Present and Future of Proving Termination of Rewriting

Albert Rubio

Universitat Politècnica de Catalunya
Barcelona

Overview of the talk

Overview of the talk

- Introduction
  - Rewriting
  - Termination
  - Motivation

Overview of the talk

- Introduction

Overview of the talk

- Introduction
- A bit of history

Overview of the talk

- Introduction
- A bit of history
- Reduction Orderings
  - Polynomial Interpretations
  - Path Orderings

Overview of the talk

- Introduction
- A bit of history
- Reduction Orderings

Overview of the talk

- Introduction
- A bit of history
- Reduction Orderings
- Powerful automatable methods
  - Dependency Pairs
  - Monotonic Semantic Path Ordering

Overview of the talk

- Introduction
- A bit of history
- Reduction Orderings
- Powerful automatable methods

Overview of the talk

- Introduction
- A bit of history
- Reduction Orderings
- Powerful automatable methods
- XXI century motivation

Overview of the talk

- Introduction
- A bit of history
- Reduction Orderings
- Powerful automatable methods
- XXI century motivation
- Potential sources of improvement

Overview of the talk

- Introduction
- A bit of history
- Reduction Orderings
- Powerful automatable methods
- XXI century motivation
- Potential sources of improvement
- Conclusions

Rewriting

Computing the product of positive integers

Rewriting

Computing the product of positive integers

$\mathrm{prod}(2, 4) = 8$ $\qquad$ $\mathrm{prod}(s(s(0)), s(s(s(s(0))))) = s(s(s(s(s(s(s(s(0))))))))$

Rewriting

Computing the product of positive integers

prod(2, 4) = 8 $\qquad$ prod($s(s(0))$, $s(s(s(s(0))))$) = $s(s(s(s(s(s(s(s(0))))))))$

int prod (*int* n, *int* m) {
 if (n==0) *return*(0);
 else *return*(prod(n-1,m)+m);
 }

$$prod(0, m) \to 0$$
$$prod(s(n'), m) \to prod(n', m) + m$$

Rewriting

Computing the product of positive integers

prod(2, 4) = 8 prod($s(s(0)), s(s(s(s(0))))$) = $s(s(s(s(s(s(s(s(0))))))))$

```
int prod (int n, int m) {
 if (n==0) return(0);
 else      return(prod(n-1,m)+m);
 }
int prod ( int n,  int m) {
 int p= 0;
 while (n!=0) {
   p+=m;
   n- -;
 }
 return(p);
}
```

$$
\begin{array}{lcl}
prod(0, m) & \to & 0 \\
prod(s(n'), m) & \to & prod(n', m) + m
\end{array}
$$

$$
\begin{array}{lcl}
prod(n, m) & \to & prod1(n, m, 0) \\
prod1(0, m, p) & \to & p \\
prod1(s(n'), m, p) & \to & prod1(n', m, p + m)
\end{array}
$$

Rewriting (cont.)

A special rule: β-reduction

Rewriting (cont.)

A special rule: β-reduction

$$(\lambda x.u) \ v \ \rightarrow_{\beta} u\{x{:=}v\}$$

Rewriting (cont.)

A special rule: β-reduction

$$@(\lambda x.u, v) \longrightarrow_\beta u\{x := v\}$$

Rewriting (cont.)

A special rule: β-reduction

$$@(\lambda x.u, v) \longrightarrow_{\beta} u\{x{:=}v\}$$

$$@(\ @(\lambda x.\lambda y.\ x + y\ ,\ \ 3\ \ )\ \ ,\ 5)$$

Rewriting (cont.)

A special rule: β-reduction

$$@(\lambda x.u, v) \longrightarrow_\beta u\{x{:=}v\}$$

$$@(\overbrace{@(\underbrace{\lambda x.\lambda y.\ x + y}_{\lambda x.u}, \underbrace{3}_{v})}, 5) \longrightarrow_\beta$$

Rewriting (cont.)

A special rule: β-reduction

$$@(\lambda x.u, v) \rightarrow_{\beta} u\{x := v\}$$

$$@(\overbrace{@(\underbrace{\lambda x.\lambda y.\ x + y}_{\lambda x.u}, \underbrace{3}_{v})}, 5) \rightarrow_{\beta}$$

$$@(\lambda y.\ 3 + y\ ,\ 5\ )$$

Rewriting (cont.)

A special rule: β-reduction

$$@(\lambda x.u, v) \rightarrow_\beta u\{x{:=}v\}$$

$$@(\overbrace{@(\underbrace{\lambda x.\lambda y.\ x+y}_{\lambda x.u}, \underbrace{3}_{v})}, 5) \rightarrow_\beta$$

$$\overbrace{@(\underbrace{\lambda y.\ 3+y}_{\lambda x.u}, \underbrace{5}_{v})} \rightarrow_\beta$$

Rewriting (cont.)

A special rule: β-reduction

$$@(\lambda x.u, v) \longrightarrow_\beta u\{x{:=}v\}$$

$$@(\overbrace{@(\underbrace{\lambda x.\lambda y.\ x+y}_{\lambda x.u}, \underbrace{3}_{v})}, 5) \longrightarrow_\beta$$

$$\overbrace{@(\underbrace{\lambda y.\ 3+y}_{\lambda x.u}, \underbrace{5}_{v})} \longrightarrow_\beta$$

$$3+5$$

Rewriting (cont.)

β-reduction may not terminate.

Rewriting (cont.)

β-reduction may not terminate.

$$@(\lambda x.\, @(x,x)\ ,\ \lambda x.\, @(x,x))$$

Rewriting (cont.)

β-reduction may not terminate.

$$\overbrace{@(\underbrace{\lambda x.\,@(x,x)}_{\lambda x.u}\,,\,\underbrace{\lambda x.\,@(x,x)}_{v})} \longrightarrow_{\beta}$$

Rewriting (cont.)

β-reduction may not terminate.

$$\overbrace{@(\underbrace{\lambda x.\,@(x,x)}_{\lambda x.u}\,,\,\underbrace{\lambda x.\,@(x,x)}_{v})} \longrightarrow_{\beta}$$

$$@(\lambda x.\,@(x,x)\,,\,\lambda x.\,@(x,x))$$

Rewriting (cont.)

β-reduction may not terminate.

$$\overbrace{@(\underbrace{\lambda x.\, @(x,x)}_{\lambda x.u}\, ,\, \underbrace{\lambda x.\, @(x,x)}_{v})} \rightarrow_{\beta}$$

$$@(\lambda x.\, @(x,x)\, ,\, \lambda x.\, @(x,x)) \rightarrow_{\beta} \ldots$$

Rewriting (cont.)

β-reduction may not terminate.

$$\overbrace{@(\underbrace{\lambda x.\, @(x,x)}_{\lambda x.u}\,,\,\underbrace{\lambda x.\, @(x,x)}_{v})} \rightarrow_\beta$$

$$@(\lambda x.\, @(x,x)\,,\,\lambda x.\, @(x,x)) \rightarrow_\beta \ldots$$

To ensure termination: Typed λ-calculus

Rewriting (cont.)

β-reduction may not terminate.

$$\overbrace{@(\underbrace{\lambda x.\, @(x,x)}_{\lambda x.u}\ ,\ \underbrace{\lambda x.\, @(x,x)}_{v})} \rightarrow_{\beta}$$

$$@(\lambda x.\, @(x,x)\ ,\ \lambda x.\, @(x,x)) \rightarrow_{\beta} \ldots$$

To ensure termination: Typed λ-calculus

$$\{x : \mathbb{N},\ y : \mathbb{N}\} \vdash\ x : \mathbb{N} \qquad \{x : \mathbb{N},\ y : \mathbb{N}\} \vdash\ y : \mathbb{N}$$

Rewriting (cont.)

β-reduction may not terminate.

$$\overbrace{@(\underbrace{\lambda x.\,@(x,x)}_{\lambda x.u}\;,\;\underbrace{\lambda x.\,@(x,x)}_{v})} \rightarrow_\beta$$

$$@(\lambda x.\,@(x,x)\;,\;\lambda x.\,@(x,x)) \rightarrow_\beta \ldots$$

To ensure termination: Typed λ-calculus

$$\{x:\mathbb{N},\ y:\mathbb{N}\} \vdash\ x:\mathbb{N} \qquad \{x:\mathbb{N},\ y:\mathbb{N}\} \vdash\ y:\mathbb{N}$$

$$\{x:\mathbb{N},\ y:\mathbb{N}\} \vdash\ x+y:\mathbb{N}$$

Rewriting (cont.)

β-reduction may not terminate.

$$\overbrace{@(\underbrace{\lambda x.\ @(x,x)}_{\lambda x.u}\ ,\ \underbrace{\lambda x.\ @(x,x)}_{v})} \rightarrow_\beta$$

$$@(\lambda x.\ @(x,x)\ ,\ \lambda x.\ @(x,x)) \rightarrow_\beta \ldots$$

To ensure termination: Typed λ-calculus

$$\{x : \mathbb{N},\ y : \mathbb{N}\} \vdash\ x : \mathbb{N} \qquad \{x : \mathbb{N},\ y : \mathbb{N}\} \vdash\ y : \mathbb{N}$$

$$\{x : \mathbb{N},\ y : \mathbb{N}\} \vdash\ x + y : \mathbb{N}$$

$$\{\} \vdash\ \lambda x{:}\mathbb{N}.\ \lambda y{:}\mathbb{N}.\ x + y : \mathbb{N} \rightarrow \mathbb{N} \rightarrow \mathbb{N}$$

Rewriting (cont.)

