Reachability Analysis of Rewriting for Software Verification

Thomas Genet

IRISA

Habilitation à diriger des recherches

IRISA - 30 novembre 2009

Motivation : proving safety properties

```
①
    n := i ;
②
    while (i>1) do {
③
          n := n*(i-1) ;
④
          i := i-1 ; }
⑤
```

If $i \geq 1$ in ①

then

$n \geq 1$ in ⑤

or

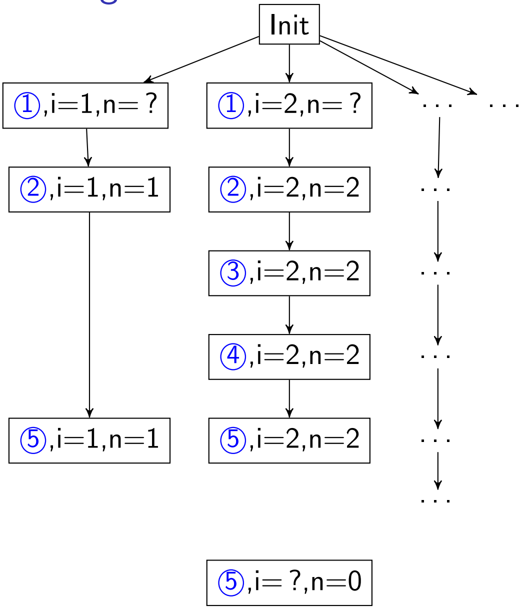
If $i \geq 1$ in ①

then

⑤ with $n = 0$ unreachable

Verification using Model-checking

```
①   {i ≥ 1}
    n := i ;
②
    while (i>1) do {
③
          n := n*(i-1) ;
④
          i := i-1 ; }
⑤   {n ≥ 1}
```



Verification using Static Analysis and Abstract Interpretation

$D = \mathbb{N}$	$D^\#$: intervals on $\mathbb{N}$
① $\{i \geq 1\}$	① $i^\# = [1; +\infty[$, $n^\# = [0; +\infty[$

```
①  {i ≥ 1}
    n := i;
②
    while (i>1) do {
③
          n := n*(i-1);
④
          i := i-1; }
⑤  {n ≥ 1}
```

Verification using Static Analysis and Abstract Interpretation

$D = \mathbb{N}$	$D^\#$: intervals on $\mathbb{N}$
① $\{i \geq 1\}$	① $i^\# = [1; +\infty[$, $n^\# = [0; +\infty[$
`n := i ;`	
②	② $i^\# = [1; +\infty[$, $n^\# = [1; +\infty[$
`while (i>1) do {`	
③	
`    n := n*(i-1) ;`	
④	
`    i := i-1 ; }`	
⑤ $\{n \geq 1\}$	

Verification using Static Analysis and Abstract Interpretation

$D = \mathbb{N}$	$D^{\#}$: intervals on $\mathbb{N}$
① $\{i \geq 1\}$	① $i^{\#} = [1; +\infty[$, $n^{\#} = [0; +\infty[$
`n := i ;`	
②	② $i^{\#} = [1; +\infty[$, $n^{\#} = [1; +\infty[$
`while (i>1) do {`	
③	③ $i^{\#} = [2; +\infty[$, $n^{\#} = [1; +\infty[$
`n := n*(i-1) ;`	
④	④ $i^{\#} = [1; +\infty[$, $n^{\#} = [1; +\infty[\, *^{\#} \, [1; +\infty[$
`i := i-1 ; }`	
⑤ $\{n \geq 1\}$	

Verification using Static Analysis and Abstract Interpretation

$D = \mathbb{N}$	$D^{\#}$: intervals on $\mathbb{N}$
① $\{i \geq 1\}$	① $i^{\#} = [1; +\infty[,\ n^{\#} = [0; +\infty[$
`n := i ;`	
②	② $i^{\#} = [1; +\infty[,\ n^{\#} = [1; +\infty[$
`while (i>1) do {`	
③	③ $i^{\#} = [2; +\infty[,\ n^{\#} = [1; +\infty[$
`    n := n*(i-1) ;`	
④	④ $i^{\#} = [1; +\infty[,\ n^{\#} = [1; +\infty[*^{\#} [1; +\infty[$
`    i := i-1 ; }`	
⑤ $\{n \geq 1\}$	⑤ $i^{\#} = [1; +\infty[,\ n^{\#} = [1; +\infty[$

Verification using a Proof Assistant

```
①  {i ≥ 1}
    n := i ;
②
    while (i>1) do {
③
         n := n*(i-1) ;
④
         i := i-1 ; }
⑤  {n ≥ 1}
```

Verification using a Proof Assistant

```
①  {$i \geq 1$}
    n := i ;
②  {$i \geq 1, n \geq 1$}
    while (i>1) do {
③          { invariant $n \geq 1$}
           n := n*(i-1) ;
④
           i := i-1 ; }
⑤  {$n \geq 1$}
```

Verification using a Proof Assistant

```
①  {$i \geq 1$}
    n := i ;
②  {$i \geq 1, n \geq 1$}
    while (i>1) do {
③          { invariant $n \geq 1$}
           n := n*(i-1) ;
④
           i := i-1 ; }
⑤  {$n \geq 1$}
```

```
FORALL (i: int):
  i >=1 IMPLIES
    (FORALL (x: int):
      x = i IMPLIES
        (FORALL (i0: int):
           FORALL (x0: int):
             x0 >= 1 IMPLIES
               i0 > 1 IMPLIES
                 (FORALL (x1: int):
                   x1 = x0 * (i0 - 1)
                   IMPLIES x1 >= 1))))
```

Verification using a Proof Assistant

```
①  ${i \geq 1}$
    n := i ;
②  ${i \geq 1, n \geq 1}$
    while (i>1) do {
③         { invariant $n \geq 1$}
          n := n*(i-1) ;
④
          i := i-1 ; }
⑤  ${n \geq 1}$
```

```
FORALL (i: int):
  i >=1 IMPLIES
    (FORALL (x: int):
      x = i IMPLIES
        (FORALL (i0: int):
           FORALL (x0: int):
             x0 >= 1 IMPLIES
               i0 > 1 IMPLIES
                 (FORALL (x1: int):
                   x1 = x0 * (i0 - 1)
                   IMPLIES x1 >= 1))))
```

```
(skosimp*)
(replace -6 1)
(lemma "both_sides_times_pos_ge1")
(inst -1 "i0!1-1" "x0!1" "1")
(grind)
```

Proving (un)reachability on infinite state systems

- Static analyzers based on abstract interpretation
- Model-checkers adapted to infinite state systems
  - Regular model-checking
  - Abstract model-checking, . . .

Proving (un)reachability on infinite state systems

- Static analyzers based on abstract interpretation
- Model-checkers adapted to infinite state systems
  - Regular model-checking
  - Abstract model-checking, . . .

+ Both are fully automatic

– When the tool fails, guiding it to finish the proof is hard

Proving (un)reachability on infinite state systems

- Static analyzers based on abstract interpretation
- Model-checkers adapted to infinite state systems
  - Regular model-checking
  - Abstract model-checking, . . .

+ Both are fully automatic

− When the tool fails, guiding it to finish the proof is hard

---

- Proof assistants

+ If a proof exists, you are likely to succeed

− . . . but you may spend weeks, months !

Proving (un)reachability on infinite state systems

- Static analyzers based on abstract interpretation
- Model-checkers adapted to infinite state systems
  - Regular model-checking
  - Abstract model-checking, ...

+ Both are fully automatic

− When the tool fails, guiding it to finish the proof is hard

---

- Proof assistants

+ If a proof exists, you are likely to succeed

− ... but you may spend weeks, months !

Is there something in between ?

Our proposition for (un)reachability analysis

A verification technique based on tree automata completion integrating

Our proposition for (un)reachability analysis

A verification technique based on tree automata completion integrating

1. A model-checking algorithm for finite (or regular) systems
2. An abstraction mechanism for infinite non regular systems
3. A way to refine, by hand, abstractions if automatic verification fails

Our proposition for (un)reachability analysis

A verification technique based on tree automata completion integrating

1. A model-checking algorithm for finite (or regular) systems
2. An abstraction mechanism for infinite non regular systems
3. A way to refine, by hand, abstractions if automatic verification fails

   and bonus :

4. In the end, the same level of confidence as with a Coq proof !

Outline

1. Term rewriting and reachability analysis
2. Regular model-checking of term rewriting systems
3. Defining abstractions for infinite non regular systems
4. Refining abstractions by hand using equations
5. Tools and applications
6. Conclusion and further work

Outline

1. Term rewriting and reachability analysis

2. Regular model-checking of term rewriting systems

3. Defining abstractions for infinite non regular systems

4. Refining abstractions by hand using equations

5. Tools and applications

6. Conclusion and further work

Term Rewriting

- Set of ranked symbols $\mathcal{F} = \{+, 0, 1\}$
- Set of variables $\mathcal{X} = \{x, y, \ldots\}$

Term Rewriting

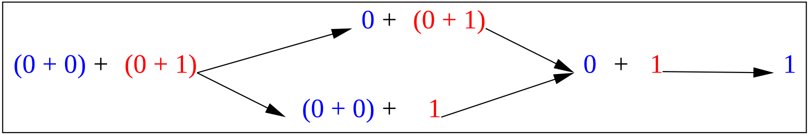
- Set of ranked symbols $\mathcal{F} = \{+, 0, 1\}$
- Set of variables $\mathcal{X} = \{x, y, \ldots\}$
- Set of ground terms $\mathcal{T}(\mathcal{F}) = \{0,\ 0+1,\ (0+0)+(0+1), \ldots\}$

Term Rewriting

- Set of ranked symbols $\mathcal{F} = \{+, 0, 1\}$
- Set of variables $\mathcal{X} = \{x, y, \ldots\}$
- Set of ground terms $\mathcal{T}(\mathcal{F}) = \{0,\ 0+1,\ (0+0)+(0+1), \ldots\}$
- Set of terms $\mathcal{T}(\mathcal{F}, \mathcal{X}) = \{x,\ 0+x,\ 1+0, \ldots\}$

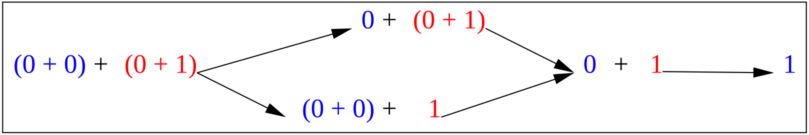
Term Rewriting

- Set of ranked symbols $\mathcal{F} = \{+, 0, 1\}$
- Set of variables $\mathcal{X} = \{x, y, \ldots\}$
- Set of ground terms $\mathcal{T}(\mathcal{F}) = \{0,\ 0 + 1,\ (0 + 0) + (0 + 1), \ldots\}$
- Set of terms $\mathcal{T}(\mathcal{F}, \mathcal{X}) = \{x,\ 0 + x,\ 1 + 0, \ldots\}$
- Rewrite rules $0 + x \rightarrow x$



Term Rewriting

- Set of ranked symbols $\mathcal{F} = \{+, 0, 1\}$
- Set of variables $\mathcal{X} = \{x, y, \dots\}$
- Set of ground terms $\mathcal{T}(\mathcal{F}) = \{0,\ 0+1,\ (0+0)+(0+1), \dots\}$
- Set of terms $\mathcal{T}(\mathcal{F},\mathcal{X}) = \{x,\ 0+x,\ 1+0, \dots\}$
- Rewrite rules $0 + x \to x$



$$\begin{tikzcd}
 & 0 + (0+1) \arrow[dr] & & \\
(0+0) + (0+1) \arrow[ur] \arrow[dr] & & 0 + 1 \arrow[r] & 1 \\
 & (0+0) + 1 \arrow[ur] & &
\end{tikzcd}$$

- Term rewriting system (TRS) = set of rewrite rules

With TRS $\mathcal{R} = \{0 + x \to x\}$:

$$0 + 1 \to_{\mathcal{R}} 1$$

$$(0+0) + (0+1) \to_{\mathcal{R}}^{*} 1$$

TRS as a formal model of programs

```
①
    n := i ;
②
    while (i>1) do {
③
        n := n*(i-1) ;
④
        i := i-1 ; }
⑤
```

$$\mathcal{F} = \{(\_,\_,\_), 0, s, +, *, \textcircled{1}, \textcircled{2}, \textcircled{3}, \textcircled{4}, \textcircled{5}\}$$
$$\mathcal{X} = \{I, N, X, Y\}$$

$$\begin{array}{ll}
(\textcircled{1}, I, N) & \to (\textcircled{2}, I, I) \\
(\textcircled{2}, s(s(I)), N) & \to (\textcircled{3}, s(s(I)), N) \\
(\textcircled{3}, s(I), N) & \to (\textcircled{4}, s(I), I * N) \\
(\textcircled{4}, s(I), N) & \to (\textcircled{2}, I, N) \\
(\textcircled{2}, 0, N) & \to (\textcircled{5}, 0, N) \\
(\textcircled{2}, s(0), N) & \to (\textcircled{5}, s(0), N) \\
 & \\
0 * X & \to 0 \\
s(X) * Y & \to Y + (X * Y) \\
\ldots &
\end{array}$$

