Automated Complexity Analysis of Rewriting

Georg Moser

Institute of Computer Science
University of Innsbruck

CONCERTO, Final meeting, June 10, 2010

Overview

- Crash Course in Rewriting
- Termination
- Complexity of Rewriting
    - The Past
    - The Present
- The Future

The Foundation

Signature

$0, \mathsf{fib}$ constants s unary $\mathsf{f}, +, ::$ binary

Signature

$0, \mathsf{fib}$ constants s unary $\mathsf{f}, +, ::$ binary

Rewrite Rules

$$\begin{array}{rcl@{\quad}rcl} 0+y & \to & y & \mathsf{fib} & \to & \mathsf{f}(\mathsf{s}(0), \mathsf{s}(0)) \\ \mathsf{s}(x)+y & \to & \mathsf{s}(x+y) & \mathsf{f}(x, y) & \to & x :: \mathsf{f}(y, x+y) \end{array}$$

Signature

$0, \mathsf{fib}$ constants s unary $\mathsf{f}, +, ::$ binary

Rewrite Rules

$$\begin{array}{rclrcl} 0+y & \to & y & \mathsf{fib} & \to & \mathsf{f}(\mathsf{s}(0), \mathsf{s}(0)) \\ \mathsf{s}(x)+y & \to & \mathsf{s}(x+y) & \mathsf{f}(x, y) & \to & x :: \mathsf{f}(y, x+y) \end{array}$$

Rewriting

fib

Signature

$0, \mathsf{fib}$ constants s unary $\mathsf{f}, +, ::$ binary

Rewrite Rules

$$\begin{aligned} 0+y &\to y & \mathsf{fib} &\to \mathsf{f}(\mathsf{s}(0), \mathsf{s}(0)) \\ \mathsf{s}(x)+y &\to \mathsf{s}(x+y) & \mathsf{f}(x, y) &\to x :: \mathsf{f}(y, x+y) \end{aligned}$$

Rewriting

$$\mathsf{fib} \to \mathsf{f}(\mathsf{s}(0), \mathsf{s}(0))$$

Signature

$0, \mathsf{fib}$ constants s unary $\mathsf{f}, +, ::$ binary

Rewrite Rules

$$\begin{array}{rclrcl} 0+y & \to & y & \mathsf{fib} & \to & \mathsf{f}(\mathsf{s}(0), \mathsf{s}(0)) \\ \mathsf{s}(x)+y & \to & \mathsf{s}(x+y) & \mathsf{f}(x, y) & \to & x :: \mathsf{f}(y, x+y) \end{array}$$

Rewriting

$$\begin{array}{rcl} \mathsf{fib} & \to & \mathsf{f}(\mathsf{s}(0), \mathsf{s}(0)) \\ & \to & \mathsf{s}(0) :: \mathsf{f}(\mathsf{s}(0), \mathsf{s}(0)+\mathsf{s}(0)) \end{array}$$

Signature

$0, \mathsf{fib}$ constants s unary $\mathsf{f}, +, ::$ binary

Rewrite Rules

$$\begin{aligned} 0+y &\to y & \mathsf{fib} &\to \mathsf{f}(\mathsf{s}(0), \mathsf{s}(0)) \\ \mathsf{s}(x)+y &\to \mathsf{s}(x+y) & \mathsf{f}(x, y) &\to x :: \mathsf{f}(y, x+y) \end{aligned}$$

Rewriting

$$\begin{aligned} \mathsf{fib} &\to \mathsf{f}(\mathsf{s}(0), \mathsf{s}(0)) \\ &\to \mathsf{s}(0) :: \mathsf{f}(\mathsf{s}(0), \mathsf{s}(0)+\mathsf{s}(0)) \\ &\to \mathsf{s}(0) :: \mathsf{f}(\mathsf{s}(0), \mathsf{s}(0+\mathsf{s}(0))) \end{aligned}$$

Signature

$0, \mathsf{fib}$ constants s unary $\mathsf{f}, +, ::$ binary

Rewrite Rules

$$\begin{aligned} 0+y &\to y & \mathsf{fib} &\to \mathsf{f}(\mathsf{s}(0),\mathsf{s}(0)) \\ \mathsf{s}(x)+y &\to \mathsf{s}(x+y) & \mathsf{f}(x,y) &\to x :: \mathsf{f}(y, x+y) \end{aligned}$$

Rewriting

$$\begin{aligned} \mathsf{fib} &\to \mathsf{f}(\mathsf{s}(0),\mathsf{s}(0)) \\ &\to \mathsf{s}(0) :: \mathsf{f}(\mathsf{s}(0),\mathsf{s}(0)+\mathsf{s}(0)) \\ &\to \mathsf{s}(0) :: \mathsf{f}(\mathsf{s}(0),\mathsf{s}(0+\mathsf{s}(0))) \\ &\to \mathsf{s}(0) :: \mathsf{f}(\mathsf{s}(0),\mathsf{s}(\mathsf{s}(0))) \end{aligned}$$

Signature

$0, \mathsf{fib}$ constants s unary $\mathsf{f}, +, ::$ binary

Rewrite Rules

$$\begin{aligned} 0+y &\to y & \mathsf{fib} &\to \mathsf{f}(\mathsf{s}(0),\mathsf{s}(0)) \\ \mathsf{s}(x)+y &\to \mathsf{s}(x+y) & \mathsf{f}(x,y) &\to x :: \mathsf{f}(y, x+y) \end{aligned}$$

Rewriting

$$\begin{aligned} \mathsf{fib} &\to \mathsf{f}(\mathsf{s}(0),\mathsf{s}(0)) \\ &\to \mathsf{s}(0) :: \mathsf{f}(\mathsf{s}(0),\mathsf{s}(0)+\mathsf{s}(0)) \\ &\to \mathsf{s}(0) :: \mathsf{f}(\mathsf{s}(0),\mathsf{s}(0+\mathsf{s}(0))) \\ &\to \mathsf{s}(0) :: \mathsf{f}(\mathsf{s}(0),\mathsf{s}(\mathsf{s}(0))) \\ &\to \mathsf{s}(0) :: \mathsf{s}(0) :: \mathsf{f}(\mathsf{s}(\mathsf{s}(0)),\mathsf{s}(0)+\mathsf{s}(\mathsf{s}(0))) \end{aligned}$$

Signature

$0, \mathsf{fib}$ constants s unary $\mathsf{f}, +, ::$ binary

Rewrite Rules

$$\begin{array}{rcl@{\qquad}rcl} 0+y & \to & y & \mathsf{fib} & \to & \mathsf{f}(\mathsf{s}(0), \mathsf{s}(0)) \\ \mathsf{s}(x)+y & \to & \mathsf{s}(x+y) & \mathsf{f}(x, y) & \to & x :: \mathsf{f}(y, x+y) \end{array}$$

Rewriting

$$\begin{array}{rcl} \mathsf{fib} & \to & \mathsf{f}(\mathsf{s}(0), \mathsf{s}(0)) \\ & \to & \mathsf{s}(0) :: \mathsf{f}(\mathsf{s}(0), \mathsf{s}(0)+\mathsf{s}(0)) \\ & \to & \mathsf{s}(0) :: \mathsf{f}(\mathsf{s}(0), \mathsf{s}(0+\mathsf{s}(0))) \\ & \to & \mathsf{s}(0) :: \mathsf{f}(\mathsf{s}(0), \mathsf{s}(\mathsf{s}(0))) \\ & \to & \mathsf{s}(0) :: \mathsf{s}(0) :: \mathsf{f}(\mathsf{s}(\mathsf{s}(0)), \mathsf{s}(0)+\mathsf{s}(\mathsf{s}(0))) \\ & \to^{+} & \mathsf{s}(0) :: \mathsf{s}(0) :: \mathsf{f}(\mathsf{s}(\mathsf{s}(0)), \mathsf{s}(\mathsf{s}(\mathsf{s}(0)))) \end{array}$$

Signature

$0, \mathsf{fib}$ constants s unary $\mathsf{f}, +, ::$ binary

Rewrite Rules

$$\begin{aligned} 0+y &\to y & \mathsf{fib} &\to \mathsf{f}(\mathsf{s}(0), \mathsf{s}(0)) \\ \mathsf{s}(x)+y &\to \mathsf{s}(x+y) & \mathsf{f}(x, y) &\to x :: \mathsf{f}(y, x+y) \end{aligned}$$

Rewriting

$$\begin{aligned} \mathsf{fib} &\to \mathsf{f}(\mathsf{s}(0), \mathsf{s}(0)) \\ &\to \mathsf{s}(0) :: \mathsf{f}(\mathsf{s}(0), \mathsf{s}(0)+\mathsf{s}(0)) \\ &\to \mathsf{s}(0) :: \mathsf{f}(\mathsf{s}(0), \mathsf{s}(0+\mathsf{s}(0))) \\ &\to \mathsf{s}(0) :: \mathsf{f}(\mathsf{s}(0), \mathsf{s}(\mathsf{s}(0))) \\ &\to \mathsf{s}(0) :: \mathsf{s}(0) :: \mathsf{f}(\mathsf{s}(\mathsf{s}(0)), \mathsf{s}(0)+\mathsf{s}(\mathsf{s}(0))) \\ &\to^+ \mathsf{s}(0) :: \mathsf{s}(0) :: \mathsf{f}(\mathsf{s}(\mathsf{s}(0)), \mathsf{s}(\mathsf{s}(\mathsf{s}(0)))) \\ &\to^+ \mathsf{s}(0) :: \mathsf{s}(0) :: \mathsf{s}^2(0) :: \mathsf{f}(\mathsf{s}^3(0), \mathsf{s}^5(0)) \\ &\to^+ \mathsf{s}(0) :: \mathsf{s}(0) :: \mathsf{s}^2(0) :: \mathsf{s}^3(0) :: \mathsf{f}(\mathsf{s}^5(0), \mathsf{s}^8(0)) \end{aligned}$$

Signature

$0, \mathsf{fib}$ constants s unary $\mathsf{f}, +, ::$ binary

Rewrite Rules

$$
\begin{array}{rcl@{\qquad}rcl}
0+y & \to & y & \mathsf{fib} & \to & \mathsf{f}(\mathsf{s}(0),\mathsf{s}(0)) \\
\mathsf{s}(x)+y & \to & \mathsf{s}(x+y) & \mathsf{f}(x,y) & \to & x :: \mathsf{f}(y,x+y)
\end{array}
$$

Rewriting

$$
\begin{array}{rll}
\mathsf{fib} & \to & \mathsf{f}(\mathsf{s}(0),\mathsf{s}(0)) \\
& \to & \mathsf{s}(0) :: \mathsf{f}(\mathsf{s}(0),\mathsf{s}(0)+\mathsf{s}(0)) \\
& \to & \mathsf{s}(0) :: \mathsf{f}(\mathsf{s}(0),\mathsf{s}(0+\mathsf{s}(0))) \\
& \to & \mathsf{s}(0) :: \mathsf{f}(\mathsf{s}(0),\mathsf{s}(\mathsf{s}(0))) \\
& \to & \mathsf{s}(0) :: \mathsf{s}(0) :: \mathsf{f}(\mathsf{s}(\mathsf{s}(0)),\mathsf{s}(0)+\mathsf{s}(\mathsf{s}(0))) \\
& \to^{+} & \mathsf{s}(0) :: \mathsf{s}(0) :: \mathsf{f}(\mathsf{s}(\mathsf{s}(0)),\mathsf{s}(\mathsf{s}(\mathsf{s}(0)))) \\
& \to^{+} & \mathsf{s}(0) :: \mathsf{s}(0) :: \mathsf{s}^{2}(0) :: \mathsf{f}(\mathsf{s}^{3}(0),\mathsf{s}^{5}(0)) \\
& \to^{+} & \mathsf{s}(0) :: \mathsf{s}(0) :: \mathsf{s}^{2}(0) :: \mathsf{s}^{3}(0) :: \mathsf{f}(\mathsf{s}^{5}(0),\mathsf{s}^{8}(0))
\end{array}
$$

infinite computations are possible

Question

how to prevent infinite computations?