β-reduction may not terminate.

$$\overbrace{@(\underbrace{\lambda x.\, @(x,x)}_{\lambda x.u}\,,\,\underbrace{\lambda x.\, @(x,x)}_{v})} \rightarrow_\beta$$

$$@(\lambda x.\, @(x,x)\,,\,\lambda x.\, @(x,x)) \rightarrow_\beta \ldots$$

To ensure termination: Typed λ-calculus

$$\{x:\mathbb{N},\ y:\mathbb{N}\} \vdash\ x:\mathbb{N} \qquad \{x:\mathbb{N},\ y:\mathbb{N}\} \vdash\ y:\mathbb{N}$$

$$\{x:\mathbb{N},\ y:\mathbb{N}\} \vdash\ x+y:\mathbb{N}$$

$$\{\} \vdash\ \lambda x\!:\!\mathbb{N}.\ \lambda y\!:\!\mathbb{N}.\ x+y : \mathbb{N} \rightarrow \mathbb{N} \rightarrow \mathbb{N}$$

$$\{\} \vdash\ @(\ @(\ \lambda x\!:\!\mathbb{N}.\ \lambda y\!:\!\mathbb{N}.\ x+y\ ,\ 3\ )\ ,\ 5\ ) : \mathbb{N}$$

Higher-Order rewriting

Combining rewriting and β-reduction.

Higher-Order rewriting

Combining rewriting and β-reduction.

With lambdas:

Higher-Order rewriting

Combining rewriting and β-reduction.

With lambdas:

- rewriting union β-reduction

Higher-Order rewriting

Combining rewriting and β-reduction.

With lambdas:

- rewriting union β-reduction
- rewriting over β-normalized terms

Termination

Absence of infinite computations

Termination

Absence of infinite computations

Example of non-termination of higher-order rewriting

Termination

Absence of infinite computations

Let $f : \alpha \times \alpha \Rightarrow \alpha$ and $g : (\alpha \to \alpha) \Rightarrow \alpha$ in

$$\{x : \alpha \to \alpha\} \vdash f(g(x), g(x)) : \alpha \longrightarrow @(x, g(x)) : \alpha$$

Termination

Absence of infinite computations

Let $f : \alpha \times \alpha \Rightarrow \alpha$ and $g : (\alpha \to \alpha) \Rightarrow \alpha$ in

$$\{x : \alpha \to \alpha\} \vdash f(g(x), g(x)) : \alpha \to @(x, g(x)) : \alpha$$

$$f(g(\lambda y.f(y,y)) \; , \; g(\lambda y.f(y,y))) : \alpha$$

Termination

Absence of infinite computations

Let $f : \alpha \times \alpha \Rightarrow \alpha$ and $g : (\alpha \to \alpha) \Rightarrow \alpha$ in

$$\{x : \alpha \to \alpha\} \vdash f(g(x), g(x)) : \alpha \longrightarrow @(x, g(x)) : \alpha$$

$$\overbrace{f(g(\underbrace{\lambda y.f(y,y)}_{x}))\ ,\ g(\underbrace{\lambda y.f(y,y)}_{x}))}\ : \alpha \ \longrightarrow$$

Termination

Absence of infinite computations

Let $f : \alpha \times \alpha \Rightarrow \alpha$ and $g : (\alpha \to \alpha) \Rightarrow \alpha$ in

$$\{x : \alpha \to \alpha\} \vdash f(g(x), g(x)) : \alpha \to @(x, g(x)) : \alpha$$

$$\overbrace{f(g(\underbrace{\lambda y.f(y,y)}_{x})\ ,\ g(\underbrace{\lambda y.f(y,y)}_{x}))} : \alpha \ \to$$

$$@(\lambda y.f(y,y)\ ,\ g(\lambda y.f(y,y))) : \alpha$$

Termination

Absence of infinite computations

Let $f : \alpha \times \alpha \Rightarrow \alpha$ and $g : (\alpha \to \alpha) \Rightarrow \alpha$ in

$$\{x : \alpha \to \alpha\} \vdash f(g(x), g(x)) : \alpha \to @(x, g(x)) : \alpha$$

$$\overbrace{f(g(\underbrace{\lambda y.f(y,y)}_{x}),\ g(\underbrace{\lambda y.f(y,y)}_{x}))} : \alpha \ \to$$

$$\overbrace{@(\underbrace{\lambda y.f(y,y)}_{\lambda y.u},\ \underbrace{g(\lambda y.f(y,y))}_{v})} : \alpha \to_{\beta}$$

Termination

Absence of infinite computations

Let $f : \alpha \times \alpha \Rightarrow \alpha$ and $g : (\alpha \to \alpha) \Rightarrow \alpha$ in

$$\{x : \alpha \to \alpha\} \vdash f(g(x), g(x)) : \alpha \to @(x, g(x)) : \alpha$$

$$\overbrace{f(g(\underbrace{\lambda y.f(y,y)}_{x}) \;,\; g(\underbrace{\lambda y.f(y,y)}_{x}))} : \alpha \;\to$$

$$\overbrace{@(\underbrace{\lambda y.f(y,y)}_{\lambda y.u} \;,\; \underbrace{g(\lambda y.f(y,y))}_{v})} : \alpha \to_{\beta}$$

$$f(g(\lambda y.f(y,y)) \;,\; g(\lambda y.f(y,y))) : \alpha \;\to \ldots$$

Motivation

Why studing termination (early days)?

Motivation

Why studing termination (early days)?

- Knuth-Bendix completion and theorem proving

Motivation

Why studing termination (early days)?

- Knuth-Bendix completion and theorem proving
- Termination of some particular TRS

Motivation

Why studing termination (early days)?

- Knuth-Bendix completion and theorem proving
- Termination of some particular TRS
  - rule-based algorithms

Motivation

Why studing termination (early days)?

- Knuth-Bendix completion and theorem proving
- Termination of some particular TRS
  - rule-based algorithms
  - rule-based specification

A bit of history

The beginings

- 1970, Manna and Ness. Reference for *reduction orderings*. One of the first papers on termination of rewriting.

A bit of history

The beginings

- 1970, Manna and Ness. Reference for *reduction orderings*. One of the first papers on termination of rewriting.
- 1970, Knuth and Bendix. Knuth Bendix Ordering (KBO).

A bit of history

The beginings

- 1970, Manna and Ness. Reference for *reduction orderings*. One of the first papers on termination of rewriting.
- 1970, Knuth and Bendix. Knuth Bendix Ordering (KBO).
- 1978, Huet and Lankford. Undecidability of the termination of Term Rewriting Systems.

A bit of history

The beginings

- 1970, Manna and Ness. Reference for *reduction orderings*. One of the first papers on termination of rewriting.
- 1970, Knuth and Bendix. Knuth Bendix Ordering (KBO).
- 1978, Huet and Lankford. Undecidability of the termination of Term Rewriting Systems.
- 1979, Dershowitz. Recursive Path Ordering (RPO).

A bit of history

The beginings

- 1970, Manna and Ness. Reference for *reduction orderings*. One of the first papers on termination of rewriting.
- 1970, Knuth and Bendix. Knuth Bendix Ordering (KBO).
- 1978, Huet and Lankford. Undecidability of the termination of Term Rewriting Systems.
- 1979, Dershowitz. Recursive Path Ordering (RPO).
- 1979, Dershowitz Manna. Multiset ordering.

A bit of history

The beginings

- 1970, Manna and Ness. Reference for *reduction orderings*. One of the first papers on termination of rewriting.
- 1970, Knuth and Bendix. Knuth Bendix Ordering (KBO).
- 1978, Huet and Lankford. Undecidability of the termination of Term Rewriting Systems.
- 1979, Dershowitz. Recursive Path Ordering (RPO).
- 1979, Dershowitz Manna. Multiset ordering.
- 1979, Lankford. One of the first references on polynomial interpretations.

A bit of history

The beginings

- 1970, Manna and Ness. Reference for *reduction orderings*. One of the first papers on termination of rewriting.
- 1970, Knuth and Bendix. Knuth Bendix Ordering (KBO).
- 1978, Huet and Lankford. Undecidability of the termination of Term Rewriting Systems.
- 1979, Dershowitz. Recursive Path Ordering (RPO).
- 1979, Dershowitz Manna. Multiset ordering.
- 1979, Lankford. One of the first references on polynomial interpretations.
- 1980, Kamin and Levy. Lexicographic Path Ordering (LPO) and Semantic Path Ordering (SPO).

Reduction Orderings

$$s(s(\overbrace{s(\underbrace{s(0)}_{n}) + \underbrace{s(s(s(s(0))))}_{m}}))\ \rightarrow\ s(s(\overbrace{s(\underbrace{s(0)}_{n} + \underbrace{s(s(s(s(0))))}_{m})}))$$

$$s(n) + m \rightarrow s(n + m)$$

Reduction Orderings

$$s(s(\overbrace{s(\underbrace{s(0)}_{n}) + \underbrace{s(s(s(s(0))))}_{m}})) \rightarrow s(s(\overbrace{s(\underbrace{s(0)}_{n} + \underbrace{s(s(s(s(0))))}_{m})}))$$

$$s(n) + m \rightarrow s(n + m)$$

$$s(n) + m \succ s(n + m)$$

Reduction Orderings

$$s(s(\overbrace{s(\underbrace{s(0)}_{n}) + \underbrace{s(s(s(s(0))))}_{m}})) \rightarrow s(s(\overbrace{s(\underbrace{s(0)}_{n} + \underbrace{s(s(s(s(0))))}_{m})}))$$

$$s(n) + m \rightarrow s(n + m)$$

$$s(n) + m \succ s(n + m)$$

$$s(\underbrace{s(0)}_{n}) + \underbrace{s(s(s(s(0))))}_{m} \succ s(\underbrace{s(0)}_{n} + \underbrace{s(s(s(s(0))))}_{m})$$

stable under substitution

Reduction Orderings

$$s(s(\overbrace{s(\underbrace{s(0)}_{n}) + \underbrace{s(s(s(s(0))))}_{m}}))\ \rightarrow\ s(s(\overbrace{s(\underbrace{s(0)}_{n} + \underbrace{s(s(s(s(0))))}_{m})}))$$

$$s(n) + m \rightarrow s(n + m)$$

$$s(n) + m \succ s(n + m)$$

$$s(\underbrace{s(0)}_{n}) + \underbrace{s(s(s(s(0))))}_{m} \succ s(\underbrace{s(0)}_{n} + \underbrace{s(s(s(s(0))))}_{m})$$ stable under substitution

$$s(s(\overbrace{s(\underbrace{s(0)}_{n}) + \underbrace{s(s(s(s(0))))}_{m}})) \succ s(s(\overbrace{s(\underbrace{s(0)}_{n} + \underbrace{s(s(s(s(0))))}_{m})}))$$ monotonic

Reduction Orderings (cont.)

