Automated Termination Analysis of Term Rewriting

Carsten Fuhs

Birkbeck, University of London

13th International School on Rewriting 2022

Advanced Track

Tbilisi, Georgia

19 & 20 September 2022

https://www.dcs.bbk.ac.uk/~carsten/isr2022/
Why Analyse Termination?
Why Analyse Termination?

1. **Program**: produces result
Why Analyse Termination?

1. **Program**: produces result
2. **Input handler**: system reacts
Why Analyse Termination?

1. **Program**: produces result
2. **Input handler**: system reacts
3. **Mathematical proof**: the induction is valid
Why Analyse Termination?

1. **Program**: produces result
2. **Input handler**: system reacts
3. **Mathematical proof**: the induction is valid
4. **Biological process**: reaches a stable state
Why Analyse Termination?

1. **Program**: produces result
2. **Input handler**: system reacts
3. **Mathematical proof**: the induction is valid
4. **Biological process**: reaches a stable state

Variations of the same problem:

- 2. special case of 1
- 3. can be interpreted as 1
- 4. probabilistic version of 1
Why Analyse Termination?

1. **Program**: produces result
2. **Input handler**: system reacts
3. **Mathematical proof**: the induction is valid
4. **Biological process**: reaches a stable state

Variations of the same problem:
- 2. special case of 1
- 3. can be interpreted as 1
- 4. probabilistic version of 1

2011: PHP and Java issues with floating-point number parser
The Bad News

Theorem (Turing 1936)

*The question if a given program terminates on a fixed input is undecidable.*
The Bad News

**Theorem (Turing 1936)**

The question if a given program terminates on a fixed input is undecidable.

- We want to solve the (harder) question if a given program terminates on all inputs.
The Bad News

Theorem (Turing 1936)

*The question if a given program terminates on a fixed input is undecidable.*

- We want to solve the (harder) question if a given program terminates on all inputs.
- That’s not even semi-decidable!
The Bad News

Theorem (Turing 1936)

The question if a given program terminates on a fixed input is undecidable.

- We want to solve the (harder) question if a given program terminates on all inputs.
- That’s not even semi-decidable!
- But, fear not . . .
Termination Analysis, Classically

Turing 1949

“Finally the checker has to verify that the process comes to an end. [...] This may take the form of a quantity which is asserted to decrease continually and vanish when the machine stops.”
“Finally the checker has to verify that the process comes to an end. [...] This may take the form of a quantity which is asserted to decrease continually and vanish when the machine stops.”

1. Find ranking function $f$ (“quantity”)
“Finally the checker has to verify that the process comes to an end. [...] This may take the form of a quantity which is asserted to decrease continually and vanish when the machine stops.”

1. Find **ranking function** $f$ (“quantity”)

2. Prove $f$ to have a **lower bound** (“vanish when the machine stops”)

---

Termination Analysis, Classically
“Finally the checker has to verify that the process comes to an end. [...] This may take the form of a quantity which is asserted to decrease continually and vanish when the machine stops.”

1. Find ranking function \( f \) (“quantity”)
2. Prove \( f \) to have a lower bound (“vanish when the machine stops”)
3. Prove that \( f \) decreases over time
“Finally the checker has to verify that the process comes to an end. [...] This may take the form of a quantity which is asserted to decrease continually and vanish when the machine stops.”

1. Find **ranking function** \( f \) (“quantity”)
2. Prove \( f \) to have a **lower bound** (“vanish when the machine stops”)
3. Prove that \( f \) **decreases** over time

Example (Termination can be simple)

\[
\textbf{while } x > 0: \\
\quad x = x - 1
\]
Question: Does program $P$ terminate?
Question: Does program $P$ terminate?

Approach: Encode termination proof template to logical constraint $\varphi$, ask SMT solver
Question: Does program $P$ terminate?

Approach: Encode termination proof template to logical constraint $\varphi$, ask SMT solver

$\rightarrow$ SMT = SATisfiability Modulo Theories, solve constraints like

$$b > 0 \quad \land \quad (4ab - 7b^2 > 1 \quad \lor \quad 3a + c \geq b^3)$$
Question: Does program $P$ terminate?

Approach: Encode termination proof template to logical constraint $\varphi$, ask SMT solver

$\rightarrow \text{SMT} = \text{SAT} \text{isfiability Modulo Theories, solve constraints like}$

$$b > 0 \land (4ab - 7b^2 > 1 \lor 3a + c \geq b^3)$$

Answer:
Question: Does program $P$ terminate?

Approach: Encode termination proof template to logical constraint $\varphi$, ask SMT solver

$$\rightarrow \text{SMT} = \text{SATisfiability Modulo Theories}, \text{solve constraints like}$$

$$b > 0 \land (4ab - 7b^2 > 1 \lor 3a + c \geq b^3)$$

Answer:

- $\varphi$ satisfiable, model $M$ (e.g., $a = 3, b = 1, c = 1$):
  - $\Rightarrow P$ terminating, $M$ fills in the gaps in the termination proof
Question: Does program $P$ terminate?

Approach: Encode termination proof template to logical constraint $\varphi$, ask SMT solver

$\rightarrow$ **SMT = SATisfiability Mduelo Theories**, solve constraints like

$$b > 0 \land (4ab - 7b^2 > 1 \lor 3a + c \geq b^3)$$

Answer:

1. $\varphi$ satisfiable, model $M$ (e.g., $a = 3, b = 1, c = 1$):
   $\Rightarrow P$ terminating, $M$ fills in the gaps in the termination proof

2. $\varphi$ unsatisfiable:
   $\Rightarrow$ termination status of $P$ unknown
   $\Rightarrow$ try a different template (proof technique)
Question: Does program $P$ terminate?

Approach: Encode termination proof template to logical constraint $\varphi$, ask SMT solver

$\rightarrow$ SMT = SATisfiability Modulo Theories, solve constraints like

$$b > 0 \land (4ab - 7b^2 > 1 \lor 3a + c \geq b^3)$$

Answer:

1. $\varphi$ satisfiable, model $M$ (e.g., $a = 3, b = 1, c = 1$):
   $\Rightarrow$ $P$ terminating, $M$ fills in the gaps in the termination proof

2. $\varphi$ unsatisfiable:
   $\Rightarrow$ termination status of $P$ unknown
   $\Rightarrow$ try a different template (proof technique)

In practice:

- Encode only one proof step at a time
  $\rightarrow$ try to prove only part of the program terminating
- Repeat until the whole program is proved terminating
I. Termination Proving for Rewrite Systems
1. Term Rewrite Systems (TRSs)
2. Logically Constrained TRSs (LCTRSs)
3. Certification of Termination Proofs

II. Beyond Termination of Rewriting
1. Proving Program Termination via Rewrite Systems: Java
2. Finding Complexity Bounds for TRSs
I. Termination Analysis of Rewriting
I.1 Termination Analysis of Term Rewrite Systems
What’s Term Rewriting?

Syntactic approach for reasoning in equational first-order logic

Core functional programming language without many restrictions
(and features) of “real” FP:
- first-order (usually)
- no fixed evaluation strategy
- non-determinism!
- no fixed order of rules to apply (Haskell: top to bottom)
- non-determinism!
- untyped (unless you really want types)
- no pre-defined data structures (integers, arrays, ...)

Example (Term Rewrite System (TRS) $R$)

\[
\text{double}(0) \rightarrow 0 \\
\text{double}(s(x)) \rightarrow s(s(\text{double}(x)))
\]

Compute “double of 3 is 6”:

\[
\text{double}(s(s(s(s(0)))))
\]

\[
R \rightarrow s(s(s(s(s(\text{double}(s(s(s(0)))))))))
\]
What’s Term Rewriting?

Syntactic approach for reasoning in equational first-order logic

Example (Term Rewrite System (TRS) R

double (0) − → 0
double (s (x)) − → s (s (double (x)))

Compute “double of 3 is 6”:
double (s (s (s (s (s (s (s (s (s (0))))))))))
What’s Term Rewriting?

Syntactic approach for reasoning in equational first-order logic

Core functional programming language without many restrictions (and features) of “real” FP:
What’s Term Rewriting?

Syntactic approach for reasoning in equational first-order logic

Core functional programming language without many restrictions (and features) of “real” FP:
- first-order (usually)
- no fixed evaluation strategy \(\rightarrow\) non-determinism!
- no fixed order of rules to apply (Haskell: top to bottom) \(\rightarrow\) non-determinism!
- untyped (unless you really want types)
- no pre-defined data structures (integers, arrays, . . .)
What’s Term Rewriting?

Syntactic approach for reasoning in equational first-order logic

Core functional programming language without many restrictions (and features) of “real” FP:

- first-order (usually)
- no fixed evaluation strategy $\rightarrow$ non-determinism!
- no fixed order of rules to apply (Haskell: top to bottom) $\rightarrow$ non-determinism!
- untyped (unless you really want types)
- no pre-defined data structures (integers, arrays, ...)

Example (Term Rewrite System (TRS) $\mathcal{R}$)

- $\text{double}(0) \rightarrow 0$
- $\text{double}(s(x)) \rightarrow s(s(\text{double}(x)))$
What’s Term Rewriting?

Syntactic approach for reasoning in equational first-order logic

Core functional programming language without many restrictions (and features) of “real” FP:
- first-order (usually)
- no fixed evaluation strategy $\rightarrow$ non-determinism!
- no fixed order of rules to apply (Haskell: top to bottom) $\rightarrow$ non-determinism!
- untyped (unless you really want types)
- no pre-defined data structures (integers, arrays, ...)

Example (Term Rewrite System (TRS) $\mathcal{R}$)

Compute “double of 3 is 6”:
$\text{double}(s(s(s(s(0)))))$

<table>
<thead>
<tr>
<th>Term</th>
<th>Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{double}(0)$</td>
<td>$\rightarrow 0$</td>
</tr>
<tr>
<td>$\text{double}(s(x))$</td>
<td>$\rightarrow s(s(\text{double}(x)))$</td>
</tr>
</tbody>
</table>
What’s Term Rewriting?

Syntactic approach for reasoning in equational first-order logic

Core functional programming language without many restrictions (and features) of “real” FP:

- first-order (usually)
- no fixed evaluation strategy \(\rightarrow\) non-determinism!
- no fixed order of rules to apply (Haskell: top to bottom) \(\rightarrow\) non-determinism!
- untyped (unless you really want types)
- no pre-defined data structures (integers, arrays, ...)

Example (Term Rewrite System (TRS) \(\mathcal{R}\))

\[
\begin{align*}
\text{double}(0) & \rightarrow 0 \\
\text{double}(s(x)) & \rightarrow s(s(\text{double}(x)))
\end{align*}
\]

Compute “double of 3 is 6”:

\[
\begin{align*}
\text{double}(s(s(s(0)))) & \\
\rightarrow_{\mathcal{R}} s(s(\text{double}(s(s(0)))))
\end{align*}
\]
What’s Term Rewriting?

Syntactic approach for reasoning in equational first-order logic

Core functional programming language without many restrictions (and features) of “real” FP:
- first-order (usually)
- no fixed evaluation strategy \( \rightarrow \) non-determinism!
- no fixed order of rules to apply (Haskell: top to bottom) \( \rightarrow \) non-determinism!
- untyped (unless you really want types)
- no pre-defined data structures (integers, arrays, ...)

Example (Term Rewrite System (TRS) \( R \))

\[
\begin{align*}
\text{double}(0) & \rightarrow 0 \\
\text{double}(s(x)) & \rightarrow s(s(\text{double}(x)))
\end{align*}
\]

Compute “double of 3 is 6”:
\[
\begin{align*}
\text{double}(s(s(s(0)))) & \\
\rightarrow_R & s(s(\text{double}(s(s(0))))) \\
\rightarrow_R & s(s(s(s(\text{double}(s(0))))))
\end{align*}
\]
What’s Term Rewriting?

Syntactic approach for reasoning in equational first-order logic

Core functional programming language without many restrictions (and features) of “real” FP:
  - first-order (usually)
  - no fixed evaluation strategy → non-determinism!
  - no fixed order of rules to apply (Haskell: top to bottom) → non-determinism!
  - untyped (unless you really want types)
  - no pre-defined data structures (integers, arrays, ...)

Example (Term Rewrite System (TRS) \( R \))

Compute “double of 3 is 6”:

\[
\begin{align*}
double(0) &\rightarrow 0 \\
double(s(x)) &\rightarrow s(s(double(x)))
\end{align*}
\]

\[
\begin{align*}
double(s(s(s(0)))) &\rightarrow R s(s(s(double(s(0)))))) \\
&\rightarrow R s(s(s(s(s(s(s(double(s(0)))))))))) \\
&\rightarrow R s(s(s(s(s(s(s(s(s(s(s(0)))))))))))
\end{align*}
\]
What’s Term Rewriting?

Syntactic approach for reasoning in equational first-order logic

Core functional programming language without many restrictions (and features) of “real” FP:
- first-order (usually)
- no fixed evaluation strategy → non-determinism!
- no fixed order of rules to apply (Haskell: top to bottom) → non-determinism!
- untyped (unless you really want types)
- no pre-defined data structures (integers, arrays, ...)

Example (Term Rewrite System (TRS) $\mathcal{R}$)

\[
\begin{align*}
\text{double}(0) & \rightarrow 0 \\
\text{double}(s(x)) & \rightarrow s(s(\text{double}(x)))
\end{align*}
\]

Compute “double of 3 is 6”:

\[
\begin{align*}
\text{double}(s(s(s(0)))) & \\
\rightarrow_{\mathcal{R}} & s(s(\text{double}(s(s(0)))))) \\
\rightarrow_{\mathcal{R}} & s(s(s(s(\text{double}(s(s(0))))))) \\
\rightarrow_{\mathcal{R}} & s(s(s(s(s(s(\text{double}(s(s(0))))))))))
\end{align*}
\]
What’s Term Rewriting?

Syntactic approach for reasoning in equational first-order logic

Core functional programming language without many restrictions (and features) of “real” FP:

- first-order (usually)
- no fixed evaluation strategy → non-determinism!
- no fixed order of rules to apply (Haskell: top to bottom) → non-determinism!
- untyped (unless you really want types)
- no pre-defined data structures (integers, arrays, ...)

Example (Term Rewrite System (TRS) $\mathcal{R}$)

\[
\begin{align*}
\text{double}(0) & \rightarrow 0 \\
\text{double}(s(x)) & \rightarrow s(s(\text{double}(x)))
\end{align*}
\]

Compute “double of 3 is 6”:

\[
\begin{align*}
\text{double}(s^3(0)) & \rightarrow_R s^2(\text{double}(s^2(0))) \\
& \rightarrow_R s^4(\text{double}(s(0))) \\
& \rightarrow_R s^6(\text{double}(0)) \\
& \rightarrow_R s^6(0)
\end{align*}
\]
Why Care about Termination of Term Rewriting?

- Termination needed by theorem provers
Why Care about Termination of Term Rewriting?

- Termination needed by theorem provers
- Translate program $P$ with inductive data structures (trees) to TRS, represent data structures as terms
  $\Rightarrow$ Termination of TRS implies termination of $P$
Why Care about Termination of Term Rewriting?

- Termination needed by theorem provers

- Translate program $P$ with inductive data structures (trees) to TRS, represent data structures as terms

$\Rightarrow$ Termination of TRS implies termination of $P$

- Logic programming: Prolog
Why Care about Termination of Term Rewriting?

- Termination needed by theorem provers

- Translate program $P$ with inductive data structures (trees) to TRS, represent data structures as terms

  $\Rightarrow$ Termination of TRS implies termination of $P$

- Logic programming: Prolog 
  [van Raamsdonk, *ICLP* '97; Schneider-Kamp et al, *TOCL* '09; Giesl et al, *PPDP* '12]

Why Care about Termination of Term Rewriting?

- Termination needed by theorem provers

- Translate program $P$ with inductive data structures (trees) to TRS, represent data structures as terms

  $\Rightarrow$ Termination of TRS implies termination of $P$

- Logic programming: Prolog

- (Lazy) functional programming: Haskell [Giesl et al, *TOPLAS ‘11*]

- Object-oriented programming: Java [Otto et al, *RTA ‘10*]
Termination via Reduction Orders: Polynomial Interpretations

Termination: no infinite evaluation sequences $t_1 \rightarrow_{\mathcal{R}} t_2 \rightarrow_{\mathcal{R}} t_3 \rightarrow_{\mathcal{R}} \ldots$
Termination via Reduction Orders: Polynomial Interpretations

Termination: no infinite evaluation sequences \( t_1 \rightarrow_R t_2 \rightarrow_R t_3 \rightarrow_R \ldots \)

Prove termination of \( R \) via reduction order \( \succ \) on terms with \( R \subseteq \succ \):
- well-founded
- transitive
- monotone (closed under contexts)
- stable (closed under substitutions)
Termination via Reduction Orders: Polynomial Interpretations

Termination: no infinite evaluation sequences $t_1 \rightarrow \mathcal{R} t_2 \rightarrow \mathcal{R} t_3 \rightarrow \mathcal{R} \ldots$

Prove termination of $\mathcal{R}$ via reduction order $\succ$ on terms with $\mathcal{R} \subseteq \succ$:
- well-founded
- transitive
- monotone (closed under contexts)
- stable (closed under substitutions)

Get $\succ$ via polynomial interpretation $\llbracket \cdot \rrbracket$ over $\mathbb{N}$ [Lankford '75]
Termination via Reduction Orders: Polynomial Interpretations

Termination: no infinite evaluation sequences \( t_1 \to_R t_2 \to_R t_3 \to_R \ldots \)

Prove termination of \( \mathcal{R} \) via **reduction order** \( \succ \) on terms with \( \mathcal{R} \subseteq \succ \):
- well-founded
- transitive
- monotone (closed under contexts)
- stable (closed under substitutions)

Get \( \succ \) via **polynomial interpretation** \([\cdot]\) over \( \mathbb{N} \) [Lankford '75]

Idea: \( \ell \succ r \iff [\ell] > [r] \) \( \succ \) boils down to \( > \) over \( \mathbb{N} \)
Termination via Reduction Orders: Polynomial Interpretations

Termination: no infinite evaluation sequences $t_1 \xrightarrow{\mathcal{R}} t_2 \xrightarrow{\mathcal{R}} t_3 \xrightarrow{\mathcal{R}} \ldots$

Prove termination of $\mathcal{R}$ via reduction order $\succ$ on terms with $\mathcal{R} \subseteq \succ$:
- well-founded
- transitive
- monotone (closed under contexts)
- stable (closed under substitutions)

Get $\succ$ via polynomial interpretation $[\cdot]$ over $\mathbb{N}$ [Lankford ’75]

Idea: $\ell \succ r \iff [\ell] > [r]$ $\succ$ boils down to $>$ over $\mathbb{N}$

Example (double)

- $\text{double}(0) \succ 0$
- $\text{double}(s(x)) \succ s(s(\text{double}(x)))$
Termination via Reduction Orders: Polynomial Interpretations

Termination: no infinite evaluation sequences \( t_1 \rightarrow_\mathcal{R} t_2 \rightarrow_\mathcal{R} t_3 \rightarrow_\mathcal{R} \ldots \)

Prove termination of \( \mathcal{R} \) via reduction order \( \succ \) on terms with \( \mathcal{R} \subseteq \succ \):
- well-founded
- transitive
- monotone (closed under contexts)
- stable (closed under substitutions)

Get \( \succ \) via polynomial interpretation \([\cdot]\) over \( \mathbb{N} \) [Lankford ’75]

Idea: \( \ell \succ r \iff [\ell] > [r] \)

\( \succ \) boils down to \( \succ \) over \( \mathbb{N} \)

**Example (double)**

\[
\begin{align*}
double(0) & \succ 0 \\
double(s(x)) & \succ s(s(double(x)))
\end{align*}
\]

Example: \([\text{double}] (x) = 3 \cdot x\), \( [s](x) = x + 1 \), \( [0] = 1 \)
Termination via Reduction Orders: Polynomial Interpretations

Termination: no infinite evaluation sequences \( t_1 \rightarrow \mathcal{R} \ t_2 \rightarrow \mathcal{R} \ t_3 \rightarrow \mathcal{R} \ldots \)

Prove termination of \( \mathcal{R} \) via reduction order \( \succ \) on terms with \( \mathcal{R} \subseteq \succ \):
- well-founded
- transitive
- monotone (closed under contexts)
- stable (closed under substitutions)

Get \( \succ \) via polynomial interpretation \( [\cdot] \) over \( \mathbb{N} \) [Lankford ’75]

Idea: \( \ell \succ r \iff [\ell] > [r] \) \( \succ \) boils down to \( > \) over \( \mathbb{N} \)

Example (double)

\[
\begin{align*}
double(0) & \succ 0 \\
double(s(x)) & \succ s(s(double(x)))
\end{align*}
\]

Example:

\[
\begin{align*}
[double](x) &= 3 \cdot x, \\
[s](x) &= x + 1, \\
[0] &= 1
\end{align*}
\]

Extend to terms:
- \([x] = x\)
- \([f(t_1, \ldots, t_n)] = [f][t_1, \ldots, t_n]\)
Termination via Reduction Orders: Polynomial Interpretations

Termination: no infinite evaluation sequences \( t_1 \rightarrow R t_2 \rightarrow R t_3 \rightarrow R \ldots \)

Prove termination of \( R \) via reduction order \( \succ \) on terms with \( R \subseteq \succ \):
- well-founded
- transitive
- monotone (closed under contexts)
- stable (closed under substitutions)

Get \( \succ \) via polynomial interpretation \([ \cdot ]\) over \( \mathbb{N} \) [Lankford '75]

Idea: \( \ell \succ r \iff [\ell] \succ [r] \)

\( \succ \) boils down to \( > \) over \( \mathbb{N} \)

Example (double)

| \text{double}(0) | \succ | 0 | 3 \succ 1 |
|------------------|-------|------------------|
| \text{double}(s(x)) | \succ | s(s(\text{double}(x))) | 3 \cdot x + 3 \succ 3 \cdot x + 2 |

Example:
\[ [\text{double}](x) = 3 \cdot x, \quad [s](x) = x + 1, \quad [0] = 1 \]

Extend to terms:
- \([x] = x\)
- \([f(t_1, \ldots, t_n)] = [f]([t_1], \ldots, [t_n])\)
Termination via Reduction Orders: Polynomial Interpretations

Termination: no infinite evaluation sequences \( t_1 \rightarrow_R t_2 \rightarrow_R t_3 \rightarrow_R \ldots \)

Prove termination of \( R \) via reduction order \( \succ \) on terms with \( R \subseteq \succ \):

- well-founded
- transitive
- monotone (closed under contexts)
- stable (closed under substitutions)

Get \( \succ \) via polynomial interpretation \( [\cdot] \) over \( \mathbb{N} \) [Lankford '75]

Idea: \( \ell \succ r \iff [\ell] > [r] \)

\( \succ \) boils down to \( > \) over \( \mathbb{N} \)

Example (double)

| \( \text{double}(0) \) | \( 0 \) |
| \( \text{double}(s(x)) \) | \( s(s(\text{double}(x))) \) |
| 3 | 1 |
| \( 3 \cdot x + 3 \) | \( 3 \cdot x + 2 \) |

Example:

\( [\text{double}](x) = 3 \cdot x \), \( [s](x) = x + 1 \), \( [0] = 1 \)

Extend to terms:

- \( [x] = x \)
- \( [f(t_1, \ldots, t_n)] = [f][t_1], \ldots, [t_n] \)

In practice: use polynomial interpretations together with Dependency Pairs
Example (Division)

\[ \mathcal{R} = \left\{ \begin{array}{ll} 
\text{minus}(x, 0) & \rightarrow x \\
\text{minus}(s(x), s(y)) & \rightarrow \text{minus}(x, y) \\
\text{quot}(0, s(y)) & \rightarrow 0 \\
\text{quot}(s(x), s(y)) & \rightarrow s(\text{quot}(\text{minus}(x, y), s(y))) 
\end{array} \right. \]
Example (Division)

\[ R = \begin{cases} 
\text{minus}(x, 0) & \rightarrow x \\
\text{minus}(s(x), s(y)) & \rightarrow \text{minus}(x, y) \\
\text{quot}(0, s(y)) & \rightarrow 0 \\
\text{quot}(s(x), s(y)) & \rightarrow s(\text{quot}(\text{minus}(x, y), s(y))) 
\end{cases} \]

Show termination using Dependency Pairs [Arts, Giesl, \textit{TCS '00}]
Example (Division)

\[ R = \begin{cases} 
\text{minus}(x, 0) & \rightarrow x \\
\text{minus}(s(x), s(y)) & \rightarrow \text{minus}(x, y) \\
\text{quot}(0, s(y)) & \rightarrow 0 \\
\text{quot}(s(x), s(y)) & \rightarrow s(\text{quot}(\text{minus}(x, y), s(y)))
\end{cases} \]

\[ P = \begin{cases} 
\text{minus}^\#(s(x), s(y)) & \rightarrow \text{minus}^\#(x, y) \\
\text{quot}^\#(s(x), s(y)) & \rightarrow \text{minus}^\#(x, y) \\
\text{quot}^\#(s(x), s(y)) & \rightarrow \text{quot}^\#(\text{minus}(x, y), s(y))
\end{cases} \]

Show termination using Dependency Pairs [Arts, Giesl, TCS ’00]

- For TRS \( R \) build dependency pairs \( P \) (\( \sim \) function calls)
- Show: No \( \infty \) call sequence with \( P \) (eval of \( P \)'s args via \( R \))
Example (Division)

\[
R = \left\{
\begin{array}{l}
\text{minus}(x, 0) \rightarrow x \\
\text{minus}(s(x), s(y)) \rightarrow \text{minus}(x, y) \\
\text{quot}(0, s(y)) \rightarrow 0 \\
\text{quot}(s(x), s(y)) \rightarrow s(\text{quot}(\text{minus}(x, y), s(y)))
\end{array}
\right.
\]

\[
P = \left\{
\begin{array}{l}
\text{minus}^#(s(x), s(y)) \rightarrow \text{minus}^#(x, y) \\
\text{quot}^#(s(x), s(y)) \rightarrow \text{minus}^#(x, y) \\
\text{quot}^#(s(x), s(y)) \rightarrow \text{quot}^#(\text{minus}(x, y), s(y))
\end{array}
\right.
\]

Show termination using Dependency Pairs [Arts, Giesl, TCS ’00]
- For TRS \( R \) build dependency pairs \( P \) (~ function calls)
- Show: No \( \infty \) call sequence with \( P \) (eval of \( P \)'s args via \( R \))
- Dependency Pair Framework [Giesl et al, JAR ’06] (simplified):
Example (Division)

\[ \mathcal{R} = \begin{cases} 
\text{minus}(x, 0) & \rightarrow x \\
\text{minus}(s(x), s(y)) & \rightarrow \text{minus}(x, y) \\
\text{quot}(0, s(y)) & \rightarrow 0 \\
\text{quot}(s(x), s(y)) & \rightarrow s(\text{quot}(\text{minus}(x, y), s(y))) 
\end{cases} \]

\[ \mathcal{P} = \begin{cases} 
\text{minus}^\#(s(x), s(y)) & \rightarrow \text{minus}^\#(x, y) \\
\text{quot}^\#(s(x), s(y)) & \rightarrow \text{minus}^\#(x, y) \\
\text{quot}^\#(s(x), s(y)) & \rightarrow \text{quot}^\#(\text{minus}(x, y), s(y)) 
\end{cases} \]

Show termination using Dependency Pairs [Arts, Giesl, TCS ’00]

- For TRS \( \mathcal{R} \) build dependency pairs \( \mathcal{P} \) (\( \sim \) function calls)
- Show: No \( \infty \) call sequence with \( \mathcal{P} \) (eval of \( \mathcal{P} \)'s args via \( \mathcal{R} \))
- Dependency Pair Framework [Giesl et al, JAR ’06] (simplified): while \( \mathcal{P} \neq \emptyset \) :
Show termination using Dependency Pairs [Arts, Giesl, TCS ’00]

- For TRS $\mathcal{R}$ build dependency pairs $\mathcal{P}$ ($\sim$ function calls)
- Show: No $\infty$ call sequence with $\mathcal{P}$ (eval of $\mathcal{P}$’s args via $\mathcal{R}$)
- Dependency Pair Framework [Giesl et al, JAR ’06] (simplified): while $\mathcal{P} \neq \emptyset$ :
  - find well-founded order $\succ$ with $\mathcal{P} \cup \mathcal{R} \subseteq \succ$

---

Example (Division)

$\mathcal{R} = \{ \begin{array}{lr}
\text{minus}(x, 0) & \sim x \\
\text{minus}(s(x), s(y)) & \sim \text{minus}(x, y) \\
\text{quot}(0, s(y)) & \sim 0 \\
\text{quot}(s(x), s(y)) & \sim s(\text{quot}(\text{minus}(x, y), s(y)))
\end{array} \}$

$\mathcal{P} = \{ \begin{array}{lr}
\text{minus}^\#(s(x), s(y)) & \sim \text{minus}^\#(x, y) \\
\text{quot}^\#(s(x), s(y)) & \sim \text{minus}^\#(x, y) \\
\text{quot}^\#(s(x), s(y)) & \sim \text{quot}^\#(\text{minus}(x, y), s(y))
\end{array} \}$
Example (Division)

\[ \mathcal{R} = \left\{ \begin{array}{ll}
\text{minus}(x, 0) & \leadsto x \\
\text{minus}(s(x), s(y)) & \leadsto \text{minus}(x, y) \\
\text{quot}(0, s(y)) & \leadsto 0 \\
\text{quot}(s(x), s(y)) & \leadsto s(\text{quot}(\text{minus}(x, y), s(y))) \\
\end{array} \right\} \]

\[ \mathcal{P} = \left\{ \begin{array}{ll}
\text{minus}^\#(s(x), s(y)) & \leadsto \text{minus}^\#(x, y) \\
\text{quot}^\#(s(x), s(y)) & \leadsto \text{minus}^\#(x, y) \\
\text{quot}^\#(s(x), s(y)) & \leadsto \text{quot}^\#(\text{minus}(x, y), s(y)) \\
\end{array} \right\} \]

Show termination using Dependency Pairs [Arts, Giesl, TCS '00]

- For TRS \( \mathcal{R} \) build dependency pairs \( \mathcal{P} \) (∼ function calls)
- Show: No \( \infty \) call sequence with \( \mathcal{P} \) (eval of \( \mathcal{P} \)'s args via \( \mathcal{R} \))
- Dependency Pair Framework [Giesl et al, JAR '06] (simplified):
  - while \( \mathcal{P} \neq \emptyset \):
    - find well-founded order \( \succ \) with \( \mathcal{P} \cup \mathcal{R} \subseteq \succ \)
    - delete \( s \rightarrow t \) with \( s \succ t \) from \( \mathcal{P} \)
### Example (Division)

\[
\mathcal{R} = \begin{cases}
\text{minus}(x, 0) & \mapsto x \\
\text{minus}(s(x), s(y)) & \mapsto \text{minus}(x, y) \\
\text{quot}(0, s(y)) & \mapsto 0 \\
\text{quot}(s(x), s(y)) & \mapsto \text{s(quot(minus(x, y), s(y)))}
\end{cases}
\]

\[
\mathcal{P} = \begin{cases}
\text{minus}^\#(s(x), s(y)) & \mapsto \text{minus}^\#(x, y) \\
\text{quot}^\#(s(x), s(y)) & \mapsto \text{minus}^\#(x, y) \\
\text{quot}^\#(s(x), s(y)) & \mapsto \text{quot}^\#(\text{minus}(x, y), s(y))
\end{cases}
\]

Show termination using Dependency Pairs [Arts, Giesl, TCS ’00]

- For TRS \( \mathcal{R} \) build dependency pairs \( \mathcal{P} \) (∼ function calls)
- Show: No \( \infty \) call sequence with \( \mathcal{P} \) (eval of \( \mathcal{P} \)'s args via \( \mathcal{R} \))
- Dependency Pair Framework [Giesl et al, JAR ’06] (simplified):
  - while \( \mathcal{P} \neq \emptyset \):
    - find well-founded order \( \succ \) with \( \mathcal{P} \cup \mathcal{R} \subseteq \preceq \)
    - delete \( s \rightarrow t \) with \( s \succ t \) from \( \mathcal{P} \)
- Find \( (\preceq, \succ) \) automatically and efficiently
Reduction Pair

$(\preceq, \succ)$ must be a reduction pair:

with $\mathcal{R} \subseteq \preceq$ and $\mathcal{P} \subseteq \succ \cup \preceq$

- Show: No $\infty$ call sequence with $\mathcal{P}$ (eval of $\mathcal{P}$'s args via $\mathcal{R}$)
- Dependency Pair Framework [Giesl et al, *JAR '06*] (simplified):
  - while $\mathcal{P} \neq \emptyset$:
    - find well-founded order $\succ$ with $\mathcal{P} \cup \mathcal{R} \subseteq \preceq$
    - delete $s \rightarrow t$ with $s \succ t$ from $\mathcal{P}$
- Find $(\preceq, \succ)$ automatically and efficiently
Reduction Pair

\((\preceq, \succ)\) must be a reduction pair:

- \(\succ\) a well-founded stable order (monotonicity not needed!)
- \(\preceq\) a monotone quasi-order
- \(\succ\) and \(\preceq\) must be compatible: \(\succ \circ \preceq \subseteq \succ\) or \(\preceq \circ \succ \subseteq \succ\)

with \(\mathcal{R} \subseteq \preceq\) and \(\mathcal{P} \subseteq \succ \cup \preceq\)

- Show: No \(\infty\) call sequence with \(\mathcal{P}\) (eval of \(\mathcal{P}\)'s args via \(\mathcal{R}\))
- Dependency Pair Framework [Giesl et al, JAR '06] (simplified):
  - while \(\mathcal{P} \neq \emptyset\):
    - find well-founded order \(\succ\) with \(\mathcal{P} \cup \mathcal{R} \subseteq \preceq\)
    - delete \(s \rightarrow t\) with \(s \succ t\) from \(\mathcal{P}\)
- Find \((\preceq, \succ)\) automatically and efficiently
(∽, ≽) must be a reduction pair:

- ≽ a well-founded stable order (monotonicity not needed!)
- ∼ a monotone quasi-order
- ≽ and ∼ must be compatible: ≽ ⪯ ⊇ ≽ or ∼ ⪯ ⊇ ∼

with \( R \subseteq ∼ \) and \( P \subseteq ≽ \cup ∼ \)

⇒ [ ⋅ ] may now ignore arguments: \([f](x_1) = 1\) is now allowed!

- Show: No \( ∞ \) call sequence with \( P \) (eval of \( P \)'s args via \( R \))
- Dependency Pair Framework [Giesl et al, JAR '06] (simplified):
  - while \( P \neq \emptyset \):
    - find well-founded order \( ≽ \) with \( P \cup R \subseteq ∼ \)
    - delete \( s \rightarrow t \) with \( s ≽ t \) from \( P \)
  - Find \((∼, ≽)\) automatically and efficiently
Reduction Pair Processor

$(\simeq, \succ)$ must be a reduction pair:

- $\succ$ a well-founded stable order (monotonicity not needed!)
- $\simeq$ a monotone quasi-order
- $\succ$ and $\simeq$ must be compatible: $\succ \circ \simeq \subseteq \succ$ or $\simeq \circ \succ \subseteq \succ$

with $\mathcal{R} \subseteq \simeq$ and $\mathcal{P} \subseteq \succ \cup \simeq$

$\Rightarrow [\cdot]$ may now ignore arguments: $[f](x_1) = 1$ is now allowed!