Proving safety by (un)reachability analysis :

$$(\textcircled{1}, i, x) \not\to_{\mathcal{R}}^{*} (\textcircled{5}, y, 0) \qquad \text{with } i \geq 1, x, y \in \mathbb{N}$$

TRS as a formal model of programs

```
①
    n := i ;
②
    while (i>1) do {
③
          n := n*(i-1) ;
④
          i := i-1 ; }
⑤
```

$$\mathcal{F} = \{(\_,\_,\_), 0, s, +, *, \textcircled{1}, \textcircled{2}, \textcircled{3}, \textcircled{4}, \textcircled{5}\}$$
$$\mathcal{X} = \{I, N, X, Y\}$$

$$\begin{array}{ll}
(\textcircled{1}, I, N) & \to (\textcircled{2}, I, I) \\
(\textcircled{2}, s(s(I)), N) & \to (\textcircled{3}, s(s(I)), N) \\
(\textcircled{3}, s(I), N) & \to (\textcircled{4}, s(I), I * N) \\
(\textcircled{4}, s(I), N) & \to (\textcircled{2}, I, N) \\
(\textcircled{2}, 0, N) & \to (\textcircled{5}, 0, N) \\
(\textcircled{2}, s(0), N) & \to (\textcircled{5}, s(0), N) \\
 & \\
0 * X & \to 0 \\
s(X) * Y & \to Y + (X * Y) \\
\ldots &
\end{array}$$

Proving safety by (un)reachability analysis :

$$(\textcircled{1}, i, x) \not\to_{\mathcal{R}}^{*} (\textcircled{5}, y, 0) \qquad \text{with } i \geq 1, x, y \in \mathbb{N}$$

TRS as a formal model of programs

```
①
    n := i ;
②
    while (i>1) do {
③
          n := n*(i-1) ;
④
          i := i-1 ; }
⑤
```

$$\mathcal{F} = \{(\_,\_,\_), 0, s, +, *, \textcircled{1}, \textcircled{2}, \textcircled{3}, \textcircled{4}, \textcircled{5}\}$$
$$\mathcal{X} = \{I, N, X, Y\}$$

$$\begin{array}{ll}
(\textcircled{1}, I, N) & \rightarrow (\textcircled{2}, I, I) \\
(\textcircled{2}, s(s(I)), N) & \rightarrow (\textcircled{3}, s(s(I)), N) \\
(\textcircled{3}, s(I), N) & \rightarrow (\textcircled{4}, s(I), I * N) \\
(\textcircled{4}, s(I), N) & \rightarrow (\textcircled{2}, I, N) \\
(\textcircled{2}, 0, N) & \rightarrow (\textcircled{5}, 0, N) \\
(\textcircled{2}, s(0), N) & \rightarrow (\textcircled{5}, s(0), N) \\
 & \\
0 * X & \rightarrow 0 \\
s(X) * Y & \rightarrow Y + (X * Y) \\
\ldots &
\end{array}$$

Proving safety by (un)reachability analysis :

$$(\textcircled{1}, i, x) \not\rightarrow_{\mathcal{R}}^{*} (\textcircled{5}, y, 0) \qquad \text{with } i \geq 1, x, y \in \mathbb{N}$$

Reachability analysis of rewriting

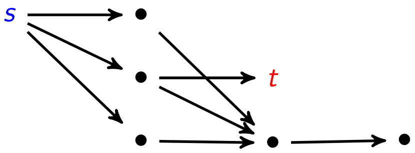
Given a TRS $\mathcal{R}$ and $s, t \in \mathcal{T}(\mathcal{F})$, is $s \rightarrow_{\mathcal{R}}^{*} t$?

- Undecidable in general (TRS are Turing-complete)

Reachability analysis of rewriting

Given a TRS $\mathcal{R}$ and $s, t \in \mathcal{T}(\mathcal{F})$, is $s \rightarrow_{\mathcal{R}}^{*} t$?

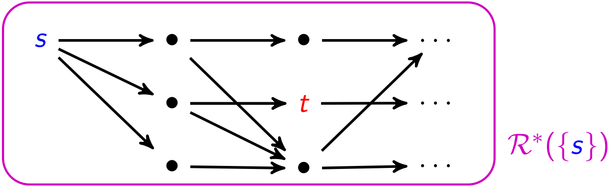
- Undecidable in general (TRS are Turing-complete)
- Decidable if $\mathcal{R}$ terminates



Reachability analysis of rewriting

Given a TRS $\mathcal{R}$ and $s, t \in \mathcal{T}(\mathcal{F})$, is $s \rightarrow_{\mathcal{R}}^{*} t$?

- Undecidable in general (TRS are Turing-complete)
- Decidable if $\mathcal{R}$ terminates



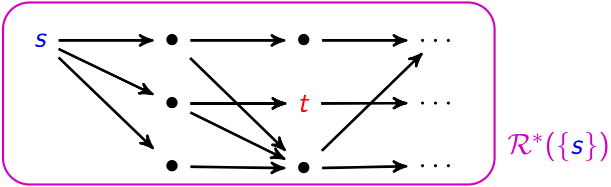
where $\mathcal{R}^*(\mathcal{L}) = \{u \mid s \in \mathcal{L} \wedge s \rightarrow_{\mathcal{R}}^{*} u\}$

- Decidable, if $\mathcal{R}^*(\{s\})$ is finite (≈ finite model-checking)

Reachability analysis of rewriting

Given a TRS $\mathcal{R}$ and $s, t \in \mathcal{T}(\mathcal{F})$, is $s \rightarrow_{\mathcal{R}}{}^* t$?

- Undecidable in general (TRS are Turing-complete)
- Decidable if $\mathcal{R}$ terminates



where $\mathcal{R}^*(\mathcal{L}) = \{u \mid s \in \mathcal{L} \wedge s \rightarrow_{\mathcal{R}}{}^* u\}$

- Decidable, if $\mathcal{R}^*(\{s\})$ is finite (≈ finite model-checking)
- Decidable, for classes of $\mathcal{R}$ such that $\mathcal{R}^*(\{s\})$ is regular (≈ regular model-checking)

Reachability analysis of rewriting (extended)

Recall that for verification, the problem we have is :

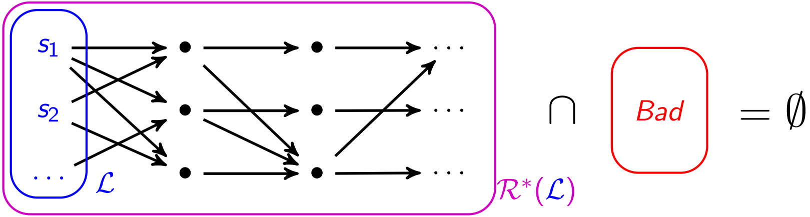
$$(①, i, x) \not\to_{\mathcal{R}}^{*} (⑤, y, 0) \qquad \text{with } i \geq 1, x, y \in \mathbb{N}$$

Reachability analysis of rewriting (extended)

Recall that for verification, the problem we have is :

$$(①, i, x) \not\to_{\mathcal{R}}^{*} (⑤, y, 0) \qquad \text{with } i \geq 1, x, y \in \mathbb{N}$$

which can be seen as :



The reachability analysis problem becomes :

$$\boxed{\mathcal{R}^*(\mathcal{L}) \cap Bad = \emptyset\,?}$$

Two applications of reachability analysis of rewriting

$$\boxed{\mathcal{R}^*(\mathcal{L}) \cap \mathit{Bad} = \emptyset\,?}$$

- Java application verification [Boichut, Genet, Jensen, Le Roux, 07]

Two applications of reachability analysis of rewriting

$$\boxed{\mathcal{R}^*(\mathcal{L}) \cap \mathit{Bad} = \emptyset\,?}$$

- Java application verification [Boichut, Genet, Jensen, Le Roux, 07]
- Cryptographic protocol verification [Genet, Klay, 00]
  - $\mathcal{L}=$ protocol initial configurations
  - $\mathcal{R}=$ | specification of protocol exchanged messages
    deduction rules of the intruder

Two applications of reachability analysis of rewriting

$$\boxed{\mathcal{R}^*(\mathcal{L}) \cap \mathit{Bad} = \emptyset\,?}$$

- Java application verification [Boichut, Genet, Jensen, Le Roux, 07]
- Cryptographic protocol verification [Genet, Klay, 00]
  - $\mathcal{L}=$ protocol initial configurations
  - $\mathcal{R}= \left|\begin{array}{l}\text{specification of protocol exchanged messages}\\ \text{deduction rules of the intruder}\end{array}\right.$
  - Properties : secrecy, authentication, freshness
  - Unbounded number of agents, protocol sessions and intruder actions
  - Verification of copy-protection on Thomson's SmartRight protocol [Genet, Tang-Talpin, Viet Triem Tong, 03]

Outline

1. Term rewriting and reachability analysis
2. Regular model-checking of term rewriting systems
3. Defining abstractions for infinite non regular systems
4. Refining abstractions by hand using equations
5. Tools and applications
6. Conclusion and further work

How to finitely represent $\mathcal{R}^*(\mathcal{L})$?

Many formalisms in the litterature :

Set constraints, Horn clauses, Tree Grammars, Tree automata, ...

How to finitely represent $\mathcal{R}^*(\mathcal{L})$?

Many formalisms in the litterature :

Set constraints, Horn clauses, Tree Grammars, Tree automata, . . .

Tree automata are well adapted (they are also based on rewriting)

- Finite Tree Automata (Regular Term Language)
- Tree Automata with constraints
- . . .

How to finitely represent $\mathcal{R}^*(\mathcal{L})$?

Many formalisms in the litterature :

Set constraints, Horn clauses, Tree Grammars, Tree automata, . . .

Tree automata are well adapted (they are also based on rewriting)

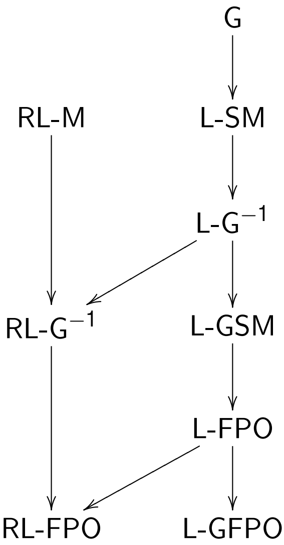
- Finite Tree Automata (Regular Term Language)
- Tree Automata with constraints
- . . .

We stick to (Non-Deterministic) Finite Tree Automata because :

We want to decide (efficiently) if $\mathcal{R}^*(\mathcal{L}) \cap Bad = \emptyset$

- The complexity of the algorithm for $\cap$ is quadratic
- The complexity of the algorithm deciding $=^? \emptyset$ is polynomial

$\mathcal{R}$ classes where $\mathcal{L}$ regular $\Rightarrow \mathcal{R}^*(\mathcal{L})$ regular



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 where $\mathcal{L}$ regular $\Rightarrow \mathcal{R}^*(\mathcal{L})$ 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 where $\mathcal{L}$ regular $\Rightarrow \mathcal{R}^*(\mathcal{L})$ regular (III)

G Ground : $s \rightarrow t$

with $s, t \in \mathcal{T}(\mathcal{F})$

$\mathcal{R}$ classes where $\mathcal{L}$ regular $\Rightarrow \mathcal{R}^*(\mathcal{L})$ regular (III)

G Ground : $s \rightarrow t$

with $s, t \in \mathcal{T}(\mathcal{F})$

RL-M Right-linear and Monadic : $s \rightarrow f(x_1, \ldots, x_n)$

with $s \in \mathcal{T}(\mathcal{F}, \mathcal{X})$

$\mathcal{R}$ classes where $\mathcal{L}$ regular $\Rightarrow \mathcal{R}^*(\mathcal{L})$ regular (III)

G Ground : $s \rightarrow t$

with $s, t \in \mathcal{T}(\mathcal{F})$

RL-M Right-linear and Monadic : $s \rightarrow f(x_1, \ldots, x_n)$

with $s \in \mathcal{T}(\mathcal{F}, \mathcal{X})$

L-SM Linear (left and right linear) Semi-Monadic :

$$s \rightarrow f(x_1, \ldots, x_n, t_1, \ldots, t_m)$$

with $s \in \mathcal{T}(\mathcal{F}, \mathcal{X}), t_1, \ldots, t_n \in \mathcal{T}(\mathcal{F})$

$\mathcal{R}$ classes where $\mathcal{L}$ regular $\Rightarrow \mathcal{R}^*(\mathcal{L})$ regular (III)

G Ground : $s \rightarrow t$

with $s, t \in \mathcal{T}(\mathcal{F})$

RL-M Right-linear and Monadic : $s \rightarrow f(x_1, \ldots, x_n)$

with $s \in \mathcal{T}(\mathcal{F}, \mathcal{X})$

L-SM Linear (left and right linear) Semi-Monadic :

$$s \rightarrow f(x_1, \ldots, x_n, t_1, \ldots, t_m)$$

with $s \in \mathcal{T}(\mathcal{F}, \mathcal{X}), t_1, \ldots, t_n \in \mathcal{T}(\mathcal{F})$

