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. \( \text{while } (i>1) \text{ do } \{ \)
3. \( \quad n := n*(i-1); \)
4. \( \quad i := i-1; \} \)
5. \( \text{with } n = 0 \text{ unreachable} \)

If \( i \geq 1 \) in 1

then

\( n \geq 1 \) in 5

or

\( \quad \)
Verification using Model-checking

1. \( \{ i \geq 1 \} \)
   
   \( n := i ; \)

2. while \( i > 1 \) do {

3.     \( n := n \times (i - 1) ; \)

4.     \( i := i - 1 ; \) }

5. \( \{ n \geq 1 \} \)
Verification using Static Analysis and Abstract Interpretation

<table>
<thead>
<tr>
<th>$D = \mathbb{N}$</th>
<th>$D^# :$ intervals on $\mathbb{N}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>① {i \geq 1}</td>
<td>① $i^# = [1; +\infty[$, $n^# = [0; +\infty[$</td>
</tr>
<tr>
<td>n := i;</td>
<td></td>
</tr>
<tr>
<td>②</td>
<td></td>
</tr>
<tr>
<td>while (i&gt;1) do {</td>
<td></td>
</tr>
<tr>
<td>③</td>
<td></td>
</tr>
<tr>
<td>n := n*(i-1);</td>
<td></td>
</tr>
<tr>
<td>④</td>
<td></td>
</tr>
<tr>
<td>i := i-1; }</td>
<td></td>
</tr>
<tr>
<td>⑤ {n \geq 1}</td>
<td></td>
</tr>
</tbody>
</table>
Verification using Static Analysis and Abstract Interpretation

\[ D = \mathbb{N} \]

\[ D^\# : \text{intervals on } \mathbb{N} \]

<table>
<thead>
<tr>
<th>Step</th>
<th>Condition/Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( { i \geq 1 } )</td>
</tr>
<tr>
<td></td>
<td>( n := i )</td>
</tr>
<tr>
<td>2</td>
<td>while (( i &gt; 1 )) do</td>
</tr>
<tr>
<td></td>
<td>{ }</td>
</tr>
<tr>
<td>3</td>
<td>( n := n \times (i-1) )</td>
</tr>
<tr>
<td>4</td>
<td>( i := i - 1 )</td>
</tr>
<tr>
<td>5</td>
<td>( { n \geq 1 } )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Step</th>
<th>Condition/Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( i^# = [1; +\infty[, \ n^# = [0; +\infty[ )</td>
</tr>
<tr>
<td>2</td>
<td>( i^# = [1; +\infty[, \ n^# = [1; +\infty[ )</td>
</tr>
</tbody>
</table>
### Verification using Static Analysis and Abstract Interpretation

**$D = \mathbb{N}$**

<table>
<thead>
<tr>
<th>Step</th>
<th>Condition/Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>${i \geq 1}$</td>
</tr>
<tr>
<td></td>
<td>$n := i$;</td>
</tr>
<tr>
<td>2</td>
<td>while ($i &gt; 1$) do</td>
</tr>
<tr>
<td></td>
<td>{</td>
</tr>
<tr>
<td>3</td>
<td>$n := n * (i - 1)$;</td>
</tr>
<tr>
<td>4</td>
<td>$i := i - 1$; }</td>
</tr>
<tr>
<td>5</td>
<td>${n \geq 1}$</td>
</tr>
</tbody>
</table>

**$D^\# : intervals on \mathbb{N}$**

<table>
<thead>
<tr>
<th>Step</th>
<th>Interval</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$i^# = [1; +\infty[, n^# = [0; +\infty[  $</td>
</tr>
<tr>
<td>2</td>
<td>$i^# = [1; +\infty[, n^# = [1; +\infty[  $</td>
</tr>
<tr>
<td>3</td>
<td>$i^# = [2; +\infty[, n^# = [1; +\infty[  $</td>
</tr>
<tr>
<td>4</td>
<td>$i^# = [1; +\infty[, n^# = [1; +\infty[ *^# [1; +\infty[  $</td>
</tr>
</tbody>
</table>

Thomas Genet (IRISA)  
Reachability Analysis of Rewriting
**Verification using Static Analysis and Abstract Interpretation**

<table>
<thead>
<tr>
<th>$D = \mathbb{N}$</th>
<th>$D^# :$ intervals on $\mathbb{N}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>① ${i \geq 1}$</td>
<td>① $i^# = [1; +\infty[, \ n^# = [0; +\infty[$</td>
</tr>
<tr>
<td>n := i;</td>
<td>② $i^# = [1; +\infty[, \ n^# = [1; +\infty[$</td>
</tr>
<tr>
<td>② while $(i &gt; 1)$ do {</td>
<td>③ $i^# = [2; +\infty[, \ n^# = [1; +\infty[$</td>
</tr>
<tr>
<td>③ n := n*(i-1);</td>
<td>④ $i^# = [1; +\infty[, \ n^# = [1; +\infty[ \ast^# [1; +\infty[$</td>
</tr>
<tr>
<td>④ i := i-1; }</td>
<td>⑤ $i^# = [1; +\infty[, \ n^# = [1; +\infty[ $</td>
</tr>
<tr>
<td>⑤ ${n \geq 1}$</td>
<td></td>
</tr>
</tbody>
</table>
Verification using a Proof Assistant

1. \{ i \geq 1 \} 
   n := i;

2. 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. \( \{ \text{invariant } n \geq 1\} \)

   \( n := n \times (i-1); \)

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\}
   \text{while} (i>1) \text{do } \{\]
3. \{\text{invariant } n \geq 1\}
   \[ n := n \ast (i-1) ; \]
4. \[ i := i-1 ; \} \]
5. \{n \geq 1\}

\[
\text{FORALL (i: int):}
\]
\[ i \geq 1 \text{ IMPLIES}
\]
\[ (\text{FORALL (x: int):}
\]
\[ x = i \text{ IMPLIES}
\]
\[ (\text{FORALL (i0: int):}
\]
\[ \text{FORALL (x0: int):}
\]
\[ x0 \geq 1 \text{ IMPLIES}
\]
\[ i0 > 1 \text{ IMPLIES}
\]
\[ (\text{FORALL (x1: int):}
\]
\[ x1 = x0 \ast (i0 - 1)
\]
\[ \text{IMPLIES } x1 \geq 1)\}
\]
<table>
<thead>
<tr>
<th>Step</th>
<th>Specification/Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>${i \geq 1}$</td>
</tr>
<tr>
<td></td>
<td>$n := i$;</td>
</tr>
<tr>
<td>2</td>
<td>${i \geq 1, n \geq 1}$</td>
</tr>
<tr>
<td></td>
<td>while (i&gt;1) do {</td>
</tr>
<tr>
<td>3</td>
<td>${\text{invariant } n \geq 1}$</td>
</tr>
<tr>
<td></td>
<td>$n := n \cdot (i-1)$;</td>
</tr>
<tr>
<td>4</td>
<td>$i := i-1$; }</td>
</tr>
<tr>
<td>5</td>
<td>${n \geq 1}$</td>
</tr>
</tbody>
</table>

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 \cdot (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!
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

Rewrite rules:

- $0 + x \rightarrow x$

Diagram:

```
(0 + 0) + (0 + 1) 0 + (0 + 1)
\arrow{0 + (0 + 1)}
\arrow{(0 + 0) + 1}
\arrow{0 + 1} \rightarrow 1
```
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 system (TRS) = set of rewrite rules

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

\[
\begin{align*}
0 + (0 + 1) & \rightarrow_{\mathcal{R}} 1 \\
(0 + 0) + 1 & \rightarrow_{\mathcal{R}} 0 + 1 \\
(0 + 0) + (0 + 1) & \rightarrow_{\mathcal{R}}^* 1
\end{align*}
\]
TRS as a formal model of programs

