Automated Complexity Analysis for Term Rewriting

Carsten Fuhs

Birkbeck, University of London

Course at the International School on Rewriting 2021
Madrid, Spain\textsuperscript{1}
5\textsuperscript{th} July 2021

https://www.dcs.bbk.ac.uk/~carsten/isr2021/

\textsuperscript{1}virtually
What is *Term Rewriting*?

(1) Core functional programming language

without many restrictions (and features) of “real” FP:

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

Compute “double of 3 is 6”:

\[
\begin{align*}
\text{double}(s(s(s(0)))) & \rightarrow R \\
\end{align*}
\]
What is \textit{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, . . .)

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

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

Compute “double of 3 is 6”:

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

in 4 steps with $\rightarrow R$. 

2/62
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
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) \( \mathcal{R} \))

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

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

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

double(0) \( \rightarrow \) 0
double(s(x)) \( \rightarrow \) s(s(double(x)))

Compute “double of 3 is 6”:

double(s(s(s(0)))) \( \rightarrow_{R} \) s(s(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) $\mathcal{R}$)

- $\text{double}(0) \rightarrow 0$
- $\text{double}(s(x)) \rightarrow s(s(\text{double}(x)))$

Compute “double of 3 is 6”:

$\text{double}(s(s(s(0)))) \rightarrow_{\mathcal{R}} s(s(\text{double}(s(s(0)))))$

$\rightarrow_{\mathcal{R}} s(s(s(s(s(\text{double}(s(s(0))))))))$
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>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>double(0) \rightarrow 0</code></td>
<td></td>
</tr>
<tr>
<td><code>double(s(x)) \rightarrow s(s(double(x)))</code></td>
<td></td>
</tr>
</tbody>
</table>

Compute “double of 3 is 6”:

\[
\begin{align*}
\text{double}(s(s(s(0)))) & \rightarrow_{\mathcal{R}} s(s(\text{double}(s(s(0))))) \\
& \rightarrow_{\mathcal{R}} s(s(s(s(\text{double}(s(0))))))) \\
& \rightarrow_{\mathcal{R}} s(s(s(s(s(s(0)))))))
\end{align*}
\]
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) \( \mathcal{R} \))**

<table>
<thead>
<tr>
<th>Rule</th>
<th>Term</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>double(0) → 0</code></td>
<td></td>
</tr>
<tr>
<td><code>double(s(x)) → s(s(double(x)))</code></td>
<td></td>
</tr>
</tbody>
</table>

Compute “double of 3 is 6”:

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

**double**$(0)$ $\rightarrow$ 0
**double**$(s(x))$ $\rightarrow$ $s(s(double(x)))$

Compute “double of 3 is 6”:

$double(s(s(s(0))))$
$\rightarrow_\mathcal{R}$ $s(s(double(s(s(0)))))$
$\rightarrow_\mathcal{R}$ $s(s(s(s(double(s(s(0)))))$)
$\rightarrow_\mathcal{R}$ $s(s(s(s(s(double(s(s(s(0)))))))))$
$\rightarrow_\mathcal{R}$ $s(s(s(s(s(s(double(s(s(s(s(0)))))))))))$

in 4 steps with $\rightarrow_\mathcal{R}$
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, . . .)

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

---

**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”:

\[
\begin{align*}
\text{double}(s^3(0)) & \rightarrow \mathcal{R} s^2(\text{double}(s^2(0))) \\
& \rightarrow \mathcal{R} s^4(\text{double}(s(0))) \\
& \rightarrow \mathcal{R} s^6(\text{double}(0)) \\
& \rightarrow \mathcal{R} s^6(0)
\end{align*}
\]

in 4 steps with $\rightarrow \mathcal{R}$
What is \textit{Complexity} of Term Rewriting?

\textbf{Given:} TRS $\mathcal{R}$ (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 $R$ (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)
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 $\xrightarrow{\mathcal{R}}$ sequence from a term of size $n$ become?  
(worst case)

**Here:** Does $\mathcal{R}$ 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 $\mathcal{R}$ (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
What is Complexity of Term Rewriting?

Given: TRS \( \mathcal{R} \) (e.g., \{ \text{double}(0) \rightarrow 0, \text{double}(\text{s}(x)) \rightarrow \text{s}(\text{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!

\[
\text{double}(\text{s}^{n-2}(0)) \overset{n-1}{\rightarrow} \text{s}^{2n-4}(0)
\]

- 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 $R$ (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
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
- $rc_{\mathcal{R}}(n)$ for program analysis

(2) No!
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
   - $rc_{\mathcal{R}}(n)$ for program analysis

2. **No!**
   
   \[
   \begin{align*}
   \text{double}^3(s(0)) & \rightarrow_{\mathcal{R}}^2 \text{double}^2(s^2(0)) \rightarrow_{\mathcal{R}}^3 \text{double}(s^4(0)) \rightarrow_{\mathcal{R}}^5 s^8(0)
   \end{align*}
   \] in 10 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
- $rc_\mathcal{R}(n)$ for program analysis

(2) No!

$$\text{double}^3(s(0)) \rightarrow_\mathcal{R}^2 \text{double}^2(s^2(0)) \rightarrow_\mathcal{R}^3 \text{double}(s^4(0)) \rightarrow_\mathcal{R}^5 s^8(0) \text{ in 10 steps}$$

- $\text{double}^{n-2}(s(0))$ allows $\Theta(2^n)$ many steps to $s^{2n-2}(0)$
What is Complexity of Term Rewriting?

Given: TRS $\mathcal{R}$ (e.g., \{ 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
- $rc_{\mathcal{R}}(n)$ 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!**

   $$\text{double}^3(s(0)) \rightarrow_{\mathcal{R}}^2 \text{double}^2(s^2(0)) \rightarrow_{\mathcal{R}}^3 \text{double}(s^4(0)) \rightarrow_{\mathcal{R}}^5 s^8(0)$$ 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
   - $dc_{\mathcal{R}}(n)$ 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 $\mathcal{R}\mathcal{E}$
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
**A Short Timeline (1/2)**

**1989:** Derivational complexity introduced, linked to termination proofs\(^2\)

\(^2\) 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\(^2\)

2001: Techniques for polynomial upper complexity bounds\(^3\)

\(^2\) D. Hofbauer, C. Lautemann: *Termination proofs and the length of derivations*, RTA ’89

\(^3\) 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\(^2\)
2001: Techniques for polynomial upper complexity bounds\(^3\)
2008: Runtime complexity introduced with first analysis techniques\(^4\)

---

\(^2\)D. Hofbauer, C. Lautemann: *Termination proofs and the length of derivations*, RTA ’89
\(^3\)G. Bonfante, A. Cichon, J. Marion, and H. Touzet: *Algorithms with polynomial interpretation termination proof*, JFP ’01
\(^4\)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

---

2 D. Hofbauer, C. Lautemann: *Termination proofs and the length of derivations*, RTA ’89
3 G. Bonfante, A. Cichon, J. Marion, and H. Touzet: *Algorithms with polynomial interpretation termination proof*, JFP ’01
4 N. Hirokawa, G. Moser: *Automated complexity analysis based on the dependency pair method*, IJCAR ’08
1989: Derivational complexity introduced, linked to termination proofs\(^2\)

2001: Techniques for polynomial upper complexity bounds\(^3\)

2008: Runtime complexity introduced with first analysis techniques\(^4\)

2008: First automated tools to find complexity bounds: TcT\(^5\), CaT\(^6\)

2008: First complexity analysis categories in the Termination Competition

http://termination-portal.org/wiki/Termination_Competition

\(^2\) D. Hofbauer, C. Lautemann: *Termination proofs and the length of derivations*, RTA ‘89

\(^3\) G. Bonfante, A. Cichon, J. Marion, and H. Touzet: *Algorithms with polynomial interpretation termination proof*, JFP ‘01

\(^4\) 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 proofs²
2001: Techniques for polynomial upper complexity bounds³
2008: Runtime complexity introduced with first analysis techniques⁴
2008: First automated tools to find complexity bounds: TcT⁵, CaT⁶
2008: First complexity analysis categories in the Termination Competition

http://termination-portal.org/wiki/Termination_Competition

---

² D. Hofbauer, C. Lautemann: *Termination proofs and the length of derivations*, RTA ’89
³ G. Bonfante, A. Cichon, J. Marion, and H. Touzet: *Algorithms with polynomial interpretation termination proof*, JFP ’01
⁴ N. Hirokawa, G. Moser: *Automated complexity analysis based on the dependency pair method*, IJCAR ’08
2021: Termination Competition 2021 with complexity analysis tools AProVE\textsuperscript{7}, TcT in July 2021

https://termcomp.github.io/Y2021-1

First run just finished!

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 $dh(t, \rightarrow) = \omega$.

Example:

$$dh\left(\text{double}\left(\text{split}(0)\right)\right), \rightarrow_{\mathcal{R}}) = 4$$

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 $\text{double}$, we have $dc_{\mathcal{R}}(n) \in \Theta(2^n)$. 

7/62
Some Definitions

Definition (Derivation Height $\text{dh}$)
For a term $t \in \mathcal{T}(\mathcal{F}, \mathcal{V})$ and a relation $\rightarrow$, the **derivation height** is:

$$
\text{dh}(t, \rightarrow) = \sup \{ \ n \mid \exists t'. t \rightarrow^n t' \} 
$$

If $t$ starts an infinite $\rightarrow$-sequence, we set $\text{dh}(t, \rightarrow) = \omega$.

$\text{dh}(t, \rightarrow)$: length of the longest $\rightarrow$-sequence from $t$. 

Example: $\text{dh}(\text{double}(\text{s}(\text{s}(\text{s}(0)))))$, $\rightarrow_R = 4$
Some Definitions

**Definition (Derivation Height \( \text{dh} \))**

For a term \( t \in \mathcal{T}(\mathcal{F}, \mathcal{V}) \) and a relation \( \rightarrow \), the **derivation height** is:

\[
\text{dh}(t, \rightarrow) = \sup \{ n \mid \exists t'. t \rightarrow^n t' \}
\]

If \( t \) starts an infinite \( \rightarrow \)-sequence, we set \( \text{dh}(t, \rightarrow) = \omega \).

\( \text{dh}(t, \rightarrow) \): length of the longest \( \rightarrow \)-sequence from \( t \).

**Example:** \( \text{dh( } \text{double}\big(s(s(s(0)))) , \rightarrow_{\mathcal{R}} \big) = 4 \)
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 $t \in \mathcal{T}(\mathcal{F}, \mathcal{V})$ 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 $\mathcal{R}$, 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 $n$
Some Definitions

Definition (Derivation Height $\text{dh}$)

For a term $t \in \mathcal{T}(\mathcal{F}, \mathcal{V})$ and a relation $\rightarrow$, the derivational height is:

$$\text{dh}(t, \rightarrow) = \sup \{ n \mid \exists t'. t \rightarrow^n t' \}$$

If $t$ starts an infinite $\rightarrow$-sequence, we set $\text{dh}(t, \rightarrow) = \omega$.

$\text{dh}(t, \rightarrow)$: length of the longest $\rightarrow$-sequence from $t$.

**Example:** $\text{dh}(\text{double}(s(s(s(0)))), \rightarrow_{\mathcal{R}}) = 4$

Definition (Derivational Complexity $\text{dc}$)

For a TRS $\mathcal{R}$, the derivational complexity is:

$$\text{dc}_{\mathcal{R}}(n) = \sup \{ \text{dh}(t, \rightarrow_{\mathcal{R}}) \mid t \in \mathcal{T}(\mathcal{F}, \mathcal{V}), |t| \leq n \}$$

$\text{dc}_{\mathcal{R}}(n)$: length of the longest $\rightarrow_{\mathcal{R}}$-sequence from a term of size at most $n$

**Example:** For $\mathcal{R}$ for double, we have $\text{dc}_{\mathcal{R}}(n) \in \Theta(2^n)$. 
The Bad News for automation:

For a given TRS $R$, the following questions are undecidable:

- $dc_R(n) = \omega$ for some $n$ (→ non-termination!)
- $dc_R(n)$ polynomially bounded?

Goal: find approximations for derivational complexity

Initial focus: find upper bounds $dc_R(n) \in O(\ldots)$

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!)

---

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:

- $\text{dc}_{\mathcal{R}}(n) = \omega$ for some $n$? ($\rightarrow$ termination!)
- $\text{dc}_{\mathcal{R}}(n)$ polynomially bounded?

\footnote{A. Schnabl and J. G. Simonsen: \textit{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:

- $dc_{\mathcal{R}}(n) = \omega$ for some $n$? ($\rightarrow$ termination!)
- $dc_{\mathcal{R}}(n)$ polynomially bounded?

Goal: find approximations for derivational complexity

Initial focus: find upper bounds

$$dc_{\mathcal{R}}(n) \in \mathcal{O}(...)$$

---

8 A. Schnabl and J. G. Simonsen: *The exact hardness of deciding derivational and runtime complexity*, CSL ’11
Example (double)

de\text{ouble}(0) \rightarrow 0
\text{double}(s(x)) \rightarrow s(s(\text{double}(x)))
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.
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\(^9\) \([ \cdot ]\) over \(\mathbb{N}\): \( \ell \succ r \iff [\ell] > [r] \)

---

\(^9\) D. Lankford: *Canonical algebraic simplification in computational logic*, U Texas ’75
Show $dc_{\mathcal{R}}(n) < \omega$ by termination proof with reduction order $\succ$ on terms. Get $\succ$ via polynomial interpretation\(^9\) $[\cdot]$ over $\mathbb{N}$: $\ell \succ r \iff [\ell] \succ [r]$

**Example:** 

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

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

---

\(^9\)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 $\text{dc}_R(n) < \omega$ by termination proof with reduction order $\succ$ on terms. Get $\succ$ via polynomial interpretation\(^9\) $[\cdot]$ over $\mathbb{N}$: $\ell \succ r \iff [\ell] \succ [r]$

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

Extend to terms:

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

---

\(^9\) D. Lankford: *Canonical algebraic simplification in computational logic*, U Texas ’75
Derivational Complexity from Polynomial Interpretations (1/2)

Example (double)

<table>
<thead>
<tr>
<th>double(0)</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>double(s(x))</td>
<td>s(s(double(x)))</td>
</tr>
<tr>
<td>3 · x + 3</td>
<td>3 · x + 2</td>
</tr>
</tbody>
</table>

Show $dc_R(n) < \omega$ by termination proof with reduction order $\succ$ on terms. Get $\succ$ via polynomial interpretation $^9$ $\cdot$ 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:

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

---

$^9$ D. Lankford: Canonical algebraic simplification in computational logic, U Texas ’75
Derivational Complexity from Polynomial Interpretations (1/2)

Example (double)

\[
\begin{align*}
\text{double}(0) & \succ 0 & \quad 3 & \succ 1 \\
\text{double}(s(x)) & \succ s(s(\text{double}(x))) & \quad 3 \cdot x + 3 & \succ 3 \cdot x + 2
\end{align*}
\]

Show \(d_{\mathcal{R}}(n) < \omega\) by termination proof with reduction order \(\succ\) on terms. Get \(\succ\) via polynomial interpretation\(^9\) \([\cdot]\) over \(\mathbb{N}\): \(\ell \succ r \iff [\ell] \succ [r]\)

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

Extend to terms:

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

Automated search for \([\cdot]\) via SAT\(^10\) or SMT\(^11\) solving

---

\(^9\) D. Lankford: Canonical algebraic simplification in computational logic, U Texas '75

\(^10\) C. Fuhs, J. Giesl, A. Middeldorp, P. Schneider-Kamp, R. Thiemann, H. Zankl: SAT solving for termination analysis with polynomial interpretations, SAT '07

\(^11\) C. Borralleras, S. Lucas, A. Oliveras, E. Rodríguez-Carbonell, A. Rubio: SAT modulo linear arithmetic for solving polynomial constraints, JAR '12
Example (double)

| double(0)       | ≻ | 0   | | 3  | > | 1   |
|-----------------|---|-----||-----|---|-----|
| double(s(x))    | ≻ | s(s(double(x))) | | 3·x + 3 | > | 3·x + 2 |

Example: [double](x) = 3·x, [s](x) = x + 1, [0] = 1

This proves more than just termination...
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 \(d_{cR}(n)\))

- **Termination proof for TRS \(R\) with polynomial interpretation**

\[
\Rightarrow d_{cR}(n) \in 2^{O(n)}
\]

\(^{12}\) D. Hofbauer, C. Lautemann: *Termination proofs and the length of derivations*, RTA ’89
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 \(d_{c_R}(n)\) from polynomial interpretations\(^{12}\))

- **Termination proof for TRS \(R\) with polynomial interpretation**
  \[
  \Rightarrow d_{c_R}(n) \in 2^{O(n)}
  \]

- **Termination proof for TRS \(R\) with linear polynomial interpretation**
  \[
  \Rightarrow d_{c_R}(n) \in 2^{O(n)}
  \]

\(^{12}\) D. Hofbauer, C. Lautemann: *Termination proofs and the length of derivations*, RTA '89
Termination proof for TRS $\mathcal{R}$ with ... 