Reduction Pair Processor: $(\mathcal{P}, \mathcal{R}) \vdash (\mathcal{P} \setminus \succ, \mathcal{R})$

- Show: No $\infty$ call sequence with $\mathcal{P}$ (eval of $\mathcal{P}$’s args via $\mathcal{R}$)
- Dependency Pair Framework [Giesl et al, JAR ’06] (simplified): 
  while $\mathcal{P} \neq \emptyset$ :
    - find well-founded order $\succ$ with $\mathcal{P} \cup \mathcal{R} \subseteq \simeq$
    - delete $s \rightarrow t$ with $s \succ t$ from $\mathcal{P}$
- Find $(\simeq, \succ)$ automatically and efficiently
Example (Constraints for Division)

\[ R = \begin{cases} 
  \text{minus}(x, 0) & \mapsto x \\
  \text{minus}(s(x), s(y)) & \mapsto \text{minus}(x, y) \\
  \text{quot}(0, s(y)) & \mapsto 0 \\
  \text{quot}(s(x), s(y)) & \mapsto s(\text{quot}(\text{minus}(x, y), s(y))) 
\end{cases} \]

\[ P = \begin{cases} 
  \text{minus}^\#(s(x), s(y)) & \mapsto \text{minus}^\#(x, y) \\
  \text{quot}^\#(s(x), s(y)) & \mapsto \text{minus}^\#(x, y) \\
  \text{quot}^\#(s(x), s(y)) & \mapsto \text{quot}^\#(\text{minus}(x, y), s(y)) 
\end{cases} \]
Example (Constraints for Division)

\[ \mathcal{R} = \begin{cases} 
\text{minus}(x, 0) & \leadsto x \\
\text{minus}(s(x), s(y)) & \leadsto \text{minus}(x, y) \\
\text{quot}(0, s(y)) & \leadsto 0 \\
\text{quot}(s(x), s(y)) & \leadsto s(\text{quot}(\text{minus}(x, y), s(y))) 
\end{cases} \]

\[ \mathcal{P} = \begin{cases} 
\text{minus}^\#(s(x), s(y)) & \succ \text{minus}^\#(x, y) \\
\text{quot}^\#(s(x), s(y)) & \succ \text{minus}^\#(x, y) \\
\text{quot}^\#(s(x), s(y)) & \succ \text{quot}^\#(\text{minus}(x, y), s(y)) 
\end{cases} \]

Use interpretation \([ \cdot ]\) over \(\mathbb{N}\) with

\[
\begin{align*}
[\text{quot}^\#](x_1, x_2) &= x_1 \\
[\text{minus}^\#](x_1, x_2) &= x_1 \\
[0] &= 0 \\
[\text{quot}](x_1, x_2) &= x_1 + x_2 \\
[\text{minus}](x_1, x_2) &= x_1 \\
[s](x_1) &= x_1 + 1
\end{align*}
\]

\(\bowtie\) order solves all constraints
Example (Constraints for Division)

\[ R = \begin{cases} 
\text{minus}(x, 0) & \ni x \\
\text{minus}(s(x), s(y)) & \ni \text{minus}(x, y) \\
\text{quot}(0, s(y)) & \ni 0 \\
\text{quot}(s(x), s(y)) & \ni s(\text{quot}(\text{minus}(x, y), s(y))) 
\end{cases} \]

\[ P = \begin{cases} 
\text{minus}^\#(x_1, x_2) = x_1 \\
\text{quot}^\#(x_1, x_2) = x_1 \\
[0] = 0 
\end{cases} \]

Use interpretation \([ \cdot ]\) over \(\mathbb{N}\) with

\[ [\text{quot}^\#](x_1, x_2) = x_1 \quad [\text{quot}](x_1, x_2) = x_1 + x_2 \]

\[ [\text{minus}^\#](x_1, x_2) = x_1 \quad [\text{minus}](x_1, x_2) = x_1 \]

\[ [s](x_1) = x_1 + 1 \]

\(\bowtie\) order solves all constraints

\(\bowtie\) \(P = \emptyset\)

\(\bowtie\) termination of division algorithm proved
Remark

Polynomial interpretations play several roles for program analysis:

Use interpretation $[\cdot]$ over $\mathbb{N}$ with

$$
\begin{align*}
[\text{quot}^\#](x_1, x_2) &= x_1 \\
[\text{minus}^\#](x_1, x_2) &= x_1 \\
[0] &= 0
\end{align*}
$$

$$
\begin{align*}
[\text{quot}](x_1, x_2) &= x_1 + x_2 \\
[\text{minus}](x_1, x_2) &= x_1 \\
[s](x_1) &= x_1 + 1
\end{align*}
$$

$\bowtie$ order solves all constraints

$\bowtie \mathcal{P} = \emptyset$

$\bowtie$ termination of division algorithm proved
Remark

Polynomial interpretations play several roles for program analysis:

- Ranking function: \([\text{quot}^\#]\) and \([\text{minus}^\#]\)

Use interpretation \([\cdot]\) over \(\mathbb{N}\) with

\[
\begin{align*}
[\text{quot}^\#](x_1, x_2) &= x_1 \\
[\text{minus}^\#](x_1, x_2) &= x_1 \\
[0] &= 0
\end{align*}
\]

\[
\begin{align*}
[\text{quot}](x_1, x_2) &= x_1 + x_2 \\
[\text{minus}](x_1, x_2) &= x_1 \\
[s](x_1) &= x_1 + 1
\end{align*}
\]

ówi order solves all constraints

ówi \(P = \emptyset\)

ówi termination of division algorithm proved
Remark

Polynomial interpretations play several roles for program analysis:

- Ranking function: \([\text{quot}^\#]\) and \([\text{minus}^\#]\)
- Summary: \([\text{quot}]\) and \([\text{minus}]\)

Use interpretation \([\cdot]\) over \(\mathbb{N}\) with

\[
\begin{align*}
[\text{quot}^\#](x_1, x_2) &= x_1 \\
[\text{minus}^\#](x_1, x_2) &= x_1 \\
[0] &= 0 \\
[\text{quot}](x_1, x_2) &= x_1 + x_2 \\
[\text{minus}](x_1, x_2) &= x_1 \\
[s](x_1) &= x_1 + 1
\end{align*}
\]

\(\bowtie\) order solves all constraints
\(\bowtie\ P = \emptyset\)
\(\bowtie\) termination of division algorithm proved
Remark

Polynomial interpretations play several roles for program analysis:

- Ranking function: \([\text{quot}^\#]\) and \([\text{minus}^\#]\)
- Summary: \([\text{quot}]\) and \([\text{minus}]\)
- Abstraction (aka norm) for data structures: \([0]\) and \([s]\)

Use interpretation \([\cdot ]\) over \(\mathbb{N}\) with

\[
\begin{align*}
[\text{quot}^\#](x_1, x_2) &= x_1 \\
[\text{minus}^\#](x_1, x_2) &= x_1 \\
[0] &= 0
\end{align*}
\]

\[
\begin{align*}
[\text{quot}](x_1, x_2) &= x_1 + x_2 \\
[\text{minus}](x_1, x_2) &= x_1 \\
[s](x_1) &= x_1 + 1
\end{align*}
\]

\(\bowtie\) order solves all constraints

\(\bowtie\) \(\mathcal{P} = \emptyset\)

\(\bowtie\) termination of division algorithm proved
Task: Solve $\text{minus}(s(x), s(y)) \preceq \text{minus}(x, y)$
Task: Solve \( \text{minus}(s(x), s(y)) \preceq \text{minus}(x, y) \)

1. Fix template polynomials with parametric coefficients, get interpretation template:

\[
\text{[minus]}(x, y) = a_m + b_m x + c_m y, \quad [s](x) = a_s + b_s x
\]
Task: Solve \( \text{minus}(s(x), s(y)) \preceq \text{minus}(x, y) \)

1. Fix template polynomials with parametric coefficients, get interpretation template:

\[
\text{[minus]}(x, y) = a_m + b_m x + c_m y, \quad [s](x) = a_s + b_s x
\]

2. From term constraint to polynomial constraint:

\[
s \preceq t \iff [s] \geq [t]
\]

Here: \( \forall x, y. \ (a_s b_m + a_s c_m) + (b_s b_m - b_m) x + (b_s c_m - c_m) y \geq 0 \)
Task: Solve \( \text{minus}(s(x), s(y)) \preceq \text{minus}(x, y) \)

1. Fix template polynomials with \textbf{parametric coefficients},
get interpretation template:

\[
\text{[minus]}(x, y) = a_m + b_m x + c_m y, \quad [s](x) = a_s + b_s x
\]

2. From term constraint to polynomial constraint:

\[
s \preceq t \leadsto [s] \geq [t]
\]

Here:

\[
\forall x, y. \ (a_s b_m + a_s c_m) + (b_s b_m - b_m) x + (b_s c_m - c_m) y \geq 0
\]

3. Eliminate \( \forall x, y \) by \textbf{absolute positiveness criterion}
[Hong, Jakuš, JAR '98]:

Here:

\[
a_s b_m + a_s c_m \geq 0 \land b_s b_m - b_m \geq 0 \land b_s c_m - c_m \geq 0
\]
Task: Solve $\text{minus}(s(x), s(y)) \preceq \text{minus}(x, y)$

1. Fix template polynomials with **parametric coefficients**, get interpretation template:

   $[\text{minus}](x, y) = a_m + b_m x + c_m y, \quad [s](x) = a_s + b_s x$

2. From term constraint to polynomial constraint:

   $s \preceq t \Leftrightarrow [s] \geq [t]$

   Here: $\forall x, y. \quad (a_s b_m + a_s c_m) + (b_s b_m - b_m) x + (b_s c_m - c_m) y \geq 0$

3. Eliminate $\forall x, y$ by **absolute positiveness criterion** [Hong, Jakuš, JAR ’98]:

   Here: $a_s b_m + a_s c_m \geq 0 \land b_s b_m - b_m \geq 0 \land b_s c_m - c_m \geq 0$
Task: Solve $\text{minus}(s(x), s(y)) \succcurlyeq \text{minus}(x, y)$

1. Fix template polynomials with parametric coefficients, get interpretation template:

\[
\text{[minus]}(x, y) = a_m + b_m x + c_m y, \quad [s](x) = a_s + b_s x
\]

2. From term constraint to polynomial constraint:

\[
s \succcurlyeq t \sim [s] \geq [t]
\]

Here: \( \forall x, y. (a_s b_m + a_s c_m) + (b_s b_m - b_m) x + (b_s c_m - c_m) y \geq 0 \)

3. Eliminate \( \forall x, y \) by absolute positiveness criterion [Hong, Jakuš, JAR '98]:

Here: \( a_s b_m + a_s c_m \geq 0 \land b_s b_m - b_m \geq 0 \land b_s c_m - c_m \geq 0 \)
Task: Solve \( \text{minus}(s(x), s(y)) \preceq \text{minus}(x, y) \)

1. Fix template polynomials with \textbf{parametric coefficients},
   get interpretation template:
   \[
   [\text{minus}](x, y) = a_m + b_m x + c_m y, \quad [s](x) = a_s + b_s x
   \]

2. From term constraint to polynomial constraint:
   \[
   s \preceq t \leadsto [s] \geq [t]
   \]
   Here:
   \[
   \forall x, y. (a_s b_m + a_s c_m) + (b_s b_m - b_m) x + (b_s c_m - c_m) y \geq 0
   \]

3. Eliminate \( \forall x, y \) by \textbf{absolute positiveness criterion} [Hong, Jakuš, \textit{JAR '98}]:
   Here:
   \[
   a_s b_m + a_s c_m \geq 0 \land b_s b_m - b_m \geq 0 \land b_s c_m - c_m \geq 0
   \]
   \textbf{Non-linear} constraints, even for \textbf{linear} interpretations
Task: Solve \( \text{minus}(s(x), s(y)) \preceq \text{minus}(x, y) \)

1. Fix template polynomials with **parametric coefficients**, get interpretation template:
   \[
   [\text{minus}](x, y) = a_m + b_m x + c_m y, \quad [s](x) = a_s + b_s x
   \]

2. From term constraint to polynomial constraint:
   \[
   s \preceq t \Rightarrow [s] \geq [t]
   \]

   Here: \( \forall x, y. (a_s b_m + a_s c_m) + (b_s b_m - b_m) x + (b_s c_m - c_m) y \geq 0 \)

3. Eliminate \( \forall x, y \) by **absolute positiveness criterion** [Hong, Jakuš, *JAR '98*]:
   \[
   \text{Here: } a_s b_m + a_s c_m \geq 0 \land b_s b_m - b_m \geq 0 \land b_s c_m - c_m \geq 0
   \]
   **Non-linear** constraints, even for **linear** interpretations

Task: Show satisfiability of non-linear constraints over \( \mathbb{N} \) (\( \rightarrow \) SMT solver!)

\( \therefore \) **Prove termination** of given term rewrite system
Polynomials with negative coefficients and max-operator [Hirokawa, Middeldorp, IC ’07; Fuhs et al, SAT ’07, RTA ’08]

- can model behaviour of functions more closely: 
  \[ \text{[pred]}(x_1) = \max(x_1 - 1, 0) \]
- automation via encoding to non-linear constraints, more complex Boolean structure
Polynomials with negative coefficients and max-operator [Hirokawa, Middeldorp, IC ’07; Fuhs et al, SAT ’07, RTA ’08]
- can model behaviour of functions more closely:
  \[\text{[pred]}(x_1) = \max(x_1 - 1, 0)\]
- automation via encoding to non-linear constraints, more complex Boolean structure

Polynomials over \(\mathbb{Q}^+\) and \(\mathbb{R}^+\) [Lucas, RAIRO ’05]
- non-integer coefficients increase proving power
- SAT/SMT-based automation [Fuhs et al, AISC ’08; Zankl, Middeldorp, LPAR ’10; Borralleras et al, JAR ’12]
Polynomials with **negative coefficients** and **max-operator**
[Hirokawa, Middeldorp, *IC ’07*; Fuhs et al, *SAT ’07, RTA ’08*]
- can model behaviour of functions more closely:
  \[
  [\text{pred}](x_1) = \max(x_1 - 1, 0)
  \]
- automation via encoding to non-linear constraints, more complex Boolean structure

Polynomials over \( \mathbb{Q}^+ \) and \( \mathbb{R}^+ \) [Lucas, *RAIRO ’05*]
- non-integer coefficients increase proving power

...
Matrix Interpretations

Linear interpretations to vectors $\mathbb{N}^k$, use square matrices as coefficients.
Linear interpretations to vectors $\mathbb{N}^k$, use square matrices as coefficients

**Example** for $k = 2$:

$\mathcal{R} = \{a(a(x)) \rightarrow a(b(a(x)))\}$.
Matrix Interpretations

Linear interpretations to vectors $\mathbb{N}^k$, use square matrices as coefficients

**Example** for $k = 2$:

$\mathcal{R} = \{ a(a(x)) \rightarrow a(b(a(x)))) \}$. Show $[a(a(x))] > [a(b(a(x)))]$ with

$[a]((x_1 \ x_2)) = \begin{pmatrix} 0 \\ 1 \end{pmatrix} + \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$,  

$[b]((x_1 \ x_2)) = \begin{pmatrix} 0 \\ 0 \end{pmatrix} + \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$
Matrix Interpretations

Linear interpretations to vectors $\mathbb{N}^k$, use square matrices as coefficients

**Example** for $k = 2$:

$\mathcal{R} = \{ a(a(x)) \rightarrow a(b(a(x)))) \}$. Show $[a(a(x))] > [a(b(a(x)))]$ with

$[a](\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}) = \begin{pmatrix} 0 \\ 1 \end{pmatrix} + \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$, $[b](\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}) = \begin{pmatrix} 0 \\ 0 \end{pmatrix} + \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$

Compare vectors $\begin{pmatrix} x_1 \\ \ldots \\ x_k \end{pmatrix} > \begin{pmatrix} y_1 \\ \ldots \\ y_k \end{pmatrix}$ by $x_1 > y_1 \land x_2 \geq y_2 \land \ldots \land x_k \geq y_k$
Matrix Interpretations

Linear interpretations to vectors $\mathbb{N}^k$, use square matrices as coefficients

**Example** for $k = 2$:

$R = \{a(a(x)) \rightarrow a(b(a(x))))\}$. Show $[a(a(x))] > [a(b(a(x)))]$ with

$$[a]\left(\begin{array}{c} x_1 \\ x_2 \end{array}\right) = \begin{pmatrix} 0 \\ 1 \end{pmatrix} + \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}, \quad [b]\left(\begin{array}{c} x_1 \\ x_2 \end{array}\right) = \begin{pmatrix} 0 \\ 0 \end{pmatrix} + \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$$

Compare vectors $\begin{pmatrix} x_1 \\ \ldots \\ x_k \end{pmatrix} > \begin{pmatrix} y_1 \\ \ldots \\ y_k \end{pmatrix}$ by $x_1 > y_1 \land x_2 \geq y_2 \land \ldots \land x_k \geq y_k$

$$[a(a(x))] = \begin{pmatrix} 1 \\ 2 \end{pmatrix} + \begin{pmatrix} 2 & 2 \\ 2 & 2 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 1 + 2x_1 + 2x_2 \\ 2 + 2x_1 + 2x_2 \end{pmatrix}$$

$$[a(b(a(x)))] = \begin{pmatrix} 0 \\ 1 \end{pmatrix} + \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 0 + x_1 + x_2 \\ 1 + x_1 + x_2 \end{pmatrix}$$
Matrix Interpretations [Endrullis, Waldmann, Zantema, JAR ’08]

- Linear interpretation to vectors over $\mathbb{N}^k$, coefficients are matrices
- Useful for deeply nested terms
- Automation: constraints with more complex atoms
- Several flavours: plus-times-semiring, max-plus-semiring [Koprowski, Waldmann, Acta Cyb. ’09], ...
- Generalisation to tuple interpretations [Yamada, JAR ’22]
Matrix Interpretations and Monotone Algebras

Matrix interpretations [Endrullis, Waldmann, Zantema, JAR ’08]
- linear interpretation to vectors over $\mathbb{N}^k$, coefficients are matrices
- useful for deeply nested terms
- automation: constraints with more complex atoms
- several flavours: plus-times-semiring, max-plus-semiring [Koprowski, Waldmann, Acta Cyb. ’09], …
- generalisation to tuple interpretations [Yamada, JAR ’22]

Polynomial and matrix interpretations: examples of monotone algebras
Matrix Interpretations and Monotone Algebras

Matrix interpretations [Endrullis, Waldmann, Zantema, JAR ’08]
- linear interpretation to vectors over $\mathbb{N}^k$, coefficients are matrices
- useful for deeply nested terms
- automation: constraints with more complex atoms
- several flavours: plus-times-semiring, max-plus-semiring [Koprowski, Waldmann, Acta Cyb. ’09], ...
- generalisation to tuple interpretations [Yamada, JAR ’22]

Polynomial and matrix interpretations: examples of monotone algebras

Get reduction pair $(\succeq, \succ)$ from weakly monotone algebra $(A, [\cdot], >, \geq)$
- $>$ well founded
- $> \circ \geq \subseteq >$
- $a_i \geq b_i \Rightarrow [f](a_1, \ldots, a_i, \ldots, a_n) \geq [f](a_1, \ldots, b_i, \ldots, a_n)$
Matrix Interpretations and Monotone Algebras

Matrix interpretations [Endrullis, Waldmann, Zantema, JAR ’08]

- linear interpretation to vectors over $\mathbb{N}^k$, coefficients are matrices
- useful for deeply nested terms
- automation: constraints with more complex atoms
- several flavours: plus-times-semiring, max-plus-semiring [Koprowski, Waldmann, Acta Cyb. ’09], ...
- generalisation to tuple interpretations [Yamada, JAR ’22]

Polynomial and matrix interpretations: examples of monotone algebras

Get reduction pair $(\preceq, \succ)$ from weakly monotone algebra $(A, [\cdot], >, \geq)$

- $>$ well founded
- $> \circ \geq \subseteq >$
- $a_i \geq b_i \Rightarrow [f](a_1, \ldots, a_i, \ldots, a_n) \geq [f](a_1, \ldots, b_i, \ldots, a_n)$
- if also $\succ$ should be monotone (extended monotone algebra): $a_i > b_i \Rightarrow [f](a_1, \ldots, a_i, \ldots, a_n) > [f](a_1, \ldots, b_i, \ldots, a_n)$
Match-bounds (1/2)

Special case: all symbols have arity 1 → String Rewrite System (SRS)
Special case: all symbols have arity 1 → String Rewrite System (SRS)

\{a(a(x)) \rightarrow a(b(a(x)))\} as SRS: \mathcal{R} = \{aa \rightarrow aba\}
Special case: all symbols have arity 1 $\rightarrow$ String Rewrite System (SRS)
\[
\{ a(a(x)) \rightarrow a(b(a(x))) \} \text{ as SRS: } R = \{ aa \rightarrow aba \}
\]

**Match-bounds** prove termination [Geser, Hofbauer, Waldmann, AAECC ’04]

**Bound** on how often a symbol or any of its descendants are matched
Special case: all symbols have arity 1 \( \rightarrow \) String Rewrite System (SRS) 
\( \{ a(a(x)) \rightarrow a(b(a(x))) \} \) as SRS: \( R = \{ aa \rightarrow aba \} \)

**Match-bounds** prove termination [Geser, Hofbauer, Waldmann, *AAECC ’04*]

**Bound** on how often a symbol or any of its descendants are matched

Idea: track the “generation” of a symbol wrt its original ancestor symbols in the start term: 
\( a_0a_0 \rightarrow a_1b_1a_1, a_1a_0 \rightarrow a_1b_1a_1, \ldots \)
Special case: all symbols have arity 1 → String Rewrite System (SRS)
\{a(a(x)) → a(b(a(x))))\} as SRS: \(\mathcal{R} = \{aa → aba\}\)

**Match-bounds** prove termination [Geser, Hofbauer, Waldmann, *AAECC '04*]

Bound on how often a symbol or any of its descendants are matched

Idea: track the “generation” of a symbol wrt its original ancestor symbols in the start term: \(a_0a_0 → a_1b_1a_1, a_1a_0 → a_1b_1a_1, \ldots\)

\[
a_0a_0a_0a_0a_0b_0
\]
Match-bounds (1/2)

Special case: all symbols have arity 1 → String Rewrite System (SRS)
\{a(a(x)) → a(b(a(x))))\} as SRS: \(\mathcal{R} = \{aa → aba\}\)

**Match-bounds** prove termination [Geser, Hofbauer, Waldmann, AAECC ’04]

**Bound** on how often a symbol or any of its descendants are matched

Idea: track the “generation” of a symbol wrt its original ancestor symbols
in the start term: \(a_0a_0 → a_1b_1a_1, a_1a_0 → a_1b_1a_1, \ldots\)

\[
\begin{align*}
a_0a_0a_0a_0a_0b_0 \\
→ a_1b_1a_1a_0a_0a_0b_0
\end{align*}
\]
Special case: all symbols have arity 1 \(\rightarrow\) String Rewrite System (SRS)
\[
\{a(a(x)) \rightarrow a(b(a(x)))\}
\]
as SRS: \(\mathcal{R} = \{aa \rightarrow aba\}\)

**Match-bounds** prove termination [Geser, Hofbauer, Waldmann, *AAECC ’04*]

**Bound** on how often a symbol or any of its descendants are matched

Idea: track the “generation” of a symbol wrt its original ancestor symbols in the start term: \(a_0a_0 \rightarrow a_1b_1a_1, a_1a_0 \rightarrow a_1b_1a_1, \ldots\)

\[
\begin{align*}
a_0a_0a_0a_0a_0b_0 & \\
\rightarrow a_1b_1a_1a_0a_0a_0b_0 & \\
\rightarrow a_1b_1a_1a_1b_1a_1a_0b_0 &
\end{align*}
\]
Special case: all symbols have arity 1 → String Rewrite System (SRS)
\{a(a(x)) → a(b(a(x))))\} as SRS: \( \mathcal{R} = \{aa → aba\} \)

**Match-bounds** prove termination [Geser, Hofbauer, Waldmann, AAECC ’04]

**Bound** on how often a symbol or any of its descendants are matched

Idea: track the “generation” of a symbol wrt its original ancestor symbols in the start term: \(a_0a_0 → a_1b_1a_1, a_1a_0 → a_1b_1a_1, \ldots\)

\[
\begin{align*}
a_0a_0a_0a_0a_0b_0 & \\
→ a_1b_1a_1a_0a_0a_0b_0 & \\
→ a_1b_1a_1a_1b_1a_1a_0b_0 & \\
→ a_1b_1a_2b_2a_2b_1a_1a_0b_0 & 
\end{align*}
\]
Special case: all symbols have arity 1 → String Rewrite System (SRS)
\{a(a(x)) → a(b(a(x))))\} as SRS: \( R = \{aa → aba\} \)

**Match-bounds** prove termination [Geser, Hofbauer, Waldmann, AAECC ’04]

**Bound** on how often a symbol or any of its descendants are matched

Idea: track the “generation” of a symbol wrt its original ancestor symbols in the start term: \( a_0 a_0 \rightarrow a_1 b_1 a_1, a_1 a_0 \rightarrow a_1 b_1 a_1, \ldots \)

\[
\begin{align*}
a_0 a_0 a_0 a_0 a_0 b_0 & \\
→ a_1 b_1 a_1 a_0 a_0 a_0 b_0 & \\
→ a_1 b_1 a_1 a_1 b_1 a_1 a_0 b_0 & \\
→ a_1 b_1 a_2 b_2 a_2 b_1 a_1 a_0 b_0 & \\
→ a_1 b_1 a_2 b_2 a_2 b_1 a_1 b_1 a_1 b_0 & 
\end{align*}
\]
Special case: all symbols have arity 1 → String Rewrite System (SRS)
\{a(a(x)) \rightarrow a(b(a(x))))\} as SRS: \mathcal{R} = \{aa \rightarrow aba\}

Match-bounds prove termination [Geser, Hofbauer, Waldmann, AAECC ’04]

Bound on how often a symbol or any of its descendants are matched

Idea: track the “generation” of a symbol wrt its original ancestor symbols in the start term: \(a_0 a_0 \rightarrow a_1 b_1 a_1, a_1 a_0 \rightarrow a_1 b_1 a_1, \ldots\)

\[
\begin{align*}
&\underline{a_0 a_0 a_0 a_0 a_0 b_0} \\
&\quad \rightarrow a_1 b_1 a_1 \underline{a_0 a_0 a_0 b_0} \\
&\quad \rightarrow a_1 b_1 a_1 \underline{a_1 b_1 a_1 a_0 b_0} \\
&\quad \rightarrow a_1 b_1 a_2 b_2 a_2 b_1 a_1 a_0 b_0 \\
&\quad \rightarrow a_1 b_1 a_2 b_2 a_2 b_1 a_1 b_1 a_1 b_0 \\
\end{align*}
\]

Symbol generation (match height) bounded by 2!
$R = \{ \text{aa} \rightarrow \text{aba} \}$ has a match-bound of 2!
$\mathcal{R} = \{\text{aa} \rightarrow \text{aba}\}$ has a match-bound of 2! Automaton as certificate:
$\mathcal{R} = \{aa \rightarrow aba\}$ has a match-bound of 2! Automaton as certificate:
\( \mathcal{R} = \{aa \rightarrow aba\} \) has a match-bound of 2! Automaton as certificate:
$\mathcal{R} = \{aa \rightarrow aba\}$ has a match-bound of 2! Automaton as certificate:

For regular language $L$: If there is $c \in \mathbb{N}$ such that from all $w \in L \times \{0\}$ match height $c$ is never reached $\Rightarrow$ SRS terminating on $L$
$R = \{ aa \rightarrow aba \}$ has a match-bound of 2! Automaton as certificate:

For regular language $L$: If there is $c \in \mathbb{N}$ such that from all $w \in L \times \{ 0 \}$ match height $c$ is never reached $\Rightarrow$ SRS terminating on $L$

Extensions:

- Right-Forward Closure match-bounds: a restricted set of start terms suffices
- Match-bounds for TRSs via tree automata [Geser et al, IC ’07; Korp, Middeldorp, IC ’09]
- Termination techniques based on (weighted) automata and on matrices are two sides of the same coin! [Waldmann, RTA ’09]
Path orders: based on precedences on function symbols

- **Knuth-Bendix Order (KBO)** [Knuth, Bendix, *CPAA ’70*]
  → polynomial time algorithm [Korovin, Voronkov, *IC ’03*]
  → SMT encoding [Zankl, Hirokawa, Middeldorp, *JAR ’09*]
Path orders: based on *precedences* on function symbols

- **Knuth-Bendix Order (KBO)** [Knuth, Bendix, *CPAA ’70*]
  - → polynomial time algorithm [Korovin, Voronkov, *IC ’03*]
  - → SMT encoding [Zankl, Hirokawa, Middeldorp, *JAR ’09*]

  - → SAT encoding [Codish et al, *JAR ’11*]

- **Weighted Path Order (WPO)** [Yamada, Kusakari, Sakabe, *SCP ’15*] → SMT encoding
Path orders: based on **precedences** on function symbols

- **Knuth-Bendix Order (KBO)** [Knuth, Bendix, *CPAA ’70*]
  → polynomial time algorithm [Korovin, Voronkov, *IC ’03*]
  → SMT encoding [Zankl, Hirokawa, Middeldorp, *JAR ’09*]

  → SAT encoding [Codish et al, *JAR ’11*]

- **Weighted Path Order (WPO)** [Yamada, Kusakari, Sakabe, *SCP ’15*]
  → SMT encoding
Example (Constraints for Division)

\[ \mathcal{R} = \{ \ldots \} \]

\[ \mathcal{P} = \left\{ \begin{array}{ll}
\text{minus}^\#(s(x), s(y)) & \text{minus}^\#(x, y) \\
\text{quot}^\#(s(x), s(y)) & \text{minus}^\#(x, y) \\
\text{quot}^\#(s(x), s(y)) & \text{quot}^\#(\text{minus}(x, y), s(y))
\end{array} \right. \]
Example (Constraints for Division)

\[
\mathcal{R} = \{ \ldots \}
\]

\[
\mathcal{P} = \begin{cases} 
\text{minus}^\#(s(x), s(y)) &\rightarrow \text{minus}^\#(x, y) \\
\text{quot}^\#(s(x), s(y)) &\rightarrow \text{minus}^\#(x, y) \\
\text{quot}^\#(s(x), s(y)) &\rightarrow \text{quot}^\#(\text{minus}(x, y), s(y)) 
\end{cases}
\]

Goal: make the input for the constraint solver smaller

Dependency Graph

which DPs can follow one another? [Arts, Giesl, TCS '00]

Undecidable! Use dep. graph over-approximation, e.g., look at roots

Consider only non-trivial Strongly Connected Components (SCCs), separately

Here:

\[
\mathcal{P}_1 = \{ (1) \}
\]

and

\[
\mathcal{P}_2 = \{ (3) \}
\]
Example (Constraints for Division)

\[ \mathcal{R} = \{ \ldots \} \]

\[ \mathcal{P} = \begin{cases} 
    \text{minus}^\#(s(x), s(y)) & \rightarrow \text{minus}^\#(x, y) \\
    \text{quot}^\#(s(x), s(y)) & \rightarrow \text{minus}^\#(x, y) \\
    \text{quot}^\#(s(x), s(y)) & \rightarrow \text{quot}^\#(\text{minus}(x, y), s(y)) 
\end{cases} \]

Goal: make the input for the constraint solver smaller
Example (Constraints for Division)

\[ \mathcal{R} = \{ \ldots \} \]
\[ \mathcal{P} = \begin{cases} 
\text{minus\#}(s(x), s(y)) & \rightarrow \text{minus\#}(x, y) \\
\text{quot\#}(s(x), s(y)) & \rightarrow \text{minus\#}(x, y) \\
\text{quot\#}(s(x), s(y)) & \rightarrow \text{quot\#}(\text{minus}(x, y), s(y)) \end{cases} \]

Goal: make the input for the constraint solver smaller

Dependency Graph: in an infinite chain

\[ s_1 \rightarrow_{\mathcal{P}} t_1 \rightarrow_{\mathcal{R}}^* s_2 \rightarrow_{\mathcal{P}} t_2 \rightarrow_{\mathcal{R}}^* s_3 \rightarrow_{\mathcal{P}} \ldots \]

which DPs can follow one another? [Arts, Giesl, TCS ’00]
Goal: make the input for the constraint solver smaller

Dependency Graph: in an infinite chain

\[ s_1 \rightarrow^p t_1 \rightarrow^* R s_2 \rightarrow^p t_2 \rightarrow^* R s_3 \rightarrow^p \ldots \]

which DPs can follow one another? [Arts, Giesl, TCS '00]

Undecidable! Use dep. graph over-approximation, e.g., look at roots \( f^\# \):
Dependency Graph

Example (Constraints for Division)

\[ \mathcal{R} = \{ \ldots \} \]

\[ \mathcal{P} = \begin{cases} 
\text{minus}\#(s(x), s(y)) \rightarrow \text{minus}\#(x, y) & (1) \\
\text{quot}\#(s(x), s(y)) \rightarrow \text{minus}\#(x, y) & (2) \\
\text{quot}\#(s(x), s(y)) \rightarrow \text{quot}\#(\text{minus}(x, y), s(y)) & (3) 
\end{cases} \]

Goal: make the input for the constraint solver smaller

Dependency Graph: in an infinite chain

\[ s_1 \rightarrow_{\mathcal{P}} t_1 \rightarrow^{*}_{\mathcal{R}} s_2 \rightarrow_{\mathcal{P}} t_2 \rightarrow^{*}_{\mathcal{R}} s_3 \rightarrow_{\mathcal{P}} \ldots \]

which DPs can follow one another? [Arts, Giesl, TCS '00]

Undecidable! Use dep. graph over-approximation, e.g., look at roots \( f\# \):

\[ (1) \leftarrow (2) \leftarrow (3) \]
Dependency Graph

Example (Constraints for Division)

\[ R = \{ \ldots \} \]

\[ P = \{
\begin{align*}
\text{minus}^\sharp(s(x), s(y)) & \rightarrow \text{minus}^\sharp(x, y) \\
\text{quot}^\sharp(s(x), s(y)) & \rightarrow \text{minus}^\sharp(x, y) \\
\text{quot}^\sharp(s(x), s(y)) & \rightarrow \text{quot}^\sharp(\text{minus}(x, y), s(y))
\end{align*}
\] (1)

Goal: make the input for the constraint solver smaller

Dependency Graph: in an infinite chain

\[ s_1 \rightarrow_P t_1 \rightarrow_R s_2 \rightarrow_P t_2 \rightarrow_R s_3 \rightarrow_P \ldots \]

which DPs can follow one another? [Arts, Giesl, TCS ’00]