Question

how to prevent infinite computations?

consider a TRS $\mathcal{R}$ and a term t such that

$$t = t_1 \to_{\mathcal{R}} t_2 \to_{\mathcal{R}} t_3 \to_{\mathcal{R}} \cdots$$

Question

how to prevent infinite computations?

consider a TRS $\mathcal{R}$ and a term t such that

$$t = t_1 \to_{\mathcal{R}} t_2 \to_{\mathcal{R}} t_3 \to_{\mathcal{R}} \cdots \to_{\mathcal{R}} t_n$$

Definition

a TRS is terminating if $\to_{\mathcal{R}}$ is well-founded

Question

how to prevent infinite computations?

consider a TRS $\mathcal{R}$ and a term t such that

$$t = t_1 \to_{\mathcal{R}} t_2 \to_{\mathcal{R}} t_3 \to_{\mathcal{R}} \cdots \to_{\mathcal{R}} t_n$$

Definition

a TRS is terminating if $\to_{\mathcal{R}}$ is well-founded

Definition

a reduction order $\succ$ is a well-founded order, such that if

$$\forall\ l \to r \in \mathcal{R} \Rightarrow l \succ r$$

compatibility

then $\mathcal{R}$ is terminating

Question

how to prevent infinite computations?

consider a TRS $\mathcal{R}$ and a term t such that

$$t = t_1 \succ t_2 \succ t_3 \succ \cdots \succ t_n$$

Definition

a TRS is terminating if $\to_{\mathcal{R}}$ is well-founded

Definition

a reduction order $\succ$ is a well-founded order, such that if

$$\forall\ l \to r \in \mathcal{R} \Rightarrow l \succ r \qquad \text{compatibility}$$

then $\mathcal{R}$ is terminating

Answer

find a compatible reduction order

The Multiset Path Order

let $>$ denote a precedence; $>$ induces order $>_{\mathsf{mpo}}$:

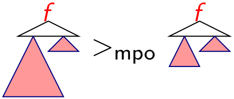
$s = f(s_1, \ldots, s_n) >_{\mathsf{mpo}} t$ if either

The Multiset Path Order

let $>$ denote a precedence; $>$ induces order $>_{\mathsf{mpo}}$:

$s = f(s_1, \ldots, s_n) >_{\mathsf{mpo}} t$ if either

2. $t = f(t_1, \ldots, t_n)$ and
   $\{\!\{ s_1, \ldots, s_n \}\!\} >^{\mathsf{mul}}_{\mathsf{mpo}} \{\!\{ t_1, \ldots, t_n \}\!\}$

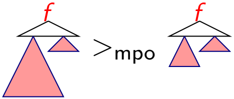


The Multiset Path Order

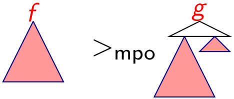
let $>$ denote a precedence; $>$ induces order $>_{\mathsf{mpo}}$:

$s = f(s_1, \ldots, s_n) >_{\mathsf{mpo}} t$ if either

2. $t = f(t_1, \ldots, t_n)$ and
   $\{\!\{s_1, \ldots, s_n\}\!\} >^{\mathsf{mul}}_{\mathsf{mpo}} \{\!\{t_1, \ldots, t_n\}\!\}$



3. $t = g(t_1, \ldots, t_m)$ with $f > g$ and
   $\forall i \ s >_{\mathsf{mpo}} t_i$

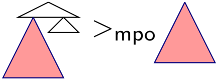


The Multiset Path Order

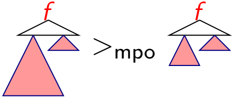
let $>$ denote a precedence; $>$ induces order $>_{\text{mpo}}$:

$s = f(s_1, \ldots, s_n) >_{\text{mpo}} t$ if either

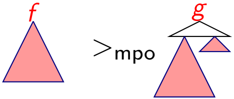
1. $\exists i \; s_i \geqslant_{\text{mpo}} t$



2. $t = f(t_1, \ldots, t_n)$ and
   $\{\!\{ s_1, \ldots, s_n \}\!\} >^{\text{mul}}_{\text{mpo}} \{\!\{ t_1, \ldots, t_n \}\!\}$



3. $t = g(t_1, \ldots, t_m)$ with $f > g$ and
   $\forall i \; s >_{\text{mpo}} t_i$



Example

consider a precedence $>$ such that

$$\mathsf{ack} \quad > \quad \mathsf{s} \quad > \quad \mathsf{0}$$

Example

consider a precedence $>$ such that

$$\mathsf{ack} \quad > \quad \mathsf{s} \quad > \quad 0$$

consider the TRS $\mathcal{R}_{\mathsf{ack}}$

$$\mathsf{ack}(0, y) \to \mathsf{s}(y) \qquad \mathsf{ack}(\mathsf{s}(x), \mathsf{s}(y)) \to \mathsf{ack}(x, \mathsf{ack}(\mathsf{s}(x), y))$$

$$\mathsf{ack}(\mathsf{s}(x), 0) \to \mathsf{ack}(x, \mathsf{s}(0))$$

Example

consider a precedence $>$ such that

$$\mathsf{ack} \quad > \quad \mathsf{s} \quad > \quad \mathsf{0}$$

consider the TRS $\mathcal{R}_{\mathsf{ack}}$

$$\mathsf{ack}(\mathsf{0}, y) \to \mathsf{s}(y) \qquad \mathsf{ack}(\mathsf{s}(x), \mathsf{s}(y)) \to \mathsf{ack}(x, \mathsf{ack}(\mathsf{s}(x), y))$$

$$\mathsf{ack}(\mathsf{s}(x), \mathsf{0}) \to \mathsf{ack}(x, \mathsf{s}(\mathsf{0}))$$

application of $>_{\mathsf{mpo}}$:

$$\mathsf{ack}(\mathsf{0}, y) \quad >_{\mathsf{mpo}} \quad \mathsf{s}(y) \qquad \checkmark$$

Example

consider a precedence $>$ such that

$$\mathsf{ack} > \mathsf{s} > \mathsf{0}$$

consider the TRS $\mathcal{R}_{\mathsf{ack}}$

$$\mathsf{ack}(\mathsf{0}, y) \rightarrow \mathsf{s}(y) \qquad \mathsf{ack}(\mathsf{s}(x), \mathsf{s}(y)) \rightarrow \mathsf{ack}(x, \mathsf{ack}(\mathsf{s}(x), y))$$

$$\mathsf{ack}(\mathsf{s}(x), \mathsf{0}) \rightarrow \mathsf{ack}(x, \mathsf{s}(\mathsf{0}))$$

application of $>_{\mathsf{mpo}}$ (attempted):

$$\mathsf{ack}(\mathsf{0}, y) \quad >_{\mathsf{mpo}} \quad \mathsf{s}(y) \qquad ✓$$

$$\mathsf{ack}(\mathsf{s}(x), \mathsf{s}(y)) \quad \not>_{\mathsf{mpo}} \quad \mathsf{ack}(x, \mathsf{ack}(\mathsf{s}(x), y)) \qquad ✗$$

Question

how to prove termination of $\mathcal{R}_{\mathsf{ack}}$?

Question

how to prove termination of $\mathcal{R}_{\mathsf{ack}}$?

Definition DP($\mathcal{R}$)

- let $\mathcal{F}^\sharp := \mathcal{F} \cup \{f^\sharp \mid f \text{ defined}\}$; $f^\sharp$ is called dependency pair symbol

Question

how to prove termination of $\mathcal{R}_{\mathsf{ack}}$?

Definition

$\mathsf{DP}(\mathcal{R})$

- let $\mathcal{F}^\sharp := \mathcal{F} \cup \{f^\sharp \mid f \text{ defined}\}$; $f^\sharp$ is called dependency pair symbol
- for $t = f(t_1, \ldots, t_n)$, let $t^\sharp = f^\sharp(t_1, \ldots, t_n)$

Question

how to prove termination of $\mathcal{R}_{\mathsf{ack}}$?

Definition

DP($\mathcal{R}$)

- let $\mathcal{F}^\sharp := \mathcal{F} \cup \{f^\sharp \mid f \text{ defined}\}$; $f^\sharp$ is called dependency pair symbol
- for $t = f(t_1, \ldots, t_n)$, let $t^\sharp = f^\sharp(t_1, \ldots, t_n)$
- for $l \to r \in \mathcal{R}$ and $u = f(t_1, \ldots, t_n)$ subterm of r, f defined
  $l^\sharp \to u^\sharp$ is a dependency pair

Question

how to prove termination of $\mathcal{R}_{\mathsf{ack}}$?

Definition

DP($\mathcal{R}$)

- let $\mathcal{F}^\sharp := \mathcal{F} \cup \{f^\sharp \mid f \text{ defined}\}$; $f^\sharp$ is called dependency pair symbol
- for $t = f(t_1, \ldots, t_n)$, let $t^\sharp = f^\sharp(t_1, \ldots, t_n)$
- for $l \to r \in \mathcal{R}$ and $u = f(t_1, \ldots, t_n)$ subterm of r, f defined
  $l^\sharp \to u^\sharp$ is a dependency pair
- the set of all dependency pairs is denoted as $\mathsf{DP}(\mathcal{R})$

Question

how to prove termination of $\mathcal{R}_{\mathsf{ack}}$?

Definition

DP($\mathcal{R}$)

- let $\mathcal{F}^\sharp := \mathcal{F} \cup \{f^\sharp \mid f \text{ defined}\}$; $f^\sharp$ is called dependency pair symbol
- for $t = f(t_1, \ldots, t_n)$, let $t^\sharp = f^\sharp(t_1, \ldots, t_n)$
- for $l \to r \in \mathcal{R}$ and $u = f(t_1, \ldots, t_n)$ subterm of r, f defined
  $l^\sharp \to u^\sharp$ is a dependency pair
- the set of all dependency pairs is denoted as DP($\mathcal{R}$)

Example

① $\mathsf{ack}^\sharp(\mathsf{s}(x), \mathsf{s}(y)) \to \mathsf{ack}^\sharp(x, \mathsf{ack}(\mathsf{s}(x), y))$

② $\mathsf{ack}^\sharp(\mathsf{s}(x), \mathsf{s}(y)) \to \mathsf{ack}^\sharp(\mathsf{s}(x), y)$

③ $\mathsf{ack}^\sharp(\mathsf{s}(x), 0) \to \mathsf{ack}^\sharp(x, \mathsf{s}(0))$

Question

how to prove termination of $\mathcal{R}_{\mathsf{ack}}$?

