# Topics in Termination of Term Rewriting

Adam Koprowski
(joint work with Hans Zantema)

Eindhoven University of Technology
Department of Mathematics and Computer Science

16 March 2006
Prose (Process Seminar)

# Outline

1. Project overview
   - Proving termination
   - Automation of termination proving
   - Certification of termination proving
   - Application of termination proving
   - Project roadmap
2. Recursive path ordering for infinite labelled systems
   - Semantic labelling
   - Recursive path ordering (RPO)
   - RPO for infinite systems
   - Automation of RPO for infinite systems

# Outline

1. Project overview
   - Proving termination
   - Automation of termination proving
   - Certification of termination proving
   - Application of termination proving
   - Project roadmap
2. Recursive path ordering for infinite labelled systems
   - Semantic labelling
   - Recursive path ordering (RPO)
   - RPO for infinite systems
   - Automation of RPO for infinite systems

# Outline

1. Project overview
   - Proving termination
   - Automation of termination proving
   - Certification of termination proving
   - Application of termination proving
   - Project roadmap
2. Recursive path ordering for infinite labelled systems
   - Semantic labelling
   - Recursive path ordering (RPO)
   - RPO for infinite systems
   - Automation of RPO for infinite systems

# Outline

1. Project overview
   - Proving termination
   - Automation of termination proving
   - Certification of termination proving
   - Application of termination proving
   - Project roadmap

2. Recursive path ordering for infinite labelled systems
   - Semantic labelling
   - Recursive path ordering (RPO)
   - RPO for infinite systems
   - Automation of RPO for infinite systems

# Outline

1. Project overview
   - Proving termination
   - Automation of termination proving
   - Certification of termination proving
   - Application of termination proving
   - Project roadmap
2. Recursive path ordering for infinite labelled systems
   - Semantic labelling
   - Recursive path ordering (RPO)
   - RPO for infinite systems
   - Automation of RPO for infinite systems

# Outline

1. Project overview
   - Proving termination
   - Automation of termination proving
   - Certification of termination proving
   - Application of termination proving
   - Project roadmap
2. Recursive path ordering for infinite labelled systems
   - Semantic labelling
   - Recursive path ordering (RPO)
   - RPO for infinite systems
   - Automation of RPO for infinite systems

# Outline

1. Project overview
   - Proving termination
   - Automation of termination proving
   - Certification of termination proving
   - Application of termination proving
   - Project roadmap
2. Recursive path ordering for infinite labelled systems
   - Semantic labelling
   - Recursive path ordering (RPO)
   - RPO for infinite systems
   - Automation of RPO for infinite systems

# Outline

1. Project overview
   - Proving termination
   - Automation of termination proving
   - Certification of termination proving
   - Application of termination proving
   - Project roadmap
2. Recursive path ordering for infinite labelled systems
   - Semantic labelling
   - Recursive path ordering (RPO)
   - RPO for infinite systems
   - Automation of RPO for infinite systems

# Outline

1. Project overview
   - Proving termination
   - Automation of termination proving
   - Certification of termination proving
   - Application of termination proving
   - Project roadmap
2. Recursive path ordering for infinite labelled systems
   - Semantic labelling
   - Recursive path ordering (RPO)
   - RPO for infinite systems
   - Automation of RPO for infinite systems

# Outline

# Introduction to term rewriting

Let's define plus in Peano arithmetic.

**Example (Plus)**

$$
\begin{aligned}
0 + y &= y \\
s(x) + y &= s(x + y)
\end{aligned}
$$

**Example (Computing with plus)**

Now let us do some some maths... how about $2 + 2$?

$$
\begin{aligned}
& s(s(0)) + s(s(0)) \rightarrow s(s(0) + s(s(0))) \rightarrow \\
& s(s(0 + s(s(0)))) \rightarrow s(s(s(s(0))))
\end{aligned}
$$

A TRS is terminating iff it does not admin infinite reductions.

# Introduction to term rewriting

Let's define plus in Peano arithmetic.

**Example (Plus)**

$$
\begin{aligned}
0 + y &= y \\
s(x) + y &= s(x + y)
\end{aligned}
$$

**Example (Computing with plus)**

Now let us do some some maths... how about $2 + 2$?

$$
\begin{aligned}
&s(s(0)) + s(s(0)) \to s(s(0) + s(s(0))) \to \\
&s(s(0 + s(s(0)))) \to s(s(s(s(0))))
\end{aligned}
$$

A TRS is terminating iff it does not admin infinite reductions.

# Introduction to term rewriting

Let's define plus in Peano arithmetic.

**Example (Plus)**

$$
\begin{array}{rcl}
0 + y & \rightarrow & y \\
s(x) + y & \rightarrow & s(x + y)
\end{array}
$$

**Example (Computing with plus)**

Now let us do some some maths... how about $2 + 2$?

$$
\begin{array}{l}
s(s(0)) + s(s(0)) \rightarrow s(s(0) + s(s(0))) \rightarrow \\
s(s(0 + s(s(0)))) \rightarrow s(s(s(s(0))))
\end{array}
$$

A TRS is terminating iff it does not admin infinite reductions.

# Introduction to term rewriting

Let's define plus in Peano arithmetic.

## Example (Plus)

$$\begin{array}{rcl} 0 + y & \to & y \\ s(x) + y & \to & s(x + y) \end{array}$$

## Example (Computing with plus)

Now let us do some some maths... how about $2 + 2$?

$$s(s(0)) + s(s(0)) \to s(s(0) + s(s(0))) \to$$
$$s(s(0 + s(s(0)))) \to s(s(s(s(0))))$$

A TRS is terminating iff it does not admin infinite reductions.

# Introduction to term rewriting

Let's define plus in Peano arithmetic.

## Example (Plus)

$$
\begin{aligned}
0 + y &\to y \\
s(x) + y &\to s(x + y)
\end{aligned}
$$

## Example (Computing with plus)

Now let us do some some maths... how about $2 + 2$?

$$
s(s(0)) + s(s(0)) \to s(s(0) + s(s(0))) \to s(s(0 + s(s(0)))) \to s(s(s(s(0))))
$$

A TRS is terminating iff it does not admin infinite reductions.

# Introduction to term rewriting

Let's define plus in Peano arithmetic.

## Example (Plus)

$$\begin{aligned} 0 + y &\to y \\ s(x) + y &\to s(x + y) \end{aligned}$$

## Example (Computing with plus)

Now let us do some some maths... how about $2 + 2$?

$$s(s(0)) + s(s(0)) \to s(s(0) + s(s(0))) \to$$
$$s(s(0 + s(s(0)))) \to s(s(s(s(0))))$$

A TRS is terminating iff it does not admin infinite reductions.

# Introduction to term rewriting

Let's define plus in Peano arithmetic.