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

1. \[ n := i ; \]
2. \[ \text{while } (i > 1) \text{ do } \{ \]
3. \[ n := n \times (i-1) ; \]
4. \[ i := i - 1 ; \}
5. \[ \]

Proving safety by (un)reachability analysis :

\[ (1, i, x) \not\rightarrow^* (5, y, 0) \quad \text{with } i \geq 1, x, y \in \mathbb{N} \]
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. }

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

\[
\begin{align*}
(1, I, N) & \rightarrow (2, I, I) \\
(2, s(s(I)), N) & \rightarrow (3, s(s(I)), N) \\
(3, s(I), N) & \rightarrow (4, s(I), I* N) \\
(4, s(I), N) & \rightarrow (2, I, N) \\
(2, 0, N) & \rightarrow (5, 0, N) \\
(2, s(0), N) & \rightarrow (5, s(0), N) \\
0*X & \rightarrow 0 \\
s(X)*Y & \rightarrow Y + (X*Y) \\
\ldots
\end{align*}
\]

Proving safety by (un)reachability analysis:
\[
(1, i, x) \not\rightarrow_R^* (5, y, 0)
\]

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]

\[
n := n \times (i - 1); \]

[4]

\[
i := i - 1; \}

[5]

\[
0 \times X \rightarrow 0
\]

\[
s(X) \times Y \rightarrow Y + (X \times Y)
\]

\[
\cdots
\]

\[
\mathcal{F} = \{((-,-,-), 0, s, +, \times, 1, 2, 3, 4, 5)\}
\]

\[
\mathcal{X} = \{I, N, X, Y\}
\]

Proving safety by (un)reachability analysis:

\[
(1, i, x) \not\rightarrow^{R*} (5, y, 0) \quad \text{with } i \geq 1, x, y \in \mathbb{N}
\]
Reachability analysis of rewriting

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

- Undecidable in general (TRS are Turing-complete)
Reachability analysis of rewriting

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

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

Diagram:

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

\[
\begin{align*}
\text{s} & \quad \rightarrow & \quad \bullet \\
\rightarrow & \quad \bullet & \quad \rightarrow & \quad t \\
\cdot & \quad \rightarrow & \quad \bullet & \quad \rightarrow & \quad \bullet
\end{align*}
\]
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

$$
\begin{align*}
\text{where } & \mathcal{R}^*(\mathcal{L}) = \{u \mid s \in \mathcal{L} \land s \rightarrow_{\mathcal{R}}^* u\} \\
\text{Decidable, if } & \mathcal{R}^*(\{s\}) \text{ is finite} \quad \approx \text{finite model-checking}
\end{align*}
$$
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

\[ \mathcal{R}^*(\{s\}) \]

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

- 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:

$$(1, i, x) \not\rightarrow^* (5, y, 0)$$

with $i \geq 1$, $x, y \in \mathbb{N}$
Reachability analysis of rewriting (extended)

Recall that for verification, the problem we have is:

\[(1, i, x) \not\rightarrow R^* (5, y, 0)\] with \(i \geq 1, x, y \in \mathbb{N}\)

which can be seen as:

\[
\begin{array}{c}
\bullet \\
\bullet \\
\vdots \\
\mathcal{L}
\end{array}
\rightarrow
\begin{array}{c}
\bullet \\
\bullet \\
\vdots \\
R^*(\mathcal{L})
\end{array}
\bigcap
\begin{array}{c}
\text{Bad}
\end{array}
= \emptyset
\]

The reachability analysis problem becomes:

\[R^*(\mathcal{L}) \cap \text{Bad} = \emptyset?\]
Two applications of reachability analysis of rewriting

\[ \mathcal{R}^*(\mathcal{L}) \cap \text{Bad} = \emptyset ? \]

- Java application verification [Boichut, Genet, Jensen, Le Roux, 07]
- Cryptographic protocol verification [Genet, Klay, 00]
- 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]
Two applications of reachability analysis of rewriting

\[ R^*(L) \cap \text{Bad} = \emptyset \]

- Java application verification [Boichut, Genet, Jensen, Le Roux, 07]
- Cryptographic protocol verification [Genet, Klay, 00]
  - \( L \) = protocol initial configurations
  - \( R \) = specification of protocol exchanged messages
deduction rules of the intruder
- 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]
Two applications of reachability analysis of rewriting

\[ \mathcal{R}^*(\mathcal{L}) \cap \text{Bad} = \emptyset ? \]

- 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
  - 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 $R^*(\mathcal{L})$?

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 literature:
- 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 $R^*(L)$?

Many formalisms in the literature:
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 $R^*(L) \cap \text{Bad} = \emptyset$
- The complexity of the algorithm for $\cap$ is quadratic
- The complexity of the algorithm deciding $\subseteq \emptyset$ is polynomial
\( R \) classes where \( L \) regular \( \Rightarrow R^*(L) \) regular

\[ \begin{array}{c}
\text{G} & \text{Ground} \\
& [\text{Dauchet, Tison, 90}, \text{Brainerd, 69}] \\
\text{RL-M} & \text{Right-linear and Monadic} [\text{Salomaa, 88}] \\
\text{L-SM} & \text{Linear and Semi-Monadic} [\text{Coquidé et al., 91}] \\
\text{L-G}^{-1} & \text{Linear and inversely Growing} [\text{Jacquemard, 96}] \\
\text{RL-G}^{-1} & \text{Right-linear and inversely Growing} [\text{Nagaya, Toyama, 99}] \\
\text{L-GSM} & \text{Linear Generalized Semi-Monadic} [\text{Gyenizse, Vágvölgyi, 98}] \\
\text{L-FPO, RL-FPO} & \text{(Right)-Linear Finite Path Overlapping} [\text{Takai et al. 00}] \\
\text{L-GFPO} & \text{Linear Generalized Finite Path Overlapping} [\text{Takai 04}] \\
\end{array} \]
$\mathcal{R}$ 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)

**Ground** : $s \rightarrow t$

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

Tree grammar $G$

\[
\{ f(s^*(a)) \} \\
\text{axiom : } N_1
\]

\[
N_1 \::= \quad f(N_2) \\
N_2 \::= \quad s(N_2) \\
N_2 \quad := \quad a
\]

$N_1 \xrightarrow{*}_G f(s(s(a)))$
Tree automata recognizing **regular** sets of terms

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

<table>
<thead>
<tr>
<th>Tree grammar $G$</th>
<th>Tree automaton $A$</th>
</tr>
</thead>
<tbody>
<tr>
<td>${f(s^*(a))}$</td>
<td>${f(s^*(a))}$</td>
</tr>
<tr>
<td>$N_1 := f(N_2)$</td>
<td>$f(q_2)$ $\rightarrow q_1$</td>
</tr>
<tr>
<td>$N_2 := s(N_2)$</td>
<td>$s(q_2)$ $\rightarrow q_2$</td>
</tr>
<tr>
<td>$N_2 := a$</td>
<td>$a$ $\rightarrow q_2$</td>
</tr>
</tbody>
</table>

$N_1 \xrightarrow{\ast} G f(s(s(a)))$

$\forall q \in Q_f, L(A) = \{f(s^*(a))\}$
Tree automata recognizing \textbf{regular} sets of terms