Constructor Constructor based + constraints on $\mathcal{L}$

Tree automata recognizing **regular** sets of terms

Representation of $f(s^*(a))$ by tree grammar/tree automaton

Tree automata recognizing **regular** sets of terms

Representation of $f(s^*(a))$ by tree grammar/tree automaton

$\{f(s^*(a))\}$	Tree grammar G axiom : N_1	
N_1	$:=$	$f(N_2)$
N_2	$:=$	$s(N_2)$
N_2	$:=$	a

$$N_1 \rightarrow_G^* f(s(s(a)))$$

Tree automata recognizing **regular** sets of terms

Representation of $f(s^*(a))$ by tree grammar/tree automaton

Tree grammar G			Tree automaton A		
$\{f(s^*(a))\}$		axiom : N_1	$\{f(s^*(a))\}$		final state : q_1
N_1	$:=$	$f(N_2)$	$f(q_2)$	$\rightarrow$	q_1
N_2	$:=$	$s(N_2)$	$s(q_2)$	$\rightarrow$	q_2
N_2	$:=$	a	a	$\rightarrow$	q_2
$N_1 \rightarrow_G^* f(s(s(a)))$			$f(s(s(a))) \rightarrow_A^* q_1$		

Tree automata recognizing **regular** sets of terms

Representation of $f(s^*(a))$ by tree grammar/tree automaton

Tree grammar G			Tree automaton A		
$\{f(s^*(a))\}$		axiom : N_1	$\{f(s^*(a))\}$		final state : q_1
N_1	$:=$	$f(N_2)$	$f(q_2)$	$\rightarrow$	q_1
N_2	$:=$	$s(N_2)$	$s(q_2)$	$\rightarrow$	q_2
N_2	$:=$	a	a	$\rightarrow$	q_2
$N_1 \rightarrow_G^* f(s(s(a)))$			$f(s(s(a))) \rightarrow_A^* q_1$		

$A = \langle \mathcal{F}, \mathcal{Q}, \mathcal{Q}_f, \Delta \rangle$ where

$\mathcal{Q} = \{q_1, q_2\}$, $\mathcal{Q}_f = \{q_1\}$, $\Delta = \{a \rightarrow q_2, s(q_2) \rightarrow q_2, f(q_2) \rightarrow q_1\}$

$f(s(s(a))) \rightarrow_A^* q_1$ and $q_1 \in \mathcal{Q}_f$. Here $\mathcal{L}(A) = \{f(s^*(a))\}$

A unified algorithm to build $\mathcal{R}^*(\mathcal{L})$

First step : an upper bound for $\mathcal{R}^*(\mathcal{L})$ [Genet, 98]

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

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

$\forall l \rightarrow r \in \mathcal{R}, \forall q \in \mathcal{Q}, \forall \sigma : \mathcal{X} \mapsto \mathcal{Q}$:

$$l\sigma \rightarrow_{\mathcal{B}}{}^* q \quad \Rightarrow \quad r\sigma \rightarrow_{\mathcal{B}}{}^* q$$

A unified algorithm to build $\mathcal{R}^*(\mathcal{L})$

First step : an upper bound for $\mathcal{R}^*(\mathcal{L})$ [Genet, 98]

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

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

$\forall l \rightarrow r \in \mathcal{R}, \forall q \in \mathcal{Q}, \forall \sigma : \mathcal{X} \mapsto \mathcal{Q}$:

$$l\sigma \rightarrow_{\mathcal{B}}{}^* q \quad \Rightarrow \quad r\sigma \rightarrow_{\mathcal{B}}{}^* q$$

Theorem (Upper bound)

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

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

A unified algorithm to build $\mathcal{R}^*(\mathcal{L})$ (II)

Tree automata completion algorithm

- Input : a TRS $\mathcal{R}$ and a tree automaton $\mathcal{A}$
- Output : a $\mathcal{R}$-closed automaton $\mathcal{A}^*_{\mathcal{R}}$

A unified algorithm to build $\mathcal{R}^*(\mathcal{L})$ (II)

Tree automata completion algorithm

- Input : a TRS $\mathcal{R}$ and a tree automaton $\mathcal{A}$
- Output : a $\mathcal{R}$-closed automaton $\mathcal{A}^*_{\mathcal{R}}$
- Principle : completion of $\mathcal{A}$ with new transitions until it is $\mathcal{R}$-closed

$$\begin{tikzcd}
l\sigma \arrow[r, "\mathcal{R}"'] \arrow[d, "\mathcal{A}"', "*"] & r\sigma \\
q &
\end{tikzcd}$$

A unified algorithm to build $\mathcal{R}^*(\mathcal{L})$ (II)

Tree automata completion algorithm

- Input : a TRS $\mathcal{R}$ and a tree automaton $\mathcal{A}$
- Output : a $\mathcal{R}$-closed automaton $\mathcal{A}^*_{\mathcal{R}}$
- Principle : completion of $\mathcal{A}$ with new transitions until it is $\mathcal{R}$-closed

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

A unified algorithm to build $\mathcal{R}^*(\mathcal{L})$ (II)

Tree automata completion algorithm

- Input : a TRS $\mathcal{R}$ and a tree automaton $\mathcal{A}$
- Output : a $\mathcal{R}$-closed automaton $\mathcal{A}^*_{\mathcal{R}}$
- Principle : completion of $\mathcal{A}$ with new transitions until it is $\mathcal{R}$-closed

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

Compute $\mathcal{A}^1_{\mathcal{R}}, \mathcal{A}^2_{\mathcal{R}}, \ldots$ until reaching $\mathcal{A}^*_{\mathcal{R}}$ a ($\mathcal{R}$-closed) fixpoint

A unified algorithm to build $\mathcal{R}^*(\mathcal{L})$ (II)

Tree automata completion algorithm

- Input : a TRS $\mathcal{R}$ and a tree automaton $\mathcal{A}$
- Output : a $\mathcal{R}$-closed automaton $\mathcal{A}^*_{\mathcal{R}}$
- Principle : completion of $\mathcal{A}$ with new transitions until it is $\mathcal{R}$-closed

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

Compute $\mathcal{A}^1_{\mathcal{R}}, \mathcal{A}^2_{\mathcal{R}}, \ldots$ until reaching $\mathcal{A}^*_{\mathcal{R}}$ a ($\mathcal{R}$-closed) fixpoint

$\mathcal{A}$ completed into $\mathcal{A}^*_{\mathcal{R}} \;\Rightarrow\; \mathcal{L}(\mathcal{A}^*_{\mathcal{R}}) \supseteq \mathcal{L}(\mathcal{A})$

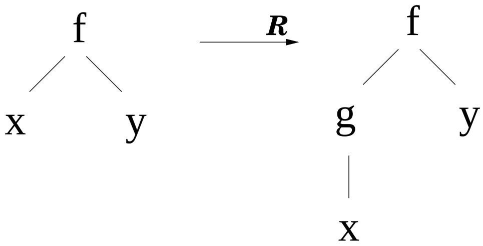
$\mathcal{A}^*_{\mathcal{R}}$ is $\mathcal{R}$-closed

$\Rightarrow \mathcal{L}(\mathcal{A}^*_{\mathcal{R}}) \supseteq \mathcal{R}^*(\mathcal{L}(\mathcal{A}))$

Tree Automata Completion may not terminate

$\mathcal{R} = \{f(x, y) \to f(g(x), y)\}$

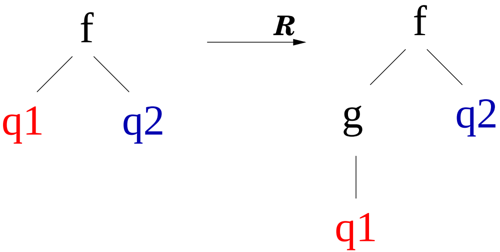
$\mathcal{A}^0$		
$f(q_1, q_2) \to q_0$ $a \to q_1$ $b \to q_2$		
$\{f(a, b)\}$		



Tree Automata Completion may not terminate

$\mathcal{R} = \{f(x,y) \rightarrow f(g(x),y)\}$

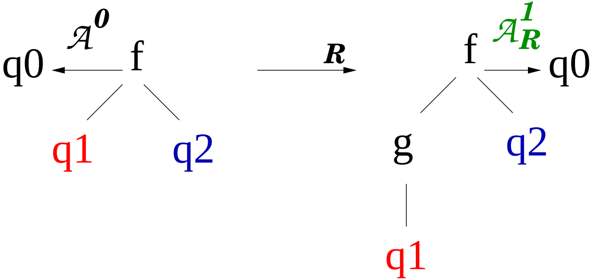
$\mathcal{A}^0$		
$f(q_1,q_2) \rightarrow q_0$ $a \rightarrow q_1$ $b \rightarrow q_2$		
$\{f(a,b)\}$		



Tree Automata Completion may not terminate

$\mathcal{R} = \{f(x,y) \rightarrow f(g(x),y)\}$

$\mathcal{A}^0$		
$f(q_1,q_2) \rightarrow q_0$ $a \rightarrow q_1$ $b \rightarrow q_2$		
$\{f(a,b)\}$		

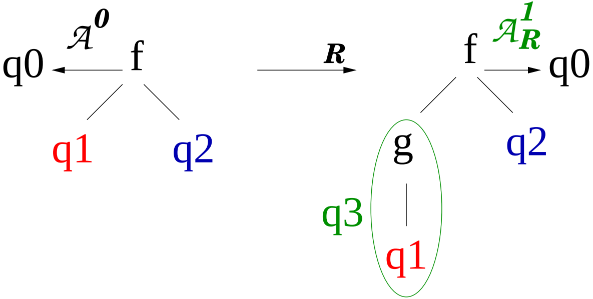


Tree Automata Completion may not terminate

$\mathcal{R} = \{f(x,y) \rightarrow f(g(x),y)\}$

$\mathcal{A}^0$	$\mathcal{A}^1_{\mathcal{R}}$	
$f(q_1,q_2) \rightarrow q_0$ $a \rightarrow q_1$ $b \rightarrow q_2$	$g(q_1) \rightarrow q_3$ $f(q_3,q_2) \rightarrow q_0$	
$\{f(a,b)\}$	$\{f(a,b), f(g(a),b)\}$	

Normalization is necessary !

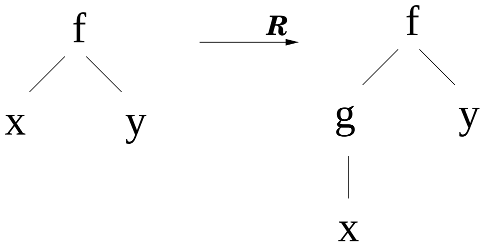


Tree Automata Completion may not terminate

$\mathcal{R} = \{f(x, y) \rightarrow f(g(x), y)\}$

$\mathcal{A}^0$	$\mathcal{A}^1_{\mathcal{R}}$	
$f(q_1, q_2) \rightarrow q_0$ $a \rightarrow q_1$ $b \rightarrow q_2$	$g(q_1) \rightarrow q_3$ $f(q_3, q_2) \rightarrow q_0$	
$\{f(a, b)\}$	$\{f(a, b), f(g(a), b)\}$	

Normalization is necessary !

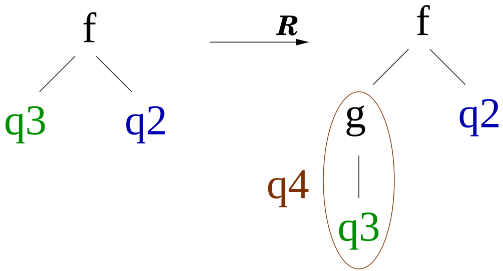


Tree Automata Completion may not terminate

$\mathcal{R} = \{f(x, y) \rightarrow f(g(x), y)\}$

$\mathcal{A}^0$	$\mathcal{A}^1_{\mathcal{R}}$	
$f(q_1, q_2) \rightarrow q_0$ $a \rightarrow q_1$ $b \rightarrow q_2$	$g(q_1) \rightarrow q_3$ $f(q_3, q_2) \rightarrow q_0$	
$\{f(a, b)\}$	$\{f(a, b), f(g(a), b)\}$	

Normalization is necessary !

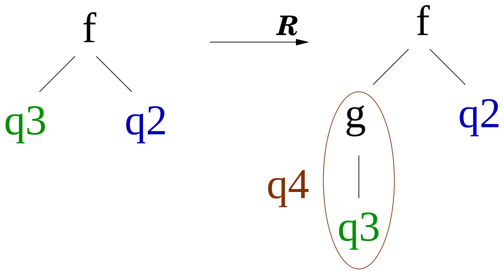


Tree Automata Completion may not terminate

$\mathcal{R} = \{f(x,y) \rightarrow f(g(x),y)\}$

$\mathcal{A}^0$	$\mathcal{A}^1_{\mathcal{R}}$	...
$f(q_1,q_2) \rightarrow q_0$ $a \rightarrow q_1$ $b \rightarrow q_2$	$g(q_1) \rightarrow q_3$ $f(q_3,q_2) \rightarrow q_0$	...
$\{f(a,b)\}$	$\{f(a,b), f(g(a),b)\}$	...

Normalization is necessary !



Exact Normalization Strategy

[Feuillade, Genet, Viet Triem Tong, 04]

Principle of Exact Normalization Strategy

Normalize new transitions added to $\mathcal{A}$ using $\mathcal{A}$ when possible, use new states otherwise.

Exact Normalization Strategy

[Feuillade, Genet, Viet Triem Tong, 04]

Principle of Exact Normalization Strategy

Normalize new transitions added to $\mathcal{A}$ using $\mathcal{A}$ when possible, use new states otherwise.