- matchbounds$^{13}$
- arctic matrix interpretations$^{14}$

$\Rightarrow d_{\mathcal{R}}(n) \in \mathcal{O}(n)$

---

$^{13}$ A. Geser, D. Hofbauer, J. Waldmann: *Match-bounded string rewriting systems*, AAECC '04

$^{14}$ A. Koprowski, J. Waldmann: *Max/plus tree automata for termination of term rewriting*, Acta Cyb. '09
Termination proof for TRS $\mathcal{R}$ with ...

- matchbounds\(^{13}\) \(\Rightarrow dc_{\mathcal{R}}(n) \in O(n)\)
- arctic matrix interpretations\(^{14}\) \(\Rightarrow dc_{\mathcal{R}}(n) \in O(n)\)
- triangular matrix interpretation\(^{15}\) \(\Rightarrow dc_{\mathcal{R}}(n)\) is at most polynomial
- matrix interpretation of spectral radius\(^{16}\) \(\leq 1\) \(\Rightarrow dc_{\mathcal{R}}(n)\) is at most polynomial

---

\(^{13}\) A. Geser, D. Hofbauer, J. Waldmann: *Match-bounded string rewriting systems*, AAECC '04

\(^{14}\) A. Koprowski, J. Waldmann: *Max/plus tree automata for termination of term rewriting*, Acta Cyb. '09

\(^{15}\) G. Moser, A. Schnabl, J. Waldmann: *Complexity analysis of term rewriting based on matrix and context dependent interpretations*, FSTTCS '08

\(^{16}\) 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\(^{13}\)
  \[ \Rightarrow \text{dc}_{\mathcal{R}}(n) \in \mathcal{O}(n) \]
- arctic matrix interpretations\(^{14}\)
  \[ \Rightarrow \text{dc}_{\mathcal{R}}(n) \in \mathcal{O}(n) \]
- triangular matrix interpretation\(^{15}\)
  \[ \Rightarrow \text{dc}_{\mathcal{R}}(n) \text{ is at most polynomial} \]
- matrix interpretation of spectral radius\(^{16}\) \(\leq 1\)
  \[ \Rightarrow \text{dc}_{\mathcal{R}}(n) \text{ is at most polynomial} \]
- standard matrix interpretation\(^{17}\)
  \[ \Rightarrow \text{dc}_{\mathcal{R}}(n) \text{ is at most exponential} \]

---

\(^{13}\) A. Geser, D. Hofbauer, J. Waldmann: *Match-bounded string rewriting systems*, AAECC ’04

\(^{14}\) A. Koprowski, J. Waldmann: *Max/plus tree automata for termination of term rewriting*, Acta Cyb. ’09

\(^{15}\) G. Moser, A. Schnabl, J. Waldmann: *Complexity analysis of term rewriting based on matrix and context dependent interpretations*, FSTTCS ’08

\(^{16}\) F. Neurauter, H. Zankl, A. Middeldorp: *Revisiting matrix interpretations for polynomial derivational complexity of term rewriting*, LPAR (Yogyakarta) ’10

\(^{17}\) J. Endrullis, J. Waldmann, and H. Zantema: *Matrix interpretations for proving termination of term rewriting*, JAR ’08
Termination proof for TRS $\mathcal{R}$ with . . .

- lexicographic path order$^{18} \Rightarrow \text{dc}_{\mathcal{R}}(n)$ is at most multiple recursive$^{19}$

---

$^{18}$ S. Kamin, J.-J. Lévy: *Two generalizations of the recursive path ordering*, U Illinois ’80

$^{19}$ A. Weiermann: *Termination proofs for term rewriting systems by lexicographic path orderings imply multiply recursive derivation lengths*, TCS ’95
Derivational Complexity from Termination Proofs (2/2)

Termination proof for TRS $\mathcal{R}$ with . . .

- lexicographic path order\(^{18}\) $\Rightarrow d_{c\mathcal{R}}(n)$ is at most multiple recursive\(^{19}\)
- Dependency Pairs method\(^{20}\) with dependency graphs and usable rules
  $\Rightarrow d_{c\mathcal{R}}(n)$ is at most primitive recursive\(^{21}\)

\[\begin{align*}
\text{\(^{18}\) S. Kamin, J.-J. Lévy: Two generalizations of the recursive path ordering, U Illinois '80} \\
\text{\(^{19}\) A. Weiermann: Termination proofs for term rewriting systems by lexicographic path} \\
\text{orderings imply multiply recursive derivation lengths, TCS '95} \\
\text{\(^{20}\) T. Arts, J. Giesl: Termination of term rewriting using dependency pairs, TCS '00} \\
\text{\(^{21}\) G. Moser, A. Schnabl: The derivational complexity induced by the dependency pair} \\
\text{method, LMCS '11}
\end{align*}\]
Termination proof for TRS $\mathcal{R}$ with . . .

- lexicographic path order\(^{18}\) $\Rightarrow d_{c\mathcal{R}}(n)$ is at most multiple recursive\(^{19}\)
- Dependency Pairs method\(^{20}\) with dependency graphs and usable rules $\Rightarrow d_{c\mathcal{R}}(n)$ is at most primitive recursive\(^{21}\)
- Dependency Pairs framework\(^{22,23}\) with dependency graphs, reduction pairs, subterm criterion $\Rightarrow d_{c\mathcal{R}}(n)$ is at most multiple recursive\(^{24}\)

---

\(^{18}\) S. Kamin, J.-J. Lévy: *Two generalizations of the recursive path ordering*, U Illinois ’80

\(^{19}\) A. Weiermann: *Termination proofs for term rewriting systems by lexicographic path orderings imply multiply recursive derivation lengths*, TCS ’95

\(^{20}\) T. Arts, J. Giesl: *Termination of term rewriting using dependency pairs*, TCS ’00

\(^{21}\) G. Moser, A. Schnabl: *The derivational complexity induced by the dependency pair method*, LMCS ’11

\(^{22}\) J. Giesl, R. Thiemann, P. Schneider-Kamp, S. Falke: *Mechanizing and improving dependency pairs*, JAR ’06

\(^{23}\) N. Hirokawa and A. Middeldorp: *Tyrolean Termination Tool: Techniques and features*, IC ’07

\(^{24}\) G. Moser, A. Schnabl: *Termination proofs in the dependency pair framework may induce multiple recursive derivational complexity*, RTA ’11
So far: upper bounds for derivational complexity
So far: upper bounds for derivational complexity
But: derivational complexity counter-intuitive, often infeasible
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)

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)

For a TRS \( R \), the runtime complexity is:

\[
rc_R(n) = \sup \left\{ dh(t, \rightarrow_R) \mid t \in T_{\text{basic}}, |t| \leq n \right\}
\]

\( rc_R(n) \): 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
- Wanted: complexity of evaluation of \texttt{double on data}: \texttt{double(s^n(0))}

Definition (Basic Term$^{25}$)

For \texttt{defined symbols} $\mathcal{D}$ and \texttt{constructor symbols} $\mathcal{C}$, the term

$$f(t_1, \ldots, t_n)$$

is in the set $\mathcal{T}_{\text{basic}}$ of \texttt{basic terms} iff $f \in \mathcal{D}$ and $t_1, \ldots, t_n \in \mathcal{T}(\mathcal{C}, \mathcal{V})$.

---

$^{25}$N. Hirokawa, G. Moser: \textit{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: `double(s^n(0))`

Definition (Basic Term\textsuperscript{25})

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 $t_1, \ldots, t_n \in \mathcal{T}(\mathcal{C}, \mathcal{V})$.

Definition (Runtime Complexity $rc$\textsuperscript{25})

For a TRS $\mathcal{R}$, the runtime complexity is:

$$rc_{\mathcal{R}}(n) = \sup \{ dh(t, \rightarrow_{\mathcal{R}}) \mid t \in \mathcal{T}_{\text{basic}}, |t| \leq n \}$$

\textsuperscript{25}N. Hirokawa, G. Moser: Automated complexity analysis based on the dependency pair method, IJCAR ’08
So far: upper bounds for derivational complexity
But: derivational complexity counter-intuitive, often infeasible
Wanted: complexity of evaluation of \texttt{double} on data: \texttt{double}(s^n(0))

\begin{itemize}
  \item For \texttt{defined symbols} $\mathcal{D}$ and \texttt{constructor symbols} $\mathcal{C}$, the term $f(t_1, \ldots, t_n)$ is in the set $\mathcal{T}_{\text{basic}}$ of \texttt{basic terms} iff $f \in \mathcal{D}$ and $t_1, \ldots, t_n \in \mathcal{T}(\mathcal{C}, \mathcal{V})$.
\end{itemize}

For a TRS $\mathcal{R}$, the \texttt{runtime complexity} is:

$$rc_{\mathcal{R}}(n) = \sup \{ dh(t, \rightarrow_{\mathcal{R}}) \mid t \in \mathcal{T}_{\text{basic}}, |t| \leq n \}$$

$rc_{\mathcal{R}}(n)$: like derivational complexity... but for basic terms only!

\footnote{N. Hirokawa, G. Moser: \textit{Automated complexity analysis based on the dependency pair method}, IJCAR '08}
Polynomial interpretations can induce upper bounds to runtime complexity:\textsuperscript{26}

**Definition (Strongly linear polynomial, restricted interpretation)**

- Polynomial $p$ is **strongly linear** iff
  \[ p(x_1, \ldots, x_n) = x_1 + \cdots + x_n + a \text{ for some } a \in \mathbb{N}. \]
- 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 $c \in \mathbb{N}$.\textsuperscript{26}

---

\textsuperscript{26} 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 $rc_\mathcal{R}(n)$ from restricted interpretations)**

Termination proof for TRS $\mathcal{R}$ with **restricted** interpretation $[\cdot]$ of degree at most $d$ for $[f]$  

$$\Rightarrow rc_\mathcal{R}(n) \in \mathcal{O}(n^d)$$

---

26 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 $[f]$  

$$\Rightarrow rc_R(n) \in O(n^d)$$

**Example:** $[\text{double}](x) = 3 \cdot x$, $[s](x) = x + 1$, $[0] = 1$ is restricted, degree 1  

$$\Rightarrow rc_R(n) \in O(n)$$ for TRS $R$ for **double**

---

26 G. Bonfante, A. Cichon, J. Marion, H. Touzet: *Algorithms with polynomial interpretation termination proof*, JFP ’01
Here: innermost rewriting ($\approx$ call-by-value)

**Example (reverse)**

<table>
<thead>
<tr>
<th>Function</th>
<th>Rule</th>
<th>Function</th>
<th>Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>app(nil, y)</code></td>
<td>$\rightarrow y$</td>
<td><code>app(add(n, x), y)</code></td>
<td>$\rightarrow add(n, app(x, y))$</td>
</tr>
<tr>
<td><code>reverse(nil)</code></td>
<td>$\rightarrow nil$</td>
<td><code>reverse(add(n, x))</code></td>
<td>$\rightarrow app(reverse(x), add(n, nil))$</td>
</tr>
</tbody>
</table>
Dependency Tuples for *Innermost* Runtime Complexity irc

Here: innermost rewriting (≈ call-by-value)

**Example (reverse)**

\[
\begin{align*}
\text{app}(\text{nil}, y) & \rightarrow y & \text{app}(\text{add}(n, x), y) & \rightarrow \text{add}(n, \text{app}(x, y)) \\
\text{reverse}(\text{nil}) & \rightarrow \text{nil} & \text{reverse}(\text{add}(n, x)) & \rightarrow \text{app}(\text{reverse}(x), \text{add}(n, \text{nil}))
\end{align*}
\]

For rule \( \ell \rightarrow r \), eval of \( \ell \) costs \( 1 + \) eval of all function calls in \( r \) together:

\[\text{Com}_k\]

\[\text{Com}_k\]

\[\text{Com}_k\]

\[\text{Com}_k\]

\[\text{Com}_k\]

---

27 L. Noschinski, F. Emmes, J. Giesl: *Analyzing innermost runtime complexity of term rewriting by dependency pairs*, JAR '13
Dependency Tuples for *Innermost* Runtime Complexity irc

Here: innermost rewriting (≈ call-by-value)

**Example (reverse)**

<table>
<thead>
<tr>
<th>Rule</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>app(nil, y) → y</code></td>
<td></td>
</tr>
<tr>
<td><code>reverse(nil) → nil</code></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Rule</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>app(add(n, x), y) → add(n, app(x, y))</code></td>
<td></td>
</tr>
<tr>
<td><code>reverse(add(n, x)) → app(reverse(x), add(n, nil))</code></td>
<td></td>
</tr>
</tbody>
</table>

For rule `ℓ → r`, eval of `ℓ` costs `1 + eval of all function calls in r together`:

**Example (Dependency Tuples$^{27}$ for reverse)**

<table>
<thead>
<tr>
<th>Rule</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>app^(nil, y) → Com_0</code></td>
<td></td>
</tr>
<tr>
<td><code>app^(add(n, x), y) → Com_1(app^(x, y))</code></td>
<td></td>
</tr>
<tr>
<td><code>reverse^(nil) → Com_0</code></td>
<td></td>
</tr>
<tr>
<td><code>reverse^(add(n, x)) → Com_2(app^(reverse(x), add(n, nil)), reverse^(x))</code></td>
<td></td>
</tr>
</tbody>
</table>

- Function calls to count marked with `#`
- Compound symbols `Com_k` group function calls together

---

$^{27}$ L. Noschinski, F. Emmes, J. Giesl: *Analyzing innermost runtime complexity of term rewriting by dependency pairs*, JAR '13
Example (reverse, Dependency Tuples for reverse)

<table>
<thead>
<tr>
<th>Function</th>
<th>Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>app#(nil, y)</code></td>
<td><code>Com_0</code></td>
</tr>
<tr>
<td><code>app#(add(n, x), y)</code></td>
<td><code>Com_1(app#(x, y))</code></td>
</tr>
<tr>
<td><code>reverse#(nil)</code></td>
<td><code>Com_0</code></td>
</tr>
<tr>
<td><code>reverse#(add(n, x))</code></td>
<td><code>Com_2(app#(reverse(x), add(n, nil)), reverse#(x))</code></td>
</tr>
<tr>
<td><code>app(nil, y)</code></td>
<td><code>y</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(nil)</code></td>
<td><code>nil</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>
Polynomial Interpretations for Dependency Tuples

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

\[
\begin{align*}
[nil] &= 0 \\
[\text{app}](x_1, x_2) &= x_1 + x_2 \\
[\text{app}^\#](x_1, x_2) &= x_1 + 1 \\
[\text{add}](x_1, x_2) &= x_2 + 1 \quad (\leq \text{restricted interpret.}) \\
[\text{reverse}](x_1) &= x_1 \quad (\text{bounds helper fct. result size}) \\
[\text{reverse}^\#](x_1) &= x_1^2 + x_1 + 1 \quad (\text{complexity of fct.})
\end{align*}
\]

to show $[\ell] \geq [r]$ for all rules and $[\ell] \geq 1 + [r]$ for all Dependency Tuples

Maximum degree of $[\cdot]$ is 2 $\Rightarrow \text{irc}_{\mathcal{R}}(n) \in \mathcal{O}(n^2)$
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
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\(^{28}\)

\(^{28}\)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\(^{28}\).
- Extensions by polynomial path orders\(^{29}\), usable replacement maps\(^{30}\), a combination framework for complexity analysis\(^{31}\), ...
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
- 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\(^\text{32}\) inspired by proving non-termination

\(^{\text{32}}\)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^{33}
(1) Induction technique, inspired by **non-looping** non-termination\(^{33}\)

- 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 \(rc_R(n) \in \Omega(p'(n))\).

---

\(^{33}\) 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

- Generate infinite family $\mathcal{T}_{\text{witness}}$ of basic terms as witnesses in

$$\forall n \in \mathbb{N}. \exists t_n \in \mathcal{T}_{\text{witness}}. \quad |t_n| \leq q(n) \land \text{dh}(t_n, \rightarrow_R) \geq p(n)$$

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)

---

33 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

- 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 $rc_\mathcal{R}(n) \in \Omega(p'(n))$.

- 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

---

33 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\(^{33}\)

- Generate infinite family \(\mathcal{T}_{\text{witness}}\) of basic terms as witnesses in

\[
\forall n \in \mathbb{N}. \ \exists t_n \in \mathcal{T}_{\text{witness}}. \ |t_n| \leq q(n) \ \land \ \text{dh}(t_n, \xrightarrow{\mathcal{R}}) \geq p(n)
\]

to conclude \(rc_{\mathcal{R}}(n) \in \Omega(p'(n))\).

- 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 \(n = 0, 1, \ldots, k\)

\(^{33}\)F. Emmes, T. Enger, J. Giesl: Proving non-looping non-termination automatically, IJCAR '12
(1) Induction technique, inspired by **non-looping** non-termination\(^{33}\)

- Generate infinite family \(T_{\text{witness}}\) of basic terms as witnesses in

\[
\forall n \in \mathbb{N}. \ \exists t_n \in T_{\text{witness}}. \ |t_n| \leq q(n) \ \land \ \text{dh}(t_n, \rightarrow_{\mathcal{R}}) \geq p(n)
\]

to conclude \(r c_{\mathcal{R}}(n) \in \Omega(p'(n))\).