$\succ$ is a *reduction ordering* if it's

- *well-founded*

$$\nexists t_1 \succ t_2 \succ t_3 \succ \ldots$$

- *stable under sustitution*

$$s \succ t \quad \text{implies} \quad s\sigma \succ t\sigma \quad \text{for every substitution } \sigma$$

- *monotonic*

$$s \succ t \quad \text{implies} \quad u[s] \succ u[t] \quad \text{for every context } u[\ ]$$

Reduction Orderings (cont.)

$\succ$ is a *reduction ordering* if it's

- *well-founded*

$$\nexists t_1 \succ t_2 \succ t_3 \succ \dots$$

- *stable under sustitution*

$$s \succ t \quad \text{implies} \quad s\sigma \succ t\sigma \quad \text{for every substitution } \sigma$$

- *monotonic*

$$s \succ t \quad \text{implies} \quad u[s] \succ u[t] \quad \text{for every context } u[\ ]$$

$$R \text{ terminates} \quad \text{if and only if} \quad l \succ r \text{ for all } l \to r \in R$$

for some reduction ordering $\succ$.

Reduction Orderings (cont.)

$\succ$ is a *higher-order* reduction ordering if it's

- *well-founded*

$$\not\exists t_1 \succ t_2 \succ t_3 \succ \ldots$$

- *stable under substitution*

$$s \succ t \quad \text{implies} \quad s\sigma \succ t\sigma \quad \text{for all substitution } \sigma$$

- *monotonic*

$$s \succ t \quad \text{implies} \quad u[s] \succ u[t] \quad \text{for all context } u[\,]$$

Reduction Orderings (cont.)

$\succ$ is a *higher-order* reduction ordering if it's

- *well-founded*

$$\nexists t_1 \succ t_2 \succ t_3 \succ \ldots$$

- *stable under substitution*

$$s \succ t \quad \text{implies} \quad s\sigma \succ t\sigma \quad \text{for all substitution } \sigma$$

- *monotonic*

$$s \succ t \quad \text{implies} \quad u[s] \succ u[t] \quad \text{for all context } u[\,]$$

- *functional*

$$s \longrightarrow_{\beta} t \quad \text{implies} \quad s \succ t$$

Reduction Orderings (cont.)

$\succ$ is a *higher-order* reduction ordering if it's

- *well-founded*

$$\nexists t_1 \succ t_2 \succ t_3 \succ \ldots$$

- *stable under substitution*

$$s \succ t \quad \text{implies} \quad s\sigma \succ t\sigma \quad \text{for all substitution } \sigma$$

- *monotonic*

$$s \succ t \quad \text{implies} \quad u[s] \succ u[t] \quad \text{for all context } u[\,]$$

- *functional*

$$s \longrightarrow_{\beta} t \quad \text{implies} \quad s \succ t$$

$$R \text{ terminates} \quad \text{if and only if} \quad l \succ r \text{ for all } l \rightarrow r \in R$$

for some higher-order reduction ordering $\succ$.

Polynomial Interpretations

Polynomial Interpretations

Interprets every function symbol as a (linear) polynomial:
$[f] = A^k \longrightarrow A$
where A is often the positive integers or the reals.

Polynomial Interpretations

Interprets every function symbol as a (linear) polynomial:
$[f] = A^k \longrightarrow A$
where A is often the positive integers or the reals.

Interpret terms by:
$Pol(f(t_1, \ldots, t_n)) = [f](Pol(t_1), \ldots, Pol(t_n))$

Polynomial Interpretations

Interprets every function symbol as a (linear) polynomial:
$[f] = A^k \longrightarrow A$
where A is often the positive integers or the reals.

Interpret terms by:
$Pol(f(t_1, \ldots, t_n)) = [f](Pol(t_1), \ldots, Pol(t_n))$

Then a reduction ordering is obtained by
$s \succ t$ if $Pol(s) - Pol(t) > 0$
if all coeficients are strictly positive.

If there are zero coeficients then it is just weakly monotonic.

The*recursive path ordering*

The *recursive path ordering*

compares terms by extending an ordering on function symbols.

Therecursive path ordering

compares terms by extending an ordering on function symbols.

Let $\succ_{\mathcal{F}}$ be a (well-founded) ordering on the function symbols
For instance: $prod \succ_{\mathcal{F}} + \succ_{\mathcal{F}} s \succ_{\mathcal{F}} 0$

The recursive path ordering

compares terms by extending an ordering on function symbols.

Let $\succ_{\mathcal{F}}$ be a (well-founded) ordering on the function symbols
For instance: $prod \succ_{\mathcal{F}} + \succ_{\mathcal{F}} s \succ_{\mathcal{F}} 0$

$$s = f(s_1, \ldots, s_n) \succ_{rpo} t \quad \text{if and only if}$$

1. $s_i \succeq_{rpo} t$ for some $1 \leq i \leq n$.
2. $t = g(t_1, \ldots, t_m)$, $f \succ_{\mathcal{F}} g$ and $s \succ_{rpo} t_j$ for all $1 \leq j \leq m$.
3. $t = f(t_1, \ldots, t_m)$ and $\{s_1, \ldots, s_n\} \succ\!\succ_{rpo} \{t_1, \ldots, t_m\}$

The *recursive path ordering* (cont.)

$$s = f(s_1, \ldots, s_n) \succ_{rpo} t \quad \text{if and only if}$$

1. $s_i \succeq_{rpo} t$ for some $1 \leq i \leq n$.
2. $t = g(t_1, \ldots, t_m)$, $f \succ_{\mathcal{F}} g$ and $s \succ_{rpo} t_j$ for all $1 \leq j \leq m$.
3. $t = f(t_1, \ldots, t_m)$ and $\{s_1, \ldots, s_n\} \succ\!\succ_{rpo} \{t_1, \ldots, t_m\}$

Let $prod \succ_{\mathcal{F}} + \succ_{\mathcal{F}} s \succ_{\mathcal{F}} 0$

The *recursive path ordering* (cont.)

$$s = f(s_1, \ldots, s_n) \succ_{rpo} t \quad \text{if and only if}$$

1. $s_i \succeq_{rpo} t$ for some $1 \leq i \leq n$.
2. $t = g(t_1, \ldots, t_m)$, $f \succ_{\mathcal{F}} g$ and $s \succ_{rpo} t_j$ for all $1 \leq j \leq m$.
3. $t = f(t_1, \ldots, t_m)$ and $\{s_1, \ldots, s_n\} \succ\succ_{rpo} \{t_1, \ldots, t_m\}$

Let $prod \succ_{\mathcal{F}} + \succ_{\mathcal{F}} s \succ_{\mathcal{F}} 0$

$$prod(s(n), m) \succ_{rpo} s(prod(n, m))$$

The *recursive path ordering* (cont.)

$$s = f(s_1, \ldots, s_n) \succ_{rpo} t \quad \text{if and only if}$$

1. $s_i \succeq_{rpo} t$ for some $1 \leq i \leq n$.

2. $t = g(t_1, \ldots, t_m)$, $f \succ_{\mathcal{F}} g$ and $s \succ_{rpo} t_j$ for all $1 \leq j \leq m$.
3. $t = f(t_1, \ldots, t_m)$ and $\{s_1, \ldots, s_n\} \succ\succ_{rpo} \{t_1, \ldots, t_m\}$

Let $prod \succ_{\mathcal{F}} + \succ_{\mathcal{F}} s \succ_{\mathcal{F}} 0$

$prod(s(n), m) \succ_{rpo} s(prod(n, m))$ Case 2

$prod(s(n), m) \succ_{rpo} prod(n, m)$

The *recursive path ordering* (cont.)

$$s = f(s_1, \ldots, s_n) \succ_{rpo} t \quad \text{if and only if}$$

1. $s_i \succeq_{rpo} t$ for some $1 \leq i \leq n$.
2. $t = g(t_1, \ldots, t_m)$, $f \succ_{\mathcal{F}} g$ and $s \succ_{rpo} t_j$ for all $1 \leq j \leq m$.