Theorem

Given a linear TRS $\mathcal{R}$ and a tree automaton $\mathcal{A}$, if tree automata completion with exact normalization strategy terminates *on $\mathcal{A}^*_{\mathcal{R}}$, then*

$$\mathcal{L}(\mathcal{A}^*_{\mathcal{R}}) = \mathcal{R}^*(\mathcal{L}(\mathcal{A}))$$

Exact Normalization Strategy

[Feuillade, Genet, Viet Triem Tong, 04]

Principle of Exact Normalization Strategy

Normalize new transitions added to $\mathcal{A}$ using $\mathcal{A}$ when possible, use new states otherwise.

Theorem

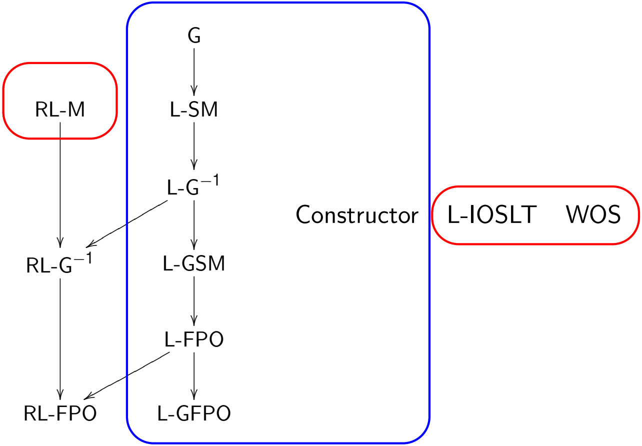
Given a linear TRS $\mathcal{R}$ and a tree automaton $\mathcal{A}$, if tree automata completion with exact normalization strategy terminates *on $\mathcal{A}^*_{\mathcal{R}}$, then*

$$\mathcal{L}(\mathcal{A}^*_{\mathcal{R}}) = \mathcal{R}^*(\mathcal{L}(\mathcal{A}))$$

Theorem

Tree automata completion with exact normalization strategy terminates for TRS in classes : **G**, **L-SM**, **L-G^{-1}**, **L-GSM**, **L-FPO** *and* **L-GFPO**.

Regular classes covered by tree automata completion



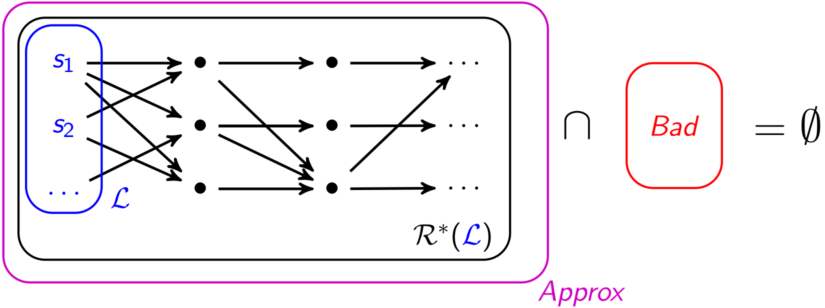
- with exact normalization strategy
- with other normalization strategies
- it also covers TRS and tree automata outside of those classes !

Outline

1. Term rewriting and reachability analysis
2. Regular model-checking of term rewriting systems
3. Defining abstractions for infinite non regular systems
4. Refining abstractions by hand using equations
5. Tools and applications
6. Conclusion and further work

Outside of the regular classes

- This is *generally* the case when the TRS models a program
- We can use over-approximations, i.e.



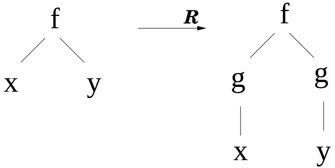
$$Approx \cap Bad = \emptyset \quad \Rightarrow \quad \mathcal{R}^*(\mathcal{L}) \cap Bad = \emptyset$$

Building approximations using normalization rules

[Genet and Viet Triem Tong 2001]

$\mathcal{R} = \{f(x, y) \to f(g(x), g(y))\}$

$\mathcal{A}^0$		
$f(q_1, q_2) \to q_0$ $a \to q_1$ $b \to q_2$		

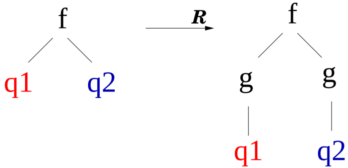


Building approximations using normalization rules

[Genet and Viet Triem Tong 2001]

$\mathcal{R} = \{f(x, y) \to f(g(x), g(y))\}$

$\mathcal{A}^0$		
$f(q_1, q_2) \to q_0$ $a \to q_1$ $b \to q_2$		

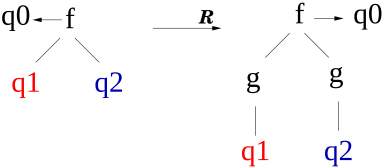


Building approximations using normalization rules

[Genet and Viet Triem Tong 2001]

$\mathcal{R} = \{f(x, y) \rightarrow f(g(x), g(y))\}$

$\mathcal{A}^0$		
$f(q_1, q_2) \rightarrow q_0$ $a \rightarrow q_1$ $b \rightarrow q_2$		

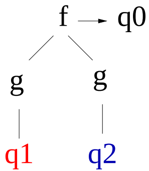


Building approximations using normalization rules

[Genet and Viet Triem Tong 2001]

$\mathcal{R} = \{f(x, y) \to f(g(x), g(y))\}$

$\mathcal{A}^0$		
$f(q_1, q_2) \to q_0$		
$a \to q_1$		
$b \to q_2$		

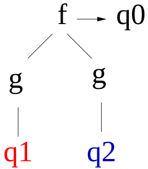


Building approximations using normalization rules

[Genet and Viet Triem Tong 2001]

$$\mathcal{R} = \{f(x, y) \to f(g(x), g(y))\}$$

$\mathcal{A}^0$		
$f(q_1, q_2) \to q_0$ $a \to q_1$ $b \to q_2$		



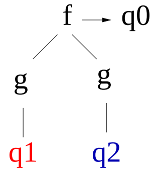
```
[f(g(q1),y) -> z] -> [g(q1) -> q1    y -> z]
```

Building approximations using normalization rules

[Genet and Viet Triem Tong 2001]

$\mathcal{R} = \{f(x, y) \rightarrow f(g(x), g(y))\}$

$\mathcal{A}^0$		
$f(q_1, q_2) \rightarrow q_0$ $a \rightarrow q_1$ $b \rightarrow q_2$		



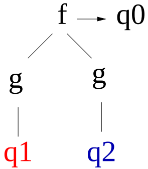
```
[f(g(q1),y) -> z] -> [g(q1) -> q1    y -> z]

[f(g(q1),g(q2)) -> z] -> [g(q1) -> q1    g(q2) -> z]
```

Building approximations using normalization rules

[Genet and Viet Triem Tong 2001]

$\mathcal{R} = \{f(x,y) \to f(g(x),g(y))\}$

$\mathcal{A}^0$		
$f(q_1,q_2) \to q_0$ $a \to q_1$ $b \to q_2$		



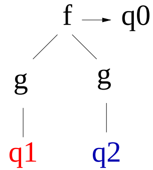
```
[f(g(q1),y) -> z] -> [g(q1) -> q1    y -> z]

[f(g(q1),g(q2)) -> q0] -> [g(q1) -> q1    g(q2) -> q0]
```

Building approximations using normalization rules

[Genet and Viet Triem Tong 2001]

$\mathcal{R} = \{f(x,y) \rightarrow f(g(x),g(y))\}$

$\mathcal{A}^0$		
$f(q_1,q_2) \rightarrow q_0$		
$a \rightarrow q_1$		
$b \rightarrow q_2$		



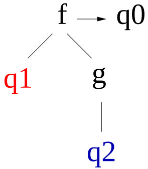
```
[f(g(q1),y) -> z] -> [g(q1) -> q1    y -> z]
[f(g(q1),g(q2)) -> q0] -> [g(q1) -> q1    g(q2) -> q0]
```

Building approximations using normalization rules

[Genet and Viet Triem Tong 2001]

$\mathcal{R} = \{f(x,y) \rightarrow f(g(x),g(y))\}$

$\mathcal{A}^0$		
$f(q_1,q_2) \rightarrow q_0$ $a \rightarrow q_1$ $b \rightarrow q_2$		



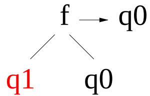
```
[f(g(q1),y) -> z] -> [g(q1) -> q1    y -> z]
[f(g(q1),g(q2)) -> q0] -> [g(q1) -> q1    g(q2) -> q0]
```

Building approximations using normalization rules

[Genet and Viet Triem Tong 2001]

$\mathcal{R} = \{f(x, y) \rightarrow f(g(x), g(y))\}$

$\mathcal{A}^0$	$\mathcal{A}^1_{\mathcal{R}}$	
$f(q_1, q_2) \rightarrow q_0$	$g(q_1) \rightarrow q_1$	
$a \rightarrow q_1$	$g(q_2) \rightarrow q_0$	
$b \rightarrow q_2$	$f(q_1, q_0) \rightarrow q_0$	



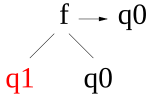
```
[f(g(q1),y) -> z] -> [g(q1) -> q1    y -> z]
[f(g(q1),g(q2)) -> q0] -> [g(q1) -> q1    g(q2) -> q0]
```

Building approximations using normalization rules

[Genet and Viet Triem Tong 2001]

$\mathcal{R} = \{f(x, y) \to f(g(x), g(y))\}$

$\mathcal{A}^0$	$\mathcal{A}^1_{\mathcal{R}}$	$\mathcal{A}^2_{\mathcal{R}}$
$f(q_1, q_2) \to q_0$	$g(q_1) \to q_1$	$g(q_0) \to q_0$
$a \to q_1$	$g(q_2) \to q_0$	
$b \to q_2$	$f(q_1, q_0) \to q_0$	



```
[f(g(q1),y) -> z] -> [g(q1) -> q1    y -> z]
[f(g(q1),g(q2)) -> q0] -> [g(q1) -> q1    g(q2) -> q0]
```

Normalization rules

The pros :

- Expressive and efficient (crypto and Java verification)
  [Genet, Tang-Talpin and Viet Triem Tong, 03]
  [Boichut, Genet, Jensen and Le Roux, 07]

Normalization rules

The pros :

- Expressive and efficient (crypto and Java verification)
  [Genet, Tang-Talpin and Viet Triem Tong, 03]
  [Boichut, Genet, Jensen and Le Roux, 07]
- Adapted for automatic synthesis (integrated in the AVISPA tool)
  [Boichut, Héam and Kouchnarenko, 04]

Normalization rules

The pros :

- Expressive and efficient (crypto and Java verification)
  [Genet, Tang-Talpin and Viet Triem Tong, 03]
  [Boichut, Genet, Jensen and Le Roux, 07]
- Adapted for automatic synthesis (integrated in the AVISPA tool)
  [Boichut, Héam and Kouchnarenko, 04]

The cons :

- Ad-hoc solution based on tree automata structure

Normalization rules

The pros :

- Expressive and efficient (crypto and Java verification)
  [Genet, Tang-Talpin and Viet Triem Tong, 03]
  [Boichut, Genet, Jensen and Le Roux, 07]
- Adapted for automatic synthesis (integrated in the AVISPA tool)
  [Boichut, Héam and Kouchnarenko, 04]

The cons :

- Ad-hoc solution based on tree automata structure
- Hard to write/read

Normalization rules

The pros :

- Expressive and efficient (crypto and Java verification)
  [Genet, Tang-Talpin and Viet Triem Tong, 03]
  [Boichut, Genet, Jensen and Le Roux, 07]
- Adapted for automatic synthesis (integrated in the AVISPA tool)
  [Boichut, Héam and Kouchnarenko, 04]

The cons :

- Ad-hoc solution based on tree automata structure
- Hard to write/read
- No formal semantics of normalization rules

Normalization rules

The pros :

- Expressive and efficient (crypto and Java verification)
[Genet, Tang-Talpin and Viet Triem Tong, 03]
[Boichut, Genet, Jensen and Le Roux, 07]
- Adapted for automatic synthesis (integrated in the AVISPA tool)
[Boichut, Héam and Kouchnarenko, 04]

The cons :

- Ad-hoc solution based on tree automata structure
- Hard to write/read
- No formal semantics of normalization rules
- Precision of approximation is difficult to estimate/compare

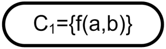
Outline

1. Term rewriting and reachability analysis
2. Regular model-checking of term rewriting systems
3. Defining abstractions for infinite non regular systems
4. Refining abstractions by hand using equations
5. Tools and applications
6. Conclusion and further work

Intuition behind equational over-approximations

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

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

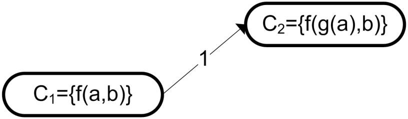


C_1={f(a,b)}

Intuition behind equational over-approximations

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

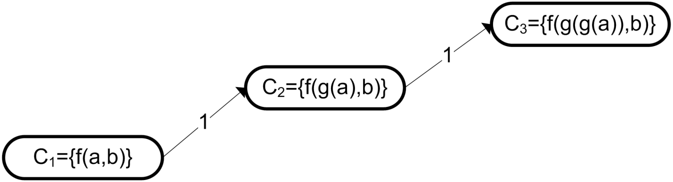
prove that $f(a,b) \not\rightarrow_{\mathcal{R}}{}^* f(a,h(g(b)))$?