- 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 \(t_n \rightarrow_{\mathcal{R}}^{\geq p(n)} t'_n\) inductively

---

\(^{33}\) F. Emmes, T. Enger, J. Giesl: *Proving non-looping non-termination automatically*, IJCAR '12
(1) Induction technique, inspired by non-looping non-termination\textsuperscript{33}

- Generate infinite family $\mathcal{T}_{\text{witness}}$ of basic terms as witnesses in

\[ \forall n \in \mathbb{N}. \quad \exists t_n \in \mathcal{T}_{\text{witness}}. \quad |t_n| \leq q(n) \quad \land \quad \text{dh}(t_n, \rightarrow_R) \geq p(n) \]

to conclude $r_c R(n) \in \Omega(p'(n))$.

- 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 $r_c R(n)$ from $p(n)$ in rewrite lemma and $q(n)$

\textsuperscript{33} F. Emmes, T. Enger, J. Giesl: Proving non-looping non-termination automatically, IJCAR ’12
Example (quicksort)

\[
\begin{align*}
qs(\text{nil}) & \rightarrow \text{nil} \\
qs(\text{cons}(x, xs)) & \rightarrow qs(\text{low}(x, xs)) ++ \text{cons}(x, qs(\text{low}(x, xs))) \\
\text{low}(x, \text{nil}) & \rightarrow \text{nil} \\
\text{low}(x, \text{cons}(y, ys)) & \rightarrow \text{if}(x \leq y, x, \text{cons}(y, ys)) \\
\text{if}(\text{tt}, x, \text{cons}(y, ys)) & \rightarrow \text{low}(x, ys) \\
\text{if}(\text{ff}, x, \text{cons}(y, ys)) & \rightarrow \text{cons}(y, \text{low}(x, ys)) \\
\ldots
\end{align*}
\]
Example (quicksort)

\[
\begin{align*}
qs(nil) & \rightarrow \text{nil} \\
qs(\text{cons}(x, xs)) & \rightarrow qs(\text{low}(x, xs)) \mathbf{++} \text{cons}(x, qs(\text{low}(x, xs))) \\
\text{low}(x, \text{nil}) & \rightarrow \text{nil} \\
\text{low}(x, \text{cons}(y, ys)) & \rightarrow \text{if}(x \leq y, x, \text{cons}(y, ys)) \\
\text{if}(\text{tt}, x, \text{cons}(y, ys)) & \rightarrow \text{low}(x, ys) \\
\text{if}(\text{ff}, x, \text{cons}(y, ys)) & \rightarrow \text{cons}(y, \text{low}(x, ys)) \\
& \ldots
\end{align*}
\]

Speculate and prove rewrite lemma:

\[
qs(\text{cons}(\text{zero}, \ldots, \text{cons}(\text{zero}, \text{nil})))) \rightarrow^{3n^2+2n+1} \text{cons}(\text{zero}, \ldots, \text{cons}(\text{zero}, \text{nil}))
\]
Finding Lower Bounds by Induction: Example

Example (quicksort)

\[
\begin{align*}
qs(\text{nil}) & \rightarrow \text{nil} \\
qs(\text{cons}(x, xs)) & \rightarrow qs(\text{low}(x, xs)) ++ \text{cons}(x, qs(\text{low}(x, xs))) \\
\text{low}(x, \text{nil}) & \rightarrow \text{nil} \\
\text{low}(x, \text{cons}(y, ys)) & \rightarrow \text{if}(x \leq y, x, \text{cons}(y, ys)) \\
\text{if}(\text{tt}, x, \text{cons}(y, ys)) & \rightarrow \text{low}(x, ys) \\
\text{if}(\text{ff}, x, \text{cons}(y, ys)) & \rightarrow \text{cons}(y, \text{low}(x, ys)) \\
& \ldots
\end{align*}
\]

Speculate and prove rewrite lemma:

\[
\begin{align*}
qs(\text{cons}(\text{zero}, \ldots, \text{cons}(\text{zero}, \text{nil}))) & \rightarrow 3n^2+2n+1 \quad \text{cons}(\text{zero}, \ldots, \text{cons}(\text{zero}, \text{nil})) \\
qs(\text{cons}^n(\text{zero}, \text{nil})) & \rightarrow 3n^2+2n+1 \quad \text{cons}(\text{zero}, \ldots, \text{cons}(\text{zero}, \text{nil}))
\end{align*}
\]
Finding Lower Bounds by Induction: Example

**Example (quicksort)**

\[
\begin{align*}
qs(\text{nil}) & \rightarrow \text{nil} \\
qs(\text{cons}(x, xs)) & \rightarrow qs(\text{low}(x, xs)) ++ \text{cons}(x, qs(\text{low}(x, xs))) \\
\text{low}(x, \text{nil}) & \rightarrow \text{nil} \\
\text{low}(x, \text{cons}(y, ys)) & \rightarrow \text{if}(x \leq y, x, \text{cons}(y, ys)) \\
\text{if}(\text{tt}, x, \text{cons}(y, ys)) & \rightarrow \text{low}(x, ys) \\
\text{if}(\text{ff}, x, \text{cons}(y, ys)) & \rightarrow \text{cons}(y, \text{low}(x, ys)) \\
& \ldots
\end{align*}
\]

Speculate and prove rewrite lemma:

\[
\begin{align*}
qs(\text{cons}(\text{zero}, \ldots, \text{cons}(\text{zero}, \text{nil})))) & \rightarrow 3n^2 + 2n + 1 \quad \text{cons}(\text{zero}, \ldots, \text{cons}(\text{zero}, \text{nil}))) \\
qs(\text{cons}^n(\text{zero}, \text{nil})) & \rightarrow 3n^2 + 2n + 1 \quad \text{cons}(\text{zero}, \ldots, \text{cons}(\text{zero}, \text{nil})))
\end{align*}
\]

From \(|qs(\text{cons}^n(\text{zero}, \text{nil}))| = 2n + 2\) we get

\[
rc_R(2n + 2) \geq 3n^2 + 2n + 1
\]
Finding Lower Bounds by Induction: Example

Example (quicksort)

\[
\begin{align*}
qs(\text{nil}) & \rightarrow \text{nil} \\
qs(\text{cons}(x, xs)) & \rightarrow qs(\text{low}(x, xs)) ++ \text{cons}(x, qs(\text{low}(x, xs))) \\
\text{low}(x, \text{nil}) & \rightarrow \text{nil} \\
\text{low}(x, \text{cons}(y, ys)) & \rightarrow \text{if}(x \leq y, x, \text{cons}(y, ys)) \\
\text{if}(\text{tt}, x, \text{cons}(y, ys)) & \rightarrow \text{low}(x, ys) \\
\text{if}(\text{ff}, x, \text{cons}(y, ys)) & \rightarrow \text{cons}(y, \text{low}(x, ys))
\end{align*}
\]

Speculate and prove rewrite lemma:

\[
\begin{align*}
qs(\text{cons}(\text{zero}, \ldots, \text{cons}(\text{zero}, \text{nil})))) & \rightarrow 3n^2 + 2n + 1 \text{cons}(\text{zero}, \ldots, \text{cons}(\text{zero}, \text{nil}))) \\
qs(\text{cons}^n(\text{zero}, \text{nil})) & \rightarrow 3n^2 + 2n + 1 \text{cons}(\text{zero}, \ldots, \text{cons}(\text{zero}, \text{nil})))
\end{align*}
\]

From \(|qs(\text{cons}^n(\text{zero}, \text{nil})))| = 2n + 2\) we get

\[
rc_R(2n + 2) \geq 3n^2 + 2n + 1 \text{ and } rc_R(n) \in \Omega(n^2).
\]
Finding Linear Lower Bounds by Decreasing Loops

(2) Decreasing loops, inspired by looping non-termination with

$$s \xrightarrow{+} R C[s\sigma] \xrightarrow{+} R C[C\sigma[s\sigma^2]] \xrightarrow{+} \cdots$$

**Example:** $f(y) \rightarrow f(s(y))$ has loop $f(y) \rightarrow R f(s(y))$ with $\sigma(y) = 0$. 
Finding Linear Lower Bounds by Decreasing Loops

(2) Decreasing loops, inspired by **looping** non-termination with

\[ s \rightarrow^+_R C[s\sigma] \rightarrow^+_R C[C\sigma[s\sigma^2]] \rightarrow^+_R \cdots \]

**Example:** \( f(y) \rightarrow f(s(y)) \) has loop \( f(y) \rightarrow^+_R f(s(y)) \) 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 **looping** non-termination with
\[ s \xrightarrow{+} R C[s\sigma] \xrightarrow{+} R C[C\sigma[s\sigma^2]] \xrightarrow{+} R \cdots \]

**Example:** \( f(y) \xrightarrow{} f(s(y)) \) has loop \( f(y) \xrightarrow{+} R f(s(y)) \) 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)

**Example:** \( \text{plus}(s(x), y) \xrightarrow{} \text{plus}(x, s(y)) \) has decreasing loop
\[ \text{plus}(s(x), y) \xrightarrow{+} R \text{plus}(x, s(y)) \text{ with } D[x] = s(x) \]
Finding Linear Lower Bounds by Decreasing Loops

(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)

**Example:** \( \text{plus}(s(x), y) \rightarrow \text{plus}(x, s(y)) \) has decreasing loop

\[ \text{plus}(s(x), y) \rightarrow^+ \mathcal{R} \text{plus}(x, s(y)) \] with \( D[x] = s(x) \)

for base term \( s = \text{plus}(x, y) \), **pumping substitution** \( \theta = [x \mapsto s(x)] \), and **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:** \( \text{fib}(s(s(n)))) \rightarrow \text{plus}(\text{fib}(s(n)), \text{fib}(n)) \) has 2 decreasing loops:

\[
\begin{align*}
\text{fib}(s(s(n))) & \rightarrow^{+} C[\text{fib}(s(n))] \quad \text{and} \quad \text{fib}(s(s(n))) \rightarrow^{+} C[\text{fib}(n)]
\end{align*}
\]

Implies \( rc(n) \in \Omega(2^n) \)!
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:

\[
\text{fib}(s(s(n))) \rightarrow^+ \mathcal{R} C[\text{fib}(s(n))] \quad \text{and} \quad \text{fib}(s(s(n))) \rightarrow^+ \mathcal{R} C[\text{fib}(n)]
\]

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:

\[
\text{tr}(\text{node}(x, y)) \rightarrow^+ \mathcal{R} C[\text{tr}(x)] \quad \text{and} \quad \text{tr}(\text{node}(x, y)) \rightarrow^+ \mathcal{R} C[\text{tr}(y)]
\]

are not compatible (their pumping substitutions do not commute).
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:

  \[
  \text{fib}(s(s(n))) \rightarrow_{\mathcal{R}}^+ C[\text{fib}(s(n))] \quad \text{and} \quad \text{fib}(s(s(n))) \rightarrow_{\mathcal{R}}^+ C[\text{fib}(n)]
  \]

  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:

  \[
  \text{tr}(\text{node}(x, y)) \rightarrow_{\mathcal{R}}^+ C[\text{tr}(x)] \quad \text{and} \quad \text{tr}(\text{node}(x, y)) \rightarrow_{\mathcal{R}}^+ C[\text{tr}(y)]
  \]

  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
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

⇒ First try decreasing loops, then induction technique
Benefits of Induction Technique:
- Can find \textbf{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!
idc, irc: like dc, rc, but for \textit{innermost} rewriting
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
A Landscape of Complexity Properties and Transformations

idc, irc: like dc, rc, but for \textit{innermost} rewriting


\textbf{TRS}

\begin{itemize}
\item dc \rightarrow rc \quad \text{FroCoS'19}^{35}
\item idc \rightarrow irc \quad \text{FroCoS'19}
\item \quad \text{LPAR'17}^{34}
\end{itemize}

\footnotesize{\textsuperscript{34} F. Frohn, J. Giesl: \textit{Analyzing runtime complexity via innermost runtime complexity}, LPAR '17
\textsuperscript{35} C. Fuhs: \textit{Transforming Derivational Complexity of Term Rewriting to Runtime Complexity}, FroCoS '19}
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 = $dc_R$ analysis tool"

- **Benefits**: Get analysis of derivational complexity "for free". Progress in runtime complexity analysis automatically improves derivational complexity analysis.
The big picture:

- **Have:** Tool for automated analysis of runtime complexity $rc_R$
- **Want:** Tool for automated analysis of derivational complexity $dc_R$
The big picture:

- **Have**: Tool for automated analysis of runtime complexity $rc_{\mathcal{R}}$
- **Want**: Tool for automated analysis of derivational complexity $dc_{\mathcal{R}}$
- **Idea**:
  
  "$rc_{\mathcal{R}}$ analysis tool + transformation on TRS $\mathcal{R} = dc_{\mathcal{R}}$ analysis tool"
The big picture:

- **Have:** Tool for automated analysis of runtime complexity $\text{rc}_R$
- **Want:** Tool for automated analysis of derivational complexity $\text{dc}_R$
- **Idea:**
  
  
  "$\text{rc}_R$ analysis tool + transformation on TRS $R = \text{dc}_R$ analysis tool"

- **Benefits:**
  
  - Get analysis of derivational complexity "for free"
  - Progress in runtime complexity analysis automatically improves derivational complexity analysis
program transformation such that runtime complexity of transformed TRS is \textit{identical} to derivational complexity of original TRS
program transformation such that runtime complexity of transformed TRS is identical to derivational complexity of original TRS

transformation correct also from idc to irc
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
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\(^\text{36}\) relative to state of the art TcT

\(^{36}\)Termination Problem Data Base, standard benchmark source for annual Termination and Complexity Competition (TermComp) with 1000s of problems, [http://termination-portal.org/wiki/TPDB](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 $f$

Example (Generator rules $G$)

- $\text{enc}\ double(x) \rightarrow double(\text{argenc}(x))$
- $\text{enc} 0 \rightarrow 0$
- $\text{enc}\ s(x) \rightarrow s(\text{argenc}(x))$
- $\text{argenc}(c\ double(x)) \rightarrow double(\text{argenc}(x))$
- $\text{argenc}(0) \rightarrow 0$
- $\text{argenc}(s(x)) \rightarrow s(\text{argenc}(x))$
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 $dc$ 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

$$t = \text{double} (\text{double} (\text{double} (s(0))))$$
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**

\[ \text{bv}(t) = \text{enc}_{\text{double}}(\text{c}_{\text{double}}(\text{c}_{\text{double}}(\text{s}(0)))) \]
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**
\[ \text{bv}(t) = \text{enc}_{\text{double}}(c_{\text{double}}(c_{\text{double}}(s(0)))) ) \]

**Example (Generator rules $G$)**

- $\text{enc}_{\text{double}}(x) \rightarrow \text{double}(\text{argenc}(x))$
- $\text{enc}_0 \rightarrow 0$
- $\text{enc}_s(x) \rightarrow s(\text{argenc}(x))$
- $\text{argenc}(c_{\text{double}}(x)) \rightarrow \text{double}(\text{argenc}(x))$
- $\text{argenc}(0) \rightarrow 0$
- $\text{argenc}(s(x)) \rightarrow s(\text{argenc}(x))$
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 $|t|$
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 $\mathcal{G}$ 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 $|t|$
- $\text{bv}(t) \rightarrow^* \text{t}$

Example (Generator rules $\mathcal{G}$)

\[
\begin{align*}
\text{enc}_{\text{double}}(x) &\rightarrow \text{double}(\text{argenc}(x)) \\
\text{enc}_0 &\rightarrow 0 \\
\text{enc}_s(x) &\rightarrow \text{s}(\text{argenc}(x)) \\
\text{argenc}(c_{\text{double}}(x)) &\rightarrow \text{double}(\text{argenc}(x)) \\
\text{argenc}(0) &\rightarrow 0 \\
\text{argenc}(s(x)) &\rightarrow \text{s}(\text{argenc}(x))
\end{align*}
\]
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)
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!
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 \( \text{idc} \): both AProVE and TcT with transformation are stronger than standard TcT

- upper bounds \( \text{dc} \): TcT stronger than AProVE and TcT with transformation, but AProVE still solves some new examples

- lower bounds \( \text{idc} \) and \( \text{dc} \): heuristics do not seem to benefit much

\[ \Rightarrow \] Transformation-based approach should be part of the portfolio of analysis tools for derivational complexity
Derivational Complexity: Future Work

- Possible applications
  - compiler simplifications
  - SMT solver preprocessing

Start terms may have nested defined symbols, so $dc_R$ is appropriate.
Possible applications

- compiler simplifications
- SMT solver preprocessing

Start terms may have nested defined symbols, so $d_{cR}$ 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 ($\mathcal{T}_{\text{basic}} \subseteq \mathcal{U} \subseteq \mathcal{T}$).
Possible applications
- compiler simplifications
- SMT solver preprocessing

Start terms may have nested defined symbols, so $d_{cR}$ is appropriate.

Go between derivational and runtime complexity
- So far: encode full term universe $T$ via basic terms $T_{\text{basic}}$
- Generalise: write relative rules to generate arbitrary set $U$ of terms "between" basic and all terms ($T_{\text{basic}} \subseteq U \subseteq T$).