Definition

DP($\mathcal{R}$)

- let $\mathcal{F}^\sharp := \mathcal{F} \cup \{f^\sharp \mid f \text{ defined}\}$; $f^\sharp$ is called dependency pair symbol
- for $t = f(t_1, \dots, t_n)$, let $t^\sharp = f^\sharp(t_1, \dots, t_n)$
- for $l \to r \in \mathcal{R}$ and $u = f(t_1, \dots, t_n)$ subterm of r, f defined $l^\sharp \to u^\sharp$ is a dependency pair
- the set of all dependency pairs is denoted as DP($\mathcal{R}$)

Example

① $\mathsf{ack}^\sharp(\mathsf{s}(x), \mathsf{s}(y)) \to \mathsf{ack}^\sharp(x, \mathsf{ack}(\mathsf{s}(x), y))$

② $\mathsf{ack}^\sharp(\mathsf{s}(x), \mathsf{s}(y)) \to \mathsf{ack}^\sharp(\mathsf{s}(x), y)$

③ $\mathsf{ack}^\sharp(\mathsf{s}(x), 0) \to \mathsf{ack}^\sharp(x, \mathsf{s}(0))$

Answer

we use DP framework + subterm criterion processor

Definition

DP problem

DP problem is a pair $(\mathcal{P}, \mathcal{R})$ such that root symbols of rules in $\mathcal{P}$

- do not occur in $\mathcal{R}$
- do not occur as proper subterms of the left- and right-hand side of rules in $\mathcal{P}$

Definition

DP problem

DP problem is a pair $(\mathcal{P}, \mathcal{R})$ such that root symbols of rules in $\mathcal{P}$

- do not occur in $\mathcal{R}$
- do not occur as proper subterms of the left- and right-hand side of rules in $\mathcal{P}$

Example

$(\mathsf{DP}(\mathcal{R}_{\mathsf{ack}}), \mathcal{R}_{\mathsf{ack}})$ is a DP problem

Definition DP problem

DP problem is a pair $(\mathcal{P}, \mathcal{R})$ such that root symbols of rules in $\mathcal{P}$

- do not occur in $\mathcal{R}$
- do not occur as proper subterms of the left- and right-hand side of rules in $\mathcal{P}$

Example

$(\mathsf{DP}(\mathcal{R}_{\mathsf{ack}}), \mathcal{R}_{\mathsf{ack}})$ is a DP problem

Definition finite DP problem

a DP problem is finite if $\neg\ \exists$

$$t_1 \xrightarrow{*}_{\mathcal{R}} t_2 \to_{\mathcal{P}} t_3 \xrightarrow{*}_{\mathcal{R}} t_4 \to_{\mathcal{P}} \cdots$$

Definition DP problem

DP problem is a pair $(\mathcal{P}, \mathcal{R})$ such that root symbols of rules in $\mathcal{P}$

- do not occur in $\mathcal{R}$
- do not occur as proper subterms of the left- and right-hand side of rules in $\mathcal{P}$

Example

$(\mathsf{DP}(\mathcal{R}_{\mathsf{ack}}), \mathcal{R}_{\mathsf{ack}})$ is a DP problem

Definition finite DP problem

a DP problem is finite if $\neg\ \exists$

$$t_1 \xrightarrow{*}_{\mathcal{R}} t_2 \to_{\mathcal{P}} t_3 \xrightarrow{*}_{\mathcal{R}} t_4 \to_{\mathcal{P}} \cdots$$

$t_1, t_2, \ldots$ terminating with respect to $\mathcal{R}$

Definition

DP problem

DP problem is a pair $(\mathcal{P}, \mathcal{R})$ such that root symbols of rules in $\mathcal{P}$

- do not occur in $\mathcal{R}$
- do not occur as proper subterms of the left- and right-hand side of rules in $\mathcal{P}$

Example

$(\mathrm{DP}(\mathcal{R}_{\mathsf{ack}}), \mathcal{R}_{\mathsf{ack}})$ is a DP problem

Definition

finite DP problem

a DP problem is finite if $\neg\,\exists$

$$t_1 \xrightarrow{*}_{\mathcal{R}} t_2 \rightarrow_{\mathcal{P}} t_3 \xrightarrow{*}_{\mathcal{R}} t_4 \rightarrow_{\mathcal{P}} \cdots$$

$t_1, t_2, \ldots$ terminating with respect to $\mathcal{R}$

Theorem

TRS $\mathcal{R}$ is terminating iff DP problem $(\mathrm{DP}(\mathcal{R}), \mathcal{R})$ is finite

Definition DP processor

a DP processor Φ is a mapping from a DP problem to a set of DP problems

a DP processor Φ is

Definition

DP processor

a DP processor Φ is a mapping from a DP problem to a set of DP problems

a DP processor Φ is

- is sound if $(\mathcal{P}, \mathcal{R})$ is finite, whenever all DP problems in $\Phi((\mathcal{P}, \mathcal{R}))$ are finite

Definition

DP processor

a DP processor Φ is a mapping from a DP problem to a set of DP problems

a DP processor Φ is

- is sound if $(\mathcal{P}, \mathcal{R})$ is finite, whenever all DP problems in $\Phi((\mathcal{P}, \mathcal{R}))$ are finite
- is complete if $(\mathcal{P}, \mathcal{R})$ is infinite, whenever one DP problem in $\Phi((\mathcal{P}, \mathcal{R}))$ is infinite

Definition DP processor

a DP processor Φ is a mapping from a DP problem to a set of DP problems

a DP processor Φ is

- is sound if $(\mathcal{P}, \mathcal{R})$ is finite, whenever all DP problems in $\Phi((\mathcal{P}, \mathcal{R}))$ are finite
- is complete if $(\mathcal{P}, \mathcal{R})$ is infinite, whenever one DP problem in $\Phi((\mathcal{P}, \mathcal{R}))$ is infinite

Definition $\mathsf{DG}(\mathcal{P}, \mathcal{R})$

- the nodes of the dependency graph $\mathsf{DG}(\mathcal{P}, \mathcal{R})$ of $(\mathcal{P}, \mathcal{R})$ are rules in $\mathcal{P}$

Definition DP processor

a DP processor Φ is a mapping from a DP problem to a set of DP problems

a DP processor Φ is

- is sound if $(\mathcal{P}, \mathcal{R})$ is finite, whenever all DP problems in $\Phi((\mathcal{P}, \mathcal{R}))$ are finite
- is complete if $(\mathcal{P}, \mathcal{R})$ is infinite, whenever one DP problem in $\Phi((\mathcal{P}, \mathcal{R}))$ is infinite

Definition $\mathsf{DG}(\mathcal{P}, \mathcal{R})$

- the nodes of the dependency graph $\mathsf{DG}(\mathcal{P}, \mathcal{R})$ of $(\mathcal{P}, \mathcal{R})$ are rules in $\mathcal{P}$
- $\exists$ edge from $s \to t$ to $u \to v$, if $\exists$ substitutions σ, τ and $t\sigma \to^*_{\mathcal{R}} u\tau$

Definition

DP processor

a DP processor Φ is a mapping from a DP problem to a set of DP problems

a DP processor Φ is

- is sound if $(\mathcal{P}, \mathcal{R})$ is finite, whenever all DP problems in $\Phi((\mathcal{P}, \mathcal{R}))$ are finite
- is complete if $(\mathcal{P}, \mathcal{R})$ is infinite, whenever one DP problem in $\Phi((\mathcal{P}, \mathcal{R}))$ is infinite

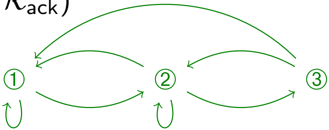
Definition

$\mathrm{DG}(\mathcal{P}, \mathcal{R})$

- the nodes of the dependency graph $\mathrm{DG}(\mathcal{P}, \mathcal{R})$ of $(\mathcal{P}, \mathcal{R})$ are rules in $\mathcal{P}$
- $\exists$ edge from $s \to t$ to $u \to v$, if $\exists$ substitutions σ, τ and $t\sigma \to_{\mathcal{R}}^{*} u\tau$

Example

consider $\mathrm{DG}(\mathrm{DG}(\mathcal{R}_{\mathsf{ack}}), \mathcal{R}_{\mathsf{ack}})$



Definition dependency graph processor

$$\Phi\colon (\mathcal{P}, \mathcal{R}) \mapsto \{(\mathcal{C}, \mathcal{R}) \mid \mathcal{C} \text{ is strongly connected in } \mathsf{DG}(\mathcal{P}, \mathcal{R})\}$$

Definition

dependency graph processor

$$\Phi\colon (\mathcal{P},\mathcal{R}) \mapsto \{(\mathcal{C},\mathcal{R}) \mid \mathcal{C} \text{ is strongly connected in } \mathsf{DG}(\mathcal{P},\mathcal{R})\}$$

a simple projection π projects arguments of dependency pair symbols and leaves other symbols unchanged

Definition dependency graph processor

$$\Phi\colon (\mathcal{P},\mathcal{R}) \mapsto \{(\mathcal{C},\mathcal{R}) \mid \mathcal{C} \text{ is strongly connected in } \mathsf{DG}(\mathcal{P},\mathcal{R})\}$$

a simple projection π projects arguments of dependency pair symbols and leaves other symbols unchanged

Definition subterm criterion processor

$$\Psi\colon (\mathcal{P},\mathcal{R}) \mapsto \begin{cases} \{(\{l = r\colon \pi(l) = \pi(r)\},\mathcal{R})\} & \text{if } \pi(\mathcal{P}) \subseteq \trianglerighteq \\ \{(\mathcal{P},\mathcal{R})\} & \text{otherwise} \end{cases}$$

Definition dependency graph processor

$$\Phi\colon (\mathcal{P},\mathcal{R}) \mapsto \{(\mathcal{C},\mathcal{R}) \mid \mathcal{C} \text{ is strongly connected in } \mathsf{DG}(\mathcal{P},\mathcal{R})\}$$

a simple projection π projects arguments of dependency pair symbols and leaves other symbols unchanged

Definition subterm criterion processor

$$\Psi\colon (\mathcal{P},\mathcal{R}) \mapsto \begin{cases} \{(\{l = r \colon \pi(l) = \pi(r)\},\mathcal{R})\} & \text{if } \pi(\mathcal{P}) \subseteq \trianglerighteq \\ \{(\mathcal{P},\mathcal{R})\} & \text{otherwise} \end{cases}$$

Lemma

the processors Φ, Ψ are sound and complete

Definition

dependency graph processor

$$\Phi\colon (\mathcal{P},\mathcal{R}) \mapsto \{(\mathcal{C},\mathcal{R}) \mid \mathcal{C} \text{ is strongly connected in } \mathsf{DG}(\mathcal{P},\mathcal{R})\}$$

a simple projection π projects arguments of dependency pair symbols and leaves other symbols unchanged

Definition

subterm criterion processor

$$\Psi\colon (\mathcal{P},\mathcal{R}) \mapsto \begin{cases} \{(\{l = r\colon \pi(l) = \pi(r)\},\mathcal{R})\} & \text{if } \pi(\mathcal{P}) \subseteq \trianglerighteq \\ \{(\mathcal{P},\mathcal{R})\} & \text{otherwise} \end{cases}$$

Lemma

the processors Φ, Ψ are sound and complete

Example

application of Ψ on $(\mathsf{DP}(\mathcal{R}),\mathcal{R})$:

① $\mathsf{ack}^\sharp(\mathsf{s}(x),\mathsf{s}(y)) \to \mathsf{ack}^\sharp(x,\mathsf{ack}(\mathsf{s}(x),y))$

② $\mathsf{ack}^\sharp(\mathsf{s}(x),\mathsf{s}(y)) \to \mathsf{ack}^\sharp(\mathsf{s}(x),y)$

③ $\mathsf{ack}^\sharp(\mathsf{s}(x),0) \to \mathsf{ack}^\sharp(x,\mathsf{s}(0))$

Definition

dependency graph processor

$$\Phi\colon (\mathcal{P},\mathcal{R}) \mapsto \{(\mathcal{C},\mathcal{R}) \mid \mathcal{C} \text{ is strongly connected in } \mathsf{DG}(\mathcal{P},\mathcal{R})\}$$

a simple projection π projects arguments of dependency pair symbols and leaves other symbols unchanged

Definition

subterm criterion processor

$$\Psi\colon (\mathcal{P},\mathcal{R}) \mapsto \begin{cases} \{(\{l = r\colon \pi(l) = \pi(r)\},\mathcal{R})\} & \text{if } \pi(\mathcal{P}) \subseteq \trianglerighteq \\ \{(\mathcal{P},\mathcal{R})\} & \text{otherwise} \end{cases}$$

Lemma

the processors Φ, Ψ are sound and complete

Example

application of Ψ on $(\mathsf{DP}(\mathcal{R}),\mathcal{R})$:

π_1

① $\mathsf{ack}^\sharp(\mathsf{s}(x),\mathsf{s}(y)) \to \mathsf{ack}^\sharp(x,\mathsf{ack}(\mathsf{s}(x),y))$

② $\mathsf{ack}^\sharp(\mathsf{s}(x),\mathsf{s}(y)) \to \mathsf{ack}^\sharp(\mathsf{s}(x),y)$

③ $\mathsf{ack}^\sharp(\mathsf{s}(x),0) \to \mathsf{ack}^\sharp(x,\mathsf{s}(0))$

Definition

dependency graph processor

$$\Phi\colon (\mathcal{P},\mathcal{R}) \mapsto \{(\mathcal{C},\mathcal{R}) \mid \mathcal{C} \text{ is strongly connected in } \mathsf{DG}(\mathcal{P},\mathcal{R})\}$$

a simple projection π projects arguments of dependency pair symbols and leaves other symbols unchanged