Intuition behind equational over-approximations

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

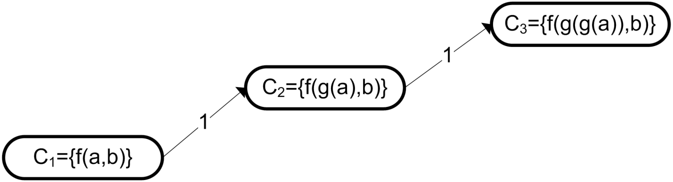
prove that $f(a,b) \not\to_{\mathcal{R}}^{*} f(a,h(g(b)))$?



Intuition behind equational over-approximations

$\mathcal{R} = \begin{cases} (1)\ f(x,y) \rightarrow f(g(x),y) \\ (2)\ f(x,y) \rightarrow f(x,h(y)) \end{cases}$ prove that $f(a,b) \not\rightarrow_{\mathcal{R}}^{*} f(a,h(g(b)))$?

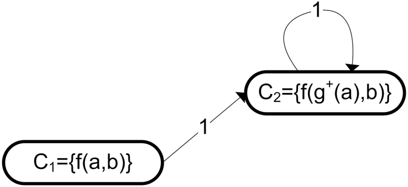
using $E = \{g(g(x)) = g(x), h(h(x)) = h(x)\}$



Intuition behind equational over-approximations

$\mathcal{R} = \begin{cases} (1)\ f(x,y) \rightarrow f(g(x),y) \\ (2)\ f(x,y) \rightarrow f(x,h(y)) \end{cases}$ prove that $f(a,b) \not\rightarrow_{\mathcal{R}}{}^{*} f(a,h(g(b)))$?

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

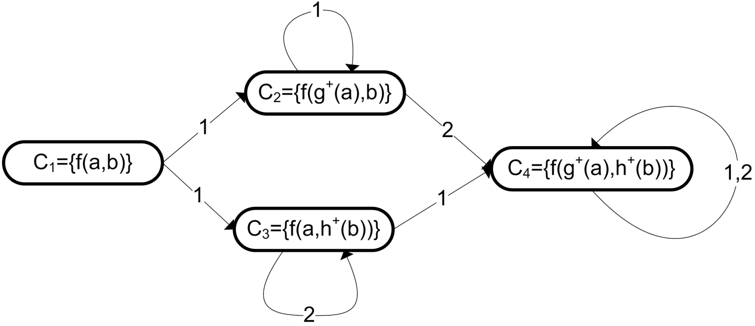


Intuition behind equational over-approximations

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

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

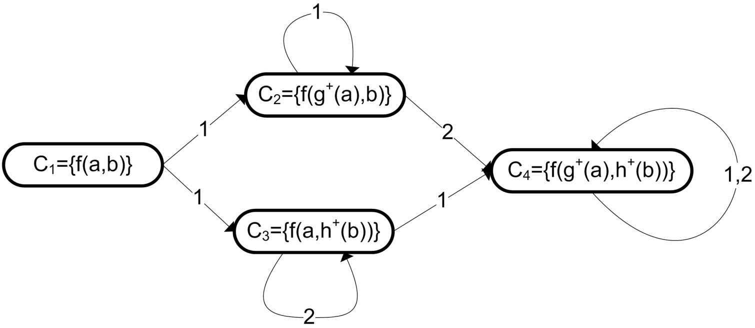
using $E = \{g(g(x)) = g(x), h(h(x)) = h(x)\}$



Intuition behind equational over-approximations

$\mathcal{R} = \begin{cases} (1)\ f(x,y) \to f(g(x),y) \\ (2)\ f(x,y) \to f(x,h(y)) \end{cases}$ prove that $f(a,b) \not\to_{\mathcal{R}}^{*} f(a,h(g(b)))$?

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



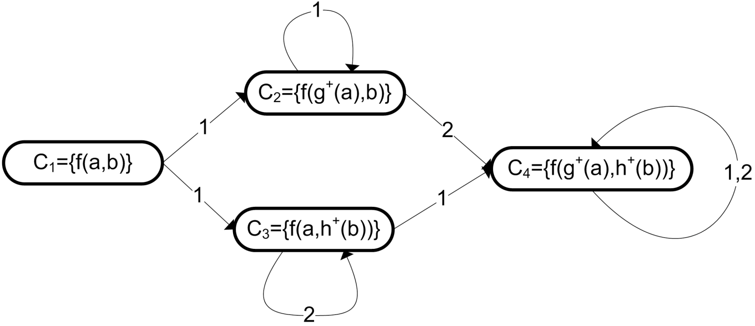
$s \to_{\mathcal{R}/E} t \quad \Leftrightarrow \quad s =_E s' \to_{\mathcal{R}} t' =_E t$

Intuition behind equational over-approximations

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

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

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

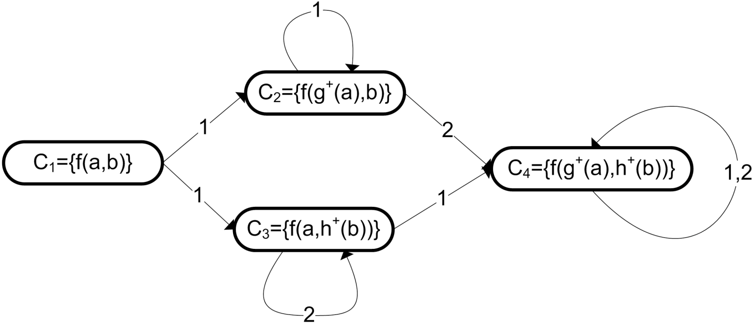


$s \rightarrow_{\mathcal{R}/E} t \quad \Leftrightarrow \quad s =_E s' \rightarrow_{\mathcal{R}} t' =_E t$ (*e.g.* $f(a,b) \rightarrow_{\mathcal{R}/E} f(g(g(g(a))),b)$)

Intuition behind equational over-approximations

$\mathcal{R} = \begin{cases} (1)\ f(x,y) \rightarrow f(g(x),y) \\ (2)\ f(x,y) \rightarrow f(x,h(y)) \end{cases}$ prove that $f(a,b) \not\rightarrow_{\mathcal{R}}^{*} f(a,h(g(b)))$?

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



$s \rightarrow_{\mathcal{R}/E} t \quad \Leftrightarrow \quad s =_E s' \rightarrow_{\mathcal{R}} t' =_E t$ (*e.g.* $f(a,b) \rightarrow_{\mathcal{R}/E} f(g(g(g(a))),b)$)

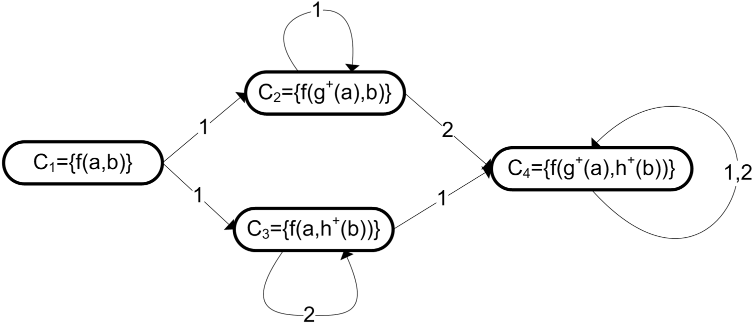
$f(a,b) \not\rightarrow_{\mathcal{R}/E}^{*} f(a,h(g(b)))$

Intuition behind equational over-approximations

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

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

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



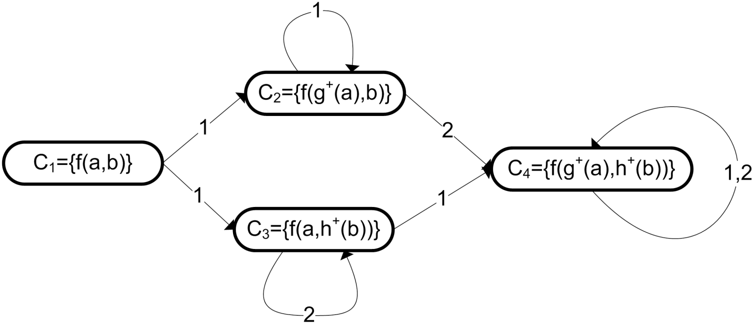
$s \to_{\mathcal{R}/E} t \quad \Leftrightarrow \quad s =_E s' \to_{\mathcal{R}} t' =_E t$ (e.g. $f(a,b) \to_{\mathcal{R}/E} f(g(g(g(a))),b)$)

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

Intuition behind equational over-approximations

$\mathcal{R} = \begin{cases} (1)\ f(x,y) \rightarrow f(g(x),y) \\ (2)\ f(x,y) \rightarrow f(x,h(y)) \end{cases}$ prove that $f(a,b) \not\rightarrow_{\mathcal{R}}{}^{*} f(a,h(g(b)))$?

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



$s \rightarrow_{\mathcal{R}/E} t \quad \Leftrightarrow \quad s =_E s' \rightarrow_{\mathcal{R}} t' =_E t$ (e.g. $f(a,b) \rightarrow_{\mathcal{R}/E} f(g(g(g(a))),b)$)

$f(a,b) \not\rightarrow^{*}_{\mathcal{R}/E} f(a,h(g(b)))$ $\Rightarrow$ $f(a,b) \not\rightarrow^{*}_{\mathcal{R}} f(a,h(g(b)))$

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

Equations for tree automata approximation

[Genet, Rusu, 09]

Simplification relation $\mathcal{A} \rightsquigarrow_E \mathcal{A}'$

Given $(u = v) \in E$ and a tree automaton $\mathcal{A}$

Equations for tree automata approximation

[Genet, Rusu, 09]

Simplification relation $\mathcal{A} \rightsquigarrow_E \mathcal{A}'$

Given $(u = v) \in E$ and a tree automaton $\mathcal{A}$

$$\begin{tikzcd} u\sigma \arrow[r, phantom, "=_E"] \arrow[d, "*"', "\mathcal{A}"] & v\sigma \arrow[d, "\mathcal{A}"', "*"] \\ q_1 & q_2 \end{tikzcd}$$

$\Rightarrow$ merging of q_1 and q_2 applied to $\mathcal{A}$

Equations for tree automata approximation

[Genet, Rusu, 09]

Simplification relation $\mathcal{A} \rightsquigarrow_E \mathcal{A}'$

Given $(u = v) \in E$ and a tree automaton $\mathcal{A}$

$$\begin{tikzcd} u\sigma \arrow[r, phantom, "=_E"] \arrow[d, "*"', "\mathcal{A}"] & v\sigma \arrow[d, "\mathcal{A}"', "*"] \\ q_1 & q_2 \end{tikzcd} \quad \Big| \quad \Rightarrow \quad \text{merging of } q_1 \text{ and } q_2 \text{ applied to } \mathcal{A}$$

denoted by $\mathcal{A} \rightsquigarrow_E \mathcal{A}'$, where $\mathcal{A}' = \mathcal{A}\{q_1 \mapsto q_2\}$

Equations for tree automata approximation

[Genet, Rusu, 09]

Simplification relation $\mathcal{A} \leadsto_E \mathcal{A}'$

Given $(u = v) \in E$ and a tree automaton $\mathcal{A}$

$$\begin{tikzcd}
u\sigma \arrow[d, "*"', "\mathcal{A}"] & =_E & v\sigma \arrow[d, "\mathcal{A}"', "*"] \\
q_1 & & q_2
\end{tikzcd}$$

$\Rightarrow$ merging of q_1 and q_2 applied to $\mathcal{A}$

denoted by $\mathcal{A} \leadsto_E \mathcal{A}'$, where $\mathcal{A}' = \mathcal{A}\{q_1 \mapsto q_2\}$

After completion step i, we propagate E on $\mathcal{A}^i_{\mathcal{R}}$ using $\leadsto_E$ up to a fixpoint

Equations for tree automata approximation

$\mathcal{R} = \{f(x, y) \rightarrow f(s(x), s(y))\}$ and $E = \{s(s(x)) = s(x)\}$

$\mathcal{A}^0$		
$f(q_a, q_b) \rightarrow q_0$ $a \rightarrow q_a$ $b \rightarrow q_b$		
$\mathcal{L}(\mathcal{A}^0) = \{f(a, b)\}$		

Equations for tree automata approximation

$\mathcal{R} = \{f(x,y) \rightarrow f(s(x),s(y))\}$ and $E = \{s(s(x)) = s(x)\}$

$\mathcal{A}^0$	$\mathcal{A}^1_{\mathcal{R}}$	
$f(q_a,q_b) \rightarrow q_0$ $a \rightarrow q_a$ $b \rightarrow q_b$	$f(q_1,q_2) \rightarrow q_0$ $s(q_a) \rightarrow q_1$ $s(q_b) \rightarrow q_2$	
$\mathcal{L}(\mathcal{A}^0) = \{f(a,b)\}$	$\mathcal{L}(\mathcal{A}^1) = \{f(a,b),$ $f(s(a),s(b))\}$	