**Example (Plus)**

$$\begin{array}{rcl} 0 + y & \rightarrow & y \\ s(x) + y & \rightarrow & s(x + y) \end{array}$$

**Example (Computing with plus)**

Now let us do some some maths... how about $2 + 2$?

$$s(s(0)) + s(s(0)) \rightarrow s(s(0) + s(s(0))) \rightarrow$$
$$s(s(0 + s(s(0)))) \rightarrow s(s(s(s(0))))$$

A TRS is terminating iff it does not admin infinite reductions.

# Introduction to term rewriting

Let's define plus in Peano arithmetic.

## Example (Plus)

$$\begin{array}{rcl} 0 + y & \to & y \\ s(x) + y & \to & s(x + y) \end{array}$$

## Example (Computing with plus)

Now let us do some some maths... how about $2 + 2$?

$$s(s(0)) + s(s(0)) \to s(s(0) + s(s(0))) \to$$
$$s(s(0 + s(s(0)))) \to s(s(s(s(0))))$$

A TRS is terminating iff it does not admin infinite reductions.

# Introduction to term rewriting

Let's define plus in Peano arithmetic.

**Example (Plus)**

$$\begin{aligned} 0 + y &\to y \\ s(x) + y &\to s(x + y) \end{aligned}$$

**Example (Computing with plus)**

Now let us do some some maths... how about $2 + 2$?

$$\begin{gathered} s(s(0)) + s(s(0)) \to s(s(0) + s(s(0))) \to \\ s(s(0 + s(s(0)))) \to s(s(s(s(0)))) \end{gathered}$$

A TRS is terminating iff it does not admin infinite reductions.

# More examples

## Example (Minus)

$$\begin{aligned} x - 0 &\rightarrow x \\ s(x) - s(y) &\rightarrow x - y \end{aligned}$$

## Example (Min)

$$\begin{aligned} min(x, 0) &\rightarrow 0 \\ min(0, x) &\rightarrow 0 \\ min(s(x), s(y)) &\rightarrow s(min(x, y)) \end{aligned}$$

# More examples

## Example (Minus)

$$\begin{array}{rcl} x - 0 & \rightarrow & x \\ s(x) - s(y) & \rightarrow & x - y \end{array}$$

## Example (Min)

$$\begin{array}{rcl} min(x, 0) & \rightarrow & 0 \\ min(0, x) & \rightarrow & 0 \\ min(s(x), s(y)) & \rightarrow & s(min(x, y)) \end{array}$$

# Relative termination

**Example**

$$\begin{array}{rcl} T(I(x), y) & \rightarrow & T(x, y) \\ T(x, y) & \rightarrow & T(x, I(y)) \end{array}$$

This system is not terminating.

**Example**

$$\begin{array}{rcl} T(I(x), y) & \rightarrow & T(x, y) \\ T(x, y) & \xrightarrow{=} & T(x, I(y)) \end{array}$$

This system is relatively terminating.

# Relative termination

**Example**

$$\begin{array}{rcl} T(I(x), y) & \rightarrow & T(x, y) \\ T(x, y) & \rightarrow & T(x, I(y)) \end{array}$$

This system is not terminating.

**Example**

$$\begin{array}{rcl} T(I(x), y) & \rightarrow & T(x, y) \\ T(x, y) & \xrightarrow{=} & T(x, I(y)) \end{array}$$

This system is relatively terminating.

# Relative termination

## Example

$$
\begin{aligned}
T(I(x), y) &\rightarrow T(x, y) \\
T(x, y) &\rightarrow T(x, I(y))
\end{aligned}
$$

This system is not terminating.

## Example

$$
\begin{aligned}
T(I(x), y) &\rightarrow T(x, y) \\
T(x, y) &\xrightarrow{=} T(x, I(y))
\end{aligned}
$$

This system is relatively terminating.

# Relative termination

**Example**

$$\begin{array}{rcl} T(I(x), y) & \to & T(x, y) \\ T(x, y) & \to & T(x, I(y)) \end{array}$$

This system is not terminating.

**Example**

$$\begin{array}{rcl} T(I(x), y) & \to & T(x, y) \\ T(x, y) & \xrightarrow{=} & T(x, I(y)) \end{array}$$

This system is relatively terminating.

# How about commutativity and associativity?

## Example (Plus with commutativity rule)

$$\begin{array}{rcl} 0 + y & \rightarrow & y \\ s(x) + y & \rightarrow & s(x + y) \\ x + y & \rightarrow & y + x \end{array}$$

This system is not terminating.

This is addressed by rewriting modulo AC, which is a special case of rewriting modulo equations, which is generalized by relative termination.

## Example (Plus with commutativity rule cont.)

$$\begin{array}{rcl} 0 + y & \rightarrow & y \\ s(x) + y & \rightarrow & s(x + y) \\ x + y & \overset{=}{\rightarrow} & y + x \end{array}$$

This system is relatively terminating.

# How about commutativity and associativity?

**Example (Plus with commutativity rule)**

$$\begin{array}{rcl} 0 + y & \to & y \\ s(x) + y & \to & s(x+y) \\ x + y & \to & y + x \end{array}$$

This system is not terminating.

This is addressed by rewriting modulo AC,
which is a special case of rewriting modulo equations,
which is generalized by relative termination.

**Example (Plus with commutativity rule cont.)**

$$\begin{array}{rcl} 0 + y & \to & y \\ s(x) + y & \to & s(x+y) \\ x + y & \xrightarrow{=} & y + x \end{array}$$

This system is relatively terminating.

# How about commutativity and associativity?

**Example (Plus with commutativity rule)**

$$\begin{array}{rcl} 0 + y & \rightarrow & y \\ s(x) + y & \rightarrow & s(x + y) \\ x + y & \rightarrow & y + x \end{array}$$

This system is not terminating.

This is addressed by rewriting modulo AC,
which is a special case of rewriting modulo equations,
which is generalized by relative termination.

**Example (Plus with commutativity rule cont.)**

$$\begin{array}{rcl} 0 + y & \rightarrow & y \\ s(x) + y & \rightarrow & s(x + y) \\ x + y & \xrightarrow{=} & y + x \end{array}$$

This system is relatively terminating.

# How about commutativity and associativity?

**Example (Plus with commutativity rule)**

$$\begin{array}{rcl} 0 + y & \rightarrow & y \\ s(x) + y & \rightarrow & s(x + y) \\ x + y & \rightarrow & y + x \end{array}$$

This system is not terminating.

This is addressed by rewriting modulo AC,
which is a special case of rewriting modulo equations,
which is generalized by relative termination.

**Example (Plus with commutativity rule cont.)**

$$\begin{array}{rcl} 0 + y & \rightarrow & y \\ s(x) + y & \rightarrow & s(x + y) \\ x + y & \xrightarrow{=} & y + x \end{array}$$

This system is relatively terminating.

# How about commutativity and associativity?

**Example (Plus with commutativity rule)**

$$\begin{array}{rcl} 0 + y & \rightarrow & y \\ s(x) + y & \rightarrow & s(x+y) \\ x + y & \rightarrow & y + x \end{array}$$

This system is not terminating.

This is addressed by rewriting modulo AC,
which is a special case of rewriting modulo equations,
which is generalized by relative termination.