Undecidable! Use dep. graph over-approximation, e.g., look at roots \( f^\sharp \):

- Consider only non-trivial Strongly Connected Components (SCCs), separately
Example (Constraints for Division)

\[ \mathcal{R} = \{ \ldots \} \]

\[ \mathcal{P} = \left\{ \begin{array}{l}
\text{minus}^\#(s(x), s(y)) \rightarrow \text{minus}^\#(x, y) \\
\text{quot}^\#(s(x), s(y)) \rightarrow \text{minus}^\#(x, y) \\
\text{quot}^\#(s(x), s(y)) \rightarrow \text{quot}^\#(\text{minus}(x, y), s(y))
\end{array} \right. \]

Goal: make the input for the constraint solver smaller

Dependency Graph: in an infinite chain

\[ s_1 \rightarrow_p t_1 \rightarrow^* R s_2 \rightarrow_p t_2 \rightarrow^* R s_3 \rightarrow_p \ldots \]

which DPs can follow one another? [Arts, Giesl, TCS ’00]

Undecidable! Use dep. graph over-approximation, e.g., look at roots \( f^\# \):

- Consider only non-trivial Strongly Connected Components (SCCs), separately
- Here: \( \mathcal{P}_1 = \{(1)\} \) and \( \mathcal{P}_2 = \{(3)\} \)
**Goal:** make the input for the constraint solver smaller

**Dependency Graph:** in an infinite chain

\[ s_1 \rightarrow_P t_1 \overset{*}{\rightarrow_R} s_2 \rightarrow_P t_2 \overset{*}{\rightarrow_R} s_3 \rightarrow_P \ldots \]

which DPs can follow one another? [Arts, Giesl, *TCS ’00*]

Undecidable! Use **dep. graph over-approximation**, e.g., look at roots $f^*$:

- Consider only non-trivial Strongly Connected Components (SCCs), separately
- Here: $\mathcal{P}_1 = \{(1)\}$ and $\mathcal{P}_2 = \{(3)\}$
Usable Rules

Example (Constraints for Division)

\[ \mathcal{R} = \begin{cases} 
\text{minus}(x, 0) & \rightarrow \ x \\
\text{minus}(s(x), s(y)) & \rightarrow \ \text{minus}(x, y) \\
\text{quot}(0, s(y)) & \rightarrow \ 0 \\
\text{quot}(s(x), s(y)) & \rightarrow \ s(\text{quot}(\text{minus}(x, y), s(y))) 
\end{cases} \]

\[ \mathcal{P}_2 = \begin{cases} 
\text{quot}^\#(s(x), s(y)) & \rightarrow \ \text{quot}^\#(\text{minus}(x, y), s(y)) 
\end{cases} \]
Usable Rules

Example (Constraints for Division)

$$\mathcal{R} = \begin{cases} 
    \text{minus}(x, 0) & \rightarrow x \\
    \text{minus}(s(x), s(y)) & \rightarrow \text{minus}(x, y) \\
    \text{quot}(0, s(y)) & \rightarrow 0 \\
    \text{quot}(s(x), s(y)) & \rightarrow s(\text{quot}(\text{minus}(x, y), s(y)))
\end{cases}$$

$$\mathcal{P}_2 = \begin{cases} 
    \text{quot}^\#(s(x), s(y)) & \rightarrow \text{quot}^\#(\text{minus}(x, y), s(y))
\end{cases}$$

Reduction Pair Processor may ignore “unused parts” of $\mathcal{R}$ for $\succsim$
Usable Rules

Example (Constraints for Division)

\[
\mathcal{R} = \begin{cases} 
\text{minus}(x, 0) & \rightarrow \ x \\
\text{minus}(s(x), s(y)) & \rightarrow \ \text{minus}(x, y) \\
\text{quot}(0, s(y)) & \rightarrow \ 0 \\
\text{quot}(s(x), s(y)) & \rightarrow \ s(\text{quot}(\text{minus}(x, y), s(y))) \\
\end{cases}
\]

\[
\mathcal{P}_2 = \{ \text{quot}^\#(s(x), s(y)) \rightarrow \text{quot}^\#(\text{minus}(x, y), s(y)) \}
\]

Reduction Pair Processor may ignore “unused parts” of \( \mathcal{R} \) for \( \preccurlyeq \):
- \( \text{quot}^\#(s(x), s(y)) \rightarrow \text{quot}^\#(\text{minus}(x, y), s(y)) \) calls \text{minus}
Usable Rules

Example (Constraints for Division)

\[ \mathcal{R} = \begin{cases} 
\text{minus}(x, 0) & \rightarrow x \\
\text{minus}(s(x), s(y)) & \rightarrow \text{minus}(x, y) \\
\text{quot}(0, s(y)) & \rightarrow 0 \\
\text{quot}(s(x), s(y)) & \rightarrow s(\text{quot}(\text{minus}(x, y), s(y))) 
\end{cases} \]

\[ \mathcal{P}_2 = \{ \text{quot}^\#(s(x), s(y)) \rightarrow \text{quot}^\#(\text{minus}(x, y), s(y)) \} \]

Reduction Pair Processor may ignore “unused parts” of \( \mathcal{R} \) for \( \preceq \):

- \( \text{quot}^\#(s(x), s(y)) \rightarrow \text{quot}^\#(\text{minus}(x, y), s(y)) \) calls \text{minus}
- \( \text{quot}^\#(s(x), s(y)) \rightarrow \text{quot}^\#(\text{minus}(x, y), s(y)) \) does not call \text{quot}
### Usable Rules

**Example (Constraints for Division)**

\[ R = \begin{cases} 
\text{minus}(x, 0) & \rightarrow x \\
\text{minus}(s(x), s(y)) & \rightarrow \text{minus}(x, y) \\
\text{quot}(0, s(y)) & \rightarrow 0 \\
\text{quot}(s(x), s(y)) & \rightarrow s(\text{quot}(\text{minus}(x, y), s(y))) 
\end{cases} \]

\[ P_2 = \{ \text{quot}^\#(s(x), s(y)) \rightarrow \text{quot}^\#(\text{minus}(x, y), s(y)) \} \]

Reduction Pair Processor may ignore “unused parts” of \( R \) for \( \Rightarrow \):

- \( \text{quot}^\#(s(x), s(y)) \rightarrow \text{quot}^\#(\text{minus}(x, y), s(y)) \) calls \text{minus}
- \( \text{quot}^\#(s(x), s(y)) \rightarrow \text{quot}^\#(\text{minus}(x, y), s(y)) \) does not call \text{quot}
- \text{minus} rules do not call \text{quot}
Usable Rules

Example (Constraints for Division)

\[ R = \begin{cases} 
    \text{minus}(x, 0) & \rightarrow x \\
    \text{minus}(s(x), s(y)) & \rightarrow \text{minus}(x, y) \\
    \text{quot}(0, s(y)) & \rightarrow 0 \\
    \text{quot}(s(x), s(y)) & \rightarrow s(\text{quot}(\text{minus}(x, y), s(y))) 
\end{cases} \]

\[ P_2 = \{ \text{quot}(s(x), s(y)) \rightarrow \text{quot}(\text{minus}(x, y), s(y)) \} \]

Reduction Pair Processor may ignore “unused parts” of \( R \) for \( \succcurlyeq \)

- \( \text{quot}(s(x), s(y)) \rightarrow \text{quot}(\text{minus}(x, y), s(y)) \) calls \text{minus}
- \( \text{quot}(s(x), s(y)) \rightarrow \text{quot}(\text{minus}(x, y), s(y)) \) does not call \text{quot}
- \text{minus} rules do not call \text{quot}

\( \Rightarrow \) instead of \( R \subseteq \succcurlyeq \), it suffices if Usable Rules \( UR(P, R) \subseteq \succcurlyeq \)

[Giesl et al, JAR ’06; Hirokawa, Middeldorp, IC ’07]
Usable Rules

Example (Constraints for Division)

\[ \mathcal{R} = \begin{cases} 
\text{minus}(x, 0) & \rightarrow x \\
\text{minus}(s(x), s(y)) & \rightarrow \text{minus}(x, y) \\
\text{quot}(0, s(y)) & \rightarrow 0 \\
\text{quot}(s(x), s(y)) & \rightarrow \text{s(quot(minus}(x, y), s(y))) 
\end{cases} \]

\[ \mathcal{P}_2 = \{ \text{quot}^\#(s(x), s(y)) \rightarrow \text{quot}^\#(\text{minus}(x, y), s(y)) \} \]

Reduction Pair Processor may ignore “unused parts” of \( \mathcal{R} \) for \( \succeq \)

- \( \text{quot}^\#(s(x), s(y)) \rightarrow \text{quot}^\#(\text{minus}(x, y), s(y)) \) calls \( \text{minus} \)
- \( \text{quot}^\#(s(x), s(y)) \rightarrow \text{quot}^\#(\text{minus}(x, y), s(y)) \) does not call \( \text{quot} \)
- \( \text{minus} \) rules do not call \( \text{quot} \)

\( \Rightarrow \) instead of \( \mathcal{R} \subseteq \succeq \), it suffices if \textbf{Usable Rules} \( \text{UR}(\mathcal{P}, \mathcal{R}) \subseteq \succeq \)

[Giesl et al, JAR ’06; Hirokawa, Middeldorp, IC ’07]

Full rewriting: \( \succeq \) must be “\( C_\varepsilon \)-compatible” (\( c(x, y) \succeq x \) and \( c(x, y) \succeq y \))

Not needed for termination of innermost rewriting!
Further Techniques and Settings for TRSs

- Many more modular **DP processors** to simplify/transform \((P, R)\)
  [Thiemann, *PhD thesis '07*]

\[
\text{Higher-order rewriting: functional variables, higher types, } \beta\text{-reduction}
\]

\[
\text{Probabilistic term rewriting: Positive/Strong Almost Sure Termination}
\]

Complexity analysis

Can re-use termination machinery to infer and prove statements like

\[
\text{"runtime complexity of this TRS is in } O(n^3)\text{"}
\]
Further Techniques and Settings for TRSs

- Many more modular **DP processors** to simplify/transform \((\mathcal{P}, \mathcal{R})\)
  [Thiemann, *PhD thesis ’07*]
- Proving **non**-termination (an infinite run is possible)
Further Techniques and Settings for TRSs

- Many more modular DP processors to simplify/transform \((\mathcal{P}, \mathcal{R})\) [Thiemann, PhD thesis ’07]
- Proving non-termination (an infinite run is possible) [Giesl, Thiemann, Schneider-Kamp, FroCoS ’05; Payet, TCS ’08; Zankl et al, SOFSEM ’10; Emmes, Enger, Giesl, IJCAR ’12; ...]
- Specific rewrite strategies: innermost, outermost, context-sensitive rewriting [Lucas, ACM Comput. Surv. ’20], ...
Further Techniques and Settings for TRSs

- Many more modular **DP processors** to simplify/transform \((P, R)\)  
  [Thiemann, *PhD thesis ’07*]

- Proving **non-termination** (an infinite run is possible)  

- **Specific rewrite strategies**: innermost, outermost, context-sensitive rewriting  

- **Higher-order** rewriting: functional variables, higher types, \(\beta\)-reduction  
  \[
  \text{map}(F, \text{Cons}(x, xs)) \rightarrow \text{Cons}(F(x), \text{map}(F, xs))
  \]
  [Kop, *PhD thesis ’12*]
Further Techniques and Settings for TRSs

- Many more modular DP processors to simplify/transform \((P, R)\) [Thiemann, PhD thesis ’07]
- Proving non-termination (an infinite run is possible) [Giesl, Thiemann, Schneider-Kamp, FroCoS ’05; Payet, TCS ’08; Zankl et al, SOFSEM ’10; Emmes, Enger, Giesl, IJCAR ’12; ...]
- Specific rewrite strategies: innermost, outermost, context-sensitive rewriting [Lucas, ACM Comput. Surv. ’20], ...
- Higher-order rewriting: functional variables, higher types, \(\beta\)-reduction
  \[
  \text{map}(F, \text{Cons}(x, xs)) \rightarrow \text{Cons}(F(x), \text{map}(F, xs))
  \]
  [Kop, PhD thesis ’12]
- Probabilistic term rewriting: Positive/Strong Almost Sure Termination [Avanzini, Dal Lago, Yamada, SCP ’20]
Further Techniques and Settings for TRSs

- Many more modular **DP processors** to simplify/transform \((P, R)\)
  [Thiemann, *PhD thesis ’07*]
- Proving **non-termination** (an infinite run is possible)
  [Giesl, Thiemann, Schneider-Kamp, *FroCoS ’05*; Payet, *TCS ’08*;
   Zankl et al, *SOFSEM ’10*; Emmes, Enger, Giesl, *IJCAR ’12*; ...]
- Specific **rewrite strategies**: innermost, outermost, context-sensitive
  rewriting [Lucas, *ACM Comput. Surv. ’20*], ...
- **Higher-order** rewriting: functional variables, higher types, \(\beta\)-reduction
  \[
  \text{map}(F, \text{Cons}(x, xs)) \rightarrow \text{Cons}(F(x), \text{map}(F, xs))
  \]
  [Kop, *PhD thesis ’12*]
- **Probabilistic** term rewriting: Positive/Strong Almost Sure
  Termination [Avanzini, Dal Lago, Yamada, *SCP ’20*]
- **Complexity analysis**
  [Hirokawa, Moser, *IJCAR ’08*; Noschinski, Emmes, Giesl, *JAR ’13*; ...]
  Can re-use termination machinery to infer and prove statements like
  “runtime complexity of this TRS is in \(\mathcal{O}(n^3)\)”
  \(\rightarrow\) more in Session 2!
SMT Solvers from Termination Analysis

Annual SMT-COMP, division QF_NIA (Quantifier-Free Non-linear Integer Arithmetic)

<table>
<thead>
<tr>
<th>Year</th>
<th>Winner</th>
</tr>
</thead>
<tbody>
<tr>
<td>2009</td>
<td>Barcelogic-QF_NIA</td>
</tr>
<tr>
<td>2010</td>
<td>MiniSmt</td>
</tr>
<tr>
<td>2011</td>
<td>AProVE</td>
</tr>
<tr>
<td>2012</td>
<td>no QF_NIA</td>
</tr>
<tr>
<td>2013</td>
<td>no SMT-COMP</td>
</tr>
<tr>
<td>2014</td>
<td>AProVE</td>
</tr>
<tr>
<td>2015</td>
<td>AProVE</td>
</tr>
<tr>
<td>2016</td>
<td>Yices</td>
</tr>
</tbody>
</table>

Termination provers can also be successful SMT solvers! (disclaimer: Z3 participated only hors concours)
### Annual SMT-COMP, division QF\_NIA (Quantifier-Free Non-linear Integer Arithmetic)

<table>
<thead>
<tr>
<th>Year</th>
<th>Winner</th>
</tr>
</thead>
<tbody>
<tr>
<td>2009</td>
<td>Barcelogic-QF_NIA</td>
</tr>
<tr>
<td>2010</td>
<td>MiniSmt (spin-off of TTT2)</td>
</tr>
<tr>
<td>2011</td>
<td>AProVE</td>
</tr>
<tr>
<td>2012</td>
<td>no QF_NIA</td>
</tr>
<tr>
<td>2013</td>
<td>no SMT-COMP</td>
</tr>
<tr>
<td>2014</td>
<td>AProVE</td>
</tr>
<tr>
<td>2015</td>
<td>AProVE</td>
</tr>
<tr>
<td>2016</td>
<td>Yices</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

⇒ **Termination provers** can also be successful SMT solvers!
Annual SMT-COMP, division QF$_{\text{NIA}}$ (Quantifier-Free Non-linear Integer Arithmetic)

<table>
<thead>
<tr>
<th>Year</th>
<th>Winner</th>
</tr>
</thead>
<tbody>
<tr>
<td>2009</td>
<td>Barcologic-QF$_{\text{NIA}}$</td>
</tr>
<tr>
<td>2010</td>
<td>MiniSmt (spin-off of TTT$_2$)</td>
</tr>
<tr>
<td>2011</td>
<td>AProVE</td>
</tr>
<tr>
<td>2012</td>
<td>no QF$_{\text{NIA}}$</td>
</tr>
<tr>
<td>2013</td>
<td>no SMT-COMP</td>
</tr>
<tr>
<td>2014</td>
<td>AProVE</td>
</tr>
<tr>
<td>2015</td>
<td>AProVE</td>
</tr>
<tr>
<td>2016</td>
<td>Yices</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

⇒ **Termination provers** can also be successful SMT solvers!

(disclaimer: Z3 participated only *hors concours*)
The Termination Competition (termCOMP) (1/3)

Termination Competition 2022

**Competition-Wide Ranking**

APROVE+LoAT(4.0811) MU-TERM(1.9331) TTT2+TcT(1.9082) NaTT(1.4268) Matchbox(1.3425) iRankFinder(1.2594) Ultimate(1.2079) MUltumNonMulti(1.1930) NTI+cTI(0.9649) SOL(0.9180) Wanda(0.8975)

**Advancing-the-State-of-the-Art Ranking**

Matchbox(67) MUltumNonMulti(48) APROVE+LoAT(31.25) SOL(16) NaTT(1) NTI+cTI(1) TTT2+TcT(0.375) iRankFinder(0) MU-TERM(0) Ultimate(0) Wanda(0)

**Termination of Rewriting**

Progress: 100%, CPU Time: 85d 8:05:33, Node Time: 34d 3:49:50

**Termination of Programs**


**Complexity Analysis**

The Termination Competition (termCOMP) (1/3)

Termination Competition 2022

Competition-Wide Ranking

APrOVE+LoAT(4.0811) MU-TERM(1.9331) TTT2+TcT(1.9062) NaTT(1.4268) Matchbox(1.3425) iRankFinder(1.2594) Ultimate(1.2079) MultumNonMultum(1.1930) NTI+cT(1.0964) SOL(0.9180) Wanda(0.8975)

Advancing-the-State-of-the-Art Ranking

Matchbox(67) MultumNonMultum(48) APrOVE+LoAT(31.25) SOL(16) NaTT(11) NTI+cT(11) TTT2+TcT(3.75) iRankFinder(0) MU-TERM(0) Ultimate(0) Wanda(0)

Termination of Rewriting

Progress: 100%, CPU Time: 85d 05:33, Node Time: 34d 34:50

Termination of Programs

Progress: 100%, CPU Time: 3d 22:33, Node Time: 2d 42:44

Complexity Analysis


https://termination-portal.org/wiki/Termination_Competition
The Termination Competition (termCOMP) (2/3)

termCOMP 2022 participants

- AProVE (RWTH Aachen, Birkbeck U London, U Innsbruck, ...)
- iRankFinder (UC Madrid)
- LoAT (RWTH Aachen)
- Matchbox (HTWK Leipzig)
- Mu-Term (UP Valencia, UP Madrid)
- MultumNonMulta (BA Saarland)
- NaTT (AIST Tokyo)
- NTI+cTI (U Réunion)
- SOL (Gunma U)
- TcT (U Innsbruck, INRIA Sophia Antipolis)
- TTT_2 (U Innsbruck)
- Ultimate Automizer (U Freiburg)
- Wanda (RU Nijmegen)
termCOMP 2022 participants ...with at least 1 developer at ISR 2022!

- AProVE (RWTH Aachen, Birkbeck U London, U Innsbruck, ...)
- iRankFinder (UC Madrid)
- LoAT (RWTH Aachen)
- Matchbox (HTWK Leipzig)
- Mu-Term (UP Valencia, UP Madrid)
- MultumNonMulta (BA Saarland)
- NaTT (AIST Tokyo)
- NTI+cTI (U Réunion)
- SOL (Gunma U)
- TcT (U Innsbruck, INRIA Sophia Antipolis)
- $TT_T^2$ (U Innsbruck)
- Ultimate Automizer (U Freiburg)
- Wanda (RU Nijmegen)
Benchmark set: Termination Problem DataBase (TPDB)
https://termination-portal.org/wiki/TPDB
→ 1000s of termination and complexity problems
Benchmark set: Termination Problem DataBase (TPDB)
https://termination-portal.org/wiki/TPDB
→ 1000s of termination and complexity problems

Timeout: 300 seconds
The Termination Competition (termCOMP) (3/3)

- Benchmark set: Termination Problem DataBase (TPDB)
  https://termination-portal.org/wiki/TPDB
  → 1000s of termination and complexity problems

- Timeout: 300 seconds

- Run on StarExec platform [Stump, Sutcliffe, Tinelli, IJCAR '14]
Benchmark set: Termination Problem DataBase (TPDB)
https://termination-portal.org/wiki/TPDB
→ 1000s of termination and complexity problems

Timeout: 300 seconds

Run on StarExec platform [Stump, Sutcliffe, Tinelli, IJCAR ’14]

Categories for proving (non-)termination and for inferring upper/lower complexity bounds for different programming languages
Benchmark set: Termination Problem DataBase (TPDB)
https://termination-portal.org/wiki/TPDB
→ 1000s of termination and complexity problems

Timeout: 300 seconds

Run on StarExec platform [Stump, Sutcliffe, Tinelli, IJCAR ’14]

Categories for proving (non-)termination and for inferring upper/lower complexity bounds for different programming languages

Part of the Olympic Games at the Federated Logic Conference
Web interfaces for termination and complexity of TRSs:

- **AProVE**: [https://aprove.informatik.rwth-aachen.de/interface](https://aprove.informatik.rwth-aachen.de/interface)
- **TcT**: [https://tcs-informatik.uibk.ac.at/tools/tct/webinterface.php](https://tcs-informatik.uibk.ac.at/tools/tct/webinterface.php)
- **TTT₂**: [http://colo6-c703.uibk.ac.at/ttt2/web/](http://colo6-c703.uibk.ac.at/ttt2/web/)
Web interfaces for termination and complexity of TRSs:

- **AProVE:** [https://aprove.informatik.rwth-aachen.de/interface](https://aprove.informatik.rwth-aachen.de/interface)
- **Mu-Term:**
- **TcT:**
  [https://tcs-informatik.uibk.ac.at/tools/tct/webinterface.php](https://tcs-informatik.uibk.ac.at/tools/tct/webinterface.php)
- **TTT2:** [http://colo6-c703.uibk.ac.at/ttt2/web/](http://colo6-c703.uibk.ac.at/ttt2/web/)

Input format for termination of TRSs:

\[
\text{(VAR } x \ y) \\
\text{(RULES)} \\
\text{plus}(0, y) \rightarrow y \\
\text{plus}(s(x), y) \rightarrow s(\text{plus}(x, y))
\]
1.2 Termination Analysis of Rewrite Systems with Logical Constraints
Papers on termination of imperative programs often about **integers** as data
Papers on termination of imperative programs often about **integers** as data

Example (Imperative Program)

\[
\begin{align*}
\textbf{if } (x \geq 0) \\
\textbf{while } (x \neq 0) \\
& x = x - 1;
\end{align*}
\]

Does this program terminate?  
\((x \text{ ranges over } \mathbb{Z})\)
Papers on termination of imperative programs often about **integers** as data

Example (Imperative Program)

| \(\ell_0\) | if (\(x \geq 0\)) |
| \(\ell_1\) | while (\(x \neq 0\)) |
| \(\ell_2\) | \(x = x - 1;\) |

Does this program terminate? (\(x\) ranges over \(\mathbb{Z}\))

Example (Equivalent Translation to an Integer Transition System, see [McCarthy, CACM '60])

| \(\ell_0(x)\) | \(\rightarrow\) | \(\ell_1(x)\) | [\(x \geq 0\)] |
| \(\ell_0(x)\) | \(\rightarrow\) | \(\ell_3(x)\) | [\(x < 0\)] |
| \(\ell_1(x)\) | \(\rightarrow\) | \(\ell_2(x)\) | [\(x \neq 0\)] |
| \(\ell_2(x)\) | \(\rightarrow\) | \(\ell_1(x - 1)\) |
| \(\ell_1(x)\) | \(\rightarrow\) | \(\ell_3(x)\) | [\(x = 0\)] |
Papers on termination of imperative programs often about **integers** as data

### Example (Imperative Program)

<table>
<thead>
<tr>
<th>$\ell_0$</th>
<th>if $(x \geq 0)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\ell_1$</td>
<td>while $(x \neq 0)$</td>
</tr>
<tr>
<td>$\ell_2$</td>
<td>$x = x - 1$;</td>
</tr>
</tbody>
</table>

Does this program terminate? (\(x\) ranges over \(\mathbb{Z}\))

### Example (Equivalent Translation to an Integer Transition System, see [McCarthy, *CACM* '60])

<table>
<thead>
<tr>
<th>$\ell_0(x)$</th>
<th>$\rightarrow$</th>
<th>$\ell_1(x)$</th>
<th>$[x \geq 0]$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\ell_0(x)$</td>
<td>$\rightarrow$</td>
<td>$\ell_3(x)$</td>
<td>$[x &lt; 0]$</td>
</tr>
<tr>
<td>$\ell_1(x)$</td>
<td>$\rightarrow$</td>
<td>$\ell_2(x)$</td>
<td>$[x \neq 0]$</td>
</tr>
<tr>
<td>$\ell_2(x)$</td>
<td>$\rightarrow$</td>
<td>$\ell_1(x - 1)$</td>
<td></td>
</tr>
<tr>
<td>$\ell_1(x)$</td>
<td>$\rightarrow$</td>
<td>$\ell_3(x)$</td>
<td>$[x = 0]$</td>
</tr>
</tbody>
</table>

Oh no! $\ell_1(-1) \rightarrow \ell_2(-1) \rightarrow \ell_1(-2) \rightarrow \ell_2(-2) \rightarrow \ell_1(-3) \rightarrow \cdots$
Papers on termination of imperative programs often about **integers** as data

**Example (Imperative Program)**

\[
\ell_0: \quad \text{if } (x \geq 0) \\
\ell_1: \quad \text{while } (x \neq 0) \\
\ell_2: \quad x = x - 1;
\]

Does this program terminate?  
\((x \text{ ranges over } \mathbb{Z})\)

**Example (Equivalent Translation to an Integer Transition System, see [McCarthy, CACM '60])**

\[
\ell_0(x) \rightarrow \ell_1(x) \quad [x \geq 0] \\
\ell_0(x) \rightarrow \ell_3(x) \quad [x < 0] \\
\ell_1(x) \rightarrow \ell_2(x) \quad [x \neq 0] \\
\ell_2(x) \rightarrow \ell_1(x - 1) \\
\ell_1(x) \rightarrow \ell_3(x) \quad [x = 0]
\]

Oh no!  
\[
\ell_1(-1) \rightarrow \ell_2(-1) \rightarrow \ell_1(-2) \rightarrow \ell_2(-2) \rightarrow \ell_1(-3) \rightarrow \cdots
\]

\[\Rightarrow \text{Restrict initial states to } \ell_0(z) \text{ for } z \in \mathbb{Z}\]
Papers on termination of imperative programs often about **integers** as data

**Example (Imperative Program)**

\[ \ell_0: \text{if } (x \geq 0) \]
\[ \ell_1: \text{while } (x \neq 0) \]
\[ \ell_2: \quad x = x - 1; \]

Does this program terminate?  
(x ranges over \( \mathbb{Z} \))

**Example (Equivalent Translation to an Integer Transition System, see [McCarthy, CACM '60])**

\[
\begin{align*}
\ell_0(x) &\rightarrow \ell_1(x) \quad [x \geq 0] \\
\ell_0(x) &\rightarrow \ell_3(x) \quad [x < 0] \\
\ell_1(x) &\rightarrow \ell_2(x) \quad [x \neq 0] \\
\ell_2(x) &\rightarrow \ell_1(x - 1) \\
\ell_1(x) &\rightarrow \ell_3(x) \quad [x = 0]
\end{align*}
\]

Oh no! \[ \ell_1(-1) \rightarrow \ell_2(-1) \rightarrow \ell_1(-2) \rightarrow \ell_2(-2) \rightarrow \ell_1(-3) \rightarrow \cdots \]

⇒ **Restrict initial states** to \( \ell_0(z) \) for \( z \in \mathbb{Z} \)
⇒ **Find invariant** \( x \geq 0 \) at \( \ell_1, \ell_2 \) (exercise)
Papers on termination of imperative programs often about integers as data

Example (Imperative Program)

\[ \ell_0: \text{if } (x \geq 0) \]
\[ \ell_1: \text{while } (x \neq 0) \]
\[ \ell_2: \quad x = x - 1; \]

Does this program terminate? (\(x\) ranges over \(\mathbb{Z}\))

Example (Equivalent Translation to an Integer Transition System, see [McCarthy, CACM ’60])

\[ \ell_0(x) \rightarrow \ell_1(x) \quad [x \geq 0] \]
\[ \ell_0(x) \rightarrow \ell_3(x) \quad [x < 0] \]
\[ \ell_1(x) \rightarrow \ell_2(x) \quad [x \neq 0 \land x \geq 0] \]
\[ \ell_2(x) \rightarrow \ell_1(x - 1) \quad [x \geq 0] \]
\[ \ell_1(x) \rightarrow \ell_3(x) \quad [x = 0 \land x \geq 0] \]

Oh no! \(\ell_1(-1) \rightarrow \ell_2(-1) \rightarrow \ell_1(-2) \rightarrow \ell_2(-2) \rightarrow \ell_1(-3) \rightarrow \cdots\)

\(\Rightarrow\) Restrict initial states to \(\ell_0(z)\) for \(z \in \mathbb{Z}\)

\(\Rightarrow\) Find invariant \(x \geq 0\) at \(\ell_1, \ell_2\) (exercise)
Papers on termination of imperative programs often about **integers** as data

**Example (Imperative Program)**

\[ \ell_0: \text{if } (x \geq 0) \]
\[ \ell_1: \text{while } (x \neq 0) \]
\[ \ell_2: \quad x = x - 1; \]

Does this program terminate?

(x ranges over \( \mathbb{Z} \))

**Example (Equivalent Translation to an Integer Transition System, see [McCarthy, CACM ’60])**

\[ \ell_0(x) \rightarrow \ell_1(x) \quad [x \geq 0] \]
\[ \ell_0(x) \rightarrow \ell_3(x) \quad [x < 0] \]
\[ \ell_1(x) \rightarrow \ell_2(x) \quad [x \neq 0 \land x \geq 0] \]
\[ \ell_2(x) \rightarrow \ell_1(x - 1) \quad [x \geq 0] \]
\[ \ell_1(x) \rightarrow \ell_3(x) \quad [x = 0 \land x \geq 0] \]

Termination of TRSs from a given set of start terms:

**Local termination**

[Endrullis, de Vrijer, Waldmann, LMCS ’10]

Oh no!

\[ \ell_1(-1) \rightarrow \ell_2(-1) \rightarrow \ell_1(-2) \rightarrow \ell_2(-2) \rightarrow \ell_1(-3) \rightarrow \cdots \]

⇒ **Restrict initial states** to \( \ell_0(z) \) for \( z \in \mathbb{Z} \)

⇒ Find **invariant** \( x \geq 0 \) at \( \ell_1, \ell_2 \) (exercise)
### Example (Transition system with invariants)

<table>
<thead>
<tr>
<th>Transition</th>
<th>New State</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\ell_0(x)$</td>
<td>$\ell_1(x)$</td>
<td>$[x \geq 0]$</td>
</tr>
<tr>
<td>$\ell_1(x)$</td>
<td>$\ell_2(x)$</td>
<td>$[x \neq 0 \land x \geq 0]$</td>
</tr>
<tr>
<td>$\ell_2(x)$</td>
<td>$\ell_1(x - 1)$</td>
<td>$[x \geq 0]$</td>
</tr>
<tr>
<td>$\ell_1(x)$</td>
<td>$\ell_3(x)$</td>
<td>$[x = 0 \land x \geq 0]$</td>
</tr>
</tbody>
</table>

Prove termination by ranking function $[\cdot]$ with $[\ell_0](x) = [\ell_1](x) = \cdots = x$.
### Example (Transition system with invariants)

<table>
<thead>
<tr>
<th>Transition</th>
<th>Invariant</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\ell_0(x)$</td>
<td>$\succsim \ell_1(x)$ [x \geq 0]*</td>
</tr>
<tr>
<td>$\ell_1(x)$</td>
<td>$\succsim \ell_2(x)$ [x \neq 0 \land x \geq 0]</td>
</tr>
<tr>
<td>$\ell_2(x)$</td>
<td>$\succ \ell_1(x - 1)$ [x \geq 0]</td>
</tr>
<tr>
<td>$\ell_1(x)$</td>
<td>$\succsim \ell_3(x)$ [x = 0 \land x \geq 0]</td>
</tr>
</tbody>
</table>

Prove termination by ranking function $[\cdot]$ with $[\ell_0](x) = [\ell_1](x) = \cdots = x$
### Example (Transition system with invariants)

<table>
<thead>
<tr>
<th>( \ell_0(x) )</th>
<th>( \succ )</th>
<th>( \ell_1(x) )</th>
<th>([x \geq 0])</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \ell_1(x) )</td>
<td>( \succ )</td>
<td>( \ell_2(x) )</td>
<td>([x \neq 0 \land x \geq 0])</td>
</tr>
<tr>
<td>( \ell_2(x) )</td>
<td>( \succ )</td>
<td>( \ell_1(x - 1) )</td>
<td>([x \geq 0])</td>
</tr>
<tr>
<td>( \ell_1(x) )</td>
<td>( \succ )</td>
<td>( \ell_3(x) )</td>
<td>([x = 0 \land x \geq 0])</td>
</tr>
</tbody>
</table>

Prove termination by ranking function \([ \cdot ]\) with \([\ell_0](x) = [\ell_1](x) = \cdots = x\)

Automate search using **parametric** ranking function:

\[
[\ell_0](x) = a_0 + b_0 \cdot x, \quad [\ell_1](x) = a_1 + b_1 \cdot x, \quad \ldots
\]
Proving Termination with Invariants

Example (Transition system with invariants)

<table>
<thead>
<tr>
<th>$\ell_0(x)$</th>
<th>$\succ$</th>
<th>$\ell_1(x)$</th>
<th>$[x \geq 0]$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\ell_1(x)$</td>
<td>$\succ$</td>
<td>$\ell_2(x)$</td>
<td>$[x \neq 0 \land x \geq 0]$</td>
</tr>
<tr>
<td>$\ell_2(x)$</td>
<td>$\succ$</td>
<td>$\ell_1(x - 1)$</td>
<td>$[x \geq 0]$</td>
</tr>
<tr>
<td>$\ell_1(x)$</td>
<td>$\succ$</td>
<td>$\ell_3(x)$</td>
<td>$[x = 0 \land x \geq 0]$</td>
</tr>
</tbody>
</table>

Prove termination by ranking function $[\cdot]$ with $[\ell_0](x) = [\ell_1](x) = \cdots = x$

Automate search using parametric ranking function:

$[\ell_0](x) = a_0 + b_0 \cdot x, \quad [\ell_1](x) = a_1 + b_1 \cdot x, \quad \ldots$

Constraints here:

- $x \geq 0 \implies a_2 + b_2 \cdot x > a_1 + b_1 \cdot (x - 1)$ \text{ “decrease …”}
- $x \geq 0 \implies a_2 + b_2 \cdot x \geq 0$ \text{ “… against a bound”}
Proving Termination with Invariants

Example (Transition system with invariants)

\[
\begin{align*}
{l_0}(x) & \succsim {l_1}(x) & [x \geq 0] \\
{l_1}(x) & \succsim {l_2}(x) & [x \neq 0 \land x \geq 0] \\
{l_2}(x) & \succsim {l_1}(x-1) & [x \geq 0] \\
{l_1}(x) & \succsim {l_3}(x) & [x = 0 \land x \geq 0]
\end{align*}
\]

Prove termination by ranking function \([ \cdot ]\) with \([l_0](x) = [l_1](x) = \cdots = x\)

Automate search using parametric ranking function:

\[
[l_0](x) = a_0 + b_0 \cdot x, \quad [l_1](x) = a_1 + b_1 \cdot x, \quad \ldots
\]

Constraints here:

\[
\begin{align*}
x \geq 0 & \implies a_2 + b_2 \cdot x > a_1 + b_1 \cdot (x - 1) & \text{“decrease …”} \\
x \geq 0 & \implies a_2 + b_2 \cdot x \geq 0 & \text{“… against a bound”}
\end{align*}
\]

Use Farkas’ Lemma to eliminate \( \forall x \), solver for linear constraints gives model for \( a_i, b_i \).
Proving Termination with Invariants

Example (Transition system with invariants)

\[
\begin{align*}
\ell_0(x) & \succcurlyeq \ell_1(x) \quad [x \geq 0] \\
\ell_1(x) & \succcurlyeq \ell_2(x) \quad [x \neq 0 \land x \geq 0] \\
\ell_2(x) & \succcurlyeq \ell_1(x - 1) \quad [x \geq 0] \\
\ell_1(x) & \succcurlyeq \ell_3(x) \quad [x = 0 \land x \geq 0]
\end{align*}
\]

Prove termination by ranking function \([ \cdot ]\) with \(\ell_0(x) = \ell_1(x) = \cdots = x\)

Automate search using parametric ranking function:

\[
\ell_0(x) = a_0 + b_0 \cdot x, \quad \ell_1(x) = a_1 + b_1 \cdot x, \quad \ldots
\]

Constraints here:

\[
\begin{align*}
x \geq 0 & \Rightarrow a_2 + b_2 \cdot x > a_1 + b_1 \cdot (x - 1) \quad \text{“decrease \ldots”} \\
x \geq 0 & \Rightarrow a_2 + b_2 \cdot x \geq 0 \quad \text{“\ldots against a bound”}
\end{align*}
\]

Use Farkas’ Lemma to eliminate \(\forall x\), solver for linear constraints gives model for \(a_i, b_i\).