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

Rec. ITS irc

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

---

37 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

---

37 M. Naaf, F. Frohn, M. Brockschmidt, C. Fuhs, J. Giesl: *Complexity analysis for term rewriting by integer transition systems*, FroCoS '17
Recently significant progress in complexity analysis tools for **Integer Transition Systems (ITSs):**

- CoFloCo\(^{38}\)
- KoAT\(^{39}\)
- PUBS\(^{40}\)

Goal: use these tools to find upper bounds for TRS complexity

---


### Example

<table>
<thead>
<tr>
<th>Expression</th>
<th>Reduced to</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>isort(nil, ys)</code></td>
<td><code>ys</code></td>
</tr>
<tr>
<td><code>isort(cons(x, xs), ys)</code></td>
<td><code>isort(xs, insert(x, ys))</code></td>
</tr>
<tr>
<td><code>insert(x, nil)</code></td>
<td><code>cons(x, nil)</code></td>
</tr>
<tr>
<td><code>insert(x, cons(y, ys))</code></td>
<td><code>if(gt(x, y), x, cons(y, ys))</code></td>
</tr>
<tr>
<td><code>if(true, x, cons(y, ys))</code></td>
<td><code>cons(y, insert(x, ys))</code></td>
</tr>
<tr>
<td><code>if(false, x, cons(y, ys))</code></td>
<td><code>cons(x, cons(y, ys))</code></td>
</tr>
<tr>
<td><code>gt(0, y)</code></td>
<td><code>false</code></td>
</tr>
<tr>
<td><code>gt(s(x), 0)</code></td>
<td><code>true</code></td>
</tr>
<tr>
<td><code>gt(s(x), s(y))</code></td>
<td><code>gt(x, y)</code></td>
</tr>
</tbody>
</table>
Analysing irc of Insertion Sort by Hand: Bottom-Up

Example

\[
\begin{align*}
\text{isort}(\text{nil}, ys) & \rightarrow ys \\
\text{isort}(\text{cons}(x, xs), ys) & \rightarrow \text{isort}(xs, \text{insert}(x, ys)) \\
\text{insert}(x, \text{nil}) & \rightarrow \text{cons}(x, \text{nil}) \\
\text{insert}(x, \text{cons}(y, ys)) & \rightarrow \text{if}(\text{gt}(x, y), x, \text{cons}(y, ys)) \\
\text{if}(\text{true}, x, \text{cons}(y, ys)) & \rightarrow \text{cons}(y, \text{insert}(x, ys)) \\
\text{if}(\text{false}, x, \text{cons}(y, ys)) & \rightarrow \text{cons}(x, \text{cons}(y, ys)) \\
\text{gt}(0, y) & \equiv \text{false} \\
\text{gt}(s(x), 0) & \equiv \text{true} \\
\text{gt}(s(x), s(y)) & \rightarrow \text{gt}(x, y)
\end{align*}
\]

Note: innermost reduction strategy
Analysing irc of Insertion Sort by Hand: Bottom-Up

Example

\[
\begin{align*}
\text{isort}(\text{nil}, ys) & \rightarrow ys \\
\text{isort}(&\text{cons}(x, xs), ys) \rightarrow \text{isort}(xs, \text{insert}(x, ys)) \\
\text{insert}(x, \text{nil}) & \rightarrow \text{cons}(x, \text{nil}) \\
\text{insert}(x, \text{cons}(y, ys)) & \rightarrow \text{if}(\text{gt}(x, y), x, \text{cons}(y, ys)) \\
\text{if}(&\text{true}, x, \text{cons}(y, ys)) \rightarrow \text{cons}(y, \text{insert}(x, ys)) \\
\text{if}(\text{false}, x, \text{cons}(y, ys)) & \rightarrow \text{cons}(x, \text{cons}(y, ys)) \\
\text{gt}(0, y) & \equiv \text{false} \\
\text{gt}(s(x), 0) & \equiv \text{true} \\
\text{gt}(s(x), s(y)) & \equiv \text{gt}(x, y)
\end{align*}
\]

\[\text{rt(}\text{gt}(x, y)\text{)} \in \mathcal{O}(1) \quad ("\Rightarrow" \text{ for relative rules})\]

Note: innermost reduction strategy
Analysing irc of Insertion Sort by Hand: Bottom-Up

Example

\[
\begin{align*}
isort(nil, ys) & \rightarrow ys \\
isort(\text{cons}(x, xs), ys) & \rightarrow isort(xs, \text{insert}(x, ys)) \\
\text{insert}(x, nil) & \rightarrow \text{cons}(x, nil) \\
\text{insert}(x, \text{cons}(y, ys)) & \rightarrow \text{if}(\text{gt}(x, y), x, \text{cons}(y, ys)) \\
\text{if}(true, x, \text{cons}(y, ys)) & \rightarrow \text{cons}(y, \text{insert}(x, ys)) \\
\text{if}(false, x, \text{cons}(y, ys)) & \rightarrow \text{cons}(x, \text{cons}(y, ys)) \\
\text{gt}(0, y) & \equiv \text{false} \\
\text{gt}(s(x), 0) & \equiv \text{true} \\
\text{gt}(s(x), s(y)) & \equiv \text{gt}(x, y)
\end{align*}
\]

- \(\text{rt}(\text{gt}(x, y)) \in \mathcal{O}(1)\) ("\(\rightarrow\"\) for relative rules)
- \(\text{rt}(\text{insert}(x, ys)) \in \mathcal{O}(\text{length}(ys))\)

Note: innermost reduction strategy
Analysing irc of Insertion Sort by Hand: Bottom-Up

Example

<table>
<thead>
<tr>
<th>Rule</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>isort(nil, ys) → ys</td>
<td>(\text{isort}(\text{nil}, \text{ys}) \rightarrow \text{ys})</td>
</tr>
<tr>
<td>isort(cons(x, xs), ys) → isort(xs, insert(x, ys))</td>
<td>(\text{isort}(\text{cons}(x, \text{xs}), \text{ys}) \rightarrow \text{isort}(\text{xs}, \text{insert}(x, \text{ys})))</td>
</tr>
<tr>
<td>insert(x, nil) → cons(x, nil)</td>
<td>(\text{insert}(x, \text{nil}) \rightarrow \text{cons}(x, \text{nil}))</td>
</tr>
<tr>
<td>insert(x, cons(y, ys)) → if(gt(x, y), x, cons(y, ys))</td>
<td>(\text{insert}(x, \text{cons}(y, \text{ys})) \rightarrow \text{if}(\text{gt}(x, y), x, \text{cons}(y, \text{ys})))</td>
</tr>
<tr>
<td>if(true, x, cons(y, ys)) → cons(y, insert(x, ys))</td>
<td>(\text{if}(\text{true}, x, \text{cons}(y, \text{ys})) \rightarrow \text{cons}(y, \text{insert}(x, \text{ys})))</td>
</tr>
<tr>
<td>if(false, x, cons(y, ys)) → cons(x, cons(y, ys))</td>
<td>(\text{if}(\text{false}, x, \text{cons}(y, \text{ys})) \rightarrow \text{cons}(x, \text{cons}(y, \text{ys})))</td>
</tr>
<tr>
<td>(\text{gt}(0, y) \equiv \text{false})</td>
<td>(\text{gt}(0, y) \equiv \text{false})</td>
</tr>
<tr>
<td>(\text{gt}(\text{s}(x), 0) \equiv \text{true})</td>
<td>(\text{gt}(\text{s}(x), 0) \equiv \text{true})</td>
</tr>
<tr>
<td>(\text{gt}(\text{s}(x), \text{s}(y)) \equiv \text{gt}(x, y))</td>
<td>(\text{gt}(\text{s}(x), \text{s}(y)) \equiv \text{gt}(x, y))</td>
</tr>
</tbody>
</table>

- \(\text{rt}(\text{gt}(x, y)) \in \mathcal{O}(1)\) ("\(\rightarrow\)" for relative rules)
- \(\text{rt}(\text{insert}(x, \text{ys})) \in \mathcal{O}(\text{length}(\text{ys}))\)
- \(\text{rt}(\text{isort}(\text{xs}, \text{ys})) \in \mathcal{O}(\text{length}(\text{xs}) \cdot \ldots)\)

Note: innermost reduction strategy
Analysing irc of Insertion Sort by Hand: Bottom-Up

Example

\[
\begin{align*}
\text{isort}(\text{nil}, ys) & \rightarrow ys \\
\text{isort}(\text{cons}(x, xs), ys) & \rightarrow \text{isort}(xs, \text{insert}(x, ys)) \\
\text{insert}(x, \text{nil}) & \rightarrow \text{cons}(x, \text{nil}) \\
\text{insert}(x, \text{cons}(y, ys)) & \rightarrow \text{if}(\text{gt}(x, y), x, \text{cons}(y, ys)) \\
\text{if}(\text{true}, x, \text{cons}(y, ys)) & \rightarrow \text{cons}(y, \text{insert}(x, ys)) \\
\text{if}(\text{false}, x, \text{cons}(y, ys)) & \rightarrow \text{cons}(x, \text{cons}(y, ys)) \\
\text{gt}(0, y) & \equiv \text{false} \\
\text{gt}(s(x), 0) & \equiv \text{true} \\
\text{gt}(s(x), s(y)) & \equiv \text{gt}(x, y)
\end{align*}
\]

- \(\text{rt}(\text{gt}(x, y)) \in O(1)\) ("\(\twoheadrightarrow\)" for relative rules)
- \(\text{rt}(\text{insert}(x, ys)) \in O(\text{length}(ys))\)
- \(\text{rt}(\text{isort}(xs, ys)) \in O(\text{length}(xs) \cdot (\text{length}(xs) + \text{length}(ys)))\)

Note: innermost reduction strategy
Using Dependency Tuples: Top-Down

Example

\[
\begin{align*}
isort(nil, ys) & \rightarrow ys \\
isort(\text{cons}(x, xs), ys) & \rightarrow \text{isort}(xs, \text{insert}(x, ys)) \\
isort(x, nil) & \rightarrow \text{cons}(x, nil) \\
\text{insert}(x, cons(y, ys)) & \rightarrow \text{if}(\text{gt}(x, y), x, \text{cons}(y, ys)) \\
\text{if}(true, x, cons(y, ys)) & \rightarrow \text{cons}(y, \text{insert}(x, ys)) \\
\text{if}(false, x, cons(y, ys)) & \rightarrow \text{cons}(x, \text{cons}(y, ys)) \\
\text{gt}(0, y) & \equiv false \\
\text{gt}(s(x), 0) & \equiv true \\
\text{gt}(s(x), s(y)) & \equiv \text{gt}(x, y)
\end{align*}
\]

- The recursive \textit{isort} rule is at most applied linearly often
Using Dependency Tuples: Top-Down

Example

\[
\begin{align*}
\text{isort}(\text{nil}, y) & \rightarrow y \\
\text{isort}(\text{cons}(x, xs), y) & \rightarrow \text{isort}(xs, \text{insert}(x, y)) \\
\text{insert}(x, \text{nil}) & \rightarrow \text{cons}(x, \text{nil}) \\
\text{insert}(x, \text{cons}(y, y)) & \rightarrow \text{if}(\text{gt}(x, y), x, \text{cons}(y, y)) \\
\text{if}(\text{true}, x, \text{cons}(y, y)) & \rightarrow \text{cons}(y, \text{insert}(x, y)) \\
\text{if}(\text{false}, x, \text{cons}(y, y)) & \rightarrow \text{cons}(x, \text{cons}(y, y)) \\
\text{gt}(0, y) & \equiv \text{false} \\
\text{gt}(s(x), 0) & \equiv \text{true} \\
\text{gt}(s(x), s(y)) & \equiv \text{gt}(x, y)
\end{align*}
\]

- the recursive \text{isort} rule is at most applied linearly often
- the recursive \text{insert} rule is at most applied quadratically often
Using Dependency Tuples: Top-Down

Example

\[
\begin{align*}
\text{isort}(\text{nil}, ys) & \rightarrow ys \\
\text{isort}(\text{cons}(x, xs), ys) & \rightarrow \text{isort}(xs, \text{insert}(x, ys)) \\
\text{insert}(x, \text{nil}) & \rightarrow \text{cons}(x, \text{nil}) \\
\text{insert}(x, \text{cons}(y, ys)) & \rightarrow \text{if}(\text{gt}(x, y), x, \text{cons}(y, ys)) \\
\text{if}(\text{true}, x, \text{cons}(y, ys)) & \rightarrow \text{cons}(y, \text{insert}(x, ys)) \\
\text{if}(\text{false}, x, \text{cons}(y, ys)) & \rightarrow \text{cons}(x, \text{cons}(y, ys)) \\
\text{gt}(0, y) & \equiv \text{false} \\
\text{gt}(s(x), 0) & \equiv \text{true} \\
\text{gt}(s(x), s(y)) & \equiv \text{gt}(x, y)
\end{align*}
\]

- the recursive \text{isort} rule is at most applied linearly often
- the recursive \text{insert} rule is at most applied quadratically often
  - note: requires reasoning about \text{isort}, \text{insert}, and \text{if} rules!
Using Dependency Tuples: Top-Down

### Example

<table>
<thead>
<tr>
<th>Rule</th>
<th>Equivalent Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>isort(nil, ys)</code></td>
<td><code>ys</code></td>
</tr>
<tr>
<td><code>isort(cons(x, xs), ys)</code></td>
<td><code>isort(xs, insert(x, ys))</code></td>
</tr>
<tr>
<td><code>insert(x, nil)</code></td>
<td><code>cons(x, nil)</code></td>
</tr>
<tr>
<td><code>insert(x, cons(y, ys))</code></td>
<td><code>if(gt(x, y), x, cons(y, ys))</code></td>
</tr>
<tr>
<td><code>if(true, x, cons(y, ys))</code></td>
<td><code>cons(y, insert(x, ys))</code></td>
</tr>
<tr>
<td><code>if(false, x, cons(y, ys))</code></td>
<td><code>cons(x, cons(y, ys))</code></td>
</tr>
<tr>
<td><code>gt(0, y)</code></td>
<td><code>false</code></td>
</tr>
<tr>
<td><code>gt(s(x), 0)</code></td>
<td><code>true</code></td>
</tr>
<tr>
<td><code>gt(s(x), s(y))</code></td>
<td><code>gt(x, y)</code></td>
</tr>
</tbody>
</table>

- the recursive `isort` rule is at most applied linearly often
- the recursive `insert` rule is at most applied quadratically often
  - note: requires reasoning about `isort`, `insert`, and `if` rules!
  - found via quadratic polynomial interpretation
**Using Dependency Tuples: Top-Down**

### Example

- $\text{isort}(\text{nil}, ys) \rightarrow ys$
- $\text{isort}(\text{cons}(x, xs), ys) \rightarrow \text{isort}(xs, \text{insert}(x, ys))$
- $\text{insert}(x, \text{nil}) \rightarrow \text{cons}(x, \text{nil})$
- $\text{insert}(x, \text{cons}(y, ys)) \rightarrow \text{if}(\text{gt}(x, y), x, \text{cons}(y, ys))$
- $\text{if}(\text{true}, x, \text{cons}(y, ys)) \rightarrow \text{cons}(y, \text{insert}(x, ys))$
- $\text{if}(\text{false}, x, \text{cons}(y, ys)) \rightarrow \text{cons}(x, \text{cons}(y, ys))$
- $\text{gt}(0, y) \equiv \text{false}$
- $\text{gt}(s(x), 0) \equiv \text{true}$
- $\text{gt}(s(x), s(y)) \equiv \text{gt}(x, y)$

- **the recursive** $\text{isort}$ **rule is at most applied linearly often**
- **the recursive** $\text{insert}$ **rule is at most applied quadratically often**
  - **note:** requires reasoning about $\text{isort}$, $\text{insert}$, and $\text{if}$ rules!
  - **found** via quadratic polynomial interpretation
- **the recursive** $\text{if}$ **rule is applied as often as the recursive** $\text{insert}$ **rule**
Example

\[
\begin{align*}
\text{isort}(\text{nil}, ys) & \rightarrow ys \\
\text{isort}(\text{cons}(x, xs), ys) & \rightarrow \text{isort}(xs, \text{insert}(x, ys)) \\
\text{insert}(x, \text{nil}) & \rightarrow \text{cons}(x, \text{nil}) \\
\text{insert}(x, \text{cons}(y, ys)) & \rightarrow \text{if}(\text{gt}(x, y), x, \text{cons}(y, ys)) \\
\text{if}(\text{true}, x, \text{cons}(y, ys)) & \rightarrow \text{cons}(y, \text{insert}(x, ys)) \\
\text{if}(\text{false}, x, \text{cons}(y, ys)) & \rightarrow \text{cons}(x, \text{cons}(y, ys)) \\
\text{gt}(0, y) & \equiv \text{false} \\
\text{gt}(s(x), 0) & \equiv \text{true} \\
\text{gt}(s(x), s(y)) & \equiv \text{gt}(x, y)
\end{align*}
\]

abstract terms to integers
## Bird’s Eye View of the Transformation

### Example

<table>
<thead>
<tr>
<th>Function</th>
<th>Rule</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>isort(xs', ys)</code></td>
<td>$\frac{1}{\rightarrow} ys$</td>
<td>$xs' = 1$</td>
</tr>
<tr>
<td><code>isort(cons(x, xs), ys)</code></td>
<td>$\rightarrow isort(xs, insert(x, ys))$</td>
<td></td>
</tr>
<tr>
<td><code>insert(x, nil)</code></td>
<td>$\rightarrow cons(x, nil)$</td>
<td></td>
</tr>
<tr>
<td><code>insert(x, cons(y, ys))</code></td>
<td>$\rightarrow if(gt(x, y), x, cons(y, ys))$</td>
<td></td>
</tr>
<tr>
<td><code>if(true, x, cons(y, ys))</code></td>
<td>$\rightarrow cons(y, insert(x, ys))$</td>
<td></td>
</tr>
<tr>
<td><code>if(false, x, cons(y, ys))</code></td>
<td>$\rightarrow cons(x, cons(y, ys))$</td>
<td></td>
</tr>
<tr>
<td><code>gt(0, y)</code></td>
<td>$\Rightarrow false$</td>
<td></td>
</tr>
<tr>
<td><code>gt(s(x), 0)</code></td>
<td>$\Rightarrow true$</td>
<td></td>
</tr>
<tr>
<td><code>gt(s(x), s(y))</code></td>
<td>$\Rightarrow gt(x, y)$</td>
<td></td>
</tr>
</tbody>
</table>

\[c(x_1, ..., x_n) = 1 + x_1 + ... + x_n\] for constructors

Note: variables range over $\mathbb{N}$

Just $+$ and $\cdot$

Analyse result size for bottom-SCC (Strongly Connected Component) of call graph using standard ITS tools
Bird's Eye View of the Transformation

