Automated Complexity Analysis for Term Rewriting

Carsten Fuhs

Birkbeck, University of London

Course at the International School on Rewriting 2021
Madrid, Spain[1]
5th July 2021

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

---

[1] virtually

What is *Term Rewriting*?

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

What is *Term Rewriting*?

(1) Core functional programming language

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

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

What is *Term Rewriting*?

(1) Core functional programming language

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

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

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

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) $\mathcal{R}$)

$$\mathsf{double}(\mathsf{0}) \to \mathsf{0}$$

$$\mathsf{double}(\mathsf{s}(x)) \to \mathsf{s}(\mathsf{s}(\mathsf{double}(x)))$$

What is *Term Rewriting*?

(1) Core functional programming language

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

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

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

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

$$\mathsf{double}(\mathsf{0}) \to \mathsf{0}$$

$$\mathsf{double}(\mathsf{s}(x)) \to \mathsf{s}(\mathsf{s}(\mathsf{double}(x)))$$

Compute “double of 3 is 6”:

$$\mathsf{double}(\mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{0}))))$$

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) $\mathcal{R}$)

$$\mathsf{double}(\mathsf{0}) \to \mathsf{0}$$

$$\mathsf{double}(\mathsf{s}(x)) \to \mathsf{s}(\mathsf{s}(\mathsf{double}(x)))$$

Compute "double of 3 is 6":

$$\begin{aligned} & \mathsf{double}(\mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{0})))) \\ \to_{\mathcal{R}} \quad & \mathsf{s}(\mathsf{s}(\mathsf{double}(\mathsf{s}(\mathsf{s}(\mathsf{0}))))) \end{aligned}$$

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) $\mathcal{R}$)

$$\mathsf{double}(\mathsf{0}) \to \mathsf{0}$$

$$\mathsf{double}(\mathsf{s}(x)) \to \mathsf{s}(\mathsf{s}(\mathsf{double}(x)))$$

Compute “double of 3 is 6”:

$$\begin{aligned} & \mathsf{double}(\mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{0})))) \\ \to_{\mathcal{R}} \quad & \mathsf{s}(\mathsf{s}(\mathsf{double}(\mathsf{s}(\mathsf{s}(\mathsf{0}))))) \\ \to_{\mathcal{R}} \quad & \mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{double}(\mathsf{s}(\mathsf{0})))))) \end{aligned}$$

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) $\mathcal{R}$)

$$\mathsf{double}(\mathsf{0}) \to \mathsf{0}$$

$$\mathsf{double}(\mathsf{s}(x)) \to \mathsf{s}(\mathsf{s}(\mathsf{double}(x)))$$

Compute "double of 3 is 6":

$$\begin{aligned} & \mathsf{double}(\mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{0})))) \\ \to_{\mathcal{R}} \; & \mathsf{s}(\mathsf{s}(\mathsf{double}(\mathsf{s}(\mathsf{s}(\mathsf{0}))))) \\ \to_{\mathcal{R}} \; & \mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{double}(\mathsf{s}(\mathsf{0})))))) \\ \to_{\mathcal{R}} \; & \mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{double}(\mathsf{0}))))))) \end{aligned}$$

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) $\mathcal{R}$)

$$\mathsf{double}(\mathsf{0}) \to \mathsf{0}$$

$$\mathsf{double}(\mathsf{s}(x)) \to \mathsf{s}(\mathsf{s}(\mathsf{double}(x)))$$

Compute “double of 3 is 6”:

$$
\begin{array}{ll}
 & \mathsf{double}(\mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{0})))) \\
\to_{\mathcal{R}} & \mathsf{s}(\mathsf{s}(\mathsf{double}(\mathsf{s}(\mathsf{s}(\mathsf{0}))))) \\
\to_{\mathcal{R}} & \mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{double}(\mathsf{s}(\mathsf{0})))))) \\
\to_{\mathcal{R}} & \mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{double}(\mathsf{0}))))))) \\
\to_{\mathcal{R}} & \mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{0}))))))
\end{array}
$$

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) $\mathcal{R}$)

$$\mathsf{double}(\mathsf{0}) \to \mathsf{0}$$

$$\mathsf{double}(\mathsf{s}(x)) \to \mathsf{s}(\mathsf{s}(\mathsf{double}(x)))$$

Compute “double of 3 is 6”:

$$\begin{aligned} & \mathsf{double}(\mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{0})))) \\ \to_{\mathcal{R}} \; & \mathsf{s}(\mathsf{s}(\mathsf{double}(\mathsf{s}(\mathsf{s}(\mathsf{0}))))) \\ \to_{\mathcal{R}} \; & \mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{double}(\mathsf{s}(\mathsf{0})))))) \\ \to_{\mathcal{R}} \; & \mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{double}(\mathsf{0}))))))) \\ \to_{\mathcal{R}} \; & \mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{0})))))) \end{aligned}$$

in 4 steps with $\to_{\mathcal{R}}$

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) $\mathcal{R}$)

$$\mathsf{double}(\mathsf{0}) \to \mathsf{0}$$

$$\mathsf{double}(\mathsf{s}(x)) \to \mathsf{s}(\mathsf{s}(\mathsf{double}(x)))$$

Compute “double of 3 is 6”:

$$\begin{array}{ll} & \mathsf{double}(\mathsf{s}^3(\mathsf{0})) \\ \to_{\mathcal{R}} & \mathsf{s}^2(\mathsf{double}(\mathsf{s}^2(\mathsf{0}))) \\ \to_{\mathcal{R}} & \mathsf{s}^4(\mathsf{double}(\mathsf{s}(\mathsf{0}))) \\ \to_{\mathcal{R}} & \mathsf{s}^6(\mathsf{double}(\mathsf{0})) \\ \to_{\mathcal{R}} & \mathsf{s}^6(\mathsf{0}) \end{array}$$

in 4 steps with $\to_{\mathcal{R}}$

What is *Complexity* of Term Rewriting?

Given: TRS $\mathcal{R}$ (e.g., $\{$ $\mathsf{double}(\mathsf{0}) \to \mathsf{0}$, $\mathsf{double}(\mathsf{s}(x)) \to \mathsf{s}(\mathsf{s}(\mathsf{double}(x)))$ $\}$)

What is *Complexity* of Term Rewriting?

Given: TRS $\mathcal{R}$ (e.g., $\{\ \mathsf{double}(\mathsf{0}) \to \mathsf{0},\ \mathsf{double}(\mathsf{s}(x)) \to \mathsf{s}(\mathsf{s}(\mathsf{double}(x)))\ \}$)

Question: How long can a $\to_{\mathcal{R}}$ sequence from a term of size n become?

(worst case)

What is *Complexity* of Term Rewriting?

Given: TRS $\mathcal{R}$ (e.g., $\{$ $\mathsf{double}(\mathsf{0}) \to \mathsf{0}$, $\mathsf{double}(\mathsf{s}(x)) \to \mathsf{s}(\mathsf{s}(\mathsf{double}(x)))$ $\}$)

Question: How long can a $\to_{\mathcal{R}}$ 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., $\{\ \mathsf{double}(\mathsf{0}) \to \mathsf{0},\ \mathsf{double}(\mathsf{s}(x)) \to \mathsf{s}(\mathsf{s}(\mathsf{double}(x)))\ \}$)

Question: How long can a $\to_{\mathcal{R}}$ 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., $\{$ $\mathsf{double}(\mathsf{0}) \to \mathsf{0}$, $\mathsf{double}(\mathsf{s}(x)) \to \mathsf{s}(\mathsf{s}(\mathsf{double}(x)))$ $\}$)

Question: How long can a $\to_{\mathcal{R}}$ sequence from a term of size n become? (worst case)

Here: Does $\mathcal{R}$ have complexity $\Theta(n)$?

(1) **Yes!**

$$\mathsf{double}(\mathsf{s}^{n-2}(\mathsf{0})) \to_{\mathcal{R}}^{n-1} \mathsf{s}^{2n-4}(\mathsf{0})$$

What is *Complexity* of Term Rewriting?

Given: TRS $\mathcal{R}$ (e.g., $\{$ $\mathsf{double}(\mathsf{0}) \to \mathsf{0}$, $\mathsf{double}(\mathsf{s}(x)) \to \mathsf{s}(\mathsf{s}(\mathsf{double}(x)))$ $\}$)

Question: How long can a $\to_{\mathcal{R}}$ sequence from a term of size n become? (worst case)

Here: Does $\mathcal{R}$ have complexity $\Theta(n)$?

(1) **Yes!**

$$\mathsf{double}(\mathsf{s}^{n-2}(\mathsf{0})) \to_{\mathcal{R}}^{n-1} \mathsf{s}^{2n-4}(\mathsf{0})$$

- **basic terms** $f(t_1, \ldots, t_n)$ with t_i constructor terms allow only n steps

What is *Complexity* of Term Rewriting?

Given: TRS $\mathcal{R}$ (e.g., $\{$ double(0) $\to$ 0, double(s(x)) $\to$ s(s(double(x))) $\}$)

Question: How long can a $\to_{\mathcal{R}}$ sequence from a term of size n become? (worst case)

Here: Does $\mathcal{R}$ have complexity $\Theta(n)$?

(1) **Yes!**

$$\mathsf{double}(\mathsf{s}^{n-2}(\mathsf{0})) \to_{\mathcal{R}}^{n-1} \mathsf{s}^{2n-4}(\mathsf{0})$$

- **basic terms** $f(t_1, \ldots, t_n)$ with t_i constructor terms allow only n steps
- **runtime complexity** $\mathrm{rc}_{\mathcal{R}}(n)$: basic terms as start terms

What is *Complexity* of Term Rewriting?

Given: TRS $\mathcal{R}$ (e.g., $\{$ $\mathsf{double}(\mathsf{0}) \to \mathsf{0}$, $\mathsf{double}(\mathsf{s}(x)) \to \mathsf{s}(\mathsf{s}(\mathsf{double}(x)))$ $\}$)

Question: How long can a $\to_{\mathcal{R}}$ sequence from a term of size n become? (worst case)

Here: Does $\mathcal{R}$ have complexity $\Theta(n)$?

(1) **Yes!**

$$\mathsf{double}(\mathsf{s}^{n-2}(\mathsf{0})) \to_{\mathcal{R}}^{n-1} \mathsf{s}^{2n-4}(\mathsf{0})$$

- **basic terms** $f(t_1, \ldots, t_n)$ with t_i constructor terms allow only n steps
- **runtime complexity** $\mathrm{rc}_{\mathcal{R}}(n)$: basic terms as start terms
- $\mathrm{rc}_{\mathcal{R}}(n)$ for **program analysis**

What is *Complexity* of Term Rewriting?

Given: TRS $\mathcal{R}$ (e.g., $\{\ \mathsf{double}(\mathsf{0}) \to \mathsf{0},\ \mathsf{double}(\mathsf{s}(x)) \to \mathsf{s}(\mathsf{s}(\mathsf{double}(x)))\ \}$)

Question: How long can a $\to_{\mathcal{R}}$ sequence from a term of size n become? (worst case)

Here: Does $\mathcal{R}$ have complexity $\Theta(n)$?

(1) **Yes!**

$$\mathsf{double}(\mathsf{s}^{n-2}(\mathsf{0})) \to_{\mathcal{R}}^{n-1} \mathsf{s}^{2n-4}(\mathsf{0})$$

- **basic terms** $f(t_1, \ldots, t_n)$ with t_i constructor terms allow only n steps
- **runtime complexity** $\mathrm{rc}_{\mathcal{R}}(n)$: basic terms as start terms
- $\mathrm{rc}_{\mathcal{R}}(n)$ for **program analysis**

(2) **No!**

What is *Complexity* of Term Rewriting?

Given: TRS $\mathcal{R}$ (e.g., $\{$ $\mathsf{double}(\mathsf{0}) \to \mathsf{0}$, $\mathsf{double}(\mathsf{s}(x)) \to \mathsf{s}(\mathsf{s}(\mathsf{double}(x)))$ $\}$)

Question: How long can a $\to_{\mathcal{R}}$ sequence from a term of size n become? (worst case)

Here: Does $\mathcal{R}$ have complexity $\Theta(n)$?

(1) **Yes!**

$$\mathsf{double}(\mathsf{s}^{n-2}(\mathsf{0})) \to_{\mathcal{R}}^{n-1} \mathsf{s}^{2n-4}(\mathsf{0})$$

- **basic terms** $f(t_1, \ldots, t_n)$ with t_i constructor terms allow only n steps
- **runtime complexity** $\mathrm{rc}_{\mathcal{R}}(n)$: basic terms as start terms
- $\mathrm{rc}_{\mathcal{R}}(n)$ for **program analysis**

(2) **No!**

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

What is *Complexity* of Term Rewriting?

Given: TRS $\mathcal{R}$ (e.g., $\{\ \mathsf{double}(\mathsf{0}) \to \mathsf{0},\ \mathsf{double}(\mathsf{s}(x)) \to \mathsf{s}(\mathsf{s}(\mathsf{double}(x)))\ \}$)

Question: How long can a $\to_{\mathcal{R}}$ sequence from a term of size n become?
(worst case)

Here: Does $\mathcal{R}$ have complexity $\Theta(n)$?

(1) **Yes!**

$$\mathsf{double}(\mathsf{s}^{n-2}(\mathsf{0})) \to_{\mathcal{R}}^{n-1} \mathsf{s}^{2n-4}(\mathsf{0})$$

- **basic terms** $f(t_1, \ldots, t_n)$ with t_i constructor terms allow only n steps
- **runtime complexity** $\mathrm{rc}_{\mathcal{R}}(n)$: basic terms as start terms
- $\mathrm{rc}_{\mathcal{R}}(n)$ for **program analysis**

(2) **No!**

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

- $\mathsf{double}^{n-2}(\mathsf{s}(\mathsf{0}))$ allows $\Theta(2^n)$ many steps to $\mathsf{s}^{2^{n-2}}(\mathsf{0})$

What is *Complexity* of Term Rewriting?

Given: TRS $\mathcal{R}$ (e.g., $\{$ double(0) $\to$ 0, double(s(x)) $\to$ s(s(double(x))) $\}$)

Question: How long can a $\to_{\mathcal{R}}$ sequence from a term of size n become? (worst case)

Here: Does $\mathcal{R}$ have complexity $\Theta(n)$?

(1) **Yes!**

$$\mathsf{double}(\mathsf{s}^{n-2}(\mathsf{0})) \to_{\mathcal{R}}^{n-1} \mathsf{s}^{2n-4}(\mathsf{0})$$

- **basic terms** $f(t_1,\ldots,t_n)$ with t_i constructor terms allow only n steps
- **runtime complexity** $\mathrm{rc}_{\mathcal{R}}(n)$: basic terms as start terms
- $\mathrm{rc}_{\mathcal{R}}(n)$ for **program analysis**

(2) **No!**

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

- $\mathsf{double}^{n-2}(\mathsf{s}(\mathsf{0}))$ allows $\Theta(2^n)$ many steps to $\mathsf{s}^{2^{n-2}}(\mathsf{0})$
- **derivational complexity** $\mathrm{dc}_{\mathcal{R}}(n)$: no restrictions on start terms

What is *Complexity* of Term Rewriting?

Given: TRS $\mathcal{R}$ (e.g., $\{\ \mathsf{double}(\mathsf{0}) \to \mathsf{0},\ \mathsf{double}(\mathsf{s}(x)) \to \mathsf{s}(\mathsf{s}(\mathsf{double}(x)))\ \}$)

Question: How long can a $\to_{\mathcal{R}}$ sequence from a term of size n become? (worst case)

Here: Does $\mathcal{R}$ have complexity $\Theta(n)$?

(1) **Yes!**

$$\mathsf{double}(\mathsf{s}^{n-2}(\mathsf{0})) \to_{\mathcal{R}}^{n-1} \mathsf{s}^{2n-4}(\mathsf{0})$$

- **basic terms** $f(t_1, \dots, t_n)$ with t_i constructor terms allow only n steps
- **runtime complexity** $\mathrm{rc}_{\mathcal{R}}(n)$: basic terms as start terms
- $\mathrm{rc}_{\mathcal{R}}(n)$ for **program analysis**

(2) **No!**

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

- $\mathsf{double}^{n-2}(\mathsf{s}(\mathsf{0}))$ allows $\Theta(2^n)$ many steps to $\mathsf{s}^{2^{n-2}}(\mathsf{0})$
- **derivational complexity** $\mathrm{dc}_{\mathcal{R}}(n)$: no restrictions on start terms
- $\mathrm{dc}_{\mathcal{R}}(n)$ 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[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]

---

[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

[5] M. Avanzini, G. Moser, M. Schaper: *TcT: Tyrolean Complexity Tool*, TACAS '16, `https://tcs-informatik.uibk.ac.at/tools/tct/`

[6] M. Korp, C. Sternagel, H. Zankl, A. Middeldorp: *Tyrolean Termination Tool 2*, RTA '09, `http://cl-informatik.uibk.ac.at/software/cat/`

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]

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

[5] M. Avanzini, G. Moser, M. Schaper: *TcT: Tyrolean Complexity Tool*, TACAS '16, https://tcs-informatik.uibk.ac.at/tools/tct/

[6] M. Korp, C. Sternagel, H. Zankl, A. Middeldorp: *Tyrolean Termination Tool 2*, RTA '09, http://cl-informatik.uibk.ac.at/software/cat/

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]

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

[5] M. Avanzini, G. Moser, M. Schaper: *TcT: Tyrolean Complexity Tool*, TACAS '16, https://tcs-informatik.uibk.ac.at/tools/tct/

[6] M. Korp, C. Sternagel, H. Zankl, A. Middeldorp: *Tyrolean Termination Tool 2*, RTA '09, http://cl-informatik.uibk.ac.at/software/cat/

A Short Timeline (2/2)

...

2021: Termination Competition 2021 with complexity analysis tools AProVE[7], TcT in July 2021

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

First run just finished!

---

[7] J. Giesl, C. Aschermann, M. Brockschmidt, F. Emmes, F. Frohn, C. Fuhs, J. Hensel, C. Otto, M. Plücker, P. Schneider-Kamp, T. Ströder, S. Swiderski, R. Thiemann: *Analyzing Program Termination and Complexity Automatically with AProVE*, JAR '17, `http://aprove.informatik.rwth-aachen.de/`

Some Definitions

Definition (Derivation Height dh)

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

$$\mathrm{dh}(t, \longrightarrow) \;=\; \sup \{\, n \mid \exists t'.\, t \longrightarrow^{n} t' \,\}$$

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

Some Definitions

Definition (Derivation Height dh)

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

$$\mathrm{dh}(t, \longrightarrow) \;\;=\;\; \sup \{\, n \mid \exists t'.\, t \longrightarrow^{n} t' \,\}$$

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

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

Some Definitions

Definition (Derivation Height dh)

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

$$\mathrm{dh}(t, \longrightarrow) \;=\; \sup \{\, n \mid \exists t'.\, t \longrightarrow^{n} t' \,\}$$

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

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

Example: $\mathrm{dh}(\,\mathsf{double}(\mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{0})))),\ \longrightarrow_{\mathcal{R}}\,) = 4$

Some Definitions

Definition (Derivation Height dh)

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

$$\mathrm{dh}(t, \longrightarrow) \;=\; \sup \{\, n \mid \exists t'.\, t \longrightarrow^{n} t' \,\}$$

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

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

Example: $\mathrm{dh}(\,\mathsf{double}(\mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{0})))),\ \longrightarrow_{\mathcal{R}}\,) = 4$

Definition (Derivational Complexity dc)

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

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

Some Definitions

Definition (Derivation Height dh)

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

$$\mathrm{dh}(t, \longrightarrow) \;=\; \sup \{ \, n \mid \exists t'.\, t \longrightarrow^{n} t' \, \}$$

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

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

Example: $\mathrm{dh}(\,\mathsf{double}(\mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{0})))),\ \longrightarrow_{\mathcal{R}}\,) = 4$

Definition (Derivational Complexity dc)

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

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

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

Some Definitions

Definition (Derivation Height dh)

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

$$\mathrm{dh}(t, \longrightarrow) \;=\; \sup \{\, n \mid \exists t'.\, t \longrightarrow^{n} t' \,\}$$

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

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

Example: $\mathrm{dh}(\,\mathsf{double}(\mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{0})))),\ \longrightarrow_{\mathcal{R}}\,) = 4$

Definition (Derivational Complexity dc)

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

$$\mathrm{dc}_{\mathcal{R}}(n) \;=\; \sup \{\, \mathrm{dh}(t, \longrightarrow_{\mathcal{R}}) \mid t \in \mathcal{T}(\mathcal{F}, \mathcal{V}), |t| \le n \,\}$$

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

Example: For $\mathcal{R}$ for double, we have $\mathrm{dc}_{\mathcal{R}}(n) \in \Theta(2^n)$.

Upper Bounds

The Bad News for automation:

Upper Bounds

The Bad News for automation:

For a given TRS $\mathcal{R}$, the following questions are undecidable:

- $\mathrm{dc}_{\mathcal{R}}(n) = \omega$ for some n? ($\longrightarrow$ termination!)