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

<table>
<thead>
<tr>
<th>Tree grammar $G$</th>
<th>Tree automaton $A$</th>
</tr>
</thead>
<tbody>
<tr>
<td>{ $f(s^*(a))$ }</td>
<td>{ $f(s^*(a))$ }</td>
</tr>
<tr>
<td>axiom : $N_1$</td>
<td>final state : $q_1$</td>
</tr>
</tbody>
</table>

\begin{align*}
N_1 & := f(N_2) \\
N_2 & := s(N_2) \\
N_2 & := a \\
N_1 \rightarrow^*_G f(s(s(a)))
\end{align*}

\begin{align*}
f(q_2) & \rightarrow q_1 \\
s(q_2) & \rightarrow q_2 \\
a & \rightarrow q_2 \\
f(s(s(a))) \rightarrow^*_A q_1
\end{align*}

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

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

$f(s(s(a))) \rightarrow^*_A q_1$ and $q_1 \in Q_f$. 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

\[
\forall l \rightarrow r \in \mathcal{R}, \forall q \in Q, \forall \sigma : \mathcal{X} \rightarrow Q : \\
\quad l\sigma \rightarrow_{\mathcal{B}^*} q \implies r\sigma \rightarrow_{\mathcal{B}^*} q
\]
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

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

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

**Theorem (Upper bound)**

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

$$\mathcal{L}(\mathcal{B}) \supseteq \mathcal{L}(\mathcal{A})$$

\(\mathcal{B}\) is $\mathcal{R}$-closed

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

Principle: completion of $\mathcal{A}$ with new transitions until it is $\mathcal{R}$-closed

\[ \mathcal{A}_{completed} \to \mathcal{A}^*_{\mathcal{R}} \]

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

\[ \mathcal{L}(\mathcal{A}^*_{\mathcal{R}}) \supseteq \mathcal{L}(\mathcal{A}) \]

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

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

Tree automata completion algorithm

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

$$\begin{align*}
l \sigma & \rightarrow_r r \sigma \\
A & \ast \rightarrow q
\end{align*}$$
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

Compute $\mathcal{A}_1^\mathcal{R}$, $\mathcal{A}_2^\mathcal{R}$, ..., 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 $\Rightarrow \mathcal{L}(\mathcal{A}_\mathcal{R}^*) \supseteq \mathcal{R}^*(\mathcal{L}(\mathcal{A}))$
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

Compute $\mathcal{A}_\mathcal{R}^1, \mathcal{A}_\mathcal{R}^2, \ldots$ 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

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

\[ \mathcal{A} \text{ completed into } \mathcal{A}^*_\mathcal{R} \Rightarrow \mathcal{L}(\mathcal{A}^*_\mathcal{R}) \supseteq \mathcal{L}(\mathcal{A}) \]

\[ \mathcal{A}^*_\mathcal{R} \text{ is } \mathcal{R}\text{-closed} \Rightarrow \mathcal{L}(\mathcal{A}^*_\mathcal{R}) \supseteq \mathcal{R}^*(\mathcal{L}(\mathcal{A})) \]
Tree Automata Completion may not terminate

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

<table>
<thead>
<tr>
<th>$A^0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f(q_1, q_2) \rightarrow q_0$</td>
</tr>
<tr>
<td>$a \rightarrow q_1$</td>
</tr>
<tr>
<td>$b \rightarrow q_2$</td>
</tr>
<tr>
<td>${ f(a, b) }$</td>
</tr>
</tbody>
</table>
Tree Automata Completion may not terminate

\[ R = \{ f(x, y) \rightarrow f(g(x), y) \} \]

<table>
<thead>
<tr>
<th>( A^0 )</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( f(q_1, q_2) \rightarrow q_0 )</td>
</tr>
<tr>
<td></td>
<td>( a \rightarrow q_1 )</td>
</tr>
<tr>
<td></td>
<td>( b \rightarrow q_2 )</td>
</tr>
<tr>
<td></td>
<td>{ ( f(a, b) ) }</td>
</tr>
</tbody>
</table>

\[ f \]

\[ q_1 \]

\[ q_2 \]

\[ f \rightarrow_R \]

\[ g \]

\[ q_2 \]

\[ q_1 \]
Tree Automata Completion may not terminate

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

\[ \begin{array}{|c|c|}
\hline
A^0 & \\
\hline
f(q_1, q_2) \rightarrow q_0 & \\
\hline
a \rightarrow q_1 & \\
b \rightarrow q_2 & \\
\{ f(a, b) \} & \\
\hline
\end{array} \]

\[ \begin{array}{c}
q_0 \xrightarrow{A^0} f \xrightarrow{R} f \xrightarrow{A_R^1} q_0 \\
q_1 \xrightarrow{f} q_2 \xrightarrow{g} q_2 \xrightarrow{f} q_0 \\
q_1 \xrightarrow{a} q_1 \xrightarrow{b} q_2
\end{array} \]

Normalization is necessary!
Tree Automata Completion may not terminate

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

<table>
<thead>
<tr>
<th>$\mathcal{A}^0$</th>
<th>$\mathcal{A}^1_{\mathcal{R}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f(q_1, q_2) \rightarrow q_0$</td>
<td>$g(q_1) \rightarrow q_3$</td>
</tr>
<tr>
<td>$a \rightarrow q_1$</td>
<td>$f(q_3, q_2) \rightarrow q_0$</td>
</tr>
<tr>
<td>$b \rightarrow q_2$</td>
<td></td>
</tr>
<tr>
<td>${ f(a, b) }$</td>
<td>${ f(a, b), f(g(a), b) }$</td>
</tr>
</tbody>
</table>

Normalization is necessary!

$q_0 \xrightarrow{\mathcal{A}^0} f \xrightarrow{\mathcal{A}^1} f \xrightarrow{\mathcal{R}} f \xrightarrow{\mathcal{A}^1_{\mathcal{R}}} q_0$

$q_1 \quad q_2$

$q_3$
Tree Automata Completion may not terminate

\[ R = \{ f(x, y) \rightarrow f(g(x), y) \} \]

\[
\begin{array}{|c|c|}
\hline
A^0 & A^1_R \\
\hline
f(q_1, q_2) \rightarrow q_0 & g(q_1) \rightarrow q_3 \\
a \rightarrow q_1 & f(q_3, q_2) \rightarrow q_0 \\
b \rightarrow q_2 & \{ f(a, b) \} \rightarrow \{ f(a, b), f(g(a), b) \} \\
\hline
\end{array}
\]

Normalization is necessary!
Tree Automata Completion may not terminate

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

<table>
<thead>
<tr>
<th>( \mathcal{A}^0 )</th>
<th>( \mathcal{A}_{\mathcal{R}}^1 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( f(q_1, q_2) \rightarrow q_0 )</td>
<td>( g(q_1) \rightarrow q_3 )</td>
</tr>
<tr>
<td>( a \rightarrow q_1 )</td>
<td>( f(q_3, q_2) \rightarrow q_0 )</td>
</tr>
<tr>
<td>( b \rightarrow q_2 )</td>
<td></td>
</tr>
<tr>
<td>{ f(a, b) }</td>
<td>{ f(a, b), f(g(a), b) }</td>
</tr>
</tbody>
</table>

Normalization is necessary!
Tree Automata Completion may not terminate

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

| $A^0$ | $A^1_R$ | ...
|-------|---------|---------|
| $f(q_1, q_2) \rightarrow q_0$ | $g(q_1) \rightarrow q_3$ | ...
| $a \rightarrow q_1$ | $f(q_3, q_2) \rightarrow q_0$ | ...
| $b \rightarrow q_2$ | \{ 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 $A$ using $A$ when possible, use new states otherwise.

Theorem

Given a linear TRS $R$ and a tree automaton $A$, if tree automata completion with exact normalization strategy terminates on $A^* R$, then $L(A^* R) = R^* L(A)$.