Definition

subterm criterion processor

$$\Psi\colon (\mathcal{P},\mathcal{R}) \mapsto \begin{cases} \{(\{l = r\colon \pi(l) = \pi(r)\},\mathcal{R})\} & \text{if } \pi(\mathcal{P}) \subseteq \trianglerighteq \\ \{(\mathcal{P},\mathcal{R})\} & \text{otherwise} \end{cases}$$

Lemma

the processors Φ, Ψ are sound and complete

Example

application of Ψ on $(\mathsf{DP}(\mathcal{R}),\mathcal{R})$:

		π_1
①	$\mathsf{ack}^\sharp(\mathsf{s}(x),\mathsf{s}(y)) \to \mathsf{ack}^\sharp(x,\mathsf{ack}(\mathsf{s}(x),y))$	$\mathsf{s}(x) \vartriangleright x$
②	$\mathsf{ack}^\sharp(\mathsf{s}(x),\mathsf{s}(y)) \to \mathsf{ack}^\sharp(\mathsf{s}(x),y)$	$\mathsf{s}(x) \trianglerighteq \mathsf{s}(x)$
③	$\mathsf{ack}^\sharp(\mathsf{s}(x),0) \to \mathsf{ack}^\sharp(x,\mathsf{s}(0))$	$\mathsf{s}(x) \vartriangleright x$

Definition

dependency graph processor

$$\Phi\colon (\mathcal{P},\mathcal{R}) \mapsto \{(\mathcal{C},\mathcal{R}) \mid \mathcal{C} \text{ is strongly connected in } \mathrm{DG}(\mathcal{P},\mathcal{R})\}$$

a simple projection π projects arguments of dependency pair symbols and leaves other symbols unchanged

Definition

subterm criterion processor

$$\Psi\colon (\mathcal{P},\mathcal{R}) \mapsto \begin{cases} \{(\{l = r\colon \pi(l) = \pi(r)\},\mathcal{R})\} & \text{if } \pi(\mathcal{P}) \subseteq \trianglerighteq \\ \{(\mathcal{P},\mathcal{R})\} & \text{otherwise} \end{cases}$$

Lemma

the processors Φ, Ψ are sound and complete

Example

application of Ψ on $(\mathrm{DP}(\mathcal{R}),\mathcal{R})$:

π_1

① $\mathsf{ack}^\sharp(\mathsf{s}(x),\mathsf{s}(y)) \to \mathsf{ack}^\sharp(x,\mathsf{ack}(\mathsf{s}(x),y))$ $\mathsf{s}(x) \triangleright x$ ✓

② $\mathsf{ack}^\sharp(\mathsf{s}(x),\mathsf{s}(y)) \to \mathsf{ack}^\sharp(\mathsf{s}(x),y)$ $\mathsf{s}(x) \trianglerighteq \mathsf{s}(x)$

③ $\mathsf{ack}^\sharp(\mathsf{s}(x),0) \to \mathsf{ack}^\sharp(x,\mathsf{s}(0))$ $\mathsf{s}(x) \triangleright x$ ✓

Definition

dependency graph processor

$$\Phi\colon (\mathcal{P},\mathcal{R}) \mapsto \{(\mathcal{C},\mathcal{R}) \mid \mathcal{C} \text{ is strongly connected in } \mathsf{DG}(\mathcal{P},\mathcal{R})\}$$

a simple projection π projects arguments of dependency pair symbols and leaves other symbols unchanged

Definition

subterm criterion processor

$$\Psi\colon (\mathcal{P},\mathcal{R}) \mapsto \begin{cases} \{(\{l = r \colon \pi(l) = \pi(r)\},\mathcal{R})\} & \text{if } \pi(\mathcal{P}) \subseteq \trianglerighteq \\ \{(\mathcal{P},\mathcal{R})\} & \text{otherwise} \end{cases}$$

Lemma

the processors Φ, Ψ are sound and complete

Example

application of Ψ on $(\mathsf{DP}(\mathcal{R}),\mathcal{R})$:

π_1

② $\mathsf{ack}^\sharp(\mathsf{s}(x),\mathsf{s}(y)) \to \mathsf{ack}^\sharp(\mathsf{s}(x),y)$

Definition

dependency graph processor

$$\Phi\colon (\mathcal{P},\mathcal{R}) \mapsto \{(\mathcal{C},\mathcal{R}) \mid \mathcal{C} \text{ is strongly connected in } \mathrm{DG}(\mathcal{P},\mathcal{R})\}$$

a simple projection π projects arguments of dependency pair symbols and leaves other symbols unchanged

Definition

subterm criterion processor

$$\Psi\colon (\mathcal{P},\mathcal{R}) \mapsto \begin{cases} \{(\{l = r\colon \pi(l) = \pi(r)\},\mathcal{R})\} & \text{if } \pi(\mathcal{P}) \subseteq \trianglerighteq \\ \{(\mathcal{P},\mathcal{R})\} & \text{otherwise} \end{cases}$$

Lemma

the processors Φ, Ψ are sound and complete

Example

application of Ψ on $(\mathrm{DP}(\mathcal{R}),\mathcal{R})$:

		π_1	π_2
②	$\mathsf{ack}^\sharp(\mathsf{s}(x),\mathsf{s}(y)) \to \mathsf{ack}^\sharp(\mathsf{s}(x),y)$		$\mathsf{s}(y) \triangleright y$

Definition

dependency graph processor

$$\Phi\colon (\mathcal{P},\mathcal{R}) \mapsto \{(\mathcal{C},\mathcal{R}) \mid \mathcal{C} \text{ is strongly connected in } \mathrm{DG}(\mathcal{P},\mathcal{R})\}$$

a simple projection π projects arguments of dependency pair symbols and leaves other symbols unchanged

Definition

subterm criterion processor

$$\Psi\colon (\mathcal{P},\mathcal{R}) \mapsto \begin{cases} \{(\{l = r \colon \pi(l) = \pi(r)\},\mathcal{R})\} & \text{if } \pi(\mathcal{P}) \subseteq \trianglerighteq \\ \{(\mathcal{P},\mathcal{R})\} & \text{otherwise} \end{cases}$$

Lemma

the processors Φ, Ψ are sound and complete

Example

application of Ψ on $(\mathrm{DP}(\mathcal{R}),\mathcal{R})$:

		π_1	π_2	
②	$\mathsf{ack}^\sharp(\mathsf{s}(x),\mathsf{s}(y)) \to \mathsf{ack}^\sharp(\mathsf{s}(x),y)$		$\mathsf{s}(y) \triangleright y$	✓

Definition

dependency graph processor

$$\Phi\colon (\mathcal{P},\mathcal{R}) \mapsto \{(\mathcal{C},\mathcal{R}) \mid \mathcal{C} \text{ is strongly connected in } \mathrm{DG}(\mathcal{P},\mathcal{R})\}$$

a simple projection π projects arguments of dependency pair symbols and leaves other symbols unchanged

Definition

subterm criterion processor

$$\Psi\colon (\mathcal{P},\mathcal{R}) \mapsto \begin{cases} \{(\{l = r\colon \pi(l) = \pi(r)\},\mathcal{R})\} & \text{if } \pi(\mathcal{P}) \subseteq \trianglerighteq \\ \{(\mathcal{P},\mathcal{R})\} & \text{otherwise} \end{cases}$$

Lemma

the processors Φ, Ψ are sound and complete

Example

application of Ψ on $(\mathrm{DP}(\mathcal{R}),\mathcal{R})$:

π_1 π_2

Complexity of Rewriting

Definition derivation height

$$\mathrm{dh}(t, \to) = \max\{n \mid \exists u \; t \to^n u\}$$
$$\mathrm{dh}(n, T, \to) = \max\{\mathrm{dh}(t, \to) \mid \exists t \in T \text{ and } |t| \leqslant n\}$$

Complexity of Rewriting

Definition derivation height

$$\mathrm{dh}(t, \to) = \max\{n \mid \exists u\ t \to^n u\}$$

$$\mathrm{dh}(n, T, \to) = \max\{\mathrm{dh}(t, \to) \mid \exists t \in T \text{ and } |t| \leqslant n\}$$

Definition runtime complexity

$$\mathrm{rc}_{\mathcal{R}}(n) = \mathrm{dh}(n, \text{"basic terms"}, \to_{\mathcal{R}})$$

Complexity of Rewriting

Definition derivation height

$$\mathrm{dh}(t, \to) = \max\{n \mid \exists u \; t \to^n u\}$$
$$\mathrm{dh}(n, T, \to) = \max\{\mathrm{dh}(t, \to) \mid \exists t \in T \text{ and } |t| \leqslant n\}$$

Definition runtime complexity

$$\mathrm{rc}_{\mathcal{R}}(n) = \mathrm{dh}(n, \text{"basic terms"}, \to_{\mathcal{R}})$$

term $f(t_1, \ldots, t_n)$ is basic if

- f is defined
- $t_1, \ldots, t_n$ contain no defined symbols

Example

consider TRS $\mathcal{R}_{\mathsf{ack}}$

$$\mathsf{ack}(0, y) \rightarrow \mathsf{s}(y) \qquad \mathsf{ack}(\mathsf{s}(x), \mathsf{s}(y)) \rightarrow \mathsf{ack}(x, \mathsf{ack}(\mathsf{s}(x), y))$$

$$\mathsf{ack}(\mathsf{s}(x), 0) \rightarrow \mathsf{ack}(x, \mathsf{s}(0))$$

Example

consider TRS $\mathcal{R}_{\mathsf{ack}}$

$$\mathsf{ack}(0,y) \to \mathsf{s}(y) \qquad \mathsf{ack}(\mathsf{s}(x),\mathsf{s}(y)) \to \mathsf{ack}(x,\mathsf{ack}(\mathsf{s}(x),y))$$

$$\mathsf{ack}(\mathsf{s}(x),0) \to \mathsf{ack}(x,\mathsf{s}(0))$$

Example

consider $\mathcal{R}_{\mathsf{div}}$

$$x - 0 \to x \qquad 0 \div \mathsf{s}(y) \to 0$$

$$\mathsf{s}(x) - \mathsf{s}(y) \to x - y \qquad \mathsf{s}(x) \div \mathsf{s}(y) \to \mathsf{s}((x - y) \div \mathsf{s}(y))$$

Example

consider TRS $\mathcal{R}_{\mathsf{ack}}$

$$\mathsf{ack}(0, y) \to \mathsf{s}(y) \qquad \mathsf{ack}(\mathsf{s}(x), \mathsf{s}(y)) \to \mathsf{ack}(x, \mathsf{ack}(\mathsf{s}(x), y))$$
$$\mathsf{ack}(\mathsf{s}(x), 0) \to \mathsf{ack}(x, \mathsf{s}(0))$$

Example

consider $\mathcal{R}_{\mathsf{div}}$

$$x - 0 \to x \qquad 0 \div \mathsf{s}(y) \to 0$$
$$\mathsf{s}(x) - \mathsf{s}(y) \to x - y \qquad \mathsf{s}(x) \div \mathsf{s}(y) \to \mathsf{s}((x - y) \div \mathsf{s}(y))$$

Question

what is the runtime complexity of $\mathcal{R}_{\mathsf{ack}}$?

Example

consider TRS $\mathcal{R}_{\mathsf{ack}}$

$$\mathsf{ack}(0,y) \to \mathsf{s}(y) \qquad \mathsf{ack}(\mathsf{s}(x),\mathsf{s}(y)) \to \mathsf{ack}(x,\mathsf{ack}(\mathsf{s}(x),y))$$

$$\mathsf{ack}(\mathsf{s}(x),0) \to \mathsf{ack}(x,\mathsf{s}(0))$$

Example

consider $\mathcal{R}_{\mathsf{div}}$

$$x - 0 \to x \qquad 0 \div \mathsf{s}(y) \to 0$$

$$\mathsf{s}(x) - \mathsf{s}(y) \to x - y \qquad \mathsf{s}(x) \div \mathsf{s}(y) \to \mathsf{s}((x-y) \div \mathsf{s}(y))$$

Question

what is the runtime complexity of $\mathcal{R}_{\mathsf{ack}}$?