Upper Bounds

The Bad News for automation:

For a given TRS $\mathcal{R}$, the following questions are undecidable:

- $\mathrm{dc}_{\mathcal{R}}(n) = \omega$ for some n? ($\longrightarrow$ termination!)
- $\mathrm{dc}_{\mathcal{R}}(n)$ polynomially bounded?[8]

---

[8] 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:

- $\mathrm{dc}_{\mathcal{R}}(n) = \omega$ for some n? ($\longrightarrow$ termination!)
- $\mathrm{dc}_{\mathcal{R}}(n)$ polynomially bounded?[8]

Goal: find **approximations** for derivational complexity

Initial focus: find upper bounds

$$\mathrm{dc}_{\mathcal{R}}(n) \in \mathcal{O}(...)$$

---

[8] A. Schnabl and J. G. Simonsen: *The exact hardness of deciding derivational and runtime complexity*, CSL '11

Derivational Complexity from Polynomial Interpretations (1/2)

Example (double)

$$\begin{array}{rcl}\mathsf{double}(\mathsf{0}) & \rightarrow & \mathsf{0} \\ \mathsf{double}(\mathsf{s}(x)) & \rightarrow & \mathsf{s}(\mathsf{s}(\mathsf{double}(x)))\end{array}$$

Derivational Complexity from Polynomial Interpretations (1/2)

Example (double)

$$\begin{aligned} \mathsf{double}(\mathsf{0}) &\succ \mathsf{0} \\ \mathsf{double}(\mathsf{s}(x)) &\succ \mathsf{s}(\mathsf{s}(\mathsf{double}(x))) \end{aligned}$$

Show $\mathrm{dc}_{\mathcal{R}}(n) < \omega$ by **termination proof** with reduction order $\succ$ on terms.

Derivational Complexity from Polynomial Interpretations (1/2)

Example (double)

$$\begin{aligned} \mathsf{double}(\mathsf{0}) &\succ \mathsf{0} \\ \mathsf{double}(\mathsf{s}(x)) &\succ \mathsf{s}(\mathsf{s}(\mathsf{double}(x))) \end{aligned}$$

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

---

[9] D. Lankford: *Canonical algebraic simplification in computational logic*, U Texas '75

Derivational Complexity from Polynomial Interpretations (1/2)

Example (double)

$$\begin{aligned} \mathsf{double}(\mathsf{0}) &\succ \mathsf{0} \\ \mathsf{double}(\mathsf{s}(x)) &\succ \mathsf{s}(\mathsf{s}(\mathsf{double}(x))) \end{aligned}$$

Show $\mathrm{dc}_{\mathcal{R}}(n) < \omega$ 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: $[\mathsf{double}](x) = 3 \cdot x, \qquad [\mathsf{s}](x) = x + 1, \qquad [\mathsf{0}] = 1$

---

[9] D. Lankford: *Canonical algebraic simplification in computational logic*, U Texas '75

Derivational Complexity from Polynomial Interpretations (1/2)

Example (double)

$$\begin{aligned} \mathsf{double}(\mathsf{0}) &\succ \mathsf{0} \\ \mathsf{double}(\mathsf{s}(x)) &\succ \mathsf{s}(\mathsf{s}(\mathsf{double}(x))) \end{aligned}$$

Show $\mathrm{dc}_{\mathcal{R}}(n) < \omega$ 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: $[\mathsf{double}](x) = 3 \cdot x,\qquad [\mathsf{s}](x) = x + 1,\qquad [\mathsf{0}] = 1$

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)

$$\begin{array}{rcl|rcl} \mathsf{double}(\mathsf{0}) & \succ & \mathsf{0} & 3 & > & 1 \\ \mathsf{double}(\mathsf{s}(x)) & \succ & \mathsf{s}(\mathsf{s}(\mathsf{double}(x))) & 3 \cdot x + 3 & > & 3 \cdot x + 2 \end{array}$$

Show $\mathrm{dc}_{\mathcal{R}}(n) < \omega$ 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: $[\mathsf{double}](x) = 3 \cdot x$, $[\mathsf{s}](x) = x + 1$, $[\mathsf{0}] = 1$

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)

$\mathsf{double}(\mathsf{0})$	$\succ$	$\mathsf{0}$	3	$>$	1
$\mathsf{double}(\mathsf{s}(x))$	$\succ$	$\mathsf{s}(\mathsf{s}(\mathsf{double}(x)))$	$3 \cdot x + 3$	$>$	$3 \cdot x + 2$

Show $\mathrm{dc}_{\mathcal{R}}(n) < \omega$ 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: $[\mathsf{double}](x) = 3 \cdot x, \quad [\mathsf{s}](x) = x + 1, \quad [\mathsf{0}] = 1$

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

Derivational Complexity from Polynomial Interpretations (2/2)

Example (double)

$$\begin{array}{rcl|rcl} \mathsf{double}(\mathsf{0}) & \succ & \mathsf{0} & 3 & > & 1 \\ \mathsf{double}(\mathsf{s}(x)) & \succ & \mathsf{s}(\mathsf{s}(\mathsf{double}(x))) & 3 \cdot x + 3 & > & 3 \cdot x + 2 \end{array}$$

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

This proves more than just termination...

Derivational Complexity from Polynomial Interpretations (2/2)

Example (double)

$$\begin{array}{rcl|rcl} \mathsf{double}(\mathsf{0}) & \succ & \mathsf{0} & 3 & > & 1 \\ \mathsf{double}(\mathsf{s}(x)) & \succ & \mathsf{s}(\mathsf{s}(\mathsf{double}(x))) & 3 \cdot x + 3 & > & 3 \cdot x + 2 \end{array}$$

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

This proves more than just termination...

Theorem (Upper bounds for $\mathrm{dc}_{\mathcal{R}}(n)$ from polynomial interpretations[12])

- *Termination proof for TRS $\mathcal{R}$ with* ***polynomial*** *interpretation*

$$\Rightarrow \mathrm{dc}_{\mathcal{R}}(n) \in 2^{2^{\mathcal{O}(n)}}$$

---

[12] D. Hofbauer, C. Lautemann: *Termination proofs and the length of derivations*, RTA '89

Derivational Complexity from Polynomial Interpretations (2/2)

Example (double)

$$\begin{array}{rcl|rcl} \mathsf{double}(\mathsf{0}) & \succ & \mathsf{0} & 3 & > & 1 \\ \mathsf{double}(\mathsf{s}(x)) & \succ & \mathsf{s}(\mathsf{s}(\mathsf{double}(x))) & 3 \cdot x + 3 & > & 3 \cdot x + 2 \end{array}$$

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

This proves more than just termination. . .

Theorem (Upper bounds for $\mathrm{dc}_{\mathcal{R}}(n)$ from polynomial interpretations[12])

- *Termination proof for TRS $\mathcal{R}$ with **polynomial** interpretation*
  $\Rightarrow \mathrm{dc}_{\mathcal{R}}(n) \in 2^{2^{\mathcal{O}(n)}}$
- *Termination proof for TRS $\mathcal{R}$ with **linear polynomial** interpretation*
  $\Rightarrow \mathrm{dc}_{\mathcal{R}}(n) \in 2^{\mathcal{O}(n)}$

---

[12] D. Hofbauer, C. Lautemann: *Termination proofs and the length of derivations*, RTA '89

Derivational Complexity from Termination Proofs (1/2)

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

- matchbounds[13] $\Rightarrow \mathrm{dc}_{\mathcal{R}}(n) \in \mathcal{O}(n)$
- arctic matrix interpretations[14] $\Rightarrow \mathrm{dc}_{\mathcal{R}}(n) \in \mathcal{O}(n)$

---

[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

Derivational Complexity from Termination Proofs (1/2)

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

- matchbounds[13] $\Rightarrow \mathrm{dc}_{\mathcal{R}}(n) \in \mathcal{O}(n)$
- arctic matrix interpretations[14] $\Rightarrow \mathrm{dc}_{\mathcal{R}}(n) \in \mathcal{O}(n)$
- triangular matrix interpretation[15] $\Rightarrow \mathrm{dc}_{\mathcal{R}}(n)$ is at most polynomial
- matrix interpretation of spectral radius[16] ≤ 1
  $\Rightarrow \mathrm{dc}_{\mathcal{R}}(n)$ 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

Derivational Complexity from Termination Proofs (1/2)

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

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

Derivational Complexity from Termination Proofs (2/2)

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

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

---

[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

Derivational Complexity from Termination Proofs (2/2)

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

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

Runtime Complexity

- So far: upper bounds for derivational complexity

Runtime 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**: $\mathsf{double}(\mathsf{s}^n(0))$

Runtime Complexity

- So far: upper bounds for derivational complexity
- But: derivational complexity counter-intuitive, often infeasible
- Wanted: complexity of evaluation of double **on data**: $\mathsf{double}(\mathsf{s}^n(0))$

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

---

[25] N. Hirokawa, G. Moser: *Automated complexity analysis based on the dependency pair method*, IJCAR '08

Runtime Complexity

- So far: upper bounds for derivational complexity
- But: derivational complexity counter-intuitive, often infeasible
- Wanted: complexity of evaluation of double **on data**: double($\mathsf{s}^n(0)$)

Definition (Basic Term[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[25])

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

$$\mathrm{rc}_{\mathcal{R}}(n) \;=\; \sup \{\, \mathrm{dh}(t, \to_{\mathcal{R}}) \mid t \in \mathcal{T}_{\text{basic}}, |t| \leq n \,\}$$

---

[25] N. Hirokawa, G. Moser: *Automated complexity analysis based on the dependency pair method*, IJCAR '08

Runtime Complexity

- So far: upper bounds for derivational complexity
- But: derivational complexity counter-intuitive, often infeasible
- Wanted: complexity of evaluation of double **on data**: $\mathsf{double}(\mathsf{s}^n(0))$

Definition (Basic Term[25])

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

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

is in the set $\mathcal{T}_{\mathrm{basic}}$ 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[25])

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

$$\mathrm{rc}_{\mathcal{R}}(n) \quad = \quad \sup \{ \, \mathrm{dh}(t, \to_{\mathcal{R}}) \mid t \in \mathcal{T}_{\mathrm{basic}}, |t| \leq n \, \}$$

$\mathrm{rc}_{\mathcal{R}}(n)$: like derivational complexity… but for basic terms only!

---

[25] N. Hirokawa, G. Moser: *Automated complexity analysis based on the dependency pair method*, IJCAR '08

Runtime Complexity from Polynomial Interpretations

Polynomial interpretations can induce upper bounds to runtime complexity:[26]

Definition (Strongly linear polynomial, restricted interpretation)

- Polynomial p is **strongly linear** iff $p(x_1, \ldots, x_n) = x_1 + \cdots + x_n + a$ 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 $c \in \mathbb{N}$.

---

[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:[26]

Definition (Strongly linear polynomial, restricted interpretation)

- Polynomial p is **strongly linear** iff $p(x_1, \ldots, x_n) = x_1 + \cdots + x_n + a$ 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 $\mathrm{rc}_{\mathcal{R}}(n)$ from restricted interpretations)

*Termination proof for TRS $\mathcal{R}$ with **restricted** interpretation $[\,\cdot\,]$ of degree at most d for $[f]$* $\Rightarrow \mathrm{rc}_{\mathcal{R}}(n) \in \mathcal{O}(n^d)$

---

[26] G. Bonfante, A. Cichon, J. Marion, H. Touzet: *Algorithms with polynomial interpretation termination proof*, JFP '01

Runtime Complexity from Polynomial Interpretations

Polynomial interpretations can induce upper bounds to runtime complexity:[26]

Definition (Strongly linear polynomial, restricted interpretation)

- Polynomial p is **strongly linear** iff $p(x_1, \ldots, x_n) = x_1 + \cdots + x_n + a$ 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 $\mathrm{rc}_{\mathcal{R}}(n)$ from restricted interpretations)

*Termination proof for TRS $\mathcal{R}$ with **restricted** interpretation $[\,\cdot\,]$ of degree at most d for $[f]$* $\Rightarrow \mathrm{rc}_{\mathcal{R}}(n) \in \mathcal{O}(n^d)$

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

$\Rightarrow \mathrm{rc}_{\mathcal{R}}(n) \in \mathcal{O}(n)$ for TRS $\mathcal{R}$ for double

---

[26] G. Bonfante, A. Cichon, J. Marion, H. Touzet: *Algorithms with polynomial interpretation termination proof*, JFP '01

Dependency Tuples for *Innermost* Runtime Complexity irc

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

Example (reverse)

$$\begin{array}{rcl|rcl} \mathsf{app}(\mathsf{nil}, y) & \to & y & \mathsf{app}(\mathsf{add}(n, x), y) & \to & \mathsf{add}(n, \mathsf{app}(x, y)) \\ \mathsf{reverse}(\mathsf{nil}) & \to & \mathsf{nil} & \mathsf{reverse}(\mathsf{add}(n, x)) & \to & \mathsf{app}(\mathsf{reverse}(x), \mathsf{add}(n, \mathsf{nil})) \end{array}$$

Dependency Tuples for *Innermost* Runtime Complexity irc

Here: innermost rewriting ($\approx$ call-by-value)

Example (reverse)

$$\begin{array}{rcl|rcl} \mathsf{app}(\mathsf{nil}, y) & \to & y & \mathsf{app}(\mathsf{add}(n, x), y) & \to & \mathsf{add}(n, \mathsf{app}(x, y)) \\ \mathsf{reverse}(\mathsf{nil}) & \to & \mathsf{nil} & \mathsf{reverse}(\mathsf{add}(n, x)) & \to & \mathsf{app}(\mathsf{reverse}(x), \mathsf{add}(n, \mathsf{nil})) \end{array}$$

For rule $\ell \to r$, eval of ℓ costs 1 + eval of all function calls in r **together**:

---

[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 ($\approx$ call-by-value)

Example (reverse)

$$\begin{array}{rcl|rcl} \mathsf{app}(\mathsf{nil}, y) & \to & y & \mathsf{app}(\mathsf{add}(n, x), y) & \to & \mathsf{add}(n, \mathsf{app}(x, y)) \\ \mathsf{reverse}(\mathsf{nil}) & \to & \mathsf{nil} & \mathsf{reverse}(\mathsf{add}(n, x)) & \to & \mathsf{app}(\mathsf{reverse}(x), \mathsf{add}(n, \mathsf{nil})) \end{array}$$

For rule $\ell \to r$, eval of ℓ costs 1 + eval of all function calls in r **together**:

Example (Dependency Tuples[27] for reverse)

$$\begin{array}{rcl} \mathsf{app}^\sharp(\mathsf{nil}, y) & \to & \mathsf{Com}_0 \\ \mathsf{app}^\sharp(\mathsf{add}(n, x), y) & \to & \mathsf{Com}_1(\mathsf{app}^\sharp(x, y)) \\ \mathsf{reverse}^\sharp(\mathsf{nil}) & \to & \mathsf{Com}_0 \\ \mathsf{reverse}^\sharp(\mathsf{add}(n, x)) & \to & \mathsf{Com}_2(\mathsf{app}^\sharp(\mathsf{reverse}(x), \mathsf{add}(n, \mathsf{nil})), \mathsf{reverse}^\sharp(x)) \end{array}$$

- Function calls to count marked with $\sharp$
- 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

Polynomial Interpretations for Dependency Tuples

Example (reverse, Dependency Tuples for reverse)

$$\begin{array}{rcl}
\mathsf{app}^\sharp(\mathsf{nil}, y) & \to & \mathsf{Com}_0 \\
\mathsf{app}^\sharp(\mathsf{add}(n, x), y) & \to & \mathsf{Com}_1(\mathsf{app}^\sharp(x, y)) \\
\mathsf{reverse}^\sharp(\mathsf{nil}) & \to & \mathsf{Com}_0 \\
\mathsf{reverse}^\sharp(\mathsf{add}(n, x)) & \to & \mathsf{Com}_2(\mathsf{app}^\sharp(\mathsf{reverse}(x), \mathsf{add}(n, \mathsf{nil})), \mathsf{reverse}^\sharp(x))
\end{array}$$

$$\begin{array}{rcl|rcl}
\mathsf{app}(\mathsf{nil}, y) & \to & y & \mathsf{app}(\mathsf{add}(n, x), y) & \to & \mathsf{add}(n, \mathsf{app}(x, y)) \\
\mathsf{reverse}(\mathsf{nil}) & \to & \mathsf{nil} & \mathsf{reverse}(\mathsf{add}(n, x)) & \to & \mathsf{app}(\mathsf{reverse}(x), \mathsf{add}(n, \mathsf{nil}))
\end{array}$$

Polynomial Interpretations for Dependency Tuples

Example (reverse, Dependency Tuples for reverse)

$$\begin{aligned}
\mathsf{app}^\sharp(\mathsf{nil}, y) &\to \mathsf{Com}_0\\
\mathsf{app}^\sharp(\mathsf{add}(n, x), y) &\to \mathsf{Com}_1(\mathsf{app}^\sharp(x, y))\\
\mathsf{reverse}^\sharp(\mathsf{nil}) &\to \mathsf{Com}_0\\
\mathsf{reverse}^\sharp(\mathsf{add}(n, x)) &\to \mathsf{Com}_2(\mathsf{app}^\sharp(\mathsf{reverse}(x), \mathsf{add}(n, \mathsf{nil})), \mathsf{reverse}^\sharp(x))
\end{aligned}$$

$$\begin{array}{rcl|rcl}
\mathsf{app}(\mathsf{nil}, y) & \to & y & \mathsf{app}(\mathsf{add}(n, x), y) & \to & \mathsf{add}(n, \mathsf{app}(x, y))\\
\mathsf{reverse}(\mathsf{nil}) & \to & \mathsf{nil} & \mathsf{reverse}(\mathsf{add}(n, x)) & \to & \mathsf{app}(\mathsf{reverse}(x), \mathsf{add}(n, \mathsf{nil}))
\end{array}$$

Use interpretation $[\,\cdot\,]$ with $[\mathsf{Com}_k](x_1, \ldots, x_k) = x_1 + \cdots + x_k$ and

$[\mathsf{nil}] = 0$ $\quad$ $[\mathsf{add}](x_1, x_2) = x_2 + 1$ ($\leq$ restricted interpret.)

$[\mathsf{app}](x_1, x_2) = x_1 + x_2$ $\quad$ $[\mathsf{reverse}](x_1) = x_1$ (bounds helper fct. result size)

$[\mathsf{app}^\sharp](x_1, x_2) = x_1 + 1$ $\quad$ $[\mathsf{reverse}^\sharp](x_1) = x_1^2 + x_1 + 1$ (complexity of fct.)

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

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

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], ...

---

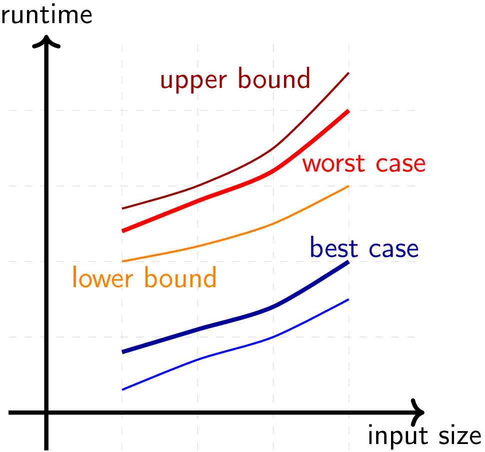
[28] N. Hirokawa, G. Moser: *Automated complexity analysis based on the dependency pair method*, IJCAR '08

[29] M. Avanzini, G. Moser: *Dependency pairs and polynomial path orders*, RTA '09

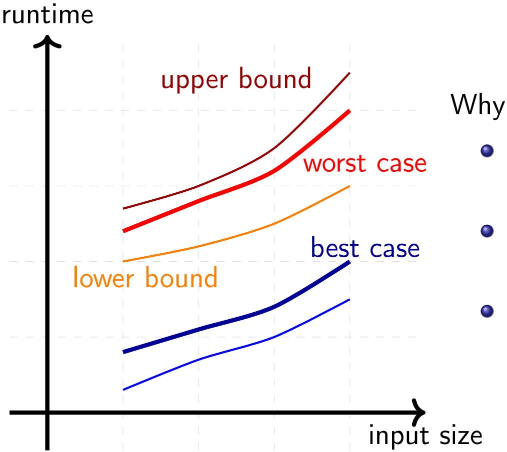
[30] N. Hirokawa, G. Moser: *Automated complexity analysis based on context-sensitive rewriting*, RTA-TLCA '14

[31] M. Avanzini, G. Moser: *A combination framework for complexity*, IC '16

How about Lower Bounds for Complexity?



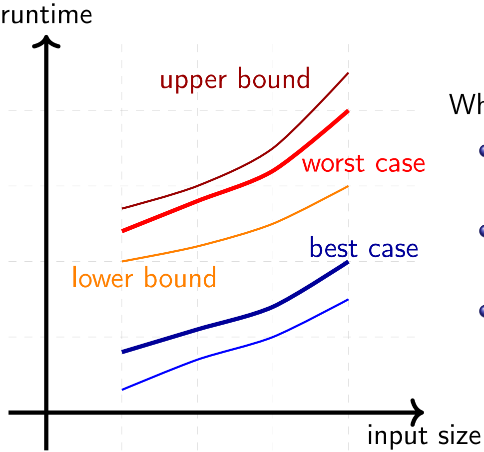
How about Lower Bounds for Complexity?



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[32] inspired by proving **non-termination**

---

[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]

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

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

  to conclude $\mathrm{rc}_{\mathcal{R}}(n) \in \Omega(p'(n))$.