3. $t = f(t_1, \ldots, t_m)$ and $\{s_1, \ldots, s_n\} \succ\succ_{rpo} \{t_1, \ldots, t_m\}$

Let $prod \succ_{\mathcal{F}} + \succ_{\mathcal{F}} s \succ_{\mathcal{F}} 0$

$prod(s(n), m) \succ_{rpo} s(prod(n, m))$ Case 2

$prod(s(n), m) \succ_{rpo} prod(n, m)$ Case 3

$\{s(n), m\} \succ\succ_{rpo} \{n, m\}$

The *recursive path ordering* (cont.)

$$s = f(s_1, \ldots, s_n) \succ_{rpo} t \quad \text{if and only if}$$

1. $s_i \succeq_{rpo} t$ for some $1 \leq i \leq n$.
2. $t = g(t_1, \ldots, t_m)$, $f \succ_{\mathcal{F}} g$ and $s \succ_{rpo} t_j$ for all $1 \leq j \leq m$.
3. $t = f(t_1, \ldots, t_m)$ and $\{s_1, \ldots, s_n\} \succ\succ_{rpo} \{t_1, \ldots, t_m\}$

Let $prod \succ_{\mathcal{F}} + \succ_{\mathcal{F}} s \succ_{\mathcal{F}} 0$

$prod(s(n), m) \succ_{rpo} s(prod(n, m))$ Case 2

$prod(s(n), m) \succ_{rpo} prod(n, m)$ Case 3

$\{s(n), \cancel{m}\} \succ\succ_{rpo} \{n, \cancel{m}\}$ multiset comparison

The *recursive path ordering* (cont.)

$$s = f(s_1, \ldots, s_n) \succ_{rpo} t \quad \text{if and only if}$$

1. $s_i \succeq_{rpo} t$ for some $1 \leq i \leq n$.
2. $t = g(t_1, \ldots, t_m)$, $f \succ_{\mathcal{F}} g$ and $s \succ_{rpo} t_j$ for all $1 \leq j \leq m$.
3. $t = f(t_1, \ldots, t_m)$ and $\{s_1, \ldots, s_n\} \succ\succ_{rpo} \{t_1, \ldots, t_m\}$

Let $prod \succ_{\mathcal{F}} + \succ_{\mathcal{F}} s \succ_{\mathcal{F}} 0$

$prod(s(n), m) \succ_{rpo} s(prod(n, m))$ Case 2

$prod(s(n), m) \succ_{rpo} prod(n, m)$ Case 3

$\{s(n), \cancel{m}\} \succ\succ_{rpo} \{n, \cancel{m}\}$ multiset comparison

$s(n) \succ_{rpo} n$

The *recursive path ordering* (cont.)

$$s = f(s_1, \ldots, s_n) \succ_{rpo} t \quad \text{if and only if}$$

1. $s_i \succeq_{rpo} t$ for some $1 \leq i \leq n$.
2. $t = g(t_1, \ldots, t_m)$, $f \succ_{\mathcal{F}} g$ and $s \succ_{rpo} t_j$ for all $1 \leq j \leq m$.
3. $t = f(t_1, \ldots, t_m)$ and $\{s_1, \ldots, s_n\} \succ\succ_{rpo} \{t_1, \ldots, t_m\}$

Let $prod \succ_{\mathcal{F}} + \succ_{\mathcal{F}} s \succ_{\mathcal{F}} 0$

$prod(s(n), m) \succ_{rpo} s(prod(n, m))$ Case 2

$prod(s(n), m) \succ_{rpo} prod(n, m)$ Case 3

$\{s(n), \cancel{m}\} \succ\succ_{rpo} \{n, \cancel{m}\}$ multiset comparison

$s(n) \succ_{rpo} n$ Case 1

Powerful automatable methods

Methods that can be fully automated

Can handle hard examples.

Powerful automatable methods

Methods that can be fully automated

Can handle hard examples.

Only two will be considered in this talk:

- The Dependency Pair Method
- The Monotonic Semantic Path Ordering

The Dependency Pair Method (cont.)

(Arts and Giesl 1997)

Basically, it transforms the rewrite system into a new reduction system which is simpler to analyze and (dis)prove termination.

The Dependency Pair Method (cont.)

(Arts and Giesl 1997)

Basically, it transforms the rewrite system into a new reduction system which is simpler to analyze and (dis)prove termination.

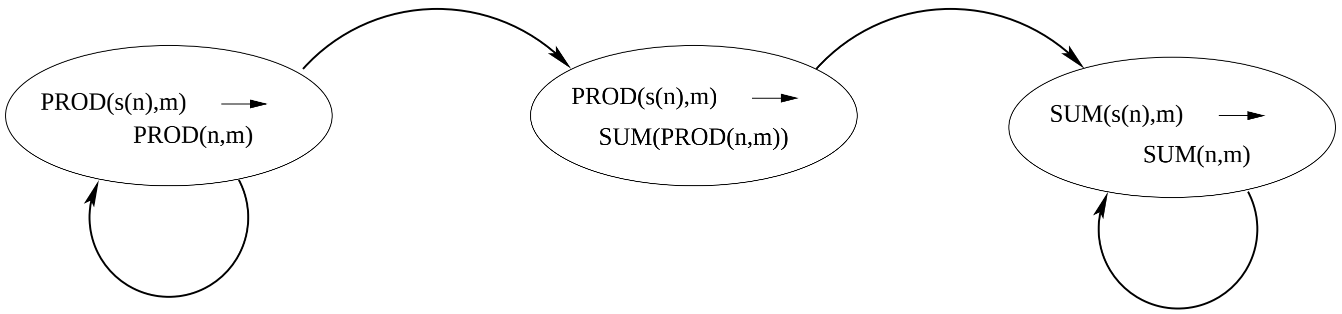
$$\begin{array}{lcllcl}
prod(0,m) & \rightarrow & 0 & & & \\
prod(s(n),m) & \rightarrow & prod(n,m)+m & PROD(s(n),m) & \Rightarrow & SUM(prod(n,m),m) \\
 & & & PROD(s(n),m) & \Rightarrow & PROD(n,m) \\
0+m & \rightarrow & m & & & \\
s(n)+m & \rightarrow & s(n+m) & SUM(s(n),m) & \Rightarrow & SUM(n,m)
\end{array}$$

The Dependency Pair Method (cont.)

(Arts and Giesl 1997)

Basically, it transforms the rewrite system into a new reduction system which is simpler to analyze and (dis)prove termination.

$$
\begin{array}{lcllcl}
prod(0,m) & \to & 0 & & & \\
prod(s(n),m) & \to & prod(n,m)+m & PROD(s(n),m) & \Rightarrow & SUM(prod(n,m),m) \\
 & & & PROD(s(n),m) & \Rightarrow & PROD(n,m) \\
0+m & \to & m & & & \\
s(n)+m & \to & s(n+m) & SUM(s(n),m) & \Rightarrow & SUM(n,m)
\end{array}
$$



The Dependency Pair Method (cont.)

(Arts and Giesl 1997)

Basically, it transforms the rewrite system into a new reduction system which is simpler to analyze and (dis)prove termination.

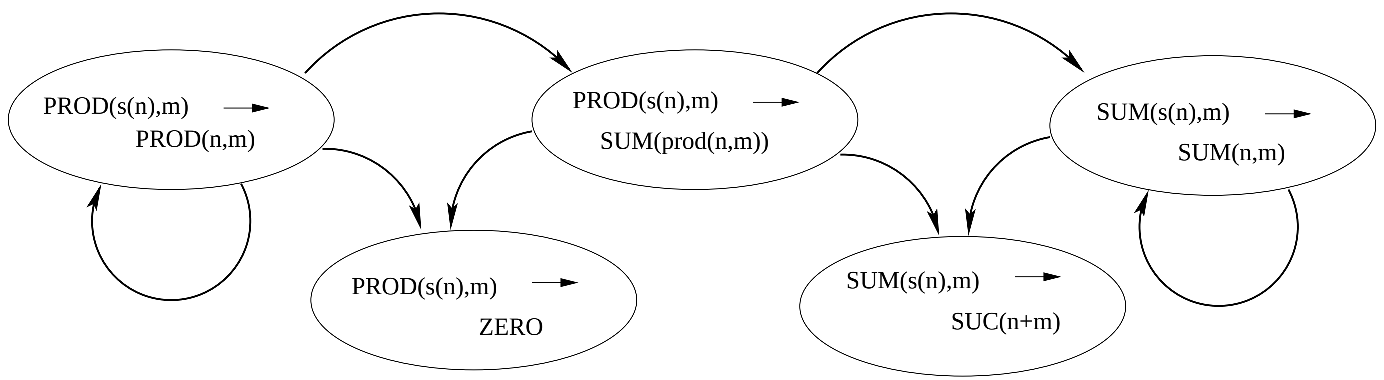
$$
\begin{array}{lcllcl}
prod(0,m) & \rightarrow & 0 & prod(0,m) & \Rightarrow & ZERO \\
prod(s(n),m) & \rightarrow & prod(n,m)+m & PROD(s(n),m) & \Rightarrow & SUM(prod(n,m),m) \\
 & & & PROD(s(n),m) & \Rightarrow & PROD(n,m) \\
0+m & \rightarrow & m & & & \\
s(n)+m & \rightarrow & s(n+m) & SUM(s(n),m) & \Rightarrow & SUM(n,m) \\
 & & & SUM(s(n),m) & \Rightarrow & SUC(n+m)
\end{array}
$$

The Dependency Pair Method (cont.)

(Arts and Giesl 1997)

Basically, it transforms the rewrite system into a new reduction system which is simpler to analyze and (dis)prove termination.

$$
\begin{array}{lcl@{\qquad}lcl}
prod(0,m) & \rightarrow & 0 & prod(0,m) & \Rightarrow & ZERO \\
prod(s(n),m) & \rightarrow & prod(n,m)+m & PROD(s(n),m) & \Rightarrow & SUM(prod(n,m),m) \\
 & & & PROD(s(n),m) & \Rightarrow & PROD(n,m) \\
0+m & \rightarrow & m & & & \\
s(n)+m & \rightarrow & s(n+m) & SUM(s(n),m) & \Rightarrow & SUM(n,m) \\
 & & & SUM(s(n),m) & \Rightarrow & SUC(n+m)
\end{array}
$$



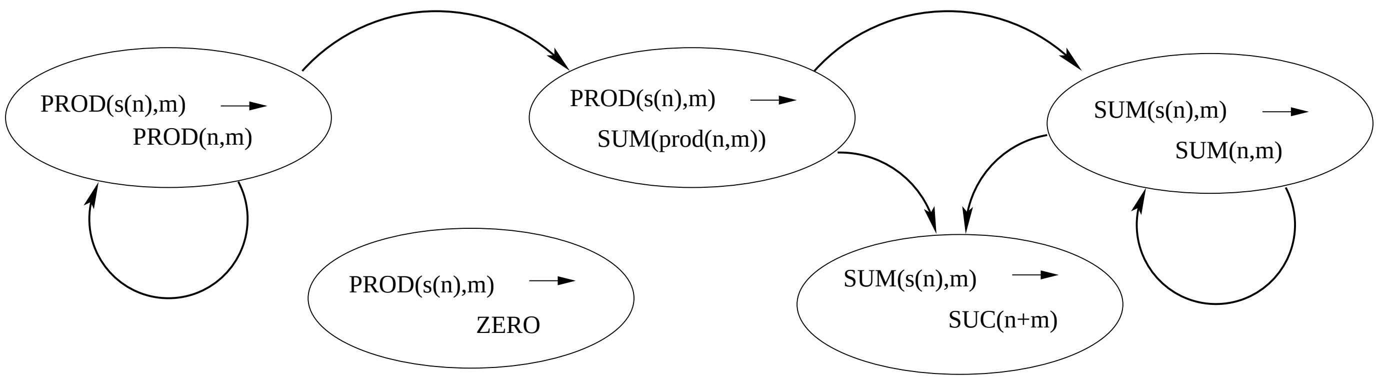
Does not change the potential cycles!!!