Example

\[
\text{isort}(xs', ys) \rightarrow ys \quad | \quad xs' = 1 \\
\text{isort}(xs', ys) \rightarrow \text{isort}(xs, \text{insert}(x, ys)) \quad | \quad xs' = 1 + x + xs \\
\text{insert}(x, \text{nil}) \rightarrow \text{cons}(x, \text{nil}) \\
\text{insert}(x, \text{cons}(y, ys)) \rightarrow \text{if}(\text{gt}(x, y), x, \text{cons}(y, ys)) \\
\text{if}(\text{true}, x, \text{cons}(y, ys)) \rightarrow \text{cons}(y, \text{insert}(x, ys)) \\
\text{if}(\text{false}, x, \text{cons}(y, ys)) \rightarrow \text{cons}(x, \text{cons}(y, ys)) \\
\text{gt}(0, y) \rightarrow \text{false} \\
\text{gt}(s(x), 0) \rightarrow \text{true} \\
\text{gt}(s(x), s(y)) \rightarrow \text{gt}(x, y)
\]
Example

\[
isort(xs', ys) \xrightarrow{1} ys \quad | \quad xs' = 1
\]
\[
isort(xs', ys) \xrightarrow{1} isort(xs, insert(x, ys)) \quad | \quad xs' = 1 + x + xs
\]
\[
insert(x, ys') \xrightarrow{1} 2 + x \quad | \quad ys' = 1
\]
\[
insert(x, cons(y, ys)) \rightarrow if(gt(x, y), x, cons(y, ys))
\]
\[
if(true, x, cons(y, ys)) \rightarrow cons(y, insert(x, ys))
\]
\[
if(false, x, cons(y, ys)) \rightarrow cons(x, cons(y, ys))
\]
\[
\begin{align*}
gt(0, y) & \rightarrow false \\
gt(s(x), 0) & \rightarrow true \\
gt(s(x), s(y)) & \rightarrow gt(x, y)
\end{align*}
\]

abstract terms to integers
Bird’s Eye View of the Transformation

**Example**

<table>
<thead>
<tr>
<th>Function</th>
<th>Rule</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{isort}(xs', ys)$</td>
<td>$\rightarrow ys$</td>
<td>$xs' = 1$</td>
</tr>
<tr>
<td>$\text{isort}(xs', ys)$</td>
<td>$\rightarrow \text{isort}(xs, \text{insert}(x, ys))$</td>
<td>$xs' = 1 + x + xs$</td>
</tr>
<tr>
<td>$\text{insert}(x, ys')$</td>
<td>$\rightarrow 2 + x$</td>
<td>$ys' = 1$</td>
</tr>
<tr>
<td>$\text{insert}(x, ys')$</td>
<td>$\rightarrow \text{if}(\text{gt}(x, y), x, ys')$</td>
<td>$ys' = 1 + y + ys$</td>
</tr>
<tr>
<td>$\text{if}(b, x, ys')$</td>
<td>$\rightarrow 1 + y + \text{insert}(x, ys)$</td>
<td>$b = 1 \land ys' = 1 + y + ys$</td>
</tr>
<tr>
<td>$\text{if}(b, x, ys')$</td>
<td>$\rightarrow 1 + ys'$</td>
<td>$b = 1 \land ys' = 1 + y + ys$</td>
</tr>
<tr>
<td>$\text{gt}(x', y')$</td>
<td>$\rightarrow 1$</td>
<td>$x' = 1$</td>
</tr>
<tr>
<td>$\text{gt}(x', y')$</td>
<td>$\rightarrow 1$</td>
<td>$x' = 1 + x \land y' = 1$</td>
</tr>
<tr>
<td>$\text{gt}(x', y')$</td>
<td>$\rightarrow \text{gt}(x, y)$</td>
<td>$x' = 1 + x \land y' = 1 + y$</td>
</tr>
</tbody>
</table>

1. abstract terms to integers
Bird’s Eye View of the Transformation

Example

\[
\begin{align*}
\text{isort}(x_s', y_s) & \xrightarrow{1} y_s & | & x_s' = 1 \\
\text{isort}(x_s', y_s) & \xrightarrow{1} \text{isort}(x_s, \text{insert}(x, y_s)) & | & x_s' = 1 + x + x_s \\
\text{insert}(x, y_s') & \xrightarrow{1} 2 + x & | & y_s' = 1 \\
\text{insert}(x, y_s') & \xrightarrow{1} \text{if}(\text{gt}(x, y), x, y_s') & | & y_s' = 1 + y + y_s \\
\text{if}(b, x, y_s') & \xrightarrow{1} 1 + y + \text{insert}(x, y_s) & | & b = 1 \land y_s' = 1 + y + y_s \\
\text{if}(b, x, y_s') & \xrightarrow{1} 1 + y_s' & | & b = 1 \land y_s' = 1 + y + y_s \\
\text{gt}(x', y') & \xrightarrow{0} 1 & | & x' = 1 \\
\text{gt}(x', y') & \xrightarrow{0} 1 & | & x' = 1 + x \land y' = 1 \\
\text{gt}(x', y') & \xrightarrow{0} \text{gt}(x, y) & | & x' = 1 + x \land y' = 1 + y
\end{align*}
\]

\[\text{abstract terms to integers}\]

- \([c](x_1, \ldots, x_n) = 1 + x_1 + \cdots + x_n\) for constructors \(c\)
- note: variables range over \(\mathbb{N}\)
- just + and ·
### Example

<table>
<thead>
<tr>
<th>Rule</th>
<th>Transformation</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>isort(xs', ys)</code></td>
<td>$\xrightarrow{1} ys$</td>
<td>$xs' = 1$</td>
</tr>
<tr>
<td><code>isort(xs', ys)</code></td>
<td>$\xrightarrow{1} isort(xs, insert(x, ys))$</td>
<td>$xs' = 1 + x + xs$</td>
</tr>
<tr>
<td><code>insert(x, ys')</code></td>
<td>$\xrightarrow{1} 2 + x$</td>
<td>$ys' = 1$</td>
</tr>
<tr>
<td><code>insert(x, ys')</code></td>
<td>$\xrightarrow{1} if(gt(x, y), x, ys')$</td>
<td>$ys' = 1 + y + ys$</td>
</tr>
<tr>
<td><code>if(b, x, ys')</code></td>
<td>$\xrightarrow{1} 1 + y + insert(x, ys)$</td>
<td>$b = 1 \land ys' = 1 + y + ys$</td>
</tr>
<tr>
<td><code>if(b, x, ys')</code></td>
<td>$\xrightarrow{1} 1 + ys'$</td>
<td>$b = 1 \land ys' = 1 + y + ys$</td>
</tr>
<tr>
<td><code>gt(x', y')</code></td>
<td>$\xrightarrow{0} 1$</td>
<td>$x' = 1$</td>
</tr>
<tr>
<td><code>gt(x', y')</code></td>
<td>$\xrightarrow{0} 1$</td>
<td>$x' = 1 + x \land y' = 1$</td>
</tr>
<tr>
<td><code>gt(x', y')</code></td>
<td>$\xrightarrow{0} gt(x, y)$</td>
<td>$x' = 1 + x \land y' = 1 + y$</td>
</tr>
</tbody>
</table>

1. **abstract terms to integers**
   - $[c](x_1, \ldots, x_n) = 1 + x_1 + \cdots + x_n$ for constructors $c$
   - note: variables range over $\mathbb{N}$
   - just $+$ and $\cdot$

2. **analyse result size for bottom-SCC (Strongly Connected Component)**
   of call graph using standard ITS tools
Call Graph & Bottom SCCs

- isort
- insert
- if
- gt
isort

insert

if

gt
### Example

<table>
<thead>
<tr>
<th>Rule</th>
<th>Description</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{isort}(xs', ys)$</td>
<td>$\frac{1}{\to} ys$</td>
<td>$xs' = 1$</td>
</tr>
<tr>
<td>$\text{isort}(xs', ys)$</td>
<td>$\frac{1}{\to} \text{isort}(xs, \text{insert}(x, ys))$</td>
<td>$xs' = 1 + x + xs$</td>
</tr>
<tr>
<td>$\text{insert}(x, ys')$</td>
<td>$\frac{1}{\to} 2 + x$</td>
<td>$ys' = 1$</td>
</tr>
<tr>
<td>$\text{insert}(x, ys')$</td>
<td>$\frac{1}{\to} \text{if}(\text{gt}(x, y), x, ys')$</td>
<td>$ys' = 1 + y + ys$</td>
</tr>
<tr>
<td>$\text{if}(b, x, ys')$</td>
<td>$\frac{1}{\to} 1 + y + \text{insert}(x, ys)$</td>
<td>$b = 1 \land ys' = 1 + y + ys$</td>
</tr>
<tr>
<td>$\text{if}(b, x, ys')$</td>
<td>$\frac{1}{\to} 1 + ys'$</td>
<td>$b = 1 \land ys' = 1 + y + ys$</td>
</tr>
<tr>
<td>$\text{gt}(x', y')$</td>
<td>$\frac{0}{\to} 1$</td>
<td>$x' = 1$</td>
</tr>
<tr>
<td>$\text{gt}(x', y')$</td>
<td>$\frac{0}{\to} 1$</td>
<td>$x' = 1 + x \land y' = 1$</td>
</tr>
<tr>
<td>$\text{gt}(x', y')$</td>
<td>$\frac{0}{\to} \text{gt}(x, y)$</td>
<td>$x' = 1 + x \land y' = 1 + y$</td>
</tr>
</tbody>
</table>

1. Abstract terms to integers
   - $[c](x_1, \ldots, x_n) = 1 + x_1 + \cdots + x_n$ for constructors $c$
   - Note: variables range over $\mathbb{N}$
   - Just $+$ and $\cdot$

2. Analyse result size for bottom-SCC using standard ITS tools
**Example**

\[
\begin{align*}
\text{isort}(xs', ys) & \overset{1}{\rightarrow} ys & | \quad xs' = 1 \\
\text{isort}(xs', ys) & \overset{1}{\rightarrow} \text{isort}(xs, \text{insert}(x, ys)) & | \quad xs' = 1 + x + xs \\
\text{insert}(x, ys') & \overset{1}{\rightarrow} 2 + x & | \quad ys' = 1 \\
\text{insert}(x, ys') & \overset{1}{\rightarrow} \text{if}(\text{gt}(x, y), x, ys') & | \quad ys' = 1 + y + ys \\
\text{if}(b, x, ys') & \overset{1}{\rightarrow} 1 + y + \text{insert}(x, ys) & | \quad b = 1 \land ys' = 1 + y + ys \\
\text{if}(b, x, ys') & \overset{1}{\rightarrow} 1 + ys' & | \quad b = 1 \land ys' = 1 + y + ys \\
\text{gt}(x', y') & \overset{0}{\rightarrow} 1 & | \quad x' = 1 \\
\text{gt}(x', y') & \overset{0}{\rightarrow} 1 & | \quad x' = 1 + x \land y' = 1 \\
\text{gt}(x', y') & \overset{0}{\rightarrow} \text{gt}(x, y) & | \quad x' = 1 + x \land y' = 1 + y 
\end{align*}
\]

1. **abstract terms to integers**
   - \([c](x_1, \ldots, x_n) = 1 + x_1 + \cdots + x_n\) for constructors \(c\)
   - note: variables range over \(\mathbb{N}\)
   - just \(+\) and \(\cdot\)

2. **analyse result size for bottom-SCC using standard ITS tools**
### Example

<table>
<thead>
<tr>
<th>Step</th>
<th>Expression</th>
<th>Condition</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( \text{isort}(xs', ys) )</td>
<td>( \rightarrow ys )</td>
<td>( xs' = 1 )</td>
</tr>
<tr>
<td>2</td>
<td>( \text{isort}(xs', ys) )</td>
<td>( \rightarrow \text{isort}(xs, \text{insert}(x, ys)) )</td>
<td>( xs' = 1 + x + xs )</td>
</tr>
<tr>
<td>3</td>
<td>( \text{insert}(x, ys') )</td>
<td>( \rightarrow 2 + x )</td>
<td>( ys' = 1 )</td>
</tr>
<tr>
<td>4</td>
<td>( \text{insert}(x, ys') )</td>
<td>( \rightarrow \text{if}(\text{gt}(x, y), x, ys') )</td>
<td>( ys' = 1 + y + ys )</td>
</tr>
<tr>
<td>5</td>
<td>( \text{if}(b, x, ys') )</td>
<td>( \rightarrow 1 + y + \text{insert}(x, ys) )</td>
<td>( b = 1 \land ys' = 1 + y + ys )</td>
</tr>
<tr>
<td>6</td>
<td>( \text{if}(b, x, ys') )</td>
<td>( \rightarrow 1 + ys' )</td>
<td>( b = 1 \land ys' = 1 + y + ys )</td>
</tr>
<tr>
<td>7</td>
<td>( \text{gt}(x', y') )</td>
<td>( \rightarrow 1 )</td>
<td>( x' = 1 )</td>
</tr>
<tr>
<td>8</td>
<td>( \text{gt}(x', y') )</td>
<td>( \rightarrow 1 )</td>
<td>( x' = 1 + x \land y' = 1 )</td>
</tr>
<tr>
<td>9</td>
<td>( \text{gt}(x', y') )</td>
<td>( \rightarrow \text{gt}(x, y) )</td>
<td>( x' = 1 + x \land y' = 1 + y )</td>
</tr>
</tbody>
</table>

1. **abstract terms to integers**
   - \( [c](x_1, \ldots, x_n) = 1 + x_1 + \cdots + x_n \) for constructors \( c \)
   - note: variables range over \( \mathbb{N} \)
   - just + and ⋅
2. **analyse result size for bottom-SCC using standard ITS tools**
3. **analyse runtime of bottom-SCC using standard ITS tools**
Example

\[
\begin{align*}
isort(xs', ys) & \rightarrow ys & | & xs' = 1 \\
isort(xs', ys) & \rightarrow isort(xs, insert(x, ys)) & | & xs' = 1 + x + xs \\
insert(x, ys') & \rightarrow 2 + x & | & ys' = 1 \\
insert(x, ys') & \rightarrow if(b, x, ys') & | & ys' = 1 + y + ys \land b \leq 1 \\
if(b, x, ys') & \rightarrow 1 + y + insert(x, ys) & | & b = 1 \land ys' = 1 + y + ys \\
if(b, x, ys') & \rightarrow 1 + ys' & | & b = 1 \land ys' = 1 + y + ys
\end{align*}
\]

1. abstract terms to integers
   - \([c](x_1, \ldots, x_n) = 1 + x_1 + \cdots + x_n\) for constructors \(c\)
   - note: variables range over \(\mathbb{N}\)
   - just + and ·

2. analyse result size for bottom-SCC using standard ITS tools

3. analyse runtime of bottom-SCC using standard ITS tools
Abstracting Terms to Integers: Pitfalls
## Terminating Variants

<table>
<thead>
<tr>
<th>Term Rewriting</th>
<th>Integer Transition Systems</th>
</tr>
</thead>
<tbody>
<tr>
<td>start terms may have variables</td>
<td>ground start terms only</td>
</tr>
</tbody>
</table>

### Example

\[
\begin{align*}
h(x) & \rightarrow f(g(x)) \\
f(x) & \rightarrow f(x) \\
g(a) & \rightarrow g(a)
\end{align*}
\]
Terminating Variants

<table>
<thead>
<tr>
<th>Term Rewriting</th>
<th>Integer Transition Systems</th>
</tr>
</thead>
<tbody>
<tr>
<td>start terms may have variables</td>
<td>ground start terms only</td>
</tr>
</tbody>
</table>

**Example**

\[
\begin{align*}
  h(x) & \rightarrow f(g(x)) & f(x) & \rightarrow f(x) & g(a) & \rightarrow g(a) \\
\text{innermost rewriting:} & & h(x) & \rightarrow f(g(x)) & \rightarrow f(g(x)) & \rightarrow \ldots
\end{align*}
\]
Terminating Variants

<table>
<thead>
<tr>
<th>Term Rewriting</th>
<th>Integer Transition Systems</th>
</tr>
</thead>
<tbody>
<tr>
<td>start terms may have variables</td>
<td>ground start terms only</td>
</tr>
</tbody>
</table>