**Example (Plus with commutativity rule cont.)**

$$\begin{array}{rcl} 0 + y & \rightarrow & y \\ s(x) + y & \rightarrow & s(x+y) \\ x + y & \xrightarrow{=} & y + x \end{array}$$

This system is relatively terminating.

# How about commutativity and associativity?

**Example (Plus with commutativity rule)**

$$\begin{array}{rcl} 0 + y & \to & y \\ s(x) + y & \to & s(x + y) \\ x + y & \to & y + x \end{array}$$

This system is not terminating.

This is addressed by rewriting modulo AC, which is a special case of rewriting modulo equations, which is generalized by relative termination.

**Example (Plus with commutativity rule cont.)**

$$\begin{array}{rcl} 0 + y & \to & y \\ s(x) + y & \to & s(x + y) \\ x + y & \xrightarrow{=} & y + x \end{array}$$

This system is relatively terminating.

technische universiteit eindhoven

# How about commutativity and associativity?

**Example (Plus with commutativity rule)**

$$\begin{array}{rcl} 0 + y & \to & y \\ s(x) + y & \to & s(x + y) \\ x + y & \to & y + x \end{array}$$

This system is not terminating.

This is addressed by rewriting modulo AC,
which is a special case of rewriting modulo equations,
which is generalized by relative termination.

**Example (Plus with commutativity rule cont.)**

$$\begin{array}{rcl} 0 + y & \to & y \\ s(x) + y & \to & s(x + y) \\ x + y & \xrightarrow{=} & y + x \end{array}$$

This system is relatively terminating.

# Outline

1. Project overview
   - Proving termination
   - Automation of termination proving
   - Certification of termination proving
   - Application of termination proving
   - Project roadmap

2. Recursive path ordering for infinite labelled systems

# Automation of termination proving

- Termination is undecidable.
- But there are many techniques for proving termination.
- Recently focus in the area is on automation of termination proving process.
- An annual competition of termination tools is being organized.
- There is a number of tools from different authors: AProVE, TTT, TORPA, TPA, Jambox, C*i*ME, Matchbox, TEPARLA, Cariboo, Termptation.

# Automation of termination proving

- Termination is undecidable.
- But there are many techniques for proving termination.
- Recently focus in the area is on automation of termination proving process.
- An annual competition of termination tools is being organized.
- There is a number of tools from different authors: AProVE, TTT, TORPA, TPA, Jambox, C*i*ME, Matchbox, TEPARLA, Cariboo, Termptation.

# Automation of termination proving

- Termination is undecidable.
- But there are many techniques for proving termination.
- Recently focus in the area is on automation of termination proving process.
- An annual competition of termination tools is being organized.
- There is a number of tools from different authors: AProVE, TTT, TORPA, TPA, Jambox, C*i*ME, Matchbox, TEPARLA, Cariboo, Termptation.

# Automation of termination proving

- Termination is undecidable.
- But there are many techniques for proving termination.
- Recently focus in the area is on automation of termination proving process.
- An annual competition of termination tools is being organized.
- There is a number of tools from different authors: AProVE, TTT, TORPA, TPA, Jambox, C*i*ME, Matchbox, TEPARLA, Cariboo, Termptation.

# Automation of termination proving

- Termination is undecidable.
- But there are many techniques for proving termination.
- Recently focus in the area is on automation of termination proving process.
- An annual competition of termination tools is being organized.
- There is a number of tools from different authors: AProVE, TTT, TORPA, TPA, Jambox, C*i*ME, Matchbox, TEPARLA, Cariboo, Termptation.

# TPA: a tool for proving termination

## TPA

TPA: Termination Proved Automatically

`http://www.win.tue.nl/tpa`

Why developing yet another tool? What makes TPA different?

- Support for relative termination.
- Usage of semantic labelling with natural numbers.
- Aiming at certified proofs.

TPA facts:

- Written in Ocaml (around 10,000 lines of code).
- 3rd place in the last edition of the termination competition (Nara 2005).

# TPA: a tool for proving termination

**TPA**

TPA: Termination Proved Automatically

`http://www.win.tue.nl/tpa`

Why developing yet another tool? What makes TPA different?

- Support for relative termination.
- Usage of semantic labelling with natural numbers.
- Aiming at certified proofs.

TPA facts:

- Written in Ocaml (around 10,000 lines of code).
- 3rd place in the last edition of the termination competition (Nara 2005).

# TPA: a tool for proving termination

**TPA**

TPA: Termination Proved Automatically

`http://www.win.tue.nl/tpa`

Why developing yet another tool? What makes TPA different?

- Support for relative termination.
- Usage of semantic labelling with natural numbers.
- Aiming at certified proofs.

TPA facts:

- Written in Ocaml (around 10,000 lines of code).
- 3rd place in the last edition of the termination competition (Nara 2005).

# TPA: a tool for proving termination

**TPA**

TPA: Termination Proved Automatically

`http://www.win.tue.nl/tpa`

Why developing yet another tool? What makes TPA different?

- Support for relative termination.
- Usage of semantic labelling with natural numbers.
- Aiming at certified proofs.

TPA facts:

- Written in Ocaml (around 10,000 lines of code).
- 3rd place in the last edition of the termination competition (Nara 2005).

# TPA: a tool for proving termination

## TPA

TPA: Termination Proved Automatically

`http://www.win.tue.nl/tpa`

Why developing yet another tool? What makes TPA different?

- Support for relative termination.
- Usage of semantic labelling with natural numbers.
- Aiming at certified proofs.

TPA facts:

- Written in Ocaml (around 10,000 lines of code).
- 3rd place in the last edition of the termination competition (Nara 2005).

# TPA: a tool for proving termination

## TPA

TPA: Termination Proved Automatically

`http://www.win.tue.nl/tpa`

Why developing yet another tool? What makes TPA different?

- Support for relative termination.
- Usage of semantic labelling with natural numbers.
- Aiming at certified proofs.

TPA facts:

- Written in Ocaml (around 10,000 lines of code).
- 3rd place in the last edition of the termination competition (Nara 2005).

# TPA: a tool for proving termination

## TPA

TPA: Termination Proved Automatically

`http://www.win.tue.nl/tpa`

Why developing yet another tool? What makes TPA different?

- Support for relative termination.
- Usage of semantic labelling with natural numbers.
- Aiming at certified proofs.

TPA facts:

- Written in Ocaml (around 10,000 lines of code).
- 3rd place in the last edition of the termination competition (Nara 2005).

# TPA: a tool for proving termination

**TPA**

TPA: Termination Proved Automatically

`http://www.win.tue.nl/tpa`

Why developing yet another tool? What makes TPA different?

- Support for relative termination.
- Usage of semantic labelling with natural numbers.
- Aiming at certified proofs.

TPA facts:

- Written in Ocaml (around 10,000 lines of code).
- 3rd place in the last edition of the termination competition (Nara 2005).

# Outline

# Certification of termination proving

It is not uncommon that termination tools contain bugs (just as any other piece of software does).