Equations for tree automata approximation

$\mathcal{R} = \{f(x,y) \rightarrow f(s(x),s(y))\}$ and $E = \{s(s(x)) = s(x)\}$

$\mathcal{A}^0$	$\mathcal{A}^1_{\mathcal{R}}$	$\mathcal{A}^2_{\mathcal{R}}$
$f(q_a, q_b) \rightarrow q_0$	$f(q_1, q_2) \rightarrow q_0$	$f(q_3, q_4) \rightarrow q_0$
$a \rightarrow q_a$	$s(q_a) \rightarrow q_1$	$s(q_1) \rightarrow q_3$
$b \rightarrow q_b$	$s(q_b) \rightarrow q_2$	$s(q_2) \rightarrow q_4$
$\mathcal{L}(\mathcal{A}^0) = \{f(a,b)\}$	$\mathcal{L}(\mathcal{A}^1) = \{f(a,b),$	$\mathcal{L}(\mathcal{A}^2_{\mathcal{R}}) = \{f(a,b),$
	$f(s(a),s(b))\}$	$f(s(a),s(b))\}$
		$f(s(s(a)),s(s(b)))\}$

Equations for tree automata approximation

$\mathcal{R} = \{f(x,y) \rightarrow f(s(x),s(y))\}$ and $E = \{s(s(x)) = s(x)\}$

$\mathcal{A}^0$	$\mathcal{A}^1_\mathcal{R}$	$\mathcal{A}^2_\mathcal{R}$
$f(q_a,q_b) \rightarrow q_0$	$f(q_1,q_2) \rightarrow q_0$	$f(q_3,q_4) \rightarrow q_0$
$a \rightarrow q_a$	$s(q_a) \rightarrow q_1$	$s(q_1) \rightarrow q_3$
$b \rightarrow q_b$	$s(q_b) \rightarrow q_2$	$s(q_2) \rightarrow q_4$
$\mathcal{L}(\mathcal{A}^0) = \{f(a,b)\}$	$\mathcal{L}(\mathcal{A}^1) = \{f(a,b),$	$\mathcal{L}(\mathcal{A}^2_\mathcal{R}) = \{f(a,b),$
	$f(s(a),s(b))\}$	$f(s(a),s(b))\}$
		$f(s(s(a)),s(s(b)))\}$

$$\begin{tikzcd}
s(s(q_a)) \arrow[d, "*"', "\mathcal{A}^2_\mathcal{R}"] & =_E & s(q_a) \arrow[d, "*", "\mathcal{A}^2_\mathcal{R}"'] \\
q_3 & & q_1
\end{tikzcd}$$

Equations for tree automata approximation

$\mathcal{R} = \{f(x, y) \rightarrow f(s(x), s(y))\}$ and $E = \{s(s(x)) = s(x)\}$

$\mathcal{A}^0$	$\mathcal{A}^1_{\mathcal{R}}$	$\mathcal{A}^2_{\mathcal{R}}$
$f(q_a, q_b) \rightarrow q_0$ $a \rightarrow q_a$ $b \rightarrow q_b$	$f(q_1, q_2) \rightarrow q_0$ $s(q_a) \rightarrow q_1$ $s(q_b) \rightarrow q_2$	$f(q_3, q_4) \rightarrow q_0$ $s(q_1) \rightarrow q_3$ $s(q_2) \rightarrow q_4$
$\mathcal{L}(\mathcal{A}^0) = \{f(a, b)\}$	$\mathcal{L}(\mathcal{A}^1) = \{f(a, b),$ $f(s(a), s(b))\}$	$\mathcal{L}(\mathcal{A}^2_{\mathcal{R}}) = \{f(a, b),$ $f(s(a), s(b))\}$ $f(s(s(a)), s(s(b)))\}$

$$\begin{tikzcd}
s(s(q_a)) \arrow[d, "*" near start, "\mathcal{A}^2_{\mathcal{R}}"] & =_E & s(q_a) \arrow[d, "*"' near start, "\mathcal{A}^2_{\mathcal{R}}"'] \\
q_3 & & q_1
\end{tikzcd}
\qquad
\begin{tikzcd}
s(s(q_b)) \arrow[d, "*" near start, "\mathcal{A}^2_{\mathcal{R}}"] & =_E & s(q_b) \arrow[d, "*"' near start, "\mathcal{A}^2_{\mathcal{R}}"'] \\
q_4 & & q_2
\end{tikzcd}$$

Equations for tree automata approximation

$\mathcal{R} = \{f(x,y) \rightarrow f(s(x),s(y))\}$ and $E = \{s(s(x)) = s(x)\}$

$\mathcal{A}^0$	$\mathcal{A}^1_\mathcal{R}$	$\mathcal{A}^2_\mathcal{R}$
$f(q_a,q_b) \rightarrow q_0$	$f(q_1,q_2) \rightarrow q_0$	$f(q_3,q_4) \rightarrow q_0$
$a \rightarrow q_a$	$s(q_a) \rightarrow q_1$	$s(q_1) \rightarrow q_3$
$b \rightarrow q_b$	$s(q_b) \rightarrow q_2$	$s(q_2) \rightarrow q_4$
$\mathcal{L}(\mathcal{A}^0) = \{f(a,b)\}$	$\mathcal{L}(\mathcal{A}^1) = \{f(a,b),$	$\mathcal{L}(\mathcal{A}^2_\mathcal{R}) = \{f(a,b),$
	$f(s(a),s(b))\}$	$f(s(a),s(b))\}$
		$f(s(s(a)),s(s(b)))\}$

$$\begin{tikzcd}
s(s(q_a)) \arrow[r, phantom, "=_E"] \arrow[d, "*\ \mathcal{A}^2_\mathcal{R}"] & s(q_a) \arrow[d, "\mathcal{A}^2_\mathcal{R}"', "*"] \\
q_3 \arrow[r, phantom, "="] & q_1
\end{tikzcd}
\qquad
\begin{tikzcd}
s(s(q_b)) \arrow[r, phantom, "=_E"] \arrow[d, "*\ \mathcal{A}^2_\mathcal{R}"] & s(q_b) \arrow[d, "\mathcal{A}^2_\mathcal{R}"', "*"] \\
q_4 \arrow[r, phantom, "="] & q_2
\end{tikzcd}$$

Equations for tree automata approximation

$\mathcal{R} = \{f(x,y) \rightarrow f(s(x),s(y))\}$ and $E = \{s(s(x)) = s(x)\}$

$\mathcal{A}^0$	$\mathcal{A}^1_{\mathcal{R}}$	$\mathcal{A}^2_{\mathcal{R}}$
$f(q_a,q_b) \rightarrow q_0$ $a \rightarrow q_a$ $b \rightarrow q_b$	$f(q_1,q_2) \rightarrow q_0$ $s(q_a) \rightarrow q_1$ $s(q_b) \rightarrow q_2$	$f(q_a,q_b) \rightarrow q_0$ $s(q_1) \rightarrow q_1$ $s(q_2) \rightarrow q_2$
$\mathcal{L}(\mathcal{A}^0) = \{f(a,b)\}$	$\mathcal{L}(\mathcal{A}^1) = \{f(a,b),$ $f(s(a),s(b))\}$	$\mathcal{L}(\mathcal{A}^2_{\mathcal{R}}) = \{f(s^*(a),s^*(b))\}$

$$\begin{tikzcd}
s(s(q_a)) \arrow[r, phantom, "=_E"] \arrow[d, "*"] & s(q_a) \arrow[d, "*"] & s(s(q_b)) \arrow[r, phantom, "=_E"] \arrow[d, "*"] & s(q_b) \arrow[d, "*"] \\
q_3 \arrow[r, phantom, "\mathcal{A}^2_{\mathcal{R}} \quad = \quad \mathcal{A}^2_{\mathcal{R}}"] & q_1 & q_4 \arrow[r, phantom, "\mathcal{A}^2_{\mathcal{R}} \quad = \quad \mathcal{A}^2_{\mathcal{R}}"] & q_2
\end{tikzcd}$$

Properties of $\rightsquigarrow_E$

The simplification relation $\rightsquigarrow_E$ enjoys the following properties

- If $\mathcal{A} \rightsquigarrow_E \mathcal{A}'$ then $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{A}')$

Properties of $\leadsto_E$

The simplification relation $\leadsto_E$ enjoys the following properties

- If $\mathcal{A} \leadsto_E \mathcal{A}'$ then $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{A}')$
- $\leadsto_E$ terminates

Properties of $\rightsquigarrow_E$

The simplification relation $\rightsquigarrow_E$ enjoys the following properties

- If $\mathcal{A} \rightsquigarrow_E \mathcal{A}'$ then $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{A}')$
- $\rightsquigarrow_E$ terminates
- $\rightsquigarrow_E$ is locally confluent, modulo isomorphism

Properties of $\rightsquigarrow_E$

The simplification relation $\rightsquigarrow_E$ enjoys the following properties

- If $\mathcal{A} \rightsquigarrow_E \mathcal{A}'$ then $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{A}')$
- $\rightsquigarrow_E$ terminates
- $\rightsquigarrow_E$ is locally confluent, modulo isomorphism
- Normal forms of $\rightsquigarrow_E$ are unique, modulo isomorphism

Properties of $\rightsquigarrow_E$

The simplification relation $\rightsquigarrow_E$ enjoys the following properties

- If $\mathcal{A} \rightsquigarrow_E \mathcal{A}'$ then $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{A}')$
- $\rightsquigarrow_E$ terminates
- $\rightsquigarrow_E$ is locally confluent, modulo isomorphism
- Normal forms of $\rightsquigarrow_E$ are unique, modulo isomorphism

$\Rightarrow$ equations of E can be used in any order for $\rightsquigarrow_E^!$

New completion algorithm : from $\mathcal{A}^{i}_{\mathcal{R},E}$ to $\mathcal{A}^{i+1}_{\mathcal{R},E}$

i-th Completion step

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

- Normalize $r\sigma \rightarrow q'$ using exact norm. strat. or new states

New completion algorithm : from $\mathcal{A}^i_{\mathcal{R},E}$ to $\mathcal{A}^{i+1}_{\mathcal{R},E}$

i-th Completion step

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

- Normalize $r\sigma \rightarrow q'$ using exact norm. strat. or new states

Simplification

- Find instances of an equation $u = v$ of E in $\mathcal{A}^{i+1}_{\mathcal{R}}$

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

New completion algorithm : from $\mathcal{A}^{i}_{\mathcal{R},E}$ to $\mathcal{A}^{i+1}_{\mathcal{R},E}$

i-th Completion step

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

- Normalize $r\sigma \rightarrow q'$ using exact norm. strat. or new states

Simplification

- Find instances of an equation $u = v$ of E in $\mathcal{A}^{i+1}_{\mathcal{R}}$

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

- Rename q_2 by q_1 in $\mathcal{A}^{i+1}_{\mathcal{R}}$
- Repeat until a fixpoint is reached

Theorems

Theorem (Upper bound)

Let $\mathcal{R}$ be a left-linear TRS, $\mathcal{A}$ be a tree automaton and E be a set of linear equations. If completion terminates on $\mathcal{A}^*_{\mathcal{R},E}$ then

$$\mathcal{L}(\mathcal{A}^*_{\mathcal{R},E}) \supseteq \mathcal{R}^*(\mathcal{L}(\mathcal{A}))$$

Theorems

Theorem (Upper bound)

Let $\mathcal{R}$ be a left-linear TRS, $\mathcal{A}$ be a tree automaton and E be a set of linear equations. 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 (Lower bound)

Let $\mathcal{R}$ be a left-linear TRS, E a set of linear equations and $\mathcal{A}$ a $\mathcal{R}/E$-coherent tree automaton. For any $i \in \mathbb{N}$:

$$\mathcal{R}^*_{/E}(\mathcal{L}(\mathcal{A})) \supseteq \mathcal{L}(\mathcal{A}^i_{\mathcal{R},E})$$

and $\mathcal{A}^i_{\mathcal{R},E}$ is $\mathcal{R}/E$-coherent.