The Dependency Pair Method (cont.)

(Arts and Giesl 1997)

Basically, it transforms the rewrite system into a new reduction system which is simpler to analyze and (dis)prove termination.

$$
\begin{array}{lcl@{\qquad}lcl}
prod(0,m) & \rightarrow & 0 & prod(0,m) & \Rightarrow & ZERO \\
prod(s(n),m) & \rightarrow & prod(n,m)+m & PROD(s(n),m) & \Rightarrow & SUM(prod(n,m),m) \\
 & & & PROD(s(n),m) & \Rightarrow & PROD(n,m) \\
0+m & \rightarrow & m & & & \\
s(n)+m & \rightarrow & s(n+m) & SUM(s(n),m) & \Rightarrow & SUM(n,m) \\
 & & & SUM(s(n),m) & \Rightarrow & SUC(n+m)
\end{array}
$$

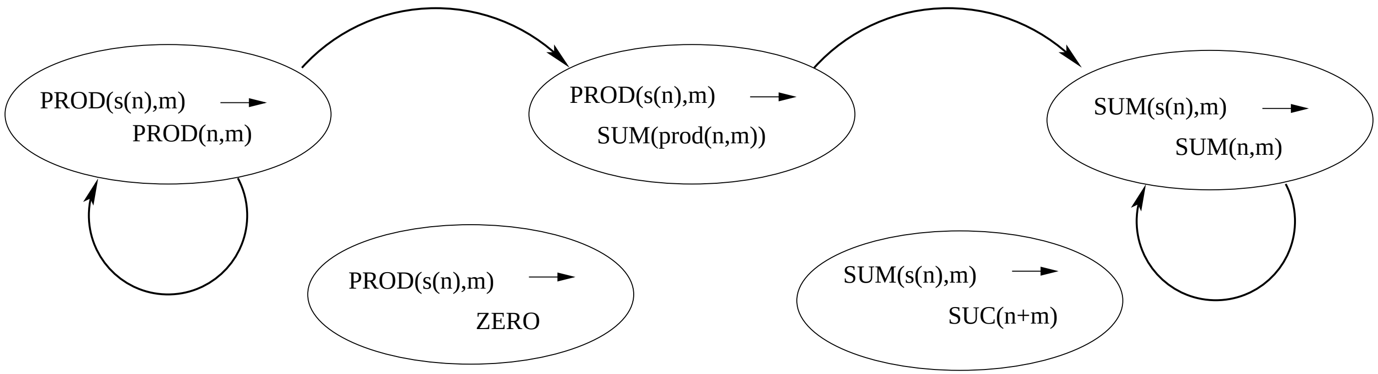


The Dependency Pair Method (cont.)

(Arts and Giesl 1997)

Basically, it transforms the rewrite system into a new reduction system which is simpler to analyze and (dis)prove termination.

$$
\begin{array}{lcllcl}
prod(0,m) & \to & 0 & prod(0,m) & \Rightarrow & ZERO \\
prod(s(n),m) & \to & prod(n,m)+m & PROD(s(n),m) & \Rightarrow & SUM(prod(n,m),m) \\
 & & & PROD(s(n),m) & \Rightarrow & PROD(n,m) \\
0+m & \to & m & & & \\
s(n)+m & \to & s(n+m) & SUM(s(n),m) & \Rightarrow & SUM(n,m) \\
 & & & SUM(s(n),m) & \Rightarrow & SUC(n+m)
\end{array}
$$

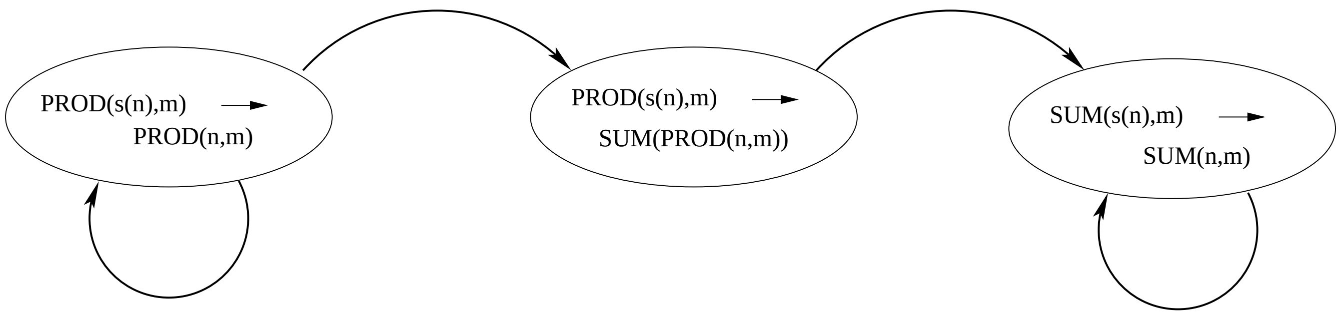


The Dependency Pair Method (cont.)

(Arts and Giesl 1997)

Basically, it transforms the rewrite system into a new reduction system which is simpler to analyze and (dis)prove termination.

$$\begin{array}{lcl@{\qquad}lcl}
prod(0,m) & \rightarrow & 0 & prod(0,m) & \Rightarrow & ZERO \\
prod(s(n),m) & \rightarrow & prod(n,m)+m & PROD(s(n),m) & \Rightarrow & SUM(prod(n,m),m) \\
 & & & PROD(s(n),m) & \Rightarrow & PROD(n,m) \\
0+m & \rightarrow & m & & & \\
s(n)+m & \rightarrow & s(n+m) & SUM(s(n),m) & \Rightarrow & SUM(n,m) \\
 & & & SUM(s(n),m) & \Rightarrow & SUC(n+m)
\end{array}$$

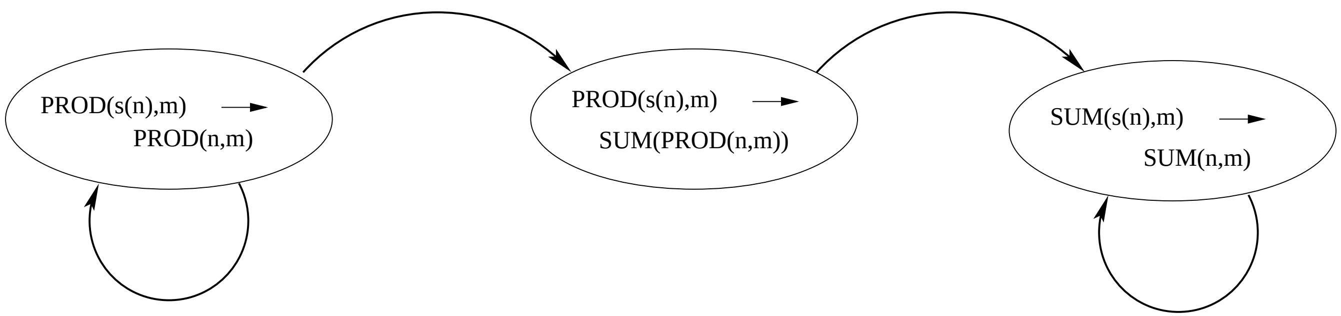


The Dependency Pair Method (cont.)

(Arts and Giesl 1997)

Basically, it transforms the rewrite system into a new reduction system which is simpler to analyze and (dis)prove termination.

$$
\begin{array}{lcllcl}
prod(0,m) & \rightarrow & 0 & & & \\
prod(s(n),m) & \rightarrow & prod(n,m)+m & PROD(s(n),m) & \Rightarrow & SUM(prod(n,m),m) \\
 & & & PROD(s(n),m) & \Rightarrow & PROD(n,m) \\
0+m & \rightarrow & m & & & \\
s(n)+m & \rightarrow & s(n+m) & SUM(s(n),m) & \Rightarrow & SUM(n,m)
\end{array}
$$

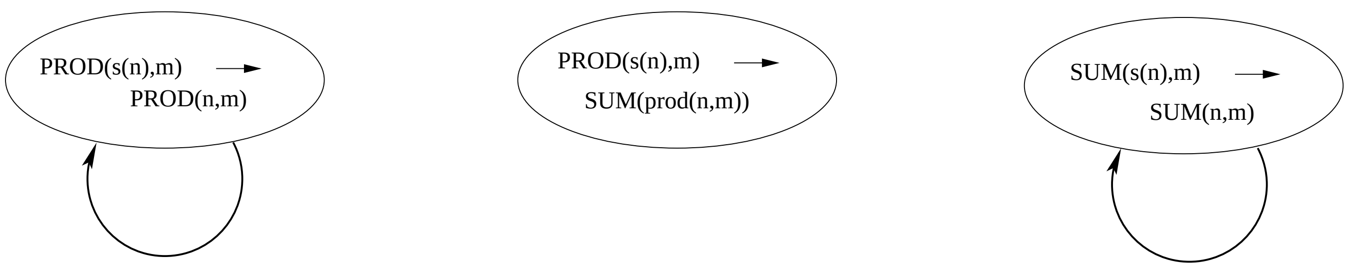


The Dependency Pair Method (cont.)