More: [Podelski, Rybalchenko, VMCAI ’04, Alias et al, SAS ’10]
Proving Termination with Invariants

Example (Transition system with invariants)

\[
\begin{align*}
\ell_0(x) & \rightarrow \ell_1(x) \quad [x \geq 0] \\
\ell_1(x) & \rightarrow \ell_2(x) \quad [x \neq 0 \land x \geq 0] \\
\ell_1(x) & \rightarrow \ell_3(x) \quad [x = 0 \land x \geq 0]
\end{align*}
\]

Prove termination by ranking function \([ \cdot ]\) with \([\ell_0](x) = [\ell_1](x) = \cdots = x\)

Automate search using parametric ranking function:

\[
[\ell_0](x) = a_0 + b_0 \cdot x, \quad [\ell_1](x) = a_1 + b_1 \cdot x, \quad \ldots
\]

Constraints here:

\[
\begin{align*}
x \geq 0 \quad &\Rightarrow \quad a_2 + b_2 \cdot x > a_1 + b_1 \cdot (x - 1) \quad \text{“decrease . . .”} \\
x \geq 0 \quad &\Rightarrow \quad a_2 + b_2 \cdot x \geq 0 \quad \text{“. . . against a bound”}
\end{align*}
\]

Use Farkas’ Lemma to eliminate \(\forall x\), solver for linear constraints gives model for \(a_i, b_i\).

More: [Podolski, Rybalchenko, VMCAI ’04, Alias et al, SAS ’10]
Termination prover needs to find invariants for programs on integers
Termination prover needs to find invariants for programs on integers

- Statically before the translation
  - [Otto et al, *RTA ’10; Ströder et al, JAR ’17, …*]
  - $\rightarrow$ abstract interpretation [Cousot, Cousot, *POPL ’77*]
  - $\rightarrow$ more in Session 2!
Searching for Invariants Using SMT

Termination prover needs to find invariants for programs on integers

- Statically before the translation
  [Otto et al, RTA ’10; Ströder et al, JAR ’17, ...]
  → abstract interpretation [Cousot, Cousot, POPL ’77]
  → more in Session 2!

- By counterexample-based reasoning + safety prover: Terminator
  [Cook, Podelski, Rybalchenko, CAV ’06, PLDI ’06]
  → prove termination of single program runs
  → termination argument often generalises

Nowadays all SMT-based!
Searching for Invariants Using SMT

Termination prover needs to find invariants for programs on integers

- Statically before the translation
  → abstract interpretation [Cousot, Cousot, *POPL '77*]
  → more in Session 2!

- By counterexample-based reasoning + safety prover: Terminator
  [Cook, Podelski, Rybalchenko, *CAV '06, PLDI '06*]
  → prove termination of single program runs
  → termination argument often generalises

- ... also cooperating with removal of terminating rules (as for TRSs): T2 [Brockschmidt, Cook, Fuhs, *CAV '13*; Brockschmidt et al, *TACAS '16*]
Searching for Invariants Using SMT

Termination prover needs to find invariants for programs on integers

- Statically before the translation
  [Otto et al, RTA ’10; Ströder et al, JAR ’17, …]  
  → abstract interpretation [Cousot, Cousot, POPL ’77]
  → more in Session 2!

- By counterexample-based reasoning + safety prover: Terminator
  [Cook, Podelski, Rybalchenko, CAV ’06, PLDI ’06]
  → prove termination of single program runs
  → termination argument often generalises

- … also cooperating with removal of terminating rules (as for TRSs): T2 [Brockschmidt, Cook, Fuhs, CAV ’13; Brockschmidt et al, TACAS ’16]

- Using Max-SMT
  [Larraz, Oliveras, Rodríguez-Carbonell, Rubio, FMCAD ’13]
Searching for Invariants Using SMT

Termination prover needs to find invariants for programs on integers

- Statically before the translation
  [Otto et al, RTA ’10; Ströder et al, JAR ’17, …]
  → abstract interpretation [Cousot, Cousot, POPL ’77]
  → more in Session 2!

- By counterexample-based reasoning + safety prover: Terminator
  [Cook, Podelski, Rybalchenko, CAV ’06, PLDI ’06]
  → prove termination of single program runs
  → termination argument often generalises

- … also cooperating with removal of terminating rules (as for TRSs): T2
  [Brockschmidt, Cook, Fuhs, CAV ’13; Brockschmidt et al, TACAS ’16]

- Using Max-SMT
  [Larraz, Oliveras, Rodríguez-Carbonell, Rubio, FMCAD ’13]

Nowadays all SMT-based!
Extensions

- Proving **non**-termination (infinite run is possible from initial states)  
Extensions

- Proving **non**-termination (infinite run is possible **from initial states**)

- Complexity bounds
Extensions

- Proving **non**-termination (infinite run is possible from initial states)

- Complexity bounds

- CTL* model checking for **infinite** state systems based on termination and non-termination provers
  [Cook, Khlaaf, Piterman, *JACM ’17*]
Extensions

- Proving non-termination (infinite run is possible from initial states)
  [Gupta et al, POPL ’08, Brockschmidt et al, FoVeOOS ’11, Chen et al, TACAS ’14, Larraz et al, CAV ’14, Cook et al, FMCAD ’14, …]

- Complexity bounds
  [Alias et al, SAS ’10, Hoffmann, Shao, JFP ’15, Brockschmidt et al, TOPLAS ’16, …]

- CTL* model checking for infinite state systems based on termination and non-termination provers
  [Cook, Khlaaf, Piterman, JACM ’17]

- Beyond sequential programs on integers:
  - structs/classes [Berdine et al, CAV ’06; Otto et al, RTA ’10; …]
  - arrays (pointer arithmetic) [Ströder et al, JAR ’17, …]
  - multi-threaded programs [Cook et al, PLDI ’07, …]
  - …
Recall: Why Care about Termination of Term Rewriting?

- Termination needed by theorem provers

- Translate program $P$ with inductive data structures (trees) to TRS, represent data structures as terms

  $\Rightarrow$ Termination of TRS implies termination of $P$

- Logic programming: Prolog

- (Lazy) functional programming: Haskell [Giesl et al, *TOPLAS ’11*]

- Object-oriented programming: Java [Otto et al, *RTA ’10*]
So far, so good . . .
but do we *really* want to represent 1000000 as $s(s(s(...)))$?!
So far, so good . . .
but do we really want to represent 1000000 as $s(s(s(...)))$?! 

**Drawbacks:**
So far, so good …
but do we really want to represent $1000000$ as $s(s(s(...)))$?!

**Drawbacks:**
- throws away domain knowledge about built-in data types like integers
So far, so good ... 
but do we *really* want to represent 1000000 as $s(s(s(...)))$?! 

**Drawbacks:**
- throws away domain knowledge about built-in data types like integers
- need to analyse recursive rules for `minus`, `quot`, ... over and over
So far, so good ... but do we really want to represent 1000000 as \( s(s(s(...))) \)?!

**Drawbacks:**

- throws away domain knowledge about built-in data types like integers
- need to analyse recursive rules for `minus`, `quot`, ... over and over
- does not benefit from dedicated constraint solvers (e.g., SMT solvers) for arithmetic operations in programs
So far, so good ... 
but do we really want to represent 1000000 as \( s(s(s(...))) \)?!

**Drawbacks:**
- throws away domain knowledge about built-in data types like integers
- need to analyse recursive rules for *minus*, *quot*, ... over and over
- does not benefit from dedicated constraint solvers (e.g., SMT solvers) for arithmetic operations in programs

**Solution:** use **constrained term rewriting**
Term rewriting “with batteries included”

- first-order
- no fixed evaluation strategy
- no fixed order of rules to apply
Constrained Term Rewriting, What’s That?

Term rewriting “with batteries included”
- first-order
- no fixed evaluation strategy
- no fixed order of rules to apply
- typed
Constrained Term Rewriting, What’s That?

Term rewriting “with batteries included”

- first-order
- no fixed evaluation strategy
- no fixed order of rules to apply
- typed
- with pre-defined data structures (integers, arrays, bitvectors, ...), usually from SMT-LIB theories
Constrained Term Rewriting, What’s That?

Term rewriting “with batteries included”

- first-order
- no fixed evaluation strategy
- no fixed order of rules to apply
- typed
- with pre-defined data structures (integers, arrays, bitvectors, ...), usually from SMT-LIB theories
- rewrite rules with SMT constraints
Constrained Term Rewriting, What’s That?

Term rewriting “with batteries included”

- first-order
- no fixed evaluation strategy
- no fixed order of rules to apply
- typed
- with pre-defined data structures (integers, arrays, bitvectors, ...), usually from SMT-LIB theories
- rewrite rules with SMT constraints

⇒ Term rewriting + SMT solving for automated reasoning
Constrained Term Rewriting, What’s That?

Term rewriting “with batteries included”

- first-order
- no fixed evaluation strategy
- no fixed order of rules to apply
- typed
- with pre-defined data structures (integers, arrays, bitvectors, ...), usually from SMT-LIB theories
- rewrite rules with SMT constraints

⇒ Term rewriting + SMT solving for automated reasoning

- General forms available, e.g., Logically Constrained TRSs
  [Kop, Nishida, FroCoS ’13]
Constrained Term Rewriting, What’s That?

Term rewriting “with batteries included”

- first-order
- no fixed evaluation strategy
- no fixed order of rules to apply
- typed
- with pre-defined data structures (integers, arrays, bitvectors, ...), usually from SMT-LIB theories
- rewrite rules with SMT constraints

⇒ Term rewriting + SMT solving for automated reasoning

- General forms available, e.g., Logically Constrained TRSs [Kop, Nishida, FroCoS ’13]
- For program termination: use term rewriting with integers [Falke, Kapur, CADE ’09; Fuhs et al, RTA ’09; Giesl et al, JAR ’17]
Constrained Term Rewriting, What’s That?

Term rewriting “with batteries included”

- first-order
- no fixed evaluation strategy
- no fixed order of rules to apply
- typed
- with pre-defined data structures (integers, arrays, bitvectors, ...), usually from SMT-LIB theories
- rewrite rules with SMT constraints

⇒ Term rewriting + SMT solving for automated reasoning

- General forms available, e.g., Logically Constrained TRSs [Kop, Nishida, FroCoS ’13]
- For program termination: use term rewriting with integers [Falke, Kapur, CADE ’09; Fuhs et al, RTA ’09; Giesl et al, JAR ’17]
- Integer transition systems are a special case of rewrite systems with integers
Example (Constrained Rewrite System)

\[\ell_0(n, r) \rightarrow \ell_1(n, r, \text{Nil})\]
\[\ell_1(n, r, xs) \rightarrow \ell_1(n - 1, r + 1, \text{Cons}(r, xs)) \quad [n > 0]\]
\[\ell_1(n, r, xs) \rightarrow \ell_2(xs) \quad [n = 0]\]
### Example (Constrained Rewrite System)

<table>
<thead>
<tr>
<th>Rule</th>
<th>Target</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \ell_0(n, r) \rightarrow \ell_1(n, r, \text{Nil}) )</td>
<td></td>
</tr>
<tr>
<td>( \ell_1(n, r, xs) \rightarrow \ell_1(n-1, r+1, \text{Cons}(r, xs)) )</td>
<td>( n &gt; 0 )</td>
</tr>
<tr>
<td>( \ell_1(n, r, xs) \rightarrow \ell_2(xs) )</td>
<td>( n = 0 )</td>
</tr>
</tbody>
</table>

Possible rewrite sequence: \( \ell_0(2, 7) \)
Example (Constrained Rewrite System)

\[
\ell_0(n, r) \rightarrow \ell_1(n, r, \text{Nil}) \\
\ell_1(n, r, xs) \rightarrow \ell_1(n - 1, r + 1, \text{Cons}(r, xs)) \quad [n > 0] \\
\ell_1(n, r, xs) \rightarrow \ell_2(xs) \quad [n = 0]
\]

Possible rewrite sequence: \( \ell_0(2, 7) \)

\( \rightarrow \ell_1(2, 7, \text{Nil}) \)
Example (Constrained Rewrite System)

\[
\begin{align*}
\ell_0(n, r) & \rightarrow \ell_1(n, r, \text{Nil}) \\
\ell_1(n, r, xs) & \rightarrow \ell_1(n - 1, r + 1, \text{Cons}(r, xs)) \quad [n > 0] \\
\ell_1(n, r, xs) & \rightarrow \ell_2(xs) \quad [n = 0]
\end{align*}
\]

Possible rewrite sequence:

\[
\ell_0(2, 7) \\
\rightarrow \ell_1(2, 7, \text{Nil}) \\
\rightarrow \ell_1(1, 8, \text{Cons}(7, \text{Nil}))
\]
Example (Constrained Rewrite System)

\[ \ell_0(n, r) \rightarrow \ell_1(n, r, \text{Nil}) \]
\[ \ell_1(n, r, xs) \rightarrow \ell_1(n - 1, r + 1, \text{Cons}(r, xs)) \quad [n > 0] \]
\[ \ell_1(n, r, xs) \rightarrow \ell_2(xs) \quad [n = 0] \]

Possible rewrite sequence:

\[ \ell_0(2, 7) \]
\[ \rightarrow \ell_1(2, 7, \text{Nil}) \]
\[ \rightarrow \ell_1(1, 8, \text{Cons}(7, \text{Nil})) \]
\[ \rightarrow \ell_1(0, 9, \text{Cons}(8, \text{Cons}(7, \text{Nil}))) \]
Example (Constrained Rewrite System)

\[
\begin{align*}
\ell_0(n, r) & \rightarrow \ell_1(n, r, \text{Nil}) \\
\ell_1(n, r, xs) & \rightarrow \ell_1(n - 1, r + 1, \text{Cons}(r, xs)) \quad [n > 0] \\
\ell_1(n, r, xs) & \rightarrow \ell_2(xs) \quad [n = 0]
\end{align*}
\]

Possible rewrite sequence:

\[
\begin{align*}
\ell_0(2, 7) & \\
& \rightarrow \ell_1(2, 7, \text{Nil}) \\
& \rightarrow \ell_1(1, 8, \text{Cons}(7, \text{Nil})) \\
& \rightarrow \ell_1(0, 9, \text{Cons}(8, \text{Cons}(7, \text{Nil}))) \\
& \rightarrow \ell_2(\text{Cons}(8, \text{Cons}(7, \text{Nil})))
\end{align*}
\]

Here 7, 8, ... are predefined constants.

Termination: reuse techniques for TRSs and integer programs [Giesl et al., JAR '17]

Techniques for LCTRSs in Ctrl [Kop, WST '13; Kop, Nishida, LPAR '15]
Example (Constrained Rewrite System)

\[
\ell_0(n, r) \rightarrow \ell_1(n, r, \text{Nil}) \\
\ell_1(n, r, xs) \rightarrow \ell_1(n - 1, r + 1, \text{Cons}(r, xs)) \quad [n > 0] \\
\ell_1(n, r, xs) \rightarrow \ell_2(xs) \quad [n = 0]
\]

Possible rewrite sequence:

\[
\ell_0(2, 7) \rightarrow \ell_1(2, 7, \text{Nil}) \\
\rightarrow \ell_1(1, 8, \text{Cons}(7, \text{Nil})) \\
\rightarrow \ell_1(0, 9, \text{Cons}(8, \text{Cons}(7, \text{Nil}))) \\
\rightarrow \ell_2(\text{Cons}(8, \text{Cons}(7, \text{Nil})))
\]

Here 7, 8, \ldots are predefined constants.
Example (Constrained Rewrite System)

\[
\begin{align*}
\ell_0(n, r) & \rightarrow \ell_1(n, r, \text{Nil}) \\
\ell_1(n, r, xs) & \rightarrow \ell_1(n - 1, r + 1, \text{Cons}(r, xs)) \quad [n > 0] \\
\ell_1(n, r, xs) & \rightarrow \ell_2(xs) \quad [n = 0]
\end{align*}
\]

Possible rewrite sequence: \[\ell_0(2, 7) \rightarrow \ell_1(2, 7, \text{Nil}) \rightarrow \ell_1(1, 8, \text{Cons}(7, \text{Nil})) \rightarrow \ell_1(0, 9, \text{Cons}(8, \text{Cons}(7, \text{Nil}))) \rightarrow \ell_2(\text{Cons}(8, \text{Cons}(7, \text{Nil})))\]

Here 7, 8, \ldots are predefined constants.

Termination: reuse techniques for TRSs and integer programs [Giesl et al, JAR ’17]
Constrained Rewriting by Example

Example (Constrained Rewrite System)

\[
\ell_0(n, r) \rightarrow \ell_1(n, r, \text{Nil})
\]
\[
\ell_1(n, r, xs) \rightarrow \ell_1(n - 1, r + 1, \text{Cons}(r, xs)) \quad [n > 0]
\]
\[
\ell_1(n, r, xs) \rightarrow \ell_2(xs) \quad [n = 0]
\]

Possible rewrite sequence: \[\ell_0(2, 7)\]
\[
\rightarrow \ell_1(2, 7, \text{Nil})
\]
\[
\rightarrow \ell_1(1, 8, \text{Cons}(7, \text{Nil}))
\]
\[
\rightarrow \ell_1(0, 9, \text{Cons}(8, \text{Cons}(7, \text{Nil})))
\]
\[
\rightarrow \ell_2(\text{Cons}(8, \text{Cons}(7, \text{Nil})))
\]

Here 7, 8, \ldots are predefined constants.

Termination: reuse techniques for TRSs and integer programs [Giesl et al, JAR '17]

Techniques for LCTRSs in Ctrl [Kop, WST '13; Kop, Nishida, LPAR '15]
II.3 Termination and Complexity

Proof Certification
Certification: Who Watches the Watchers?

- Termination and complexity analysis tools are large, e.g., AProVE has several 100,000s LOC – most likely with hidden bugs!

Step 1: Require human-readable proof output. But: can be large!

Step 2: Machine-readable XML proof output, can be certified independently by trustworthy tools based on Coq and Isabelle.

∼ 2007/8: projects A3PAT, CoLoR, IsaFoR formalise term rewriting, termination, proof techniques − → automatic proof checkers

- solution: extract source code (Haskell, OCaml, ...) for proof checker − → CeTA tool from IsaFoR

1 E. Contejean, P. Courtieu, J. Forest, O. Pons, X. Urbain: Automated Certified Proofs with CIME3, RTA '11
2 F. Blanqui, A. Koprowski: CoLoR: a Coq library on well-founded rewrite relations and its application to the automated verification of termination certificates, MSCS '11
3 R. Thiemann, C. Sternagel: Certification of Termination Proofs using CeTA, TPHOLs '09
Termination and complexity analysis tools are large, e.g., AProVE has several 100,000s LOC – most likely with hidden bugs!

Observation in early Termination Competitions: some tools disagreed on YES / NO for termination.
Termination and complexity analysis tools are large, e.g., AProVE has several 100,000s LOC – most likely with hidden bugs!

Observation in early Termination Competitions: some tools disagreed on YES / NO for termination

**Step 1**: Require human-readable proof output. But: can be large!
Certification: Who Watches the Watchers?

- Termination and complexity analysis tools are large, e.g., AProVE has several 100,000s LOC – most likely with hidden bugs!
- Observation in early Termination Competitions: some tools disagreed on YES / NO for termination
- **Step 1**: Require human-readable proof output. But: can be large!
- **Step 2**: Machine-readable XML proof output, can be certified independently by **trustworthy** tools based on Coq and Isabelle

---

1 E. Contejean, P. Courtieu, J. Forest, O. Pons, X. Urbain: Automated Certified Proofs with CiME3, RTA '11
2 F. Blanqui, A. Koprowski: CoLoR: a Coq library on well-founded rewrite relations and its application to the automated verification of termination certificates, MSCS '11
3 R. Thiemann, C. Sternagel: Certification of Termination Proofs using CeTA, TPHOLs '09
Termination and complexity analysis tools are large, e.g., AProVE has several 100,000s LOC – most likely with hidden bugs!

Observation in early Termination Competitions: some tools disagreed on YES / NO for termination

Step 1: Require human-readable proof output. But: can be large!

Step 2: Machine-readable XML proof output, can be certified independently by trustworthy tools based on Coq and Isabelle

∼ 2007/8: projects A3PAT\(^1\), CoLoR\(^2\), IsaFoR\(^3\) formalise term rewriting, termination, proof techniques → automatic proof checkers

---

\(^1\) E. Contejean, P. Courtieu, J. Forest, O. Pons, X. Urbain: Automated Certified Proofs with CiME3, RTA ‘11

\(^2\) F. Blanqui, A. Koprowski: CoLoR: a Coq library on well-founded rewrite relations and its application to the automated verification of termination certificates, MSCS ’11

\(^3\) R. Thiemann, C. Sternagel: Certification of Termination Proofs using CeTA, TPHOLs ’09
Certification: Who Watches the Watchers?

- Termination and complexity analysis tools are large, e.g., AProVE has several 100,000s LOC – most likely with hidden bugs!
- Observation in early Termination Competitions: some tools disagreed on YES / NO for termination
- **Step 1:** Require human-readable proof output. But: can be large!
- **Step 2:** Machine-readable XML proof output, can be certified independently by trustworthy tools based on Coq and Isabelle
- ~2007/8: projects A3PAT\(^1\), CoLoR\(^2\), IsaFoR\(^3\) formalise term rewriting, termination, proof techniques → automatic proof checkers
- performance bottleneck: computations in theorem prover

---

\(^{1}\) E. Contejean, P. Courtieu, J. Forest, O. Pons, X. Urbain: *Automated Certified Proofs with CiME3*, RTA '11

\(^{2}\) F. Blanqui, A. Koprowski: *CoLoR: a Coq library on well-founded rewrite relations and its application to the automated verification of termination certificates*, MSCS '11

\(^{3}\) R. Thiemann, C. Sternagel: *Certification of Termination Proofs using CeTA*, TPHOLs '09
Certification: Who Watches the Watchers?

- Termination and complexity analysis tools are large, e.g., AProVE has several 100,000s LOC – most likely with hidden bugs!
- Observation in early Termination Competitions: some tools disagreed on YES / NO for termination

Step 1: Require human-readable proof output. But: can be large!
Step 2: Machine-readable XML proof output, can be certified independently by trustworthy tools based on Coq and Isabelle

≈ 2007/8: projects A3PAT\(^1\), CoLoR\(^2\), IsaFoR\(^3\) formalise term rewriting, termination, proof techniques → automatic proof checkers

- performance bottleneck: computations in theorem prover
- solution: extract source code (Haskell, OCaml, ...) for proof checker

→ CeTA tool from IsaFoR

\(^1\) E. Contejean, P. Courtieu, J. Forest, O. Pons, X. Urbain: Automated Certified Proofs with CiME3, RTA ’11

\(^2\) F. Blanqui, A. Koprowski: CoLoR: a Coq library on well-founded rewrite relations and its application to the automated verification of termination certificates, MSCS ’11

\(^3\) R. Thiemann, C. Sternagel: Certification of Termination Proofs using CeTA, TPHOLs ’09
CeTA can certify proofs for...

If certification unsuccessful: CeTA indicates which part of the proof it could not follow.

4. M. Haslbeck, R. Thiemann: An Isabelle/HOL formalization of AProVE's termination method for LLVM IR, CPP '21

CeTA can certify proofs for...

- termination of TRSs (several flavours), Integer Transition Systems, and LLVM programs

---

4 M. Haslbeck, R. Thiemann: An Isabelle/HOL formalization of AProVE’s termination method for LLVM IR, CPP ’21
CeTA can certify proofs for...

- termination of TRSs (several flavours), Integer Transition Systems, and LLVM programs\(^4\)
- non-termination for TRSs

\(^4\)M. Haslbeck, R. Thiemann: *An Isabelle/HOL formalization of AProVE’s termination method for LLVM IR*, CPP ’21
CeTA can certify proofs for...

- termination of TRSs (several flavours), Integer Transition Systems, and LLVM programs
- non-termination for TRSs
- upper bounds for complexity

---

\(^4\) M. Haslbeck, R. Thiemann: *An Isabelle/HOL formalization of AProVE’s termination method for LLVM IR*, CPP ’21
CeTA can certify proofs for...

- termination of TRSs (several flavours), Integer Transition Systems, and LLVM programs\(^4\)
- non-termination for TRSs
- upper bounds for complexity
- confluence and non-confluence proofs for TRSs

\(^4\) M. Haslbeck, R. Thiemann: *An Isabelle/HOL formalization of AProVE’s termination method for LLVM IR*, CPP ’21
CeTA can certify proofs for...

- termination of TRSs (several flavours), Integer Transition Systems, and LLVM programs\(^4\)
- non-termination for TRSs
- upper bounds for complexity
- confluence and non-confluence proofs for TRSs
- safety: invariants for Integer Transition Systems\(^5\)

\[^4\text{M. Haslbeck, R. Thiemann: An Isabelle/HOL formalization of AProVE’s termination method for LLVM IR, CPP ’21}\]
\[^5\text{M. Brockschmidt, S. Joosten, R. Thiemann, A. Yamada: Certifying Safety and Termination Proofs for Integer Transition Systems, CADE ’17}\]
Proof Certification with CeTA

http://cl-informatik.uibk.ac.at/isafor/

CeTA can certify proofs for...

- termination of TRSs (several flavours), Integer Transition Systems, and LLVM programs
- non-termination for TRSs
- upper bounds for complexity
- confluence and non-confluence proofs for TRSs
- safety: invariants for Integer Transition Systems

If certification unsuccessful:
CeTA indicates which part of the proof it could not follow

---

4 M. Haslbeck, R. Thiemann: An Isabelle/HOL formalization of AProVE’s termination method for LLVM IR, CPP ’21
**TermCOMP with Certification (✓) (1/2)**

### Termination Competition 2022

**Competition-Wide Ranking**

- AProVE+LoAT(4.0811)
- MU-TERM(1.9331)
- TTT2+TcT(1.9062)
- NaTT(1.4268)
- Matchbox(1.3425)
- iRankFinder(1.2594)
- Ultimate(1.2079)
- MultumNonMultum(1.1930)
- NTi+CTi(0.9649)
- SOL(0.9180)
- Wanda(0.8975)

**Advancing-the-State-of-the-Art Ranking**

- Matchbox(67)
- MultumNonMultum(48)
- AProVE+LoAT(31.25)
- SOL(16)
- NaTT(11)
- NTi+CTi(11)
- TTT2+TcT(0.375)
- iRankFinder(0)
- MU-TERM(0)
- Ultimate(0)
- Wanda(0)

### Termination of Rewriting

- **Progress:** 100%, **CPU Time:** 85d 8:05:33, **Node Time:** 34d 3:49:50

#### TRS Standard

<table>
<thead>
<tr>
<th>54200</th>
<th>54199</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. AProVE21</td>
<td>✓1. AProVE21</td>
</tr>
<tr>
<td>2. NaTT 2.5.2</td>
<td>2. AProVE21</td>
</tr>
<tr>
<td>3. NTi+CTi 2.20</td>
<td>3. NTi+CTi 2.20</td>
</tr>
<tr>
<td>4. muterm 6.0.3</td>
<td>4. muterm 6.0.3</td>
</tr>
<tr>
<td>5. NaTT 1.6.2</td>
<td>5. NaTT 1.6.2</td>
</tr>
<tr>
<td>6. muterm 6.0.3</td>
<td>6. muterm 6.0.3</td>
</tr>
</tbody>
</table>

#### SRS Standard

<table>
<thead>
<tr>
<th>54203</th>
<th>54201</th>
</tr>
</thead>
<tbody>
<tr>
<td>2. MutuTerm 6.1</td>
<td>2. AProVE21</td>
</tr>
<tr>
<td>3. NTi+CTi 2.20</td>
<td>3. NTi+CTi 2.20</td>
</tr>
<tr>
<td>4. muterm 6.0.3</td>
<td>4. muterm 6.0.3</td>
</tr>
<tr>
<td>5. NaTT 1.6.2</td>
<td>5. NaTT 1.6.2</td>
</tr>
<tr>
<td>6. muterm 6.0.3</td>
<td>6. muterm 6.0.3</td>
</tr>
</tbody>
</table>

### Termination of Programs

- **Progress:** 100%, **CPU Time:** 3d 2:32:33, **Node Time:** 2d 4:20:44

#### C

<table>
<thead>
<tr>
<th>54224</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Aprove22-C</td>
</tr>
<tr>
<td>2. UltimateAutomizer2022v2</td>
</tr>
</tbody>
</table>

#### C Integer

<table>
<thead>
<tr>
<th>54225</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Aprove22-C</td>
</tr>
<tr>
<td>2. UltimateAutomizer2022v2</td>
</tr>
</tbody>
</table>

#### Integer Transition Systems

<table>
<thead>
<tr>
<th>54213</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. iRankFinder v1.3.2</td>
</tr>
<tr>
<td>2. LoAT TermComp 2021</td>
</tr>
</tbody>
</table>

#### Logic Programming

<table>
<thead>
<tr>
<th>54212</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. NTi+CTi 22</td>
</tr>
<tr>
<td>2. AProVE21</td>
</tr>
</tbody>
</table>

### Complexity Analysis

- **Progress:** 100%, **CPU Time:** 123d 22:10:39, **Node Time:** 42d 19:13:03

#### Derivational Complexity: TRS

<table>
<thead>
<tr>
<th>54235</th>
<th>54234</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. AProVE21</td>
<td>✓1. AProVE21</td>
</tr>
</tbody>
</table>

#### Derivational Complexity: TRS Innermost

<table>
<thead>
<tr>
<th>54221</th>
<th>54227</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓1. tct-trs_v3.2.0_2020-06-28</td>
<td>✓1. tct-trs_v3.2.0_2020-06-28</td>
</tr>
</tbody>
</table>

#### Runtime Complexity: TRS

<table>
<thead>
<tr>
<th>54216</th>
<th>54216</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. AProVE21</td>
<td>1. AProVE21</td>
</tr>
</tbody>
</table>

---

41/104
Let’s zoom in . . .

**Termination of Rewriting**

Progress: 100%, CPU Time: 85d 8:05:33, Node Time: 34d 3:4

<table>
<thead>
<tr>
<th>TRS Standard</th>
<th>SRS Standard</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. AProVE21</td>
<td>1. matchbox-2022-07-22</td>
</tr>
<tr>
<td>✓1. AProVE21</td>
<td>✓1. matchbox-2022-07-22</td>
</tr>
<tr>
<td>2. NaTT 2.3.2</td>
<td>2. MnM3.19c</td>
</tr>
<tr>
<td>3. ttt2-1.20</td>
<td>3. AProVE21</td>
</tr>
<tr>
<td>✓2. ttt2-1.20</td>
<td>✓2. AProVE21</td>
</tr>
<tr>
<td>4. muterm 6.0.3</td>
<td>4. ttt2-1.20</td>
</tr>
<tr>
<td>✓3. NaTT 1.6.2</td>
<td>✓3. ttt2-1.20</td>
</tr>
<tr>
<td>5. NTI_22</td>
<td>5. NaTT 2.3.2</td>
</tr>
<tr>
<td></td>
<td>✓4. NaTT 1.6.2</td>
</tr>
<tr>
<td></td>
<td>6. muterm 6.0.3</td>
</tr>
</tbody>
</table>
Let's zoom in . . .

⇒ proof certification is competitive!
Automated termination analysis for term rewriting and for imperative programs developed in parallel over the last \( \sim 20 \) years.
Automated termination analysis for term rewriting and for imperative programs developed in parallel over the last \( \sim 20 \) years

Term rewriting: handles **inductive data structures** well
Automated termination analysis for term rewriting and for imperative programs developed in parallel over the last \(\sim 20\) years

Term rewriting: handles \textbf{inductive data structures} well

Imperative programs on integers: need to consider \textit{reachability/safety} and \textit{invariants}
Conclusion: Termination Proving for Rewrite Systems

- Automated termination analysis for term rewriting and for imperative programs developed in parallel over the last ~ 20 years
- Term rewriting: handles **inductive data structures** well
- Imperative programs on integers: need to consider **reachability/safety** and **invariants**
- Since a few years cross-fertilisation
Automated termination analysis for term rewriting and for imperative programs developed in parallel over the last \( \sim 20 \) years

Term rewriting: handles \textbf{inductive data structures} well

Imperative programs on integers: need to consider \textbf{reachability/safety} and \textbf{invariants}

Since a few years cross-fertilisation

Constrained term rewriting: best of both worlds as back-end language
Conclusion: Termination Proving for Rewrite Systems

- Automated termination analysis for term rewriting and for imperative programs developed in parallel over the last \(\sim 20\) years
- Term rewriting: handles **inductive data structures** well
- Imperative programs on integers: need to consider reachability/safety and **invariants**
- Since a few years cross-fertilisation
- Constrained term rewriting: best of both worlds as back-end language
- Proof search heavily relies on SMT solving for automation
Automated termination analysis for term rewriting and for imperative programs developed in parallel over the last $\sim$ 20 years.