---

[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[33]

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

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

  to conclude $\mathrm{rc}_{\mathcal{R}}(n) \in \Omega(p'(n))$.
- Constructor terms for arguments can be built recursively after type inference: $\mathsf{0}, \mathsf{s}(\mathsf{0}), \mathsf{s}(\mathsf{s}(\mathsf{0})), \ldots$ (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[33]

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

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

  to conclude $\mathrm{rc}_{\mathcal{R}}(n) \in \Omega(p'(n))$.
- Constructor terms for arguments can be built recursively after type inference: $0, \mathsf{s}(0), \mathsf{s}(\mathsf{s}(0)), \ldots$ (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

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

  to conclude $\mathrm{rc}_{\mathcal{R}}(n) \in \Omega(p'(n))$.
- Constructor terms for arguments can be built recursively after type inference: $0, \mathsf{s}(0), \mathsf{s}(\mathsf{s}(0)), \ldots$ (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

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

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

  to conclude $\text{rc}_{\mathcal{R}}(n) \in \Omega(p'(n))$.
- Constructor terms for arguments can be built recursively after type inference: $\mathsf{0}, \mathsf{s}(\mathsf{0}), \mathsf{s}(\mathsf{s}(\mathsf{0})), \ldots$ (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

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

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

  to conclude $\mathrm{rc}_{\mathcal{R}}(n) \in \Omega(p'(n))$.
- Constructor terms for arguments can be built recursively after type inference: $\mathsf{0}, \mathsf{s}(\mathsf{0}), \mathsf{s}(\mathsf{s}(\mathsf{0})), \ldots$ (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 $t_n \to_{\mathcal{R}}^{\geq p(n)} t'_n$ inductively
- Get lower bound for $\mathrm{rc}_{\mathcal{R}}(n)$ from $p(n)$ in rewrite lemma and $q(n)$

---

[33] F. Emmes, T. Enger, J. Giesl: *Proving non-looping non-termination automatically*, IJCAR '12

Finding Lower Bounds by Induction: Example

Example (quicksort)

$$
\begin{array}{rcl}
\mathsf{qs}(\mathsf{nil}) & \to & \mathsf{nil} \\
\mathsf{qs}(\mathsf{cons}(x, xs)) & \to & \mathsf{qs}(\mathsf{low}(x, xs)) \mathbin{++} \mathsf{cons}(x, \mathsf{qs}(\mathsf{low}(x, xs))) \\
\mathsf{low}(x, \mathsf{nil}) & \to & \mathsf{nil} \\
\mathsf{low}(x, \mathsf{cons}(y, ys)) & \to & \mathsf{if}(x \leq y, x, \mathsf{cons}(y, ys)) \\
\mathsf{if}(\mathsf{tt}, x, \mathsf{cons}(y, ys)) & \to & \mathsf{low}(x, ys) \\
\mathsf{if}(\mathsf{ff}, x, \mathsf{cons}(y, ys)) & \to & \mathsf{cons}(y, \mathsf{low}(x, ys)) \\
 & \ldots &
\end{array}
$$

Finding Lower Bounds by Induction: Example

Example (quicksort)

$$\begin{array}{rcl}
\mathsf{qs}(\mathsf{nil}) & \to & \mathsf{nil} \\
\mathsf{qs}(\mathsf{cons}(x, xs)) & \to & \mathsf{qs}(\mathsf{low}(x, xs)) \mathbin{++} \mathsf{cons}(x, \mathsf{qs}(\mathsf{low}(x, xs))) \\
\mathsf{low}(x, \mathsf{nil}) & \to & \mathsf{nil} \\
\mathsf{low}(x, \mathsf{cons}(y, ys)) & \to & \mathsf{if}(x \leq y, x, \mathsf{cons}(y, ys)) \\
\mathsf{if}(\mathsf{tt}, x, \mathsf{cons}(y, ys)) & \to & \mathsf{low}(x, ys) \\
\mathsf{if}(\mathsf{ff}, x, \mathsf{cons}(y, ys)) & \to & \mathsf{cons}(y, \mathsf{low}(x, ys)) \\
 & \dots &
\end{array}$$

Speculate and prove rewrite lemma:

$$\mathsf{qs}(\mathsf{cons}(\mathsf{zero}, \dots, \mathsf{cons}(\mathsf{zero}, \mathsf{nil}))) \to^{3n^2+2n+1} \mathsf{cons}(\mathsf{zero}, \dots, \mathsf{cons}(\mathsf{zero}, \mathsf{nil}))$$

Finding Lower Bounds by Induction: Example

Example (quicksort)

$$\begin{array}{rcl}
\mathsf{qs}(\mathsf{nil}) & \to & \mathsf{nil} \\
\mathsf{qs}(\mathsf{cons}(x, xs)) & \to & \mathsf{qs}(\mathsf{low}(x, xs)) \mathrel{++} \mathsf{cons}(x, \mathsf{qs}(\mathsf{low}(x, xs))) \\
\mathsf{low}(x, \mathsf{nil}) & \to & \mathsf{nil} \\
\mathsf{low}(x, \mathsf{cons}(y, ys)) & \to & \mathsf{if}(x \le y, x, \mathsf{cons}(y, ys)) \\
\mathsf{if}(\mathsf{tt}, x, \mathsf{cons}(y, ys)) & \to & \mathsf{low}(x, ys) \\
\mathsf{if}(\mathsf{ff}, x, \mathsf{cons}(y, ys)) & \to & \mathsf{cons}(y, \mathsf{low}(x, ys)) \\
 & \ldots &
\end{array}$$

Speculate and prove rewrite lemma:

$$\begin{array}{r}
\mathsf{qs}(\mathsf{cons}(\mathsf{zero}, \ldots, \mathsf{cons}(\mathsf{zero}, \mathsf{nil}))) \to^{3n^2+2n+1} \mathsf{cons}(\mathsf{zero}, \ldots, \mathsf{cons}(\mathsf{zero}, \mathsf{nil})) \\
\mathsf{qs}(\mathsf{cons}^n(\mathsf{zero}, \mathsf{nil})) \to^{3n^2+2n+1} \mathsf{cons}(\mathsf{zero}, \ldots, \mathsf{cons}(\mathsf{zero}, \mathsf{nil}))
\end{array}$$

Finding Lower Bounds by Induction: Example

Example (quicksort)

$$\begin{array}{rcl}
\mathsf{qs}(\mathsf{nil}) & \to & \mathsf{nil} \\
\mathsf{qs}(\mathsf{cons}(x, xs)) & \to & \mathsf{qs}(\mathsf{low}(x, xs)) \mathbin{++} \mathsf{cons}(x, \mathsf{qs}(\mathsf{low}(x, xs))) \\
\mathsf{low}(x, \mathsf{nil}) & \to & \mathsf{nil} \\
\mathsf{low}(x, \mathsf{cons}(y, ys)) & \to & \mathsf{if}(x \leq y, x, \mathsf{cons}(y, ys)) \\
\mathsf{if}(\mathsf{tt}, x, \mathsf{cons}(y, ys)) & \to & \mathsf{low}(x, ys) \\
\mathsf{if}(\mathsf{ff}, x, \mathsf{cons}(y, ys)) & \to & \mathsf{cons}(y, \mathsf{low}(x, ys)) \\
 & \ldots &
\end{array}$$

Speculate and prove rewrite lemma:

$$\mathsf{qs}(\mathsf{cons}(\mathsf{zero}, \ldots, \mathsf{cons}(\mathsf{zero}, \mathsf{nil}))) \to^{3n^2+2n+1} \mathsf{cons}(\mathsf{zero}, \ldots, \mathsf{cons}(\mathsf{zero}, \mathsf{nil}))$$

$$\mathsf{qs}(\mathsf{cons}^n(\mathsf{zero}, \mathsf{nil})) \to^{3n^2+2n+1} \mathsf{cons}(\mathsf{zero}, \ldots, \mathsf{cons}(\mathsf{zero}, \mathsf{nil}))$$

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

$$\mathrm{rc}_{\mathcal{R}}(2n + 2) \geq 3n^2 + 2n + 1$$

Finding Lower Bounds by Induction: Example

Example (quicksort)

$$\begin{array}{rcl}
\mathsf{qs}(\mathsf{nil}) & \to & \mathsf{nil} \\
\mathsf{qs}(\mathsf{cons}(x, xs)) & \to & \mathsf{qs}(\mathsf{low}(x, xs)) \mathrel{++} \mathsf{cons}(x, \mathsf{qs}(\mathsf{low}(x, xs))) \\
\mathsf{low}(x, \mathsf{nil}) & \to & \mathsf{nil} \\
\mathsf{low}(x, \mathsf{cons}(y, ys)) & \to & \mathsf{if}(x \leq y, x, \mathsf{cons}(y, ys)) \\
\mathsf{if}(\mathsf{tt}, x, \mathsf{cons}(y, ys)) & \to & \mathsf{low}(x, ys) \\
\mathsf{if}(\mathsf{ff}, x, \mathsf{cons}(y, ys)) & \to & \mathsf{cons}(y, \mathsf{low}(x, ys)) \\
 & \ldots &
\end{array}$$

Speculate and prove rewrite lemma:

$$\mathsf{qs}(\mathsf{cons}(\mathsf{zero}, \ldots, \mathsf{cons}(\mathsf{zero}, \mathsf{nil}))) \to^{3n^2+2n+1} \mathsf{cons}(\mathsf{zero}, \ldots, \mathsf{cons}(\mathsf{zero}, \mathsf{nil}))$$

$$\mathsf{qs}(\mathsf{cons}^n(\mathsf{zero}, \mathsf{nil})) \to^{3n^2+2n+1} \mathsf{cons}(\mathsf{zero}, \ldots, \mathsf{cons}(\mathsf{zero}, \mathsf{nil}))$$

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

$\mathrm{rc}_{\mathcal{R}}(2n + 2) \geq 3n^2 + 2n + 1$ and $\mathrm{rc}_{\mathcal{R}}(n) \in \Omega(n^2)$.

Finding Linear Lower Bounds by Decreasing Loops

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

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

Example: $\mathsf{f}(y) \to \mathsf{f}(\mathsf{s}(y))$ has loop $\mathsf{f}(y) \to_{\mathcal{R}}^{+} \mathsf{f}(\mathsf{s}(y))$ with $\sigma(y) = \mathsf{0}$.

Finding Linear Lower Bounds by Decreasing Loops

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

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

Example: $\mathsf{f}(y) \to \mathsf{f}(\mathsf{s}(y))$ has loop $\mathsf{f}(y) \to_{\mathcal{R}}^{+} \mathsf{f}(\mathsf{s}(y))$ with $\sigma(y) = \mathsf{0}$.

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)

Finding Linear Lower Bounds by Decreasing Loops

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

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

Example: $\mathsf{f}(y) \to \mathsf{f}(\mathsf{s}(y))$ has loop $\mathsf{f}(y) \to_{\mathcal{R}}^{+} \mathsf{f}(\mathsf{s}(y))$ with $\sigma(y) = \mathsf{0}$.

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: $\mathsf{plus}(\mathsf{s}(x), y) \to \mathsf{plus}(x, \mathsf{s}(y))$ has **decreasing** loop

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

Finding Linear Lower Bounds by Decreasing Loops

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

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

Example: $\mathsf{f}(y) \to \mathsf{f}(\mathsf{s}(y))$ has loop $\mathsf{f}(y) \to_{\mathcal{R}}^{+} \mathsf{f}(\mathsf{s}(y))$ with $\sigma(y) = \mathsf{0}$.

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: $\mathsf{plus}(\mathsf{s}(x), y) \to \mathsf{plus}(x, \mathsf{s}(y))$ has **decreasing** loop

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

for *base term* $s = \mathsf{plus}(x, y)$, *pumping substitution* $\theta = [x \mapsto \mathsf{s}(x)]$, and *result substitution* $\sigma = [y \mapsto \mathsf{s}(y)]$:

$$s\theta \to_{\mathcal{R}}^{+} C[s\sigma]$$

Implies $\mathrm{rc}(n) \in \Omega(n)$!

Finding Exponential Lower Bounds by Decreasing Loops

Exponential lower bounds: several "compatible" parallel recursive calls:

- **Example:** $\mathsf{fib}(\mathsf{s}(\mathsf{s}(n))) \to \mathsf{plus}(\mathsf{fib}(\mathsf{s}(n)), \mathsf{fib}(n))$ has 2 decreasing loops:

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

  Implies $\mathrm{rc}(n) \in \Omega(2^n)$!

Finding Exponential Lower Bounds by Decreasing Loops

Exponential lower bounds: several "compatible" parallel recursive calls:

- **Example:** $\mathsf{fib}(\mathsf{s}(\mathsf{s}(n))) \to \mathsf{plus}(\mathsf{fib}(\mathsf{s}(n)), \mathsf{fib}(n))$ has 2 decreasing loops:

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

  Implies $\mathrm{rc}(n) \in \Omega(2^n)$!

- **(Non-)Example:** $\mathsf{tr}(\mathsf{node}(x, y)) \to \mathsf{node}(\mathsf{tr}(x), \mathsf{tr}(y))$

  Has **linear** complexity. But:

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

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

Finding Exponential Lower Bounds by Decreasing Loops

Exponential lower bounds: several "compatible" parallel recursive calls:

- **Example:** $\mathsf{fib}(\mathsf{s}(\mathsf{s}(n))) \to \mathsf{plus}(\mathsf{fib}(\mathsf{s}(n)), \mathsf{fib}(n))$ has 2 decreasing loops:

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

  Implies $\mathrm{rc}(n) \in \Omega(2^n)$!

- **(Non-)Example:** $\mathsf{tr}(\mathsf{node}(x, y)) \to \mathsf{node}(\mathsf{tr}(x), \mathsf{tr}(y))$

  Has **linear** complexity. But:

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

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

Automation for decreasing loops: **narrowing**.

Lower Bounds: Induction Technique vs Decreasing Loops

Benefits of Induction Technique:

- Can find **non-linear** polynomial lower bounds
- Also works on non-left-linear TRSs

Lower Bounds: Induction Technique vs Decreasing Loops

Benefits of Induction Technique:

- Can find **non-linear** polynomial lower bounds
- Also works on non-left-linear TRSs

Benefits of Decreasing Loops:

- Does not rely as much on heuristics
- Computationally more lightweight

Lower Bounds: Induction Technique vs Decreasing Loops

Benefits of Induction Technique:

- Can find **non-linear** polynomial lower bounds
- Also works on non-left-linear TRSs

Benefits of Decreasing Loops:

- Does not rely as much on heuristics
- Computationally more lightweight

⇒ First try decreasing loops, then induction technique

Lower Bounds: Induction Technique vs Decreasing Loops

Benefits of Induction Technique:

- Can find **non-linear** polynomial lower bounds
- Also works on non-left-linear TRSs

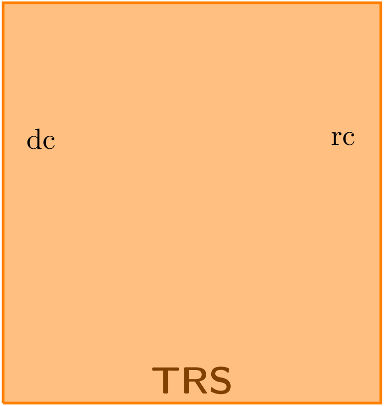
Benefits of Decreasing Loops:

- Does not rely as much on heuristics
- Computationally more lightweight

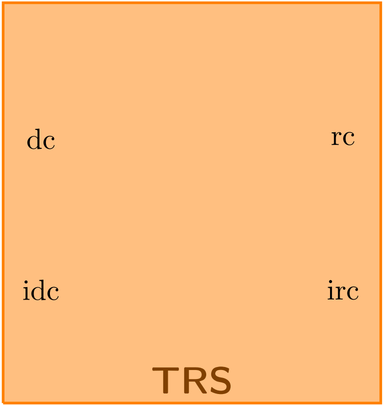
⇒ First try decreasing loops, then induction technique

Both techniques can be adapted to innermost runtime complexity!

A Landscape of Complexity Properties and Transformations

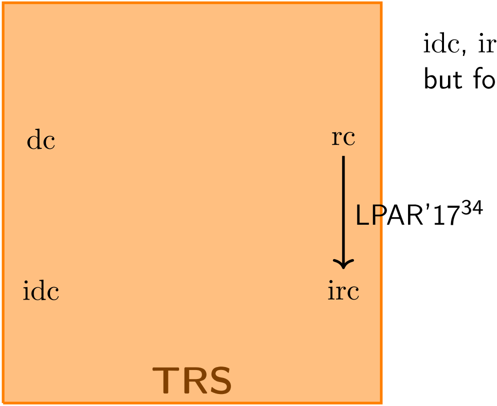


A Landscape of Complexity Properties and Transformations



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

A Landscape of Complexity Properties and Transformations

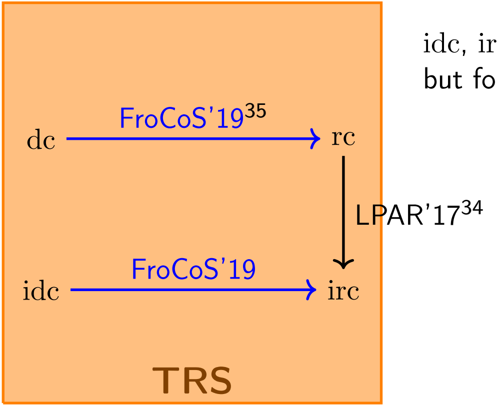


idc, irc: like dc, rc,
but for *innermost* rewriting

---

[34] F. Frohn, J. Giesl: *Analyzing runtime complexity via innermost runtime complexity*, LPAR '17

A Landscape of Complexity Properties and Transformations



$$\begin{tikzcd}
\mathrm{dc} \arrow[r, "\text{FroCoS'19}^{35}"] & \mathrm{rc} \arrow[d, "\text{LPAR'17}^{34}"] \\
\mathrm{idc} \arrow[r, "\text{FroCoS'19}"] & \mathrm{irc}
\end{tikzcd}$$

TRS

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

---

[34] F. Frohn, J. Giesl: *Analyzing runtime complexity via innermost runtime complexity*, LPAR '17

[35] C. Fuhs: *Transforming Derivational Complexity of Term Rewriting to Runtime Complexity*, FroCoS '19

Transforming Derivational Complexity to Runtime Complexity

The big picture:

- **Have:** Tool for automated analysis of runtime complexity $\mathrm{rc}_{\mathcal{R}}$

Transforming Derivational Complexity to Runtime Complexity

The big picture:

- **Have:** Tool for automated analysis of runtime complexity $\mathrm{rc}_{\mathcal{R}}$
- **Want:** Tool for automated analysis of derivational complexity $\mathrm{dc}_{\mathcal{R}}$

Transforming Derivational Complexity to Runtime Complexity

The big picture:

- **Have:** Tool for automated analysis of runtime complexity $\mathrm{rc}_{\mathcal{R}}$
- **Want:** Tool for automated analysis of derivational complexity $\mathrm{dc}_{\mathcal{R}}$
- **Idea:**

  "$\mathrm{rc}_{\mathcal{R}}$ analysis tool + transformation on TRS $\mathcal{R}$ = $\mathrm{dc}_{\mathcal{R}}$ analysis tool"

Transforming Derivational Complexity to Runtime Complexity

The big picture:

- **Have:** Tool for automated analysis of runtime complexity $\mathrm{rc}_{\mathcal{R}}$
- **Want:** Tool for automated analysis of derivational complexity $\mathrm{dc}_{\mathcal{R}}$
- **Idea:**

  "$\mathrm{rc}_{\mathcal{R}}$ analysis tool + transformation on TRS $\mathcal{R}$ = $\mathrm{dc}_{\mathcal{R}}$ analysis tool"
- **Benefits:**
  - Get analysis of derivational complexity "for free"
  - Progress in runtime complexity analysis automatically improves derivational complexity analysis

From dc to rc: Results

- program transformation such that runtime complexity of transformed TRS is **identical** to derivational complexity of original TRS

From dc to rc: Results

- program transformation such that runtime complexity of transformed TRS is **identical** to derivational complexity of original TRS
- transformation correct also from idc to irc

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

From dc to rc: Results

- program transformation such that runtime complexity of transformed TRS is **identical** to derivational complexity of original TRS
- transformation correct also from idc to irc
- **implemented** in program analysis tool AProVE
- **evaluated** successfully on TPDB[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`

From dc to rc: Transformation

Issue:

- Runtime complexity assumes **basic** terms as start terms
- We want to analyse complexity for **arbitrary** terms

From dc to rc: Transformation

Issue:

- Runtime complexity assumes **basic** terms as start terms
- We want to analyse complexity for **arbitrary** terms

Idea:

- Introduce constructor symbol c_f for defined symbol f

From dc to rc: Transformation

Issue:

- Runtime complexity assumes **basic** terms as start terms
- We want to analyse complexity for **arbitrary** terms

Idea:

- Introduce constructor symbol c_f for defined symbol f
- Add **generator rewrite rules** $\mathcal{G}$ 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** $\mathcal{G}$ to reconstruct arbitrary term with f from basic term with c_f

Represent

$t = \mathsf{double}(\mathsf{double}(\mathsf{double}(\mathsf{s}(\mathsf{0}))))$

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** $\mathcal{G}$ to reconstruct arbitrary term with f from basic term with c_f

Represent

$$t = \mathsf{double}(\mathsf{double}(\mathsf{double}(\mathsf{s}(0))))$$

by **basic variant**

$$\mathrm{bv}(t) = \mathsf{enc}_{\mathsf{double}}(\mathsf{c}_{\mathsf{double}}(\mathsf{c}_{\mathsf{double}}(\mathsf{s}(0))))$$

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** $\mathcal{G}$ to reconstruct arbitrary term with f from basic term with c_f

