q}_f, \Delta \rangle$
with $\mathcal{Q} = \{q_a, q_b, q_{la}, q_{lb}, q_f\}$
$\mathcal{Q}_f = \{q_f\}$ and

$$\Delta = \begin{cases} a \rightarrow q_a \\ b \rightarrow q_b \\ q_b \xrightarrow{\epsilon} q_e \\ nil \rightarrow q_{la} \\ nil \rightarrow q_{lb} \\ cons(q_a, q_{la}) \rightarrow q_{la} \\ cons(q_b, q_{lb}) \rightarrow q_{lb} \\ \\ app(q_{la}, q_{lb}) \rightarrow q_f \end{cases}$$

$\mathcal{L}(\mathcal{A}, q_{la}) = \{nil, cons(a, nil), cons(a, ...)\}$

$\mathcal{L}(\mathcal{A}, q_{lb}) = \{nil, cons(b, nil), cons(b, ...)\}$

$\mathcal{L}(\mathcal{A}, q_f) = \{app(la, lb) \mid la \in \mathcal{L}(\mathcal{A}, q_{la}) \wedge lb \in \mathcal{L}(\mathcal{A}, q_{lb})\}$

$\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}, q_f)$

Effect of epsilon transition $q_b \xrightarrow{\epsilon} q_e$ is:
$\mathcal{L}(\mathcal{A}, q_b) \subseteq \mathcal{L}(\mathcal{A}, q_e)$

Background: $\mathcal{R}$-closed tree automata

Definition 15 ($\mathcal{R}$-closed tree automaton)

Given a tree automaton $\mathcal{A}$ and a TRS $\mathcal{R}$, $\mathcal{A}$ is $\mathcal{R}$-*closed* if

$\forall l \to r \in \mathcal{R}$, $\forall q \in \mathcal{Q}$, $\forall \sigma : \mathcal{X} \mapsto \mathcal{Q}$: $\quad l\sigma \to_{\mathcal{B}}^{*} q \implies r\sigma \to_{\mathcal{B}}^{*} q$

Theorem 16 (Upper bound)

Given a left-linear TRS $\mathcal{R}$ and tree automata $\mathcal{A}, \mathcal{B}$:

$$\left.\begin{array}{c} \mathcal{L}(\mathcal{B}) \supseteq \mathcal{L}(\mathcal{A}) \\ \textit{and} \\ \mathcal{B} \textit{ is } \mathcal{R}\textit{-closed} \end{array}\right| \implies \mathcal{L}(\mathcal{B}) \supseteq \mathcal{R}^*(\mathcal{L}(\mathcal{A}))$$

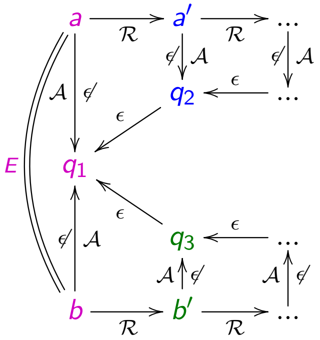
Background: $\mathcal{R}/E$-Coherent tree automata

In $\mathcal{R}/E$-Coherent tree automata we distinguish between:

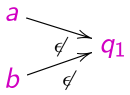
- Transitions $f(q_1, \ldots, q_n) \rightarrow q$ recognize E-equivalence classes $(\rightarrow^{\not\epsilon})$
- Epsilon transitions $q \xrightarrow{\epsilon} q'$ represent rewriting between classes

$\mathcal{R} = \{a \rightarrow a', b \rightarrow b'\}$

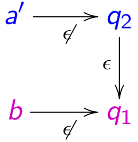
$E = \{a = b\}$



In a $\mathcal{R}/E$-coherent automaton:



$\Longrightarrow a =_E b$



$\Longrightarrow b =_E a \rightarrow_{\mathcal{R}} a'$

Completion vs Narrowing + Eq Abstraction

- – Limited to "regular properties"
- – Left-linear (conditional) TRS, no AC etc.
- – By default, no built-in support
- – Safety only (some experiments with temporal properties
- = Efficiency
- = Precision theorem for the analysis
- + Criterion for the termination of analysis
- + Counter Example Guided Abstraction Refinement (CEGAR)
- + Coq certified results
- + Finite and storable approximation of $\mathcal{R}^*(\mathcal{L})$