Term rewriting: handles **inductive data structures** well.

Imperative programs on integers: need to consider **reachability/safety** and **invariants**.

Since a few years cross-fertilisation.

Constrained term rewriting: best of both worlds as back-end language.

Proof search heavily relies on SMT solving for automation.

Needs of termination analysis have also led to better SMT solvers.
Conclusion: Termination Proving for Rewrite Systems

- Automated termination analysis for term rewriting and for imperative programs developed in parallel over the last \( \sim 20 \) years
- Term rewriting: handles **inductive data structures** well
- Imperative programs on integers: need to consider reachability/safety and invariants
- Since a few years cross-fertilisation
- Constrained term rewriting: best of both worlds as back-end language
- Proof search heavily relies on SMT solving for automation
- Needs of termination analysis have also led to better SMT solvers
- More information ... 

[http://termination-portal.org](http://termination-portal.org)
Conclusion: Termination Proving for Rewrite Systems

- Automated termination analysis for term rewriting and for imperative programs developed in parallel over the last $\sim 20$ years
- Term rewriting: handles **inductive data structures** well
- Imperative programs on integers: need to consider reachability/safety and invariants
- Since a few years cross-fertilisation
- Constrained term rewriting: best of both worlds as back-end language
- Proof search heavily relies on SMT solving for automation
- Needs of termination analysis have also led to better SMT solvers
- More information ...

http://termination-portal.org

Behind (almost) every successful termination prover . . .
Conclusion: Termination Proving for Rewrite Systems

- Automated termination analysis for term rewriting and for imperative programs developed in parallel over the last ∼ 20 years
- Term rewriting: handles **inductive data structures** well
- Imperative programs on integers: need to consider **reachability/safety** and **invariants**
- Since a few years cross-fertilisation
- Constrained term rewriting: best of both worlds as back-end language
- Proof search heavily relies on SMT solving for automation
- Needs of termination analysis have also led to better SMT solvers
- More information ...

http://termination-portal.org

**Behind (almost) every successful termination prover ...**

... there is a powerful SAT / SMT solver!
II. Beyond Termination of TRSs
II.1 Termination Analysis of Java Programs via TRSs
execute program \textit{symbolically} from initial states of the program, handle language peculiarities here (\(\rightarrow\) Java: sharing, cyclicity analysis)
execute program symbolically from initial states of the program, handle language peculiarities here (→ Java: sharing, cyclicity analysis)

\[ f: \text{if} \ldots \]
\[ \text{init}(\ldots) \]
\[ \text{else} \]
\[ \text{g: while} \ldots \]
execute program **symbolically** from initial states of the program, handle language peculiarities here (→ Java: sharing, cyclicity analysis)

\[
\begin{align*}
\text{f: } & \text{if } \ldots \text{ else } \ldots \text{ g: while } \ldots \\
& \quad \quad \quad \quad \quad \downarrow \\
& \quad \quad \quad \quad \quad \quad \text{f}(\ldots)
\end{align*}
\]

\text{init}(\ldots)
execute program **symbolically** from initial states of the program, handle language peculiarities here (→ Java: sharing, cyclicity analysis)

\[
\text{f: if } \ldots \\
\quad \ldots \\
\text{else} \\
\quad \ldots \\
\quad \text{g: while } \ldots \\
\quad \ldots \\
\]

\[
\begin{array}{c}
\text{init(...)} \\
\downarrow \\
\text{f(...)} \\
\end{array} \\
\begin{array}{c}
\ldots \\
\end{array} \\
\begin{array}{c}
\text{g(s)} \\
\end{array}
\]
execute program **symbolically** from initial states of the program, handle language peculiarities here (→ Java: sharing, cyclicity analysis)

\[
\begin{align*}
\text{f: if } & \ldots \\
\phantom{\text{f: if }} & \ldots \\
\text{else} & \\
\phantom{\text{else: }} & \ldots \\
\text{g: while } & \ldots \\
\phantom{\text{g: while }} & \ldots
\end{align*}
\]
execute program **symbolically** from initial states of the program, handle language peculiarities here (→ Java: sharing, cyclicity analysis)

use **generalisation** of program states, get **over-approximation** of all possible program runs (≈ control-flow graph with extra info)

closely related: Abstract Interpretation [Cousot and Cousot, *POPL ’77*]
execute program **symbolically** from initial states of the program, handle language peculiarities here (→ Java: sharing, cyclicity analysis)

- use **generalisation** of program states, get **over-approximation** of all possible program runs (≈ control-flow graph with extra info)
- closely related: Abstract Interpretation ([Cousot and Cousot, POPL ’77](#))
- **extract TRS** from cycles in the representation

```
f: if ... 
    ... 
else 
    ... 
g: while ... 
    ... 
```

Diagram:

```
init(...) 
    ↓
  f(...) 
    ↓
... 
  g(s) ← g(t) instance of g(s)
... 
  g(t)
```


execute program symbolically from initial states of the program, handle language peculiarities here (→ Java: sharing, cyclicity analysis)

use generalisation of program states, get over-approximation of all possible program runs (≈ control-flow graph with extra info)

closely related: Abstract Interpretation [Cousot and Cousot, POPL ’77]

extract TRS from cycles in the representation

if TRS terminates
⇒ any concrete program execution can use cycles only finitely often
⇒ the program must terminate

\[
\begin{align*}
\text{f: } & \text{if } \ldots \\
& \ldots \\
& \text{else} \\
& \ldots \\
& \text{g: while } \ldots \\
& \ldots \\
\text{init(\ldots)} & \downarrow \\
\text{f(\ldots)} & \downarrow \\
\downarrow & \\
\ldots & \text{g(\overrightarrow{s})} \\
\downarrow & \text{g(\overrightarrow{t}) \text{ instance of } g(\overrightarrow{s})} \\
\downarrow & \\
\ldots & \\
\end{align*}
\]
Recipe for proving program termination by reusing TRS termination provers
Recipe for proving program termination by reusing TRS termination provers

1. Decide on suitable symbolic representation of abstract program states (abstract domain)
   → here: what data objects can we represent as terms?
Recipe for proving program termination by reusing TRS termination provers

- Decide on suitable symbolic representation of abstract program states
  (abstract domain)
  → here: what data objects can we represent as terms?

- Execute program symbolically from its initial states
Recipe for proving program termination by reusing TRS termination provers

- Decide on suitable symbolic representation of abstract program states (abstract domain)
  → here: what data objects can we represent as terms?
- Execute program symbolically from its initial states
- Use generalisation of program states to get closed finite representation (symbolic execution graph, abstract interpretation)
Recipe for proving program termination by reusing TRS termination provers

- Decide on suitable symbolic representation of abstract program states
  - (abstract domain)
  - here: what data objects can we represent as terms?
- Execute program symbolically from its initial states
- Use generalisation of program states to get closed finite representation (symbolic execution graph, abstract interpretation)
- Extract rewrite rules that “over-approximate” program executions in strongly-connected components of graph
Recipe for proving program termination by reusing TRS termination provers

- Decide on suitable symbolic representation of abstract program states (abstract domain)
  - here: what data objects can we represent as terms?
- Execute program symbolically from its initial states
- Use generalisation of program states to get closed finite representation (symbolic execution graph, abstract interpretation)
- Extract rewrite rules that “over-approximate” program executions in strongly-connected components of graph
- Prove termination of these rewrite rules
  - implies termination of program from initial states
Java Challenges

Java: object-oriented imperative language

- sharing and aliasing (several references to the same object)
- side effects
- cyclic data objects (e.g., `list.next == list`)
- object-orientation with inheritance
- ...


public class MyInt {

    // only wrap a primitive int
    private int val;

    // count "num" up to the value in "limit"
    public static void count(MyInt num, MyInt limit) {
        if (num == null || limit == null) {
            return;
        }
        // introduce sharing
        MyInt copy = num;
        while (num.val < limit.val) {
            copy.val++;
        }
    }
}

Does **count** terminate for all inputs? Why (not)?
(Assume that **num** and **limit** are not references to the same object.)
Approach to Termination Analysis of Java

Tailor two-stage approach to Java [Otto et al, RTA ’10]
Tailor two-stage approach to Java [Otto et al, RTA ’10]

**Back-end:** From rewrite system to termination proof
- Constrained term rewriting with integers [Giesl et al, JAR ’17]
- Termination techniques for rewriting and for integers can be integrated
Approach to Termination Analysis of Java

Tailor two-stage approach to Java [Otto et al, RTA ’10]

**Back-end:** From rewrite system to termination proof
- Constrained term rewriting with integers [Giesl et al, JAR ’17]
- Termination techniques for rewriting and for integers can be integrated

**Front-end:** From Java to constrained rewrite system
- Build *symbolic execution graph* that over-approximates all runs of Java program (abstract interpretation)
- Symbolic execution graph has *invariants* for integers and heap object shape (trees?)
- Extract rewrite system from symbolic execution graph
Approach to Termination Analysis of Java

Tailor two-stage approach to Java [Otto et al, RTA ’10]

**Back-end:** From rewrite system to termination proof
- Constrained term rewriting with integers [Giesl et al, JAR ’17]
- Termination techniques for rewriting and for integers can be integrated

**Front-end:** From Java to constrained rewrite system
- Build *symbolic execution graph* that over-approximates all runs of Java program (abstract interpretation)
- Symbolic execution graph has *invariants* for integers and heap object shape (trees?)
- Extract rewrite system from symbolic execution graph

Implemented in the tool AProVE (→ web interface)

http://aprove.informatik.rwth-aachen.de/
[Otto et al, RTA '10] describe their technique for compiled Java programs: Java Bytecode
[Otto et al, *RTA ’10*] describe their technique for compiled Java programs: **Java Bytecode**

- desugared machine code for a (virtual) stack machine, still has all the (relevant) information from source code
- input for Java interpreter and for many program analysis tools
- somewhat inconvenient for presentation, though . . .
[Otto et al, RTA ’10] describe their technique for compiled Java programs: **Java Bytecode**

- desugared machine code for a (virtual) stack machine
- still has all the (relevant) information from source input for Java interpreter and for many program analysis tools
- somewhat inconvenient for presentation, though
[Otto et al, RTA ’10] describe their technique for compiled Java programs: **Java Bytecode**

- desugared machine code for a (virtual) stack machine, still has all the (relevant) information from source code
- input for Java interpreter and for many program analysis tools
- somewhat inconvenient for presentation, though . . .

Here: **Java source code**
Ingredients for the Abstract Domain

1. program counter value (line number)
2. values of variables (treating int as \( \mathbb{Z} \))
3. over-approximating info on possible variable values
   - integers: use intervals, e.g. \( x \in [4, 7] \) or \( y \in [0, \infty) \)
   - heap memory with objects, **no sharing** unless stated otherwise
   - MyInt(?): maybe null, maybe a MyInt object

**Heap predicates:**
- Two references may be equal: \( o_1 = ? o_2 \)

<table>
<thead>
<tr>
<th>03</th>
<th>num: ( o_1 ), limit: ( o_2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( o_1 ): MyInt(?)</td>
<td></td>
</tr>
<tr>
<td>( o_2 ): MyInt(val = ( i_1 ))</td>
<td></td>
</tr>
<tr>
<td>( i_1 ): [4, 80]</td>
<td></td>
</tr>
</tbody>
</table>
Ingredients for the Abstract Domain

1. program counter value (line number)
2. values of variables (treating int as $\mathbb{Z}$)
3. over-approximating info on possible variable values

- integers: use intervals, e.g. $x \in [4, 7]$ or $y \in [0, \infty)$
- heap memory with objects, **no sharing** unless stated otherwise
- MyInt(?): maybe null, maybe a MyInt object

**Heap predicates:**

- Two references may be equal: $o_1 = ? o_2$
- Two references may share: $o_1 \not\equiv o_2$

<table>
<thead>
<tr>
<th>03</th>
<th>num: $o_1$, limit: $o_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$o_1$</td>
<td>MyInt(?)</td>
</tr>
<tr>
<td>$o_2$</td>
<td>MyInt(val = $i_1$)</td>
</tr>
<tr>
<td>$i_1$</td>
<td>[4, 80]</td>
</tr>
</tbody>
</table>
Ingredients for the Abstract Domain

1. program counter value (line number)
2. values of variables (treating int as \( \mathbb{Z} \))
3. over-approximating info on possible variable values
   - integers: use intervals, e.g. \( x \in [4, 7] \) or \( y \in [0, \infty) \)
   - heap memory with objects, **no sharing** unless stated otherwise
   - MyInt(?): maybe null, maybe a MyInt object

**Heap predicates:**
- Two references may be equal: \( o_1 \equiv o_2 \)
- Two references may share: \( o_1 \owns o_2 \)
- Reference may have cycles: \( o_1 \iota \)

<table>
<thead>
<tr>
<th>03</th>
<th>num: ( o_1 ), limit: ( o_2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( o_1 ): MyInt(?)</td>
<td></td>
</tr>
<tr>
<td>( o_2 ): MyInt(val = ( i_1 ))</td>
<td></td>
</tr>
<tr>
<td>( i_1 ): [4, 80]</td>
<td></td>
</tr>
</tbody>
</table>
Building the Symbolic Execution Graph

public class MyInt {
    private int val;
    static void count(MyInt num, MyInt limit) {
        if (num == null || limit == null)
            return;
        MyInt copy = num;
        while (num.val < limit.val)
            copy.val++;
    }
}
Building the Symbolic Execution Graph

public class MyInt {
    private int val;
    static void count(MyInt num, MyInt limit) {
        1: if (num == null || limit == null) return;
        2: MyInt copy = num;
        3: while (num.val < limit.val) copy.val++;
        4: return;
    }
}

X -----> Y

\( \text{cond} \)

means: refine X with \( \text{cond} \), then evaluate to Y; here combined for brevity (narrowing)
Building the Symbolic Execution Graph

public class MyInt {
    private int val;
    static void count(MyInt num, MyInt limit) {
        1: if (num == null || limit == null) return;
        2: MyInt copy = num;
        3: while (num.val < limit.val) copy.val++;
        7: }
    }

cond means: refine \( X \) with \( \text{cond} \), then evaluate to \( Y \); here combined for brevity (narrowing)
Building the Symbolic Execution Graph

```
public class MyInt {
    private int val;
    static void count(MyInt num, MyInt limit) {
        if (num == null || limit == null) {
            return;
        }
        MyInt copy = num;
        while (num.val < limit.val) {
            copy.val++;
        }
    }
}
```

**X → Y**

means: evaluate X to Y
public class MyInt {
    private int val;
    static void count(MyInt num, MyInt limit) {
        1: if (num == null || limit == null)
        2: return;
        3: MyInt copy = num;
        4: while (num.val < limit.val)
        5: copy.val++;
        6: }
    }
}

Building the Symbolic Execution Graph

A

B

C

D

E

F

G

H
Building the Symbolic Execution Graph

```java
public class MyInt {
    private int val;
    static void count(MyInt num, MyInt limit) {
        if (num == null || limit == null)
            return;
        MyInt copy = num;
        while (num.val < limit.val)
            copy.val++;
    }
}
```

---

**A**
- `o1 = null` (num: `o1`, limit: `o2`)
- `o1`: MyInt(?)
- `o2`: MyInt(?)

**B**
- `o1`: null
- `o2`: MyInt(?)

**C**
- `o1 ≠ null` (num: `o1`, limit: `o2`)
- `o1`: MyInt(val = `i1`)
- `o2`: MyInt(?)
- `i1`: (−∞, ∞)

**D**
- `o2 = null` (num: `o1`, limit: `o2`)
- `o1`: MyInt(val = `i1`)
- `o2`: null
- `i1`: (−∞, ∞)

**E**
- `o2 ≠ null` (num: `o1`, limit: `o2`)
- `o1`: MyInt(val = `i1`)
- `o2`: MyInt(val = `i2`)
- `i1`: (−∞, ∞)
- `i2`: (−∞, ∞)

**F**
- `i1 ≥ i2`

**G**
- `7 | num: o1, ...`
- `...`

**H**
- `i3 = i1 + 1`
- `i3`: (−∞, ∞)
- `i2`: (−∞, ∞)

- `6 | num: o1, limit: o2, copy: o1`
- `o1`: MyInt(val = `i1`)
- `o2`: MyInt(val = `i2`)
- `i1`: (−∞, ∞)
- `i2`: (−∞, ∞)

- `5 | num: o1, limit: o2, copy: o1`
- `o1`: MyInt(val = `i1`)
- `o2`: MyInt(val = `i2`)
- `i1`: (−∞, ∞)
- `i2`: (−∞, ∞)
Building the Symbolic Execution Graph

```
public class MyInt {
    private int val;
    static void count(MyInt num, MyInt limit) {
        if (num == null || limit == null)
            return;
        MyInt copy = num;
        while (num.val < limit.val)
            copy.val++;
    }
}
```

A: $o_1 = \text{null}$
B: $o_1 = \text{null}$
C: $o_1 \neq \text{null}$
D: $o_2 = \text{null}$
E: $o_2 \neq \text{null}$
F: $i_1 \geq i_2$
G:...
H: $i_3 = i_1 + 1$

**Diagram:**
- Node 1: $o_1 : \text{MyInt(?)}, o_2 : \text{MyInt(?)}, num : o_1, limit : o_2$
- Node 2: $o_1 : \text{MyInt(val = i_1)}, o_2 : \text{MyInt(?)}, num : o_1, limit : o_2, copy : o_1$
- Node 3: $o_1 : \text{MyInt(val = i_1)}, o_2 : \text{MyInt(val = i_2)}, num : o_1, limit : o_2, copy : o_1$
- Node 4: $o_1 : \text{MyInt(val = i_1)}, o_2 : \text{MyInt(val = i_2)}, num : o_1, limit : o_2, copy : o_1$
- Node 5: $o_1 : \text{MyInt(val = i_1)}, o_2 : \text{MyInt(val = i_2)}, num : o_1, limit : o_2, copy : o_1$
- Node 6: $o_1 : \text{MyInt(val = i_1)}, o_2 : \text{MyInt(val = i_2)}, num : o_1, limit : o_2, copy : o_1$
- Node 7: $o_1 : \text{MyInt(val = i_1)}, o_2 : \text{MyInt(val = i_2)}, num : o_1,...$

**Annotations:**
- X: $i_3 = i_1 + 1$
- Y: $i_1 \leq i_2$
- X is instance of Y
Symbolic Execution Graphs

- symbolic over-approximation of all computations (abstract interpretation)
- expand nodes until all leaves correspond to program ends
- by suitable generalisation steps (widening), one can always get a finite symbolic execution graph
- state $s_1$ is instance of state $s_2$ if all concrete states described by $s_1$ are also described by $s_2$
Symbolic Execution Graphs

- symbolic over-approximation of all computations (abstract interpretation)
- expand nodes until all leaves correspond to program ends
- by suitable generalisation steps (widening), one can always get a finite symbolic execution graph
- state $s_1$ is instance of state $s_2$ if all concrete states described by $s_1$ are also described by $s_2$

Using Symbolic Execution Graphs for Termination Proofs

- every concrete Java computation corresponds to a computation path in the symbolic execution graph
- symbolic execution graph is called terminating iff it has no infinite computation path
For every class $C$ with $n$ fields, introduce an $n$-ary function symbol $C$

- **term** for $o_1$: $o_1$
- **term** for $o_2$: $\text{MyInt}(i_2)$
- **term** for $o_3$: $\text{null}$
- **term** for $o_4$: $x$ (new variable)
- **term** for $i_1$: $i_1$ with side constraint $i_1 \geq 7$
  
  (add invariant $i_1 \geq 7$ to constrained rewrite rules from state $Q$)
public class A {
    int a;
}

public class B extends A {
    int b;
}

... 
A x = new A();
x.a = 1;

B y = new B();
y.a = 2;
y.b = 3;

Dealing with subclasses:
Dealing with **subclasses**:

- for every class $C$ with $n$ fields, introduce $(n + 1)$-ary function symbol $C$
- first argument: part of the object corresponding to subclasses of $C$
- **term** for $x$: $A(eoc, 1)$
  $\rightarrow$ eoc for end of class
- **term** for $y$: $A(B(eoc, 3), 2)$
Dealing with subclasses:

- for every class $C$ with $n$ fields, introduce $(n + 1)$-ary function symbol $C$
- first argument: part of the object corresponding to subclasses of $C$
- term for $x$: $jlO(A(eoc, 1))$  
  $\rightarrow eoc$ for end of class
- term for $y$: $jlO(A(B(eoc, 3), 2))$
- every class extends Object!  
  ($\rightarrow jlO \equiv java.lang.Object$)
From the Symbolic Execution Graph to Terms and Rules

\[ i_3 = i_1 + 1 \]

\[ i_3 < i_2 \]

\[ i_3 = i_1 + 1 \]

\[ i_3 < i_2 \]

\[ i_3 = i_1 + 1 \]

\[ i_3 < i_2 \]
From the Symbolic Execution Graph to Terms and Rules

State F: \( \ell_F( \text{jO(MyInt(eoc, i_1)), jO(MyInt(eoc, i_2))} ) \)

State H: \( \ell_H( \text{jO(MyInt(eoc, i_1)), jO(MyInt(eoc, i_2))} ) \)
State F: $\ell_F( j\text{O(MyInt(eoc, } i_1)), j\text{O(MyInt(eoc, } i_2)) )$

$\rightarrow$

State H: $\ell_H( j\text{O(MyInt(eoc, } i_1)), j\text{O(MyInt(eoc, } i_2)) )$  \[ i_1 < i_2 \]
### From the Symbolic Execution Graph to Terms and Rules

<table>
<thead>
<tr>
<th>State F:</th>
<th>[ \ell_F( j\text{LO}(\text{MyInt}(\text{eoc}, i_1)), j\text{LO}(\text{MyInt}(\text{eoc}, i_2)) ) ]</th>
</tr>
</thead>
<tbody>
<tr>
<td>( i_3 = i_1 + 1 )</td>
<td></td>
</tr>
</tbody>
</table>
| State H: | \[ \ell_H( j\text{LO}(\text{MyInt}(\text{eoc}, i_1)), j\text{LO}(\text{MyInt}(\text{eoc}, i_2)) ) \]  
\[ i_1 < i_2 \] |
| State H: | \[ \ell_H( j\text{LO}(\text{MyInt}(\text{eoc}, i_1)), j\text{LO}(\text{MyInt}(\text{eoc}, i_2)) ) \] |
| State I: | \[ \ell_F( j\text{LO}(\text{MyInt}(\text{eoc}, i_1 + 1)), j\text{LO}(\text{MyInt}(\text{eoc}, i_2)) ) \] |
From the Symbolic Execution Graph to Terms and Rules

<table>
<thead>
<tr>
<th>5</th>
<th>num: o₁, limit: o₂, copy: o₁</th>
</tr>
</thead>
<tbody>
<tr>
<td>o₁: MyInt(val = i₃)</td>
<td></td>
</tr>
<tr>
<td>o₂: MyInt(val = i₂)</td>
<td></td>
</tr>
<tr>
<td>i₃: (−∞, ∞)</td>
<td></td>
</tr>
<tr>
<td>i₂: (−∞, ∞)</td>
<td></td>
</tr>
</tbody>
</table>

\[ i₃ = i₁ + 1 \]

<table>
<thead>
<tr>
<th>6</th>
<th>num: o₁, limit: o₂, copy: o₁</th>
</tr>
</thead>
<tbody>
<tr>
<td>o₁: MyInt(val = i₁)</td>
<td></td>
</tr>
<tr>
<td>o₂: MyInt(val = i₂)</td>
<td></td>
</tr>
<tr>
<td>i₁: (−∞, ∞)</td>
<td></td>
</tr>
<tr>
<td>i₂: (−∞, ∞)</td>
<td></td>
</tr>
</tbody>
</table>

**State F:** \[ ℓ_F( jLO(MyInt(eoc, i₁)), jLO(MyInt(eoc, i₂)) ) \]

→

**State H:** \[ ℓ_H( jLO(MyInt(eoc, i₁)), jLO(MyInt(eoc, i₂)) ) \]

\[ i₁ < i₂ \]

**State H:** \[ ℓ_H( jLO(MyInt(eoc, i₁)), jLO(MyInt(eoc, i₂)) ) \]

→

**State I:** \[ ℓ_F( jLO(MyInt(eoc, i₁ + 1)), jLO(MyInt(eoc, i₂)) ) \]
From the Symbolic Execution Graph to Terms and Rules

State F: \( \ell_F( jLO(MyInt(eoc, i_1)), jLO(MyInt(eoc, i_2)) ) \)
\[ \rightarrow \]
State H: \( \ell_H( jLO(MyInt(eoc, i_1)), jLO(MyInt(eoc, i_2)) ) \) \[ [i_1 < i_2] \]

State H: \( \ell_H( jLO(MyInt(eoc, i_1)), jLO(MyInt(eoc, i_2)) ) \)
\[ \rightarrow \]
State I: \( \ell_F( jLO(MyInt(eoc, i_1 + 1)), jLO(MyInt(eoc, i_2)) ) \)

Termination easy to show (intuitively: \( i_2 - i_1 \) decreases against bound 0)
Extensions

- **modular** termination proofs and **recursion**
  [Brockschmidt et al, *RTA ’11*]
Extensions

- **modular termination proofs and recursion**
  [Brockschmidt et al, *RTA ’11*]

- proving **reachability** and **non-termination** (uses only symbolic execution graph)
  [Brockschmidt et al, *FoVeOOS ’11*]
Extensions

- **modular** termination proofs and **recursion**
  [Brockschmidt et al, *RTA ’11*]

- proving **reachability** and **non-termination** (uses only symbolic execution graph)
  [Brockschmidt et al, *FoVeOOS ’11*]

- proving termination with **cyclic data objects** (preprocessing in symbolic execution graph)
  [Brockschmidt et al, *CAV ’12*]
Extensions

- **modular** termination proofs and **recursion**  
  [Brockschmidt et al, *RTA ’11*]

- proving **reachability** and **non-termination** (uses only symbolic execution graph)  
  [Brockschmidt et al, *FoVeOOS ’11*]

- proving termination with **cyclic data objects** (preprocessing in symbolic execution graph)  
  [Brockschmidt et al, *CAV ’12*]

- proving upper bounds for **time complexity** (abstracts terms to numbers)  
  [Frohn and Giesl, *iFM ’17*]
Front-Ends for Haskell and Prolog

**Haskell** [Giesl et al, *TOPLAS ’11*]
- lazy evaluation
- polymorphic types
- higher-order
Haskell [Giesl et al, *TOPLAS ’11*]

- lazy evaluation
- polymorphic types
- higher-order

⇒ abstract domain: a single term; extract (non-constrained) TRS
Front-Ends for Haskell and Prolog

**Haskell** [Giesl et al, *TOPLAS ’11*]
- lazy evaluation
- polymorphic types
- higher-order

⇒ abstract domain: a single term; extract (non-constrained) TRS

**Prolog** [Schneider-Kamp et al, *TOCL ’09*; Giesl et al, *PPDP ’12*]
- backtracking
- uses unification instead of matching
- extra-logical language features (e.g., cut)
Front-Ends for Haskell and Prolog

**Haskell** [Giesl et al, *TOPLAS '11*]
- lazy evaluation
- polymorphic types
- higher-order

⇒ abstract domain: a single term; extract (non-constrained) TRS

**Prolog** [Schneider-Kamp et al, *TOCL '09*; Giesl et al, *PPDP '12*]
- backtracking
- uses unification instead of matching
- extra-logical language features (e.g., cut)

⇒ abstract domain based on equivalent **linear** Prolog semantics [Ströder et al, *LOPSTR '11*], tracks which variables are for ground terms vs arbitrary terms
LLVM [Ströder et al, JAR ’17]
- LLVM bitcode: intermediate language of LLVM compiler framework
- clang compiler has prominent frontend for C
- challenges: memory safety, pointer arithmetic
Front-End for LLVM

**LLVM** [Ströder et al, *JAR* ‘17]

- LLVM bytecode: intermediate language of LLVM compiler framework
- `clang` compiler has prominent frontend for C
- challenges: memory safety, pointer arithmetic

⇒ track information about allocated memory and its content; extract Integer Transition System (no `struct` so far)
Conclusion: Termination Analysis for Programs

- Termination proving for (LC)TRSs driven by SAT and SMT solvers
Conclusion: Termination Analysis for Programs

- Termination proving for (LC)TRSs driven by SAT and SMT solvers
- Constrained rewriting: Term rewriting + pre-defined primitive data structures
Conclusion: Termination Analysis for Programs

- Termination proving for (LC)TRSs driven by SAT and SMT solvers
- Constrained rewriting: Term rewriting + pre-defined primitive data structures
- Common theme for analysis of program termination by (constrained) rewriting:
  - Handle language specifics in **front-end**
  - Transitions between program states become (constrained) rewrite rules for **termination back-end**
Termination proving for (LC)TRSs driven by SAT and SMT solvers

Constrained rewriting: Term rewriting + pre-defined primitive data structures

Common theme for analysis of program termination by (constrained) rewriting:
  - Handle language specifics in front-end
  - Transitions between program states become (constrained) rewrite rules for termination back-end

Works across paradigms: Java, Haskell, Prolog, ...
II.2 Complexity Analysis for Term Rewriting
What is *Term Rewriting*?

(1) Core functional programming language
    without many restrictions (and features) of “real” FP:

Example (Term Rewrite System (TRS) $R$)

\[
\text{double}(s(s(s(0)))) \rightarrow R \text{ s(s(s(s(s(s(0)))))))}
\]

Compute “double of 3 is 6”:

\[
\text{double}(s(s(s(0)))) \rightarrow R \text{ s(s(s(s(s(s(0)))))))}
\]

in 4 steps with $\rightarrow R$.
What is Term Rewriting?

(1) Core functional programming language without many restrictions (and features) of “real” FP:

- first-order (usually)
- no fixed evaluation strategy
- untyped
- no pre-defined data structures (integers, arrays, . . .)
What is *Term Rewriting*?

(1) Core functional programming language without many restrictions (and features) of “real” FP:

- first-order (usually)
- no fixed evaluation strategy
- untyped
- no pre-defined data structures (integers, arrays, ...)

(2) Syntactic approach for reasoning in equational first-order logic

Example (Term Rewrite System (TRS) $R$)

\[
\text{double}(0) \rightarrow 0 \\
\text{double}(s(x)) \rightarrow s(s(\text{double}(x)))
\]

Compute “double of 3 is 6”:

\[
\text{double}(s(s(s(0)))) \rightarrow R \\
\text{double}(s(s(s(\text{double}(s(s(0))))))) \rightarrow R \\
\text{double}(s(s(s(s(s(s(0)))))))) \rightarrow R \\
\text{double}(s(s(s(s(s(s(s(s(0))))))))) \rightarrow R
\]

in 4 steps.
What is **Term Rewriting**?

(1) Core functional programming language without many restrictions (and features) of “real” FP:
- first-order (usually)
- no fixed evaluation strategy
- untyped
- no pre-defined data structures (integers, arrays, . . .)

(2) Syntactic approach for reasoning in equational first-order logic

Example (Term Rewrite System (TRS) \( R \))

\[
\text{double}(0) \rightarrow 0 \\
\text{double}(s(x)) \rightarrow s(s(\text{double}(x)))
\]
What is *Term Rewriting*?

(1) Core functional programming language without many restrictions (and features) of “real” FP:
- first-order (usually)
- no fixed evaluation strategy
- untyped
- no pre-defined data structures (integers, arrays, ...)

(2) Syntactic approach for reasoning in equational first-order logic

Compute “double of 3 is 6”:
\[ \text{double}(\text{double}(\text{double}(0))) \]

**Example (Term Rewrite System (TRS) \( \mathcal{R} \))**

\[
\begin{align*}
\text{double}(0) & \rightarrow 0 \\
\text{double}(\text{double}(x)) & \rightarrow \text{double}(\text{double}(x)) \\
\end{align*}
\]
What is *Term Rewriting*?

(1) Core functional programming language without many restrictions (and features) of “real” FP:
- first-order (usually)
- no fixed evaluation strategy
- untyped
- no pre-defined data structures (integers, arrays, ...)

(2) Syntactic approach for reasoning in equational first-order logic

Example (Term Rewrite System (TRS) $\mathcal{R}$)

- $\text{double}(0) \rightarrow 0$
- $\text{double}(s(x)) \rightarrow s(s(\text{double}(x)))$

Compute “double of 3 is 6”:

$\text{double}(s(s(s(0)))) \rightarrow_{\mathcal{R}} s(s(\text{double}(s(s(0))))))$
What is *Term Rewriting*?

(1) Core functional programming language without many restrictions (and features) of “real” FP:
- first-order (usually)
- no fixed evaluation strategy
- untyped
- no pre-defined data structures (integers, arrays, ...)

(2) Syntactic approach for reasoning in equational first-order logic

**Example (Term Rewrite System (TRS) $\mathcal{R}$)**

<table>
<thead>
<tr>
<th>Rule</th>
<th>Equation</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{double}(0)$ $\rightarrow$</td>
<td>0</td>
</tr>
<tr>
<td>$\text{double}(s(x))$ $\rightarrow$</td>
<td>$s(s(\text{double}(x)))$</td>
</tr>
</tbody>
</table>

Compute “double of 3 is 6”:

$$
\text{double}(s(s(s(0)))) \xrightarrow{\mathcal{R}} \text{double}(s(s(s(0))))
\xrightarrow{\mathcal{R}} s(s(s(s(s(s(s(s(0))))))))
$$
What is Term Rewriting?

(1) Core functional programming language without many restrictions (and features) of “real” FP:
- first-order (usually)
- no fixed evaluation strategy
- untyped
- no pre-defined data structures (integers, arrays, ...)

(2) Syntactic approach for reasoning in equational first-order logic

Example (Term Rewrite System (TRS) $\mathcal{R}$)

- double($0$) $\rightarrow 0$
- double($s(x)$) $\rightarrow s(s(double(x)))$

Compute “double of 3 is 6”:
- $double(s(s(s(0))))$
- $\rightarrow_{\mathcal{R}} s(s(double(s(s(0)))))$
- $\rightarrow_{\mathcal{R}} s(s(s(s(double(s(0)))))))$
- $\rightarrow_{\mathcal{R}} s(s(s(s(s(s(double(0))))))))$
What is *Term Rewriting*?

(1) Core functional programming language without many restrictions (and features) of “real” FP:

- first-order (usually)
- no fixed evaluation strategy
- untyped
- no pre-defined data structures (integers, arrays, ...)