Represent

$$t = \mathsf{double}(\mathsf{double}(\mathsf{double}(\mathsf{s}(\mathsf{0}))))$$

by **basic variant**

$$\mathrm{bv}(t) = \mathsf{enc}_{\mathsf{double}}(\mathsf{c}_{\mathsf{double}}(\mathsf{c}_{\mathsf{double}}(\mathsf{s}(\mathsf{0}))))$$

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

$$\begin{aligned}
\mathsf{enc}_{\mathsf{double}}(x) &\to \mathsf{double}(\mathsf{argenc}(x)) \\
\mathsf{enc}_{\mathsf{0}} &\to \mathsf{0} \\
\mathsf{enc}_{\mathsf{s}}(x) &\to \mathsf{s}(\mathsf{argenc}(x)) \\
\mathsf{argenc}(\mathsf{c}_{\mathsf{double}}(x)) &\to \mathsf{double}(\mathsf{argenc}(x)) \\
\mathsf{argenc}(\mathsf{0}) &\to \mathsf{0} \\
\mathsf{argenc}(\mathsf{s}(x)) &\to \mathsf{s}(\mathsf{argenc}(x))
\end{aligned}$$

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

$$t = \mathsf{double}(\mathsf{double}(\mathsf{double}(\mathsf{s}(\mathsf{0}))))$$

by **basic variant**

$$\mathrm{bv}(t) = \mathsf{enc}_{\mathsf{double}}(\mathsf{c}_{\mathsf{double}}(\mathsf{c}_{\mathsf{double}}(\mathsf{s}(\mathsf{0}))))$$

Then:

- $\mathrm{bv}(t)$ is **basic** term, size $|t|$

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

$$\begin{aligned}
\mathsf{enc}_{\mathsf{double}}(x) &\to \mathsf{double}(\mathsf{argenc}(x)) \\
\mathsf{enc}_{\mathsf{0}} &\to \mathsf{0} \\
\mathsf{enc}_{\mathsf{s}}(x) &\to \mathsf{s}(\mathsf{argenc}(x)) \\
\mathsf{argenc}(\mathsf{c}_{\mathsf{double}}(x)) &\to \mathsf{double}(\mathsf{argenc}(x)) \\
\mathsf{argenc}(\mathsf{0}) &\to \mathsf{0} \\
\mathsf{argenc}(\mathsf{s}(x)) &\to \mathsf{s}(\mathsf{argenc}(x))
\end{aligned}$$

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

$$t = \mathsf{double}(\mathsf{double}(\mathsf{double}(\mathsf{s}(\mathsf{0}))))$$

by **basic variant**

$$\mathrm{bv}(t) = \mathsf{enc}_{\mathsf{double}}(\mathsf{c}_{\mathsf{double}}(\mathsf{c}_{\mathsf{double}}(\mathsf{s}(\mathsf{0}))))$$

Then:

- $\mathrm{bv}(t)$ is **basic** term, size $|t|$
- $\mathrm{bv}(t) \to_{\mathcal{G}}^{*} t$

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

$$\begin{aligned}
\mathsf{enc}_{\mathsf{double}}(x) &\to \mathsf{double}(\mathsf{argenc}(x))\\
\mathsf{enc}_{\mathsf{0}} &\to \mathsf{0}\\
\mathsf{enc}_{\mathsf{s}}(x) &\to \mathsf{s}(\mathsf{argenc}(x))\\
\mathsf{argenc}(\mathsf{c}_{\mathsf{double}}(x)) &\to \mathsf{double}(\mathsf{argenc}(x))\\
\mathsf{argenc}(\mathsf{0}) &\to \mathsf{0}\\
\mathsf{argenc}(\mathsf{s}(x)) &\to \mathsf{s}(\mathsf{argenc}(x))
\end{aligned}$$

General Case: Relative Rewriting

Issue:

- $\rightarrow_{\mathcal{R}\cup\mathcal{G}}$ has extra rewrite steps not present in $\rightarrow_{\mathcal{R}}$
- may change complexity

General Case: Relative Rewriting

Issue:

- $\to_{\mathcal{R}\cup\mathcal{G}}$ has extra rewrite steps not present in $\to_{\mathcal{R}}$
- may change complexity

Solution:

- add $\mathcal{G}$ as **relative** rewrite rules:
  $\to_{\mathcal{G}}$ steps are **not counted** for complexity analysis!
- transform $\mathcal{R}$ to $\mathcal{R}/\mathcal{G}$ ($\to_{\mathcal{R}}$ steps are counted, $\to_{\mathcal{G}}$ steps are not)

General Case: Relative Rewriting

Issue:

- $\to_{\mathcal{R}\cup\mathcal{G}}$ has extra rewrite steps not present in $\to_{\mathcal{R}}$
- may change complexity

Solution:

- add $\mathcal{G}$ as **relative** rewrite rules:
  $\to_{\mathcal{G}}$ steps are **not counted** for complexity analysis!
- transform $\mathcal{R}$ to $\mathcal{R}/\mathcal{G}$ ($\to_{\mathcal{R}}$ steps are counted, $\to_{\mathcal{G}}$ steps are not)
- more generally: transform $\mathcal{R}/\mathcal{S}$ to $\mathcal{R}/(\mathcal{S} \cup \mathcal{G})$
  (input may contain relative rules $\mathcal{S}$, too)

From dc to rc: Correctness

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. $\mathrm{dc}_{\mathcal{R}/\mathcal{S}}(n) = \mathrm{rc}_{\mathcal{R}/(\mathcal{S}\cup\mathcal{G})}(n)$ *(arbitrary rewrite strategies)*
2. $\mathrm{idc}_{\mathcal{R}/\mathcal{S}}(n) = \mathrm{irc}_{\mathcal{R}/(\mathcal{S}\cup\mathcal{G})}(n)$ *(innermost rewriting)*

Note: equalities hold also non-asymptotically!

From (i)dc to (i)rc: Experiments

Experiments on TPDB, compare with state of the art in TcT:

- upper bounds idc: both AProVE and TcT with transformation are stronger than standard TcT
- upper bounds dc: TcT stronger than AProVE and TcT with transformation, but AProVE still solves some new examples
- lower bounds idc and dc: heuristics do not seem to benefit much

From (i)dc to (i)rc: Experiments

Experiments on TPDB, compare with state of the art in TcT:

- upper bounds idc: both AProVE and TcT with transformation are stronger than standard TcT
- upper bounds dc: TcT stronger than AProVE and TcT with transformation, but AProVE still solves some new examples
- lower bounds idc and dc: heuristics do not seem to benefit much

⇒ 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 $\mathrm{dc}_{\mathcal{R}}$ is appropriate

Derivational Complexity: Future Work

- **Possible applications**
  - compiler simplifications
  - SMT solver preprocessing

  Start terms may have nested defined symbols, so $\mathrm{dc}_{\mathcal{R}}$ is appropriate

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

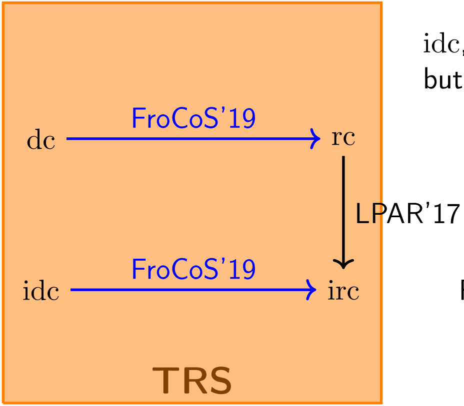
Derivational Complexity: Future Work

- **Possible applications**
  - compiler simplifications
  - SMT solver preprocessing

  Start terms may have nested defined symbols, so $\mathrm{dc}_{\mathcal{R}}$ is appropriate
- Go **between** derivational and runtime complexity
  - So far: encode *full* term universe $\mathcal{T}$ via basic terms $\mathcal{T}_{\mathrm{basic}}$
  - Generalise: write relative rules to generate **arbitrary** set $\mathcal{U}$ of terms "between" basic and all terms ($\mathcal{T}_{\mathrm{basic}} \subseteq \mathcal{U} \subseteq \mathcal{T}$).
- Want to adapt **techniques** from runtime complexity analysis to derivational complexity! How?
  - (Useful) adaptation of Dependency Pairs?
  - Abstractions to numbers?
  - ...

A Landscape of Complexity Properties and Transformations

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



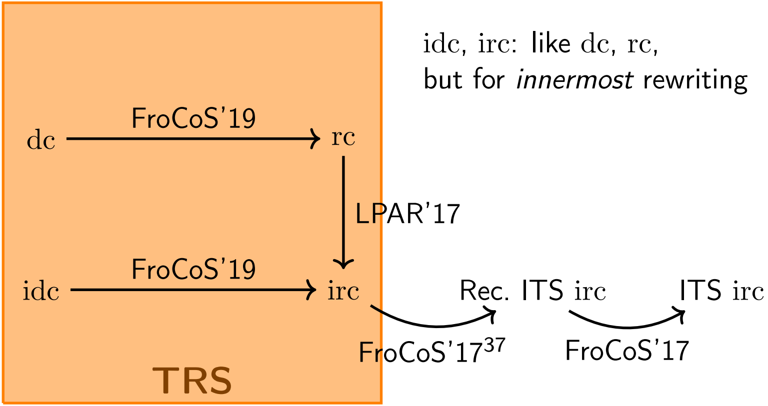
$$\begin{tikzcd}
\mathrm{dc} \arrow[r, "\text{FroCoS'19}"] & \mathrm{rc} \arrow[d, "\text{LPAR'17}"] \\
\mathrm{idc} \arrow[r, "\text{FroCoS'19}"] & \mathrm{irc}
\end{tikzcd}$$

TRS

Rec. ITS irc

ITS irc

A Landscape of Complexity Properties and Transformations



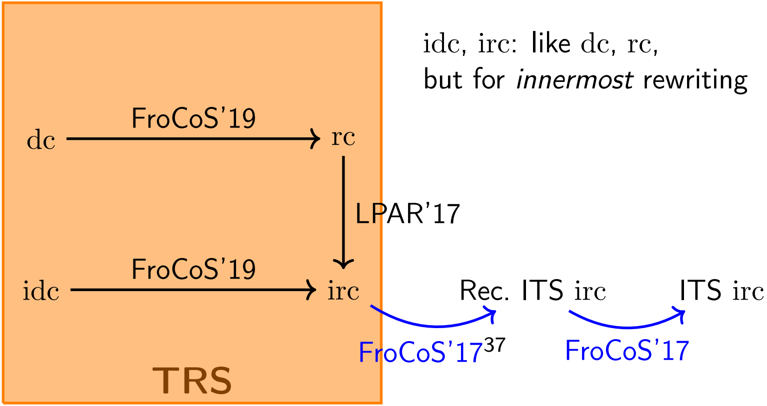
idc, irc: like dc, rc,
but for *innermost* rewriting

---

[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



$$\begin{tikzcd}
\mathrm{dc} \arrow[r, "\text{FroCoS'19}"] & \mathrm{rc} \arrow[d, "\text{LPAR'17}"] & & \\
\mathrm{idc} \arrow[r, "\text{FroCoS'19}"] & \mathrm{irc} \arrow[r, bend right, "\text{FroCoS'17}^{37}"'] & \text{Rec. ITS } \mathrm{irc} \arrow[r, bend right, "\text{FroCoS'17}"'] & \text{ITS } \mathrm{irc}
\end{tikzcd}$$

---

[37] M. Naaf, F. Frohn, M. Brockschmidt, C. Fuhs, J. Giesl: *Complexity analysis for term rewriting by integer transition systems*, FroCoS '17

Bottom-Up Complexity Analysis for Imperative Programs

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

---

[38] A. Flores-Montoya, R. Hähnle: *Resource analysis of complex programs with cost equations*, APLAS '14, `https://github.com/aeflores/CoFloCo`

[39] M. Brockschmidt, F. Emmes, S. Falke, C. Fuhs, J. Giesl: *Analyzing Runtime and Size Complexity of Integer Programs*, TOPLAS '16, `https://github.com/s-falke/kittel-koat`

[40] E. Albert, P. Arenas, S. Genaim, G. Puebla: *Closed-Form Upper Bounds in Static Cost Analysis*, JAR '11, `https://costa.fdi.ucm.es/pubs/`

Analysing irc of Insertion Sort by Hand: Bottom-Up

Example

$$\begin{aligned}
\mathsf{isort}(\mathsf{nil}, ys) &\quad\longrightarrow\quad ys \\
\mathsf{isort}(\mathsf{cons}(x, xs), ys) &\quad\longrightarrow\quad \mathsf{isort}(xs, \mathsf{insert}(x, ys)) \\
\mathsf{insert}(x, \mathsf{nil}) &\quad\longrightarrow\quad \mathsf{cons}(x, \mathsf{nil}) \\
\mathsf{insert}(x, \mathsf{cons}(y, ys)) &\quad\longrightarrow\quad \mathsf{if}(\mathsf{gt}(x, y), x, \mathsf{cons}(y, ys)) \\
\mathsf{if}(\mathsf{true}, x, \mathsf{cons}(y, ys)) &\quad\longrightarrow\quad \mathsf{cons}(y, \mathsf{insert}(x, ys)) \\
\mathsf{if}(\mathsf{false}, x, \mathsf{cons}(y, ys)) &\quad\longrightarrow\quad \mathsf{cons}(x, \mathsf{cons}(y, ys)) \\
\mathsf{gt}(\mathsf{0}, y) &\quad\xrightarrow{=}\quad \mathsf{false} \\
\mathsf{gt}(\mathsf{s}(x), \mathsf{0}) &\quad\xrightarrow{=}\quad \mathsf{true} \\
\mathsf{gt}(\mathsf{s}(x), \mathsf{s}(y)) &\quad\xrightarrow{=}\quad \mathsf{gt}(x, y)
\end{aligned}$$

Analysing irc of Insertion Sort by Hand: Bottom-Up

Example

$$\begin{array}{rcl}
\mathsf{isort}(\mathsf{nil}, ys) & \longrightarrow & ys \\
\mathsf{isort}(\mathsf{cons}(x, xs), ys) & \longrightarrow & \mathsf{isort}(xs, \mathsf{insert}(x, ys)) \\
\mathsf{insert}(x, \mathsf{nil}) & \longrightarrow & \mathsf{cons}(x, \mathsf{nil}) \\
\mathsf{insert}(x, \mathsf{cons}(y, ys)) & \longrightarrow & \mathsf{if}(\mathsf{gt}(x, y), x, \mathsf{cons}(y, ys)) \\
\mathsf{if}(\mathsf{true}, x, \mathsf{cons}(y, ys)) & \longrightarrow & \mathsf{cons}(y, \mathsf{insert}(x, ys)) \\
\mathsf{if}(\mathsf{false}, x, \mathsf{cons}(y, ys)) & \longrightarrow & \mathsf{cons}(x, \mathsf{cons}(y, ys)) \\
\mathsf{gt}(\mathsf{0}, y) & \xrightarrow{=} & \mathsf{false} \\
\mathsf{gt}(\mathsf{s}(x), \mathsf{0}) & \xrightarrow{=} & \mathsf{true} \\
\mathsf{gt}(\mathsf{s}(x), \mathsf{s}(y)) & \xrightarrow{=} & \mathsf{gt}(x, y)
\end{array}$$

Note: innermost reduction strategy

Analysing irc of Insertion Sort by Hand: Bottom-Up

Example

$$\begin{array}{rcl}
\mathsf{isort}(\mathsf{nil}, ys) & \longrightarrow & ys \\
\mathsf{isort}(\mathsf{cons}(x, xs), ys) & \longrightarrow & \mathsf{isort}(xs, \mathsf{insert}(x, ys)) \\
\mathsf{insert}(x, \mathsf{nil}) & \longrightarrow & \mathsf{cons}(x, \mathsf{nil}) \\
\mathsf{insert}(x, \mathsf{cons}(y, ys)) & \longrightarrow & \mathsf{if}(\mathsf{gt}(x, y), x, \mathsf{cons}(y, ys)) \\
\mathsf{if}(\mathsf{true}, x, \mathsf{cons}(y, ys)) & \longrightarrow & \mathsf{cons}(y, \mathsf{insert}(x, ys)) \\
\mathsf{if}(\mathsf{false}, x, \mathsf{cons}(y, ys)) & \longrightarrow & \mathsf{cons}(x, \mathsf{cons}(y, ys)) \\
\mathsf{gt}(\mathsf{0}, y) & \xrightarrow{=} & \mathsf{false} \\
\mathsf{gt}(\mathsf{s}(x), \mathsf{0}) & \xrightarrow{=} & \mathsf{true} \\
\mathsf{gt}(\mathsf{s}(x), \mathsf{s}(y)) & \xrightarrow{=} & \mathsf{gt}(x, y)
\end{array}$$

- $\mathsf{rt}(\mathsf{gt}(x, y)) \in \mathcal{O}(1)$ ("$\xrightarrow{=}$" for relative rules)

Note: innermost reduction strategy

Analysing irc of Insertion Sort by Hand: Bottom-Up

Example

$$\begin{aligned}
\mathsf{isort}(\mathsf{nil}, ys) &\longrightarrow ys \\
\mathsf{isort}(\mathsf{cons}(x, xs), ys) &\longrightarrow \mathsf{isort}(xs, \mathsf{insert}(x, ys)) \\
\mathsf{insert}(x, \mathsf{nil}) &\longrightarrow \mathsf{cons}(x, \mathsf{nil}) \\
\mathsf{insert}(x, \mathsf{cons}(y, ys)) &\longrightarrow \mathsf{if}(\mathsf{gt}(x, y), x, \mathsf{cons}(y, ys)) \\
\mathsf{if}(\mathsf{true}, x, \mathsf{cons}(y, ys)) &\longrightarrow \mathsf{cons}(y, \mathsf{insert}(x, ys)) \\
\mathsf{if}(\mathsf{false}, x, \mathsf{cons}(y, ys)) &\longrightarrow \mathsf{cons}(x, \mathsf{cons}(y, ys)) \\
\mathsf{gt}(\mathsf{0}, y) &\xrightarrow{=} \mathsf{false} \\
\mathsf{gt}(\mathsf{s}(x), \mathsf{0}) &\xrightarrow{=} \mathsf{true} \\
\mathsf{gt}(\mathsf{s}(x), \mathsf{s}(y)) &\xrightarrow{=} \mathsf{gt}(x, y)
\end{aligned}$$

- $\mathsf{rt}(\mathsf{gt}(x, y)) \in \mathcal{O}(1)$ ("$\xrightarrow{=}$" for relative rules)
- $\mathsf{rt}(\mathsf{insert}(x, ys)) \in \mathcal{O}(\mathsf{length}(ys))$

Note: innermost reduction strategy

Analysing irc of Insertion Sort by Hand: Bottom-Up

Example

$$\begin{aligned}
\mathsf{isort}(\mathsf{nil}, ys) &\to ys \\
\mathsf{isort}(\mathsf{cons}(x, xs), ys) &\to \mathsf{isort}(xs, \mathsf{insert}(x, ys)) \\
\mathsf{insert}(x, \mathsf{nil}) &\to \mathsf{cons}(x, \mathsf{nil}) \\
\mathsf{insert}(x, \mathsf{cons}(y, ys)) &\to \mathsf{if}(\mathsf{gt}(x, y), x, \mathsf{cons}(y, ys)) \\
\mathsf{if}(\mathsf{true}, x, \mathsf{cons}(y, ys)) &\to \mathsf{cons}(y, \mathsf{insert}(x, ys)) \\
\mathsf{if}(\mathsf{false}, x, \mathsf{cons}(y, ys)) &\to \mathsf{cons}(x, \mathsf{cons}(y, ys)) \\
\mathsf{gt}(\mathsf{0}, y) &\xrightarrow{=} \mathsf{false} \\
\mathsf{gt}(\mathsf{s}(x), \mathsf{0}) &\xrightarrow{=} \mathsf{true} \\
\mathsf{gt}(\mathsf{s}(x), \mathsf{s}(y)) &\xrightarrow{=} \mathsf{gt}(x, y)
\end{aligned}$$

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

Note: innermost reduction strategy

Analysing irc of Insertion Sort by Hand: Bottom-Up

Example

$$\begin{array}{rcl}
\mathsf{isort}(\mathsf{nil}, ys) & \to & ys \\
\mathsf{isort}(\mathsf{cons}(x, xs), ys) & \to & \mathsf{isort}(xs, \mathsf{insert}(x, ys)) \\
\mathsf{insert}(x, \mathsf{nil}) & \to & \mathsf{cons}(x, \mathsf{nil}) \\
\mathsf{insert}(x, \mathsf{cons}(y, ys)) & \to & \mathsf{if}(\mathsf{gt}(x, y), x, \mathsf{cons}(y, ys)) \\
\mathsf{if}(\mathsf{true}, x, \mathsf{cons}(y, ys)) & \to & \mathsf{cons}(y, \mathsf{insert}(x, ys)) \\
\mathsf{if}(\mathsf{false}, x, \mathsf{cons}(y, ys)) & \to & \mathsf{cons}(x, \mathsf{cons}(y, ys)) \\
\mathsf{gt}(0, y) & \xrightarrow{=} & \mathsf{false} \\
\mathsf{gt}(\mathsf{s}(x), 0) & \xrightarrow{=} & \mathsf{true} \\
\mathsf{gt}(\mathsf{s}(x), \mathsf{s}(y)) & \xrightarrow{=} & \mathsf{gt}(x, y)
\end{array}$$