Theorem

Tree automata completion with exact normalization strategy terminates for TRS in classes: $G$, $L\text{-SM}$, $L\text{-G}$, $L\text{-GSM}$, $L\text{-FPO}$ and $L\text{-GFPO}$.
Exact Normalization Strategy

[Feuillade, Genet, Viet Triem Tong, 04]

Principle of Exact Normalization Strategy

Normalize new transitions added to $A$ using $A$ when possible, use new states otherwise.

Theorem

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

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

[Feuillade, Genet, Viet Triem Tong, 04]

Principle of Exact Normalization Strategy
Normalize new transitions added to $A$ using $A$ when possible, use new states otherwise.

Theorem
Given a linear TRS $R$ and a tree automaton $A$, if tree automata completion with exact normalization strategy terminates on $A_R^*$, then

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

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.

\[
\text{Approx} \cap \text{Bad} = \emptyset \Rightarrow \mathcal{R}^*(\mathcal{L}) \cap \text{Bad} = \emptyset
\]
Building approximations using normalization rules

\[ R = \{ f(x, y) \rightarrow f(g(x), g(y)) \} \]

\[
\begin{array}{|c|c|}
\hline
A^0 & \ \ \\
\hline
f(q_1, q_2) \rightarrow q_0 & a \rightarrow q_1 \\
& b \rightarrow q_2 \\
\hline
\end{array}
\]

\[ f \quad R \quad f \]

\[ x \quad y \]

\[ g \quad g \]

\[ x \quad y \]
Building approximations using normalization rules

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

\[
\begin{array}{|c|}
\hline
A^0 \\
\hline
f(q_1, q_2) \rightarrow q_0 \\
a \rightarrow q_1 \\
b \rightarrow q_2 \\
\hline
\end{array}
\]

[Genet and Viet Triem Tong 2001]
Building approximations using normalization rules

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

\[
\begin{array}{|c|}
\hline
\mathcal{A}^0 \\
\hline
f(q_1, q_2) \rightarrow q_0 \\
a \rightarrow q_1 \\
b \rightarrow q_2 \\
\hline
\end{array}
\]

[Genet and Viet Triem Tong 2001]
Building approximations using normalization rules

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

\[ A^0 \]

<table>
<thead>
<tr>
<th>( f(q_1, q_2) \rightarrow q_0 )</th>
<th>( a \rightarrow q_1 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( b \rightarrow q_2 )</td>
<td></td>
</tr>
</tbody>
</table>

\[ f \rightarrow q_0 \]

\[ g \]

\[ g \]

\[ q_1 \]

\[ q_2 \]
Building approximations using normalization rules

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

\[ R = \{ f(q_1, q_2) \rightarrow q_0 \}
\]

- \( a \rightarrow q_1 \)
- \( b \rightarrow q_2 \)

\[ f \rightarrow q_0 \]

\[ g \]

\[ g \]

\[ q_1 \]

\[ q_2 \]

\[ [f(g(q_1), y) \rightarrow z] \rightarrow [g(q_1) \rightarrow q_1, y \rightarrow z] \]
Building approximations using normalization rules

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

\[
\begin{array}{|l|}
\hline
A^0 \\
\hline
f(q_1, q_2) \rightarrow q_0 \\
a \rightarrow q_1 \\
b \rightarrow q_2 \\
\hline
\end{array}
\]

\[
f \rightarrow q_0
\]

\[
g \quad g
\]

\[
q_1 \\
q_2
\]

\[ [f(g(q_1), y) \rightarrow z] \rightarrow [g(q_1) \rightarrow q_1 \quad y \rightarrow z] \]

\[ [f(g(q_1), g(q_2)) \rightarrow z] \rightarrow [g(q_1) \rightarrow q_1 \quad g(q_2) \rightarrow z] \]
Building approximations using normalization rules

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

<table>
<thead>
<tr>
<th>[ A^0 ]</th>
</tr>
</thead>
<tbody>
<tr>
<td>( f(q_1, q_2) \rightarrow q_0 )</td>
</tr>
<tr>
<td>( a \rightarrow q_1 )</td>
</tr>
<tr>
<td>( b \rightarrow q_2 )</td>
</tr>
</tbody>
</table>

\[ f \rightarrow q_0 \]
\[ g \quad g \]
\[ q_1 \quad q_2 \]

\[[f(g(q_1), y) \rightarrow z] \rightarrow [g(q_1) \rightarrow q_1 \quad y \rightarrow z] \]

\[[f(g(q_1), g(q_2)) \rightarrow q_0] \rightarrow [g(q_1) \rightarrow q_1 \quad g(q_2) \rightarrow q_0] \]
Building approximations using normalization rules

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

\[
\begin{array}{|c|}
\hline
A^0 \\
\hline
f(q_1, q_2) \rightarrow q_0 \\
a \rightarrow q_1 \\
b \rightarrow q_2 \\
\hline
\end{array}
\]

- \( f \rightarrow q_0 
- g 
- g 
- q_1 
- q_2 

\[[f(g(q_1), y) \rightarrow z] -> [g(q_1) \rightarrow q_1 \hspace{1cm} y \rightarrow z] \]

\[[f(g(q_1), g(q_2)) \rightarrow q_0] -> [g(q_1) \rightarrow q_1 \hspace{1cm} g(q_2) \rightarrow q_0] \]
Building approximations using normalization rules

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

<table>
<thead>
<tr>
<th>A^0</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>f(q_1, q_2) \rightarrow q_0</td>
<td></td>
</tr>
<tr>
<td>a \rightarrow q_1</td>
<td></td>
</tr>
<tr>
<td>b \rightarrow q_2</td>
<td></td>
</tr>
</tbody>
</table>

\[
\begin{align*}
[f(g(q_1), y) \rightarrow z] & \rightarrow [g(q_1) \rightarrow q_1 \quad y \rightarrow z] \\
[f(g(q_1), g(q_2)) \rightarrow q_0] & \rightarrow [g(q_1) \rightarrow q_1 \quad g(q_2) \rightarrow q_0]
\end{align*}
\]
Building approximations using normalization rules

\[ R = \{ f(x, y) \rightarrow f(g(x), g(y)) \} \]

\[
\begin{array}{|c|c|}
\hline
& A^0 & A^1_R \\
\hline
f(q_1, q_2) & q_0 & g(q_1) \rightarrow q_1 \\
a & q_1 & g(q_2) \rightarrow q_0 \\
b & q_2 & f(q_1, q_0) \rightarrow q_0 \\
\hline
\end{array}
\]

\[
f \rightarrow q_0
\]

\[
\begin{array}{c}
q_1 \\
q_0
\end{array}
\]

[f(g(q_1), y) \rightarrow z] \rightarrow [g(q_1) \rightarrow q_1 \quad y \rightarrow z]

[f(g(q_1), g(q_2)) \rightarrow q_0] \rightarrow [g(q_1) \rightarrow q_1 \quad g(q_2) \rightarrow q_0]
Building approximations using normalization rules