(2) Syntactic approach for reasoning in equational first-order logic

**Example (Term Rewrite System (TRS) $\mathcal{R}$)**

<table>
<thead>
<tr>
<th>$\text{double}(0)$</th>
<th>$\rightarrow$</th>
<th>$0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{double}(s(x))$</td>
<td>$\rightarrow$</td>
<td>$s(s(\text{double}(x)))$</td>
</tr>
</tbody>
</table>

Compute “double of 3 is 6”:

$$
\begin{align*}
\text{double}(s(s(s(0)))) & \rightarrow_{\mathcal{R}} s(s(\text{double}(s(s(0))))) \\
& \rightarrow_{\mathcal{R}} s(s(s(s(\text{double}(s(s(0))))))) \\
& \rightarrow_{\mathcal{R}} s(s(s(s(s(\text{double}(s(s(0)))))))) \\
& \rightarrow_{\mathcal{R}} s(s(s(s(s(s(\text{double}(s(s(0)))))))))
\end{align*}
$$
What is *Term Rewriting*?

(1) Core functional programming language
   without many restrictions (and features) of “real” FP:
   - first-order (usually)
   - no fixed evaluation strategy
   - untyped
   - no pre-defined data structures (integers, arrays, . . .)

(2) Syntactic approach for reasoning in equational first-order logic

Example (Term Rewrite System (TRS) $\mathcal{R}$)

- $\text{double}(0) \rightarrow 0$
- $\text{double}(s(x)) \rightarrow s(s(\text{double}(x)))$

Compute “double of 3 is 6”:

- $\text{double}(s^3(0))$
- $\rightarrow_\mathcal{R} s^2(\text{double}(s^2(0)))$
- $\rightarrow_\mathcal{R} s^4(\text{double}(s(0)))$
- $\rightarrow_\mathcal{R} s^6(\text{double}(0))$
- $\rightarrow_\mathcal{R} s^6(0)$
What is Term Rewriting?

(1) Core functional programming language without many restrictions (and features) of “real” FP:
- first-order (usually)
- no fixed evaluation strategy
- untyped
- no pre-defined data structures (integers, arrays, . . .)

(2) Syntactic approach for reasoning in equational first-order logic

Example (Term Rewrite System (TRS) \( \mathcal{R} \))

\[
\begin{align*}
double(0) & \rightarrow 0 \\
double(s(x)) & \rightarrow s(s(double(x)))
\end{align*}
\]

Compute “double of 3 is 6”:

\[
\begin{align*}
double(s^3(0)) & \rightarrow_\mathcal{R} s^2(double(s^2(0))) \\
& \rightarrow_\mathcal{R} s^4(double(s(0))) \\
& \rightarrow_\mathcal{R} s^6(double(0)) \\
& \rightarrow_\mathcal{R} s^6(0)
\end{align*}
\]
in 4 steps with \( \rightarrow_\mathcal{R} \)
What is Complexity of Term Rewriting?

**Given:** TRS \( R \) (e.g., \{ \text{double}(0) \rightarrow 0, \text{double}(s(x)) \rightarrow s(s(\text{double}(x))) \})
What is Complexity of Term Rewriting?

**Given:** TRS $\mathcal{R}$ (e.g., $\{ \text{double}(0) \rightarrow 0, \text{double}(s(x)) \rightarrow s(s(\text{double}(x))) \}$)

**Question:** How long can a $\rightarrow \mathcal{R}$ sequence from a term of size $n$ become? (worst case)
What is Complexity of Term Rewriting?

Given: TRS $R$ (e.g., \{ \text{double}(0) \rightarrow 0, \text{double}(s(x)) \rightarrow s(s(\text{double}(x))) \})

Question: How long can a $\rightarrow R$ sequence from a term of size $n$ become? (worst case)

Here: Does $R$ have complexity $\Theta(n)$?
What is *Complexity* of Term Rewriting?

**Given:** TRS $\mathcal{R}$ (e.g., \{ double(0) $\rightarrow$ 0, double(s(x)) $\rightarrow$ s(s(double(x))) \})

**Question:** How long can a $\rightarrow_{\mathcal{R}}$ sequence from a term of size $n$ become? (worst case)

**Here:** Does $\mathcal{R}$ have complexity $\Theta(n)$?

(1) Yes!
What is *Complexity* of Term Rewriting?

**Given:** TRS $\mathcal{R}$ (e.g., \{ $\text{double}(0) \rightarrow 0$, $\text{double}(s(x)) \rightarrow s(s(\text{double}(x)))$ \})

**Question:** How long can a $\rightarrow \mathcal{R}$ sequence from a term of size $n$ become? (worst case)

**Here:** Does $\mathcal{R}$ have complexity $\Theta(n)$?

(1) Yes!

$$\text{double}(s^{n-2}(0)) \rightarrow_{\mathcal{R}}^{n-1} s^{2n-4}(0)$$
What is *Complexity* of Term Rewriting?

**Given:** TRS \( R \) (e.g., \{ double(0) \rightarrow 0, \text{double}(s(x)) \rightarrow s(\text{double}(x))) \})

**Question:** How long can a \( \rightarrow R \) sequence from a term of size \( n \) become? (worst case)

**Here:** Does \( R \) have complexity \( \Theta(n) \)?

(1) Yes!

\[
\text{double}(s^{n-2}(0)) \rightarrow_R^{n-1} s^{2n-4}(0)
\]

- **basic terms** \( f(t_1, \ldots, t_n) \) with \( t_i \) constructor terms allow only \( n \) steps
What is *Complexity* of Term Rewriting?

**Given:** TRS $\mathcal{R}$ (e.g., \{ double(0) $\rightarrow$ 0, double(s(x)) $\rightarrow$ s(s(double(x))) \})

**Question:** How long can a $\rightarrow_{\mathcal{R}}$ sequence from a term of size $n$ become? (worst case)

**Here:** Does $\mathcal{R}$ have complexity $\Theta(n)$?

(1) Yes!

$$\text{double}(s^{n-2}(0)) \rightarrow_{\mathcal{R}}^{n-1} s^{2n-4}(0)$$

- basic terms $f(t_1, \ldots, t_n)$ with $t_i$ constructor terms allow only $n$ steps
- runtime complexity $rc_{\mathcal{R}}(n)$: basic terms as start terms
What is Complexity of Term Rewriting?

**Given:** TRS $\mathcal{R}$ (e.g., \{ $\text{double}(0) \rightarrow 0$, $\text{double}(s(x)) \rightarrow s(s(\text{double}(x)))$ \})

**Question:** How long can a $\rightarrow_{\mathcal{R}}$ sequence from a term of size $n$ become? (worst case)

**Here:** Does $\mathcal{R}$ have complexity $\Theta(n)$?

(1) Yes!

$$\text{double}(s^{n-2}(0)) \rightarrow_{\mathcal{R}}^{n-1} s^{2n-4}(0)$$

- basic terms $f(t_1, \ldots, t_n)$ with $t_i$ constructor terms allow only $n$ steps
- runtime complexity $rc_{\mathcal{R}}(n)$: basic terms as start terms
- $rc_{\mathcal{R}}(n)$ for program analysis
What is *Complexity* of Term Rewriting?

Given: TRS $R$ (e.g., \{ \text{double}(0) \rightarrow 0, \text{double}(s(x)) \rightarrow s(s(\text{double}(x))) \})

**Question:** How long can a $\rightarrow_R$ sequence from a term of size $n$ become? (worst case)

**Here:** Does $R$ have complexity $\Theta(n)$?

1. Yes!
   \[
   \text{double}(s^{n-2}(0)) \rightarrow_R^{n-1} s^{2n-4}(0)
   \]
   - basic terms $f(t_1, \ldots, t_n)$ with $t_i$ constructor terms allow only $n$ steps
   - runtime complexity $rc_R(n)$: basic terms as start terms
   - $rc_R(n)$ for program analysis

2. No!
What is *Complexity* of Term Rewriting?

Given: TRS $\mathcal{R}$ (e.g., \{ $\mathsf{double}(0) \rightarrow 0$, $\mathsf{double}(s(x)) \rightarrow s(s(\mathsf{double}(x)))$ \})

Question: How long can a $\rightarrow \mathcal{R}$ sequence from a term of size $n$ become?

Here: Does $\mathcal{R}$ have complexity $\Theta(n)$?

(1) Yes!

\[ \mathsf{double}(s^{n-2}(0)) \rightarrow^{n-1} \mathcal{R} s^{2n-4}(0) \]

- basic terms $f(t_1, \ldots, t_n)$ with $t_i$ constructor terms allow only $n$ steps
- runtime complexity $rc_{\mathcal{R}}(n)$: basic terms as start terms
- $rc_{\mathcal{R}}(n)$ for program analysis

(2) No!

$\mathsf{double}^3(s(0)) \rightarrow^2 \mathcal{R} \mathsf{double}^2(s^2(0)) \rightarrow^3 \mathcal{R} \mathsf{double}(s^4(0)) \rightarrow^5 \mathcal{R} s^8(0)$ in 10 steps
What is *Complexity of Term Rewriting*?

**Given:** TRS $\mathcal{R}$ (e.g., \{ $\text{double}(0) \rightarrow 0$, $\text{double}(s(x)) \rightarrow s(s(\text{double}(x)))$ \})

**Question:** How long can a $\rightarrow_{\mathcal{R}}$ sequence from a term of size $n$ become? (worst case)

**Here:** Does $\mathcal{R}$ have complexity $\Theta(n)$?

1. **Yes!**
   
   $$\text{double}(s^{n-2}(0)) \rightarrow_{\mathcal{R}}^{n-1} s^{2n-4}(0)$$
   
   - basic terms $f(t_1, \ldots, t_n)$ with $t_i$ constructor terms allow only $n$ steps
   - runtime complexity $rc_{\mathcal{R}}(n)$: basic terms as start terms
   - $rc_{\mathcal{R}}(n)$ for program analysis

2. **No!**
   
   $$\text{double}^3(s(0)) \rightarrow_{\mathcal{R}}^2 \text{double}^2(s^2(0)) \rightarrow_{\mathcal{R}}^3 \text{double}(s^4(0)) \rightarrow_{\mathcal{R}}^5 s^8(0)$$ in 10 steps
   
   - $\text{double}^{n-2}(s(0))$ allows $\Theta(2^n)$ many steps to $s^{2n-2}(0)$
What is *Complexity* of Term Rewriting?

**Given:** TRS $\mathcal{R}$ (e.g., $\{ \text{double}(0) \rightarrow 0, \text{double}(s(x)) \rightarrow s(s(\text{double}(x))) \}$)

**Question:** How long can a $\rightarrow_{\mathcal{R}}$ sequence from a term of size $n$ become? (worst case)

**Here:** Does $\mathcal{R}$ have complexity $\Theta(n)$?

(1) Yes!

$\text{double}(s^{n-2}(0)) \rightarrow_{\mathcal{R}}^{n-1} s^{2n-4}(0)$

- basic terms $f(t_1, \ldots, t_n)$ with $t_i$ constructor terms allow only $n$ steps
- runtime complexity $rc_{\mathcal{R}}(n)$: basic terms as start terms
- $rc_{\mathcal{R}}(n)$ for program analysis

(2) No!

$\text{double}^3(s(0)) \rightarrow_{\mathcal{R}}^2 \text{double}^2(s^2(0)) \rightarrow_{\mathcal{R}}^3 \text{double}(s^4(0)) \rightarrow_{\mathcal{R}}^5 s^8(0)$ in 10 steps

- $\text{double}^{n-2}(s(0))$ allows $\Theta(2^n)$ many steps to $s^{2n-2}(0)$
- derivational complexity $dc_{\mathcal{R}}(n)$: no restrictions on start terms
What is *Complexity* of Term Rewriting?

**Given:** TRS $\mathcal{R}$ (e.g., \{ \text{double}(0) \rightarrow 0, \text{double}(s(x)) \rightarrow s(s(\text{double}(x)))) \})

**Question:** How long can a $\rightarrow_\mathcal{R}$ sequence from a term of size $n$ become? (worst case)

**Here:** Does $\mathcal{R}$ have complexity $\Theta(n)$?

(1) Yes!

\[
\text{double}(s^{n-2}(0)) \rightarrow_{\mathcal{R}}^{n-1} s^{2n-4}(0)
\]

- basic terms $f(t_1, \ldots, t_n)$ with $t_i$ constructor terms allow only $n$ steps
- runtime complexity $rc_{\mathcal{R}}(n)$: basic terms as start terms
- $rc_{\mathcal{R}}(n)$ for *program analysis*

(2) No!

\[
\text{double}^3(s(0)) \rightarrow_{\mathcal{R}}^2 \text{double}^2(s^2(0)) \rightarrow_{\mathcal{R}}^3 \text{double}(s^4(0)) \rightarrow_{\mathcal{R}}^5 s^8(0)
\]

- $\text{double}^{n-2}(s(0))$ allows $\Theta(2^n)$ many steps to $s^{2n-2}(0)$
- derivational complexity $dc_{\mathcal{R}}(n)$: no restrictions on start terms
- $dc_{\mathcal{R}}(n)$ for *equational reasoning*: cost of solving the word problem $\mathcal{E} \models s \equiv t$ by rewriting $s$ and $t$ via an equivalent convergent TRS $\mathcal{R}_E$
Complexity Analysis for TRSs: Overview

1. Introduction
2. Automatically Finding Upper Bounds
3. Automatically Finding Lower Bounds
4. Transformational Techniques
5. Analysing Program Complexity via TRS Complexity
6. Current Developments
1989: Derivational complexity introduced, linked to termination proofs\textsuperscript{6}

\textsuperscript{6} D. Hofbauer, C. Lautemann: *Termination proofs and the length of derivations*, RTA ’89
A Short Timeline (1/2)

1989: Derivational complexity introduced, linked to termination proofs\textsuperscript{6}

2001: Techniques for polynomial upper complexity bounds\textsuperscript{7}

---

\textsuperscript{6} D. Hofbauer, C. Lautemann: *Termination proofs and the length of derivations*, RTA ’89

\textsuperscript{7} G. Bonfante, A. Cichon, J. Marion, and H. Touzet: *Algorithms with polynomial interpretation termination proof*, JFP ’01
A Short Timeline (1/2)

1989: Derivational complexity introduced, linked to termination proofs\textsuperscript{6}
2001: Techniques for polynomial upper complexity bounds\textsuperscript{7}
2008: Runtime complexity introduced with first analysis techniques\textsuperscript{8}

\textsuperscript{6} D. Hofbauer, C. Lautemann: \textit{Termination proofs and the length of derivations}, RTA '89
\textsuperscript{7} G. Bonfante, A. Cichon, J. Marion, and H. Touzet: \textit{Algorithms with polynomial interpretation termination proof}, JFP '01
\textsuperscript{8} N. Hirokawa, G. Moser: \textit{Automated complexity analysis based on the dependency pair method}, IJCAR '08
A Short Timeline (1/2)

1989: Derivational complexity introduced, linked to termination proofs$^6$
2001: Techniques for polynomial upper complexity bounds$^7$
2008: Runtime complexity introduced with first analysis techniques$^8$
2008: First automated tools to find complexity bounds: TcT$^9$, CaT$^{10}$

---

$^6$ D. Hofbauer, C. Lautemann: *Termination proofs and the length of derivations*, RTA '89
$^7$ G. Bonfante, A. Cichon, J. Marion, and H. Touzet: *Algorithms with polynomial interpretation termination proof*, JFP '01
$^8$ N. Hirokawa, G. Moser: *Automated complexity analysis based on the dependency pair method*, IJCAR '08
A Short Timeline (1/2)

1989: Derivational complexity introduced, linked to termination proofs\(^6\)
2001: Techniques for polynomial upper complexity bounds\(^7\)
2008: Runtime complexity introduced with first analysis techniques\(^8\)
2008: First automated tools to find complexity bounds: TcT\(^9\), CaT\(^10\)
2008: First complexity analysis categories in the Termination Competition

\[\text{http://termination-portal.org/wiki/Termination\_Competition}\]

\(^6\) D. Hofbauer, C. Lautemann: *Termination proofs and the length of derivations*, RTA '89
\(^7\) G. Bonfante, A. Cichon, J. Marion, and H. Touzet: *Algorithms with polynomial interpretation termination proof*, JFP '01
\(^8\) N. Hirokawa, G. Moser: *Automated complexity analysis based on the dependency pair method*, IJCAR '08
A Short Timeline (1/2)

1989: Derivational complexity introduced, linked to termination proofs

2001: Techniques for polynomial upper complexity bounds

2008: Runtime complexity introduced with first analysis techniques

2008: First automated tools to find complexity bounds: TcT, CaT

2008: First complexity analysis categories in the Termination Competition

http://termination-portal.org/wiki/Termination_Competition

---

6 D. Hofbauer, C. Lautemann: *Termination proofs and the length of derivations*, RTA '89

7 G. Bonfante, A. Cichon, J. Marion, and H. Touzet: *Algorithms with polynomial interpretation termination proof*, JFP '01

8 N. Hirokawa, G. Moser: *Automated complexity analysis based on the dependency pair method*, IJCAR '08


2022: Termination Competition 2022 with complexity analysis tools AProVE\textsuperscript{11}, TcT in August 2022

https://termcomp.github.io/Y2022

Some Definitions

Definition (Derivation Height $dh$)

For a term $t \in \mathcal{T}(\mathcal{F}, \mathcal{V})$ and a relation $\rightarrow$, the derivation height is:

$$dh(t, \rightarrow) = \sup \{ n \mid \exists t'. t \rightarrow^n t' \}$$

If $t$ starts an infinite $\rightarrow$-sequence, we set $dh(t, \rightarrow) = \omega$. 

Definition (Derivational Complexity $dc$)

For a TRS $R$, the derivational complexity is:

$$dc_R(n) = \sup \{ dh(t, \rightarrow_R) \mid t \in \mathcal{T}(\mathcal{F}, \mathcal{V}), |t| \leq n \}$$

$dc_R(n)$: length of the longest $\rightarrow_R$-sequence from a term of size at most $n$.

Example: For $R$ for `double`, we have $dc_R(n) \in \Theta(2^n)$. 

Some Definitions

**Definition (Derivation Height $dh$)**

For a term $t \in \mathcal{T}(\mathcal{F}, \mathcal{V})$ and a relation $\rightarrow$, the **derivation height** is:

$$dh(t, \rightarrow) = \sup \{ n \mid \exists t'. t \rightarrow^n t' \}$$

If $t$ starts an infinite $\rightarrow$-sequence, we set $dh(t, \rightarrow) = \omega$.

dh$(t, \rightarrow)$: length of the longest $\rightarrow$-sequence from $t$. 

**Example:**

$\text{dh}($double$(s(s(s(0))))), \rightarrow_R) = 4$
Some Definitions

Definition (Derivation Height $dh$)

For a term $t \in \mathcal{T}(\mathcal{F}, \mathcal{V})$ and a relation $\rightarrow$, the derivation height is:

$$dh(t, \rightarrow) = \sup \{ n \mid \exists t'. t \rightarrow^n t' \}$$

If $t$ starts an infinite $\rightarrow$-sequence, we set $dh(t, \rightarrow) = \omega$.

$dh(t, \rightarrow)$: length of the longest $\rightarrow$-sequence from $t$.

Example: $dh(\text{double}(s(s(s(0))))), \rightarrow_{\mathcal{R}}) = 4$
Some Definitions

Definition (Derivation Height $dh$)

For a term $t \in \mathcal{T}(\mathcal{F}, \mathcal{V})$ and a relation $\rightarrow$, the *derivation height* is:

$$ dh(t, \rightarrow) = \sup \{ n \mid \exists t'. t \rightarrow^n t' \} $$

If $t$ starts an infinite $\rightarrow$-sequence, we set $dh(t, \rightarrow) = \omega$.

$dh(t, \rightarrow)$: length of the longest $\rightarrow$-sequence from $t$.

**Example:** $dh(\text{double}(s(s(s(0)))), \rightarrow_{R}) = 4$

Definition (Derivational Complexity $dc$)

For a TRS $R$, the *derivational complexity* is:

$$ dc_R(n) = \sup \{ dh(t, \rightarrow_R) \mid t \in \mathcal{T}(\mathcal{F}, \mathcal{V}), |t| \leq n \} $$
Some Definitions

**Definition (Derivation Height $dh$)**

For a term $t \in T(F, V)$ and a relation $\rightarrow$, the **derivation height** is:

$$dh(t, \rightarrow) = \sup \{ n \mid \exists t'. t \rightarrow^n t' \}$$

If $t$ starts an infinite $\rightarrow$-sequence, we set $dh(t, \rightarrow) = \omega$.

$dh(t, \rightarrow)$: length of the longest $\rightarrow$-sequence from $t$.

**Example:** $dh(\text{double}(s(s(s(0)))), \rightarrow_R) = 4$

**Definition (Derivational Complexity $dc$)**

For a TRS $R$, the **derivational complexity** is:

$$dc_R(n) = \sup \{ dh(t, \rightarrow_R) \mid t \in T(F, V), |t| \leq n \}$$

$dc_R(n)$: length of the longest $\rightarrow_R$-sequence from a term of size at most $n$.
Some Definitions

Definition (Derivation Height $dh$)

For a term $t \in \mathcal{T}(\mathcal{F}, \mathcal{V})$ and a relation $\rightarrow$, the derivation height is:

$$dh(t, \rightarrow) = \sup \{ n \mid \exists t'. t \rightarrow^n t' \}$$

If $t$ starts an infinite $\rightarrow$-sequence, we set $dh(t, \rightarrow) = \omega$.

dh$(t, \rightarrow)$: length of the longest $\rightarrow$-sequence from $t$.

Example: $dh(\text{double}(s(s(s(0))))), \rightarrow_\mathcal{R}) = 4$

Definition (Derivational Complexity $dc$)

For a TRS $\mathcal{R}$, the derivational complexity is:

$$dc_\mathcal{R}(n) = \sup \{ dh(t, \rightarrow_\mathcal{R}) \mid t \in \mathcal{T}(\mathcal{F}, \mathcal{V}), |t| \leq n \}$$

dc$_\mathcal{R}(n)$: length of the longest $\rightarrow_\mathcal{R}$-sequence from a term of size at most $n$

Example: For $\mathcal{R}$ for double, we have $dc_\mathcal{R}(n) \in \Theta(2^n)$. 
The Bad News for automation:

For a given TRS $R$, the following questions are undecidable:

- $dc_R(n) = \omega$ for some $n$? (→ termination!)
- $dc_R(n)$ polynomially bounded?

Goal: find approximations for derivational complexity

Initial focus: find upper bounds $dc_R(n) \in O(...)$

A. Schnabl and J. G. Simonsen: The exact hardness of deciding derivational and runtime complexity, CSL '11

---
Upper Bounds

The Bad News for automation:

For a given TRS $\mathcal{R}$, the following questions are undecidable:

- $d_{c_\mathcal{R}}(n) = \omega$ for some $n$? ($\rightarrow$ termination!)
Upper Bounds

The Bad News for automation:

For a given TRS $\mathcal{R}$, the following questions are undecidable:

- $dc_\mathcal{R}(n) = \omega$ for some $n$? ($\rightarrow$ termination!)
- $dc_\mathcal{R}(n)$ polynomially bounded?\(^{12}\)

---

\(^{12}\) A. Schnabl and J. G. Simonsen: *The exact hardness of deciding derivational and runtime complexity*, CSL ’11
The Bad News for automation:

For a given TRS $\mathcal{R}$, the following questions are undecidable:

- $d_{c_{\mathcal{R}}}(n) = \omega$ for some $n$? ($\rightarrow$ termination!)
- $d_{c_{\mathcal{R}}}(n)$ polynomially bounded?

**Goal:** find approximations for derivational complexity

**Initial focus:** find upper bounds

$$d_{c_{\mathcal{R}}}(n) \in O(...)$$

---

12 A. Schnabl and J. G. Simonsen: *The exact hardness of deciding derivational and runtime complexity*, CSL ’11
Example (double)

\[
\begin{align*}
\text{double}(0) & \rightarrow 0 \\
\text{double}(s(x)) & \rightarrow s(s(\text{double}(x)))
\end{align*}
\]
Example (double)

\[
\begin{align*}
\text{double}(0) & \succ 0 \\
\text{double}(s(x)) & \succ s(s(\text{double}(x)))
\end{align*}
\]

Show \( dc_\mathcal{R}(n) < \omega \) by termination proof with reduction order \( \succ \) on terms.
Example (double)

\[
\begin{align*}
\text{double}(0) & \succ 0 \\
\text{double}(s(x)) & \succ s(s(\text{double}(x)))
\end{align*}
\]

Show \( dc_R(n) < \omega \) by termination proof with reduction order \( \succ \) on terms. Get \( \succ \) via polynomial interpretation\(^{13}\) \([ \cdot ]\) over \( \mathbb{N} \):

\[ \ell \succ r \iff [\ell] \succ [r] \]

---

\(^{13}\) D. Lankford: *Canonical algebraic simplification in computational logic*, U Texas ’75
Derivational Complexity from Polynomial Interpretations (1/2)

Example (double)

\[
\begin{align*}
\text{double}(0) & \succ 0 \\
\text{double}(s(x)) & \succ s(s(\text{double}(x)))
\end{align*}
\]

Show \( dc_R(n) < \omega \) by termination proof with reduction order \( \succ \) on terms. Get \( \succ \) via polynomial interpretation\(^{13}\) \([ \cdot ]\) over \( \mathbb{N} \): \( \ell \succ r \iff [\ell] \succ [r] \)

Example: \([\text{double}](x) = 3 \cdot x\), \([s](x) = x + 1\), \([0] = 1\)

\(^{13}\) D. Lankford: Canonical algebraic simplification in computational logic, U Texas ’75
Example (double)

\[
\begin{align*}
\text{double}(0) & \succ 0 \\
\text{double}(s(x)) & \succ s(s(\text{double}(x)))
\end{align*}
\]

Show \( dc_\mathcal{R}(n) < \omega \) by termination proof with reduction order \( \succ \) on terms. Get \( \succ \) via polynomial interpretation\(^{13}\) \([\cdot]\) over \(\mathbb{N}\): \( l \succ r \iff [l] \succ [r] \)

Example: \([\text{double}](x) = 3 \cdot x\), \([s](x) = x + 1\), \([0] = 1\)

Extend to terms:

- \([x] = x\)
- \([f(t_1, \ldots, t_n)] = [f([t_1], \ldots, [t_n])]\)

\(^{13}\) D. Lankford: *Canonical algebraic simplification in computational logic*, U Texas ’75
Derivational Complexity from Polynomial Interpretations (1/2)

Example (double)

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>double(0)</td>
<td>≻</td>
<td>0</td>
</tr>
<tr>
<td>double(s(x))</td>
<td>≻</td>
<td>s(s(double(x)))</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>&gt;</td>
<td>1</td>
</tr>
<tr>
<td>3 \cdot x + 3</td>
<td>&gt;</td>
<td>3 \cdot x + 2</td>
</tr>
</tbody>
</table>

Show $\text{dc}_R(n) < \omega$ by termination proof with reduction order $\triangleright$ on terms.

Get $\triangleright$ via polynomial interpretation\textsuperscript{13} $[\cdot]$ over $\mathbb{N}$: $\ell \triangleright r \iff [\ell] \triangleright [r]$

Example: $[\text{double}](x) = 3 \cdot x$, $[s](x) = x + 1$, $[0] = 1$

Extend to terms:

- $[x] = x$
- $[f(t_1, \ldots, t_n)] = [f([t_1], \ldots, [t_n])]$

\textsuperscript{13} D. Lankford: *Canonical algebraic simplification in computational logic*, U Texas ’75
Derivational Complexity from Polynomial Interpretations (1/2)

Example (double)

\[
\begin{align*}
\text{double}(0) & \succ 0 & 3 & > 1 \\
\text{double}(s(x)) & \succ s(s(\text{double}(x))) & 3 \cdot x + 3 & > 3 \cdot x + 2
\end{align*}
\]

Show $dc_R(n) < \omega$ by termination proof with reduction order $\succ$ on terms. Get $\succ$ via polynomial interpretation\(^{13}\) $[\cdot]$ over $\mathbb{N}$: $\ell \succ r \iff [\ell] \succ [r]$.

Example: $[\text{double}](x) = 3 \cdot x$, $[s](x) = x + 1$, $[0] = 1$

Extend to terms:

- $[x] = x$
- $[f(t_1, \ldots, t_n)] = [f([t_1], \ldots, [t_n])]$

Automated search for $[\cdot]$ via SAT\(^{14}\) or SMT\(^{15}\) solving

\(^{13}\)D. Lankford: *Canonical algebraic simplification in computational logic*, U Texas ’75

\(^{14}\)C. Fuhs, J. Giesl, A. Middeldorp, P. Schneider-Kamp, R. Thiemann, H. Zankl: *SAT solving for termination analysis with polynomial interpretations*, SAT ’07

\(^{15}\)C. Borralleras, S. Lucas, A. Oliveras, E. Rodríguez-Carbonell, A. Rubio: *SAT modulo linear arithmetic for solving polynomial constraints*, JAR ’12
Example (double)

\[
\begin{align*}
\text{double}(0) & \succ 0 & 3 & > 1 \\
\text{double}(s(x)) & \succ s(s(\text{double}(x))) & 3 \cdot x + 3 & > 3 \cdot x + 2
\end{align*}
\]

Example: \([\text{double}](x) = 3 \cdot x, \quad [s](x) = x + 1, \quad [0] = 1\]

This proves more than just termination…
Derivational Complexity from Polynomial Interpretations (2/2)

Example (double)

\[\begin{align*}
\text{double}(0) & \succ 0 & 3 & > 1 \\
\text{double}(s(x)) & \succ s(s(\text{double}(x))) & 3 \cdot x + 3 & > 3 \cdot x + 2
\end{align*}\]

Example: \([\text{double}](x) = 3 \cdot x, \quad [s](x) = x + 1, \quad [0] = 1\]

This proves more than just termination...

Theorem (Upper bounds for \(dc_R(n)\) from polynomial interpretations\(^{16}\))

- Termination proof for TRS \(R\) with polynomial interpretation

\[\Rightarrow dc_R(n) \in 2^{O(n)}\]

\(^{16}\)D. Hofbauer, C. Lautemann: Termination proofs and the length of derivations, RTA ’89
Example (double)

\[
\begin{align*}
\text{double}(0) & \succ 0 & 3 & > 1 \\
\text{double}(s(x)) & \succ s(s(\text{double}(x))) & 3 \cdot x + 3 & > 3 \cdot x + 2
\end{align*}
\]

Example: \[\text{[double]}(x) = 3 \cdot x, \quad [s](x) = x + 1, \quad [0] = 1\]

This proves more than just termination...

Theorem (Upper bounds for \( dc_R(n) \) from polynomial interpretations\(^{16}\))

- **Termination proof for TRS \( R \) with polynomial interpretation**
  \[
  \Rightarrow dc_R(n) \in 2^{O(n)}
  \]
- **Termination proof for TRS \( R \) with linear polynomial interpretation**
  \[
  \Rightarrow dc_R(n) \in 2^{O(n)}
  \]

\(^{16}\) D. Hofbauer, C. Lautemann: *Termination proofs and the length of derivations*, RTA ’89
Termination proof for TRS $\mathcal{R}$ with ...

- matchbounds$^{17}$
- arctic matrix interpretations$^{18}$

$\Rightarrow \text{dc}_\mathcal{R}(n) \in \mathcal{O}(n)$

---

$^{17}$ A. Geser, D. Hofbauer, J. Waldmann: *Match-bounded string rewriting systems*, AAECC '04

$^{18}$ A. Koprowski, J. Waldmann: *Max/plus tree automata for termination of term rewriting*, Acta Cyb. '09
Termination proof for TRS $\mathcal{R}$ with...

- matchbounds $^{17}$
- arctic matrix interpretations $^{18}$
- triangular matrix interpretation $^{19}$
- matrix interpretation of spectral radius $^{20} \leq 1$

$\Rightarrow \ dc_\mathcal{R}(n) \in \mathcal{O}(n)$

$\Rightarrow \ dc_\mathcal{R}(n)$ is at most polynomial

---

$^{17}$ A. Geser, D. Hofbauer, J. Waldmann: *Match-bounded string rewriting systems*, AAECC ’04

$^{18}$ A. Koprowski, J. Waldmann: *Max/plus tree automata for termination of term rewriting*, Acta Cyb. ’09

$^{19}$ G. Moser, A. Schnabl, J. Waldmann: *Complexity analysis of term rewriting based on matrix and context dependent interpretations*, FSTTCS ’08

$^{20}$ F. Neurauter, H. Zankl, A. Middeldorp: *Revisiting matrix interpretations for polynomial derivational complexity of term rewriting*, LPAR (Yogyakarta) ’10
Termination proof for TRS $\mathcal{R}$ with...

- matchbounds\textsuperscript{17} \quad \Rightarrow \quad \text{dc}_{\mathcal{R}}(n) \in \mathcal{O}(n)
- arctic matrix interpretations\textsuperscript{18} \quad \Rightarrow \quad \text{dc}_{\mathcal{R}}(n) \in \mathcal{O}(n)
- triangular matrix interpretation\textsuperscript{19} \quad \Rightarrow \quad \text{dc}_{\mathcal{R}}(n) \text{ is at most polynomial}
- matrix interpretation of spectral radius\textsuperscript{20} \leq 1 \quad \Rightarrow \quad \text{dc}_{\mathcal{R}}(n) \text{ is at most polynomial}
- standard matrix interpretation\textsuperscript{21} \quad \Rightarrow \quad \text{dc}_{\mathcal{R}}(n) \text{ is at most exponential}

\textsuperscript{17} A. Geser, D. Hofbauer, J. Waldmann: \textit{Match-bounded string rewriting systems}, AAECC '04
\textsuperscript{18} A. Koprowski, J. Waldmann: \textit{Max/plus tree automata for termination of term rewriting}, Acta Cyb. '09
\textsuperscript{19} G. Moser, A. Schnabl, J. Waldmann: \textit{Complexity analysis of term rewriting based on matrix and context dependent interpretations}, FSTTCS '08
\textsuperscript{20} F. Neurauter, H. Zankl, A. Middeldorp: \textit{Revisiting matrix interpretations for polynomial derivational complexity of term rewriting}, LPAR (Yogyakarta) '10
\textsuperscript{21} J. Endrullis, J. Waldmann, and H. Zantema: \textit{Matrix interpretations for proving termination of term rewriting}, JAR '08
Termination proof for TRS $\mathcal{R}$ with . . .

- lexicographic path order\footnote{S. Kamin, J.-J. Lévy: \textit{Two generalizations of the recursive path ordering}, U Illinois ’80} $\Rightarrow d_{\mathcal{R}}(n)$ is at most multiple recursive\footnote{A. Weiermann: \textit{Termination proofs for term rewriting systems by lexicographic path orderings imply multiply recursive derivation lengths}, TCS ’95}
Termination proof for TRS $\mathcal{R}$ with . . .

- lexicographic path order$^{22}$ \quad $\Rightarrow$ $d_{\mathcal{R}}(n)$ is at most multiple recursive$^{23}$
- Dependency Pairs method$^{24}$ with dependency graphs and usable rules
  \quad $\Rightarrow$ $d_{\mathcal{R}}(n)$ is at most primitive recursive$^{25}$

---

$^{22}$ S. Kamin, J.-J. Lévy: *Two generalizations of the recursive path ordering*, U Illinois '80
$^{23}$ A. Weiermann: *Termination proofs for term rewriting systems by lexicographic path orderings imply multiply recursive derivation lengths*, TCS '95
$^{24}$ T. Arts, J. Giesl: *Termination of term rewriting using dependency pairs*, TCS '00
$^{25}$ G. Moser, A. Schnabl: *The derivational complexity induced by the dependency pair method*, LMCS '11
Termination proof for TRS $\mathcal{R}$ with . . .