- $\mathsf{rt}(\mathsf{gt}(x, y)) \in \mathcal{O}(1)$ ("$\xrightarrow{=}$" for relative rules)
- $\mathsf{rt}(\mathsf{insert}(x, ys)) \in \mathcal{O}(\mathsf{length}(ys))$
- $\mathsf{rt}(\mathsf{isort}(xs, ys)) \in \mathcal{O}(\mathsf{length}(xs) \cdot (\mathsf{length}(xs) + \mathsf{length}(ys)))$

Note: innermost reduction strategy

Using Dependency Tuples: Top-Down

Example

$$\begin{aligned}
\mathsf{isort}(\mathsf{nil}, ys) &\longrightarrow ys \\
\mathsf{isort}(\mathsf{cons}(x, xs), ys) &\longrightarrow \mathsf{isort}(xs, \mathsf{insert}(x, ys)) \\
\mathsf{insert}(x, \mathsf{nil}) &\longrightarrow \mathsf{cons}(x, \mathsf{nil}) \\
\mathsf{insert}(x, \mathsf{cons}(y, ys)) &\longrightarrow \mathsf{if}(\mathsf{gt}(x, y), x, \mathsf{cons}(y, ys)) \\
\mathsf{if}(\mathsf{true}, x, \mathsf{cons}(y, ys)) &\longrightarrow \mathsf{cons}(y, \mathsf{insert}(x, ys)) \\
\mathsf{if}(\mathsf{false}, x, \mathsf{cons}(y, ys)) &\longrightarrow \mathsf{cons}(x, \mathsf{cons}(y, ys)) \\
\mathsf{gt}(\mathsf{0}, y) &\xrightarrow{=} \mathsf{false} \\
\mathsf{gt}(\mathsf{s}(x), \mathsf{0}) &\xrightarrow{=} \mathsf{true} \\
\mathsf{gt}(\mathsf{s}(x), \mathsf{s}(y)) &\xrightarrow{=} \mathsf{gt}(x, y)
\end{aligned}$$

- the recursive isort rule is at most applied linearly often

Using Dependency Tuples: Top-Down

Example

$$
\begin{array}{rcl}
\mathsf{isort}(\mathsf{nil}, ys) & \longrightarrow & ys \\
\mathsf{isort}(\mathsf{cons}(x, xs), ys) & \longrightarrow & \mathsf{isort}(xs, \mathsf{insert}(x, ys)) \\
\mathsf{insert}(x, \mathsf{nil}) & \longrightarrow & \mathsf{cons}(x, \mathsf{nil}) \\
\mathsf{insert}(x, \mathsf{cons}(y, ys)) & \longrightarrow & \mathsf{if}(\mathsf{gt}(x, y), x, \mathsf{cons}(y, ys)) \\
\mathsf{if}(\mathsf{true}, x, \mathsf{cons}(y, ys)) & \longrightarrow & \mathsf{cons}(y, \mathsf{insert}(x, ys)) \\
\mathsf{if}(\mathsf{false}, x, \mathsf{cons}(y, ys)) & \longrightarrow & \mathsf{cons}(x, \mathsf{cons}(y, ys)) \\
\mathsf{gt}(\mathsf{0}, y) & \xrightarrow{=} & \mathsf{false} \\
\mathsf{gt}(\mathsf{s}(x), \mathsf{0}) & \xrightarrow{=} & \mathsf{true} \\
\mathsf{gt}(\mathsf{s}(x), \mathsf{s}(y)) & \xrightarrow{=} & \mathsf{gt}(x, y)
\end{array}
$$

- the recursive isort rule is at most applied linearly often
- the recursive insert rule is at most applied quadratically often

Using Dependency Tuples: Top-Down

Example

$$\begin{array}{rcl}
\mathsf{isort}(\mathsf{nil}, ys) & \to & ys \\
\mathsf{isort}(\mathsf{cons}(x, xs), ys) & \to & \mathsf{isort}(xs, \mathsf{insert}(x, ys)) \\
\mathsf{insert}(x, \mathsf{nil}) & \to & \mathsf{cons}(x, \mathsf{nil}) \\
\mathsf{insert}(x, \mathsf{cons}(y, ys)) & \to & \mathsf{if}(\mathsf{gt}(x, y), x, \mathsf{cons}(y, ys)) \\
\mathsf{if}(\mathsf{true}, x, \mathsf{cons}(y, ys)) & \to & \mathsf{cons}(y, \mathsf{insert}(x, ys)) \\
\mathsf{if}(\mathsf{false}, x, \mathsf{cons}(y, ys)) & \to & \mathsf{cons}(x, \mathsf{cons}(y, ys)) \\
\mathsf{gt}(\mathsf{0}, y) & \xrightarrow{=} & \mathsf{false} \\
\mathsf{gt}(\mathsf{s}(x), \mathsf{0}) & \xrightarrow{=} & \mathsf{true} \\
\mathsf{gt}(\mathsf{s}(x), \mathsf{s}(y)) & \xrightarrow{=} & \mathsf{gt}(x, y)
\end{array}$$

- 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!

Using Dependency Tuples: Top-Down

Example

$$\begin{array}{rcl}
\mathsf{isort}(\mathsf{nil}, ys) & \to & ys \\
\mathsf{isort}(\mathsf{cons}(x, xs), ys) & \to & \mathsf{isort}(xs, \mathsf{insert}(x, ys)) \\
\mathsf{insert}(x, \mathsf{nil}) & \to & \mathsf{cons}(x, \mathsf{nil}) \\
\mathsf{insert}(x, \mathsf{cons}(y, ys)) & \to & \mathsf{if}(\mathsf{gt}(x, y), x, \mathsf{cons}(y, ys)) \\
\mathsf{if}(\mathsf{true}, x, \mathsf{cons}(y, ys)) & \to & \mathsf{cons}(y, \mathsf{insert}(x, ys)) \\
\mathsf{if}(\mathsf{false}, x, \mathsf{cons}(y, ys)) & \to & \mathsf{cons}(x, \mathsf{cons}(y, ys)) \\
\mathsf{gt}(\mathsf{0}, y) & \xrightarrow{=} & \mathsf{false} \\
\mathsf{gt}(\mathsf{s}(x), \mathsf{0}) & \xrightarrow{=} & \mathsf{true} \\
\mathsf{gt}(\mathsf{s}(x), \mathsf{s}(y)) & \xrightarrow{=} & \mathsf{gt}(x, y)
\end{array}$$

- 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

$$\begin{array}{rcl}
\mathsf{isort}(\mathsf{nil}, ys) & \longrightarrow & ys \\
\mathsf{isort}(\mathsf{cons}(x, xs), ys) & \longrightarrow & \mathsf{isort}(xs, \mathsf{insert}(x, ys)) \\
\mathsf{insert}(x, \mathsf{nil}) & \longrightarrow & \mathsf{cons}(x, \mathsf{nil}) \\
\mathsf{insert}(x, \mathsf{cons}(y, ys)) & \longrightarrow & \mathsf{if}(\mathsf{gt}(x, y), x, \mathsf{cons}(y, ys)) \\
\mathsf{if}(\mathsf{true}, x, \mathsf{cons}(y, ys)) & \longrightarrow & \mathsf{cons}(y, \mathsf{insert}(x, ys)) \\
\mathsf{if}(\mathsf{false}, x, \mathsf{cons}(y, ys)) & \longrightarrow & \mathsf{cons}(x, \mathsf{cons}(y, ys)) \\
\mathsf{gt}(\mathsf{0}, y) & \xrightarrow{=} & \mathsf{false} \\
\mathsf{gt}(\mathsf{s}(x), \mathsf{0}) & \xrightarrow{=} & \mathsf{true} \\
\mathsf{gt}(\mathsf{s}(x), \mathsf{s}(y)) & \xrightarrow{=} & \mathsf{gt}(x, y)
\end{array}$$

- 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
- the recursive if rule is applied as often as the recursive insert rule

Bird's Eye View of the Transformation

Example

$$
\begin{array}{rcl}
\mathsf{isort}(\mathsf{nil}, ys) & \to & ys \\
\mathsf{isort}(\mathsf{cons}(x, xs), ys) & \to & \mathsf{isort}(xs, \mathsf{insert}(x, ys)) \\
\mathsf{insert}(x, \mathsf{nil}) & \to & \mathsf{cons}(x, \mathsf{nil}) \\
\mathsf{insert}(x, \mathsf{cons}(y, ys)) & \to & \mathsf{if}(\mathsf{gt}(x, y), x, \mathsf{cons}(y, ys)) \\
\mathsf{if}(\mathsf{true}, x, \mathsf{cons}(y, ys)) & \to & \mathsf{cons}(y, \mathsf{insert}(x, ys)) \\
\mathsf{if}(\mathsf{false}, x, \mathsf{cons}(y, ys)) & \to & \mathsf{cons}(x, \mathsf{cons}(y, ys)) \\
\mathsf{gt}(\mathsf{0}, y) & \xrightarrow{=} & \mathsf{false} \\
\mathsf{gt}(\mathsf{s}(x), \mathsf{0}) & \xrightarrow{=} & \mathsf{true} \\
\mathsf{gt}(\mathsf{s}(x), \mathsf{s}(y)) & \xrightarrow{=} & \mathsf{gt}(x, y)
\end{array}
$$

1. abstract terms to integers

Bird's Eye View of the Transformation

Example

$$
\begin{array}{rcll}
\mathsf{isort}(xs', ys) & \xrightarrow{1} & ys & \mid \quad xs' = 1 \\
\mathsf{isort}(\mathsf{cons}(x, xs), ys) & \to & \mathsf{isort}(xs, \mathsf{insert}(x, ys)) & \\
\mathsf{insert}(x, \mathsf{nil}) & \to & \mathsf{cons}(x, \mathsf{nil}) & \\
\mathsf{insert}(x, \mathsf{cons}(y, ys)) & \to & \mathsf{if}(\mathsf{gt}(x, y), x, \mathsf{cons}(y, ys)) & \\
\mathsf{if}(\mathsf{true}, x, \mathsf{cons}(y, ys)) & \to & \mathsf{cons}(y, \mathsf{insert}(x, ys)) & \\
\mathsf{if}(\mathsf{false}, x, \mathsf{cons}(y, ys)) & \to & \mathsf{cons}(x, \mathsf{cons}(y, ys)) & \\
\mathsf{gt}(0, y) & \xrightarrow{=} & \mathsf{false} & \\
\mathsf{gt}(\mathsf{s}(x), 0) & \xrightarrow{=} & \mathsf{true} & \\
\mathsf{gt}(\mathsf{s}(x), \mathsf{s}(y)) & \xrightarrow{=} & \mathsf{gt}(x, y) &
\end{array}
$$

1. abstract terms to integers

Bird's Eye View of the Transformation

Example

$$\begin{array}{rcll}
\mathsf{isort}(xs', ys) & \xrightarrow{1} & ys & \mid \quad xs' = 1 \\
\mathsf{isort}(xs', ys) & \xrightarrow{1} & \mathsf{isort}(xs, \mathsf{insert}(x, ys)) & \mid \quad xs' = 1 + x + xs \\
\mathsf{insert}(x, \mathsf{nil}) & \to & \mathsf{cons}(x, \mathsf{nil}) & \\
\mathsf{insert}(x, \mathsf{cons}(y, ys)) & \to & \mathsf{if}(\mathsf{gt}(x, y), x, \mathsf{cons}(y, ys)) & \\
\mathsf{if}(\mathsf{true}, x, \mathsf{cons}(y, ys)) & \to & \mathsf{cons}(y, \mathsf{insert}(x, ys)) & \\
\mathsf{if}(\mathsf{false}, x, \mathsf{cons}(y, ys)) & \to & \mathsf{cons}(x, \mathsf{cons}(y, ys)) & \\
\mathsf{gt}(0, y) & \xrightarrow{=} & \mathsf{false} & \\
\mathsf{gt}(\mathsf{s}(x), 0) & \xrightarrow{=} & \mathsf{true} & \\
\mathsf{gt}(\mathsf{s}(x), \mathsf{s}(y)) & \xrightarrow{=} & \mathsf{gt}(x, y) &
\end{array}$$

1. abstract terms to integers

Bird's Eye View of the Transformation

Example

$$
\begin{array}{rcll}
\mathsf{isort}(xs', ys) & \xrightarrow{1} & ys & \mid \quad xs' = 1 \\
\mathsf{isort}(xs', ys) & \xrightarrow{1} & \mathsf{isort}(xs, \mathsf{insert}(x, ys)) & \mid \quad xs' = 1 + x + xs \\
\mathsf{insert}(x, ys') & \xrightarrow{1} & 2 + x & \mid \quad ys' = 1 \\
\mathsf{insert}(x, \mathsf{cons}(y, ys)) & \to & \mathsf{if}(\mathsf{gt}(x, y), x, \mathsf{cons}(y, ys)) & \\
\mathsf{if}(\mathsf{true}, x, \mathsf{cons}(y, ys)) & \to & \mathsf{cons}(y, \mathsf{insert}(x, ys)) & \\
\mathsf{if}(\mathsf{false}, x, \mathsf{cons}(y, ys)) & \to & \mathsf{cons}(x, \mathsf{cons}(y, ys)) & \\
\mathsf{gt}(\mathsf{0}, y) & \xrightarrow{=} & \mathsf{false} & \\
\mathsf{gt}(\mathsf{s}(x), \mathsf{0}) & \xrightarrow{=} & \mathsf{true} & \\
\mathsf{gt}(\mathsf{s}(x), \mathsf{s}(y)) & \xrightarrow{=} & \mathsf{gt}(x, y) &
\end{array}
$$

1. abstract terms to integers

Bird's Eye View of the Transformation

Example

$$
\begin{array}{rcl|l}
\mathsf{isort}(xs', ys) & \xrightarrow{1} & ys & xs' = 1 \\
\mathsf{isort}(xs', ys) & \xrightarrow{1} & \mathsf{isort}(xs, \mathsf{insert}(x, ys)) & xs' = 1 + x + xs \\
\mathsf{insert}(x, ys') & \xrightarrow{1} & 2 + x & ys' = 1 \\
\mathsf{insert}(x, ys') & \xrightarrow{1} & \mathsf{if}(\mathsf{gt}(x, y), x, ys') & ys' = 1 + y + ys \\
\mathsf{if}(b, x, ys') & \xrightarrow{1} & 1 + y + \mathsf{insert}(x, ys) & b = 1 \wedge ys' = 1 + y + ys \\
\mathsf{if}(b, x, ys') & \xrightarrow{1} & 1 + ys' & b = 1 \wedge ys' = 1 + y + ys \\
\mathsf{gt}(x', y') & \xrightarrow{0} & 1 & x' = 1 \\
\mathsf{gt}(x', y') & \xrightarrow{0} & 1 & x' = 1 + x \wedge y' = 1 \\
\mathsf{gt}(x', y') & \xrightarrow{0} & \mathsf{gt}(x, y) & x' = 1 + x \wedge y' = 1 + y
\end{array}
$$

1. abstract terms to integers

Bird's Eye View of the Transformation

Example

$$\begin{array}{rcll}
\mathsf{isort}(xs', ys) & \xrightarrow{1} & ys & \mid \quad xs' = 1 \\
\mathsf{isort}(xs', ys) & \xrightarrow{1} & \mathsf{isort}(xs, \mathsf{insert}(x, ys)) & \mid \quad xs' = 1 + x + xs \\
\mathsf{insert}(x, ys') & \xrightarrow{1} & 2 + x & \mid \quad ys' = 1 \\
\mathsf{insert}(x, ys') & \xrightarrow{1} & \mathsf{if}(\mathsf{gt}(x, y), x, ys') & \mid \quad ys' = 1 + y + ys \\
\mathsf{if}(b, x, ys') & \xrightarrow{1} & 1 + y + \mathsf{insert}(x, ys) & \mid \quad b = 1 \wedge ys' = 1 + y + ys \\
\mathsf{if}(b, x, ys') & \xrightarrow{1} & 1 + ys' & \mid \quad b = 1 \wedge ys' = 1 + y + ys \\
\mathsf{gt}(x', y') & \xrightarrow{0} & 1 & \mid \quad x' = 1 \\
\mathsf{gt}(x', y') & \xrightarrow{0} & 1 & \mid \quad x' = 1 + x \wedge y' = 1 \\
\mathsf{gt}(x', y') & \xrightarrow{0} & \mathsf{gt}(x, y) & \mid \quad x' = 1 + x \wedge y' = 1 + y
\end{array}$$

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

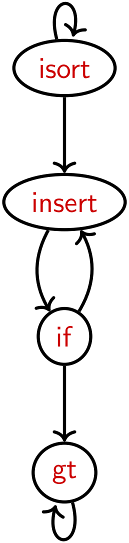
Bird's Eye View of the Transformation

Example

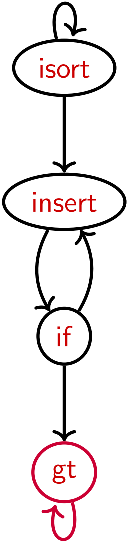
$$\begin{array}{rcll}
\mathsf{isort}(xs', ys) & \xrightarrow{1} & ys & \mid \quad xs' = 1 \\
\mathsf{isort}(xs', ys) & \xrightarrow{1} & \mathsf{isort}(xs, \mathsf{insert}(x, ys)) & \mid \quad xs' = 1 + x + xs \\
\mathsf{insert}(x, ys') & \xrightarrow{1} & 2 + x & \mid \quad ys' = 1 \\
\mathsf{insert}(x, ys') & \xrightarrow{1} & \mathsf{if}(\mathsf{gt}(x, y), x, ys') & \mid \quad ys' = 1 + y + ys \\
\mathsf{if}(b, x, ys') & \xrightarrow{1} & 1 + y + \mathsf{insert}(x, ys) & \mid \quad b = 1 \land ys' = 1 + y + ys \\
\mathsf{if}(b, x, ys') & \xrightarrow{1} & 1 + ys' & \mid \quad b = 1 \land ys' = 1 + y + ys \\
\mathsf{gt}(x', y') & \xrightarrow{0} & 1 & \mid \quad x' = 1 \\
\mathsf{gt}(x', y') & \xrightarrow{0} & 1 & \mid \quad x' = 1 + x \land y' = 1 \\
\mathsf{gt}(x', y') & \xrightarrow{0} & \mathsf{gt}(x, y) & \mid \quad x' = 1 + x \land y' = 1 + y
\end{array}$$

1. abstract terms to integers
    - $[c](x_1, \dots, x_n) = 1 + x_1 + \cdots + x_n$ 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



Call Graph & Bottom SCCs



Bird's Eye View

Example

$$\begin{array}{rcl|l}
\mathsf{isort}(xs', ys) & \xrightarrow{1} & ys & xs' = 1 \\
\mathsf{isort}(xs', ys) & \xrightarrow{1} & \mathsf{isort}(xs, \mathsf{insert}(x, ys)) & xs' = 1 + x + xs \\
\mathsf{insert}(x, ys') & \xrightarrow{1} & 2 + x & ys' = 1 \\
\mathsf{insert}(x, ys') & \xrightarrow{1} & \mathsf{if}(\mathsf{gt}(x, y), x, ys') & ys' = 1 + y + ys \\
\mathsf{if}(b, x, ys') & \xrightarrow{1} & 1 + y + \mathsf{insert}(x, ys) & b = 1 \wedge ys' = 1 + y + ys \\
\mathsf{if}(b, x, ys') & \xrightarrow{1} & 1 + ys' & b = 1 \wedge ys' = 1 + y + ys \\
\mathsf{gt}(x', y') & \xrightarrow{0} & 1 & x' = 1 \\
\mathsf{gt}(x', y') & \xrightarrow{0} & 1 & x' = 1 + x \wedge y' = 1 \\
\mathsf{gt}(x', y') & \xrightarrow{0} & \mathsf{gt}(x, y) & x' = 1 + x \wedge y' = 1 + y
\end{array}$$

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

Bird’s Eye View

Example

$$\begin{array}{rclcl}
\mathsf{isort}(xs', ys) & \xrightarrow{1} & ys & \mid & xs' = 1 \\
\mathsf{isort}(xs', ys) & \xrightarrow{1} & \mathsf{isort}(xs, \mathsf{insert}(x, ys)) & \mid & xs' = 1 + x + xs \\
\mathsf{insert}(x, ys') & \xrightarrow{1} & 2 + x & \mid & ys' = 1 \\
\mathsf{insert}(x, ys') & \xrightarrow{1} & \mathsf{if}(\mathsf{gt}(x, y), x, ys') & \mid & ys' = 1 + y + ys \\
\mathsf{if}(b, x, ys') & \xrightarrow{1} & 1 + y + \mathsf{insert}(x, ys) & \mid & b = 1 \wedge ys' = 1 + y + ys \\
\mathsf{if}(b, x, ys') & \xrightarrow{1} & 1 + ys' & \mid & b = 1 \wedge ys' = 1 + y + ys \\
\mathsf{gt}(x', y') & \xrightarrow{0} & 1 & \mid & x' = 1 \\
\mathsf{gt}(x', y') & \xrightarrow{0} & 1 & \mid & x' = 1 + x \wedge y' = 1 \\
\mathsf{gt}(x', y') & \xrightarrow{0} & \mathsf{gt}(x, y) & \mid & x' = 1 + x \wedge y' = 1 + y
\end{array}$$