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

<table>
<thead>
<tr>
<th>( \mathcal{A}^0 )</th>
<th>( \mathcal{A}^1_\mathcal{R} )</th>
<th>( \mathcal{A}^2_\mathcal{R} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( f(q_1, q_2) \rightarrow q_0 )</td>
<td>( g(q_1) \rightarrow q_1 )</td>
<td>( g(q_0) \rightarrow q_0 )</td>
</tr>
<tr>
<td>( a \rightarrow q_1 )</td>
<td>( g(q_2) \rightarrow q_0 )</td>
<td></td>
</tr>
<tr>
<td>( b \rightarrow q_2 )</td>
<td>( f(q_1, q_0) \rightarrow q_0 )</td>
<td></td>
</tr>
</tbody>
</table>

\[
\begin{array}{c}
  f \\
  \downarrow \\
  q_1 \quad q_0
\end{array}
\]

\[ [f(g(q_1), y) \rightarrow z] \rightarrow [g(q_1) \rightarrow q_1 \quad y \rightarrow z] \]
\[ [f(g(q_1), g(q_2)) \rightarrow q_0] \rightarrow [g(q_1) \rightarrow q_1 \quad g(q_2) \rightarrow q_0] \]
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
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

\[ R = \begin{cases} 
(1) & f(x, y) \rightarrow f(g(x), y) \\
(2) & f(x, y) \rightarrow f(x, h(y)) 
\end{cases} \]

prove that \( f(a, b) \not\rightarrow_{R^*} f(a, h(g(b))) \)?
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 \( f(a, b) \not\rightarrow_{\mathcal{R}^*} f(a, h(g(b))) \)?
Intuition behind equational over-approximations

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

prove that \( f(a, b) \not\rightarrow^{R^*} f(a, h(g(b))) \)?
Intuition behind equational over-approximations

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

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

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

\[ C_1 = \{ f(a, b) \} \]

\[ C_2 = \{ f(g(a), b) \} \]

\[ C_3 = \{ f(g(g(a)), b) \} \]
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 \( E = \{ g(g(x)) = g(x), h(h(x)) = h(x) \} \)
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 \( \mathcal{E} = \{ g(g(x)) = g(x), h(h(x)) = h(x) \} \)
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 \( E = \{ g(g(x)) = g(x), h(h(x)) = h(x) \} \)

\[ s \rightarrow_{\mathcal{R} / E} t \iff s =_{E} s' \rightarrow_{\mathcal{R}} t' =_{E} t \]
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 \( E = \{ g(g(x)) = g(x), h(h(x)) = h(x) \} \)

\[ s \rightarrow_{\mathcal{R}/E} t \iff s = _{E} s' \rightarrow_{\mathcal{R}} t' = _{E} t \quad (\text{e.g. } f(a, b) \rightarrow_{\mathcal{R}/E} f(g(g(g(a))), b)) \]
Intuition behind equational over-approximations

\[ \mathcal{R} = \left\{ \begin{array}{l}
(1) f(x, y) \rightarrow f(g(x), y) \\
(2) f(x, y) \rightarrow f(x, h(y))
\end{array} \right. \]

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

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

\[ s \rightarrow_{\mathcal{R}/E} t \iff s = E s' \rightarrow_{\mathcal{R}} t' = E t \]

(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

\( \mathcal{R} = \{ \)
\[
\begin{align*}
(1) & \quad f(x, y) \rightarrow f(g(x), y) \\
(2) & \quad f(x, y) \rightarrow f(x, h(y))
\end{align*}
\]
\( \}

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

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

\[
\begin{array}{c}
C_1 = \{ f(a, b) \} \\
C_2 = \{ f(g^+(a), b) \} \\
C_3 = \{ f(a, h^+(b)) \} \\
C_4 = \{ f(g^+(a), h^+(b)) \}
\end{array}
\]

\[
\begin{array}{c}
s \rightarrow_{\mathcal{R}/E} t \iff s =_{E} s' \rightarrow_{\mathcal{R}} t' =_{E} t \\
f(a, b) \not\rightarrow_{\mathcal{R}/E}^* f(a, h(g(b))) \\
(e.g. f(a, b) \rightarrow_{\mathcal{R}/E} f(g(g(g(a))), b)) \\
\Rightarrow f(a, b) \not\rightarrow_{\mathcal{R}}^* f(a, h(g(b)))
\end{array}
\]
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 \( E = \{ g(g(x)) = g(x), h(h(x)) = h(x) \} \)

\[
\begin{align*}
\text{s} \rightarrow_{\mathcal{R}/E} \text{t} & \iff s = E s' \rightarrow_{\mathcal{R}} t' = E t \\
\text{f}(a, b) \not\rightarrow^*_{\mathcal{R}/E} \text{f}(a, h(g(b))) & \implies \text{f}(a, b) \not\rightarrow^*_{\mathcal{R}} \text{f}(a, h(g(b)))
\end{align*}
\]

[Meseguer, Palomino, Marti-Oliet, 03] [Takai, 04]
Equations for tree automata approximation

Simplification relation \( \mathcal{A} \leadsto_E \mathcal{A}' \)

Given \((u = v) \in E\) and a tree automaton \(\mathcal{A}\)
Equations for tree automata approximation

Simplification relation $A \sim_E A'$

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

\[
\begin{align*}
u \sigma & \equiv_E v \sigma \\
* \downarrow A & \quad A \downarrow * \\
q_1 & \quad q_2
\end{align*}
\]  

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

$$u\sigma =_E v\sigma$$

* $\downarrow \mathcal{A}$ $\mathcal{A} \downarrow*$  \quad \Rightarrow \quad \text{merging of $q_1$ and $q_2$ applied to $\mathcal{A}$}

$q_1$ $q_2$

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 $A \sim_E A'$

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

\[
\begin{align*}
    u\sigma & =_E v\sigma \\
    * \downarrow A & A \downarrow * \\
    q_1 & q_2
\end{align*}
\]

$\Rightarrow$ merging of $q_1$ and $q_2$ applied to $A$

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

After completion step $i$, we propagate $E$ on $A^i_R$ using $\sim_E$ up to a fixpoint
Equations for tree automata approximation

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

<table>
<thead>
<tr>
<th>( A^0 )</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>( f(q_a, q_b) \rightarrow q_0 )</td>
<td></td>
</tr>
<tr>
<td>( a \rightarrow q_a )</td>
<td></td>
</tr>
<tr>
<td>( b \rightarrow q_b )</td>
<td></td>
</tr>
</tbody>
</table>

\( \mathcal{L}(A^0) = \{ f(a, b) \} \)
Equations for tree automata approximation

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

<table>
<thead>
<tr>
<th>( A^0 )</th>
<th>( A^1_{\mathcal{R}} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( f(q_a, q_b) \to q_0 )</td>
<td>( f(q_1, q_2) \to q_0 )</td>
</tr>
<tr>
<td>( a \to q_a )</td>
<td>( s(q_a) \to q_1 )</td>
</tr>
<tr>
<td>( b \to q_b )</td>
<td>( s(q_b) \to q_2 )</td>
</tr>
</tbody>
</table>

\( \mathcal{L}(A^0) = \{ f(a, b) \} \) \quad \mathcal{L}(A^1) = \{ f(a, b), f(s(a), s(b)) \} \)
Equations for tree automata approximation

\[ R = \{ f(x, y) \rightarrow f(s(x), s(y)) \} \quad \text{and} \quad E = \{ s(s(x)) = s(x) \} \]

\begin{array}{|c|c|c|}
\hline
\mathcal{A}^0 & \mathcal{A}^1_R & \mathcal{A}^2_R \\
\hline
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 \\
\hline
\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_R) = \{ f(a, b), f(s(s(a)), s(s(b))) \} \\
\hline
\end{array}
Equations for tree automata approximation

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

<table>
<thead>
<tr>
<th>( A^0 )</th>
<th>( A^1_R )</th>
<th>( A^2_R )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( f(q_a, q_b) \rightarrow q_0 )</td>
<td>( f(q_1, q_2) \rightarrow q_0 )</td>
<td>( f(q_3, q_4) \rightarrow q_0 )</td>
</tr>
<tr>
<td>( a \rightarrow q_a )</td>
<td>( s(q_a) \rightarrow q_1 )</td>
<td>( s(q_1) \rightarrow q_3 )</td>
</tr>
<tr>
<td>( b \rightarrow q_b )</td>
<td>( s(q_b) \rightarrow q_2 )</td>
<td>( s(q_2) \rightarrow q_4 )</td>
</tr>
</tbody>
</table>