- lexicographic path order\(^{22}\) $\Rightarrow d_{\mathcal{R}}(n)$ is at most multiple recursive\(^{23}\)
- Dependency Pairs method\(^{24}\) with dependency graphs and usable rules
  $\Rightarrow d_{\mathcal{R}}(n)$ is at most primitive recursive\(^{25}\)
- Dependency Pairs framework\(^{26,27}\) with dependency graphs, reduction
  pairs, subterm criterion $\Rightarrow d_{\mathcal{R}}(n)$ is at most multiple recursive\(^{28}\)

\(^{22}\) S. Kamin, J.-J. Lévy: *Two generalizations of the recursive path ordering*, U Illinois '80

\(^{23}\) A. Weiermann: *Termination proofs for term rewriting systems by lexicographic path
orderings imply multiply recursive derivation lengths*, TCS '95

\(^{24}\) T. Arts, J. Giesl: *Termination of term rewriting using dependency pairs*, TCS '00

\(^{25}\) G. Moser, A. Schnabl: *The derivational complexity induced by the dependency pair
method*, LMCS '11

\(^{26}\) J. Giesl, R. Thiemann, P. Schneider-Kamp, S. Falke: *Mechanizing and improving
dependency pairs*, JAR '06

\(^{27}\) N. Hirokawa and A. Middeldorp: *Tyrolean Termination Tool: Techniques and
features*, IC '07

\(^{28}\) G. Moser, A. Schnabl: *Termination proofs in the dependency pair framework may
induce multiple recursive derivational complexity*, RTA '11
Runtime Complexity

- So far: upper bounds for derivational complexity

\[ \text{Definition (Basic Term 29)} \]
\[ \text{For defined symbols } D \text{ and constructor symbols } C, \text{ the term } f(t_1, \ldots, t_n) \text{ is in the set } T_{\text{basic}} \text{ iff } f \in D \text{ and } t_1, \ldots, t_n \in T(C, V). \]

\[ \text{Definition (Runtime Complexity } r_{c 29} ) \]
\[ r_{c} R(n) = \sup \{ d_{h}(t, \rightarrow_{R}) \mid t \in T_{\text{basic}}, |t| \leq n \} \]

\[ r_{c} R(n) \] : like derivational complexity... but for basic terms only!

N. Hirokawa, G. Moser: Automated complexity analysis based on the dependency pair method, IJCAR '08
Runtime Complexity

- So far: upper bounds for derivational complexity
- But: derivational complexity counter-intuitive, often infeasible
So far: upper bounds for derivational complexity
But: derivational complexity counter-intuitive, often infeasible
Wanted: complexity of evaluation of double on data: double(s^n(0))
Runtime Complexity

- So far: upper bounds for derivational complexity
- But: derivational complexity counter-intuitive, often infeasible
- Wanted: complexity of evaluation of double on data: $\text{double}(s^n(0))$

**Definition (Basic Term)$^{29}$**

For defined symbols $\mathcal{D}$ and constructor symbols $\mathcal{C}$, the term

$$f(t_1, \ldots, t_n)$$

is in the set $\mathcal{T}_{\text{basic}}$ of basic terms iff $f \in \mathcal{D}$ and $t_1, \ldots, t_n \in \mathcal{T}(\mathcal{C}, \mathcal{V})$.

---

$^{29}$N. Hirokawa, G. Moser: *Automated complexity analysis based on the dependency pair method*, IJCAR '08
Runtime Complexity

- So far: upper bounds for derivational complexity
- But: derivational complexity counter-intuitive, often infeasible
- Wanted: complexity of evaluation of double on data: \( \text{double}(s^n(0)) \)

**Definition (Basic Term\(^{29}\))**

For defined symbols \( D \) and constructor symbols \( C \), the term \( f(t_1, \ldots, t_n) \) is in the set \( T_{\text{basic}} \) of basic terms iff \( f \in D \) and \( t_1, \ldots, t_n \in T(C, \mathcal{V}) \).

**Definition (Runtime Complexity \( rc^{29} \))**

For a TRS \( \mathcal{R} \), the runtime complexity is:

\[
    rc_\mathcal{R}(n) = \sup \{ \text{dh}(t, \rightarrow_\mathcal{R}) \mid t \in T_{\text{basic}}, |t| \leq n \}
\]

---

\(^{29}\) N. Hirokawa, G. Moser: *Automated complexity analysis based on the dependency pair method*, IJCAR ’08
Runtime Complexity

- So far: upper bounds for derivational complexity
- But: derivational complexity counter-intuitive, often infeasible
- Wanted: complexity of evaluation of double on data: \( \text{double}(s^n(0)) \)

**Definition (Basic Term\(^{29}\))**

For **defined symbols** \( D \) and **constructor symbols** \( C \), the term

\[
f(t_1, \ldots, t_n)
\]

is in the set \( T_{\text{basic}} \) of **basic terms** iff \( f \in D \) and \( t_1, \ldots, t_n \in T(C, V) \).

**Definition (Runtime Complexity \( rc^{29} \))**

For a TRS \( R \), the **runtime complexity** is:

\[
rc_R(n) = \sup \{ dh(t, \rightarrow_R) \mid t \in T_{\text{basic}}, |t| \leq n \}
\]

\( rc_R(n) \): like derivational complexity... but for basic terms only!

---

\(^{29}\) N. Hirokawa, G. Moser: *Automated complexity analysis based on the dependency pair method*, IJCAR '08
Polynomial interpretations can induce upper bounds to runtime complexity:

**Definition (Strongly linear polynomial, restricted interpretation)**

- Polynomial $p$ is **strongly linear** iff $p(x_1,\ldots,x_n) = x_1 + \cdots + x_n + a$ for some $a \in \mathbb{N}$.
- Polynomial interpretation $[\cdot]$ is **restricted** iff for all constructor symbols $f$, $[f](x_1,\ldots,x_n)$ is strongly linear.

Idea: $[t] \leq c \cdot |t|$ for fixed $c \in \mathbb{N}$.

---

30 G. Bonfante, A. Cichon, J. Marion, H. Touzet: *Algorithms with polynomial interpretation termination proof*, JFP ’01
Polynomial interpretations can induce upper bounds to runtime complexity:

**Definition (Strongly linear polynomial, restricted interpretation)**

- Polynomial $p$ is **strongly linear** iff
  
  $$p(x_1, \ldots, x_n) = x_1 + \cdots + x_n + a$$
  
  for some $a \in \mathbb{N}$.

- Polynomial interpretation $[\cdot]$ is **restricted** iff
  
  for all constructor symbols $f$, $[f](x_1, \ldots, x_n)$ is strongly linear.

**Idea**: $[t] \leq c \cdot |t|$ for fixed $c \in \mathbb{N}$.

**Theorem (Upper bounds for $rc_R(n)$ from restricted interpretations)**

Termination proof for TRS $R$ with **restricted** interpretation $[\cdot]$ of degree at most $d$ for $[f]$

$$\Rightarrow rc_R(n) \in \mathcal{O}(n^d)$$

---

---

30 G. Bonfante, A. Cichon, J. Marion, H. Touzet: *Algorithms with polynomial interpretation termination proof*, JFP ’01
Runtime Complexity from Polynomial Interpretations

Polynomial interpretations can induce upper bounds to runtime complexity:

**Definition (Strongly linear polynomial, restricted interpretation)**

- Polynomial $p$ is **strongly linear** iff
  
  $$p(x_1, \ldots, x_n) = x_1 + \cdots + x_n + a$$
  
  for some $a \in \mathbb{N}$.

- Polynomial interpretation $[\cdot]$ is **restricted** iff
  
  for all constructor symbols $f$, $[f](x_1, \ldots, x_n)$ is strongly linear.

**Idea:** $[t] \leq c \cdot |t|$ for fixed $c \in \mathbb{N}$.

**Theorem (Upper bounds for $rc_R(n)$ from restricted interpretations)**

Termination proof for TRS $R$ with **restricted** interpretation $[\cdot]$ of degree at most $d$ for $[f]$

$$\Rightarrow rc_R(n) \in \mathcal{O}(n^d)$$

**Example:** $[\text{double}](x) = 3 \cdot x$, $[\text{s}](x) = x + 1$, $[0] = 1$ is restricted, degree 1

$$\Rightarrow rc_R(n) \in \mathcal{O}(n)$$ for TRS $R$ for double

---

\[30\]
G. Bonfante, A. Cichon, J. Marion, H. Touzet: *Algorithms with polynomial interpretation termination proof*, JFP '01
Dependency Tuples for Innermost Runtime Complexity irc

Here: innermost rewriting (≈ call-by-value)

Example (reverse)

\[
\begin{align*}
\text{app}(\text{nil}, y) & \rightarrow y \\
\text{reverse}(\text{nil}) & \rightarrow \text{nil} \\
\text{app}(\text{add}(n, x), y) & \rightarrow \text{add}(n, \text{app}(x, y)) \\
\text{reverse}(\text{add}(n, x)) & \rightarrow \text{app}(\text{reverse}(x), \text{add}(n, \text{nil}))
\end{align*}
\]
Dependency Tuples for *Innermost* Runtime Complexity irc

Here: innermost rewriting (≈ call-by-value)

**Example (reverse)**

<table>
<thead>
<tr>
<th>Term</th>
<th>Reduction</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>app(nil, y)</code></td>
<td><code>y</code></td>
</tr>
<tr>
<td><code>reverse(nil)</code></td>
<td><code>nil</code></td>
</tr>
<tr>
<td><code>app(add(n, x), y)</code></td>
<td><code>add(n, app(x, y))</code></td>
</tr>
<tr>
<td><code>reverse(add(n, x))</code></td>
<td><code>app(reverse(x), add(n, nil))</code></td>
</tr>
</tbody>
</table>

For rule \( \ell \rightarrow r \), eval of \( \ell \) costs 1 + eval of all function calls in \( r \) together:

---

\(^{31}\) L. Noschinski, F. Emmes, J. Giesl: *Analyzing innermost runtime complexity of term rewriting by dependency pairs*, JAR '13
Here: innermost rewriting ($\approx$ call-by-value)

Example (reverse)

<table>
<thead>
<tr>
<th>Rule</th>
<th>Execution</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{app}(\text{nil}, y) \rightarrow y$</td>
<td>$\text{app}(\text{add}(n, x), y) \rightarrow \text{add}(n, \text{app}(x, y))$</td>
</tr>
<tr>
<td>$\text{reverse}(\text{nil}) \rightarrow \text{nil}$</td>
<td>$\text{reverse}(\text{add}(n, x)) \rightarrow \text{app}(\text{reverse}(x), \text{add}(n, \text{nil}))$</td>
</tr>
</tbody>
</table>

For rule $\ell \rightarrow r$, eval of $\ell$ costs $1 + \text{eval of all function calls in } r$ together:

Example (Dependency Tuples\textsuperscript{31} for reverse)

<table>
<thead>
<tr>
<th>Rule</th>
<th>Execution</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{app}^#(\text{nil}, y) \rightarrow \text{Com}_0$</td>
<td></td>
</tr>
<tr>
<td>$\text{app}^#(\text{add}(n, x), y) \rightarrow \text{Com}_1(\text{app}^#(x, y))$</td>
<td></td>
</tr>
<tr>
<td>$\text{reverse}^#(\text{nil}) \rightarrow \text{Com}_0$</td>
<td></td>
</tr>
<tr>
<td>$\text{reverse}^#(\text{add}(n, x)) \rightarrow \text{Com}_2(\text{app}^#(\text{reverse}(x), \text{add}(n, \text{nil})), \text{reverse}^#(x))$</td>
<td></td>
</tr>
</tbody>
</table>

- Function calls to count marked with $^\#$
- Compound symbols $\text{Com}_k$ group function calls together

\textsuperscript{31}L. Noschinski, F. Emmes, J. Giesl: Analyzing innermost runtime complexity of term rewriting by dependency pairs, JAR '13
Example (reverse, Dependency Tuples for reverse)

\[
\begin{align*}
\text{app}^{\#}(\text{nil}, y) & \rightarrow \text{Com}_0 \\
\text{app}^{\#}(\text{add}(n, x), y) & \rightarrow \text{Com}_1(\text{app}^{\#}(x, y)) \\
\text{reverse}^{\#}(\text{nil}) & \rightarrow \text{Com}_0 \\
\text{reverse}^{\#}(\text{add}(n, x)) & \rightarrow \text{Com}_2(\text{app}^{\#}(\text{reverse}(x), \text{add}(n, \text{nil})), \text{reverse}^{\#}(x)) \\
\text{app}(\text{nil}, y) & \rightarrow y \\
\text{app}(\text{add}(n, x), y) & \rightarrow \text{add}(n, \text{app}(x, y)) \\
\text{reverse}(\text{nil}) & \rightarrow \text{nil} \\
\text{reverse}(\text{add}(n, x)) & \rightarrow \text{app}(\text{reverse}(x), \text{add}(n, \text{nil}))
\end{align*}
\]
Example (reverse, Dependency Tuples for reverse)

\[
\begin{align*}
\text{app}^\#(\text{nil}, y) & \rightarrow \text{Com}_0 \\
\text{app}^\#(\text{add}(n, x), y) & \rightarrow \text{Com}_1(\text{app}^\#(x, y)) \\
\text{reverse}^\#(\text{nil}) & \rightarrow \text{Com}_0 \\
\text{reverse}^\#(\text{add}(n, x)) & \rightarrow \text{Com}_2(\text{app}^\#(\text{reverse}(x), \text{add}(n, \text{nil})), \text{reverse}^\#(x)) \\
\text{app}(\text{nil}, y) & \rightarrow y \\
\text{app}(\text{add}(n, x), y) & \rightarrow \text{add}(n, \text{app}(x, y)) \\
\text{reverse}(\text{nil}) & \rightarrow \text{nil} \\
\text{reverse}(\text{add}(n, x)) & \rightarrow \text{app}(\text{reverse}(x), \text{add}(n, \text{nil}))
\end{align*}
\]

Use interpretation \([ \cdot ]\) with \([\text{Com}_k](x_1, \ldots, x_k) = x_1 + \cdots + x_k\) and

\[
\begin{align*}
[\text{nil}] &= 0 \\
[\text{app}](x_1, x_2) &= x_1 + x_2 \\
[\text{app}^\#](x_1, x_2) &= x_1 + 1 \\
[\text{add}](x_1, x_2) &= x_2 + 1 \ (\leq \text{restricted interpret.}) \\
[\text{reverse}](x_1) &= x_1 \ (\text{bounds helper fct. result size}) \\
[\text{reverse}^\#](x_1) &= x_1^2 + x_1 + 1 \ (\text{complexity of fct.})
\end{align*}
\]

to show \([\ell] \geq [r]\) for all rules and \([\ell] \geq 1 + [r]\) for all Dependency Tuples

Maximum degree of \([ \cdot ]\) is 2 \(\Rightarrow \text{irc}_R(n) \in \mathcal{O}(n^2)\)
Dependency Tuples are an adaptation of Dependency Pairs (DPs) from termination analysis to complexity analysis, allow for **incremental** complexity proofs with several techniques.
Related Techniques

- Dependency Tuples are an adaptation of Dependency Pairs (DPs) from termination analysis to complexity analysis, allow for **incremental** complexity proofs with several techniques.

- Further adaptation of DPs (incomparable): Weak (Innermost) Dependency Pairs for (innermost) runtime complexity\textsuperscript{32}

---

\textsuperscript{32}N. Hirokawa, G. Moser: *Automated complexity analysis based on the dependency pair method*, IJCAR '08
Related Techniques

- Dependency Tuples are an adaptation of Dependency Pairs (DPs) from termination analysis to complexity analysis, allow for **incremental** complexity proofs with several techniques
- Further adaptation of DPs (incomparable): Weak (Innermost) Dependency Pairs for (innermost) runtime complexity\(^\text{32}\)
- Extensions by polynomial path orders\(^\text{33}\), usable replacement maps\(^\text{34}\), a combination framework for complexity analysis\(^\text{35}\), …

---

\(^{32}\) N. Hirokawa, G. Moser: *Automated complexity analysis based on the dependency pair method*, IJCAR ’08

\(^{33}\) M. Avanzini, G. Moser: *Dependency pairs and polynomial path orders*, RTA ’09

\(^{34}\) N. Hirokawa, G. Moser: *Automated complexity analysis based on context-sensitive rewriting*, RTA-TLCA ’14

\(^{35}\) M. Avanzini, G. Moser: *A combination framework for complexity*, IC ’16
How about Lower Bounds for Complexity?

- Input size
- Runtime
- Upper bound
- Worst case
- Lower bound
- Best case

Why lower bounds?
- Get tight bounds with upper bounds
- Can indicate implementation bugs
- Security: single query can trigger Denial of Service

Here: Two techniques for finding lower bounds inspired by proving non-termination

F. Frohn, J. Giesl, J. Hensel, C. Aschermann, and T. Ströder: Lower bounds for runtime complexity of term rewriting, JAR '17
How about Lower Bounds for Complexity?

Why lower bounds?
- get **tight bounds** with upper bounds
- can indicate implementation bugs
- security: single query can trigger Denial of Service
How about Lower Bounds for Complexity?

Why lower bounds?
- get **tight bounds** with upper bounds
- can indicate implementation bugs
- security: single query can trigger Denial of Service

Here: Two techniques for finding lower bounds\(^\text{36}\) inspired by proving non-termination

\(^{36}\) F. Frohn, J. Giesl, J. Hensel, C. Aschermann, and T. Ströder: Lower bounds for runtime complexity of term rewriting, JAR ’17
Finding Lower Bounds by Induction

(1) Induction technique, inspired by non-looping non-termination\textsuperscript{37}
Finding Lower Bounds by Induction

(1) Induction technique, inspired by non-looping non-termination\(^\text{37}\)

- Generate infinite family \(\mathcal{T}_{\text{witness}}\) of basic terms as witnesses in

\[
\forall n \in \mathbb{N}. \ \exists t_n \in \mathcal{T}_{\text{witness}}. \ |t_n| \leq q(n) \ \land \ \text{dh}(t_n, \rightarrow_{\mathcal{R}}) \geq p(n)
\]

to conclude \(rc_{\mathcal{R}}(n) \in \Omega(p'(n))\).

\(^\text{37}\) F. Emmes, T. Enger, J. Giesl: *Proving non-looping non-termination automatically*, IJCAR '12
Finding Lower Bounds by Induction

(1) Induction technique, inspired by **non-looping** non-termination\(^{37}\)

- Generate infinite family \( \mathcal{T}_{\text{witness}} \) of basic terms as witnesses in

\[
\forall n \in \mathbb{N}. \; \exists t_n \in \mathcal{T}_{\text{witness}}. \; |t_n| \leq q(n) \land \text{dh}(t_n, \rightarrow_R) \geq p(n)
\]

To conclude \( rc_R(n) \in \Omega(p'(n)) \).

- Constructor terms for arguments can be built recursively after type inference: \( 0, s(0), s(s(0)), \ldots \) (here \( q(n) = n + 1 \), often linear)

---

\(^{37}\) F. Emmes, T. Enger, J. Giesl: *Proving non-looping non-termination automatically*, IJCAR ’12
Finding Lower Bounds by Induction

(1) Induction technique, inspired by non-looping non-termination\(^{37}\)

- Generate infinite family \(\mathcal{T}_{\text{witness}}\) of basic terms as witnesses in

\[
\forall n \in \mathbb{N}. \ \exists t_n \in \mathcal{T}_{\text{witness}}. \ |t_n| \leq q(n) \ \land \ \text{dh}(t_n, \rightarrow_R) \geq p(n)
\]

to conclude \(rc_R(n) \in \Omega(p'(n))\).

- Constructor terms for arguments can be built recursively after type inference: \(0, s(0), s(s(0)), \ldots\) (here \(q(n) = n + 1\), often linear)

- Evaluate \(t_n\) by narrowing, get rewrite sequences with recursive calls

---

\(^{37}\) F. Emmes, T. Enger, J. Giesl: Proving non-looping non-termination automatically, IJCAR ’12
Finding Lower Bounds by Induction

(1) Induction technique, inspired by **non-looping** non-termination\(^{37}\)

- Generate infinite family \(T_{\text{witness}}\) of basic terms as witnesses in

\[
\forall n \in \mathbb{N}. \ \exists t_n \in T_{\text{witness}}. \ |t_n| \leq q(n) \land \text{dh}(t_n, \rightarrow_R) \geq p(n)
\]

to conclude \(\text{rc}_R(n) \in \Omega(p'(n))\).

- Constructor terms for arguments can be built recursively after type inference: \(0, s(0), s(s(0)), \ldots\) (here \(q(n) = n + 1\), often linear)

- Evaluate \(t_n\) by narrowing, get rewrite sequences with recursive calls

- Speculate polynomial \(p(n)\) based on values for \(n = 0, 1, \ldots, k\)

---

\(^{37}\) F. Emmes, T. Enger, J. Giesl: *Proving non-looping non-termination automatically*, IJCAR '12
Finding Lower Bounds by Induction

(1) Induction technique, inspired by non-looping non-termination\(^{37}\)

- Generate infinite family \(T_{\text{witness}}\) of basic terms as witnesses in

\[
\forall n \in \mathbb{N}. \ \exists t_n \in T_{\text{witness}}. \ |t_n| \leq q(n) \land \text{dh}(t_n, \rightarrow_R) \geq p(n)
\]

(to conclude \(\text{rc}_R(n) \in \Omega(p'(n))\)).

- Constructor terms for arguments can be built recursively after type inference: \(0, s(0), s(s(0)), \ldots\) (here \(q(n) = n + 1\), often linear)

- Evaluate \(t_n\) by narrowing, get rewrite sequences with recursive calls

- Speculate polynomial \(p(n)\) based on values for \(n = 0, 1, \ldots, k\)

- Prove rewrite lemma \(t_n \xrightarrow{\geq p(n)} t'_n\) inductively

---

\(^{37}\) F. Emmes, T. Enger, J. Giesl: Proving non-looping non-termination automatically, IJCAR '12
Finding Lower Bounds by Induction

(1) Induction technique, inspired by **non-looping** non-termination\(^{37}\)

- Generate infinite family \(T_{\text{witness}}\) of basic terms as witnesses in

\[
\forall n \in \mathbb{N}. \quad \exists t_n \in T_{\text{witness}}. \quad |t_n| \leq q(n) \land \text{dh}(t_n, \rightarrow_R) \geq p(n)
\]

to conclude \(\text{rc}_R(n) \in \Omega(p'(n))\).

- Constructor terms for arguments can be built recursively after type inference: \(0, s(0), s(s(0)), \ldots\) (here \(q(n) = n + 1\), often linear)

- Evaluate \(t_n\) by narrowing, get rewrite sequences with recursive calls

- Speculate polynomial \(p(n)\) based on values for \(n = 0, 1, \ldots, k\)

- Prove rewrite lemma \(t_n \rightarrow_R^{\geq p(n)} t'_n\) inductively

- Get lower bound for \(\text{rc}_R(n)\) from \(p(n)\) in rewrite lemma and \(q(n)\)

\(^{37}\) F. Emmes, T. Enger, J. Giesl: *Proving non-looping non-termination automatically*, IJCAR ’12
Example (quicksort)

<table>
<thead>
<tr>
<th>Function</th>
<th>Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>(qs(\text{nil}))</td>
<td>(\rightarrow \text{nil})</td>
</tr>
<tr>
<td>(qs(\text{cons}(x, xs)))</td>
<td>(\rightarrow qs(\text{low}(x, xs)) ++ \text{cons}(x, qs(\text{low}(x, xs))))</td>
</tr>
<tr>
<td>(\text{low}(x, \text{nil}))</td>
<td>(\rightarrow \text{nil})</td>
</tr>
<tr>
<td>(\text{low}(x, \text{cons}(y, ys)))</td>
<td>(\rightarrow \text{if}(x \leq y, x, \text{cons}(y, ys)))</td>
</tr>
<tr>
<td>(\text{if}(\text{tt}, x, \text{cons}(y, ys)))</td>
<td>(\rightarrow \text{low}(x, ys))</td>
</tr>
<tr>
<td>(\text{if}(\text{ff}, x, \text{cons}(y, ys)))</td>
<td>(\rightarrow \text{cons}(y, \text{low}(x, ys)))</td>
</tr>
</tbody>
</table>

...
Finding Lower Bounds by Induction: Example

### Example (quicksort)

<table>
<thead>
<tr>
<th>Function</th>
<th>Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>(qs(\text{nil}))</td>
<td>(\rightarrow \text{nil})</td>
</tr>
<tr>
<td>(qs(\text{cons}(x, xs)))</td>
<td>(\rightarrow qs(\text{low}(x, xs)) +\ cons(x, qs(\text{low}(x, xs))))</td>
</tr>
<tr>
<td>(\text{low}(x, \text{nil}))</td>
<td>(\rightarrow \text{nil})</td>
</tr>
<tr>
<td>(\text{low}(x, \text{cons}(y, ys)))</td>
<td>(\rightarrow \text{if}(x \leq y, x, \text{cons}(y, ys)))</td>
</tr>
<tr>
<td>(\text{if}(\text{tt}, x, \text{cons}(y, ys)))</td>
<td>(\rightarrow \text{low}(x, ys))</td>
</tr>
<tr>
<td>(\text{if}(\text{ff}, x, \text{cons}(y, ys)))</td>
<td>(\rightarrow \text{cons}(y, \text{low}(x, ys)))</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

Speculate and prove rewrite lemma:

\[
qs(\text{cons}(\text{zero}, \ldots, \text{cons}(\text{zero}, \text{nil}))) \rightarrow 3n^2 + 2n + 1 \quad \text{cons}(\text{zero}, \ldots, \text{cons}(\text{zero}, \text{nil}))
\]
### Example (quicksort)

<table>
<thead>
<tr>
<th>Function</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>(qs(\text{nil}))</td>
<td>(\text{nil})</td>
</tr>
<tr>
<td>(qs(\text{cons}(x, xs)))</td>
<td>(qs(\text{low}(x, xs)) + + \text{cons}(x, qs(\text{low}(x, xs))))</td>
</tr>
<tr>
<td>(\text{low}(x, \text{nil}))</td>
<td>(\text{nil})</td>
</tr>
<tr>
<td>(\text{low}(x, \text{cons}(y, ys)))</td>
<td>(\text{if}(x \leq y, x, \text{cons}(y, ys)))</td>
</tr>
<tr>
<td>(\text{if}(\text{tt}, x, \text{cons}(y, ys)))</td>
<td>(\text{low}(x, ys))</td>
</tr>
<tr>
<td>(\text{if}(\text{ff}, x, \text{cons}(y, ys)))</td>
<td>(\text{cons}(y, \text{low}(x, ys)))</td>
</tr>
</tbody>
</table>

...  

### Speculate and prove rewrite lemma:

\[
qs(\text{cons}(\text{zero}, \ldots, \text{cons}(\text{zero}, \text{nil}))) \rightarrow^{3n^2 + 2n + 1} \text{cons}(\text{zero}, \ldots, \text{cons}(\text{zero}, \text{nil})) \\
qs(\text{cons}^n(\text{zero}, \text{nil})) \rightarrow^{3n^2 + 2n + 1} \text{cons}(\text{zero}, \ldots, \text{cons}(\text{zero}, \text{nil}))
\]
Example (quicksort)

\[
\begin{align*}
\text{qs}(\text{nil}) & \rightarrow \text{nil} \\
\text{qs}(\text{cons}(x, \text{x}s)) & \rightarrow \text{qs}(\text{low}(x, \text{x}s)) ++ \text{cons}(x, \text{qs}(\text{low}(x, \text{x}s))) \\
\text{low}(x, \text{nil}) & \rightarrow \text{nil} \\
\text{low}(x, \text{cons}(y, \text{y}s)) & \rightarrow \text{if}(x \leq y, x, \text{cons}(y, \text{y}s)) \\
\text{if}(\text{tt}, x, \text{cons}(y, \text{y}s)) & \rightarrow \text{low}(x, \text{y}s) \\
\text{if}(\text{ff}, x, \text{cons}(y, \text{y}s)) & \rightarrow \text{cons}(y, \text{low}(x, \text{y}s)) \\
\end{align*}
\]

Speculate and prove rewrite lemma:

\[
\begin{align*}
\text{qs}(\text{cons}(\text{zero}, \ldots, \text{cons}(\text{zero}, \text{nil})))) & \rightarrow 3n^2 + 2n + 1 \quad \text{cons}(\text{zero}, \ldots, \text{cons}(\text{zero}, \text{nil}))) \\
\text{qs}(\text{cons}^n(\text{zero}, \text{nil})) & \rightarrow 3n^2 + 2n + 1 \quad \text{cons}(\text{zero}, \ldots, \text{cons}(\text{zero}, \text{nil}))
\end{align*}
\]

From \(|\text{qs}(\text{cons}^n(\text{zero}, \text{nil}))| = 2n + 2\) we get

\[
rc_R(2n + 2) \geq 3n^2 + 2n + 1
\]
### Example (quicksort)

<table>
<thead>
<tr>
<th>Function Call</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>(qs(\text{nil}))</td>
<td>(\text{nil})</td>
</tr>
<tr>
<td>(qs(\text{cons}(x, xs)))</td>
<td>(qs(\text{low}(x, xs)) \rightleftharpoons \text{cons}(x, qs(\text{low}(x, xs))))</td>
</tr>
<tr>
<td>(\text{low}(x, \text{nil}))</td>
<td>(\text{nil})</td>
</tr>
<tr>
<td>(\text{low}(x, \text{cons}(y, ys)))</td>
<td>(\text{if}(x \leq y, x, \text{cons}(y, ys)))</td>
</tr>
<tr>
<td>(\text{if}(\text{tt}, x, \text{cons}(y, ys)))</td>
<td>(\text{low}(x, ys))</td>
</tr>
<tr>
<td>(\text{if}(\text{ff}, x, \text{cons}(y, ys)))</td>
<td>(\text{cons}(y, \text{low}(x, ys)))</td>
</tr>
</tbody>
</table>

...  

### Speculate and prove rewrite lemma:

\[
qs(\text{cons}(\text{zero}, \ldots, \text{cons}(\text{zero}, \text{nil}))) \rightarrow 3n^2 + 2n + 1 \quad \text{cons}(\text{zero}, \ldots, \text{cons}(\text{zero}, \text{nil}))
\]

\[
qs(\text{cons}^n(\text{zero}, \text{nil})) \rightarrow 3n^2 + 2n + 1 \quad \text{cons}(\text{zero}, \ldots, \text{cons}(\text{zero}, \text{nil}))
\]

From \(|qs(\text{cons}^n(\text{zero}, \text{nil}))| = 2n + 2\) we get

\[
rc_R(2n + 2) \geq 3n^2 + 2n + 1 \quad \text{and} \quad rc_R(n) \in \Omega(n^2).
\]
(2) Decreasing loops, inspired by **looping** non-termination with

\[ s \rightarrow_{\mathcal{R}}^+ C[s\sigma] \rightarrow_{\mathcal{R}}^+ C[C\sigma[s\sigma^2]] \rightarrow_{\mathcal{R}}^+ \cdots \]

**Example:** \( f(y) \rightarrow f(s(y)) \) has loop \( f(y) \rightarrow_{\mathcal{R}}^+ f(s(y)) \) with \( \sigma(y) = 0 \).
(2) Decreasing loops, inspired by **looping** non-termination with

\[ s \rightarrow^+ \mathcal{R} C[s\sigma] \rightarrow^+ \mathcal{R} C[C\sigma[s\sigma^2]] \rightarrow^+ \mathcal{R} \cdots \]

**Example:** \( f(y) \rightarrow f(s(y)) \) has loop \( f(y) \rightarrow^+ \mathcal{R} f(s(y)) \) with \( \sigma(y) = 0 \).

Intuition for **linear** lower bounds:

some fixed context \( D \) is **removed** in an argument of recursive call, other arguments may grow, sequence can be repeated (loop)
(2) Decreasing loops, inspired by \textbf{looping} non-termination with
\[
s \rightarrow^+ \mathcal{R} C[s\sigma] \rightarrow^+ \mathcal{R} C[C\sigma[s\sigma^2]] \rightarrow^+ \mathcal{R} \cdots
\]

\textbf{Example:} \( f(y) \rightarrow f(s(y)) \) has loop \( f(y) \rightarrow^+ \mathcal{R} f(s(y)) \) with \( \sigma(y) = 0 \).

Intuition for \textbf{linear} lower bounds:
some fixed context \( D \) is \textbf{removed} in an argument of recursive call, other arguments may grow, sequence can be repeated (loop)

\textbf{Example:} \( \text{plus}(s(x), y) \rightarrow \text{plus}(x, s(y)) \) has \textbf{decreasing} loop
\[
\text{plus}(s(x), y) \rightarrow^+ \mathcal{R} \text{plus}(x, s(y)) \quad \text{with} \quad D[x] = s(x)
\]
(2) Decreasing loops, inspired by \textbf{looping} non-termination with

\[ s \rightarrow_{\mathcal{R}}^{+} C[s\sigma] \rightarrow_{\mathcal{R}}^{+} C[C\sigma[s\sigma^{2}]] \rightarrow_{\mathcal{R}}^{+} \cdots \]

\textbf{Example:} \( f(y) \rightarrow f(s(y)) \) has loop \( f(y) \rightarrow_{\mathcal{R}}^{+} f(s(y)) \) with \( \sigma(y) = 0 \).