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

Bird's Eye View

Example

$$
\begin{array}{rcl|l}
\mathsf{isort}(xs', ys) & \xrightarrow{1} & ys & xs' = 1 \\
\mathsf{isort}(xs', ys) & \xrightarrow{1} & \mathsf{isort}(xs, \mathsf{insert}(x, ys)) & xs' = 1 + x + xs \\
\mathsf{insert}(x, ys') & \xrightarrow{1} & 2 + x & ys' = 1 \\
\mathsf{insert}(x, ys') & \xrightarrow{1} & \mathsf{if}(\mathsf{gt}(x, y), x, ys') & ys' = 1 + y + ys \\
\mathsf{if}(b, x, ys') & \xrightarrow{1} & 1 + y + \mathsf{insert}(x, ys) & b = 1 \wedge ys' = 1 + y + ys \\
\mathsf{if}(b, x, ys') & \xrightarrow{1} & 1 + ys' & b = 1 \wedge ys' = 1 + y + ys \\
\mathsf{gt}(x', y') & \xrightarrow{0} & 1 & x' = 1 \\
\mathsf{gt}(x', y') & \xrightarrow{0} & 1 & x' = 1 + x \wedge y' = 1 \\
\mathsf{gt}(x', y') & \xrightarrow{0} & \mathsf{gt}(x, y) & x' = 1 + x \wedge y' = 1 + y
\end{array}
$$

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
3. analyse runtime of bottom-SCC using standard ITS tools

Bird's Eye View

Example

$$\begin{array}{rcll}
\mathsf{isort}(xs', ys) & \xrightarrow{1} & ys & \mid \quad xs' = 1 \\
\mathsf{isort}(xs', ys) & \xrightarrow{1} & \mathsf{isort}(xs, \mathsf{insert}(x, ys)) & \mid \quad xs' = 1 + x + xs \\
\mathsf{insert}(x, ys') & \xrightarrow{1} & 2 + x & \mid \quad ys' = 1 \\
\mathsf{insert}(x, ys') & \xrightarrow{1} & \mathsf{if}(b, x, ys') & \mid \quad ys' = 1 + y + ys \land b \leq 1 \\
\mathsf{if}(b, x, ys') & \xrightarrow{1} & 1 + y + \mathsf{insert}(x, ys) & \mid \quad b = 1 \land ys' = 1 + y + ys \\
\mathsf{if}(b, x, ys') & \xrightarrow{1} & 1 + ys' & \mid \quad b = 1 \land ys' = 1 + y + ys
\end{array}$$

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
3. analyse runtime of bottom-SCC using standard ITS tools

Abstracting Terms to Integers: Pitfalls

Terminating Variants

Term Rewriting	**Integer Transition Systems**
start terms may have variables	ground start terms only

Example

$\mathsf{h}(x) \to \mathsf{f}(\mathsf{g}(x)) \qquad \mathsf{f}(x) \to \mathsf{f}(x) \qquad \mathsf{g}(\mathsf{a}) \xrightarrow{=} \mathsf{g}(\mathsf{a})$

Terminating Variants

Term Rewriting	**Integer Transition Systems**
start terms may have variables	ground start terms only

Example

$$\mathsf{h}(x) \longrightarrow \mathsf{f}(\mathsf{g}(x)) \qquad \mathsf{f}(x) \longrightarrow \mathsf{f}(x) \qquad \mathsf{g}(\mathsf{a}) \xrightarrow{=} \mathsf{g}(\mathsf{a})$$

innermost rewriting: $\mathsf{h}(x) \longrightarrow \mathsf{f}(\mathsf{g}(x)) \longrightarrow \mathsf{f}(\mathsf{g}(x)) \longrightarrow \ldots$

Terminating Variants

Term Rewriting	**Integer Transition Systems**
start terms may have variables	ground start terms only

Example

$\mathsf{h}(x) \longrightarrow \mathsf{f}(\mathsf{g}(x)) \qquad \mathsf{f}(x) \longrightarrow \mathsf{f}(x) \qquad \mathsf{g}(\mathsf{a}) \overset{=}{\longrightarrow} \mathsf{g}(\mathsf{a})$

innermost rewriting: $\mathsf{h}(x) \longrightarrow \mathsf{f}(\mathsf{g}(x)) \longrightarrow \mathsf{f}(\mathsf{g}(x)) \longrightarrow \ldots$ $\mathcal{O}(\infty)$

Terminating Variants

Term Rewriting	**Integer Transition Systems**
start terms may have variables	ground start terms only

Example

$$\mathsf{h}(x) \longrightarrow \mathsf{f}(\mathsf{g}(x)) \qquad \mathsf{f}(x) \longrightarrow \mathsf{f}(x) \qquad \mathsf{g}(\mathsf{a}) \xrightarrow{=} \mathsf{g}(\mathsf{a})$$

innermost rewriting: $\mathsf{h}(x) \longrightarrow \mathsf{f}(\mathsf{g}(x)) \longrightarrow \mathsf{f}(\mathsf{g}(x)) \longrightarrow \dots$ $\mathcal{O}(\infty)$

- Just ground rewriting?

Terminating Variants

Term Rewriting	**Integer Transition Systems**
start terms may have variables	ground start terms only

Example

$$\mathsf{h}(x) \longrightarrow \mathsf{f}(\mathsf{g}(x)) \qquad \mathsf{f}(x) \longrightarrow \mathsf{f}(x) \qquad \mathsf{g}(\mathsf{a}) \xrightarrow{=} \mathsf{g}(\mathsf{a})$$

innermost rewriting: $\mathsf{h}(x) \longrightarrow \mathsf{f}(\mathsf{g}(x)) \longrightarrow \mathsf{f}(\mathsf{g}(x)) \longrightarrow \ldots$ $\mathcal{O}(\infty)$

ground rewriting: $\mathsf{h}(\mathsf{a}) \longrightarrow \mathsf{f}(\mathsf{g}(\mathsf{a})) \xrightarrow{=} \mathsf{f}(\mathsf{g}(\mathsf{a})) \xrightarrow{=} \ldots$

- Just ground rewriting?

Terminating Variants

Term Rewriting	**Integer Transition Systems**
start terms may have variables	ground start terms only

Example

$$\mathsf{h}(x) \longrightarrow \mathsf{f}(\mathsf{g}(x)) \qquad \mathsf{f}(x) \longrightarrow \mathsf{f}(x) \qquad \mathsf{g}(\mathsf{a}) \xrightarrow{=} \mathsf{g}(\mathsf{a})$$

innermost rewriting: $\mathsf{h}(x) \longrightarrow \mathsf{f}(\mathsf{g}(x)) \longrightarrow \mathsf{f}(\mathsf{g}(x)) \longrightarrow \dots$ $\mathcal{O}(\infty)$

ground rewriting: $\mathsf{h}(\mathsf{a}) \longrightarrow \mathsf{f}(\mathsf{g}(\mathsf{a})) \xrightarrow{=} \mathsf{f}(\mathsf{g}(\mathsf{a})) \xrightarrow{=} \dots$ $\mathcal{O}(1)$

- Just ground rewriting?

Terminating Variants

Term Rewriting	**Integer Transition Systems**
start terms may have variables	ground start terms only

Example

$\mathsf{h}(x) \to \mathsf{f}(\mathsf{g}(x)) \qquad \mathsf{f}(x) \to \mathsf{f}(x) \qquad \mathsf{g}(\mathsf{a}) \xrightarrow{=} \mathsf{g}(\mathsf{a})$

innermost rewriting: $\mathsf{h}(x) \to \mathsf{f}(\mathsf{g}(x)) \to \mathsf{f}(\mathsf{g}(x)) \to \dots$ $\mathcal{O}(\infty)$

ground rewriting: $\mathsf{h}(\mathsf{a}) \to \mathsf{f}(\mathsf{g}(\mathsf{a})) \xrightarrow{=} \mathsf{f}(\mathsf{g}(\mathsf{a})) \xrightarrow{=} \dots$ $\mathcal{O}(1)$

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

Terminating Variants

Term Rewriting	**Integer Transition Systems**
start terms may have variables	ground start terms only

Example

$$\mathsf{h}(x) \to \mathsf{f}(\mathsf{g}(x)) \qquad \mathsf{f}(x) \to \mathsf{f}(x) \qquad \mathsf{g}(\mathsf{a}) \xrightarrow{=} \mathsf{g}(\mathsf{a})$$

innermost rewriting: $\mathsf{h}(x) \to \mathsf{f}(\mathsf{g}(x)) \to \mathsf{f}(\mathsf{g}(x)) \to \dots$ $\mathcal{O}(\infty)$

ground rewriting: $\mathsf{h}(\mathsf{a}) \to \mathsf{f}(\mathsf{g}(\mathsf{a})) \xrightarrow{=} \mathsf{f}(\mathsf{g}(\mathsf{a})) \xrightarrow{=} \dots$ $\mathcal{O}(1)$

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

Definition

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

Terminating Variants

Term Rewriting	**Integer Transition Systems**
start terms may have variables	ground start terms only

Example

$$\mathsf{h}(x) \to \mathsf{f}(\mathsf{g}(x)) \qquad \mathsf{f}(x) \to \mathsf{f}(x) \qquad \mathsf{g}(\mathsf{a}) \xrightarrow{=} \mathsf{g}(\mathsf{a}) \qquad \mathsf{g}(\mathsf{a}) \xrightarrow{=} \mathsf{a}$$

innermost rewriting: $\mathsf{h}(x) \to \mathsf{f}(\mathsf{g}(x)) \to \mathsf{f}(\mathsf{g}(x)) \to \dots$ $\mathcal{O}(\infty)$

ground rewriting: $\mathsf{h}(\mathsf{a}) \to \mathsf{f}(\mathsf{g}(\mathsf{a})) \xrightarrow{=} \mathsf{f}(\mathsf{g}(\mathsf{a})) \xrightarrow{=} \dots$ $\mathcal{O}(1)$

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

Definition

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

Terminating Variants

Term Rewriting	**Integer Transition Systems**
start terms may have variables	ground start terms only

Example

$$\mathsf{h}(x) \to \mathsf{f}(\mathsf{g}(x)) \qquad \mathsf{f}(x) \to \mathsf{f}(x) \qquad \mathsf{g}(\mathsf{a}) \xrightarrow{=} \mathsf{g}(\mathsf{a}) \qquad \mathsf{g}(\mathsf{a}) \xrightarrow{=} \mathsf{a}$$

innermost rewriting: $\mathsf{h}(x) \to \mathsf{f}(\mathsf{g}(x)) \to \mathsf{f}(\mathsf{g}(x)) \to \dots$ $\qquad \mathcal{O}(\infty)$

ground rewriting: $\mathsf{h}(\mathsf{a}) \to \mathsf{f}(\mathsf{g}(\mathsf{a})) \xrightarrow{=} \mathsf{f}(\mathsf{g}(\mathsf{a})) \xrightarrow{=} \dots$ $\qquad \mathcal{O}(1)$

with terminating variant: $\mathsf{h}(\mathsf{a}) \to \mathsf{f}(\mathsf{g}(\mathsf{a})) \xrightarrow{=} \mathsf{f}(\mathsf{a}) \to \mathsf{f}(\mathsf{a}) \to \dots$

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

Definition

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

Terminating Variants

Term Rewriting	**Integer Transition Systems**
start terms may have variables	ground start terms only

Example

$\mathsf{h}(x) \to \mathsf{f}(\mathsf{g}(x)) \qquad \mathsf{f}(x) \to \mathsf{f}(x) \qquad \mathsf{g}(\mathsf{a}) \xrightarrow{=} \mathsf{g}(\mathsf{a}) \qquad \mathsf{g}(\mathsf{a}) \xrightarrow{=} \mathsf{a}$

innermost rewriting: $\mathsf{h}(x) \to \mathsf{f}(\mathsf{g}(x)) \to \mathsf{f}(\mathsf{g}(x)) \to \ldots$ $\mathcal{O}(\infty)$

ground rewriting: $\mathsf{h}(\mathsf{a}) \to \mathsf{f}(\mathsf{g}(\mathsf{a})) \xrightarrow{=} \mathsf{f}(\mathsf{g}(\mathsf{a})) \xrightarrow{=} \ldots$ $\mathcal{O}(1)$

with terminating variant: $\mathsf{h}(\mathsf{a}) \to \mathsf{f}(\mathsf{g}(\mathsf{a})) \xrightarrow{=} \mathsf{f}(\mathsf{a}) \to \mathsf{f}(\mathsf{a}) \to \ldots$ $\mathcal{O}(\infty)$

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

Definition

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

Ensuring Complete Definedness

Term Rewriting	**Integer Transition Systems**
arbitrary matchers	integer substitutions only

Example

$\mathsf{f}(x) \longrightarrow \mathsf{f}(\mathsf{g}(\mathsf{a})) \qquad \mathsf{g}(\mathsf{b}(\mathsf{a})) \longrightarrow \mathsf{a}$

Ensuring Complete Definedness

Term Rewriting	**Integer Transition Systems**
arbitrary matchers	integer substitutions only

Example

$$\mathsf{f}(x) \longrightarrow \mathsf{f}(\mathsf{g}(\mathsf{a})) \qquad \mathsf{g}(\mathsf{b}(\mathsf{a})) \longrightarrow \mathsf{a}$$

original TRS: $\mathsf{f}(\mathsf{a}) \longrightarrow \mathsf{f}(\mathsf{g}(\mathsf{a})) \longrightarrow \mathsf{f}(\mathsf{g}(\mathsf{a})) \longrightarrow \dots$

Ensuring Complete Definedness

Term Rewriting	**Integer Transition Systems**
arbitrary matchers	integer substitutions only

Example

$$\mathsf{f}(x) \longrightarrow \mathsf{f}(\mathsf{g}(\mathsf{a})) \qquad \mathsf{g}(\mathsf{b}(\mathsf{a})) \longrightarrow \mathsf{a}$$

original TRS: $\mathsf{f}(\mathsf{a}) \longrightarrow \mathsf{f}(\mathsf{g}(\mathsf{a})) \longrightarrow \mathsf{f}(\mathsf{g}(\mathsf{a})) \longrightarrow \dots$ $\mathcal{O}(\infty)$

Ensuring Complete Definedness

Term Rewriting	**Integer Transition Systems**
arbitrary matchers	integer substitutions only

Example

$$\mathsf{f}(x) \longrightarrow \mathsf{f}(\mathsf{g}(\mathsf{a})) \qquad \mathsf{g}(\mathsf{b}(\mathsf{a})) \longrightarrow \mathsf{a}$$

original TRS: $\mathsf{f}(\mathsf{a}) \longrightarrow \mathsf{f}(\mathsf{g}(\mathsf{a})) \longrightarrow \mathsf{f}(\mathsf{g}(\mathsf{a})) \longrightarrow \ldots$ $\mathcal{O}(\infty)$

resulting ITS: $\mathsf{f}(1) \xrightarrow{1} \mathsf{f}(\mathsf{g}(1))$

Ensuring Complete Definedness

Term Rewriting	**Integer Transition Systems**
arbitrary matchers	integer substitutions only

Example

$$\mathsf{f}(x) \longrightarrow \mathsf{f}(\mathsf{g}(\mathsf{a})) \qquad \mathsf{g}(\mathsf{b}(\mathsf{a})) \longrightarrow \mathsf{a}$$

original TRS: $\mathsf{f}(\mathsf{a}) \longrightarrow \mathsf{f}(\mathsf{g}(\mathsf{a})) \longrightarrow \mathsf{f}(\mathsf{g}(\mathsf{a})) \longrightarrow \dots$ $\mathcal{O}(\infty)$

resulting ITS: $\mathsf{f}(1) \xrightarrow{1} \mathsf{f}(\mathsf{g}(1))$ $\mathcal{O}(1)$

Ensuring Complete Definedness

Term Rewriting	**Integer Transition Systems**
arbitrary matchers	integer substitutions only

Example

$\mathsf{f}(x) \longrightarrow \mathsf{f}(\mathsf{g}(\mathsf{a}))$ $\qquad$ $\mathsf{g}(\mathsf{b}(\mathsf{a})) \longrightarrow \mathsf{a}$

original TRS: $\mathsf{f}(\mathsf{a}) \longrightarrow \mathsf{f}(\mathsf{g}(\mathsf{a})) \longrightarrow \mathsf{f}(\mathsf{g}(\mathsf{a})) \longrightarrow \dots$ $\qquad$ $\mathcal{O}(\infty)$

resulting ITS: $\mathsf{f}(1) \xrightarrow{1} \mathsf{f}(\mathsf{g}(1))$ $\qquad$ $\mathcal{O}(1)$

Definition

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

Ensuring Complete Definedness

Term Rewriting	**Integer Transition Systems**
arbitrary matchers	integer substitutions only

Example

$$\mathsf{f}(x) \to \mathsf{f}(\mathsf{g}(\mathsf{a})) \qquad \mathsf{g}(\mathsf{b}(\mathsf{a})) \to \mathsf{a} \qquad \mathsf{g}(x) \xrightarrow{=} \mathsf{a}$$

original TRS: $\mathsf{f}(\mathsf{a}) \to \mathsf{f}(\mathsf{g}(\mathsf{a})) \to \mathsf{f}(\mathsf{g}(\mathsf{a})) \to \dots$ $\mathcal{O}(\infty)$

resulting ITS: $\mathsf{f}(1) \xrightarrow{1} \mathsf{f}(\mathsf{g}(1))$ $\mathcal{O}(1)$

Definition

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

TRS not completely defined? $\curvearrowright$ Add suitable terminating variant!

Ensuring Complete Definedness

Term Rewriting	**Integer Transition Systems**
arbitrary matchers	integer substitutions only

Example

$$\mathsf{f}(x) \to \mathsf{f}(\mathsf{g}(\mathsf{a})) \qquad \mathsf{g}(\mathsf{b}(\mathsf{a})) \to \mathsf{a} \qquad \mathsf{g}(x) \xrightarrow{=} \mathsf{a}$$

original TRS: $\mathsf{f}(\mathsf{a}) \to \mathsf{f}(\mathsf{g}(\mathsf{a})) \to \mathsf{f}(\mathsf{g}(\mathsf{a})) \to \dots$ $\mathcal{O}(\infty)$

resulting ITS: $\mathsf{f}(1) \xrightarrow{1} \mathsf{f}(\mathsf{g}(1))$ $\mathcal{O}(1)$

ITS after completion: $\mathsf{f}(1) \xrightarrow{1} \mathsf{f}(\mathsf{g}(1)) \xrightarrow{0} \mathsf{f}(1) \xrightarrow{1} \mathsf{f}(\mathsf{g}(1)) \xrightarrow{0} \dots$

Definition

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

TRS not completely defined? $\curvearrowright$ Add suitable terminating variant!

Ensuring Complete Definedness

Term Rewriting	**Integer Transition Systems**
arbitrary matchers	integer substitutions only

Example

$$\mathsf{f}(x) \longrightarrow \mathsf{f}(\mathsf{g}(\mathsf{a})) \qquad \mathsf{g}(\mathsf{b}(\mathsf{a})) \longrightarrow \mathsf{a} \qquad \mathsf{g}(x) \xrightarrow{=} \mathsf{a}$$

original TRS: $\mathsf{f}(\mathsf{a}) \longrightarrow \mathsf{f}(\mathsf{g}(\mathsf{a})) \longrightarrow \mathsf{f}(\mathsf{g}(\mathsf{a})) \longrightarrow \dots$ $\mathcal{O}(\infty)$

resulting ITS: $\mathsf{f}(1) \xrightarrow{1} \mathsf{f}(\mathsf{g}(1))$ $\mathcal{O}(1)$

ITS after completion: $\mathsf{f}(1) \xrightarrow{1} \mathsf{f}(\mathsf{g}(1)) \xrightarrow{0} \mathsf{f}(1) \xrightarrow{1} \mathsf{f}(\mathsf{g}(1)) \xrightarrow{0} \dots$ $\mathcal{O}(\infty)$

Definition

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

TRS not completely defined? $\curvearrowright$ Add suitable terminating variant!

Ensuring Complete Definedness

Term Rewriting	**Integer Transition Systems**
arbitrary matchers	integer substitutions only

Example

$$\mathsf{f}(x) \longrightarrow \mathsf{f}(\mathsf{g}(\mathsf{a})) \qquad \mathsf{g}(\mathsf{b}(\mathsf{a})) \longrightarrow \mathsf{a} \qquad \mathsf{g}(x) \xrightarrow{=} \mathsf{a}$$

original TRS: $\mathsf{f}(\mathsf{a}) \longrightarrow \mathsf{f}(\mathsf{g}(\mathsf{a})) \longrightarrow \mathsf{f}(\mathsf{g}(\mathsf{a})) \longrightarrow \ldots$ $\mathcal{O}(\infty)$

resulting ITS: $\mathsf{f}(1) \xrightarrow{1} \mathsf{f}(\mathsf{g}(1))$ $\mathcal{O}(1)$

ITS after completion: $\mathsf{f}(1) \xrightarrow{1} \mathsf{f}(\mathsf{g}(1)) \xrightarrow{0} \mathsf{f}(1) \xrightarrow{1} \mathsf{f}(\mathsf{g}(1)) \xrightarrow{0} \ldots$ $\mathcal{O}(\infty)$

Definition

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

TRS not completely defined? $\curvearrowright$ Add suitable terminating variant!