**CoLoR**

CoLoR: a Coq library on rewriting and termination

http://color.loria.fr

- Objective: formalization of theory of term rewriting in the theorem prover Coq.
- Ultimate objective: certification of termination proofs produced by tools for proving termination of rewriting.

# Certification of termination proving

It is not uncommon that termination tools contain bugs (just as any other piece of software does).

**CoLoR**

CoLoR: a Coq library on rewriting and termination

`http://color.loria.fr`

- Objective: formalization of theory of term rewriting in the theorem prover Coq.
- Ultimate objective: certification of termination proofs produced by tools for proving termination of rewriting.

# Certification of termination proving

It is not uncommon that termination tools contain bugs (just as any other piece of software does).

**CoLoR**

CoLoR: a Coq library on rewriting and termination
`http://color.loria.fr`

- Objective: formalization of theory of term rewriting in the theorem prover Coq.
- Ultimate objective: certification of termination proofs produced by tools for proving termination of rewriting.

# Certification of termination proving

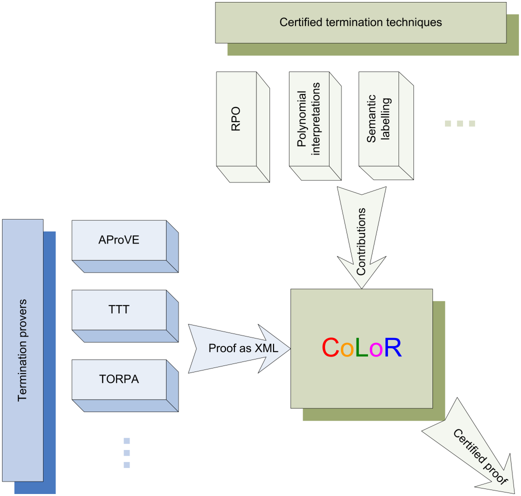
It is not uncommon that termination tools contain bugs (just as any other piece of software does).

**CoLoR**

CoLoR: a Coq library on rewriting and termination

`http://color.loria.fr`

- Objective: formalization of theory of term rewriting in the theorem prover Coq.
- Ultimate objective: certification of termination proofs produced by tools for proving termination of rewriting.


Certified termination techniques
RPO
Polynomial interpretations
Semantic labelling
Contributions
Termination provers
AProVE
TTT
TORPA
Proof as XML
CoLoR
Certified proof

# CoLoR contributions

To date contributions to CoLoR project:

- Polynomial interpretations [Hinderer]
- Dependancy pairs [Blanqui]
- Recursive path ordering (RPO) [Coupet-Grimal,Delobel]
- Higher-order recursive path ordering (HORPO) [Koprowski]

# CoLoR contributions

To date contributions to CoLoR project:

- Polynomial interpretations [Hinderer]
- Dependancy pairs [Blanqui]
- Recursive path ordering (RPO) [Coupet-Grimal,Delobel]
- Higher-order recursive path ordering (HORPO) [Koprowski]

# CoLoR contributions

To date contributions to CoLoR project:

- Polynomial interpretations [Hinderer]
- Dependancy pairs [Blanqui]
- Recursive path ordering (RPO) [Coupet-Grimal,Delobel]
- Higher-order recursive path ordering (HORPO) [Koprowski]

# CoLoR contributions

To date contributions to CoLoR project:

- Polynomial interpretations [Hinderer]
- Dependancy pairs [Blanqui]
- Recursive path ordering (RPO) [Coupet-Grimal,Delobel]
- Higher-order recursive path ordering (HORPO) [Koprowski]

# CoLoR contributions

To date contributions to CoLoR project:

- Polynomial interpretations [Hinderer]
- Dependancy pairs [Blanqui]
- Recursive path ordering (RPO) [Coupet-Grimal,Delobel]
- Higher-order recursive path ordering (HORPO) [Koprowski]

# Outline

1. Project overview
   - Proving termination
   - Automation of termination proving
   - Certification of termination proving
   - Application of termination proving
   - Project roadmap

2. Recursive path ordering for infinite labelled systems

# Verification and rewriting

- There are many theoretical results in term rewriting.
- There is also a tool support for termination.
- Idea: let's use that for verification.
- Concentrate on liveness properties.
- Concentrate on termination in rewriting.
- A restricted set of liveness properties.

# Verification and rewriting

- There are many theoretical results in term rewriting.
- There is also a tool support for termination.
- Idea: let's use that for verification.
- Concentrate on liveness properties.
- Concentrate on termination in rewriting.
- A restricted set of liveness properties.

# Verification and rewriting

- There are many theoretical results in term rewriting.
- There is also a tool support for termination.
- Idea: let's use that for verification.
- Concentrate on liveness properties.
- Concentrate on termination in rewriting.
- A restricted set of liveness properties.

# Verification and rewriting

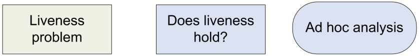
- There are many theoretical results in term rewriting.
- There is also a tool support for termination.
- Idea: let's use that for verification.
- Concentrate on liveness properties.
- Concentrate on termination in rewriting.
- A restricted set of liveness properties.

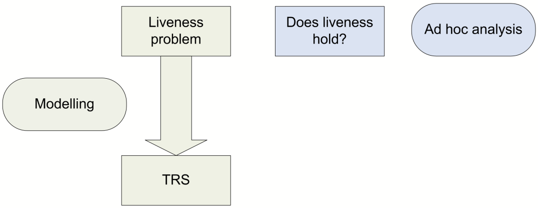
# Verification and rewriting

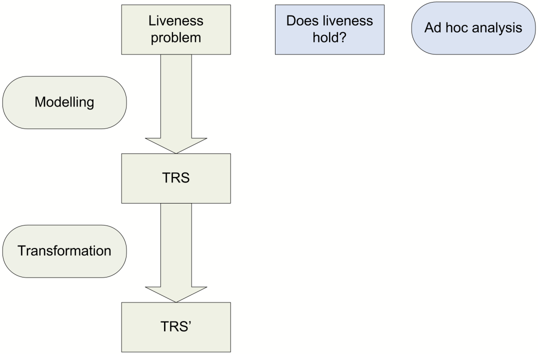
- There are many theoretical results in term rewriting.
- There is also a tool support for termination.
- Idea: let's use that for verification.
- Concentrate on liveness properties.
- Concentrate on termination in rewriting.
- A restricted set of liveness properties.