Intuition for \textbf{linear} lower bounds:

some fixed context \( D \) is \textbf{removed} in an argument of recursive call, other arguments may grow, sequence can be repeated (loop)

\textbf{Example:} \( \text{plus}(s(x), y) \rightarrow \text{plus}(x, s(y)) \) has \textbf{decreasing} loop

\[ \text{plus}(s(x), y) \rightarrow_{\mathcal{R}}^{+} \text{plus}(x, s(y)) \] with \( D[x] = s(x) \)

for \textit{base term} \( s = \text{plus}(x, y) \), \textit{pumping substitution} \( \theta = [x \mapsto s(x)] \), and \textit{result substitution} \( \sigma = [y \mapsto s(y)] \):

\[ s\theta \rightarrow_{\mathcal{R}}^{+} C[s\sigma] \]

Implies \( rc(n) \in \Omega(n)! \)
Exponential lower bounds: several “compatible” parallel recursive calls:

- **Example:** $\text{fib}(\text{s}(\text{s}(n)))) \rightarrow \text{plus}(\text{fib}(\text{s}(n)), \text{fib}(n))$ has 2 decreasing loops:

  $$\text{fib}(\text{s}(\text{s}(n)))) \rightarrow^+ \mathcal{R} \left[ \text{fib}(\text{s}(n)) \right] \quad \text{and} \quad \text{fib}(\text{s}(\text{s}(n)))) \rightarrow^+ \mathcal{R} \left[ \text{fib}(n) \right]$$

  Implies $rc(n) \in \Omega(2^n)$!
Exponential lower bounds: several “compatible” parallel recursive calls:

- **Example**: $\text{fib}(s(s(n))) \rightarrow \text{plus}(\text{fib}(s(n)), \text{fib}(n))$ has 2 decreasing loops:
  
  
  $\text{fib}(s(s(n))) \rightarrow^{+}_R C[\text{fib}(s(n))]$ and $\text{fib}(s(s(n))) \rightarrow^{+}_R C[\text{fib}(n)]$

  Implies $rc(n) \in \Omega(2^n)!$

- **(Non-)Example**: $\text{tr}(\text{node}(x, y)) \rightarrow \text{node}(\text{tr}(x), \text{tr}(y))$

  Has linear complexity. But:

  $\text{tr}(\text{node}(x, y)) \rightarrow^{+}_R C[\text{tr}(x)]$ and $\text{tr}(\text{node}(x, y)) \rightarrow^{+}_R C[\text{tr}(y)]$

  are not compatible (their pumping substitutions do not commute).
Finding Exponential Lower Bounds by Decreasing Loops

**Exponential** lower bounds: several “compatible” parallel recursive calls:

- **Example:** \( \text{fib}(s(s(n))) \rightarrow \text{plus}(\text{fib}(s(n)), \text{fib}(n)) \) has 2 decreasing loops:

  \[
  \text{fib}(s(s(n))) \rightarrow \underbrace{+}_{\mathcal{R}} C[\text{fib}(s(n))] \quad \text{and} \quad \text{fib}(s(s(n))) \rightarrow \underbrace{+}_{\mathcal{R}} C[\text{fib}(n)]
  \]

  Implies \( rc(n) \in \Omega(2^n) \)!

- **(Non-)Example:** \( \text{tr}(\text{node}(x, y)) \rightarrow \text{node}(\text{tr}(x), \text{tr}(y)) \)

  Has linear complexity. But:

  \[
  \text{tr}(\text{node}(x, y)) \rightarrow \underbrace{+}_{\mathcal{R}} C[\text{tr}(x)] \quad \text{and} \quad \text{tr}(\text{node}(x, y)) \rightarrow \underbrace{+}_{\mathcal{R}} C[\text{tr}(y)]
  \]

  are not compatible (their pumping substitutions do not commute).

Automation for decreasing loops: **narrowing**.
Lower Bounds: Induction Technique vs Decreasing Loops

Benefits of Induction Technique:

- Can find non-linear polynomial lower bounds
- Also works on non-left-linear TRSs
Lower Bounds: Induction Technique vs Decreasing Loops

Benefits of Induction Technique:
- Can find **non-linear** polynomial lower bounds
- Also works on non-left-linear TRSs

Benefits of Decreasing Loops:
- Does not rely as much on heuristics
- Computationally more lightweight
Benefits of Induction Technique:

- Can find **non-linear** polynomial lower bounds
- Also works on non-left-linear TRSs

Benefits of Decreasing Loops:

- Does not rely as much on heuristics
- Computationally more lightweight

$\Rightarrow$ First try decreasing loops, then induction technique
Lower Bounds: Induction Technique vs Decreasing Loops

Benefits of Induction Technique:
- Can find non-linear polynomial lower bounds
- Also works on non-left-linear TRSs

Benefits of Decreasing Loops:
- Does not rely as much on heuristics
- Computationally more lightweight

⇒ First try decreasing loops, then induction technique

Both techniques can be adapted to innermost runtime complexity!
A Landscape of Complexity Properties and Transformations
A Landscape of Complexity Properties and Transformations

idc, irc: like dc, rc, but for *innermost* rewriting

TRS
A Landscape of Complexity Properties and Transformations

idc, irc: like dc, rc, but for innermost rewriting

TRS

\footnote{F. Frohn, J. Giesl: Analyzing runtime complexity via innermost runtime complexity, LPAR '17}
A Landscape of Complexity Properties and Transformations

idc, irc: like dc, rc, but for *innermost* rewriting

---

F. Frohn, J. Giesl: *Analyzing runtime complexity via innermost runtime complexity*, LPAR '17

C. Fuhs: *Transforming Derivational Complexity of Term Rewriting to Runtime Complexity*, FroCoS '19
The big picture:

- **Have:** Tool for automated analysis of runtime complexity $r_{C_R}$
The big picture:

- **Have:** Tool for automated analysis of runtime complexity $rc_\mathcal{R}$
- **Want:** Tool for automated analysis of derivational complexity $dc_\mathcal{R}$
The big picture:

- **Have:** Tool for automated analysis of runtime complexity $rc_R$
- **Want:** Tool for automated analysis of derivational complexity $dc_R$
- **Idea:**
  
  “$rc_R$ analysis tool + transformation on TRS $R = dc_R$ analysis tool”
The big picture:

- **Have:** Tool for automated analysis of runtime complexity $rc_\mathcal{R}$
- **Want:** Tool for automated analysis of derivational complexity $dc_\mathcal{R}$
- **Idea:**
  
  \[ rc_\mathcal{R} \text{ analysis tool } + \text{ transformation on TRS } \mathcal{R} = dc_\mathcal{R} \text{ analysis tool} \]

- **Benefits:**
  - Get analysis of derivational complexity “for free”
  - Progress in runtime complexity analysis automatically improves derivational complexity analysis
From \( dc \) to \( rc \): Results

- program transformation such that runtime complexity of transformed TRS is **identical** to derivational complexity of original TRS
From dc to rc: Results

- program transformation such that runtime complexity of transformed TRS is *identical* to derivational complexity of original TRS
- transformation correct also from idc to irc
program transformation such that runtime complexity of transformed TRS is *identical* to derivational complexity of original TRS

transformation correct also from *idc* to *irc*

implemented in program analysis tool AProVE
From dc to rc: Results

- program transformation such that runtime complexity of transformed TRS is identical to derivational complexity of original TRS
- transformation correct also from idc to irc
- implemented in program analysis tool AProVE
- evaluated successfully on TPDB\textsuperscript{40} relative to state of the art TcT

\textsuperscript{40}Termination Problem DataBase, standard benchmark source for annual Termination Competition (termCOMP) with 1000s of problems, http://termination-portal.org/wiki/TPDB
From dc to rc: Transformation

Issue:

- Runtime complexity assumes basic terms as start terms
- We want to analyse complexity for arbitrary terms
From dc to rc: Transformation

Issue:
- Runtime complexity assumes basic terms as start terms
- We want to analyse complexity for arbitrary terms

Idea:
- Introduce constructor symbol $c_f$ for defined symbol $f$
From dc to rc: Transformation

Issue:
- Runtime complexity assumes **basic** terms as start terms
- We want to analyse complexity for **arbitrary** terms

Idea:
- Introduce **constructor symbol** $c_f$ for **defined symbol** $f$
- Add **generator rewrite rules** $G$ to reconstruct arbitrary term with $f$
  from basic term with $c_f$
From \texttt{dc} to \texttt{rc}: Transformation

**Issue:**
- Runtime complexity assumes \texttt{basic} terms as start terms
- We want to analyse complexity for \texttt{arbitrary} terms

**Idea:**
- Introduce \texttt{constructor symbol} $c_f$ for \texttt{defined symbol} $f$
- Add \texttt{generator rewrite rules} $G$ to reconstruct arbitrary term with $f$ from basic term with $c_f$

Represent
\[
t = \text{double}(\text{double}(\text{double}(s(0))))
\]
From dc to rc: Transformation

**Issue:**
- Runtime complexity assumes **basic** terms as start terms
- We want to analyse complexity for **arbitrary** terms

**Idea:**
- Introduce **constructor symbol** $c_f$ for **defined symbol** $f$
- Add **generator rewrite rules** $G$ to reconstruct arbitrary term with $f$ from basic term with $c_f$

Represent

$$t = \text{double(\text{double(\text{double(s(0))))})}$$

by **basic variant**

$$bv(t) = \text{enc}_{\text{double}}(c_{\text{double}}(c_{\text{double}}(c_{\text{double}}(s(0)))))$$
From dc to rc: Transformation

**Issue:**
- Runtime complexity assumes *basic* terms as start terms
- We want to analyse complexity for *arbitrary* terms

**Idea:**
- Introduce *constructor symbol* $c_f$ for *defined symbol* $f$
- Add *generator rewrite rules* $G$ to reconstruct arbitrary term with $f$ from basic term with $c_f$

Represent
$t = \text{double}(\text{double}(\text{double}(s(0))))$

by *basic variant*

$\text{bv}(t) = \text{enc}\text{double}(c_{\text{double}}(c_{\text{double}}(s(0))))$
From dc to rc: Transformation

Issue:
- Runtime complexity assumes basic terms as start terms
- We want to analyse complexity for arbitrary terms

Idea:
- Introduce constructor symbol $c_f$ for defined symbol $f$
- Add generator rewrite rules $G$ to reconstruct arbitrary term with $f$ from basic term with $c_f$

Represent

$t = \text{double}(\text{double}(\text{double} (\text{s}(0))))$

by basic variant

$\text{bv}(t) = \text{enc}_{\text{double}}(c_{\text{double}}(c_{\text{double}}(\text{s}(0))))$

Then:
- $\text{bv}(t)$ is basic term, size $|t|$
From $dc$ to $rc$: Transformation

**Issue:**
- Runtime complexity assumes basic terms as start terms
- We want to analyse complexity for arbitrary terms

**Idea:**
- Introduce constructor symbol $c_f$ for defined symbol $f$
- Add generator rewrite rules $G$ to reconstruct arbitrary term with $f$ from basic term with $c_f$

Represent

$$ t = \text{double}(\text{double}(\text{double}(s(0)))) $$

by basic variant

$$ \text{bv}(t) = \text{enc}_{\text{double}}(c_{\text{double}}(c_{\text{double}}(s(0)))) $$

Then:
- $\text{bv}(t)$ is basic term, size $|t|$
- $\text{bv}(t) \rightarrow^* t$

**Example (Generator rules $G$)**

<table>
<thead>
<tr>
<th>Generator Rule</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{enc}_{\text{double}}(x) \rightarrow \text{double}(\text{argenc}(x))$</td>
<td></td>
</tr>
<tr>
<td>$\text{enc}_0 \rightarrow 0$</td>
<td></td>
</tr>
<tr>
<td>$\text{enc}_s(x) \rightarrow s(\text{argenc}(x))$</td>
<td></td>
</tr>
<tr>
<td>$\text{argenc}(c_{\text{double}}(x)) \rightarrow \text{double}(\text{argenc}(x))$</td>
<td></td>
</tr>
<tr>
<td>$\text{argenc}(0) \rightarrow 0$</td>
<td></td>
</tr>
<tr>
<td>$\text{argenc}(s(x)) \rightarrow s(\text{argenc}(x))$</td>
<td></td>
</tr>
</tbody>
</table>
General Case: Relative Rewriting

Issue:

- $\rightarrow_{R \cup G}$ has extra rewrite steps not present in $\rightarrow_R$
- may change complexity
General Case: Relative Rewriting

Issue:
- $\rightarrow_{R \cup G}$ has extra rewrite steps not present in $\rightarrow_R$
- may change complexity

Solution:
- add $G$ as relative rewrite rules:
  - $\rightarrow_G$ steps are not counted for complexity analysis!
- transform $R$ to $R/G$ ($\rightarrow_R$ steps are counted, $\rightarrow_G$ steps are not)
General Case: Relative Rewriting

Issue:
- $\rightarrow_{R \cup G}$ has extra rewrite steps not present in $\rightarrow_R$
- may change complexity

Solution:
- add $G$ as relative rewrite rules:
  - $\rightarrow_G$ steps are not counted for complexity analysis!
- transform $R$ to $R/G$ ($\rightarrow_R$ steps are counted, $\rightarrow_G$ steps are not)
- more generally: transform $R/S$ to $R/(S \cup G)$
  (input may contain relative rules $S$, too)
Theorem (Derivational Complexity via Runtime Complexity)

Let $\mathcal{R}/\mathcal{S}$ be a relative TRS, let $\mathcal{G}$ be the generator rules for $\mathcal{R}/\mathcal{S}$. Then

1. $\text{dc}_{\mathcal{R}/\mathcal{S}}(n) = \text{rc}_{\mathcal{R}/(\mathcal{S} \cup \mathcal{G})}(n)$ (arbitrary rewrite strategies)
2. $\text{idc}_{\mathcal{R}/\mathcal{S}}(n) = \text{irc}_{\mathcal{R}/(\mathcal{S} \cup \mathcal{G})}(n)$ (innermost rewriting)

Note: equalities hold also non-asymptotically!
From (i)dc to (i)rc: Experiments

Experiments on TPDB, compare with state of the art in TcT:

- upper bounds idc: both AProVE and TcT with transformation are stronger than standard TcT

- upper bounds dc: TcT stronger than AProVE and TcT with transformation, but AProVE still solves some new examples

- lower bounds idc and dc: heuristics do not seem to benefit much
Experiments on TPDB, compare with state of the art in TcT:

- upper bounds \( idc \): both AProVE and TcT with transformation are stronger than standard TcT

- upper bounds \( dc \): TcT stronger than AProVE and TcT with transformation, but AProVE still solves some new examples

- lower bounds \( idc \) and \( dc \): heuristics do not seem to benefit much

⇒ Transformation-based approach should be part of the portfolio of analysis tools for derivational complexity
Possible applications

- compiler simplifications
- SMT solver preprocessing

Start terms may have nested defined symbols, so $dc_R$ is appropriate
Derivational Complexity: Future Work

- **Possible applications**
  - compiler simplifications
  - SMT solver preprocessing

Start terms may have nested defined symbols, so $dc_R$ is appropriate

- **Go between** derivational and runtime complexity
  - So far: encode *full* term universe $\mathcal{T}$ via basic terms $\mathcal{T}_{\text{basic}}$
  - Generalise: write relative rules to generate **arbitrary** set $\mathcal{U}$ of terms “between” basic and all terms ($\mathcal{T}_{\text{basic}} \subseteq \mathcal{U} \subseteq \mathcal{T}$).
Derivational Complexity: Future Work

Possible applications
- compiler simplifications
- SMT solver preprocessing

Start terms may have nested defined symbols, so $dc_R$ is appropriate

Go between derivational and runtime complexity
- So far: encode full term universe $\mathcal{T}$ via basic terms $\mathcal{T}_{\text{basic}}$
- Generalise: write relative rules to generate arbitrary set $\mathcal{U}$ of terms “between” basic and all terms ($\mathcal{T}_{\text{basic}} \subseteq \mathcal{U} \subseteq \mathcal{T}$).

Want to adapt techniques from runtime complexity analysis to derivational complexity! How?
- (Useful) adaptation of Dependency Pairs?
- Abstractions to numbers?
- ...
A Landscape of Complexity Properties and Transformations

idc, irc: like dc, rc, but for *innermost* rewriting

TRS

dc \rightarrow rc

\downarrow

\rightarrow

irc

FroCoS'19

LPAR'17

Rec. ITS irc

ITS irc
idc, irc: like dc, rc, but for innermost rewriting

M. Naaf, F. Frohn, M. Brockschmidt, C. Fuhs, J. Giesl: Complexity analysis for term rewriting by integer transition systems, FroCoS '17
A Landscape of Complexity Properties and Transformations

idc, irc: like dc, rc, but for *innermost* rewriting

TRS

FroCoS'19

dc → rc

FroCoS'19

idc → irc

LPAR'17

Rec. ITS irc

FroCoS'17

ITS irc

FroCoS'17

41 M. Naaf, F. Frohn, M. Brockschmidt, C. Fuhs, J. Giesl: *Complexity analysis for term rewriting by integer transition systems*, FroCoS '17
Bottom-Up Complexity Analysis for TRSs

Recently significant progress in complexity analysis tools for Integer Transition Systems (ITSs):

- CoFloCo\textsuperscript{42}
- KoAT\textsuperscript{43}
- PUBS\textsuperscript{44}

\textsuperscript{42} A. Flores-Montoya, R. Hähnle: \textit{Resource analysis of complex programs with cost equations}, APLAS '14, https://github.com/aeflores/CoFloCo

\textsuperscript{43} M. Brockschmidt, F. Emmes, S. Falke, C. Fuhs, J. Giesl: \textit{Analyzing Runtime and Size Complexity of Integer Programs}, TOPLAS '16, https://github.com/s-falke/kittel-koat

Recently significant progress in complexity analysis tools for Integer Transition Systems (ITSs):

- CoFloCo\textsuperscript{42}
- KoAT\textsuperscript{43}
- PUBS\textsuperscript{44}

Goal: use these tools to find upper bounds for TRS complexity in a modular way.

\textsuperscript{42} A. Flores-Montoya, R. Hähnle: Resource analysis of complex programs with cost equations, APLAS '14, https://github.com/aeflores/CoFloCo


Bottom-Up Complexity Analysis for TRSs

Recently significant progress in complexity analysis tools for **Integer Transition Systems (ITSs)**:

- CoFloCo\(^{42}\)
- KoAT\(^{43}\)
- PUBS\(^{44}\)

Goal: use these tools to find upper bounds for TRS complexity in a modular way

Works well in practice after resolving some technical pitfalls

---


Bottom-Up Complexity Analysis for TRSs

Recently significant progress in complexity analysis tools for Integer Transition Systems (ITSs):

- CoFloCo\textsuperscript{42}
- KoAT\textsuperscript{43}
- PUBS\textsuperscript{44}

Goal: use these tools to find upper bounds for TRS complexity in a modular way

Works well in practice after resolving some technical pitfalls

To do: Find “best” abstraction of data structures to integers automatically

\textsuperscript{42} A. Flores-Montoya, R. Hähnle: Resource analysis of complex programs with cost equations, APLAS '14, https://github.com/aeflores/CoFloCo


Recently significant progress in complexity analysis tools for **Integer Transition Systems (ITSs)**:

- CoFloCo\(^{42}\)
- KoAT\(^{43}\)
- PUBS\(^{44}\)

**Goal**: use these tools to find upper bounds for TRS complexity in a modular way

Works well in practice after resolving some technical pitfalls

**To do**: Find “best” abstraction of data structures to integers automatically

Abstract a list to its length, its size, its maximum element, ...?

---


AProVE finds (tight) upper bound $O(n^4)$ for $dc_R$:

1. Add generator rules $G$, so analyse $rc_R/G$ instead (FroCoS'19)
2. Detect: innermost is worst case here, analyse $irc_R/G$ instead (LPAR'17)
3. Transform TRS to Recursive Integer Transition System (RITS), analyse complexity of RITS instead (FroCoS'17)
4. ITS tools CoFloCo and KoAT find upper bounds for runtime and size of individual RITS functions, combine to complexity of RITS
5. Upper bound $O(n^4)$ for RITS complexity carries over to $dc_R$ of input!

AProVE finds lower bound $\Omega(n^3)$ for $dc_R$ using induction technique.

At termCOMP 2022:
https://www.starexec.org/starexec/services/jobs/pairs/567601324/stdout/1?limit=-1
AProVE finds (tight) upper bound $\mathcal{O}(n^4)$ for $\text{dc}_R$:
app(nil, y) → y          |          app(add(n, x), y) → add(n, app(x, y))
reverse(nil) → nil        |          reverse(add(n, x)) → app(reverse(x), add(n, nil))
shuffle(nil) → nil        |          shuffle(add(n, x)) → add(n, shuffle(reverse(x)))

AProVE finds (tight) upper bound $O(n^4)$ for $dc_R$:

1. Add generator rules $G$, so analyse $rc_{R/G}$ instead (FroCoS’19)
AProVE finds (tight) upper bound $O(n^4)$ for $\text{dc}_R$:

1. Add generator rules $\mathcal{G}$, so analyse $\text{rc}_R/\mathcal{G}$ instead (FroCoS’19)
2. Detect: innermost is worst case here, analyse $\text{irc}_R/\mathcal{G}$ instead (LPAR’17)

Derivational Complexity Full Rewriting/AG01/#3.12, TPDB
Applicatives:

\[
\text{app}(\text{nil, } y) \rightarrow y \\
\text{app}(\text{add}(n, x), y) \rightarrow \text{add}(n, \text{app}(x, y)) \\
\text{reverse}(\text{nil}) \rightarrow \text{nil} \\
\text{reverse}(\text{add}(n, x)) \rightarrow \text{app}(\text{reverse}(x), \text{add}(n, \text{nil})) \\
\text{shuffle}(\text{nil}) \rightarrow \text{nil} \\
\text{shuffle}(\text{add}(n, x)) \rightarrow \text{add}(n, \text{shuffle}(\text{reverse}(x)))
\]

AProVE finds (tight) upper bound $O(n^4)$ for $\text{dc}_R$:

1. Add generator rules $G$, so analyse $\text{rc}_{R/G}$ instead (FroCoS’19)
2. Detect: innermost is worst case here, analyse $\text{irc}_{R/G}$ instead (LPAR’17)
3. Transform TRS to Recursive Integer Transition System (RITS), analyse complexity of RITS instead (FroCoS’17)
AProVE finds (tight) upper bound $O(n^4)$ for $dc_R$:

1. Add generator rules $\mathcal{G}$, so analyse $rc_{R/G}$ instead (FroCoS’19)
2. Detect: innermost is worst case here, analyse $irc_{R/G}$ instead (LPAR’17)
3. Transform TRS to Recursive Integer Transition System (RITS), analyse complexity of RITS instead (FroCoS’17)
4. ITS tools CoFloCo and KoAT find upper bounds for runtime and size of individual RITS functions, combine to complexity of RITS
Derivational_Complexity_Full_Rewriting/AG01/#3.12, TPDB

\[
\begin{align*}
\text{app}(\text{nil}, y) & \rightarrow y & \text{app}(\text{add}(n, x), y) & \rightarrow \text{add}(n, \text{app}(x, y)) \\
\text{reverse}(\text{nil}) & \rightarrow \text{nil} & \text{reverse}(\text{add}(n, x)) & \rightarrow \text{app}(\text{reverse}(x), \text{add}(n, \text{nil})) \\
\text{shuffle}(\text{nil}) & \rightarrow \text{nil} & \text{shuffle}(\text{add}(n, x)) & \rightarrow \text{add}(n, \text{shuffle}(\text{reverse}(x)))
\end{align*}
\]

AProVE finds (tight) upper bound $O(n^4)$ for $d_{c_R}:$

1. Add generator rules $\mathcal{G}$, so analyse $r_{c_R/\mathcal{G}}$ instead (FroCoS’19)
2. Detect: innermost is worst case here, analyse $i_{c_R/\mathcal{G}}$ instead (LPAR’17)
3. Transform TRS to Recursive Integer Transition System (RITS), analyse complexity of RITS instead (FroCoS’17)
4. ITS tools CoFloCo and KoAT find upper bounds for runtime and size of individual RITS functions, combine to complexity of RITS
5. Upper bound $O(n^4)$ for RITS complexity carries over to $d_{c_R}$ of input!
\[
\text{app}(\text{nil}, y) \rightarrow y \quad \text{app}(\text{add}(n, x), y) \rightarrow \text{add}(n, \text{app}(x, y))
\]
\[
\text{reverse}(\text{nil}) \rightarrow \text{nil} \quad \text{reverse}(\text{add}(n, x)) \rightarrow \text{app}(\text{reverse}(x), \text{add}(n, \text{nil}))
\]
\[
\text{shuffle}(\text{nil}) \rightarrow \text{nil} \quad \text{shuffle}(\text{add}(n, x)) \rightarrow \text{add}(n, \text{shuffle}(\text{reverse}(x)))
\]

AProVE finds (tight) upper bound $\mathcal{O}(n^4)$ for $\text{dc}_R$:

1. Add generator rules $\mathcal{G}$, so analyse $\text{rc}_R/\mathcal{G}$ instead (FroCoS’19)
2. Detect: innermost is worst case here, analyse $\text{irc}_R/\mathcal{G}$ instead (LPAR’17)
3. Transform TRS to Recursive Integer Transition System (RITS), analyse complexity of RITS instead (FroCoS’17)
4. ITS tools CoFloCo and KoAT find upper bounds for runtime and size of individual RITS functions, combine to complexity of RITS
5. Upper bound $\mathcal{O}(n^4)$ for RITS complexity carries over to $\text{dc}_R$ of input!

AProVE finds lower bound $\Omega(n^3)$ for $\text{dc}_R$ using induction technique.
app(nil, y) → y | app(add(n, x), y) → add(n, app(x, y))
reverse(nil) → nil | reverse(add(n, x)) → app(reverse(x), add(n, nil))
shuffle(nil) → nil | shuffle(add(n, x)) → add(n, shuffle(reverse(x)))

AProVE finds (tight) upper bound $O(n^4)$ for $dc_R$:

1. Add generator rules $\mathcal{G}$, so analyse $rc_{R/\mathcal{G}}$ instead (FroCoS’19)
2. Detect: innermost is worst case here, analyse $irc_{R/\mathcal{G}}$ instead (LPAR’17)
3. Transform TRS to Recursive Integer Transition System (RITS), analyse complexity of RITS instead (FroCoS’17)
4. ITS tools CoFloCo and KoAT find upper bounds for runtime and size of individual RITS functions, combine to complexity of RITS
5. Upper bound $O(n^4)$ for RITS complexity carries over to $dc_R$ of input!

AProVE finds lower bound $\Omega(n^3)$ for $dc_R$ using induction technique.

At termCOMP 2022:

https://www.starexec.org/starexec/services/jobs/pairs/567601324/stdout/1?limit=-1
Automated tools for TRS Complexity at the Termination Competition 2022:

- **APoVE:** [https://aprove.informatik.rwth-aachen.de/](https://aprove.informatik.rwth-aachen.de/)
- **TcT:** [https://tcs-informatik.uibk.ac.at/tools/tct/](https://tcs-informatik.uibk.ac.at/tools/tct/)

For TcT Web, use only `VAR` and `RULES` entries in the text format and configure other aspects (e.g., start terms) in the web interface.
Automated tools for TRS Complexity at the Termination Competition 2022:

- AProVE: https://aprove.informatik.rwth-aachen.de/
- TcT: https://tcs-informatik.uibk.ac.at/tools/tct/

Web interfaces available:

- AProVE: https://aprove.informatik.rwth-aachen.de/interface
- TcT: http://colo6-c703.uibk.ac.at/tct/tct-trs/

---

45 For TcT Web, use only VAR and RULES entries in the text format and configure other aspects (e.g., start terms) in the web interface.
Automated tools for TRS Complexity at the Termination Competition 2022:

- **APoVE**: [https://aprove.informatik.rwth-aachen.de/](https://aprove.informatik.rwth-aachen.de/)
- **TcT**: [https://tcs-informatik.uibk.ac.at/tools/tct/](https://tcs-informatik.uibk.ac.at/tools/tct/)

Web interfaces available:

- **APoVE**: [https://aprove.informatik.rwth-aachen.de/interface](https://aprove.informatik.rwth-aachen.de/interface)
- **TcT**: [http://colo6-c703.uibk.ac.at/tct/tct-trs/](http://colo6-c703.uibk.ac.at/tct/tct-trs/)

Input format for runtime complexity:\(^{45}\)

```
(VAR x y)
(GOAL COMPLEXITY)
(STARTTERM CONSTRUCTOR-BASED)
(RULES
   plus(0, y) -> y
   plus(s(x), y) -> s(plus(x, y))
)
```

\(^{45}\)For TcT Web, use only VAR and RULES entries in the text format and configure other aspects (e.g., start terms) in the web interface.
Innermost runtime complexity:

(VAR x y)
(GOAL COMPLEXITY)
(STARTTERM CONSTRUCTOR-BASED)
(STRATEGY INNERMOST)
(RULES
   plus(0, y) -> y
   plus(s(x), y) -> s(plus(x, y))
)

Input for Automated Tools (2/4)
Derivational complexity:

(VAR x y)
(GOAL COMPLEXITY)
(STARTTERM UNRESTRICTED)
(RULES
  plus(0, y) -> y
  plus(s(x), y) -> s(plus(x, y))
)
Innermost derivational complexity:

(VAR x y)
(GOAL COMPLEXITY)
(STARTTERM UNRESTRICTED)
(STRATEGY INNERMOST)
(RULES
   plus(0, y) -> y
   plus(s(x), y) -> s(plus(x, y))
)

Input for Automated Tools (4/4)
A Landscape of Complexity Properties and Transformations

idc, irc: like dc, rc, but for *innermost* rewriting

TRS

FroCoS'19

dc → rc

FroCoS'19

idc → irc

LPAR'17

Rec. ITS irc

FroCoS'17

ITS irc

FroCoS'17
A Landscape of Complexity Properties and Transformations

idc, irc: like dc, rc, but for *innermost* rewriting

O Caml

dc \rightarrow rc

rec

Java

idc \rightarrow irc

FroCoS'19

Prolog

Rec. ITS irc

FroCoS'19

TRS

FroCoS'17

FroCoS'17
A Landscape of Complexity Properties and Transformations

OCaml

Java

Prolog

idc, irc: like dc, rc, but for innermost rewriting

related to:

- ICFP’15
- FroCoS’19
- I’18
- LPAR’17
- PPDP’12


G. Moser, M. Schaper: From Jinja bytecode to term rewriting: A complexity reflecting transformation, IC ’18

J. Giesl, T. Ströder, P. Schneider-Kamp, F. Emmes, C. Fuhs: Symbolic evaluation graphs and term rewriting: A general methodology for analyzing logic programs, PPDP ’12
Complexity analysis for functional programs (OCaml) by translation to term rewriting
Complexity analysis for functional programs (OCaml) by translation to term rewriting

Challenge for translation to TRS: OCaml is higher-order – functions can take functions as arguments:  \texttt{map}(F, xs)
Complexity analysis for functional programs (OCaml) by translation to term rewriting

Challenge for translation to TRS: OCaml is higher-order – functions can take functions as arguments: $\text{map}(F, xs)$

Solution:

- Defunctionalisation to: $a(a(\text{map}, F), xs)$
- Analyse start term with non-functional parameter types, then partially evaluate functions to instantiate higher-order variables
- Further program transformations

$\Rightarrow$ First-order TRS $\mathcal{R}$ with $rc_\mathcal{R}(n)$ an upper bound for the complexity of the OCaml program
Complexity analysis for Prolog programs and for Java programs by translation to term rewriting
Program Complexity Analysis via Term Rewriting: Prolog and Java

Complexity analysis for Prolog programs and for Java programs by translation to term rewriting

Common ideas:

- Analyse program via symbolic execution and generalisation (a form of abstract interpretation\(^49\))
- Deal with language specifics in program analysis
- Extract TRS \( R \) such that \( rc_R(n) \) is provably at least as high as runtime of program on input of size \( n \)
- Can represent tree structures of program as terms in TRS!

---

\(^{49}\) P. Cousot, R. Cousot: *Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints*, POPL ’77
Current Developments

- amortised complexity analysis for term rewriting

---

50 G. Moser, M. Schneckenreither: *Automated amortised resource analysis for term rewrite systems*, SCP '20
Current Developments

- **amortised** complexity analysis for term rewriting\(^{50}\)
- **probabilistic** term rewriting $\rightarrow$ upper bounds on expected runtime\(^{51}\)

---

\(^{50}\) G. Moser, M. Schneckenreither: *Automated amortised resource analysis for term rewrite systems*, SCP '20

\(^{51}\) M. Avanzini, U. Dal Lago, A. Yamada: *On probabilistic term rewriting*, SCP '20
Current Developments

- amortised complexity analysis for term rewriting\textsuperscript{50}
- probabilistic term rewriting $\rightarrow$ upper bounds on expected runtime\textsuperscript{51}
- complexity analysis for logically constrained rewriting with built-in data types from SMT theories (integers, booleans, arrays, \ldots)\textsuperscript{52}

\textsuperscript{50} G. Moser, M. Schneckenreither: Automated amortised resource analysis for term rewrite systems, SCP '20
\textsuperscript{51} M. Avanzini, U. Dal Lago, A. Yamada: On probabilistic term rewriting, SCP '20
\textsuperscript{52} S. Winkler, G. Moser: Runtime complexity analysis of logically constrained rewriting, LOPSTR '20
Current Developments

- **amortised** complexity analysis for term rewriting\(^{50}\)
- **probabilistic** term rewriting $\rightarrow$ upper bounds on **expected runtime**\(^{51}\)
- complexity analysis for **logically constrained rewriting** with built-in data types from SMT theories (integers, booleans, arrays, ... )\(^{52}\)
- direct analysis of complexity for **higher-order term rewriting**\(^{53}\)

\(^{50}\) G. Moser, M. Schneckenreither: *Automated amortised resource analysis for term rewrite systems*, SCP '20

\(^{51}\) M. Avanzini, U. Dal Lago, A. Yamada: *On probabilistic term rewriting*, SCP '20

\(^{52}\) S. Winkler, G. Moser: *Runtime complexity analysis of logically constrained rewriting*, LOPSTR '20

\(^{53}\) C. Kop, D. Vale: *Tuple interpretations for higher-order rewriting*, FSCD '21
Current Developments

- **amortised** complexity analysis for term rewriting\(^{50}\)
- **probabilistic** term rewriting \(\rightarrow\) upper bounds on expected runtime\(^{51}\)
- complexity analysis for **logically constrained rewriting** with built-in data types from SMT theories (integers, booleans, arrays, \ldots )\(^{52}\)
- direct analysis of complexity for **higher-order term rewriting**\(^{53}\)
- analysis of **parallel**-innermost runtime complexity\(^{54}\)

\(^{50}\) G. Moser, M. Schneckenreither: *Automated amortised resource analysis for term rewrite systems*, SCP '20

\(^{51}\) M. Avanzini, U. Dal Lago, A. Yamada: *On probabilistic term rewriting*, SCP '20

\(^{52}\) S. Winkler, G. Moser: *Runtime complexity analysis of logically constrained rewriting*, LOPSTR '20

\(^{53}\) C. Kop, D. Vale: *Tuple interpretations for higher-order rewriting*, FSCD '21

\(^{54}\) T. Baudon, C. Fuhs, L. Gonnord: *Analysing parallel complexity of term rewriting*, LOPSTR '22
Termination and complexity analysis: active fields of research
Termination and complexity analysis: active fields of research

Push-button tools to prove (non-)termination and to infer upper and lower complexity bounds available
Termination and complexity analysis: active fields of research

Push-button tools to prove (non-)termination and to infer upper and lower complexity bounds available

Cross-fertilisation between techniques for different formalisms (integer transition systems, functional programs, . . . )
Termination and Complexity: Conclusion

- Termination and complexity analysis: active fields of research
- Push-button tools to prove (non-)termination and to infer upper and lower complexity bounds available
- Cross-fertilisation between techniques for different formalisms (integer transition systems, functional programs, . . .)
- Certification helps raise trust in automatically found proofs of (non-)termination and complexity bounds

Thanks a lot for your attention!
Termination and Complexity: Conclusion

- Termination and complexity analysis: active fields of research
- Push-button tools to prove (non-)termination and to infer upper and lower complexity bounds available
- Cross-fertilisation between techniques for different formalisms (integer transition systems, functional programs, . . .)
- Certification helps raise trust in automatically found proofs of (non-)termination and complexity bounds

Thanks a lot for your attention!