(Arts and Giesl 1997)

Basically, it transforms the rewrite system into a new reduction system which is simpler to analyze and (dis)prove termination.

$$
\begin{array}{lcllcl}
prod(0,m) & \rightarrow & 0 & & & \\
prod(s(n),m) & \rightarrow & prod(n,m)+m & PROD(s(n),m) & \Rightarrow & SUM(prod(n,m),m) \\
 & & & PROD(s(n),m) & \Rightarrow & PROD(n,m) \\
0+m & \rightarrow & m & & & \\
s(n)+m & \rightarrow & s(n+m) & SUM(s(n),m) & \Rightarrow & SUM(n,m)
\end{array}
$$

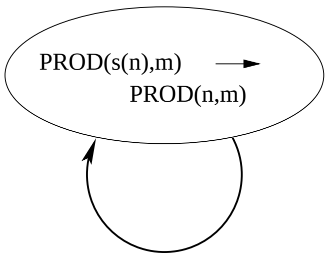


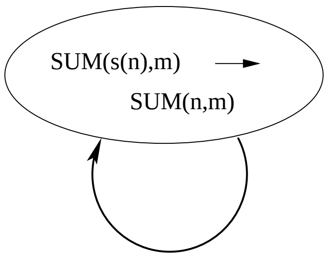
The Dependency Pair Method (cont.)

(Arts and Giesl 1997)

Basically, it transforms the rewrite system into a new reduction system which is simpler to analyze and (dis)prove termination.

$$
\begin{array}{lcllcl}
prod(0,m) & \rightarrow & 0 & & & \\
prod(s(n),m) & \rightarrow & prod(n,m)+m & PROD(s(n),m) & \Rightarrow & SUM(prod(n,m),m) \\
 & & & PROD(s(n),m) & \Rightarrow & PROD(n,m) \\
0+m & \rightarrow & m & & & \\
s(n)+m & \rightarrow & s(n+m) & SUM(s(n),m) & \Rightarrow & SUM(n,m)
\end{array}
$$





The Dependency Pair Method (cont.)

(Arts and Giesl 1997)

Basically, it transforms the rewrite system into a new reduction system which is simpler to analyze and (dis)prove termination.

$$
\begin{array}{lcllcl}
prod(0,m) & \rightarrow & 0 & & & \\
prod(s(n),m) & \rightarrow & prod(n,m)+m & PROD(s(n),m) & \Rightarrow & SUM(prod(n,m),m) \\
 & & & PROD(s(n),m) & \Rightarrow & PROD(n,m) \\
0+m & \rightarrow & m & & & \\
s(n)+m & \rightarrow & s(n+m) & SUM(s(n),m) & \Rightarrow & SUM(n,m)
\end{array}
$$

The first phase its a simple rewriting trace analysis.

This method was a breakthrough in the development of automated termination provers for term rewriting

The Dependency Pair Method (cont.)

After simplifying the graph we have to show that all pontential cycles are finite.

Solve the ordering constraints ensuring that all cycles in the graph are strictly decreasing.

Find an ordering satisfying the constraint:

- Polynomial Interpretations.
- RPO-like orderings with argument filterings.
- ...

The (Monotonic) Semantic Path Ordering

(Borralleras, Ferreira and Rubio 2000)

The (Monotonic) Semantic Path Ordering

(Borralleras, Ferreira and Rubio 2000)

$s = f(s_1, \ldots, s_n) \succ_{rpo} t$ if and only if

1. $s_i \succeq_{rpo} t$ for some $1 \leq i \leq n$.
2. $t = g(t_1, \ldots, t_m)$, $f \succ_{\mathcal{F}} g$ and $s \succ_{rpo} t_j$ for all $1 \leq j \leq m$.
3. $t = f(t_1, \ldots, t_m)$ and $\{s_1, \ldots, s_n\} \succ\succ_{rpo} \{t_1, \ldots, t_m\}$

The (Monotonic) Semantic Path Ordering

(Borralleras, Ferreira and Rubio 2000)

$$s = f(s_1, \ldots, s_n) \succ_{spo} t \quad \text{if and only if}$$

1. $s_i \succeq_{spo} t$ for some $1 \leq i \leq n$.
2. $t = g(t_1, \ldots, t_m)$, $s \sqsupset t$ and $s \succ_{spo} t_j$ for all $1 \leq j \leq m$.
3. $t = g(t_1, \ldots, t_m)$, $s \sqsupseteq t$ and $\{s_1, \ldots, s_n\} \succ\succ_{spo} \{t_1, \ldots, t_m\}$

in addition you need $s \sqsupseteq t$ to ensure monotonicity,
where $\sqsupseteq$ is monotonic.

The (Monotonic) Semantic Path Ordering

(Borralleras, Ferreira and Rubio 2000)

$$s = f(s_1, \ldots, s_n) \succ_{spo} t \quad \text{if and only if}$$

1. $s_i \succeq_{spo} t$ for some $1 \leq i \leq n$.
2. $s \sqsupset t$ and $s \succ_{spo} t_j$ for all $1 \leq j \leq m$.

3. $s \sqsupseteq t$ and $\{s_1, \ldots, s_n\} \succ\succ_{spo} \{t_1, \ldots, t_m\}$

in addition you need $s \sqsupseteq t$ to ensure monotonicity, where $\sqsupseteq$ is monotonic.

The (Monotonic) Semantic Path Ordering

(Borralleras, Ferreira and Rubio 2000)

$$s = f(s_1, \ldots, s_n) \succ_{spo} t \quad \text{if and only if}$$

1. $s_i \succeq_{spo} t$ for some $1 \leq i \leq n$.
2. $s \sqsupset t$ and $s \succ_{spo} t_j$ for all $1 \leq j \leq m$.

in addition you need $s \sqsupseteq t$ to ensure monotonicity, where $\sqsupseteq$ is monotonic.

The (Monotonic) Semantic Path Ordering

(Borralleras, Ferreira and Rubio 2000)

$$s = f(s_1, \ldots, s_n) \succ_{spo} t \quad \text{if and only if}$$

1. $s_i \succeq_{spo} t$ for some $1 \leq i \leq n$.
2. $s \sqsupset t$ and $s \succ_{spo} t_j$ for all $1 \leq j \leq m$.

The (Monotonic) Semantic Path Ordering

(Borralleras, Ferreira and Rubio 2000)

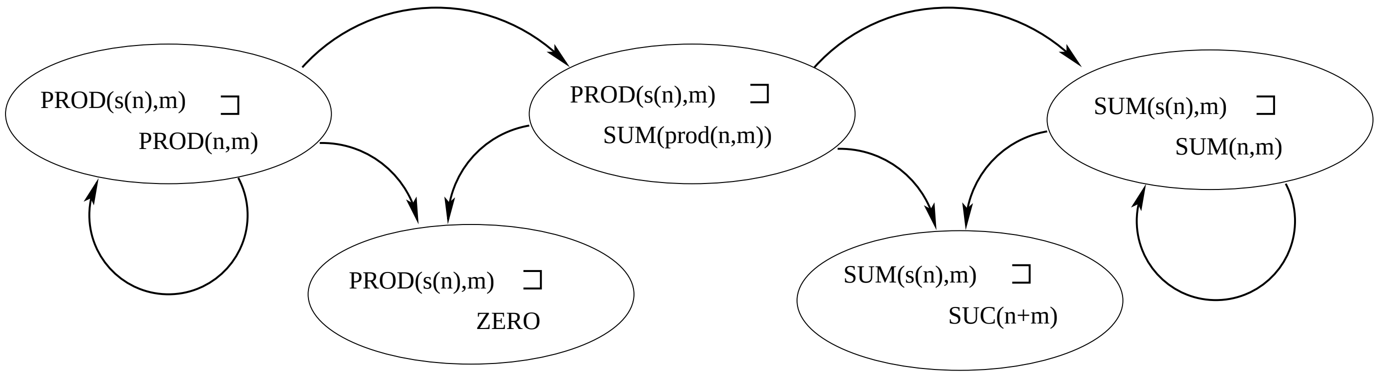
$$s = f(s_1, \ldots, s_n) \succ_{spo} t \quad \text{if and only if}$$

1. $s_i \succeq_{spo} t$ for some $1 \leq i \leq n$.
2. $s \sqsupset t$ and $s \succ_{spo} t_j$ for all $1 \leq j \leq m$.

$$\begin{array}{lcllcl}
prod(0, m) & \rightarrow & 0 & prod(0, m) & \sqsupset & ZERO \\
prod(s(n), m) & \rightarrow & prod(n, m) + m & PROD(s(n), m) & \sqsupset & SUM(prod(n, m), m) \\
 & & & PROD(s(n), m) & \sqsupset & PROD(n, m) \\
0 + m & \rightarrow & m & & & \\
s(n) + m & \rightarrow & s(n + m) & SUM(s(n), m) & \sqsupset & SUM(n, m) \\
 & & & SUM(s(n), m) & \sqsupset & SUC(n + m)
\end{array}$$