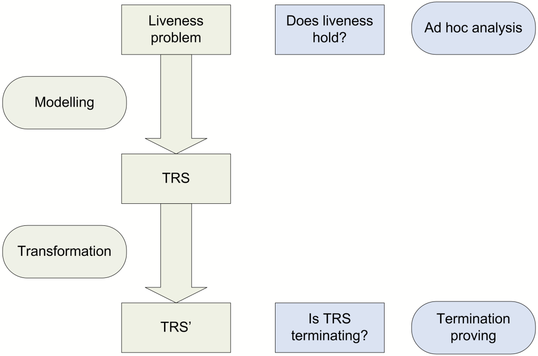
# Verification and rewriting

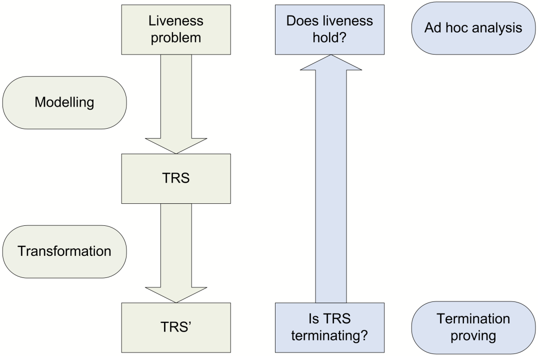
- There are many theoretical results in term rewriting.
- There is also a tool support for termination.
- Idea: let's use that for verification.
- Concentrate on liveness properties.
- Concentrate on termination in rewriting.
- A restricted set of liveness properties.


Liveness problem
Does liveness hold?
Ad hoc analysis


Liveness problem
Does liveness hold?
Ad hoc analysis
Modelling
TRS

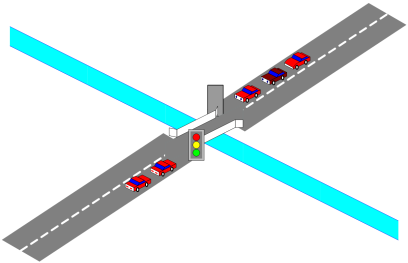

Liveness problem
Does liveness hold?
Ad hoc analysis
Modelling
TRS
Transformation
TRS'


Liveness problem
Does liveness hold?
Ad hoc analysis
Modelling
TRS
Transformation
TRS'
Is TRS terminating?
Termination proving


Liveness problem
Modelling
TRS
Transformation
TRS'
Does liveness hold?
Is TRS terminating?
Ad hoc analysis
Termination proving

# Liveness example: cars over a bridge

Liveness: no car will wait forever.

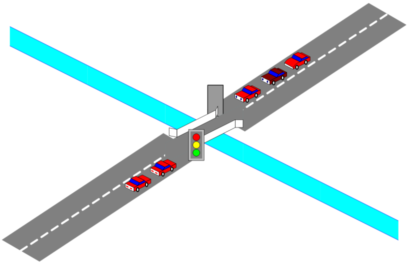


Remarks:

- We need an assumption about the lights: Fairness
- State space is infinite!

# Liveness example: cars over a bridge

Liveness: no car will wait forever.

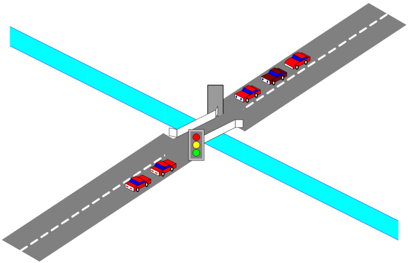


Remarks:

- We need an assumption about the lights: Fairness
- State space is infinite!

# Liveness example: cars over a bridge

Liveness: no car will wait forever.

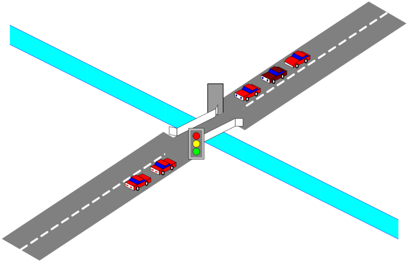


Remarks:

- We need an assumption about the lights: Fairness
- State space is infinite!

# Liveness example: cars over a bridge

Liveness: no car will wait forever.



Remarks:

- We need an assumption about the lights: Fairness
- State space is infinite!

# Cars over a bridge: model (1/2)

**System configuration**

top
left/right
left lane    right lane

**Possible system configuration**

top
left
car    bot
new    car
old    bot

**Lane configuration**

car
old/new    car
old/new
bot

# Cars over a bridge: model (1/2)

**System configuration**

top
left/right
left lane   right lane

**Possible system configuration**

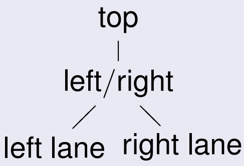
top
left
car   bot
new   car
old   bot

**Lane configuration**

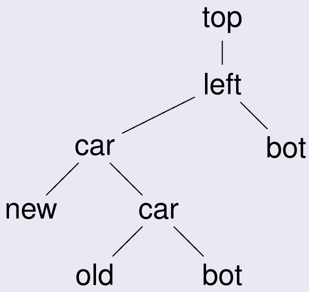
car
old/new   car
old/new
bot

# Cars over a bridge: model (1/2)

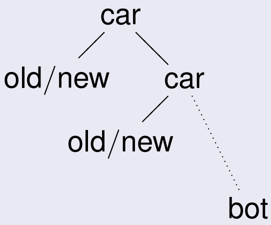
## System configuration




## Possible system configuration




## Lane configuration



# Cars over a bridge: model (2/2)

## Example

$$
\begin{array}{lrcl}
(1) & \mathrm{top}(\mathrm{left}(\mathrm{car}(x), y)) & \to & \mathrm{top}(\mathrm{right}(x, y)) \\
(2) & \mathrm{top}(\mathrm{right}(x, \mathrm{car}(y))) & \to & \mathrm{top}(\mathrm{left}(x, y)) \\
(3) & \mathrm{top}(\mathrm{left}(\mathrm{bot}, y)) & \to & \mathrm{top}(\mathrm{right}(\mathrm{bot}, y)) \\
(4) & \mathrm{top}(\mathrm{right}(x, \mathrm{bot})) & \to & \mathrm{top}(\mathrm{left}(x, \mathrm{bot})) \\
(5) & \mathrm{top}(\mathrm{left}(\mathrm{car}(x), y)) & \xrightarrow{=} & \mathrm{top}(\mathrm{left}(x, y)) \\
(6) & \mathrm{top}(\mathrm{right}(x, \mathrm{car}(y))) & \xrightarrow{=} & \mathrm{top}(\mathrm{right}(x, y)) \\
(7) & \mathrm{bot} & \xrightarrow{=} & \mathrm{new}(\mathrm{bot})
\end{array}
$$

# Cars over a bridge: model (2/2)

## Example

$$
\begin{array}{lrcl}
(1) & \mathsf{top}(\mathsf{left}(\mathsf{car}(x), y)) & \to & \mathsf{top}(\mathsf{right}(x, y)) \\
(2) & \mathsf{top}(\mathsf{right}(x, \mathsf{car}(y))) & \to & \mathsf{top}(\mathsf{left}(x, y)) \\
(3) & \mathsf{top}(\mathsf{left}(\mathsf{bot}, y)) & \to & \mathsf{top}(\mathsf{right}(\mathsf{bot}, y)) \\
(4) & \mathsf{top}(\mathsf{right}(x, \mathsf{bot})) & \to & \mathsf{top}(\mathsf{left}(x, \mathsf{bot})) \\
(5) & \mathsf{top}(\mathsf{left}(\mathsf{car}(x), y)) & \xrightarrow{=} & \mathsf{top}(\mathsf{left}(x, y)) \\
(6) & \mathsf{top}(\mathsf{right}(x, \mathsf{car}(y))) & \xrightarrow{=} & \mathsf{top}(\mathsf{right}(x, y)) \\
(7) & \mathsf{bot} & \xrightarrow{=} & \mathsf{new}(\mathsf{bot})
\end{array}
$$