Example:

\[
\begin{align*}
    h(x) &\rightarrow f(g(x)) & f(x) &\rightarrow f(x) & g(a) &\equiv g(a) \\
\text{innermost rewriting:} & h(x) &\rightarrow f(g(x)) &\rightarrow f(g(x)) &\rightarrow \ldots & \mathcal{O}(\infty)
\end{align*}
\]
Terminating Variants

<table>
<thead>
<tr>
<th>Term Rewriting</th>
<th>Integer Transition Systems</th>
</tr>
</thead>
<tbody>
<tr>
<td>start terms may have variables</td>
<td>ground start terms only</td>
</tr>
</tbody>
</table>

**Example**

- \( h(x) \rightarrow f(g(x)) \)  
- \( f(x) \rightarrow f(x) \)  
- \( g(a) \rightarrow g(a) \)

**innermost rewriting:**

- \( h(x) \rightarrow f(g(x)) \rightarrow f(g(x)) \rightarrow \ldots \)

Just ground rewriting?
Terminating Variants

<table>
<thead>
<tr>
<th>Term Rewriting</th>
<th>Integer Transition Systems</th>
</tr>
</thead>
<tbody>
<tr>
<td>start terms may have variables</td>
<td>ground start terms only</td>
</tr>
</tbody>
</table>

Example

\[
\begin{align*}
  h(x) & \rightarrow f(g(x)) \\
  f(x) & \rightarrow f(x) \\
  g(a) & \equiv g(a)
\end{align*}
\]

innermost rewriting:

\[
\begin{align*}
  h(x) & \rightarrow f(g(x)) \\
  & \rightarrow f(g(x)) \\
  & \rightarrow \ldots \\
  & \mathcal{O}(\infty)
\end{align*}
\]

ground rewriting:

\[
\begin{align*}
  h(a) & \rightarrow f(g(a)) \\
  & \equiv f(g(a)) \\
  & \equiv \ldots
\end{align*}
\]

- Just ground rewriting?
**Terminating Variants**

<table>
<thead>
<tr>
<th>Term Rewriting</th>
<th>Integer Transition Systems</th>
</tr>
</thead>
<tbody>
<tr>
<td>start terms may have variables</td>
<td>ground start terms only</td>
</tr>
</tbody>
</table>

**Example**

- Innermost rewriting: $h(x) \rightarrow f(g(x)) \rightarrow f(g(x)) \rightarrow \ldots \quad O(\infty)$
- Ground rewriting: $h(a) \rightarrow f(g(a)) \rightarrow f(g(a)) \rightarrow \ldots \quad O(1)$

- Just ground rewriting?
Terminating Variants

<table>
<thead>
<tr>
<th>Term Rewriting</th>
<th>Integer Transition Systems</th>
</tr>
</thead>
<tbody>
<tr>
<td>start terms may have variables</td>
<td>ground start terms only</td>
</tr>
</tbody>
</table>

**Example**

\[
\begin{align*}
  h(x) &\rightarrow f(g(x)) \\
  f(x) &\rightarrow f(x) \\
  g(a) &\rightarrow g(a)
\end{align*}
\]

**innermost rewriting:**

\[
\begin{align*}
  h(x) &\rightarrow f(g(x))\rightarrow f(g(x))\rightarrow \ldots \\
  O(\infty)
\end{align*}
\]

**ground rewriting:**

\[
\begin{align*}
  h(a) &\rightarrow f(g(a))\rightarrow f(g(a))\rightarrow \ldots \\
  O(1)
\end{align*}
\]

- Just ground rewriting?
- Add terminating variant of relative rules!
## Terminating Variants

<table>
<thead>
<tr>
<th>Term Rewriting</th>
<th>Integer Transition Systems</th>
</tr>
</thead>
<tbody>
<tr>
<td>start terms may have variables</td>
<td>ground start terms only</td>
</tr>
</tbody>
</table>

### Example

- **h(x) → f(g(x))**
- **f(x) → f(x)**
- **g(a) → g(a)**

**innermost rewriting:**

\[ h(x) \rightarrow f(g(x)) \rightarrow f(g(x)) \rightarrow \ldots \quad O(\infty) \]

**ground rewriting:**

\[ h(a) \rightarrow f(g(a)) \rightarrow f(g(a)) \rightarrow \ldots \quad O(1) \]

- Just ground rewriting?
- Add terminating variant of relative rules!

### Definition

\[ N \] is a terminating variant of \( S \) iff \( N \) terminates and every \( N \)-normal form is an \( S \)-normal form.
Terminating Variants

<table>
<thead>
<tr>
<th>Term Rewriting</th>
<th>Integer Transition Systems</th>
</tr>
</thead>
<tbody>
<tr>
<td>start terms may have variables</td>
<td>ground start terms only</td>
</tr>
</tbody>
</table>

Example

\[ h(x) \rightarrow f(g(x)) \quad f(x) \rightarrow f(x) \quad g(a) \rightarrow g(a) \quad g(a) \rightarrow a \]

innermost rewriting: \[ h(x) \rightarrow f(g(x)) \rightarrow f(g(x)) \rightarrow \ldots \quad O(\infty) \]

ground rewriting: \[ h(a) \rightarrow f(g(a)) \rightarrow f(g(a)) \rightarrow \ldots \quad O(1) \]

- Just ground rewriting?
- Add terminating variant of relative rules!

Definition

\( \mathcal{N} \) is a terminating variant of \( S \) iff \( \mathcal{N} \) terminates and every \( \mathcal{N} \)-normal form is an \( S \)-normal form.
## Terminating Variants

<table>
<thead>
<tr>
<th>Term Rewriting</th>
<th>Integer Transition Systems</th>
</tr>
</thead>
<tbody>
<tr>
<td>start terms may have variables</td>
<td>ground start terms only</td>
</tr>
</tbody>
</table>

### Example

- **h(x) → f(g(x))**
- **f(x) → f(x)**
- **g(a) → g(a)**
- **g(a) → a**

**innermost rewriting:**
- **h(x) → f(g(x)) → f(g(x)) → ...** \( \mathcal{O}(\infty) \)

**ground rewriting:**
- **h(a) → f(g(a)) → f(g(a)) → ...** \( \mathcal{O}(1) \)

**with terminating variant:**
- **h(a) → f(g(a)) → f(a) → f(a) → ...**

- Just ground rewriting?
- Add terminating variant of relative rules!

### Definition

\( \mathcal{N} \) is a terminating variant of \( S \) iff \( \mathcal{N} \) terminates and every \( \mathcal{N} \)-normal form is an \( S \)-normal form.
Terminating Variants

<table>
<thead>
<tr>
<th>Term Rewriting</th>
<th>Integer Transition Systems</th>
</tr>
</thead>
<tbody>
<tr>
<td>start terms may have variables</td>
<td>ground start terms only</td>
</tr>
</tbody>
</table>

**Example**

\[
\begin{align*}
    h(x) & \rightarrow f(g(x)) & f(x) & \rightarrow f(x) & g(a) & \rightarrow g(a) & g(a) & \rightarrow a \\
\text{innermost rewriting:} & h(x) & \rightarrow f(g(x)) & \rightarrow f(g(x)) & \rightarrow \ldots & \mathcal{O}(\infty) \\
\text{ground rewriting:} & h(a) & \rightarrow f(g(a)) & \rightarrow f(g(a)) & \rightarrow \ldots & \mathcal{O}(1) \\
\text{with terminating variant:} & h(a) & \rightarrow f(g(a)) & \rightarrow f(a) & \rightarrow f(a) & \rightarrow \ldots & \mathcal{O}(\infty)
\end{align*}
\]

- Just ground rewriting?
- Add terminating variant of relative rules!

**Definition**

\( \mathcal{N} \) is a terminating variant of \( S \) iff \( \mathcal{N} \) terminates and every \( \mathcal{N} \)-normal form is an \( S \)-normal form.
Ensuring Complete Definedness

<table>
<thead>
<tr>
<th>Term Rewriting</th>
<th>Integer Transition Systems</th>
</tr>
</thead>
<tbody>
<tr>
<td>arbitrary matchers</td>
<td>integer substitutions only</td>
</tr>
</tbody>
</table>

Example

\[ f(x) \rightarrow f(g(a)) \]
\[ g(b(a)) \rightarrow a \]

Definition

A TRS is completely defined iff its ground normal forms do not contain defined symbols.

TRS not completely defined?

\[ \rightarrow \]

Add suitable terminating variant!
Ensuring Complete Definedness

<table>
<thead>
<tr>
<th>Term Rewriting</th>
<th>Integer Transition Systems</th>
</tr>
</thead>
<tbody>
<tr>
<td>arbitrary matchers</td>
<td>integer substitutions only</td>
</tr>
</tbody>
</table>

Example

\[
\begin{align*}
  f(x) & \rightarrow f(g(a)) & g(b(a)) & \rightarrow a \\
\end{align*}
\]

original TRS: \[f(a) \rightarrow f(g(a)) \rightarrow f(g(a)) \rightarrow \ldots\]
Ensuring Complete Definedness

<table>
<thead>
<tr>
<th>Term Rewriting</th>
<th>Integer Transition Systems</th>
</tr>
</thead>
<tbody>
<tr>
<td>arbitrary matchers</td>
<td>integer substitutions only</td>
</tr>
</tbody>
</table>

Example

\[
\begin{align*}
f(x) & \rightarrow f(g(a)) \\
g(b(a)) & \rightarrow a
\end{align*}
\]

original TRS: \[
\begin{align*}
f(a) & \rightarrow f(g(a)) \\
f(g(a)) & \rightarrow f(g(a)) \\
& \cdots
\end{align*}
\]

\(O(\infty)\)
Ensuring Complete Definedness

<table>
<thead>
<tr>
<th>Term Rewriting</th>
<th>Integer Transition Systems</th>
</tr>
</thead>
<tbody>
<tr>
<td>arbitrary matchers</td>
<td>integer substitutions only</td>
</tr>
</tbody>
</table>

Example

\[ f(x) \rightarrow f(g(a)) \quad g(b(a)) \rightarrow a \]

original TRS: \[ f(a) \rightarrow f(g(a)) \rightarrow f(g(a)) \rightarrow \ldots \quad O(\infty) \]

resulting ITS: \[ f(1) \overset{1}{\rightarrow} f(g(1)) \]

Definition

A TRS is completely defined iff its ground normal forms do not contain defined symbols.

TRS not completely defined? \[ \rightarrow \]

Add suitable terminating variant!
## Ensuring Complete Definedness

### Term Rewriting vs. Integer Transition Systems

<table>
<thead>
<tr>
<th>Term Rewriting</th>
<th>Integer Transition Systems</th>
</tr>
</thead>
<tbody>
<tr>
<td>arbitrary matchers</td>
<td>integer substitutions only</td>
</tr>
</tbody>
</table>

### Example

Original TRS:

\[
\begin{align*}
    f(x) &\rightarrow f(g(a)) \\
    g(b(a)) &\rightarrow a \\
    f(a) &\rightarrow f(g(a)) \rightarrow f(g(a)) \rightarrow \ldots \quad O(\infty)
\end{align*}
\]

Resulting ITS:

\[
\begin{align*}
    f(1) &\rightarrow f(g(1)) \\
    f(1) &\rightarrow f(g(1)) \quad O(1)
\end{align*}
\]

**Definition**

A TRS is completely defined iff its ground normal forms do not contain defined symbols.

If a TRS is not completely defined, add suitable terminating variants!
Ensuring Complete Definedness

<table>
<thead>
<tr>
<th>Term Rewriting</th>
<th>Integer Transition Systems</th>
</tr>
</thead>
<tbody>
<tr>
<td>arbitrary matchers</td>
<td>integer substitutions only</td>
</tr>
</tbody>
</table>

**Example**

\[
f(x) \rightarrow f(g(a)) \quad g(b(a)) \rightarrow a
\]

**original TRS:**

\[
f(a) \rightarrow f(g(a)) \rightarrow f(g(a)) \rightarrow \ldots \quad O(\infty)
\]

**resulting ITS:**

\[
f(1) \xrightarrow{1} f(g(1)) \quad O(1)
\]

**Definition**

A TRS is completely defined iff its ground normal forms do not contain defined symbols.
Ensuring Complete Definedness

<table>
<thead>
<tr>
<th>Term Rewriting</th>
<th>Integer Transition Systems</th>
</tr>
</thead>
<tbody>
<tr>
<td>arbitrary matchers</td>
<td>integer substitutions only</td>
</tr>
</tbody>
</table>

**Example**

Original TRS:

\[
\begin{align*}
f(x) &\rightarrow f(g(a)) \\
g(b(a)) &\rightarrow a \\
g(x) &\rightarrow a
\end{align*}
\]

Resulting ITS:

\[
\begin{align*}
f(1) &\rightarrow f(g(1))
\end{align*}
\]

Definition

A TRS is completely defined iff its ground normal forms do not contain defined symbols.

TRS not completely defined? \(\sim\) Add suitable terminating variant!
Ensuring Complete Definedness

<table>
<thead>
<tr>
<th>Term Rewriting</th>
<th>Integer Transition Systems</th>
</tr>
</thead>
<tbody>
<tr>
<td>arbitrary matchers</td>
<td>integer substitutions only</td>
</tr>
</tbody>
</table>

Example

\[
\begin{align*}
\text{original TRS:} & \quad f(x) \rightarrow f(g(a)) & g(b(a)) \rightarrow a & \quad g(x) \rightarrow a \\
\text{resulting ITS:} & \quad f(a) \rightarrow f(g(a)) \rightarrow f(g(a)) \rightarrow \ldots & \quad O(\infty) \\
\text{ITS after completion:} & \quad f(1) \rightarrow f(g(1)) & \quad O(1)
\end{align*}
\]

Definition

A TRS is completely defined iff its ground normal forms do not contain defined symbols.

TRS not completely defined? \(\sim\) Add suitable terminating variant!
Ensuring Complete Definedness

<table>
<thead>
<tr>
<th>Term Rewriting</th>
<th>Integer Transition Systems</th>
</tr>
</thead>
<tbody>
<tr>
<td>arbitrary matchers</td>
<td>integer substitutions only</td>
</tr>
</tbody>
</table>

Example

\[ f(x) \rightarrow f(g(a)) \quad g(b(a)) \rightarrow a \quad g(x) \rightarrow a \]

original TRS: \[ f(a) \rightarrow f(g(a)) \rightarrow f(g(a)) \rightarrow \ldots \quad \mathcal{O}(\infty) \]

resulting ITS: \[ f(1) \xrightarrow{1} f(g(1)) \quad \mathcal{O}(1) \]

ITS after completion: \[ f(1) \xrightarrow{1} f(g(1)) \xrightarrow{0} f(1) \xrightarrow{1} f(g(1)) \xrightarrow{0} \ldots \quad \mathcal{O}(\infty) \]

Definition

A TRS is completely defined iff its ground normal forms do not contain defined symbols.

TRS not completely defined? \( \rightsquigarrow \) Add suitable terminating variant!
Ensuring Complete Definedness

<table>
<thead>
<tr>
<th>Term Rewriting</th>
<th>Integer Transition Systems</th>
</tr>
</thead>
<tbody>
<tr>
<td>arbitrary matchers</td>
<td>integer substitutions only</td>
</tr>
</tbody>
</table>

**Example**

\[
f(x) \rightarrow f(g(a)) \quad g(b(a)) \rightarrow a \quad g(x) \equiv a
\]

**original TRS:**
\[
f(a) \rightarrow f(g(a)) \rightarrow f(g(a)) \rightarrow \ldots \quad \mathcal{O}(\infty)
\]

**resulting ITS:**
\[
f(1) \xrightarrow{1} f(g(1)) \quad \mathcal{O}(1)
\]

**ITS after completion:**
\[
f(1) \xrightarrow{1} f(g(1)) \xrightarrow{0} f(1) \xrightarrow{1} f(g(1)) \xrightarrow{0} \ldots \quad \mathcal{O}(\infty)
\]

**Definition**

A TRS is completely defined iff its well-typed ground normal forms do not contain defined symbols.