Outline

1. Term rewriting and reachability analysis
2. Regular model-checking of term rewriting systems
3. Defining abstractions for infinite non regular systems
4. Refining abstractions by hand using equations
5. Tools and applications
6. Conclusion and further work

The **Timbuk** library

[Genet, Viet Triem Tong, Boichut, Boyer]
(Around 13000 lines of Ocaml)

Timbuk provides

- Tree automata implementation with $\cap, \cup, =^? \emptyset, \subseteq, \ldots$

The **Timbuk** library

[Genet, Viet Triem Tong, Boichut, Boyer]
(Around 13000 lines of Ocaml)

Timbuk provides

- Tree automata implementation with $\cap, \cup, =^? \emptyset, \subseteq, \ldots$
- Tree automata completion
  - Exact computation of (covered) regular classes
  - Approximations with normalization rules/equations

The **Timbuk** library

[Genet, Viet Triem Tong, Boichut, Boyer]
(Around 13000 lines of Ocaml)

Timbuk provides

- Tree automata implementation with $\cap, \cup, =^? \emptyset, \subseteq, \ldots$
- Tree automata completion
  - Exact computation of (covered) regular classes
  - Approximations with normalization rules/equations
- Tree automata completion checker

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

$$\texttt{checker}(\mathcal{A}, \mathcal{R}, \mathcal{B}) = \texttt{true} \quad \Rightarrow \quad \mathcal{L}(\mathcal{B}) \supseteq \mathcal{R}^*(\mathcal{L}(\mathcal{A}))$$

The **Timbuk** library

[Genet, Viet Triem Tong, Boichut, Boyer]
(Around 13000 lines of Ocaml)

Timbuk provides

- Tree automata implementation with $\cap, \cup, =^? \emptyset, \subseteq, \ldots$
- Tree automata completion
  - Exact computation of (covered) regular classes
  - Approximations with normalization rules/equations

- Tree automata completion `checker`

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

$$\texttt{checker}(\mathcal{A}, \mathcal{R}, \mathcal{B}) = \texttt{true} \quad \Rightarrow \quad \mathcal{L}(\mathcal{B}) \supseteq \mathcal{R}^*(\mathcal{L}(\mathcal{A}))$$

`checker` extracted from a Coq spec. [Boyer, Genet, Jensen, 08]

Applications : Java bytecode verification

[Boichut, Genet, Jensen, Le Roux, 07]

$$\boxed{\mathcal{R}^*(\mathcal{L}) \cap \mathit{Bad} = \emptyset}$$

- $\mathcal{R}= \left|\begin{array}{l}\text{A Java byte code program } P\\ \text{Java Virtual Machine (JVM) semantics}\end{array}\right.$
- $\mathcal{L}=$ Java Virtual Machine (JVM) initial state

Applications : Java bytecode verification

[Boichut, Genet, Jensen, Le Roux, 07]

$$\boxed{\mathcal{R}^*(\mathcal{L}) \cap \mathit{Bad} = \emptyset}$$

- $\mathcal{R}=\left|\begin{array}{l}\text{A Java byte code program } P\\ \text{Java Virtual Machine (JVM) semantics}\end{array}\right.$
- $\mathcal{L}=$ Java Virtual Machine (JVM) initial state
- $\mathcal{R}^*(\mathcal{L})=$ all JVM states reachable while executing P
- $\mathit{Bad}=$ set of forbidden states (*e.g.* bad control flow, data races, etc.)

Encoding JVM semantics and bytecode into rewriting

Copster tool [Barré, Hubert, Le Roux, Genet]

- Translates `.class` into a *left-linear* TRS

Encoding JVM semantics and bytecode into rewriting

Copster tool [Barré, Hubert, Le Roux, Genet]

- Translates `.class` into a *left-linear* TRS
- **Copster** covers the following Java aspects :
  - Class and inheritance
  - Object allocation, initialization, access and modification of fields
  - Virtual method invocation
  - Integer, boolean, characters and string types
  - Basic arithmetic and comparisons
  - Basic standard library methods (strings, I/O)
  - Basic thread operations (creation, synchronization, join)

An example of verification performed on a Java program

```
class T1 extends java.lang.Thread{
  private int l;

  public T1(int l){this.l=l;}

  public void run(){
    while (true){
      synchronized(Top.lock){
         System.out.println(Top.f);
         Top.f=l;
         System.out.println(Top.f);
         Top.f=0;
      }}}}
```

```
class Top{
  public static Object lock;
  public static int f;
  public static void main(String[]
    int i=1;
    lock = new Object();
    Top.f=0;
    while (i<=2){
      T1 t1 = new T1(i++);
      t1.start();
    }}}
```

An example of verification performed on a Java program

```
class T1 extends java.lang.Thread{
  private int l;

  public T1(int l){this.l=l;}

  public void run(){
    while (true){
      synchronized(Top.lock){
         System.out.println(Top.f);
         Top.f=l;
         System.out.println(Top.f);
         Top.f=0;
      }}}}
```

```
class Top{
  public static Object lock;
  public static int f;
  public static void main(String[]
    int i=1;
    lock = new Object();
    Top.f=0;
    while (i<=2){
      T1 t1 = new T1(i++);
      t1.start();
    }}}
```

- Because of thread synchronization with Java locks (semaphores) : infinite sequences of outputs should be of the form $0, 1, 0, 2, 0, 1, 0, \ldots$

An example of verification performed on a Java program

```
class T1 extends java.lang.Thread{
  private int l;

  public T1(int l){this.l=l;}

  public void run(){
    while (true){
      synchronized(Top.lock){
         System.out.println(Top.f);
         Top.f=l;
         System.out.println(Top.f);
         Top.f=0;
    }}}}
```

```
class Top{
  public static Object lock;
  public static int f;
  public static void main(String[]
    int i=1;
    lock = new Object();
    Top.f=0;
    while (i<=2){
      T1 t1 = new T1(i++);
      t1.start();
    }}}
```

- Because of thread synchronization with Java locks (semaphores) : infinite sequences of outputs should be of the form $0, 1, 0, 2, 0, 1, 0, \ldots$
- Subsequences of the form $\ldots, i, i, \ldots$ with $i \geq 1$ should not occur

An example of verification performed on a Java program

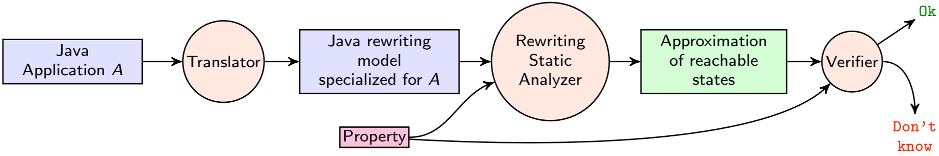
```
class T1 extends java.lang.Thread{
  private int l;

  public T1(int l){this.l=l;}

  public void run(){
    while (true){
      synchronized(Top.lock){
         System.out.println(Top.f);
         Top.f=l;
         System.out.println(Top.f);
         Top.f=0;
    }}}}
```

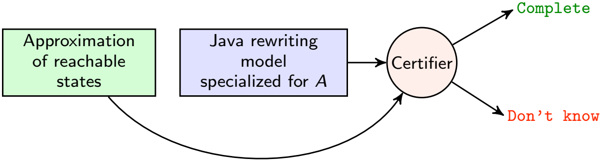
```
class Top{
  public static Object lock;
  public static int f;
  public static void main(String[]
    int i=1;
    lock = new Object();
    Top.f=0;
    while (i<=2){
      T1 t1 = new T1(i++);
      t1.start();
    }}}
```

- Because of thread synchronization with Java locks (semaphores) : infinite sequences of outputs should be of the form $0, 1, 0, 2, 0, 1, 0, \ldots$
- Subsequences of the form $\ldots, i, i, \ldots$ with $i \geq 1$ should not occur
- One equation is enough : `outstack(x,outstack(y,z))=z`

The RAVAJ Java verification chain

- RAVAJ is an ANR Project between
  LORIA (Nancy), LIFC (Besançon), France Telecom and IRISA
- Certified reachability analysis chain for Java bytecode programs


Java Application A
Translator
Java rewriting model specialized for A
Rewriting Static Analyzer
Approximation of reachable states
Verifier
Ok
Don't know
Property


Approximation of reachable states
Java rewriting model specialized for A
Certifier
Complete
Don't know

Outline

1. Term rewriting and reachability analysis
2. Regular model-checking of term rewriting systems
3. Defining abstractions for infinite non regular systems
4. Refining abstractions by hand using equations
5. Tools and applications
6. Conclusion and further work

Comparison with Regular (Abstract) Tree Model-Checking

- Comparison between Tree Tranducers and TRS is difficult

Comparison with Regular (Abstract) Tree Model-Checking

- Comparison between Tree Tranducers and TRS is difficult

– $\mathcal{R}(\mathcal{L})$ can be computed with TT, not easy with TRS

Comparison with Regular (Abstract) Tree Model-Checking

- Comparison between Tree Tranducers and TRS is difficult

– $\mathcal{R}(\mathcal{L})$ can be computed with TT, not easy with TRS

– Verification of temporal properties more difficult in our case

Comparison with Regular (Abstract) Tree Model-Checking

- Comparison between Tree Tranducers and TRS is difficult

– $\mathcal{R}(\mathcal{L})$ can be computed with TT, not easy with TRS

– Verification of temporal properties more difficult in our case

– Counterexample generation and refinement better defined with TT

Comparison with Regular (Abstract) Tree Model-Checking

- Comparison between Tree Tranducers and TRS is difficult

– $\mathcal{R}(\mathcal{L})$ can be computed with TT, not easy with TRS

– Verification of temporal properties more difficult in our case

– Counterexample generation and refinement better defined with TT

+ Translation of an operationnal semantics into a TRS is easier

Comparison with Regular (Abstract) Tree Model-Checking

- Comparison between Tree Tranducers and TRS is difficult

– $\mathcal{R}(\mathcal{L})$ can be computed with TT, not easy with TRS

– Verification of temporal properties more difficult in our case

– Counterexample generation and refinement better defined with TT

+ Translation of an operationnal semantics into a TRS is easier

+ Precision result w.r.t approximation (i.e. w.r.t. $\mathcal{R}/E$)

Comparison with Regular (Abstract) Tree Model-Checking

- Comparison between Tree Tranducers and TRS is difficult

– $\mathcal{R}(\mathcal{L})$ can be computed with TT, not easy with TRS

– Verification of temporal properties more difficult in our case

– Counterexample generation and refinement better defined with TT

+ Translation of an operationnal semantics into a TRS is easier

+ Precision result w.r.t approximation (i.e. w.r.t. $\mathcal{R}/E$)

≈ Equations could be used on TT, and predicate abstraction on TRS

Comparison with Static Analysis and Abstract Interpretation

– Regular tree languages are only *one* particular abstract domain !

Comparison with Static Analysis and Abstract Interpretation

– Regular tree languages are only *one* particular abstract domain !

+ Other domains may be encoded into this one (*e.g.* for *k*-CFA)

Comparison with Static Analysis and Abstract Interpretation

– Regular tree languages are only *one* particular abstract domain !

+ Other domains may be encoded into this one (*e.g.* for *k*-CFA)

– Abstract interpretation can be optimized w.r.t. the domain

Comparison with Static Analysis and Abstract Interpretation

− Regular tree languages are only *one* particular abstract domain !

+ Other domains may be encoded into this one (*e.g.* for *k*-CFA)

− Abstract interpretation can be optimized w.r.t. the domain

+ Refinement of approximation (automatic/by hand)

Comparison with Static Analysis and Abstract Interpretation

− Regular tree languages are only *one* particular abstract domain !

+ Other domains may be encoded into this one (*e.g.* for *k*-CFA)

− Abstract interpretation can be optimized w.r.t. the domain

+ Refinement of approximation (automatic/by hand)

+ A *unique* checker for certifying all approximations

Comparison with other verification techniques

- Classes of $\mathcal{R}$ for which $\mathcal{R}^*(\mathcal{L})$ is regular
  - − only left and right linear TRS
  - − only free (*e.g.* no AC) ranked (*e.g.* no hedge) TRS
  - + A *common* algorithm and an optimized tool for all the covered classes

Comparison with other verification techniques

- Classes of $\mathcal{R}$ for which $\mathcal{R}^*(\mathcal{L})$ is regular
  - − only left and right linear TRS
  - − only free (*e.g.* no AC) ranked (*e.g.* no hedge) TRS
  - + A *common* algorithm and an optimized tool for all the covered classes

- Others equational abstractions
  - − Completion is more expensive than a pure rewriting approach
  - + Even in the finite case, automata can be faster than tabling rewriting
  - − Generate equations automatically (in some cases)
  - + In practice, strong restrictions on equations (syntactical/coherence)

Comparison with other verification techniques

- Classes of $\mathcal{R}$ for which $\mathcal{R}^*(\mathcal{L})$ is regular
  - – only left and right linear TRS
  - – only free (*e.g.* no AC) ranked (*e.g.* no hedge) TRS
  - + A *common* algorithm and an optimized tool for all the covered classes

- Others equational abstractions
  - – Completion is more expensive than a pure rewriting approach
  - + Even in the finite case, automata can be faster than tabling rewriting
  - – Generate equations automatically (in some cases)
  - + In practice, strong restrictions on equations (syntactical/coherence)

- Other techniques based on rewriting
  - – Limited to ≪ regular ≫ properties (*e.g.* no induction !)
  - + Simpler properties ⇒ needs less interaction
  - + No need for termination or confluence of the TRS

To sum-up

From the initial (theoretical) idea of tree automata completion, we have shown that this technique

1. covers many regular classes of the litterature
2. deals with automatic/guided approximations
3. is feasible in practice
4. scales up to verify real software
5. can be certified using an external proof assistant

Further Research

- Now : extend the verification capabilities of tree automata completion

  $\rightsquigarrow$ lift-up to temporal properties

  [Boyer, Genet, 09]

Further Research

- Now : extend the verification capabilities of tree automata completion

  $\leadsto$ lift-up to temporal properties

  [Boyer, Genet, 09]

- Next year : improve the completion-based verification framework
  - Counter-example extraction
  - Automatic refinement of equational approximations

Further Research

- Now : extend the verification capabilities of tree automata completion

  $\rightsquigarrow$ lift-up to temporal properties

  [Boyer, Genet, 09]

- Next year : improve the completion-based verification framework
  - Counter-example extraction
  - Automatic refinement of equational approximations

Further Research

- Now : extend the verification capabilities of tree automata completion

  $\rightsquigarrow$ lift-up to temporal properties

  [Boyer, Genet, 09]

- Next year : improve the completion-based verification framework
  - Counter-example extraction
  - Automatic refinement of equational approximations
  - Vizualization of completion divergence