# Cars over a bridge: model (2/2)

## Example

$$
\begin{array}{lrcl}
(1) & \mathrm{top}(\mathrm{left}(\mathrm{car}(x), y)) & \to & \mathrm{top}(\mathrm{right}(x, y)) \\
(2) & \mathrm{top}(\mathrm{right}(x, \mathrm{car}(y))) & \to & \mathrm{top}(\mathrm{left}(x, y)) \\
(3) & \mathrm{top}(\mathrm{left}(\mathrm{bot}, y)) & \to & \mathrm{top}(\mathrm{right}(\mathrm{bot}, y)) \\
(4) & \mathrm{top}(\mathrm{right}(x, \mathrm{bot})) & \to & \mathrm{top}(\mathrm{left}(x, \mathrm{bot})) \\
(5) & \mathrm{top}(\mathrm{left}(\mathrm{car}(x), y)) & \xrightarrow{=} & \mathrm{top}(\mathrm{left}(x, y)) \\
(6) & \mathrm{top}(\mathrm{right}(x, \mathrm{car}(y))) & \xrightarrow{=} & \mathrm{top}(\mathrm{right}(x, y)) \\
(7) & \mathrm{bot} & \xrightarrow{=} & \mathrm{new}(\mathrm{bot})
\end{array}
$$

# Cars over a bridge: model (2/2)

## Example

$$
\begin{array}{lrcl}
(1) & \mathsf{top}(\mathsf{left}(\mathsf{car}(x), y)) & \to & \mathsf{top}(\mathsf{right}(x, y)) \\
(2) & \mathsf{top}(\mathsf{right}(x, \mathsf{car}(y))) & \to & \mathsf{top}(\mathsf{left}(x, y)) \\
(3) & \mathsf{top}(\mathsf{left}(\mathsf{bot}, y)) & \to & \mathsf{top}(\mathsf{right}(\mathsf{bot}, y)) \\
(4) & \mathsf{top}(\mathsf{right}(x, \mathsf{bot})) & \to & \mathsf{top}(\mathsf{left}(x, \mathsf{bot})) \\
(5) & \mathsf{top}(\mathsf{left}(\mathsf{car}(x), y)) & \xrightarrow{=} & \mathsf{top}(\mathsf{left}(x, y)) \\
(6) & \mathsf{top}(\mathsf{right}(x, \mathsf{car}(y))) & \xrightarrow{=} & \mathsf{top}(\mathsf{right}(x, y)) \\
(7) & \mathsf{bot} & \xrightarrow{=} & \mathsf{new}(\mathsf{bot})
\end{array}
$$

# Cars over a bridge: model (2/2)

## Example

$$
\begin{array}{lrcl}
(1) & \mathrm{top}(\mathrm{left}(\mathrm{car}(x), y)) & \rightarrow & \mathrm{top}(\mathrm{right}(x, y)) \\
(2) & \mathrm{top}(\mathrm{right}(x, \mathrm{car}(y))) & \rightarrow & \mathrm{top}(\mathrm{left}(x, y)) \\
(3) & \mathrm{top}(\mathrm{left}(\mathrm{bot}, y)) & \rightarrow & \mathrm{top}(\mathrm{right}(\mathrm{bot}, y)) \\
(4) & \mathrm{top}(\mathrm{right}(x, \mathrm{bot})) & \rightarrow & \mathrm{top}(\mathrm{left}(x, \mathrm{bot})) \\
(5) & \mathrm{top}(\mathrm{left}(\mathrm{car}(x), y)) & \xrightarrow{=} & \mathrm{top}(\mathrm{left}(x, y)) \\
(6) & \mathrm{top}(\mathrm{right}(x, \mathrm{car}(y))) & \xrightarrow{=} & \mathrm{top}(\mathrm{right}(x, y)) \\
(7) & \mathrm{bot} & \xrightarrow{=} & \mathrm{new}(\mathrm{bot})
\end{array}
$$

# Cars over a bridge: model (2/2)

## Example

$$
\begin{array}{lrcl}
(1) & \mathrm{top}(\mathrm{left}(\mathrm{car}(x), y)) & \rightarrow & \mathrm{top}(\mathrm{right}(x, y)) \\
(2) & \mathrm{top}(\mathrm{right}(x, \mathrm{car}(y))) & \rightarrow & \mathrm{top}(\mathrm{left}(x, y)) \\
(3) & \mathrm{top}(\mathrm{left}(\mathrm{bot}, y)) & \rightarrow & \mathrm{top}(\mathrm{right}(\mathrm{bot}, y)) \\
(4) & \mathrm{top}(\mathrm{right}(x, \mathrm{bot})) & \rightarrow & \mathrm{top}(\mathrm{left}(x, \mathrm{bot})) \\
(5) & \mathrm{top}(\mathrm{left}(\mathrm{car}(x), y)) & \stackrel{=}{\rightarrow} & \mathrm{top}(\mathrm{left}(x, y)) \\
(6) & \mathrm{top}(\mathrm{right}(x, \mathrm{car}(y))) & \stackrel{=}{\rightarrow} & \mathrm{top}(\mathrm{right}(x, y)) \\
(7) & \mathrm{bot} & \stackrel{=}{\rightarrow} & \mathrm{new}(\mathrm{bot})
\end{array}
$$

# Cars over a bridge: model (2/2)

## Example

$$
\begin{array}{lrcl}
(1) & \mathsf{top}(\mathsf{left}(\mathsf{car}(x), y)) & \rightarrow & \mathsf{top}(\mathsf{right}(x, y)) \\
(2) & \mathsf{top}(\mathsf{right}(x, \mathsf{car}(y))) & \rightarrow & \mathsf{top}(\mathsf{left}(x, y)) \\
(3) & \mathsf{top}(\mathsf{left}(\mathsf{bot}, y)) & \rightarrow & \mathsf{top}(\mathsf{right}(\mathsf{bot}, y)) \\
(4) & \mathsf{top}(\mathsf{right}(x, \mathsf{bot})) & \rightarrow & \mathsf{top}(\mathsf{left}(x, \mathsf{bot})) \\
(5) & \mathsf{top}(\mathsf{left}(\mathsf{car}(x), y)) & \xrightarrow{=} & \mathsf{top}(\mathsf{left}(x, y)) \\
(6) & \mathsf{top}(\mathsf{right}(x, \mathsf{car}(y))) & \xrightarrow{=} & \mathsf{top}(\mathsf{right}(x, y)) \\
(7) & \mathsf{bot} & \xrightarrow{=} & \mathsf{new}(\mathsf{bot})
\end{array}
$$

# Liveness transformation

J. Giesl and H. Zantema in 2003 proposed two transformations:

**Transformation L**

- Sound and complete.
- Complicated.