TRS not completely defined? \( \bowtie \) Add suitable terminating variant!
Bird’s Eye View

Example

<table>
<thead>
<tr>
<th>Rule</th>
<th>Left-hand Side</th>
<th>Right-hand Side</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{isort}(xs', ys)$</td>
<td>$\rightarrow ys$</td>
<td>$xs' = 1$</td>
</tr>
<tr>
<td>$\text{isort}(xs', ys)$</td>
<td>$\rightarrow \text{isort}(xs, \text{insert}(x, ys))$</td>
<td>$xs' = 1 + x + xs$</td>
</tr>
<tr>
<td>$\text{insert}(x, ys')$</td>
<td>$\rightarrow 2 + x$</td>
<td>$ys' = 1$</td>
</tr>
<tr>
<td>$\text{insert}(x, ys')$</td>
<td>$\rightarrow \text{if}(b, x, ys')$</td>
<td>$ys' = 1 + y + ys \land b \leq 1$</td>
</tr>
<tr>
<td>$\text{if}(b, x, ys')$</td>
<td>$\rightarrow 1 + y + \text{insert}(x, ys)$</td>
<td>$b = 1 \land ys' = 1 + y + ys$</td>
</tr>
<tr>
<td>$\text{if}(b, x, ys')$</td>
<td>$\rightarrow 1 + ys'$</td>
<td>$b = 1 \land ys' = 1 + y + ys$</td>
</tr>
</tbody>
</table>

1. abstract terms to integers
2. analyse result size for bottom-SCC using standard ITS tools
3. analyse runtime of bottom-SCC using standard ITS tools
Call Graph & Bottom SCCs

Diagram:

- isort
- insert
- if

Arrows indicate the call graph and bottom SCCs.
Call Graph & Bottom SCCs

isort

insert

if
Example

<table>
<thead>
<tr>
<th></th>
<th>Rule</th>
<th>Invariant</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>isort(xs', ys)</code></td>
<td>1 → ys</td>
<td><code>xs' = 1</code></td>
</tr>
<tr>
<td><code>isort(xs', ys)</code></td>
<td>1 → <code>isort(xs, insert(x, ys))</code></td>
<td><code>xs' = 1 + x + xs</code></td>
</tr>
<tr>
<td><code>insert(x, ys' )</code></td>
<td>1 → 2 + x</td>
<td><code>ys' = 1</code></td>
</tr>
<tr>
<td><code>insert(x, ys' )</code></td>
<td>1 → <code>if(b, x, ys')</code></td>
<td><code>ys' = 1 + y + ys ∧ b ≤ 1</code></td>
</tr>
<tr>
<td><code>if(b, x, ys')</code></td>
<td>1 → 1 + y + <code>insert(x, ys)</code></td>
<td><code>b = 1 ∧ ys' = 1 + y + ys</code></td>
</tr>
<tr>
<td><code>if(b, x, ys')</code></td>
<td>1 → 1 + <code>ys'</code></td>
<td><code>b = 1 ∧ ys' = 1 + y + ys</code></td>
</tr>
</tbody>
</table>

1. abstract terms to integers
2. analyse result size for bottom-SCC using standard ITS tools
3. analyse runtime of bottom-SCC using standard ITS tools
Example

\[
\begin{align*}
\text{isort}(xs', ys) & \rightarrow ys & | & xs' = 1 \\
\text{isort}(xs', ys) & \rightarrow \text{isort}(xs, \text{insert}(x, ys)) & | & xs' = 1 + x + xs \\
\text{insert}(x, ys') & \rightarrow 2 + x & | & ys' = 1 \\
\text{insert}(x, ys') & \rightarrow \text{if}(b, x, ys') & | & ys' = 1 + y + ys \land b \leq 1 \\
\text{if}(b, x, ys') & \rightarrow 1 + y + \text{insert}(x, ys) & | & b = 1 \land ys' = 1 + y + ys \\
\text{if}(b, x, ys') & \rightarrow 1 + ys' & | & b = 1 \land ys' = 1 + y + ys
\end{align*}
\]

1 abstract terms to integers
2 analyse result size for bottom-SCC using standard ITS tools
3 analyse runtime of bottom-SCC using standard ITS tools
Analyse Size Using Standard ITS Tools
Idea: time bound for `insert` in transformed rules gives size bound for `insert` in original rules

Example

<table>
<thead>
<tr>
<th>Rule</th>
<th>Term</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>insert(x, ys')</code></td>
<td>$\xrightarrow{1} 2 + x$</td>
<td>$ys' = 1$</td>
</tr>
<tr>
<td><code>insert(x, ys')</code></td>
<td>$\xrightarrow{1} \text{if}(b, x, ys')$</td>
<td>$ys' = 1 + y + ys \land b \leq 1$</td>
</tr>
<tr>
<td><code>if(b, x, ys')</code></td>
<td>$\xrightarrow{1} 1 + y + \text{insert}(x, ys)$</td>
<td>$b = 1 \land ys' = 1 + y + ys$</td>
</tr>
<tr>
<td><code>if(b, x, ys')</code></td>
<td>$\xrightarrow{1} 1 + ys'$</td>
<td>$b = 1 \land ys' = 1 + y + ys$</td>
</tr>
</tbody>
</table>
Using Runtime Analysis to Compute Size Bounds

**Idea:** time bound for `insert` in transformed rules gives size bound for `insert` in original rules

**Example**

<table>
<thead>
<tr>
<th>Rule</th>
<th>Transformation</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>insert(x, ys')</code></td>
<td>$\xrightarrow{1} 2 + x$</td>
<td>$ys' = 1$</td>
</tr>
<tr>
<td><code>insert(x, ys')</code></td>
<td>$\xrightarrow{1}$ <code>if (b, x, ys')</code></td>
<td>$ys' = 1 + y + ys \land b \leq 1$</td>
</tr>
<tr>
<td><code>if (b, x, ys')</code></td>
<td>$\xrightarrow{1} 1 + y + insert(x, ys)$</td>
<td>$b = 1 \land ys' = 1 + y + ys$</td>
</tr>
<tr>
<td><code>if (b, x, ys')</code></td>
<td>$\xrightarrow{1} 1 + ys'$</td>
<td>$b = 1 \land ys' = 1 + y + ys$</td>
</tr>
</tbody>
</table>

**Idea:** move “integer context” to weights
Idea: time bound for `insert` in transformed rules gives size bound for `insert` in original rules

Example

\[
\begin{align*}
\text{insert}(x, ys') & \xrightarrow{2+x} 2 + x & | & ys' = 1 \\
\text{insert}(x, ys') & \xrightarrow{1} \text{if}(b, x, ys') & | & ys' = 1 + y + ys \land b \leq 1 \\
\text{if}(b, x, ys') & \xrightarrow{1} 1 + y + \text{insert}(x, ys) & | & b = 1 \land ys' = 1 + y + ys \\
\text{if}(b, x, ys') & \xrightarrow{1} 1 + ys' & | & b = 1 \land ys' = 1 + y + ys
\end{align*}
\]

Idea: move “integer context” to weights
**Idea:** time bound for `insert` in transformed rules gives size bound for `insert` in original rules

### Example

<table>
<thead>
<tr>
<th>Rule</th>
<th>Transformation</th>
<th>Size Bound</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>insert(x, ys')</code></td>
<td>$\frac{2+x}{2}$</td>
<td>$2 + x$</td>
<td>$ys' = 1$</td>
</tr>
<tr>
<td><code>insert(x, ys')</code></td>
<td>$0$</td>
<td><code>if (b, x, ys')</code></td>
<td>$ys' = 1 + y + ys \land b \leq 1$</td>
</tr>
<tr>
<td><code>if (b, x, ys')</code></td>
<td>$1$</td>
<td>$1 + y + insert(x, ys)$</td>
<td>$b = 1 \land ys' = 1 + y + ys$</td>
</tr>
<tr>
<td><code>if (b, x, ys')</code></td>
<td>$1$</td>
<td>$1 + ys'$</td>
<td>$b = 1 \land ys' = 1 + y + ys$</td>
</tr>
</tbody>
</table>

**Idea:** move “integer context” to weights
Using Runtime Analysis to Compute Size Bounds

**Idea:** time bound for `insert` in transformed rules gives size bound for `insert` in original rules

### Example

<table>
<thead>
<tr>
<th>Rule</th>
<th>Transformation</th>
<th>Expression</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>insert(x, ys')</code></td>
<td>$\frac{2+x}{\rightarrow}$</td>
<td>$2 + x$</td>
<td>$ys' = 1$</td>
</tr>
<tr>
<td><code>insert(x, ys')</code></td>
<td>$\frac{0}{\rightarrow}$</td>
<td><code>if(b, x, ys')</code></td>
<td>$ys' = 1 + y + ys \land b \leq 1$</td>
</tr>
<tr>
<td><code>if(b, x, ys')</code></td>
<td>$\frac{1+y}{\rightarrow}$</td>
<td>$1 + y + \text{insert}(x, ys)$</td>
<td>$b = 1 \land ys' = 1 + y + ys$</td>
</tr>
<tr>
<td><code>if(b, x, ys')</code></td>
<td>$\frac{1}{\rightarrow}$</td>
<td>$1 + ys'$</td>
<td>$b = 1 \land ys' = 1 + y + ys$</td>
</tr>
</tbody>
</table>

**Idea:** move “integer context” to weights
Using Runtime Analysis to Compute Size Bounds

**Idea:** time bound for \text{insert} in transformed rules gives size bound for \text{insert} in original rules

**Example**

\[
\begin{align*}
\text{insert}(x, ys') & \xrightarrow{2+x} 2 + x & | & ys' = 1 \\
\text{insert}(x, ys') & \xrightarrow{0} \text{if}(b, x, ys') & | & ys' = 1 + y + ys \land b \leq 1 \\
\text{if}(b, x, ys') & \xrightarrow{1+y} 1 + y + \text{insert}(x, ys) & | & b = 1 \land ys' = 1 + y + ys \\
\text{if}(b, x, ys') & \xrightarrow{1+y's'} 1 + y's' & | & b = 1 \land ys' = 1 + y + ys
\end{align*}
\]

**Idea:** move “integer context” to weights
Using Runtime Analysis to Compute Size Bounds

Idea: time bound for *insert* in transformed rules gives size bound for *insert* in original rules

Example

<table>
<thead>
<tr>
<th>Rule</th>
<th>Time Bound</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{insert}(x, ys') \rightarrow 2 + x )</td>
<td>( 2 + x )</td>
<td>( ys' = 1 )</td>
</tr>
<tr>
<td>( \text{insert}(x, ys') \rightarrow 0 )</td>
<td>( \text{if}(b, x, ys') )</td>
<td>( ys' = 1 + y + ys \land b \leq 1 )</td>
</tr>
<tr>
<td>( \text{if}(b, x, ys') \rightarrow 1 + y )</td>
<td>( 1 + y + \text{insert}(x, ys) )</td>
<td>( b = 1 \land ys' = 1 + y + ys )</td>
</tr>
<tr>
<td>( \text{if}(b, x, ys') \rightarrow 1 + ys' )</td>
<td>( 1 + ys' )</td>
<td>( b = 1 \land ys' = 1 + y + ys )</td>
</tr>
</tbody>
</table>

Idea: move “integer context” to weights \( \bowtie \text{sz}(\text{insert}(x, ys')) \leq 1 + x + ys' \)
Using Runtime Analysis to Compute Size Bounds

**Idea:** time bound for `insert` in transformed rules gives size bound for `insert` in original rules

**Example**

<table>
<thead>
<tr>
<th>Rule</th>
<th>Transformation</th>
<th>Result</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>insert(x, ys')</code></td>
<td>2+x</td>
<td>2+x</td>
<td><code>ys' = 1</code></td>
</tr>
<tr>
<td><code>insert(x, ys')</code></td>
<td>0</td>
<td><code>if(b, x, ys')</code></td>
<td><code>ys' = 1 + y + ys \land b \leq 1</code></td>
</tr>
<tr>
<td><code>if(b, x, ys')</code></td>
<td>1+y</td>
<td><code>1 + y + insert(x, ys)</code></td>
<td><code>b = 1 \land ys' = 1 + y + ys</code></td>
</tr>
<tr>
<td><code>if(b, x, ys')</code></td>
<td>1+ys'</td>
<td><code>1 + ys'</code></td>
<td><code>b = 1 \land ys' = 1 + y + ys</code></td>
</tr>
</tbody>
</table>

**Idea:** move “integer context” to weights \( \bowtie \text{sz}(\text{insert}(x, ys')) \leq 1 + x + ys' \)

**Example**

<table>
<thead>
<tr>
<th>Rule</th>
<th>Transformation</th>
<th>Expression</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>f(x)</code></td>
<td>1</td>
<td><code>2 + x \cdot f(x - 1)</code></td>
<td>( x &gt; 0 )</td>
</tr>
</tbody>
</table>
Using Runtime Analysis to Compute Size Bounds

**Idea:** time bound for `insert` in transformed rules gives size bound for `insert` in original rules

**Example**

\[
\begin{align*}
\text{insert}(x, ys') & \xrightarrow{2+x} 2 + x \quad | \quad ys' = 1 \\
\text{insert}(x, ys') & \xrightarrow{0} \text{if}(b, x, ys') \quad | \quad ys' = 1 + y + ys \land b \leq 1 \\
\text{if}(b, x, ys') & \xrightarrow{1+y} 1 + y + \text{insert}(x, ys) \quad | \quad b = 1 \land ys' = 1 + y + ys \\
\text{if}(b, x, ys') & \xrightarrow{1+ys'} 1 + ys' \quad | \quad b = 1 \land ys' = 1 + y + ys
\end{align*}
\]

**Idea:** move “integer context” to weights \( \bowtie \text{sz}(\text{insert}(x, ys')) \leq 1 + x + ys' \)

**Example**

\[
\begin{align*}
f(x) & \xrightarrow{1} 2 + x \cdot f(x - 1) \quad | \quad x > 0
\end{align*}
\]

**Idea:** use accumulator
Using Runtime Analysis to Compute Size Bounds

**Idea:** time bound for insert in transformed rules gives size bound for insert in original rules

### Example

<table>
<thead>
<tr>
<th>Rule</th>
<th>Transformation</th>
<th>Size Bound</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>insert(x, ys')</code></td>
<td><code>2 + x</code></td>
<td><code>2 + x</code></td>
<td><code>ys' = 1</code></td>
</tr>
<tr>
<td><code>insert(x, ys')</code></td>
<td><code>0</code></td>
<td><code>if(b, x, ys')</code></td>
<td><code>ys' = 1 + y + ys \land b \leq 1</code></td>
</tr>
<tr>
<td><code>if(b, x, ys')</code></td>
<td><code>1 + y</code></td>
<td><code>1 + y + insert(x, ys)</code></td>
<td><code>b = 1 \land ys' = 1 + y + ys</code></td>
</tr>
<tr>
<td><code>if(b, x, ys')</code></td>
<td><code>1 + ys'</code></td>
<td><code>1 + ys'</code></td>
<td><code>b = 1 \land ys' = 1 + y + ys</code></td>
</tr>
</tbody>
</table>

**Idea:** move “integer context” to weights ∼ `sz(insert(x, ys')) ≤ 1 + x + ys'`

### Example

<table>
<thead>
<tr>
<th>Rule</th>
<th>Transformation</th>
<th>Size Bound</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>f(x)</code></td>
<td><code>1</code></td>
<td><code>2 + x \cdot f(x - 1)</code></td>
<td><code>x &gt; 0</code></td>
</tr>
<tr>
<td><code>f(x, acc)</code></td>
<td><code>acc \cdot 2</code></td>
<td><code>2 + x \cdot f(x - 1, acc \cdot x)</code></td>
<td><code>x &gt; 0</code></td>
</tr>
</tbody>
</table>

**Idea:** use accumulator
Example

\[
\begin{align*}
\text{isort}(xs', ys) & \xrightarrow{1} ys & | & xs' = 1 \\
\text{isort}(xs', ys) & \xrightarrow{1} \text{isort}(xs, \text{insert}(x, ys)) & | & xs' = 1 + x + xs \\
\text{insert}(x, ys') & \xrightarrow{1} 2 + x & | & ys' = 1 \\
\text{insert}(x, ys') & \xrightarrow{1} \text{if}(b, x, ys') & | & ys' = 1 + y + ys \land b \leq 1 \\
\text{if}(b, x, ys') & \xrightarrow{1} 1 + y + \text{insert}(x, ys) & | & b = 1 \land ys' = 1 + y + ys \\
\text{if}(b, x, ys') & \xrightarrow{1} 1 + ys' & | & b = 1 \land ys' = 1 + y + ys
\end{align*}
\]

1. abstract terms to integers
2. analyse result size for bottom-SCC using standard ITS tools
3. analyse runtime of bottom-SCC using standard ITS tools
Bird’s Eye View

Example

\[ \text{isort}(xs', ys) \xrightarrow{1} ys \quad | \quad xs' = 1 \]
\[ \text{isort}(xs', ys) \xrightarrow{1} \text{isort}(xs, \text{insert}(x, ys)) \quad | \quad xs' = 1 + x + xs \]

1. abstract terms to integers
2. analyse result size for bottom-SCC using standard ITS tools
3. analyse runtime of bottom-SCC using standard ITS tools
Analyse Runtime Using Standard Tools
Removing Nested Function Calls

Example

\[ \text{isort}(xs', ys) \xrightarrow{1} ys \quad | \quad xs' = 1 \]
\[ \text{isort}(xs', ys) \xrightarrow{1} \text{isort}(xs, \text{insert}(x, ys)) \quad | \quad xs' = 1 + x + xs \]

- \( \text{sz}(\text{insert}(x, ys)) \leq 1 + x + ys \)
- \( \text{rt}(\text{insert}(x, ys)) \leq 2 \cdot ys \)
## Removing Nested Function Calls

### Example

\[
\begin{align*}
\text{isort}(xs', ys) & \xrightarrow{1} ys & \text{if } xs' = 1 \\
\text{isort}(xs', ys) & \xrightarrow{1} \text{isort}(xs, \text{insert}(x, ys)) & \text{if } xs' = 1 + x + xs
\end{align*}
\]

- \(\text{sz(insert}(x, ys)) \leq 1 + x + ys\)
- \(\text{rt(insert}(x, ys)) \leq 2 \cdot ys\)
- add costs of nested function call
Removing Nested Function Calls