The (Monotonic) Semantic Path Ordering

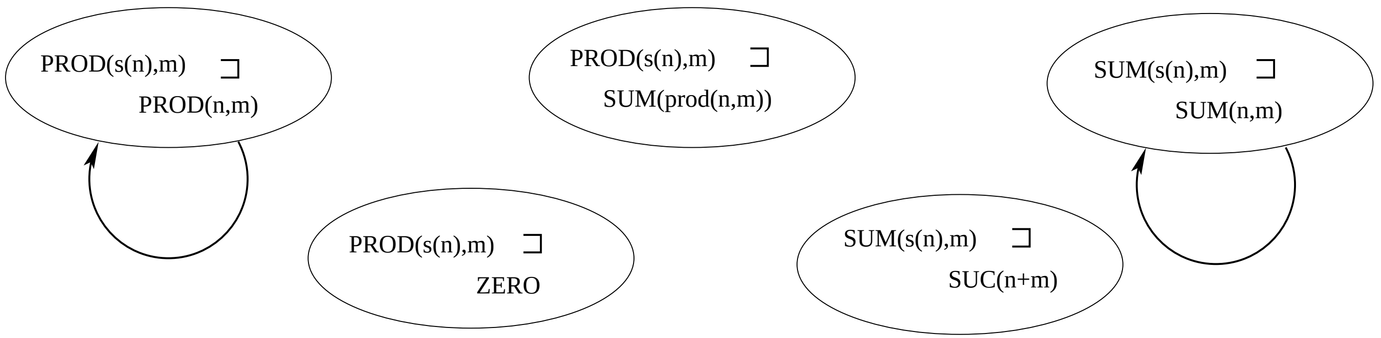
(Borralleras, Ferreira and Rubio 2000)



$$
\begin{array}{lcllcl}
prod(0,m) & \to & 0 & prod(0,m) & \sqsupset & ZERO \\
prod(s(n),m) & \to & prod(n,m)+m & PROD(s(n),m) & \sqsupset & SUM(prod(n,m),m) \\
 & & & PROD(s(n),m) & \sqsupset & PROD(n,m) \\
0+m & \to & m & & & \\
s(n)+m & \to & s(n+m) & SUM(s(n),m) & \sqsupset & SUM(n,m) \\
 & & & SUM(s(n),m) & \sqsupset & SUC(n+m)
\end{array}
$$

The (Monotonic) Semantic Path Ordering

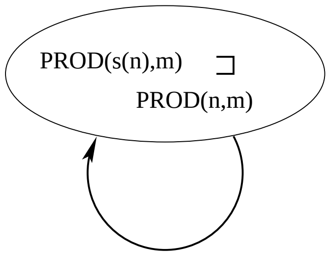
(Borralleras, Ferreira and Rubio 2000)

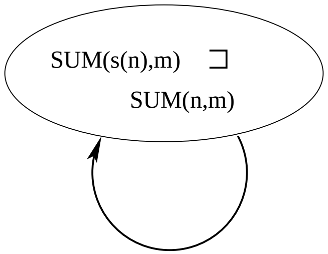


$prod(0,m)$	$\to$	0	$prod(0,m)$	$\sqsupset$	$ZERO$
$prod(s(n),m)$	$\to$	$prod(n,m)+m$	$PROD(s(n),m)$	$\sqsupset$	$SUM(prod(n,m),m)$
			$PROD(s(n),m)$	$\sqsupset$	$PROD(n,m)$
$0+m$	$\to$	m			
$s(n)+m$	$\to$	$s(n+m)$	$SUM(s(n),m)$	$\sqsupset$	$SUM(n,m)$
			$SUM(s(n),m)$	$\sqsupset$	$SUC(n+m)$

The (Monotonic) Semantic Path Ordering

(Borralleras, Ferreira and Rubio 2000)





$$
\begin{array}{lcllcl}
prod(0,m) & \rightarrow & 0 & prod(0,m) & \sqsupset & ZERO \\
prod(s(n),m) & \rightarrow & prod(n,m)+m & PROD(s(n),m) & \sqsupset & SUM(prod(n,m),m) \\
 & & & PROD(s(n),m) & \sqsupset & PROD(n,m) \\
0+m & \rightarrow & m & & & \\
s(n)+m & \rightarrow & s(n+m) & SUM(s(n),m) & \sqsupset & SUM(n,m) \\
 & & & SUM(s(n),m) & \sqsupset & SUC(n+m)
\end{array}
$$

The (Monotonic) Semantic Path Ordering

(Borralleras, Ferreira and Rubio 2000)

$$s = f(s_1, \ldots, s_n) \succ_{spo} t \quad \text{if and only if}$$

1. $s_i \succeq_{spo} t$ for some $1 \leq i \leq n$.
2. $s \sqsupset t$ and $s \succ_{spo} t_j$ for all $1 \leq j \leq m$.
3.  $s \sqsupseteq t$ and $\{s_1, \ldots, s_n\} \succ\!\succ_{spo} \{t_1, \ldots, t_m\}$

The (Monotonic) Semantic Path Ordering

(Borralleras, Ferreira and Rubio 2000)

$$s = f(s_1, \ldots, s_n) \succ_{spo} t \quad \text{if and only if}$$

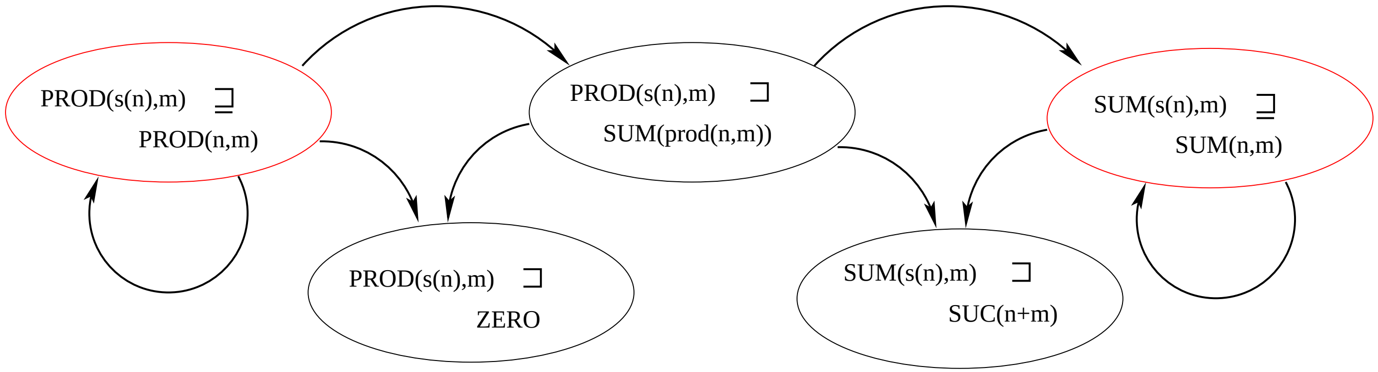
1. $s_i \succeq_{spo} t$ for some $1 \leq i \leq n$.
2. $s \sqsupset t$ and $s \succ_{spo} t_j$ for all $1 \leq j \leq m$.

3. $s \sqsupseteq t$ and $\{s_1, \ldots, s_n\} \succ\succ_{spo} \{t_1, \ldots, t_m\}$

$$
\begin{array}{lcllcl}
prod(0,m) & \rightarrow & 0 & prod(0,m) & \sqsupset & ZERO \\
prod(s(n),m) & \rightarrow & prod(n,m)+m & PROD(s(n),m) & \sqsupset & SUM(prod(n,m),m) \\
 & & & PROD(s(n),m) & \sqsupseteq & PROD(n,m) \\
0+m & \rightarrow & m & & & \\
s(n)+m & \rightarrow & s(n+m) & SUM(s(n),m) & \sqsupset & SUM(n,m) \\
 & & & SUM(s(n),m) & \sqsupseteq & SUC(n+m)
\end{array}
$$

The (Monotonic) Semantic Path Ordering

(Borralleras, Ferreira and Rubio 2000)

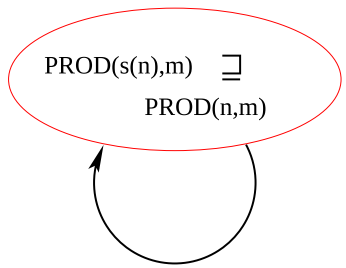


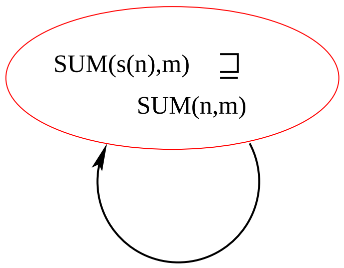
$$
\begin{array}{lcl}
prod(0,m) & \rightarrow & 0 \\
prod(s(n),m) & \rightarrow & prod(n,m)+m \\
 & & \\
0+m & \rightarrow & m \\
s(n)+m & \rightarrow & s(n+m)
\end{array}
$$

$$
\begin{array}{lcl}
prod(0,m) & \sqsupset & ZERO \\
PROD(s(n),m) & \sqsupset & SUM(prod(n,m),m) \\
PROD(s(n),m) & \sqsupseteq & PROD(n,m) \\
 & & \\
SUM(s(n),m) & \sqsupset & SUM(n,m) \\
SUM(s(n),m) & \sqsupseteq & SUC(n+m)
\end{array}
$$

The (Monotonic) Semantic Path Ordering

(Borralleras, Ferreira and Rubio 2000)





$$
\begin{array}{lcl@{\qquad}lcl}
prod(0,m) & \to & 0 & prod(0,m) & \sqsupset & ZERO \\
prod(s(n),m) & \to & prod(n,m)+m & PROD(s(n),m) & \sqsupset & SUM(prod(n,m),m) \\
 & & & PROD(s(n),m) & \sqsupseteq & PROD(n,m) \\
0+m & \to & m & & & \\
s(n)+m & \to & s(n+m) & SUM(s(n),m) & \sqsupset & SUM(n,m) \\
 & & & SUM(s(n),m) & \sqsupseteq & SUC(n+m)
\end{array}
$$

The (Monotonic) Semantic Path Ordering

(Borralleras, Ferreira and Rubio 2000)

Different pathes provide different constraints to be solved.