**Transformation LS**

- Only sound.
- Significantly simpler.

A. Koprowski and H. Zantema in 2005 presented another transformation:

**Transformation LT**

- Sound.
- Complete (with some additional assumptions).
- Not as complicated as L.

# Liveness transformation

J. Giesl and H. Zantema in 2003 proposed two transformations:

**Transformation L**

- Sound and complete.
- Complicated.

**Transformation LS**

- Only sound.
- Significantly simpler.

A. Koprowski and H. Zantema in 2005 presented another transformation:

**Transformation LT**

- Sound.
- Complete (with some additional assumptions).
- Not as complicated as L.

# Liveness transformation

J. Giesl and H. Zantema in 2003 proposed two transformations:

**Transformation L**

- Sound and complete.
- Complicated.

**Transformation LS**

- Only sound.
- Significantly simpler.

A. Koprowski and H. Zantema in 2005 presented another transformation:

**Transformation LT**

- Sound.
- Complete (with some additional assumptions).
- Not as complicated as L.

# Liveness transformation

J. Giesl and H. Zantema in 2003 proposed two transformations:

**Transformation L**

- Sound and complete.
- Complicated.

**Transformation LS**

- Only sound.
- Significantly simpler.

A. Koprowski and H. Zantema in 2005 presented another transformation:

**Transformation LT**

- Sound.
- Complete (with some additional assumptions).
- Not as complicated as L.

# Liveness transformation

J. Giesl and H. Zantema in 2003 proposed two transformations:

**Transformation L**
- Sound and complete.
- Complicated.

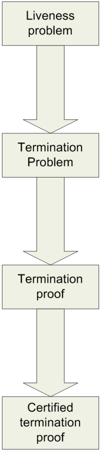
**Transformation LS**
- Only sound.
- Significantly simpler.

A. Koprowski and H. Zantema in 2005 presented another transformation:

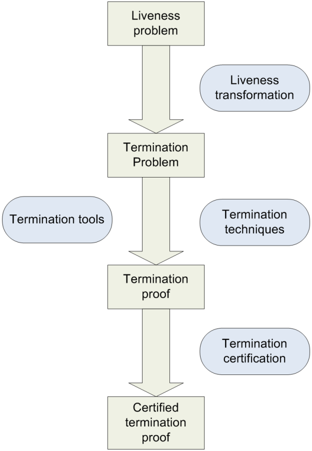
**Transformation LT**
- Sound.
- Complete (with some additional assumptions).
- Not as complicated as L.

# Outline

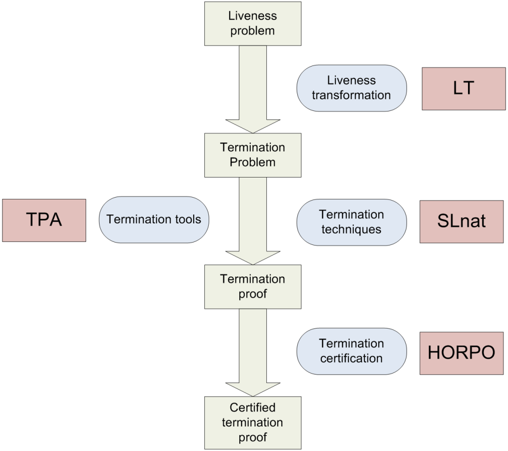
# Project roadmap



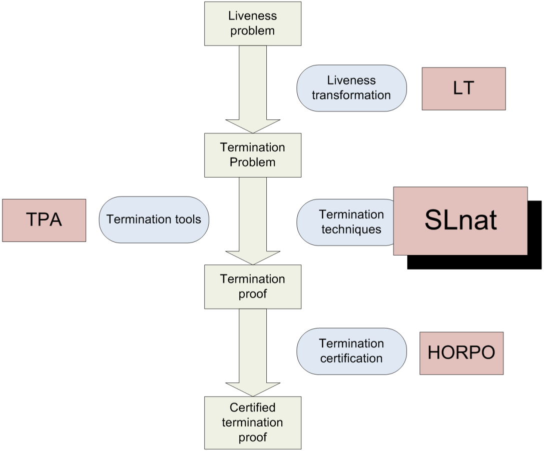
# Project roadmap



# Project roadmap



# Project roadmap



# Outline

1. Project overview

2. Recursive path ordering for infinite labelled systems
   - Semantic labelling
   - Recursive path ordering (RPO)
   - RPO for infinite systems
   - Automation of RPO for infinite systems

# Semantic labelling - introduction

Is the following system terminating?

## Example

$$
\begin{array}{lrcl}
(1) & \mathsf{f}(x, \mathsf{c}) & \to & \mathsf{c} \\
(2) & \mathsf{f}(\mathsf{c}, y) & \to & \mathsf{c} \\
(3) & \mathsf{f}(\mathsf{p}(x), \mathsf{p}(y)) & \to & \mathsf{p}(\mathsf{f}(x, y)) \\
(5) & \mathsf{g}(x, \mathsf{c}) & \to & x \\
(4) & \mathsf{g}(\mathsf{c}, y) & \to & y \\
(6) & \mathsf{g}(\mathsf{p}(x), \mathsf{p}(y)) & \to & \mathsf{p}(\mathsf{g}(x, y)) \\
(7) & \mathsf{w}(x, \mathsf{c}) & \to & x \\
(8) & \mathsf{w}(\mathsf{p}(x), \mathsf{p}(y) & \to & \mathsf{w}(x, y) \\
(9) & \mathsf{h}(\mathsf{p}(x), \mathsf{p}(y)) & \to & \mathsf{h}(\mathsf{w}(\mathsf{p}(\mathsf{g}(x, y)), \mathsf{p}(\mathsf{f}(x, y))), \mathsf{p}(\mathsf{f}(x, y)))
\end{array}
$$

# Semantic labelling - introduction

And how about this one?

## Example (GCD)

$$
\begin{array}{lrcl}
(1) & \mathrm{min}(x, 0) & \to & 0 \\
(2) & \mathrm{min}(0, y) & \to & 0 \\
(3) & \mathrm{min}(\mathrm{s}(x), \mathrm{s}(y)) & \to & \mathrm{s}(\mathrm{min}(x, y)) \\
(4) & \mathrm{max}(x, 0) & \to & x \\
(5) & \mathrm{max}(0, y) & \to & y \\
(6) & \mathrm{max}(\mathrm{s}(x), \mathrm{s}(y)) & \to & \mathrm{s}(\mathrm{max}(x, y)) \\
(7) & x - 0 & \to & x \\
(8) & \mathrm{s}(x) - \mathrm{s}(y) & \to & x - y \\
(9) & \mathrm{gcd}(\mathrm{s}(x), \mathrm{s}(y)) & \to & \mathrm{gcd}(\mathrm{s}(\mathrm{max}(x, y)) - \mathrm{s}(\mathrm{min}(x, y)), \\
 & & & \mathrm{s}(\mathrm{min}(x, y)))
\end{array}
$$

technische universiteit eindhoven

# Semantic labelling - introduction

And how about this one?