Answer

2-recursive

Example

consider TRS $\mathcal{R}_{\mathsf{ack}}$

$$\mathsf{ack}(0,y) \to \mathsf{s}(y) \qquad \mathsf{ack}(\mathsf{s}(x),\mathsf{s}(y)) \to \mathsf{ack}(x,\mathsf{ack}(\mathsf{s}(x),y))$$
$$\mathsf{ack}(\mathsf{s}(x),0) \to \mathsf{ack}(x,\mathsf{s}(0))$$

Example

consider $\mathcal{R}_{\mathsf{div}}$

$$x - 0 \to x \qquad 0 \div \mathsf{s}(y) \to 0$$
$$\mathsf{s}(x) - \mathsf{s}(y) \to x - y \qquad \mathsf{s}(x) \div \mathsf{s}(y) \to \mathsf{s}((x - y) \div \mathsf{s}(y))$$

Question

what is the runtime complexity of $\mathcal{R}_{\mathsf{div}}$?

Answer

2-recursive

Example

consider TRS $\mathcal{R}_{\mathsf{ack}}$

$$\mathsf{ack}(0, y) \to \mathsf{s}(y) \qquad \mathsf{ack}(\mathsf{s}(x), \mathsf{s}(y)) \to \mathsf{ack}(x, \mathsf{ack}(\mathsf{s}(x), y))$$
$$\mathsf{ack}(\mathsf{s}(x), 0) \to \mathsf{ack}(x, \mathsf{s}(0))$$

Example

consider $\mathcal{R}_{\mathsf{div}}$

$$x - 0 \to x \qquad 0 \div \mathsf{s}(y) \to 0$$
$$\mathsf{s}(x) - \mathsf{s}(y) \to x - y \qquad \mathsf{s}(x) \div \mathsf{s}(y) \to \mathsf{s}((x - y) \div \mathsf{s}(y))$$

Question

what is the runtime complexity of $\mathcal{R}_{\mathsf{div}}$?

Answer

linear

How To Analyse Complexity

$$t_1 \to_{\mathcal{R}} t_2 \to_{\mathcal{R}} t_3 \to_{\mathcal{R}} \cdots$$

How To Analyse Complexity

$$t_1 \to_{\mathcal{R}} t_2 \to_{\mathcal{R}} t_3 \to_{\mathcal{R}} \dots \to_{\mathcal{R}} t_n$$

consider

1. $\exists$ termination technique such that
2. termination of $\mathcal{R}$ is certified

How To Analyse Complexity

$$t_1 \to_{\mathcal{R}} t_2 \to_{\mathcal{R}} t_3 \to_{\mathcal{R}} \dots \to_{\mathcal{R}} t_n$$

consider

1. ∃ termination technique such that
2. termination of $\mathcal{R}$ is certified

Observation

termination techniques can be used to measure the derivation height

How To Analyse Complexity

$$t_1 \to_{\mathcal{R}} t_2 \to_{\mathcal{R}} t_3 \to_{\mathcal{R}} \dots \to_{\mathcal{R}} t_n$$

consider

1. $\exists$ termination technique such that
2. termination of $\mathcal{R}$ is certified

Observation

termination techniques can be used to measure the derivation height

Example

(restricted) polynomial interpretations induce polynomial runtime complexity

G. Bonfante, A. Cichon, J. Marion, and H. Touzet.
Algorithms with Polynomial Interpretation Termination Proof.
JFP, 11(1):33–53, 2001.

The Past

Complexity in Rewriting: A History

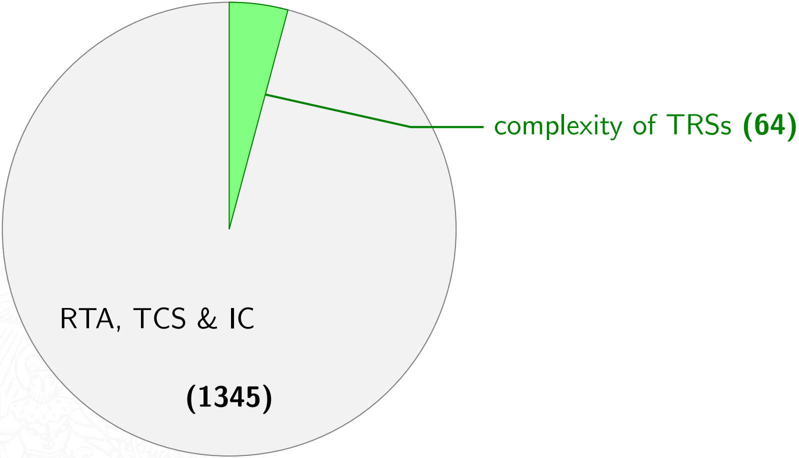
of papers

RTA, TCS & IC

(1345)

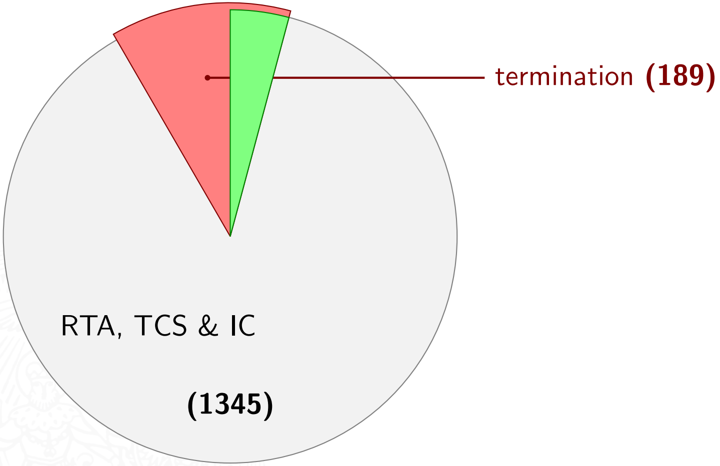
Complexity in Rewriting: A History

of papers



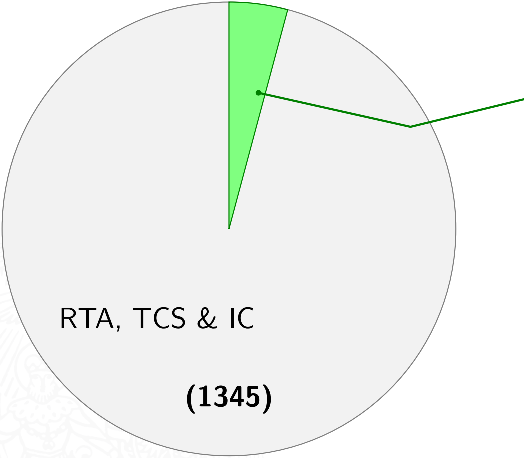
Complexity in Rewriting: A History

of papers



Complexity in Rewriting: A History

3 selected papers



ALGORITHMIC COMPLEXITY OF TERM REWRITING SYSTEMS

C. CHOPPY, S. KAPLAN, M. SORIA
Laboratoire de Recherche en Informatique
U.A. C.N.R.S. 410
Université Paris-Sud, Bât 490
F-91405 ORSAY Cédex, FRANCE
net : mcvax!inria!lri!cc, mcvax!inria!lri!kaplan, mcvax!inria!lri!soria

Introduction

Algebraic specifications are now widely used for data structuring and they turn out to be quite useful for various aspects of program development, such as prototyping, assisted program construction, proving properties, etc [BCV 85, FG 84, FGJM 84, GHW 85, Kap 86]. Some of these applications require to add a notion of computation to algebraic specifications, for instance by providing a (convergent) rewrite rule system that expresses the properties of the operators. In this context, it may be of first interest to define a notion of algorithmic complexity for an algebraic specification, or, more precisely, a notion of complexity for each operator defined in the specification. Computing operator complexity within a given specification helps understanding how evaluation costs are distributed ; it may point out "costly" operators, and motivate the search for an equivalent, but "cheaper", specification.

In [CLR 80], the cost of a term is defined as the number of rewriting steps for reducing it to its normal form, and the cost of an operator is defined as the general cost of a term obtained by applying this operator to terms in normal form. In this paper, we further formalize this notion of operator complexity and investigate its computation through analysis methods developed for instance in [Stey 84] and [Fla 87]. We show how these methods apply to the computation of the enumerative series over the terms of an algebraic specification. We define the notion of *regular* rewriting systems, and consider cost series of operators that are described by such systems. We show how these analysis methods apply to compute such costs and provide an asymptotic evaluation of the average cost of an operator. Our results allow costs to be computed without any explicit manipulation of series. We provide an eventual user with ready-to-use formulae, where the different parameters only depend on the "geometry" of the system, e.g. the number of constructors in the left handside of rules, number of occurrences of a derived operator in the right handside, etc.

Quantitative evaluation of rewriting systems had not yet been studied under such an approach (except in [CLR 80]), to our knowledge. From a different point of view, complexity of algebraic implementations has been studied in [BBWT 81, E&M 81, etc.] w.r.t. computability issues.

1. Introductory example

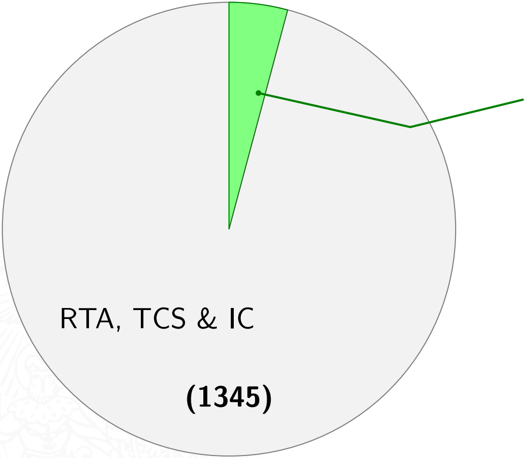
Let us assume that one wants to evaluate the average cost of a given computation on some data set. The data belong to a set of objects, a size can be computed for each object ; let D_n denote the set of

Algorithmic Complexity
of Term Rewrite Systems;
Choppy et al., RTA 1987

to be continued

Complexity in Rewriting: A History

3 selected papers



Termination proofs and the length of derivations
(Preliminary version)

Dieter Hofbauer
Technische Universität Berlin

Clemens Lautemann *
Universität Bremen

Abstract

The derivation height of a term t, relative to a set R of *rewrite rules*, $dh_R(t)$, is the length of a longest derivation from t. We investigate in which way certain termination proof methods impose bounds on dh_R. In particular we show that, if termination of R can be proved by *polynomial interpretation* then dh_R is bounded from above by a doubly exponential function[1], whereas termination proofs by *Knuth–Bendix ordering* are possible even for systems where dh_R cannot be bounded by any primitive recursive functions. For both methods, conditions are given which guarantee a singly exponential upper bound on dh_R. Moreover, all upper bounds are tight.

1 Introduction

The complexity of a term rewriting system R can be measured in different ways. In this paper, we consider the *derivation height*, expressed in the function Dh_R, where $Dh_R(n)$ is the length of a longest derivation which starts from a term of size n. Thus Dh_R describes the worst possible behaviour of R.

Obviously, Dh_R is partial recursive, and is total if and only if R is terminating. The rate of growth of Dh_R then quantifies termination of R in much the same way as the rate of convergence of a series (q_n) quantifies its convergence.

As termination is one of the basic concepts in term rewriting theory, crucial e.g. in Knuth–Bendix completion procedures or for inductive proofs, a lot of techniques have been developed to support or even automatically generate termination proofs (see [De87] for a survey).

Proving termination with one of these specific techniques in general proves more than just the absence of infinite derivation sequences. It turns out that such a proof in many cases implies an upper bound on Dh_R, so that the rate of growth of Dh_R can be used for measuring the power of termination proof methods.

A similar point of view is taken in [DO88], where Dershowitz and Okada investigate the order type of certain simplification orderings. However, the order type of such an ordering as such does not place a bound on its power for termination proofs, as it does not say anything about the intersection of the simplification ordering with the rewrite relation induced by a finite term rewriting system.