Further Research

- Now : extend the verification capabilities of tree automata completion

  $\rightsquigarrow$ lift-up to temporal properties

  [Boyer, Genet, 09]

- Next year : improve the completion-based verification framework
  - Counter-example extraction
  - Automatic refinement of equational approximations
  - Vizualization of completion divergence
  - Equation inference

Further Research

- Now : extend the verification capabilities of tree automata completion

  $\rightsquigarrow$ lift-up to temporal properties

  [Boyer, Genet, 09]

- Next year : improve the completion-based verification framework
  - Counter-example extraction
  - Automatic refinement of equational approximations
  - Vizualization of completion divergence
  - Equation inference

- Within 3 years : certification of distant computation (a.k.a. result certification)

Further Research (II)

- Extend (word) lattice automata to trees

  with T. Legall

- Improve automatic approximations for crypto. protocols

  with Y. Boichut

- Other applications of $\mathcal{R}^*(\mathcal{L})$
  - Checking transformations of SQL query
  - Checking transformations of UML model
  - Javascript programs verification

Further Research (II)

Since the new completion algorithm is based on :

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

instead of

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

from the ϵ-graph we can obtain the $\mathcal{R}/E$-rewriting graph

Further Research (II)

Since the new completion algorithm is based on :

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

instead of

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

from the ϵ-graph we can obtain the $\mathcal{R}/E$-rewriting graph

$$\mathcal{R} = \{f(x,y) \to f(g(x),y),$$
$$f(x,y) \to f(x,h(y))\}$$

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

$$\mathcal{L} = \{f(a,b)\}$$

Further Research (II)

Since the new completion algorithm is based on :

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

instead of

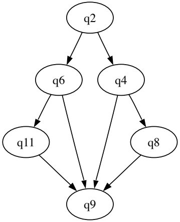
$$\begin{tikzcd}
l\sigma \arrow[r, "\mathcal{R}"'] \arrow[d, "\mathcal{A}^i_{\mathcal{R}}"', "*"] & r\sigma \arrow[dl, bend left, "*"', "\mathcal{A}^{i+1}_{\mathcal{R}}"] \\
q &
\end{tikzcd}$$

from the ϵ-graph we can obtain the $\mathcal{R}/E$-rewriting graph

$\mathcal{R} = \{f(x,y) \rightarrow f(g(x),y),$
$f(x,y) \rightarrow f(x,h(y))\}$

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

$\mathcal{L} = \{f(a,b)\}$



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

In the tree automata we distinguish between

- Transitions $f(q_1, \ldots, q_n) \rightarrow q$ recognizing ≪ equivalence classes ≫
- Epsilon transitions $q \xrightarrow{\epsilon} q'$ representing rewriting between classes

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

In the tree automata we distinguish between

- Transitions $f(q_1, \ldots, q_n) \rightarrow q$ recognizing ≪ equivalence classes ≫
- Epsilon transitions $q \xrightarrow{\epsilon} q'$ representing rewriting between classes

$\mathcal{R} = \{s \rightarrow t, u \rightarrow v\}$
$E = \{s = u\}$

New completion

$$\begin{tikzcd}
s \arrow[r, "\mathcal{R}"'] \arrow[d, equal, bend right, "E"'] & t \\
u \arrow[r, "\mathcal{R}"'] & v
\end{tikzcd}$$

Old completion

$$\begin{tikzcd}
s \arrow[r, "\mathcal{R}"'] \arrow[d, equal, bend right, "E"'] & t \\
u \arrow[r, "\mathcal{R}"'] & v
\end{tikzcd}$$

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

In the tree automata we distinguish between

- Transitions $f(q_1, \ldots, q_n) \rightarrow q$ recognizing « equivalence classes »
- Epsilon transitions $q \xrightarrow{\epsilon} q'$ representing rewriting between classes

$\mathcal{R} = \{s \rightarrow t, u \rightarrow v\}$
$E = \{s = u\}$

New completion

$$\begin{tikzcd}
s \arrow[r, "\mathcal{R}"'] \arrow[dd, "\mathcal{A}"', "\not\epsilon"] \arrow[dddd, dash, double, bend right, "E"'] & t \arrow[d, "\not\epsilon"', "\mathcal{A}"] \\
 & q_2 \\
q_1 & \\
 & q_3 \\
u \arrow[uu, "\not\epsilon", "\mathcal{A}"'] \arrow[r, "\mathcal{R}"'] & v \arrow[u, "\mathcal{A}", "\not\epsilon"']
\end{tikzcd}$$

Old completion

$$\begin{tikzcd}
s \arrow[r, "\mathcal{R}"'] \arrow[dd, "\mathcal{A}"', "\not\epsilon"] \arrow[dddd, dash, double, bend right, "E"'] & t \\
 & \\
q_1 & \\
 & \\
u \arrow[uu, "\not\epsilon", "\mathcal{A}"'] \arrow[r, "\mathcal{R}"'] & v
\end{tikzcd}$$

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

In the tree automata we distinguish between

- Transitions $f(q_1, \ldots, q_n) \rightarrow q$ recognizing ≪ equivalence classes ≫
- Epsilon transitions $q \xrightarrow{\epsilon} q'$ representing rewriting between classes

$\mathcal{R} = \{s \rightarrow t, u \rightarrow v\}$
$E = \{s = u\}$

New completion

$$\begin{tikzcd}
s \arrow[r, "\mathcal{R}"'] \arrow[dd, "\mathcal{A}"', "\not\epsilon"] \arrow[dddd, equal, bend right=40, "E"'] & t \arrow[d, "\not\epsilon"', "\mathcal{A}"] \\
 & q_2 \arrow[dl, "\epsilon"'] \\
q_1 & \\
 & q_3 \arrow[ul, "\epsilon"] \\
u \arrow[uu, "\not\epsilon", "\mathcal{A}"'] \arrow[r, "\mathcal{R}"'] & v \arrow[u, "\mathcal{A}", "\not\epsilon"']
\end{tikzcd}$$

Old completion

$$\begin{tikzcd}
s \arrow[r, "\mathcal{R}"'] \arrow[d, "\mathcal{A}"', "\not\epsilon"] \arrow[dd, equal, bend right=50, "E"'] & t \arrow[dl, "\not\epsilon"', "\mathcal{A}"] \\
q_1 & \\
u \arrow[u, "\not\epsilon", "\mathcal{A}"'] \arrow[r, "\mathcal{R}"'] & v \arrow[ul, "\not\epsilon"', "\mathcal{A}"]
\end{tikzcd}$$

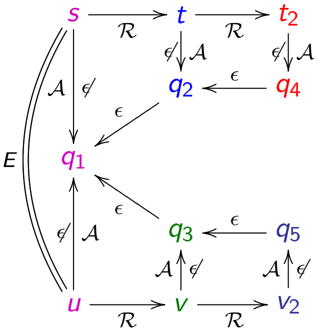
$\mathcal{R}/E$-Coherent tree automata

In the tree automata we distinguish between

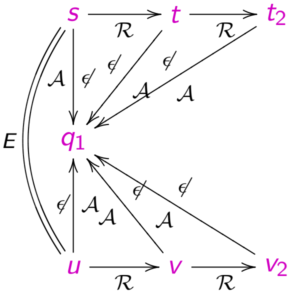
- Transitions $f(q_1, \ldots, q_n) \rightarrow q$ recognizing ≪ equivalence classes ≫
- Epsilon transitions $q \xrightarrow{\epsilon} q'$ representing rewriting between classes

$\mathcal{R} = \{s \rightarrow t, u \rightarrow v\}$
$E = \{s = u\}$

New completion



Old completion



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

In the tree automata we distinguish between

- Transitions $f(q_1, \ldots, q_n) \rightarrow q$ recognizing ≪ equivalence classes ≫
- Epsilon transitions $q \xrightarrow{\epsilon} q'$ representing rewriting between classes

Definition ($\mathcal{R}/E$-coherent automaton)

Let $\mathcal{A} = \langle \mathcal{F}, \mathcal{Q}, \mathcal{Q}_f, \Delta \rangle$ be a tree automaton, $\mathcal{R}$ a TRS and E a set of equations. The automaton $\mathcal{A}$ is said to be *$\mathcal{R}/E$-coherent* if
$\forall q \in \mathcal{Q} : \exists s \in \mathcal{T}(\mathcal{F}) :$

$$s \rightarrow_{\mathcal{A}}^{\not\epsilon *} q \wedge [\forall t \in \mathcal{T}(\mathcal{F}) : (t \rightarrow_{\mathcal{A}}^{\not\epsilon *} q \Rightarrow s =_E t) \wedge (t \rightarrow_{\mathcal{A}}{}^{*} q \Rightarrow s \rightarrow_{\mathcal{R}/E}^{*} t)]$$

Benchmarks

	Combinatory	NSPK	View-Only	Java prog. 1	Java prog. 2
TRS nb of rules	1	13	15	279	303
Initial Aut. size	43 / 23	14 / 4	21 / 18	26 / 49	33 / 33
Timbuk 2.2 :					
Final Aut. size	8043 / 23	151 / 16	730 / 74	1127 / 334	751 / 335
Time (secs)	**51.1**	**19.7**	**6420**	**25266**	**37387**
Timbuk 3.0 :					
Final Aut. size	8043 / 23	259 / 104	353 / 100		
Time (secs)	**60.1**	**3.1**	**2452**		
Tom-based :					
Final Aut. size	8043 / 23	171 / 21	938 / 89	1974 / 637	1611 / 672
Time (secs)	**5.9**	**5.9**	**150**	**360**	**303**
Bddbddb-based :					
Final Aut. size	? / 25	? / 183	? / 97		
Time (secs)	**0.008**	**2.9**	**3.3**		

Applications : Java bytecode verification (II)

Proving safety properties on Java bytecode using reachability analysis

Java Source .java

```
class TestList{
  public static void main(String[] argv){
    List lpos=null;
    InvList lneg=null;
    int x;
    boolean pos;
    pos= true;
    try {x=System.in.read()};}
    catch(java.io.IOException e){x=0;}
    while (x != -1){
      if (pos) {lpos= new List(x, lpos);
                pos=false;}
      else {lneg= new InvList(x, lneg);
            pos=true;}
      try {x=System.in.read();}
      catch(java.io.IOException e){x=0;}
    }
  }
}
```

Java Byte Code .class

```
public static void main(java.l
  Code:
   0:   aconst_null
   1:   astore_1
   2:   aconst_null
   3:   astore_2
   4:   iconst_1
   5:   istore  4
   7:   getstatic       #2; //
   10:  invokevirtual   #3; //
   13:  istore_3
   ...
   47:  new
   50:  dup
   51:  iload_3
```

Encoding JVM semantics and bytecode into rewriting (II)

Encoding of an add bytecode

$$\boxed{\text{add} : \frac{(m, pc, x :: y :: s, l)}{(m, pc+1, x+y :: s, l)}}$$

Encoding JVM semantics and bytecode into rewriting (II)

Encoding of an add bytecode

$$\boxed{\text{add} : \frac{(m, pc, x :: y :: s, l)}{(m, pc+1, x+y :: s, l)}}$$

1. Associate add bytecode to m, pc

```
public static void foo(...)
   ...
   11 :   add
```

```
frame(foo,11,s,l) -> xframe(add,foo,11,s,l)
```

Encoding JVM semantics and bytecode into rewriting (II)

Encoding of an add bytecode

$$\boxed{\text{add} : \frac{(m, pc, x :: y :: s, l)}{(m, pc+1, x+y :: s, l)}}$$

1. Associate add bytecode to m, pc

```
public static void foo(...)
    ...
    11 :   add
```

```
frame(foo,11,s,l) -> xframe(add,foo,11,s,l)
```

2. Pop x and y, start evaluation of $(x + y)$

```
xframe(add,m,pc,stack(y,stack(x,s)),l) -> xframe(xadd(x,y),m,pc,s,l)
```

Encoding JVM semantics and bytecode into rewriting (II)

Encoding of an add bytecode

$$\boxed{\text{add} : \frac{(m, pc, x :: y :: s, l)}{(m, pc+1, x+y :: s, l)}}$$

1. Associate add bytecode to m, pc

```
public static void foo(...)
    ...
    11 :    add
```

```
frame(foo,11,s,l) -> xframe(add,foo,11,s,l)
```

2. Pop x and y, start evaluation of $(x+y)$

```
xframe(add,m,pc,stack(y,stack(x,s)),l) -> xframe(xadd(x,y),m,pc,s,l)
```

3. Compute $(x+y)$

```
xadd(...) -> ...
... -> result(x)
```

Encoding JVM semantics and bytecode into rewriting (II)

Encoding of an add bytecode

$$\boxed{\text{add} : \frac{(m, pc, x :: y :: s, l)}{(m, pc + 1, x + y :: s, l)}}$$

1. Associate add bytecode to m, pc

```
public static void foo(...)
    ...
    11 :   add
```

```
frame(foo,11,s,l) -> xframe(add,foo,11,s,l)
```

2. Pop x and y, start evaluation of $(x + y)$

```
xframe(add,m,pc,stack(y,stack(x,s)),l) -> xframe(xadd(x,y),m,pc,s,l)
```

3. Compute $(x + y)$

```
xadd(...) -> ...
... -> result(x)
```

4. Push the result on top of s and move to next pc

```
xframe(result(x),m,pc,s,l) -> frame(m,next(pc),stack(x,s),l)
```