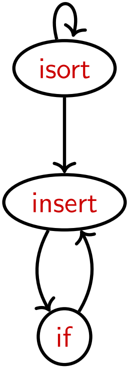
Bird's Eye View

Example

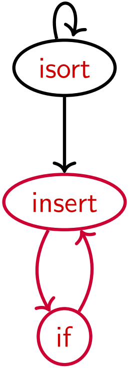
$$\begin{array}{rcl|l}
\mathsf{isort}(xs', ys) & \xrightarrow{1} & ys & xs' = 1 \\
\mathsf{isort}(xs', ys) & \xrightarrow{1} & \mathsf{isort}(xs, \mathsf{insert}(x, ys)) & xs' = 1 + x + xs \\
\mathsf{insert}(x, ys') & \xrightarrow{1} & 2 + x & ys' = 1 \\
\mathsf{insert}(x, ys') & \xrightarrow{1} & \mathsf{if}(b, x, ys') & ys' = 1 + y + ys \wedge b \leq 1 \\
\mathsf{if}(b, x, ys') & \xrightarrow{1} & 1 + y + \mathsf{insert}(x, ys) & b = 1 \wedge ys' = 1 + y + ys \\
\mathsf{if}(b, x, ys') & \xrightarrow{1} & 1 + ys' & b = 1 \wedge ys' = 1 + y + ys
\end{array}$$

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



Call Graph & Bottom SCCs



Bird's Eye View

Example

$$\begin{array}{rcll}
\mathsf{isort}(xs', ys) & \xrightarrow{1} & ys & \mid \quad xs' = 1 \\
\mathsf{isort}(xs', ys) & \xrightarrow{1} & \mathsf{isort}(xs, \mathsf{insert}(x, ys)) & \mid \quad xs' = 1 + x + xs \\
\mathsf{insert}(x, ys') & \xrightarrow{1} & 2 + x & \mid \quad ys' = 1 \\
\mathsf{insert}(x, ys') & \xrightarrow{1} & \mathsf{if}(b, x, ys') & \mid \quad ys' = 1 + y + ys \wedge b \leq 1 \\
\mathsf{if}(b, x, ys') & \xrightarrow{1} & 1 + y + \mathsf{insert}(x, ys) & \mid \quad b = 1 \wedge ys' = 1 + y + ys \\
\mathsf{if}(b, x, ys') & \xrightarrow{1} & 1 + ys' & \mid \quad b = 1 \wedge ys' = 1 + y + ys
\end{array}$$

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

$$
\begin{array}{rcll}
\mathsf{isort}(xs', ys) & \xrightarrow{1} & ys & \mid \quad xs' = 1 \\
\mathsf{isort}(xs', ys) & \xrightarrow{1} & \mathsf{isort}(xs, \mathsf{insert}(x, ys)) & \mid \quad xs' = 1 + x + xs \\
\mathsf{insert}(x, ys') & \xrightarrow{1} & 2 + x & \mid \quad ys' = 1 \\
\mathsf{insert}(x, ys') & \xrightarrow{1} & \mathsf{if}(b, x, ys') & \mid \quad ys' = 1 + y + ys \wedge b \leq 1 \\
\mathsf{if}(b, x, ys') & \xrightarrow{1} & 1 + y + \mathsf{insert}(x, ys) & \mid \quad b = 1 \wedge ys' = 1 + y + ys \\
\mathsf{if}(b, x, ys') & \xrightarrow{1} & 1 + ys' & \mid \quad b = 1 \wedge ys' = 1 + y + ys
\end{array}
$$

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

Using Runtime Analysis to Compute Size Bounds

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

Example

$$\begin{array}{llll}
\mathsf{insert}(x, ys') & \xrightarrow{1} & 2 + x & \mid \quad ys' = 1 \\
\mathsf{insert}(x, ys') & \xrightarrow{1} & \mathsf{if}(b, x, ys') & \mid \quad ys' = 1 + y + ys \land b \leq 1 \\
\mathsf{if}(b, x, ys') & \xrightarrow{1} & 1 + y + \mathsf{insert}(x, ys) & \mid \quad b = 1 \land ys' = 1 + y + ys \\
\mathsf{if}(b, x, ys') & \xrightarrow{1} & 1 + ys' & \mid \quad b = 1 \land ys' = 1 + y + ys
\end{array}$$

Using Runtime Analysis to Compute Size Bounds

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

Example

$$\begin{array}{llll}
\mathsf{insert}(x, ys') & \xrightarrow{1} & 2 + x & \mid\quad ys' = 1 \\
\mathsf{insert}(x, ys') & \xrightarrow{1} & \mathsf{if}(b, x, ys') & \mid\quad ys' = 1 + y + ys \wedge b \leq 1 \\
\mathsf{if}(b, x, ys') & \xrightarrow{1} & 1 + y + \mathsf{insert}(x, ys) & \mid\quad b = 1 \wedge ys' = 1 + y + ys \\
\mathsf{if}(b, x, ys') & \xrightarrow{1} & 1 + ys' & \mid\quad b = 1 \wedge ys' = 1 + y + ys
\end{array}$$

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

$$\begin{array}{rclcl}
\mathsf{insert}(x, ys') & \xrightarrow{2+x} & 2 + x & | & ys' = 1 \\
\mathsf{insert}(x, ys') & \xrightarrow{1} & \mathsf{if}(b, x, ys') & | & ys' = 1 + y + ys \wedge b \leq 1 \\
\mathsf{if}(b, x, ys') & \xrightarrow{1} & 1 + y + \mathsf{insert}(x, ys) & | & b = 1 \wedge ys' = 1 + y + ys \\
\mathsf{if}(b, x, ys') & \xrightarrow{1} & 1 + ys' & | & b = 1 \wedge ys' = 1 + y + ys
\end{array}$$

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

$$\begin{array}{rclcl}
\mathsf{insert}(x, ys') & \xrightarrow{2+x} & 2 + x & \mid & ys' = 1 \\
\mathsf{insert}(x, ys') & \xrightarrow{0} & \mathsf{if}(b, x, ys') & \mid & ys' = 1 + y + ys \land b \leq 1 \\
\mathsf{if}(b, x, ys') & \xrightarrow{1} & 1 + y + \mathsf{insert}(x, ys) & \mid & b = 1 \land ys' = 1 + y + ys \\
\mathsf{if}(b, x, ys') & \xrightarrow{1} & 1 + ys' & \mid & b = 1 \land ys' = 1 + y + ys
\end{array}$$

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

$$\begin{array}{rclcl}
\mathsf{insert}(x, ys') & \xrightarrow{2+x} & 2 + x & \mid & ys' = 1 \\
\mathsf{insert}(x, ys') & \xrightarrow{0} & \mathsf{if}(b, x, ys') & \mid & ys' = 1 + y + ys \wedge b \leq 1 \\
\mathsf{if}(b, x, ys') & \xrightarrow{1+y} & 1 + y + \mathsf{insert}(x, ys) & \mid & b = 1 \wedge ys' = 1 + y + ys \\
\mathsf{if}(b, x, ys') & \xrightarrow{1} & 1 + ys' & \mid & b = 1 \wedge ys' = 1 + y + ys
\end{array}$$

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

$$\begin{array}{rcll}
\mathsf{insert}(x, ys') & \xrightarrow{2+x} & 2 + x & \mid \quad ys' = 1 \\
\mathsf{insert}(x, ys') & \xrightarrow{0} & \mathsf{if}(b, x, ys') & \mid \quad ys' = 1 + y + ys \wedge b \leq 1 \\
\mathsf{if}(b, x, ys') & \xrightarrow{1+y} & 1 + y + \mathsf{insert}(x, ys) & \mid \quad b = 1 \wedge ys' = 1 + y + ys \\
\mathsf{if}(b, x, ys') & \xrightarrow{1+ys'} & 1 + ys' & \mid \quad b = 1 \wedge ys' = 1 + y + ys
\end{array}$$

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

$$\begin{array}{rcll}
\mathsf{insert}(x, ys') & \xrightarrow{2+x} & 2 + x & \mid \quad ys' = 1 \\
\mathsf{insert}(x, ys') & \xrightarrow{0} & \mathsf{if}(b, x, ys') & \mid \quad ys' = 1 + y + ys \wedge b \leq 1 \\
\mathsf{if}(b, x, ys') & \xrightarrow{1+y} & 1 + y + \mathsf{insert}(x, ys) & \mid \quad b = 1 \wedge ys' = 1 + y + ys \\
\mathsf{if}(b, x, ys') & \xrightarrow{1+ys'} & 1 + ys' & \mid \quad b = 1 \wedge ys' = 1 + y + ys
\end{array}$$

Idea: move "integer context" to weights $\curvearrowright$ $\mathsf{sz}(\mathsf{insert}(x, ys')) \leq 1 + x + ys'$

Using Runtime Analysis to Compute Size Bounds

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

Example

$$\begin{array}{rcll} \mathsf{insert}(x, ys') & \xrightarrow{2+x} & 2 + x & \mid \quad ys' = 1 \\ \mathsf{insert}(x, ys') & \xrightarrow{0} & \mathsf{if}(b, x, ys') & \mid \quad ys' = 1 + y + ys \wedge b \leq 1 \\ \mathsf{if}(b, x, ys') & \xrightarrow{1+y} & 1 + y + \mathsf{insert}(x, ys) & \mid \quad b = 1 \wedge ys' = 1 + y + ys \\ \mathsf{if}(b, x, ys') & \xrightarrow{1+ys'} & 1 + ys' & \mid \quad b = 1 \wedge ys' = 1 + y + ys \end{array}$$

Idea: move "integer context" to weights $\curvearrowright$ $\mathsf{sz}(\mathsf{insert}(x, ys')) \leq 1 + x + ys'$

Example

$$\mathsf{f}(x) \xrightarrow{1} 2 + x \cdot \mathsf{f}(x-1) \quad \mid \quad x > 0$$

Using Runtime Analysis to Compute Size Bounds

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

Example

$$\begin{array}{rcll}
\mathsf{insert}(x, ys') & \xrightarrow{2+x} & 2 + x & \mid \quad ys' = 1 \\
\mathsf{insert}(x, ys') & \xrightarrow{0} & \mathsf{if}(b, x, ys') & \mid \quad ys' = 1 + y + ys \wedge b \leq 1 \\
\mathsf{if}(b, x, ys') & \xrightarrow{1+y} & 1 + y + \mathsf{insert}(x, ys) & \mid \quad b = 1 \wedge ys' = 1 + y + ys \\
\mathsf{if}(b, x, ys') & \xrightarrow{1+ys'} & 1 + ys' & \mid \quad b = 1 \wedge ys' = 1 + y + ys
\end{array}$$

Idea: move "integer context" to weights $\curvearrowright$ $\mathsf{sz}(\mathsf{insert}(x, ys')) \leq 1 + x + ys'$

Example

$$\mathsf{f}(x) \xrightarrow{1} 2 + x \cdot \mathsf{f}(x - 1) \quad \mid \quad x > 0$$

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

$$\begin{array}{rcll}
\mathsf{insert}(x, ys') & \xrightarrow{2+x} & 2 + x & \mid\quad ys' = 1 \\
\mathsf{insert}(x, ys') & \xrightarrow{0} & \mathsf{if}(b, x, ys') & \mid\quad ys' = 1 + y + ys \wedge b \leq 1 \\
\mathsf{if}(b, x, ys') & \xrightarrow{1+y} & 1 + y + \mathsf{insert}(x, ys) & \mid\quad b = 1 \wedge ys' = 1 + y + ys \\
\mathsf{if}(b, x, ys') & \xrightarrow{1+ys'} & 1 + ys' & \mid\quad b = 1 \wedge ys' = 1 + y + ys
\end{array}$$

Idea: move "integer context" to weights $\curvearrowright$ $\mathsf{sz}(\mathsf{insert}(x, ys')) \leq 1 + x + ys'$

Example

$$\begin{array}{rcll}
\mathsf{f}(x) & \xrightarrow{1} & 2 + x \cdot \mathsf{f}(x - 1) & \mid\quad x > 0 \\
\mathsf{f}(x, acc) & \xrightarrow{acc \cdot 2} & 2 + x \cdot \mathsf{f}(x - 1, acc \cdot x) & \mid\quad x > 0
\end{array}$$

Idea: use accumulator

Bird's Eye View

Example

$$
\begin{array}{rcl|l}
\mathsf{isort}(xs', ys) & \xrightarrow{1} & ys & xs' = 1 \\
\mathsf{isort}(xs', ys) & \xrightarrow{1} & \mathsf{isort}(xs, \mathsf{insert}(x, ys)) & xs' = 1 + x + xs \\
\mathsf{insert}(x, ys') & \xrightarrow{1} & 2 + x & ys' = 1 \\
\mathsf{insert}(x, ys') & \xrightarrow{1} & \mathsf{if}(b, x, ys') & ys' = 1 + y + ys \land b \leq 1 \\
\mathsf{if}(b, x, ys') & \xrightarrow{1} & 1 + y + \mathsf{insert}(x, ys) & b = 1 \land ys' = 1 + y + ys \\
\mathsf{if}(b, x, ys') & \xrightarrow{1} & 1 + ys' & b = 1 \land ys' = 1 + y + ys
\end{array}
$$

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

$$\begin{array}{lcllcl} \mathsf{isort}(xs', ys) & \xrightarrow{1} & ys & \mid & xs' = 1 \\ \mathsf{isort}(xs', ys) & \xrightarrow{1} & \mathsf{isort}(xs, \mathsf{insert}(x, ys)) & \mid & xs' = 1 + x + xs \end{array}$$

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

$$\begin{array}{lcll} \mathsf{isort}(xs', ys) & \xrightarrow{1} & ys & \mid \quad xs' = 1 \\ \mathsf{isort}(xs', ys) & \xrightarrow{1} & \mathsf{isort}(xs, \mathsf{insert}(x, ys)) & \mid \quad xs' = 1{+}x{+}xs \end{array}$$

- $\mathsf{sz}(\mathsf{insert}(x, ys)) \leq 1 + x + ys$
- $\mathsf{rt}(\mathsf{insert}(x, ys)) \leq 2 \cdot ys$

Removing Nested Function Calls

Example

$$\begin{array}{lcll} \mathsf{isort}(xs', ys) & \xrightarrow{1} & ys & \mid \quad xs' = 1 \\ \mathsf{isort}(xs', ys) & \xrightarrow{1} & \mathsf{isort}(xs, \mathsf{insert}(x, ys)) & \mid \quad xs' = 1{+}x{+}xs \end{array}$$

- $\mathsf{sz}(\mathsf{insert}(x, ys)) \leq 1 + x + ys$
- $\mathsf{rt}(\mathsf{insert}(x, ys)) \leq 2 \cdot ys$
- add costs of nested function call

Removing Nested Function Calls

Example

$$\begin{array}{lcl|l} \mathsf{isort}(xs', ys) & \xrightarrow{1} & ys & xs' = 1 \\ \mathsf{isort}(xs', ys) & \xrightarrow{1+2\cdot ys} & \mathsf{isort}(xs, \mathsf{insert}(x, ys)) & xs' = 1{+}x{+}xs \end{array}$$

- $\mathsf{sz}(\mathsf{insert}(x, ys)) \leq 1 + x + ys$
- $\mathsf{rt}(\mathsf{insert}(x, ys)) \leq 2 \cdot ys$
- add costs of nested function call

Removing Nested Function Calls

Example

$$\begin{array}{lcllc}
\mathsf{isort}(xs', ys) & \xrightarrow{1} & ys & \mid & xs' = 1 \\
\mathsf{isort}(xs', ys) & \xrightarrow{1+2\cdot ys} & \mathsf{isort}(xs, \mathsf{insert}(x, ys)) & \mid & xs' = 1+x+xs
\end{array}$$

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

Removing Nested Function Calls

Example

$$\begin{array}{llll} \mathsf{isort}(xs', ys) & \xrightarrow{1} & ys & \mid \quad xs' = 1 \\ \mathsf{isort}(xs', ys) & \xrightarrow{1+2\cdot ys} & \mathsf{isort}(xs, x_f) & \mid \quad xs' = 1{+}x{+}xs \end{array}$$

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

Removing Nested Function Calls

Example

$$\begin{array}{lcllcl} \mathsf{isort}(xs', ys) & \xrightarrow{1} & ys & & \mid & xs' = 1 \\ \mathsf{isort}(xs', ys) & \xrightarrow{1+2\cdot ys} & \mathsf{isort}(xs, x_f) & & \mid & xs' = 1{+}x{+}xs \end{array}$$

- $\mathsf{sz}(\mathsf{insert}(x, ys)) \leq 1 + x + ys$
- $\mathsf{rt}(\mathsf{insert}(x, ys)) \leq 2 \cdot ys$
- add costs of nested function call
- replace nested function call by fresh variable x_f
- add constraint "$x_f \leq$ size bound"

Removing Nested Function Calls

Example

$$\begin{array}{lcl|l} \mathsf{isort}(xs', ys) & \xrightarrow{1} & ys & xs' = 1 \\ \mathsf{isort}(xs', ys) & \xrightarrow{1+2\cdot ys} & \mathsf{isort}(xs, x_f) & xs' = 1+x+xs \wedge x_f \leq 1+x+ys \end{array}$$

- $\mathsf{sz}(\mathsf{insert}(x, ys)) \leq 1 + x + ys$
- $\mathsf{rt}(\mathsf{insert}(x, ys)) \leq 2 \cdot ys$
- add costs of nested function call
- replace nested function call by fresh variable x_f
- add constraint "$x_f \leq$ size bound"

Removing Nested Function Calls

Example

$$\begin{array}{lcl|l} \mathsf{isort}(xs', ys) & \xrightarrow{1} & ys & xs' = 1 \\ \mathsf{isort}(xs', ys) & \xrightarrow{1+2\cdot ys} & \mathsf{isort}(xs, x_f) & xs' = 1{+}x{+}xs \land x_f \leq 1{+}x{+}ys \end{array}$$

- $\mathsf{sz}(\mathsf{insert}(x, ys)) \leq 1 + x + ys$
- $\mathsf{rt}(\mathsf{insert}(x, ys)) \leq 2 \cdot ys$
- add costs of nested function call
- replace nested function call by fresh variable x_f
- add constraint "$x_f \leq$ size bound"

↷ $\mathsf{rt}(\mathsf{isort}(xs', ys)) \leq \mathcal{O}(xs'^2 + xs' \cdot ys)$

Removing Nested Function Calls

Example

$$\begin{array}{lcll} \mathsf{isort}(xs', ys) & \xrightarrow{1} & ys & \mid \quad xs' = 1 \\ \mathsf{isort}(xs', ys) & \xrightarrow{1+2\cdot ys} & \mathsf{isort}(xs, x_f) & \mid \quad xs' = 1 + x + xs \wedge x_f \leq 1 + x + ys \end{array}$$

- $\mathsf{sz}(\mathsf{insert}(x, ys)) \leq 1 + x + ys$
- $\mathsf{rt}(\mathsf{insert}(x, ys)) \leq 2 \cdot ys$
- add costs of nested function call
- replace nested function call by fresh variable x_f
- add constraint "$x_f \leq$ size bound"

↷ $\mathsf{rt}(\mathsf{isort}(xs', ys)) \leq \mathcal{O}(xs'^2 + xs' \cdot ys)$

- similar techniques to eliminate *outer* function calls

Removing Nested Function Calls

Example

$$\begin{array}{lcllcl} \mathsf{isort}(xs', ys) & \xrightarrow{1} & ys & & \mid & xs' = 1 \\ \mathsf{isort}(xs', ys) & \xrightarrow{1+2\cdot ys} & \mathsf{isort}(xs, x_f) & & \mid & xs' = 1{+}x{+}xs \wedge x_f \leq 1{+}x{+}ys \end{array}$$

- $\mathsf{sz}(\mathsf{insert}(x, ys)) \leq 1 + x + ys$
- $\mathsf{rt}(\mathsf{insert}(x, ys)) \leq 2 \cdot ys$
- add costs of nested function call
- replace nested function call by fresh variable x_f
- add constraint "$x_f \leq$ size bound"

↷ $\mathsf{rt}(\mathsf{isort}(xs', ys)) \leq \mathcal{O}(xs'^2 + xs' \cdot ys)$

- similar techniques to eliminate *outer* function calls

$$\mathsf{times}(\mathsf{s}(x), y) \longrightarrow \mathsf{plus}(\mathsf{times}(x, y), y)$$

Removing Nested Function Calls

Example

$$\begin{array}{llll} \mathsf{isort}(xs', ys) & \xrightarrow{1} & ys & \mid \quad xs' = 1 \\ \mathsf{isort}(xs', ys) & \xrightarrow{1+2\cdot ys} & \mathsf{isort}(xs, x_f) & \mid \quad xs' = 1+x+xs \wedge x_f \leq 1+x+ys \end{array}$$

- $\mathsf{sz}(\mathsf{insert}(x, ys)) \leq 1 + x + ys$
- $\mathsf{rt}(\mathsf{insert}(x, ys)) \leq 2 \cdot ys$
- add costs of nested function call
- replace nested function call by fresh variable x_f
- add constraint "$x_f \leq$ size bound"

↷ $\mathsf{rt}(\mathsf{isort}(xs', ys)) \leq \mathcal{O}(xs'^2 + xs' \cdot ys)$

- similar techniques to eliminate *outer* function calls $\Longrightarrow$ see paper!

$$\mathsf{times}(\mathsf{s}(x), y) \longrightarrow \mathsf{plus}(\mathsf{times}(x, y), y)$$

Experiments

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

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?