\( \mathcal{L}(A^0) = \{ f(a, b) \} \) \quad \( \mathcal{L}(A^1) = \{ f(a, b), f(s(a), s(b)) \} \) \quad \( \mathcal{L}(A^2_R) = \{ f(a, b), f(s(s(a)), s(s(b))) \} \)

\[ s(s(q_a)) \quad \mathcal{E} \quad s(q_a) \]

\[ \downarrow^* \quad A^2 \quad \downarrow^* \]

\[ q_3 \quad q_1 \]
Equations for tree automata approximation

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

<table>
<thead>
<tr>
<th>( \mathcal{A}^0 )</th>
<th>( \mathcal{A}_R^1 )</th>
<th>( \mathcal{A}_R^2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( f(q_a, q_b) \rightarrow q_0 )</td>
<td>( f(q_1, q_2) \rightarrow q_0 )</td>
<td>( f(q_3, q_4) \rightarrow q_0 )</td>
</tr>
<tr>
<td>( a \rightarrow q_a )</td>
<td>( s(q_a) \rightarrow q_1 )</td>
<td>( s(q_1) \rightarrow q_3 )</td>
</tr>
<tr>
<td>( b \rightarrow q_b )</td>
<td>( s(q_b) \rightarrow q_2 )</td>
<td>( s(q_2) \rightarrow q_4 )</td>
</tr>
</tbody>
</table>

\( \mathcal{L}(\mathcal{A}^0) = \{ f(a, b) \} \)

\( \mathcal{L}(\mathcal{A}^1) = \{ f(a, b), f(s(a), s(b)) \} \)

\( \mathcal{L}(\mathcal{A}_R^2) = \{ f(a, b), f(s(s(a)), s(s(b))) \} \)

\[
\begin{align*}
s(s(q_a)) & =_E s(q_a) \\
\downarrow^* & \mathcal{A}_R^2 \downarrow^* \\
q_3 & \mathcal{A}_R^2 q_1
\end{align*}
\]

\[
\begin{align*}
s(s(q_b)) & =_E s(q_b) \\
\downarrow^* & \mathcal{A}_R^2 \downarrow^* \\
q_4 & \mathcal{A}_R^2 q_2
\end{align*}
\]
Equations for tree automata approximation

\[ \mathcal{R} = \{ f(x, y) \rightarrow f(s(x), s(y)) \} \quad \text{and} \quad E = \{ s(s(x)) = s(x) \} \]

<table>
<thead>
<tr>
<th>( \mathcal{A}^0 )</th>
<th>( \mathcal{A}^1_R )</th>
<th>( \mathcal{A}^2_R )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( f(q_a, q_b) \rightarrow q_0 )</td>
<td>( f(q_1, q_2) \rightarrow q_0 )</td>
<td>( f(q_3, q_4) \rightarrow q_0 )</td>
</tr>
<tr>
<td>( a \rightarrow q_a )</td>
<td>( s(q_a) \rightarrow q_1 )</td>
<td>( s(q_1) \rightarrow q_3 )</td>
</tr>
<tr>
<td>( b \rightarrow q_b )</td>
<td>( s(q_b) \rightarrow q_2 )</td>
<td>( s(q_2) \rightarrow q_4 )</td>
</tr>
</tbody>
</table>

\( \mathcal{L}(\mathcal{A}^0) = \{ f(a, b) \} \)
\( \mathcal{L}(\mathcal{A}^1) = \{ f(a, b), f(s(a), s(b)) \} \)
\( \mathcal{L}(\mathcal{A}^2_R) = \{ f(a, b), f(s(s(a)), s(s(b))) \} \)

\[
\begin{align*}
\downarrow^* & \quad s(s(q_a)) & =_E & s(q_a) \\
\downarrow^* & \quad A^2_R & = & q_3 \\
\downarrow^* & \quad A^2_R & = & q_1 \\
\downarrow^* & \quad s(s(q_b)) & =_E & s(q_b) \\
\downarrow^* & \quad A^2_R & = & q_4 \\
\downarrow^* & \quad A^2_R & = & q_2
\end{align*}
\]
Equations for tree automata approximation

\[ R = \{ f(x, y) \rightarrow f(s(x), s(y)) \} \text{ and } E = \{ s(s(x)) = s(x) \} \]

<table>
<thead>
<tr>
<th>( A^0 )</th>
<th>( A^1_R )</th>
<th>( A^2_R )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( f(q_a, q_b) \rightarrow q_0 )</td>
<td>( f(q_1, q_2) \rightarrow q_0 )</td>
<td>( f(q_a, q_b) \rightarrow q_0 )</td>
</tr>
<tr>
<td>( a \rightarrow q_a )</td>
<td>( s(q_a) \rightarrow q_1 )</td>
<td>( s(q_1) \rightarrow q_1 )</td>
</tr>
<tr>
<td>( b \rightarrow q_b )</td>
<td>( s(q_b) \rightarrow q_2 )</td>
<td>( s(q_2) \rightarrow q_2 )</td>
</tr>
</tbody>
</table>

\[ \mathcal{L}(A^0) = \{ f(a, b) \} \]
\[ \mathcal{L}(A^1) = \{ f(a, b), f(s(a), s(b)) \} \]
\[ \mathcal{L}(A^2_R) = \{ f(s^*(a), s^*(b)) \} \]

\[ s(s(q_a)) =_E s(q_a) \]
\[ s(s(q_a)) \downarrow^*_{A^2_R} = q_1 \]

\[ s(s(q_b)) =_E s(q_b) \]
\[ s(s(q_b)) \downarrow^*_{A^2_R} = q_2 \]
Properties of $\rightsquigarrow_E$

The simplification relation $\rightsquigarrow_E$ enjoys the following properties

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

The simplification relation $\rightsquigarrow_E$ enjoys the following properties

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

- $\rightsquigarrow_E$ terminates
Properties of $\leadsto_E$

The simplification relation $\leadsto_E$ enjoys the following properties:

- If $A \leadsto_E A'$ then $\mathcal{L}(A) \subseteq \mathcal{L}(A')$
- $\leadsto_E$ terminates
- $\leadsto_E$ 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 $A \rightsquigarrow_E A'$ then $\mathcal{L}(A) \subseteq \mathcal{L}(A')$
- $\rightsquigarrow_E$ 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 $A^i_{\mathcal{R},E}$ to $A^{i+1}_{\mathcal{R},E}$

$i$-th Completion step

- 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

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

Simplification

- Find instances of an equation $u = v$ of $E$ in $\mathcal{A}^{i+1}_{\mathcal{R}}$
New completion algorithm: from $\mathcal{A}_{\mathcal{R},E}^i$ to $\mathcal{A}_{\mathcal{R},E}^{i+1}$

$i$-th Completion step

$\mathcal{A}_{\mathcal{R}}^i \xrightarrow{l \sigma} \mathcal{A}_{\mathcal{R}}^i \xrightarrow{\mathcal{R}} r \sigma \xrightarrow{\mathcal{A}_{\mathcal{R}}^{i+1}} q \xleftarrow{\epsilon} q'$

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

Simplification

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

$u \sigma \xrightarrow{E} v \sigma$

$\mathcal{A}_{\mathcal{R}}^{i+1}, q' \xleftarrow{*} \mathcal{A}_{\mathcal{R}}^{i+1}, q' \xrightarrow{*} \mathcal{A}_{\mathcal{R}}^{i+1}$