---

*in the early stages of this research supported by ESPRIT project PROSPECTRA, ref. #390

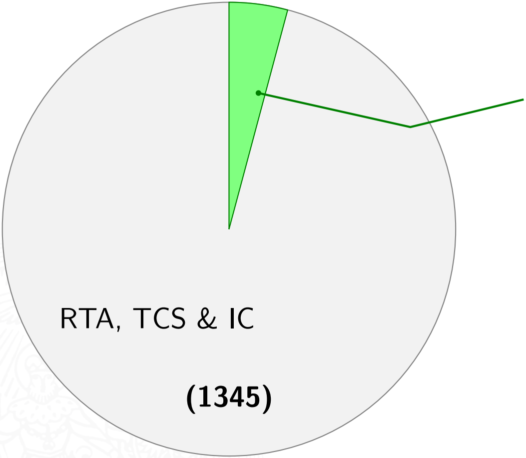
[1] this result was independently obtained by Oliver Geupel

Termination Proofs and the Length of Derivations; Hofbauer, Lautemann, RTA 1989

polynomial interpretations induce double-exponential

Complexity in Rewriting: A History

3 selected papers



Complexity classes and rewrite systems with polynomial interpretation

G. Bonfante, A. Cichon, J.Y Marion and H. Touzet

Loria, Calligramme project,
B.P. 239, 54506 Vandœuvre-lès-Nancy Cedex, France
{bonfante,cichon,marionjy,touzet}@loria.fr.

Abstract. We are concerned with functions over words which are computable by means of a rewrite system admitting polynomial interpretation termination proofs. We classify them according to the interpretations of successor symbols. This leads to the definition of three classes, which turn out to be exactly the poly-time, linear exponential-time and doubly linear exponential time computable functions. As a consequence, we also characterize the linear space computable functions.

1 Introduction

We are interested in studying the relationship between termination orderings for rewrite systems and feasible computation. One might suspect that polynomial interpretations, introduced in [5], would be a good candidate for the investigation of small complexity classes of functions. Various implementations have been carried out [2,6]. However, it has been shown in [8] that a rewrite system with a polynomial interpretation termination proof can admit doubly-exponential derivation lengths.

The work of [3] initiated an alternative analysis of the effects of termination proofs based on the use of polynomial interpretations. It was shown that a particularly important aspect was the interpretations of the constructors which were restricted to the set $\mathbb{N} = \{0, s\}$, 0 a constant and s a unary successor symbol.

The present paper reconsiders the situation. Firstly, we study functions over strings, that is, rewrite systems based on several successors. Indeed words are the canonical domains for computations. Secondly, we provide a stratification of the systems which is related to the kind of polynomials interpreting the successors.

We show that Ptime functions are characterized by rewrite systems admitting polynomial interpretation termination proofs where the successors are interpreted by *linear* polynomials with leading coefficient 1. As a consequence, functions over $\mathbb{N}$ turn out to be precisely Linspace functions.

Within the same framework, linear exponential time functions, i.e. functions in Etime = Dtime($2^{O(n)}$), are characterized by rewrite systems admitting polynomial interpretation termination proofs where the successors are interpreted by *linear* polynomials.

(Complexity Classes and Rewrite Systems with Polynomial Interpretation, Bonfante, Cichon, Marion, Touzet, CSL 1998)

(restricted) polynomial interpretations induce polynomial

Algorithmic Complexity of Term Rewrite Systems

- first paper (during RTA) that mentions complexity
- the cost of a term is roughly the runtime complexity
- the cost of an operator f is the runtime complexity starting with terms of form $f(t_1, \dots, t_n)$
- analyse the average cost of an operator, using generics series and some analysis
- only applicable for completely defined, orthogonal constructor TRSs with no nesting of defined symbols in the right-hand side

The Present

Runtime Complexity Analysis in the Large

Example

consider the TRS $\mathcal{R}_{\mathsf{mlt}}$

$$\mathsf{ack}_k(\overline{0}, 0, n) \to \mathsf{s}(n) \qquad \mathsf{ack}_k(\bar{l}, \mathsf{s}(m), 0) \to \mathsf{ack}_k(\bar{l}, m, \mathsf{s}(0))$$

$$\mathsf{ack}_k(\bar{l}, \mathsf{s}(m), \mathsf{s}(n)) \to \mathsf{ack}_k(\bar{l}, m, \mathsf{ack}_k(\bar{l}, \mathsf{s}(m), n))$$

$$\mathsf{ack}_k(\bar{l}, \mathsf{s}(l_i), \overline{0}, n) \to \mathsf{ack}_k(\bar{l}, l_i, n, \overline{0}, n)$$

Runtime Complexity Analysis in the Large

Example

consider the TRS $\mathcal{R}_{\mathsf{mlt}}$

$$\mathsf{ack}_k(\overline{0}, 0, n) \to \mathsf{s}(n) \qquad \mathsf{ack}_k(\bar{l}, \mathsf{s}(m), 0) \to \mathsf{ack}_k(\bar{l}, m, \mathsf{s}(0))$$

$$\mathsf{ack}_k(\bar{l}, \mathsf{s}(m), \mathsf{s}(n)) \to \mathsf{ack}_k(\bar{l}, m, \mathsf{ack}_k(\bar{l}, \mathsf{s}(m), n))$$

$$\mathsf{ack}_k(\bar{l}, \mathsf{s}(l_i), \overline{0}, n) \to \mathsf{ack}_k(\bar{l}, l_i, n, \overline{0}, n)$$

Observation

termination of $\mathcal{R}_{\mathsf{mlt}}$ follows by iterated use of the subterm criterion processor

Runtime Complexity Analysis in the Large

Example

consider the TRS $\mathcal{R}_{\mathsf{mlt}}$

$$\mathsf{ack}_k(\overline{0}, 0, n) \to \mathsf{s}(n) \qquad \mathsf{ack}_k(\bar{l}, \mathsf{s}(m), 0) \to \mathsf{ack}_k(\bar{l}, m, \mathsf{s}(0))$$

$$\mathsf{ack}_k(\bar{l}, \mathsf{s}(m), \mathsf{s}(n)) \to \mathsf{ack}_k(\bar{l}, m, \mathsf{ack}_k(\bar{l}, \mathsf{s}(m), n))$$

$$\mathsf{ack}_k(\bar{l}, \mathsf{s}(l_i), \overline{0}, n) \to \mathsf{ack}_k(\bar{l}, l_i, n, \overline{0}, n)$$

Observation

termination of $\mathcal{R}_{\mathsf{mlt}}$ follows by iterated use of the subterm criterion processor

Theorem

let $\mathcal{R}$ be a TRS such that termination of $\mathcal{R}$ follows by (iterated) use of dependency graph processor and subterm criterion processor, then

1. $\mathsf{rc}_{\mathcal{R}}$ is bounded by a multiply recursive function
2. this bound is optimal

Relation to Implicit Computational Complexity

Corollary

DP framework + subterm criterion processor characterises the multiple recursive functions

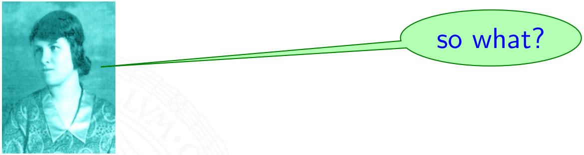
G. M. and A. Schnabl.
The Derivational Complexity Induced by the Dependency Pair Method.
In *RTA*, pages 255–269, 2009.

Relation to Implicit Computational Complexity

Corollary

DP framework + subterm criterion processor characterises the multiple recursive functions

G. M. and A. Schnabl.
The Derivational Complexity Induced by the Dependency Pair Method.
In *RTA*, pages 255–269, 2009.

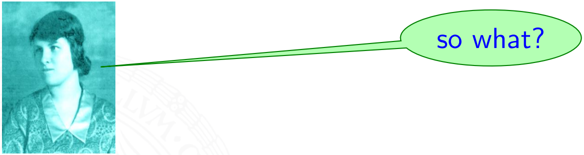


Relation to Implicit Computational Complexity

Corollary

DP framework + subterm criterion processor characterises the multiple recursive functions

G. M. and A. Schnabl.
The Derivational Complexity Induced by the Dependency Pair Method.
In *RTA*, pages 255–269, 2009.



Claim

any termination technique currently implemented in a termination tool induces multiple recursion

Runtime Complexity Analysis in the Small

Definition

weak dependency pairs

$$\mathrm{WDP}(\mathcal{R}) = \{\, l^\sharp \to \underbrace{\mathrm{COM}}_{\text{compound symbol}}(u_1^\sharp, \dots, u_n^\sharp) \mid (l \to r) \in \mathcal{R},\ r = \underbrace{C[u_1, \dots, u_n]}_{\text{no } \mathcal{D} \cup \mathcal{V} \text{ in } C} \}$$

$\mathrm{COM}(t_1, \dots, t_n)$ is t_1 if $n = 1$, and $c(t_1, \dots, t_n)$ otherwise

Runtime Complexity Analysis in the Small

Definition

weak dependency pairs

$$\mathsf{WDP}(\mathcal{R}) = \{\, l^\sharp \to \underbrace{\mathsf{COM}}_{\text{compound symbol}}(u_1^\sharp, \dots, u_n^\sharp) \mid (l \to r) \in \mathcal{R},\ r = \underbrace{C[u_1, \dots, u_n]}_{\text{no } \mathcal{D} \cup \mathcal{V} \text{ in } C} \}$$

$\mathsf{COM}(t_1, \dots, t_n)$ is t_1 if $n = 1$, and $c(t_1, \dots, t_n)$ otherwise

Example

consider $\mathsf{WDP}(\mathcal{R}_{\mathsf{div}})$:

$$x -^\sharp 0 \to x \qquad 0 \div^\sharp \mathsf{s}(y) \to \mathsf{c}$$

$$\mathsf{s}(x) -^\sharp \mathsf{s}(y) \to x -^\sharp y \qquad \mathsf{s}(x) \div^\sharp \mathsf{s}(y) \to (x - y) \div^\sharp \mathsf{s}(y)$$

Runtime Complexity Analysis in the Small

Definition

weak dependency pairs

$$\mathsf{WDP}(\mathcal{R}) = \{\, l^\sharp \to \underbrace{\mathsf{COM}}_{\text{compound symbol}}(u_1^\sharp, \dots, u_n^\sharp) \mid (l \to r) \in \mathcal{R},\ r = \underbrace{C[u_1, \dots, u_n]}_{\text{no } \mathcal{D} \cup \mathcal{V} \text{ in } C} \}$$

$\mathsf{COM}(t_1, \dots, t_n)$ is t_1 if $n = 1$, and $c(t_1, \dots, t_n)$ otherwise

Example

consider $\mathsf{WDP}(\mathcal{R}_{\mathsf{div}})$:

$$x -^\sharp 0 \to x \qquad 0 \div^\sharp \mathsf{s}(y) \to \mathsf{c}$$

$$\mathsf{s}(x) -^\sharp \mathsf{s}(y) \to x -^\sharp y \qquad \mathsf{s}(x) \div^\sharp \mathsf{s}(y) \to (x - y) \div^\sharp \mathsf{s}(y)$$

Remark

weak dependency pairs and standard dependency pairs are incomparable

Recall

TRS $\mathcal{R}$ is terminating iff DP problem $(\mathsf{DP}(\mathcal{R}), \mathcal{R})$ is finite

Recall

TRS $\mathcal{R}$ is terminating iff DP problem $(\mathrm{DP}(\mathcal{R}), \mathcal{R})$ is finite

Theorem

TRS $\mathcal{R}$ has at most polynomial runtime complexity, whenever

Recall

TRS $\mathcal{R}$ is terminating iff DP problem $(\mathsf{DP}(\mathcal{R}), \mathcal{R})$ is finite

Theorem

TRS $\mathcal{R}$ has at most polynomial runtime complexity, whenever

- $(\mathsf{WDP}(\mathcal{R}), \mathcal{R})$ is finite

Recall

TRS $\mathcal{R}$ is terminating iff DP problem $(\mathsf{DP}(\mathcal{R}), \mathcal{R})$ is finite

Theorem

TRS $\mathcal{R}$ has at most polynomial runtime complexity, whenever

- $(\mathsf{WDP}(\mathcal{R}), \mathcal{R})$ is finite
- $\to_{\mathsf{WDP}(\mathcal{R})/\mathcal{R}}$ is polynomially bounded

Recall

TRS $\mathcal{R}$ is terminating iff DP problem $(\mathsf{DP}(\mathcal{R}), \mathcal{R})$ is finite

Theorem

TRS $\mathcal{R}$ has at most polynomial runtime complexity, whenever

- $(\mathsf{WDP}(\mathcal{R}), \mathcal{R})$ is finite
- $\to_{\mathsf{WDP}(\mathcal{R})/\mathcal{R}}$ is polynomially bounded
- some technical conditions are enforced

Recall

TRS $\mathcal{R}$ is terminating iff DP problem $(\mathsf{DP}(\mathcal{R}), \mathcal{R})$ is finite

Theorem

TRS $\mathcal{R}$ has at most polynomial runtime complexity, whenever

- $(\mathsf{WDP}(\mathcal{R}), \mathcal{R})$ is finite
- $\rightarrow_{\mathsf{WDP}(\mathcal{R})/\mathcal{R}}$ is polynomially bounded
- some technical conditions are enforced

Remark

similar transfers are possible for:

- dependency graphs
- subterm criterion

N. Hirokawa and G. M.
Automated Complexity Analysis Based on the Dependency Pair Method.
In *IJCAR*, pages 364–379, 2008.