Example

\[
\begin{align*}
\text{isort}(x_s', y_s) & \xrightarrow{1} y_s & | & x_s' = 1 \\
\text{isort}(x_s', y_s) & \xrightarrow{1 + 2 \cdot y_s} \text{isort}(x_s, \text{insert}(x, y_s)) & | & x_s' = 1 + x + x_s
\end{align*}
\]

- \( sz(\text{insert}(x, y_s)) \leq 1 + x + y_s \)
- \( rt(\text{insert}(x, y_s)) \leq 2 \cdot y_s \)
- add costs of nested function call
Removing Nested Function Calls

Example

\[
\begin{align*}
\text{isort}(xs', ys) &\xrightarrow{1} ys & | & xs' = 1 \\
\text{isort}(xs', ys) &\xrightarrow{1 + 2 \cdot ys} \text{isort}(xs, \text{insert}(x, ys)) & | & xs' = 1 + x + xs
\end{align*}
\]

- \(\text{sz(\text{insert}(x, ys))} \leq 1 + x + ys\)
- \(\text{rt(\text{insert}(x, ys))} \leq 2 \cdot ys\)
- add costs of nested function call
- replace nested function call by fresh variable \(x_f\)
Removing Nested Function Calls

Example

\[
\begin{align*}
\text{isort}(xs', ys) & \xrightarrow{1} ys \\
\text{isort}(xs', ys) & \xrightarrow{1+2\cdot ys} \text{isort}(xs, xf) \\
\end{align*}
\]

\[
\begin{align*}
xs' & = 1 \\
xsf & = 1 + x + xs
\end{align*}
\]

- \( \text{sz}(\text{insert}(x, ys)) \leq 1 + x + ys \)
- \( \text{rt}(\text{insert}(x, ys)) \leq 2 \cdot ys \)
- add costs of nested function call
- replace nested function call by fresh variable \( xsf \)
Removing Nested Function Calls

Example

\[
\text{isort}(xs', ys) \xrightarrow{1} ys \\
\text{isort}(xs', ys) \xrightarrow{1+2\cdot ys} \text{isort}(xs, xf) \\
\text{sz} \left( \text{insert} \left( x, ys \right) \right) \leq 1 + x + ys \\
\text{rt} \left( \text{insert} \left( x, ys \right) \right) \leq 2 \cdot ys \\
\text{add costs of nested function call} \\
\text{replace nested function call by fresh variable } xf \\
\text{add constraint } \left( xf \leq \text{size bound} \right)
\]
Removing Nested Function Calls

Example

\[
isort(x', y) \quad \xrightarrow{1} \quad y \\
isort(x', y) \quad \xrightarrow{1+2\cdot y} \quad isort(x, x_f) \\
\]

\[
\text{sz}(\text{insert}(x, y)) \leq 1 + x + y \\
\text{rt}(\text{insert}(x, y)) \leq 2 \cdot y \\
\text{add costs of nested function call} \\
\text{replace nested function call by fresh variable } x_f \\
\text{add constraint } x_f \leq \text{size bound}
\]
Removing Nested Function Calls

**Example**

\[
\begin{align*}
\text{isort}(x_s', y_s) & \xrightarrow{1} y_s & | & x_s' = 1 \\
isort(x_s', y_s) & \xrightarrow{1 + 2 \cdot y_s} \text{isort}(x_s, x_f) & | & x_s' = 1 + x + x_s \land x_f \leq 1 + x + y_s
\end{align*}
\]

- \(\text{sz}(\text{insert}(x, y_s)) \leq 1 + x + y_s\)
- \(\text{rt}(\text{insert}(x, y_s)) \leq 2 \cdot y_s\)
- add costs of nested function call
- replace nested function call by fresh variable \(x_f\)
- add constraint \("x_f \leq \text{size bound}"\)
- \(\text{rt}(\text{isort}(x_s', y_s)) \leq O(x_s'^2 + x_s' \cdot y_s)\)
Removing Nested Function Calls

Example

\[
\begin{align*}
\text{isort}(x_s', y_s) & \xrightarrow{1} y_s & | & x_s' = 1 \\
\text{isort}(x_s', y_s) & \xrightarrow{1+2\cdot y_s} \text{isort}(x_s, x_f) & | & x_s' = 1 + x + x_s \land x_f \leq 1 + x + y_s
\end{align*}
\]

- \(\text{sz}(\text{insert}(x, y_s)) \leq 1 + x + y_s\)
- \(\text{rt}(\text{insert}(x, y_s)) \leq 2 \cdot y_s\)
- add costs of nested function call
- replace nested function call by fresh variable \(x_f\)
- add constraint “\(x_f \leq \text{size bound}\)”
- \(\text{rt}(\text{isort}(x_s', y_s)) \leq O(x_s'^2 + x_s' \cdot y_s)\)
- similar techniques to eliminate outer function calls
Removing Nested Function Calls

Example

\[
\begin{align*}
isort(xs', ys) & \xrightarrow{1} ys & | & xs' = 1 \\
isort(xs', ys) & \xrightarrow{1+2\cdot ys} isort(xs, xf) & | & xs' = 1 + x + xs \land xf \leq 1 + x + ys
\end{align*}
\]

- \(\text{sz}(\text{insert}(x, ys)) \leq 1 + x + ys\)
- \(\text{rt}(\text{insert}(x, ys)) \leq 2 \cdot ys\)
- Add costs of nested function call
- Replace nested function call by fresh variable \(xf\)
- Add constraint \(xf \leq \text{size bound}\)
- \(\text{rt}(\text{isort}(xs', ys)) \leq O(xs'^2 + xs' \cdot ys)\)
- Similar techniques to eliminate outer function calls

\[
\text{times}(s(x), y) \rightarrow \text{plus}(\text{times}(x, y), y)
\]
Removing Nested Function Calls

**Example**

\[
\begin{align*}
\text{isort}(xs', ys) & \xrightarrow{1} ys & | & xs' = 1 \\
\text{isort}(xs', ys) & \xrightarrow{1+2\cdot ys} \text{isort}(xs, xf) & | & xs' = 1 + x + xs \land xf \leq 1 + x + ys
\end{align*}
\]

- \( \text{sz}(\text{insert}(x, ys)) \leq 1 + x + ys \)
- \( \text{rt}(\text{insert}(x, ys)) \leq 2 \cdot ys \)
- add costs of nested function call
- replace nested function call by fresh variable \( xf \)
- add constraint “\( xf \leq \text{size bound} \)”
- \( \text{rt}(\text{isort}(xs', ys)) \leq O(xs'^2 + xs' \cdot ys) \)
- similar techniques to eliminate outer function calls \( \xrightarrow{} \) see paper!

\[
\text{times}(s(x), y) \rightarrow \text{plus}(\text{times}(x, y), y)
\]
Experiments

ITS tools CoFloCo, KoAT, and PUBS used as backends.

AProVE + ITS backend finds better bounds than AProVE & TcT for 127 TRSs, which can be a useful additional inference technique for upper bounds.
Experiments

ITS tools CoFloCo, KoAT, and PUBS used as backends.

Results on the TPDB (922 examples):
Experiments

ITS tools CoFloCo, KoAT, and PUBS used as backends.

Results on the TPDB (922 examples):

- AProVE + ITS backend finds better bounds than AProVE & TcT for 127 TRSs
- transformation a useful additional inference technique for upper bounds
From irc of TRSs to Integer Transition Systems: Summary

- Abstraction from terms to integers
- Modular bottom-up approach using standard ITS tools
- Approach complements and improves state of the art
- Note: abstraction **hard-coded** to term size

⇒ Future work: more flexible approach?
\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*}
<table>
<thead>
<tr>
<th>Rule</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>app(nil, y) → y</code></td>
<td></td>
</tr>
<tr>
<td><code>app(add(n, x), y) → add(n, app(x, y))</code></td>
<td></td>
</tr>
<tr>
<td><code>reverse(nil) → nil</code></td>
<td></td>
</tr>
<tr>
<td><code>reverse(add(n, x)) → app(reverse(x), add(n, nil))</code></td>
<td></td>
</tr>
<tr>
<td><code>shuffle(nil) → nil</code></td>
<td></td>
</tr>
<tr>
<td><code>shuffle(add(n, x)) → add(n, shuffle(reverse(x)))</code></td>
<td></td>
</tr>
</tbody>
</table>

AProVE finds (tight) upper bound $O(n^4)$ for $\text{dc}_R$: 
AProVE finds (tight) upper bound $\mathcal{O}(n^4)$ for $\text{dc}_R$:

1. Add generator rules $\mathcal{G}$, so analyse $\text{rc}_{R/G}$ instead (FroCoS'19)
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)
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$_\mathcal{R}$:

1. Add generator rules $\mathcal{G}$, so analyse rc$_{\mathcal{R}/\mathcal{G}}$ instead (FroCoS’19)
2. Detect: innermost is worst case here, analyse irc$_{\mathcal{R}/\mathcal{G}}$ 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 $\text{dc}_R$:

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

\[
\begin{align*}
\text{app}(\text{nil}, y) & \rightarrow y & \text{app}(\text{add}(n, x), y) & \rightarrow \text{add}(n, \text{app}(x, y)) \\
\text{reverse}(\text{nil}) & \rightarrow \text{nil} & \text{reverse}(\text{add}(n, x)) & \rightarrow \text{app}(\text{reverse}(x), \text{add}(n, \text{nil})) \\
\text{shuffle}(\text{nil}) & \rightarrow \text{nil} & \text{shuffle}(\text{add}(n, x)) & \rightarrow \text{add}(n, \text{shuffle}(\text{reverse}(x))))
\end{align*}
\]
app(nil, y) → y
reverse(nil) → nil
shuffle(nil) → nil

app(add(n, x), y) → add(n, app(x, y))
reverse(add(n, x)) → app(reverse(x), add(n, nil))
shuffle(add(n, x)) → add(n, shuffle(reverse(x)))

AProVE finds (tight) upper bound $\mathcal{O}(n^4)$ 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 $\mathcal{O}(n^4)$ for RITS complexity carries over to $dc_R$ of input!

AProVE finds lower bound $\Omega(n^3)$ for $dc_R$ using induction technique.
Automated tools at the Termination and Complexity Competition 2021:

- AProVE: https://aprove.informatik.rwth-aachen.de/
- 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 at the Termination and Complexity Competition 2021:

- 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/

\[\text{(VAR x y)}\]
\[\text{(GOAL COMPLEXITY)}\]
\[\text{(STARTTERM CONSTRUCTOR-BASED)}\]
\[\text{(RULES}\]
\[\text{plus(0, y) \rightarrow y}\]
\[\text{plus(s(x), y) \rightarrow s(plus(x, y))}\]

\[^{41}\text{For TcT Web, use only VAR and RULES entries in the text format and configure other aspects (e.g., start terms) in the web interface.}\]
Automated tools at the Termination and Complexity Competition 2021:

- 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/

Input format for runtime complexity:\(^{41}\)

(VAR x y)
(GOAL COMPLEXITY)
(STARTTERM CONSTRUCTOR-BASED)
(RULES
  plus(0, y) -> y
  plus(s(x), y) -> s(plus(x, y))
)

\(^{41}\)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))
)

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))
)
Problem noted in the early Termination Competitions:

- Tools may give contradictory answers on some (few) inputs.
Problem noted in the early Termination Competitions:

- Tools may give contradictory answers on some (few) inputs.
- Also program analysis tools may have bugs! But verifying tool correctness seems infeasible.
Problem noted in the early Termination Competitions:
- Tools may give contradictory answers on some (few) inputs.
- Also program analysis tools may have bugs! But verifying tool correctness seems infeasible.

Solution for termination and complexity of TRSs:
- Proof output by TRS tools in a standard (XML) format
- Proof certifiers based on trusted proof assistants (Isabelle/HOL, Coq, ...) check proofs independently
Problem noted in the early Termination Competitions:

- Tools may give contradictory answers on some (few) inputs.
- Also program analysis tools may have bugs! But verifying tool correctness seems infeasible.

Solution for termination and complexity of TRSs:

- Proof output by TRS tools in a standard (XML) format
- Proof certifiers based on trusted proof assistants (Isabelle/HOL, Coq, ...) check proofs independently
- Example for TRS complexity: IsaFoR with certifier CeTA\textsuperscript{42}

idc, irc: like dc, rc, but for innermost rewriting

TRS

FroCoS'19

dc \rightarrow rc

FroCoS'19

idc \rightarrow irc

LPAR'17

Rec. ITS irc

FroCoS'17

ITS irc

FroCoS'17
idc, irc: like dc, rc, but for *innermost rewriting*

OCaml

Java

Prolog

TRS
idc, irc: like dc, rc, but for *innermost* rewriting

---


44 G. Moser, M. Schaper: *From Jinja bytecode to term rewriting: A complexity reflecting transformation*, IC ’18

45 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: \( \text{map}(F, xs) \)
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

⇒ First-order TRS \( \mathcal{R} \) with \( \text{rc}_\mathcal{R}(n) \) 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\(^{46}\))
- Deal with language specifics in program analysis
- Extract TRS \( \mathcal{R} \) such that \( rc_\mathcal{R}(n) \) 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!

\(^{46}\) P. Cousot, R. Cousot: Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints, POPL '77
amortised complexity analysis for term rewriting\textsuperscript{47}

\textsuperscript{47} G. Moser, M. Schneckenreither: \textit{Automated amortised resource analysis for term rewrite systems}, SCP '20
Current Developments

- **amortised** complexity analysis for term rewriting\(^{47}\)
- **probabilistic** term rewriting $\rightarrow$ upper bounds on **expected runtime**\(^{48}\)

---

\(^{47}\) G. Moser, M. Schneckenreither: *Automated amortised resource analysis for term rewrite systems*, SCP '20

\(^{48}\) M. Avanzini, U. Dal Lago, A. Yamada: *On probabilistic term rewriting*, SCP '20
Current Developments

- **amortised** complexity analysis for term rewriting\(^{47}\)
- **probabilistic** term rewriting $\rightarrow$ upper bounds on expected runtime\(^ {48}\)
- complexity analysis for **logically constrained rewriting** with built-in data types from SMT theories (integers, booleans, arrays, ... )\(^ {49}\)

---

\(^{47}\) G. Moser, M. Schneckenreither: *Automated amortised resource analysis for term rewrite systems*, SCP '20

\(^{48}\) M. Avanzini, U. Dal Lago, A. Yamada: *On probabilistic term rewriting*, SCP '20

\(^{49}\) S. Winkler, G. Moser: *Runtime complexity analysis of logically constrained rewriting*, LOPSTR '20
Current Developments

- **amortised** complexity analysis for term rewriting\(^{47}\)
- **probabilistic** term rewriting \(\rightarrow\) upper bounds on expected runtime\(^{48}\)
- complexity analysis for **logically constrained rewriting** with built-in data types from SMT theories (integers, booleans, arrays, . . . )\(^{49}\)
- direct analysis of complexity for **higher-order term rewriting**\(^{50}\)

---

\(^{47}\) G. Moser, M. Schneckenreither: *Automated amortised resource analysis for term rewrite systems*, SCP ’20


\(^{49}\) S. Winkler, G. Moser: *Runtime complexity analysis of logically constrained rewriting*, LOPSTR ’20

\(^{50}\) C. Kop, D. Vale: *Tuple interpretations for higher-order rewriting*, FSCD ’21
Current Developments

- **amortised** complexity analysis for term rewriting\(^{47}\)
- **probabilistic** term rewriting $\rightarrow$ upper bounds on expected runtime\(^{48}\)
- complexity analysis for **logically constrained rewriting** with built-in data types from SMT theories (integers, booleans, arrays, ... )\(^{49}\)
- direct analysis of complexity for **higher-order term rewriting**\(^{50}\)
- analysis of **parallel**-innermost runtime complexity\(^{51}\)

\(^{47}\) G. Moser, M. Schneckenreither: *Automated amortised resource analysis for term rewrite systems*, SCP '20
\(^{48}\) M. Avanzini, U. Dal Lago, A. Yamada: *On probabilistic term rewriting*, SCP '20
\(^{49}\) S. Winkler, G. Moser: *Runtime complexity analysis of logically constrained rewriting*, LOPSTR '20
\(^{50}\) C. Kop, D. Vale: *Tuple interpretations for higher-order rewriting*, FSCD '21
\(^{51}\) T. Baudon, C. Fuhs, L. Gonnord: *Parallel complexity of term rewriting systems*, WST '21
Conclusion

Complexity analysis for term rewriting: active field of research
Conclusion

- Complexity analysis for term rewriting: active field of research
- Push-button tools to infer upper and lower complexity bounds available
Conclusion

- Complexity analysis for term rewriting: active field of research
- Push-button tools to infer upper and lower complexity bounds available
- Runtime complexity a popular translation target
Conclusion

- Complexity analysis for term rewriting: active field of research
- Push-button tools to infer upper and lower complexity bounds available
- Runtime complexity a popular translation target
- Cross-fertilisation with techniques for other formalisms (integer transition systems, functional programs, ...)

Thanks a lot for your attention!
Conclusion

- Complexity analysis for term rewriting: active field of research
- Push-button tools to infer upper and lower complexity bounds available
- Runtime complexity a popular translation target
- Cross-fertilisation with techniques for other formalisms (integer transition systems, functional programs, ...)

Thanks a lot for your attention!