- Rename $q_2$ by $q_1$ in $\mathcal{A}_{\mathcal{R}}^{i+1}$
- 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 \( \cap, \cup, \subseteq, \ldots \)
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 $\mathcal{A}, \mathcal{B}$:

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

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

**Timbuk** provides

- Tree automata implementation with $\cap$, $\cup$, $\neq \emptyset$, $\subseteq$, ...
- 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 $\mathcal{A}$, $\mathcal{B}$:

$$\text{checker}(\mathcal{A}, \mathcal{R}, \mathcal{B}) = \text{true} \implies \mathcal{L}(\mathcal{B}) \supseteq \mathcal{R}^{*}(\mathcal{L}(\mathcal{A}))$$

**checker** extracted from a Coq spec.  
[Boyer, Genet, Jensen, 08]
Applications: Java bytecode verification

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

\[ R^* (L) \cap Bad = \emptyset \]

- \( R = \) A Java byte code program \( P \)
- \( L = \) Java Virtual Machine (JVM) semantics
- \( L = \) Java Virtual Machine (JVM) initial state
Applications: Java bytecode verification

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

\[ R^*(\mathcal{L}) \cap \text{Bad} = \emptyset \]

- \( R \) = A Java byte code program \( P \)
- \( \mathcal{L} \) = Java Virtual Machine (JVM) initial state
- \( R^*(\mathcal{L}) \) = all JVM states reachable while executing \( P \)
- \( \text{Bad} \) = 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 \textit{left-linear} TRS

- \textbf{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

```java
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[] argv){
        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, ...
Subsequences of the form ..., i, i, ... with i ≥ 1 should not occur
One equation is enough:
outstack(x,outstack(y,z))=z
An example of verification performed on a Java program

```java
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[] argv)
    {
        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, ...
An example of verification performed on a Java program

```java
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[] argv){
        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[] argv)
    {
        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, ... 

Subsequences of the form ..., i, i, ... 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

![Diagram of the RAVAJ Java verification chain]

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

- R(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 operational semantics into a TRS is easier

- Precision result w.r.t. approximation (i.e. w.r.t. R/E) ≈ Equations could be used on TT, and predicate abstraction on TRS
Comparison with Regular (Abstract) Tree Model-Checking

- Comparison between Tree Tranducers and TRS is difficult
  - $R(L)$ can be computed with TT, not easy with TRS
Comparison with Regular (Abstract) Tree Model-Checking

- Comparison between Tree Transducers 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 Transducers 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 Transducers and TRS is difficult
  - $R(L)$ 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 operational 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 operational 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

Others equational abstractions
- Completion is more expensive than a pure rewriting approach
  - Generate equations automatically (in some cases)
- In practice, strong restrictions on equations (syntactical/coherence)

Other techniques based on rewriting
- Limited to $\ll$regular$\gg$ properties (e.g. no induction!)
  - Simpler properties $\Rightarrow$ needs less interaction
  - No need for termination or confluence of the TRS
Comparison with other verification techniques

- Classes of $R$ for which $R^*(L)$ 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 $\Rightarrow$ 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 literature
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
  - 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

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

- **Now**: extend the verification capabilities of tree automata completion
  - → 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
  - lift-up to temporal properties
  
  [Boyer, Genet, 09]

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

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

- **Now**: extend the verification capabilities of tree automata completion
  - lift-up to temporal properties
  
  [Boyer, Genet, 09]

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

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

- **Now**: extend the verification capabilities of tree automata completion
  $\leadsto$ lift-up to temporal properties  
  \[ \text{Boyer, Genet, 09} \]

- **Next year**: improve the completion-based verification framework
  - Counter-example extraction
  - Automatic refinement of equational approximations
  - Visualization 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 $R^*(L)$
  - Checking transformations of SQL query
  - Checking transformations of UML model
  - Javascript programs verification
Further Research (II)

Since the new completion algorithm is based on:

\[ l_\sigma \xrightarrow{R} r_\sigma \]

\[ A^i_\mathcal{R} \xrightarrow{\epsilon} q' \]

\[ q \xleftarrow{\epsilon} A^{i+1}_\mathcal{R} \]

instead of

\[ l_\sigma \xrightarrow{*} r_\sigma \]

\[ A^i_\mathcal{R} \xrightarrow{*} q \]

\[ q \xleftarrow{*} A^{i+1}_\mathcal{R} \]

from the \( \epsilon \)-graph we can obtain the \( \mathcal{R}/E \)-rewriting graph.
Further Research (II)

Since the new completion algorithm is based on:

\[ \frac{\sigma}{R} \rightarrow \frac{\sigma}{R} \]

\[ A^i_R \quad \Downarrow \quad A^{i+1}_R \]

\[ q \leftarrow \frac{\epsilon}{A^{i+1}_R} \quad q' \]

instead of

\[ \frac{\sigma}{R} \rightarrow \frac{\sigma}{R} \]

\[ A^i_R \quad * \quad A^{i+1}_R \]

\[ q \leftarrow \frac{*}{A^{i+1}_R} \]

from the \( \epsilon \)-graph we can obtain the \( R/E \)-rewriting graph

\[ R = \{ f(x, y) \rightarrow f(g(x), y), \]
\[ f(x, y) \rightarrow f(x, h(y)) \} \]

\[ E = \{ g(g(x)) = g(x), \]
\[ h(h(x)) = h(x) \} \]

\[ L = \{ f(a, b) \} \]
Further Research (II)

Since the new completion algorithm is based on:

\[
\begin{align*}
\frac{l_\sigma}{\mathcal{R}} & \rightarrow r_\sigma \\
\mathcal{A}^i_{\mathcal{R}} & \downarrow \quad \downarrow \\
q & \leftarrow \epsilon \quad q' \\
\mathcal{A}^{i+1}_{\mathcal{R}} & \downarrow \\
\frac{l_\sigma}{\mathcal{R}} & \rightarrow r_\sigma \\
\mathcal{A}^i_{\mathcal{R}} & \downarrow \quad \downarrow \\
q & \leftarrow \epsilon \quad q' \\
\mathcal{A}^{i+1}_{\mathcal{R}} & \downarrow \\
\end{align*}
\]

instead of

\[
\begin{align*}
\frac{l_\sigma}{\mathcal{R}} & \rightarrow r_\sigma \\
\mathcal{A}^i_{\mathcal{R}} & \downarrow \quad \downarrow \\
q & \leftarrow * \\
\mathcal{A}^{i+1}_{\mathcal{R}} & \downarrow \\
\frac{l_\sigma}{\mathcal{R}} & \rightarrow r_\sigma \\
\mathcal{A}^i_{\mathcal{R}} & \downarrow \quad \downarrow \\
q & \leftarrow * \\
\mathcal{A}^{i+1}_{\mathcal{R}} & \downarrow \\
\end{align*}
\]

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

\[
\mathcal{R} = \{ f(x, y) \rightarrow f(g(x), y), \\
\quad f(x, y) \rightarrow f(x, h(y)) \}
\]

\[
E = \{ g(g(x)) = g(x), \\
\quad h(h(x)) = h(x) \}
\]

\[
\mathcal{L} = \{ f(a, b) \} 
\]
\( \mathcal{R}/E \)-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 \(\ll\) equivalence classes \(\gg\)
- 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

Thomas Genet (IRISA)
Reachability Analysis of Rewriting
$\mathcal{R}/E$-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
**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

\[ R = \{ s \rightarrow t, u \rightarrow v \} \]
\[ E = \{ s = u \} \]

New completion

Old completion
**R/E-Coherent tree automata**

In the tree automata we distinguish between:

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

\[ R = \{ s \to t, u \to v \} \]
\[ E = \{ s = u \} \]

New completion

Old completion
**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 (R/E-coherent automaton)**

Let \( A = \langle \mathcal{F}, Q, Q_f, \Delta \rangle \) be a tree automaton, \( R \) a TRS and \( E \) a set of equations. The automaton \( A \) is said to be \( R/E \)-coherent if

\[
\forall q \in Q : \exists s \in \mathcal{T}(\mathcal{F}) : \\
\quad \quad s \xrightarrow{\phi}_A^* \ q \land [\forall t \in \mathcal{T}(\mathcal{F}) : (t \xrightarrow{\phi}_A^* q \Rightarrow s \equiv_E t) \land (t \xrightarrow{A^*} q \Rightarrow s \xrightarrow{R/E}_t^*)]
\]

Thomas Genet (IRISA)  
Reachability Analysis of Rewriting  
51 / 54
## Benchmarks

<table>
<thead>
<tr>
<th></th>
<th>Combinatory</th>
<th>NSPK</th>
<th>View-Only</th>
<th>Java prog. 1</th>
<th>Java prog. 2</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>TRS nb of rules</strong></td>
<td>1</td>
<td>13</td>
<td>15</td>
<td>279</td>
<td>303</td>
</tr>
<tr>
<td><strong>Initial Aut. size</strong></td>
<td>43 / 23</td>
<td>14 / 4</td>
<td>21 / 18</td>
<td>26 / 49</td>
<td>33 / 33</td>
</tr>
<tr>
<td><strong>Timbuk 2.2 :</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Final Aut. size</td>
<td>8043 / 23</td>
<td>151 / 16</td>
<td>730 / 74</td>
<td>1127 / 334</td>
<td>751 / 335</td>
</tr>
<tr>
<td>Time (secs)</td>
<td><strong>51.1</strong></td>
<td><strong>19.7</strong></td>
<td><strong>6420</strong></td>
<td><strong>25266</strong></td>
<td><strong>37387</strong></td>
</tr>
<tr>
<td><strong>Timbuk 3.0 :</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Final Aut. size</td>
<td>8043 / 23</td>
<td>259 / 104</td>
<td>353 / 100</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Time (secs)</td>
<td><strong>60.1</strong></td>
<td><strong>3.1</strong></td>
<td><strong>2452</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Tom-based :</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Final Aut. size</td>
<td>8043 / 23</td>
<td>171 / 21</td>
<td>938 / 89</td>
<td>1974 / 637</td>
<td>1611 / 672</td>
</tr>
<tr>
<td>Time (secs)</td>
<td><strong>5.9</strong></td>
<td><strong>5.9</strong></td>
<td><strong>150</strong></td>
<td><strong>360</strong></td>
<td><strong>303</strong></td>
</tr>
<tr>
<td><strong>Bddbddd-based :</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Time (secs)</td>
<td><strong>0.008</strong></td>
<td><strong>2.9</strong></td>
<td><strong>3.3</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Applications: Java bytecode verification (II)

Proving safety properties on Java bytecode using reachability analysis

<table>
<thead>
<tr>
<th>Java Source .java</th>
<th>Java Byte Code .class</th>
</tr>
</thead>
<tbody>
<tr>
<td>class TestList{</td>
<td>public static void main(java.lang.String[]);</td>
</tr>
<tr>
<td>public static void main(String[] argv){</td>
<td>Code:</td>
</tr>
<tr>
<td>List lpos=null;</td>
<td>0:   aconst_null</td>
</tr>
<tr>
<td>InvList lneg=null;</td>
<td>1:   astore_1</td>
</tr>
<tr>
<td>int x;</td>
<td>2:   aconst_null</td>
</tr>
<tr>
<td>boolean pos;</td>
<td>3:   astore_2</td>
</tr>
<tr>
<td>pos= true;</td>
<td>4:   iconst_1</td>
</tr>
<tr>
<td>try {x=System.in.read();}</td>
<td>5:   istore_4</td>
</tr>
<tr>
<td>catch(java.io.IOException e){x=0;}</td>
<td>7:   getstatic #2; // #2 = java/io/InputStream</td>
</tr>
<tr>
<td>while (x != -1){</td>
<td>10: invokevirtual #3; // invokevirtual</td>
</tr>
<tr>
<td>if (pos) {lpos= new List(x, lpos);</td>
<td>13: istore_3</td>
</tr>
<tr>
<td>pos=false;}</td>
<td>...</td>
</tr>
<tr>
<td>else {lneg= new InvList(x, lneg);</td>
<td>47: new</td>
</tr>
<tr>
<td>pos=true;}</td>
<td>50: dup</td>
</tr>
<tr>
<td>try {x=System.in.read();}</td>
<td>51: iload_3</td>
</tr>
<tr>
<td>catch(java.io.IOException e){x=0;}</td>
<td></td>
</tr>
<tr>
<td>}</td>
<td></td>
</tr>
<tr>
<td>}</td>
<td></td>
</tr>
</tbody>
</table>
Encoding JVM semantics and bytecode into rewriting (II)

Encoding of an add bytecode

$$\begin{align*}
\text{add} : & \quad \frac{(m, pc, x :: y :: s, l)}{(m, pc + 1, x + y :: s, l)}
\end{align*}$$
Encoding JVM semantics and bytecode into rewriting (II)

Encoding of an add bytecode

1. Associate add bytecode to \( m, pc \)

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

\[
\text{public static void foo(...) ... 11 : add}
\]

\[
\text{frame(\text{foo,11,s,l}) \rightarrow xframe(\text{add,foo,11,s,l})}
\]
Encoding JVM semantics and bytecode into rewriting (II)

Encoding of an add bytecode