Runtime Complexity and Polytime Computability

Definition LMPO

- LMPO is a restriction of MPO: $>_{\mathsf{lmpo}} \subseteq >_{\mathsf{mpo}}$

Runtime Complexity and Polytime Computability

Definition LMPO

- LMPO is a restriction of MPO: $>_{\mathsf{lmpo}} \subseteq >_{\mathsf{mpo}}$
- $>_{\mathsf{lmpo}} = >_{\mathsf{mpo}} \cap$ predicative recursion

Runtime Complexity and Polytime Computability

Definition LMPO

- LMPO is a restriction of MPO: $>_{\mathsf{lmpo}} \subseteq >_{\mathsf{mpo}}$
- $>_{\mathsf{lmpo}} = >_{\mathsf{mpo}} \cap$ predicative recursion

predicative recursion:

$$f(0, \vec{x}; \vec{y}) >_{\mathsf{lmpo}} g(\vec{x}; \vec{y}) \qquad f(z0, \vec{x}; \vec{y}) >_{\mathsf{lmpo}} h_0(z, \vec{x}; \vec{y}, f(z, \vec{x}; \vec{y}))$$

$$f(z1, \vec{x}; \vec{y}) >_{\mathsf{lmpo}} h_1(z, \vec{x}; \vec{y}, f(z, \vec{x}; \vec{y}))$$

Runtime Complexity and Polytime Computability

Definition LMPO

- LMPO is a restriction of MPO: $>_{\mathsf{lmpo}} \subseteq >_{\mathsf{mpo}}$
- $>_{\mathsf{lmpo}} = >_{\mathsf{mpo}} \cap$ predicative recursion

predicative recursion:

$$f(0,\vec{x};\vec{y}) >_{\mathsf{lmpo}} g(\vec{x};\vec{y}) \qquad f(z0,\vec{x};\vec{y}) >_{\mathsf{lmpo}} h_0(z,\vec{x};\vec{y},f(z,\vec{x};\vec{y}))$$
$$f(z1,\vec{x};\vec{y}) >_{\mathsf{lmpo}} h_1(z,\vec{x};\vec{y},f(z,\vec{x};\vec{y}))$$

Theorem

LMPO characterises the polytime computable functions (on constructor TRS)

J.-Y. Marion.
Analysing the Implicit Complexity of Programs.
IC, 183:2–18, 2003.

Definition POP*

- $>_{\mathsf{pop}*} = >_{\mathsf{lmpo}} \cap$ no multiple recursion

Definition

POP*

- $>_{\mathsf{pop}*} = >_{\mathsf{lmpo}} \cap$ no multiple recursion

Theorem

POP* induces poly. runtime complexity (on constructor, right-linear TRS)

POP*

Definition

- $>_{\mathsf{pop*}} = >_{\mathsf{lmpo}} \cap$ no multiple recursion

Theorem

POP* induces poly. runtime complexity (on constructor, right-linear TRS)

Theorem

innermost, runtime complexity induces polytime computability on orthogonal TRS

U. Dal Lago and S. Martini.
On Constructor Rewrite Systems and the Lambda-Calculus.
In *ICALP*, pages 163–174, 2009.

Definition

POP*

- $>_{\mathsf{pop*}} = >_{\mathsf{lmpo}} \cap$ no multiple recursion

Theorem

POP* induces poly. runtime complexity (on constructor, right-linear TRS)

Theorem

innermost, outermost runtime complexity induces polytime computability on orthogonal TRS

U. Dal Lago and S. Martini.
On Constructor Rewrite Systems and the Lambda-Calculus.
In *ICALP*, pages 163–174, 2009.

U. Dal Lago and S. Martini.
Derivational Complexity is an Invariant Cost Model.
In *FOPARA*, 2009.

POP*

Definition

- $>_{\mathsf{pop*}} = >_{\mathsf{lmpo}} \cap$ no multiple recursion

Theorem

POP* induces poly. runtime complexity (on constructor, right-linear TRS)

Theorem

runtime complexity induces polytime computability

on TRS

U. Dal Lago and S. Martini.
On Constructor Rewrite Systems and the Lambda-Calculus.
In *ICALP*, pages 163–174, 2009.

U. Dal Lago and S. Martini.
Derivational Complexity is an Invariant Cost Model.
In *FOPARA*, 2009.

M. Avanzini and G. M.
Closing the Gap between Runtime Complexity and Polytime Computability.
In *RTA*, to appear, 2010.

Automated Complexity Analysis: A Snapshot

polynomial runtime complexity on TPDB 7.0.2

Automated Complexity Analysis: A Snapshot

polynomial runtime complexity on TPDB 7.0.2

dc–full
(455)

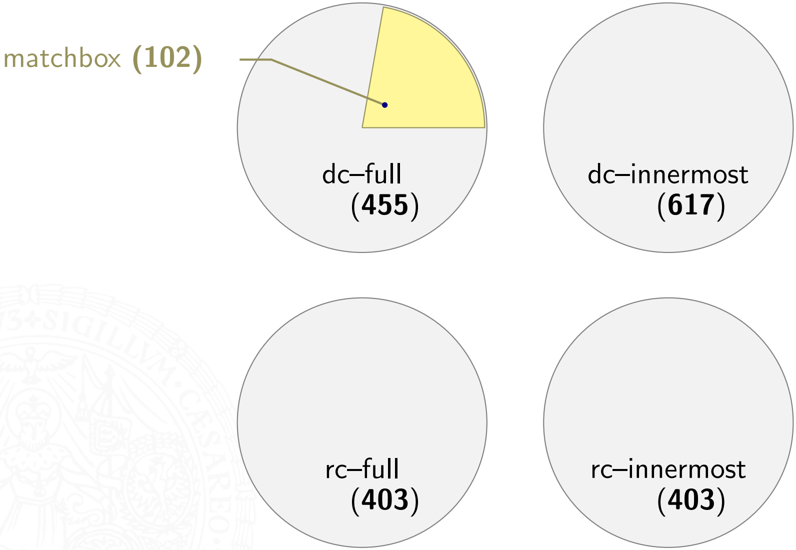
dc–innermost
(617)

rc–full
(403)

rc–innermost
(403)

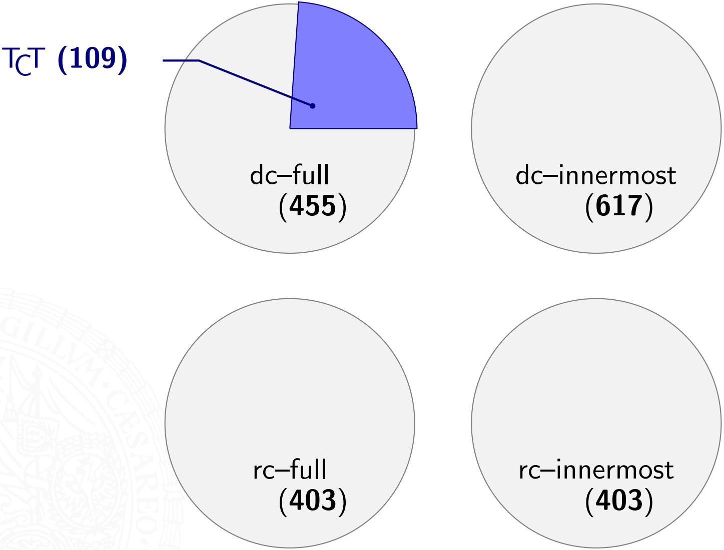
Automated Complexity Analysis: A Snapshot

polynomial runtime complexity on TPDB 7.0.2



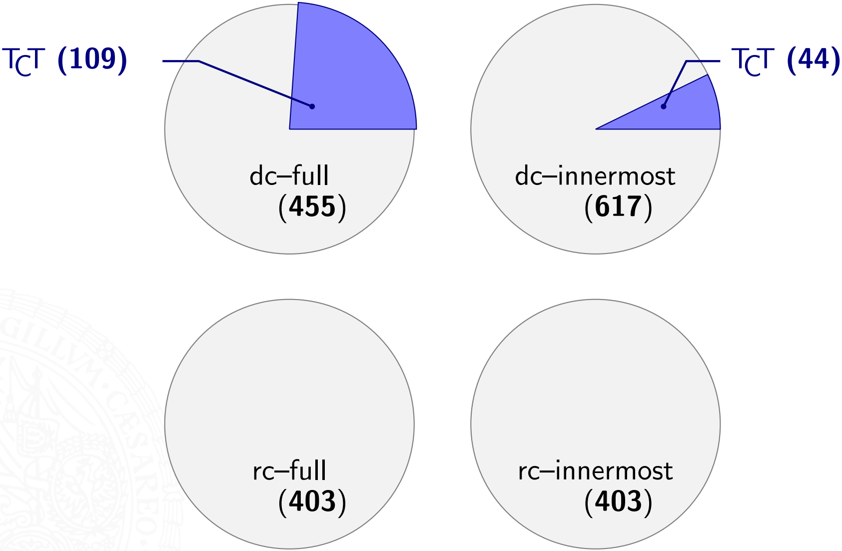
Automated Complexity Analysis: A Snapshot

polynomial runtime complexity on TPDB 7.0.2



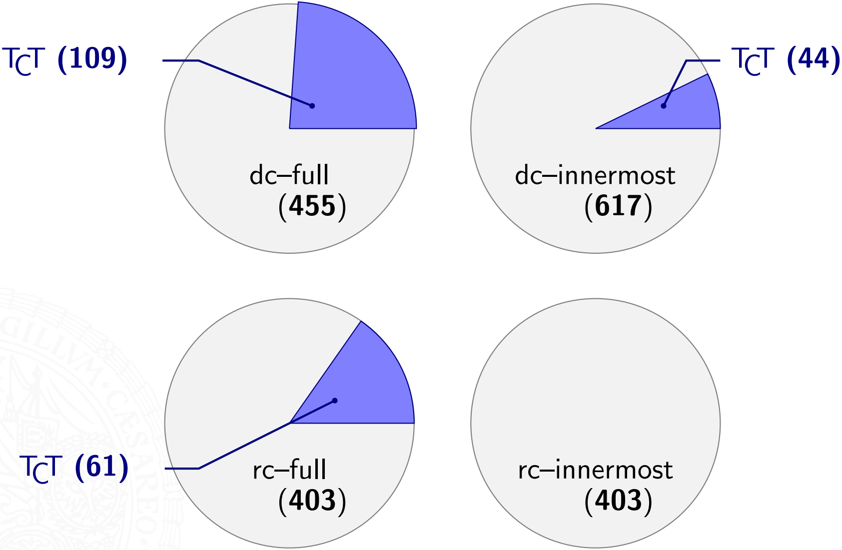
Automated Complexity Analysis: A Snapshot