Derivational_Complexity_Full_Rewriting/AG01/#3.12, TPDB

$$\begin{array}{rcl|rcl}
\mathsf{app}(\mathsf{nil}, y) & \rightarrow & y & \mathsf{app}(\mathsf{add}(n, x), y) & \rightarrow & \mathsf{add}(n, \mathsf{app}(x, y)) \\
\mathsf{reverse}(\mathsf{nil}) & \rightarrow & \mathsf{nil} & \mathsf{reverse}(\mathsf{add}(n, x)) & \rightarrow & \mathsf{app}(\mathsf{reverse}(x), \mathsf{add}(n, \mathsf{nil})) \\
\mathsf{shuffle}(\mathsf{nil}) & \rightarrow & \mathsf{nil} & \mathsf{shuffle}(\mathsf{add}(n, x)) & \rightarrow & \mathsf{add}(n, \mathsf{shuffle}(\mathsf{reverse}(x)))
\end{array}$$

Derivational_Complexity_Full_Rewriting/AG01/#3.12, TPDB

$$\begin{array}{rcl|rcl}
\mathsf{app}(\mathsf{nil}, y) & \to & y & \mathsf{app}(\mathsf{add}(n, x), y) & \to & \mathsf{add}(n, \mathsf{app}(x, y)) \\
\mathsf{reverse}(\mathsf{nil}) & \to & \mathsf{nil} & \mathsf{reverse}(\mathsf{add}(n, x)) & \to & \mathsf{app}(\mathsf{reverse}(x), \mathsf{add}(n, \mathsf{nil})) \\
\mathsf{shuffle}(\mathsf{nil}) & \to & \mathsf{nil} & \mathsf{shuffle}(\mathsf{add}(n, x)) & \to & \mathsf{add}(n, \mathsf{shuffle}(\mathsf{reverse}(x)))
\end{array}$$

AProVE finds (tight) upper bound $\mathcal{O}(n^4)$ for $\mathrm{dc}_{\mathcal{R}}$:

Derivational_Complexity_Full_Rewriting/AG01/#3.12, TPDB

$$\begin{array}{rcl|rcl}
\mathsf{app}(\mathsf{nil}, y) & \to & y & \mathsf{app}(\mathsf{add}(n, x), y) & \to & \mathsf{add}(n, \mathsf{app}(x, y)) \\
\mathsf{reverse}(\mathsf{nil}) & \to & \mathsf{nil} & \mathsf{reverse}(\mathsf{add}(n, x)) & \to & \mathsf{app}(\mathsf{reverse}(x), \mathsf{add}(n, \mathsf{nil})) \\
\mathsf{shuffle}(\mathsf{nil}) & \to & \mathsf{nil} & \mathsf{shuffle}(\mathsf{add}(n, x)) & \to & \mathsf{add}(n, \mathsf{shuffle}(\mathsf{reverse}(x)))
\end{array}$$

AProVE finds (tight) upper bound $\mathcal{O}(n^4)$ for $\mathrm{dc}_{\mathcal{R}}$:

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

Derivational_Complexity_Full_Rewriting/AG01/#3.12, TPDB

$$\begin{array}{rcl|rcl}
\mathsf{app}(\mathsf{nil}, y) & \to & y & \mathsf{app}(\mathsf{add}(n, x), y) & \to & \mathsf{add}(n, \mathsf{app}(x, y)) \\
\mathsf{reverse}(\mathsf{nil}) & \to & \mathsf{nil} & \mathsf{reverse}(\mathsf{add}(n, x)) & \to & \mathsf{app}(\mathsf{reverse}(x), \mathsf{add}(n, \mathsf{nil})) \\
\mathsf{shuffle}(\mathsf{nil}) & \to & \mathsf{nil} & \mathsf{shuffle}(\mathsf{add}(n, x)) & \to & \mathsf{add}(n, \mathsf{shuffle}(\mathsf{reverse}(x)))
\end{array}$$

AProVE finds (tight) upper bound $\mathcal{O}(n^4)$ for $\mathrm{dc}_{\mathcal{R}}$:

1. Add generator rules $\mathcal{G}$, so analyse $\mathrm{rc}_{\mathcal{R}/\mathcal{G}}$ instead (FroCoS'19)
2. Detect: innermost is worst case here, analyse $\mathrm{irc}_{\mathcal{R}/\mathcal{G}}$ instead (LPAR'17)

Derivational_Complexity_Full_Rewriting/AG01/#3.12, TPDB

$$\begin{array}{rcl|rcl}
\mathsf{app}(\mathsf{nil}, y) & \to & y & \mathsf{app}(\mathsf{add}(n, x), y) & \to & \mathsf{add}(n, \mathsf{app}(x, y)) \\
\mathsf{reverse}(\mathsf{nil}) & \to & \mathsf{nil} & \mathsf{reverse}(\mathsf{add}(n, x)) & \to & \mathsf{app}(\mathsf{reverse}(x), \mathsf{add}(n, \mathsf{nil})) \\
\mathsf{shuffle}(\mathsf{nil}) & \to & \mathsf{nil} & \mathsf{shuffle}(\mathsf{add}(n, x)) & \to & \mathsf{add}(n, \mathsf{shuffle}(\mathsf{reverse}(x)))
\end{array}$$

AProVE finds (tight) upper bound $\mathcal{O}(n^4)$ for $\mathrm{dc}_{\mathcal{R}}$:

1. Add generator rules $\mathcal{G}$, so analyse $\mathrm{rc}_{\mathcal{R}/\mathcal{G}}$ instead (FroCoS'19)
2. Detect: innermost is worst case here, analyse $\mathrm{irc}_{\mathcal{R}/\mathcal{G}}$ instead (LPAR'17)
3. Transform TRS to Recursive Integer Transition System (RITS), analyse complexity of RITS instead (FroCoS'17)

Derivational_Complexity_Full_Rewriting/AG01/#3.12, TPDB

$$\begin{array}{rcl|rcl}
\mathsf{app}(\mathsf{nil}, y) & \to & y & \mathsf{app}(\mathsf{add}(n, x), y) & \to & \mathsf{add}(n, \mathsf{app}(x, y)) \\
\mathsf{reverse}(\mathsf{nil}) & \to & \mathsf{nil} & \mathsf{reverse}(\mathsf{add}(n, x)) & \to & \mathsf{app}(\mathsf{reverse}(x), \mathsf{add}(n, \mathsf{nil})) \\
\mathsf{shuffle}(\mathsf{nil}) & \to & \mathsf{nil} & \mathsf{shuffle}(\mathsf{add}(n, x)) & \to & \mathsf{add}(n, \mathsf{shuffle}(\mathsf{reverse}(x)))
\end{array}$$

AProVE finds (tight) upper bound $\mathcal{O}(n^4)$ for $\mathrm{dc}_{\mathcal{R}}$:

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

Derivational_Complexity_Full_Rewriting/AG01/#3.12, TPDB

$$\begin{array}{rcl|rcl}
\mathsf{app}(\mathsf{nil}, y) & \to & y & \mathsf{app}(\mathsf{add}(n, x), y) & \to & \mathsf{add}(n, \mathsf{app}(x, y)) \\
\mathsf{reverse}(\mathsf{nil}) & \to & \mathsf{nil} & \mathsf{reverse}(\mathsf{add}(n, x)) & \to & \mathsf{app}(\mathsf{reverse}(x), \mathsf{add}(n, \mathsf{nil})) \\
\mathsf{shuffle}(\mathsf{nil}) & \to & \mathsf{nil} & \mathsf{shuffle}(\mathsf{add}(n, x)) & \to & \mathsf{add}(n, \mathsf{shuffle}(\mathsf{reverse}(x)))
\end{array}$$

AProVE finds (tight) upper bound $\mathcal{O}(n^4)$ for $\mathrm{dc}_{\mathcal{R}}$:

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

AProVE finds lower bound $\Omega(n^3)$ for $\mathrm{dc}_{\mathcal{R}}$ using induction technique.

Input for Automated Tools (1/4)

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/`

---

[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.

Input for Automated Tools (1/4)

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/`

---

[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.

Input for Automated Tools (1/4)

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.

Input for Automated Tools (2/4)

Innermost runtime complexity:

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

Input for Automated Tools (3/4)

Derivational complexity:

```
(VAR x y)
(GOAL COMPLEXITY)
(STARTTERM UNRESTRICTED)
(RULES
  plus(0, y) -> y
  plus(s(x), y) -> s(plus(x, y))
)
```

Input for Automated Tools (4/4)

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))
)
```

What if Complexity Analysis Tools have Bugs?

Problem noted in the early Termination Competitions:

- Tools may give contradictory answers on some (few) inputs.

What if Complexity Analysis Tools have Bugs?

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.

What if Complexity Analysis Tools have Bugs?

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

---

What if Complexity Analysis Tools have Bugs?

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:

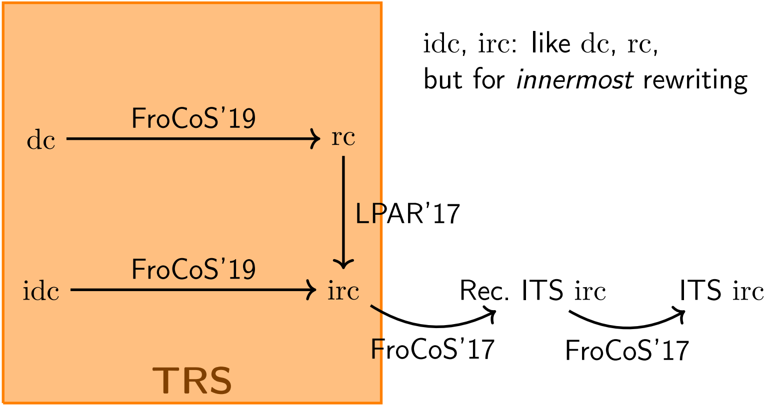
- Proof output by TRS tools in a standard (XML) format
- Proof certifiers based on trusted proof assistants (Isabelle/HOL, Coq, . . .) check proofs independently
- Example for TRS complexity: IsaFoR with certifier CeTA[42]

---

[42] R. Thiemann, C. Sternagel: *Certification of Termination Proofs Using CeTA*, TPHOLs 2009, `http://cl-informatik.uibk.ac.at/software/ceta/`

A Landscape of Complexity Properties and Transformations

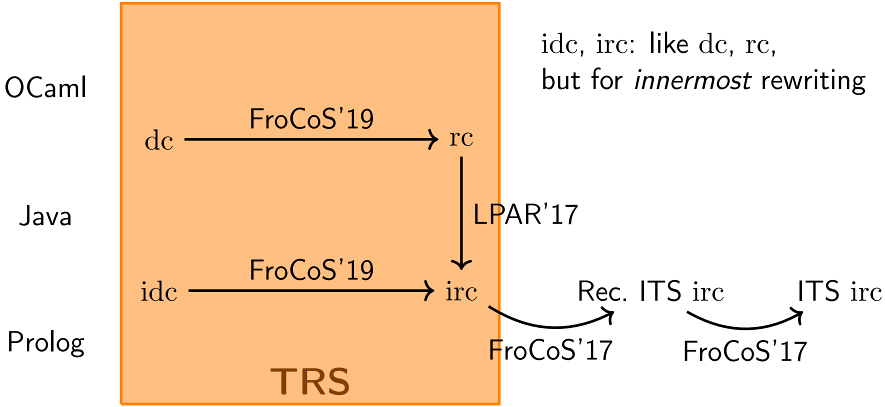
idc, irc: like dc, rc,
but for *innermost* rewriting



$$\begin{tikzcd}
\mathrm{dc} \arrow[r, "\text{FroCoS'19}"] & \mathrm{rc} \arrow[d, "\text{LPAR'17}"] & & \\
\mathrm{idc} \arrow[r, "\text{FroCoS'19}"] & \mathrm{irc} \arrow[r, bend right, "\text{FroCoS'17}"'] & \text{Rec. ITS } \mathrm{irc} \arrow[r, bend right, "\text{FroCoS'17}"'] & \text{ITS } \mathrm{irc}
\end{tikzcd}$$

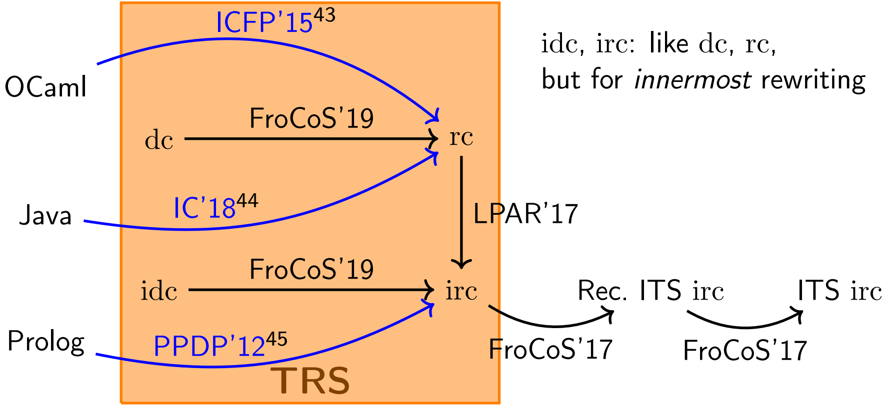
A Landscape of Complexity Properties and Transformations

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



$$\begin{tikzcd}
\mathrm{dc} \arrow[r, "\text{FroCoS'19}"] & \mathrm{rc} \arrow[d, "\text{LPAR'17}"] & & \\
\mathrm{idc} \arrow[r, "\text{FroCoS'19}"] & \mathrm{irc} \arrow[r, bend right, "\text{FroCoS'17}"'] & \text{Rec. ITS } \mathrm{irc} \arrow[r, bend right, "\text{FroCoS'17}"'] & \text{ITS } \mathrm{irc}
\end{tikzcd}$$

A Landscape of Complexity Properties and Transformations



OCaml $\xrightarrow{\text{ICFP'15[43]}}$ rc

dc $\xrightarrow{\text{FroCoS'19}}$ rc

Java $\xrightarrow{\text{IC'18[44]}}$ rc

rc $\xrightarrow{\text{LPAR'17}}$ irc

idc $\xrightarrow{\text{FroCoS'19}}$ irc

Prolog $\xrightarrow{\text{PPDP'12[45]}}$ irc

irc $\xrightarrow{\text{FroCoS'17}}$ Rec. ITS irc $\xrightarrow{\text{FroCoS'17}}$ ITS irc

TRS

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

---

[43] M. Avanzini, U. Dal Lago, G. Moser: *Analysing the Complexity of Functional Programs: Higher-Order Meets First-Order*, ICFP '15

[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

Program Complexity Analysis via Term Rewriting: OCaml

Complexity analysis for functional programs (OCaml) by translation to term rewriting

Program Complexity Analysis via Term Rewriting: OCaml

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: $\mathsf{map}(F, xs)$

Program Complexity Analysis via Term Rewriting: OCaml

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: $\mathsf{map}(F, xs)$

Solution:

- Defunctionalisation to: $\mathsf{a}(\mathsf{a}(\mathsf{map}, F), xs)$
- 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 $\mathrm{rc}_{\mathcal{R}}(n)$ an upper bound for the complexity of the OCaml program

Program Complexity Analysis via Term Rewriting: Prolog and Java

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 $\mathrm{rc}_{\mathcal{R}}(n)$ 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

Current Developments

- **amortised** complexity analysis for term rewriting[47]

---

[47] G. Moser, M. Schneckenreither: *Automated amortised resource analysis for term rewrite systems*, SCP '20

Current Developments

- **amortised** complexity analysis for term rewriting[47]
- **probabilistic** term rewriting $\longrightarrow$ 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 $\longrightarrow$ 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 $\longrightarrow$ 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

[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

Current Developments

- **amortised** complexity analysis for term rewriting[47]
- **probabilistic** term rewriting ⟶ 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, . . .)

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!

References I

- T. Arts and J. Giesl. Termination of term rewriting using dependency pairs. *Theoretical Computer Science*, 236(1-2):133–178, 2000.
- M. Avanzini and G. Moser. Dependency pairs and polynomial path orders. In *RTA '09*, pages 48–62, 2009.
- M. Avanzini and G. Moser. A combination framework for complexity. *Information and Computation*, 248:22–55, 2016.
- M. Avanzini, G. Moser, and M. Schaper. TcT: Tyrolean Complexity Tool. In *TACAS '16*, pages 407–423, 2016.
- M. Avanzini, U. Dal Lago, and A. Yamada. On probabilistic term rewriting. *Science of Computer Programming*, 185, 2020.
- T. Baudon, C. Fuhs, and L. Gonnord. Parallel complexity of term rewriting systems. In *WST '21*, pages 45–50, 2021.

References II

G. Bonfante, A. Cichon, J. Marion, and H. Touzet. Algorithms with polynomial interpretation termination proof. *Journal of Functional Programming*, 11(1):33–53, 2001.

C. Borralleras, S. Lucas, A. Oliveras, E. Rodríguez-Carbonell, and A. Rubio. SAT modulo linear arithmetic for solving polynomial constraints. *Journal of Automated Reasoning*, 48(1):107–131, 2012.

P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *POPL '77*, pages 238–252, 1977.

F. Emmes, T. Enger, and J. Giesl. Proving non-looping non-termination automatically. In *IJCAR '12*, pages 225–240.

J. Endrullis, J. Waldmann, and H. Zantema. Matrix interpretations for proving termination of term rewriting. *Journal of Automated Reasoning*, 40(2–3):195–220, 2008.

References III

- F. Frohn and J. Giesl. Analyzing runtime complexity via innermost runtime complexity. In *Proc. LPAR '17*, pages 249–268, 2017.
- F. Frohn, J. Giesl, J. Hensel, C. Aschermann, and T. Ströder. Lower bounds for runtime complexity of term rewriting. *Journal of Automated Reasoning*, 59(1):121–163, 2017.
- C. Fuhs. Transforming derivational complexity of term rewriting to runtime complexity. In *FroCoS '19*, pages 348–364, 2019.
- C. Fuhs, J. Giesl, A. Middeldorp, P. Schneider-Kamp, R. Thiemann, and H. Zankl. SAT solving for termination analysis with polynomial interpretations. In *SAT '07*, pages 340–354, 2007.
- A. Geser, D. Hofbauer, and J. Waldmann. Match-bounded string rewriting systems. *Applicable Algebra in Engineering, Communication and Computing*, 15(3–4):149–171, 2004.

References IV

- J. Giesl, R. Thiemann, P. Schneider-Kamp, and S. Falke. Mechanizing and improving dependency pairs. *Journal of Automated Reasoning*, 37 (3):155–203, 2006.
- J. Giesl, T. Ströder, P. Schneider-Kamp, F. Emmes, and C. Fuhs. Symbolic evaluation graphs and term rewriting: A general methodology for analyzing logic programs. In *PPDP '12*, pages 1–12, 2012.
- N. Hirokawa and A. Middeldorp. Tyrolean Termination Tool: Techniques and features. *Information and Computation*, 205(4): 474–511, 2007.
- N. Hirokawa and G. Moser. Automated complexity analysis based on the dependency pair method. In *IJCAR '08*, pages 364–379, 2008.
- N. Hirokawa and G. Moser. Automated complexity analysis based on context-sensitive rewriting. In *RTA-TLCA '14*, pages 257–271, 2014.

References V

- D. Hofbauer and C. Lautemann. Termination proofs and the length of derivations. In *RTA '89*, pages 167–177, 1989.
- S. Kamin and J.-J. Lévy. Two generalizations of the recursive path ordering. Unpublished Manuscript, University of Illinois, Urbana, IL, USA, 1980.
- C. Kop and D. Vale. Tuple interpretations for higher-order rewriting. In *FSCD '21*, 2021. To appear.
- A. Koprowski and J. Waldmann. Max/plus tree automata for termination of term rewriting. *Acta Cybernetica*, 19(2):357–392, 2009.
- M. Korp, C. Sternagel, H. Zankl, and A. Middeldorp. Tyrolean Termination Tool 2. In *RTA '09*, pages 295–304, 2009.
- D. S. Lankford. Canonical algebraic simplification in computational logic. Technical Report ATP-25, University of Texas, 1975.

References VI

- G. Moser and M. Schaper. From Jinja bytecode to term rewriting: A complexity reflecting transformation. *Information and Computation*, 261:116–143, 2018.
- G. Moser and A. Schnabl. The derivational complexity induced by the dependency pair method. *Logical Methods in Computer Science*, 7(3), 2011a.
- G. Moser and A. Schnabl. Termination proofs in the dependency pair framework may induce multiple recursive derivational complexity. In *RTA '11*, pages 235–250, 2011b.
- G. Moser and M. Schneckenreither. Automated amortised resource analysis for term rewrite systems. *Science of Computer Programming*, 185, 2020.

References VII

- G. Moser, A. Schnabl, and J. Waldmann. Complexity analysis of term rewriting based on matrix and context dependent interpretations. In *FSTTCS '08*, pages 304–315, 2008.
- M. Naaf, F. Frohn, M. Brockschmidt, C. Fuhs, and J. Giesl. Complexity analysis for term rewriting by integer transition systems. In *FroCoS '17*, pages 132–150, 2017.
- F. Neurauter, H. Zankl, and A. Middeldorp. Revisiting matrix interpretations for polynomial derivational complexity of term rewriting. In *LPAR (Yogyakarta) '10*, pages 550–564, 2010.
- L. Noschinski, F. Emmes, and J. Giesl. Analyzing innermost runtime complexity of term rewriting by dependency pairs. *Journal of Automated Reasoning*, 51(1):27–56, 2013.
- A. Schnabl and J. G. Simonsen. The exact hardness of deciding derivational and runtime complexity. In *CSL '11*, pages 481–495, 2011.

References VIII

- R. Thiemann and C. Sternagel. Certification of termination proofs using CeTA. In *TPHOLs '09*, pages 452–468, 2009.
- A. Weiermann. Termination proofs for term rewriting systems by lexicographic path orderings imply multiply recursive derivation lengths. *Theoretical Computer Science*, 139(1&2):355–362, 1995.
- S. Winkler and G. Moser. Runtime complexity analysis of logically constrained rewriting. In *LOPSTR '20*, pages 37–55, 2020.