The Dependency Pair constraint is one of them

It is unkown whether there are, in general, better constraints than the one used by the Dependency Pair Method.

$$
\begin{array}{lcllcl}
prod(0,m) & \rightarrow & 0 & prod(0,m) & \sqsupset & ZERO \\
prod(s(n),m) & \rightarrow & prod(n,m)+m & PROD(s(n),m) & \sqsupset & SUM(prod(n,m),m) \\
 & & & PROD(s(n),m) & \sqsupseteq & PROD(n,m) \\
0+m & \rightarrow & m & & & \\
s(n)+m & \rightarrow & s(n+m) & SUM(s(n),m) & \sqsupset & SUM(n,m) \\
 & & & SUM(s(n),m) & \sqsupseteq & SUC(n+m)
\end{array}
$$

Comparison of both methods

Monotonic Semantic Path Ordering	Dependency Pairs
Positive	Positive
Easy to extend	Conceptually simple
AC-case [Borralleras Thesis 2003]*	
CS-case [Borralleras Thesis 2003]	
HO-case [Borralleras Thesis 2003]*	
Constraint Framework [Borralleras Thesis 2003]*	Constraint Framework [GTS2004]

Comparison of both methods

Monotonic Semantic Path Ordering	Dependency Pairs
Negative	Negative
Conceptualy more difficult	Harder to extend e.g. HO-case

XXI century motivation

Present and Future

Our main goal should be to provide termination tools for

XXI century motivation

Present and Future

Our main goal should be to provide termination tools for

- programming languages
  - logic programs (PROLOG [SGST2006]; AProVE)
  - functional programs (Haskell [GSST2006]; AProVE)
  - imperative programs (e.g. recent work [FK2008])

XXI century motivation

Present and Future

Our main goal should be to provide termination tools for

- programming languages
  - logic programs (PROLOG [SGST2006]; AProVE)
  - functional programs (Haskell [GSST2006]; AProVE)
  - imperative programs (e.g. recent work [FK2008])
- logical frameworks
  - Maude (AProVE amd Mu-Term)
  - Isabelle (normal HO-rewriting with lambdas)
  - Coq (HO-rewriting with lambdas)

XXI century motivation

Present and Future

Our main goal should be to provide termination tools for

- programming languages
  - logic programs (PROLOG [SGST2006]; AProVE)
  - functional programs (Haskell [GSST2006]; AProVE)
  - imperative programs (e.g. recent work [FK2008])
- logical frameworks
  - Maude (AProVE amd Mu-Term)
  - Isabelle (normal HO-rewriting with lambdas)
  - Coq (HO-rewriting with lambdas)

Making a lot of plublicity of performance results.
Certified Termination.
Keep existing successful known tools alive: e.g. CiME

XXI century motivation

Discourage risks

We have to avoid being a small closed community,
making it accessible for users, developers and researchers.
We need new fresh ideas from time to time.

XXI century motivation

Discourage risks

We have to avoid being a small closed community,
making it accessible for users, developers and researchers.
We need new fresh ideas from time to time.
Are there too many paper on termination of rewriting?

XXI century motivation

Discourage risks

We have to avoid being a small closed community,
making it accessible for users, developers and researchers.
We need new fresh ideas from time to time.
Are there too many paper on termination of rewriting?
The numbers are the following (only for RTA):

XXI century motivation

Discourage risks

We have to avoid being a small closed community,
making it accessible for users, developers and researchers.
We need new fresh ideas from time to time.
Are there too many paper on termination of rewriting?
The numbers are the following (only for RTA):

- RTA 2008: one third of the papers are on termination.
- RTA 2005-2008: it is the 30 % of the papers (34 papers).
- RTA 2001-2004: it is the 18 % of the papers (18.5 papers).

XXI century motivation

Discourage risks

We have to avoid being a small closed community,
making it accessible for users, developers and researchers.
We need new fresh ideas from time to time.
Are there too many paper on termination of rewriting?
The numbers are the following (only for RTA):

- RTA 2008: one third of the papers are on termination.
- RTA 2005-2008: it is the 30 % of the papers (34 papers).
- RTA 2001-2004: it is the 18 % of the papers (18.5 papers).

It's hard to compare the real progress in two periods, but, at least it's a bit surprising.

XXI century motivation

Discourage risks

XXI century motivation

Discourage risks

Attacking undecidable problems has a risk of always finding an example to handle by a new improvement

XXI century motivation

Discourage risks

Attacking undecidable problems has a risk of always finding an example to handle by a new improvement

Research on first-order theorem proving has had a different behavior

even though they also have a competition, a large list of problems and an undecidable problem.

Potential sources of improvement

Constraint Solving techniques

Finding orderings to ensure that all cycles decrease it's a key ingredient in all provers
Like, for instance, SAT solvers or SMT solvers in many verification tools.
There have been several recent results on finding:

- RPO, LPO, KBO
- polynomial interpretations (integers and reals)
- matrix interpretations

by translating the problem into SAT or integer programming.

Potential sources of improvement

Constraint Solving techniques

These solvers can be used as an ingredient for the termination tool, and it is possible to use them as a black box.
Termination tools for other languages can also make use of them.

Potential sources of improvement

Constraint Solving techniques

These solvers can be used as an ingredient for the termination tool, and it is possible to use them as a black box.
Termination tools for other languages can also make use of them.

How can we compare them?

Are there other useful functionalities?

Potential sources of improvement

Constraint Solving techniques

These solvers can be used as an ingredient for the termination tool, and it is possible to use them as a black box.
Termination tools for other languages can also make use of them.

How can we compare them?

Are there other useful functionalities?

- being incremental, i.e. adding new constraints

Potential sources of improvement

Constraint Solving techniques

These solvers can be used as an ingredient for the termination tool, and it is possible to use them as a black box.
Termination tools for other languages can also make use of them.

How can we compare them?

Are there other useful functionalities?

- being incremental, i.e. adding new constraints
- being backtrackable, i.e. removing the last added constraints

reusing the work done in previous stages.

Potential sources of improvement

Constraint Solving techniques

These solvers can be used as an ingredient for the termination tool, and it is possible to use them as a black box.
Termination tools for other languages can also make use of them.

How can we compare them?

Are there other useful functionalities?

- being incremental, i.e. adding new constraints
- being backtrackable, i.e. removing the last added constraints

reusing the work done in previous stages.

These solvers, in particular the ones on polynomial interpretations, may be useful in other contexts.

Potential sources of improvement

Constraint Solving techniques

Like in 2003 with the termination on competition, I have a proposal to start a competion on solving these constraints.

Potential sources of improvement

Constraint Solving techniques

Like in 2003 with the termination on competition, I have a proposal to start a competion on solving these constraints.
For instance,

- finding RPOs
- finding polynomial interpretations over the integers
- finding polynomial interpretations over the reals
- ....

Potential sources of improvement

Constraint Solving techniques

Like in 2003 with the termination on competition, I have a proposal to start a competion on solving these constraints.
For instance,

- finding RPOs
- finding polynomial interpretations over the integers
- finding polynomial interpretations over the reals
- ....

The existing systems can provide the problems to be solved.
There is no need to implement a full competitive termination prover.
It can be atractive for new participants.
It can be done together with the termination competition or not

Potential sources of improvement

Trace analysis

In all currently existing methods potentially looping traces are detected by analyzing the dependency graph.

There exist many other techniques, for instance in program analysis, for approximating exectution traces, and so detecting potential loops.

I don't know about any work on termination of rewriting using, for instance, predicate abstraction techniques.

New techniques need to be scalable to large programs.

This would be an alternative way of obtaining ordering constraints to ensure termination of all potential loops.
These constraints can be solved by the same solvers we have just described.

Potential sources of improvement

Built-in theories

This is a crucial matter in other verification tools.

Only very recently [FK2008] there has been an attempt to handle built-in intergers and other theories when proving termination of rewriting.
Having built-in integers is mandatory for many aplications.

We need to study how it combines with

- existing and new trace analysis techniques and
- the constraint solving techniques.

Maybe constraint solving should be restricted to polynomial interpretations, but not necessarily.

Conclusion

How termination will be in 10 years?

- We have to make it more accessible to other communities.

Conclusion

How termination will be in 10 years?

- We have to make it more accessible to other communities.
- Make solvers useful for other purposes, and use this to make publicity of our termination tools.

Conclusion

How termination will be in 10 years?

- We have to make it more accessible to other communities.
- Make solvers useful for other purposes, and use this to make publicity of our termination tools.
- Find new fresh ideas maybe coming from other areas.

Conclusion

How termination will be in 10 years?

- We have to make it more accessible to other communities.
- Make solvers useful for other purposes, and use this to make publicity of our termination tools.
- Find new fresh ideas maybe coming from other areas.
- Make a lot of publicity of real code termination proofs (better if there are thousands of lines!).

Conclusion

How termination will be in 10 years?

- We have to make it more accessible to other communities.
- Make solvers useful for other purposes, and use this to make publicity of our termination tools.
- Find new fresh ideas maybe coming from other areas.
- Make a lot of publicity of real code termination proofs (better if there are thousands of lines!).
- Make our tools available for logical frameworks:
  Maude, Isabelle, Coq, ...

Conclusion

How termination will be in 10 years?

- We have to make it more accessible to other communities.
- Make solvers useful for other purposes, and use this to make publicity of our termination tools.
- Find new fresh ideas maybe coming from other areas.
- Make a lot of publicity of real code termination proofs (better if there are thousands of lines!).
- Make our tools available for logical frameworks: Maude, Isabelle, Coq, ...
- Look for a new breakthrough!

Conclusion

How termination will be in 10 years?

- We have to make it more accessible to other communities.
- Make solvers useful for other purposes, and use this to make publicity of our termination tools.
- Find new fresh ideas maybe coming from other areas.
- Make a lot of publicity of real code termination proofs (better if there are thousands of lines!).
- Make our tools available for logical frameworks: Maude, Isabelle, Coq, ...
- Look for a new breakthrough!
- and will see....