polynomial runtime complexity on TPDB 7.0.2



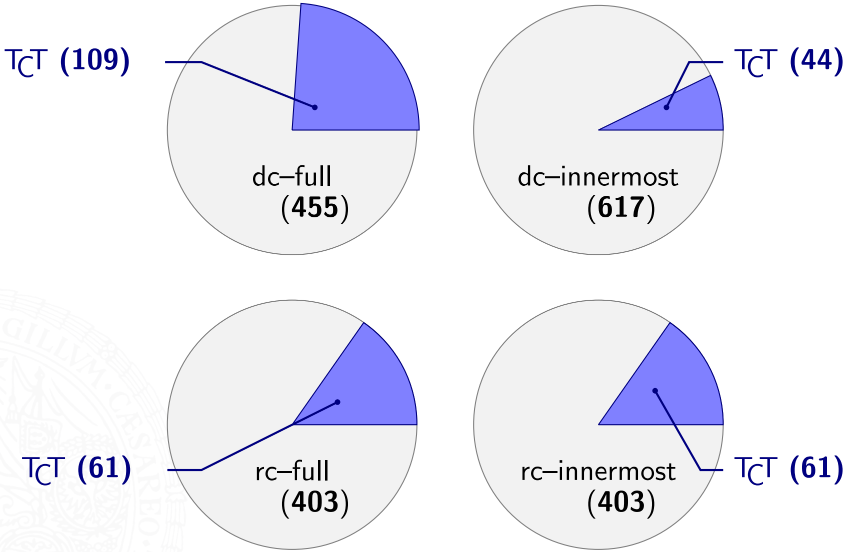
Automated Complexity Analysis: A Snapshot

polynomial runtime complexity on TPDB 7.0.2



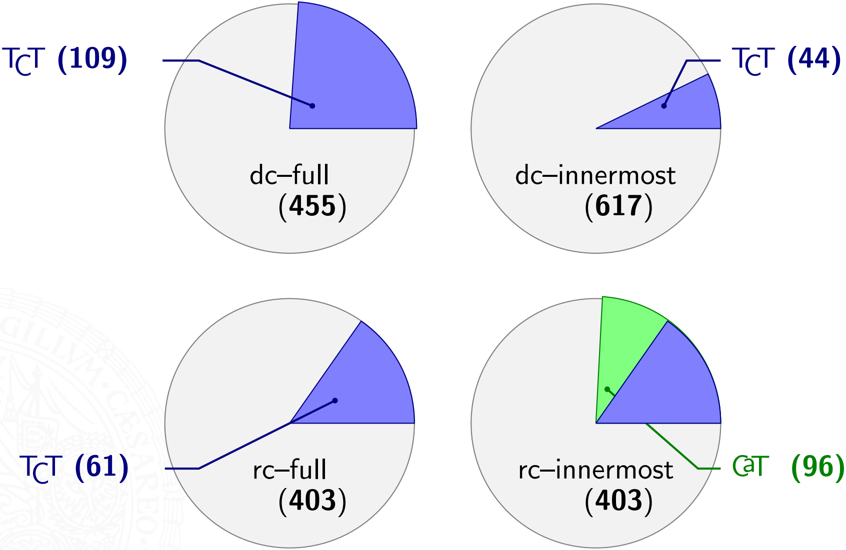
Automated Complexity Analysis: A Snapshot

polynomial runtime complexity on TPDB 7.0.2



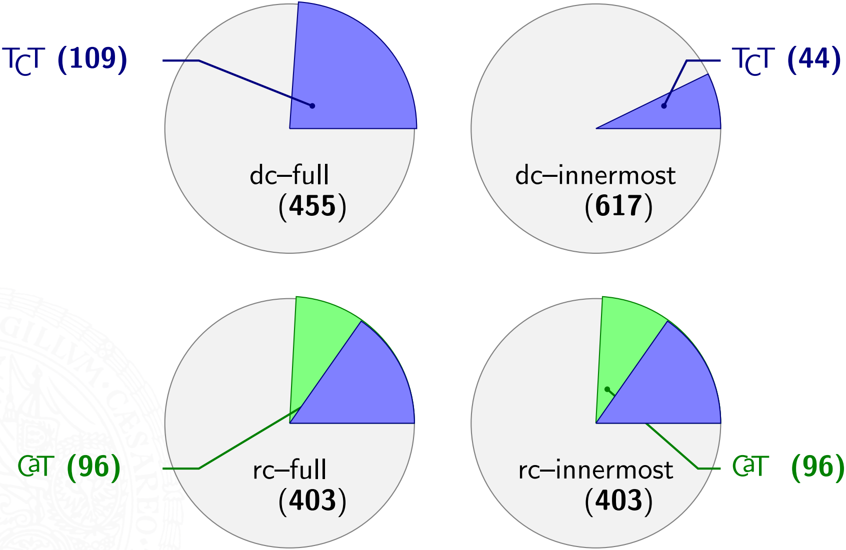
Automated Complexity Analysis: A Snapshot

polynomial runtime complexity on TPDB 7.0.2



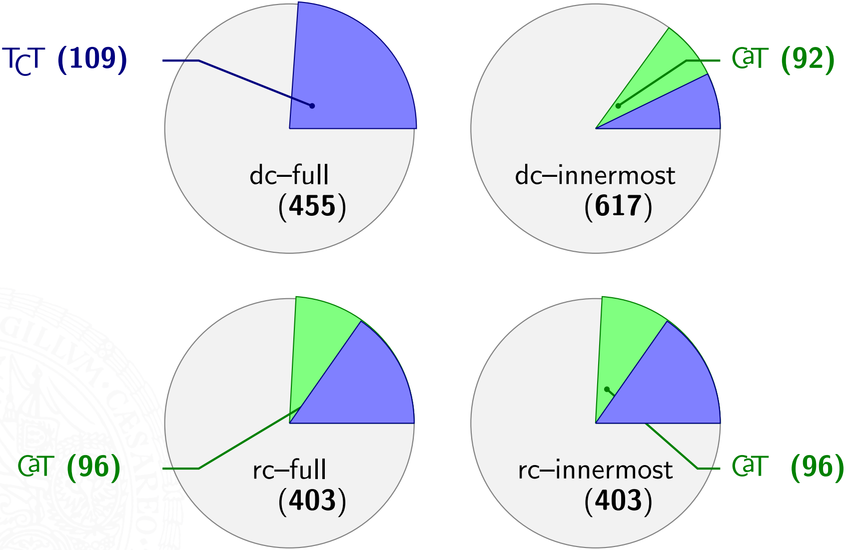
Automated Complexity Analysis: A Snapshot

polynomial runtime complexity on TPDB 7.0.2



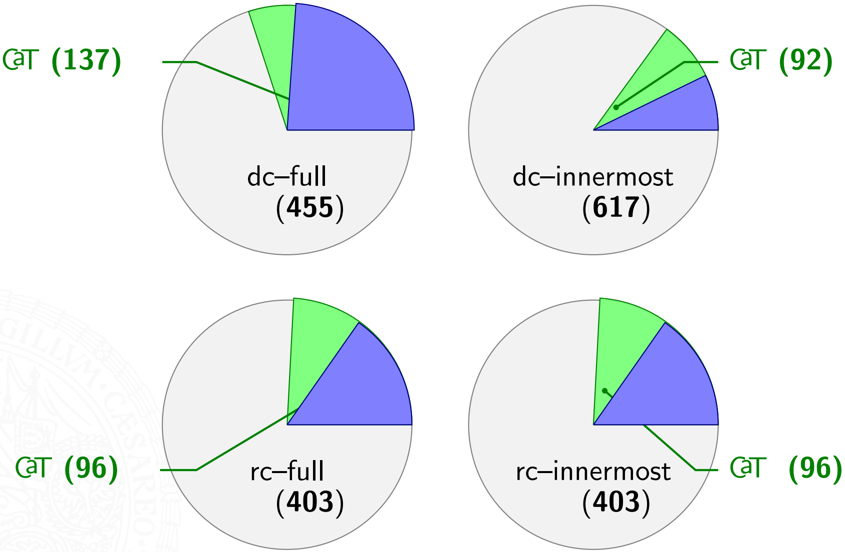
Automated Complexity Analysis: A Snapshot

polynomial runtime complexity on TPDB 7.0.2



Automated Complexity Analysis: A Snapshot

polynomial runtime complexity on TPDB 7.0.2



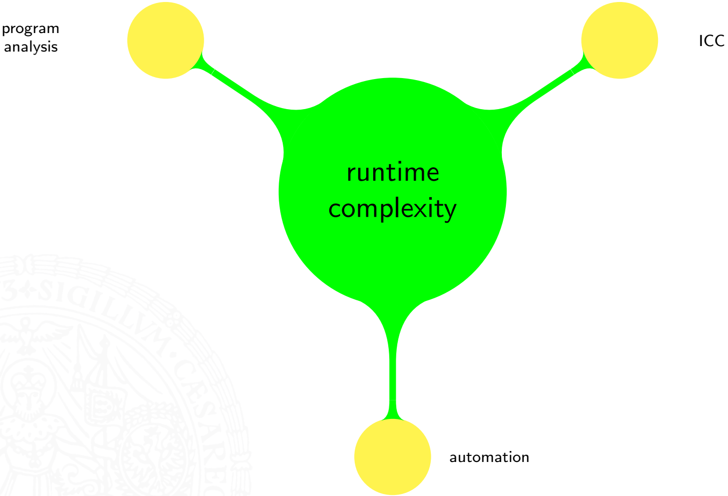
The Future

Question

where to go from here?

Question

where to go from here?



runtime complexity

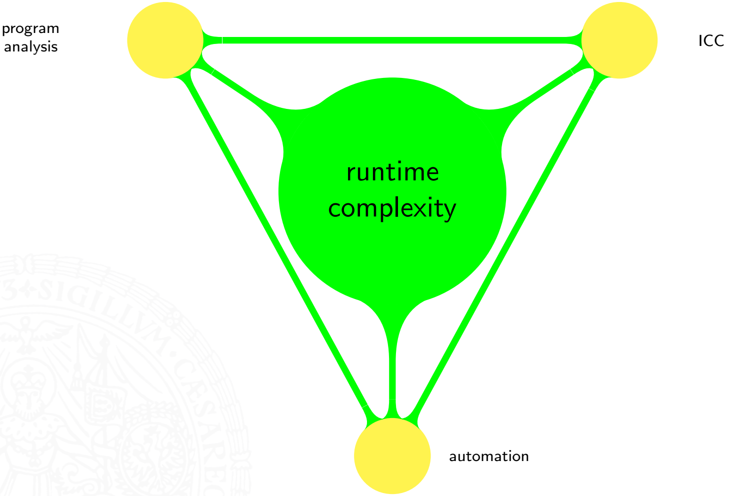
program analysis

ICC

automation

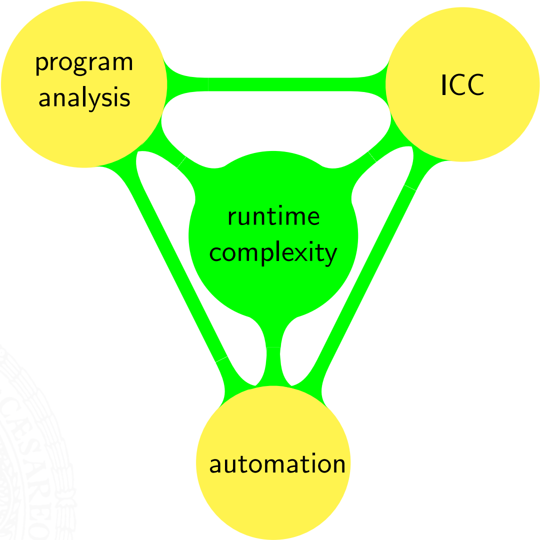
Question

where to go from here?



Question

where to go from here?



Question

where to go from here?