```
add : (m, pc, x :: y :: s, l) → (m, pc + 1, x + y :: s, l)
```

1. Associate add bytecode to \( m, pc \)

   ```
   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)
   ```

---

public static void foo(...)
...
11 : add
Encoding JVM semantics and bytecode into rewriting (II)

Encoding of an add bytecode

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

1. Associate add bytecode to \( m, pc \)

\[
\text{public static void foo(...)}
\]

\[
\begin{align*}
11 : & \quad \text{add} \\
\text{frame}(\text{foo}, 11, s, l) & \rightarrow \text{xframe}(\text{add}, \text{foo}, 11, s, l)
\end{align*}
\]

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

\[
\text{xframe}(\text{add}, m, pc, \text{stack}(y, \text{stack}(x, s)), l) \rightarrow \text{xframe}(\text{xadd}(x, y), m, pc, s, l)
\]

3. Compute \( (x + y) \)

\[
\begin{align*}
\text{xadd}(\ldots) & \rightarrow \ldots \\
\ldots & \rightarrow \text{result}(x)
\end{align*}
\]
Encoding JVM semantics and bytecode into rewriting (II)

Encoding of an add bytecode

\[
\text{add : } (m, pc, x :: y :: s, l) \xrightarrow{} (m, pc + 1, x + y :: s, l)
\]

1. Associate add bytecode to \( m, pc \)

\[
\text{frame}(\text{foo,11,s,l}) \rightarrow \text{xframe}(\text{add,foo,11,s,l})
\]

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

\[
\text{xframe}(\text{add,m,pc,stack}(y,\text{stack}(x,s)),l) \rightarrow \text{xframe}(\text{xadd}(x,y),m,pc,s,l)
\]

3. Compute \((x + y)\)

\[
\text{xadd}(\ldots) \rightarrow \ldots \\
\ldots \rightarrow \text{result}(x)
\]

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

\[
\text{xframe}(\text{result}(x),m,pc,s,l) \rightarrow \text{frame}(m,\text{next}(pc),\text{stack}(x,s),l)
\]