## Example (GCD)

$$
\begin{array}{rrcl}
(1) & \min(x, 0) & \to & 0 \\
(2) & \min(0, y) & \to & 0 \\
(3) & \min(\mathrm{s}(x), \mathrm{s}(y)) & \to & \mathrm{s}(\min(x, y)) \\
(4) & \max(x, 0) & \to & x \\
(5) & \max(0, y) & \to & y \\
(6) & \max(\mathrm{s}(x), \mathrm{s}(y)) & \to & \mathrm{s}(\max(x, y)) \\
(7) & x - 0 & \to & x \\
(8) & \mathrm{s}(x) - \mathrm{s}(y) & \to & x - y \\
(9) & \mathrm{gcd}(\mathrm{s}(x), \mathrm{s}(y)) & \to & \mathrm{gcd}(\mathrm{s}(\max(x, y)) - \mathrm{s}(\min(x, y)), \\
 & & & \mathrm{s}(\min(x, y)))
\end{array}
$$

technische universiteit eindhoven

# Semantic labelling - introduction

And how about this one?

## Example (GCD)

$$
\begin{array}{lrcl}
(1) & \mathrm{min}(x, 0) & \rightarrow & 0 \\
(2) & \mathrm{min}(0, y) & \rightarrow & 0 \\
(3) & \mathrm{min}(\mathrm{s}(x), \mathrm{s}(y)) & \rightarrow & \mathrm{s}(\mathrm{min}(x, y)) \\
(4) & \mathrm{max}(x, 0) & \rightarrow & x \\
(5) & \mathrm{max}(0, y) & \rightarrow & y \\
(6) & \mathrm{max}(\mathrm{s}(x), \mathrm{s}(y)) & \rightarrow & \mathrm{s}(\mathrm{max}(x, y)) \\
(7) & x - 0 & \rightarrow & x \\
(8) & \mathrm{s}(x) - \mathrm{s}(y) & \rightarrow & x - y \\
(9) & \mathrm{gcd}(\mathrm{s}(x), \mathrm{s}(y)) & \rightarrow & \mathrm{gcd}(\mathrm{s}(\mathrm{max}(x, y)) - \mathrm{s}(\mathrm{min}(x, y)), \\
 & & & \mathrm{s}(\mathrm{min}(x, y)))
\end{array}
$$

technische universiteit eindhoven

# Semantic labelling - introduction

And how about this one?

## Example (GCD)

$$
\begin{array}{lrcl}
(1) & \mathrm{min}(x, 0) & \to & 0 \\
(2) & \mathrm{min}(0, y) & \to & 0 \\
(3) & \mathrm{min}(\mathrm{s}(x), \mathrm{s}(y)) & \to & \mathrm{s}(\mathrm{min}(x, y)) \\
(4) & \mathrm{max}(x, 0) & \to & x \\
(5) & \mathrm{max}(0, y) & \to & y \\
(6) & \mathrm{max}(\mathrm{s}(x), \mathrm{s}(y)) & \to & \mathrm{s}(\mathrm{max}(x, y)) \\
(7) & x - 0 & \to & x \\
(8) & \mathrm{s}(x) - \mathrm{s}(y) & \to & x - y \\
(9) & \mathrm{gcd}(\mathrm{s}(x), \mathrm{s}(y)) & \to & \mathrm{gcd}(\mathrm{s}(\mathrm{max}(x, y)) - \mathrm{s}(\mathrm{min}(x, y)), \\
 & & & \mathrm{s}(\mathrm{min}(x, y)))
\end{array}
$$

technische universiteit eindhoven

# Semantic labelling - introduction

And how about this one?

## Example (GCD)

$$
\begin{array}{lrcl}
(1) & \mathsf{min}(x, 0) & \to & 0 \\
(2) & \mathsf{min}(0, y) & \to & 0 \\
(3) & \mathsf{min}(\mathsf{s}(x), \mathsf{s}(y)) & \to & \mathsf{s}(\mathsf{min}(x, y)) \\
(4) & \mathsf{max}(x, 0) & \to & x \\
(5) & \mathsf{max}(0, y) & \to & y \\
(6) & \mathsf{max}(\mathsf{s}(x), \mathsf{s}(y)) & \to & \mathsf{s}(\mathsf{max}(x, y)) \\
(7) & x - 0 & \to & x \\
(8) & \mathsf{s}(x) - \mathsf{s}(y) & \to & x - y \\
(9) & \mathsf{gcd}(\mathsf{s}(x), \mathsf{s}(y)) & \to & \mathsf{gcd}(\mathsf{s}(\mathsf{max}(x, y)) - \mathsf{s}(\mathsf{min}(x, y)), \\
 & & & \mathsf{s}(\mathsf{min}(x, y)))
\end{array}
$$

# GCD: interpretation

**Example (GCD)**

$$
\begin{array}{lrcl}
(1) & \mathsf{min}(x, 0) & \to & 0 \\
(2) & \mathsf{min}(0, y) & \to & 0 \\
(3) & \mathsf{min}(\mathsf{s}(x), \mathsf{s}(y)) & \to & \mathsf{s}(\mathsf{min}(x, y)) \\
(4) & \mathsf{max}(x, 0) & \to & x \\
(5) & \mathsf{max}(0, y) & \to & y \\
(6) & \mathsf{max}(\mathsf{s}(x), \mathsf{s}(y)) & \to & \mathsf{s}(\mathsf{max}(x, y)) \\
(7) & x - 0 & \to & x \\
(8) & \mathsf{s}(x) - \mathsf{s}(y) & \to & x - y \\
(9) & \mathsf{gcd}(\mathsf{s}(x), \mathsf{s}(y)) & \to & \mathsf{gcd}(\mathsf{s}(\mathsf{max}(x, y)) - \mathsf{s}(\mathsf{min}(x, y)), \mathsf{s}(\mathsf{min}(x, y)))
\end{array}
$$

Natural interpretation for function symbols:

**Example (Interpretation for GCD)**

$$
\begin{array}{rcl}
[0] & = & 0 \\
[\mathsf{s}(x)] & = & x + 1 \\
[x - y] & = & x - y
\end{array}
\qquad
\begin{array}{rcl}
[\mathsf{min}(x, y)] & = & \mathsf{min}(x, y) \\
[\mathsf{max}(x, y)] & = & \mathsf{max}(x, y) \\
[\mathsf{gcd}(x, y)] & = & \mathsf{gcd}(x, y)
\end{array}
$$

# GCD: labelled system

## Example (Labelled GCD)

$$
\begin{array}{rcll}
\mathrm{min}_{i,0}(x,0) & \to & 0 & \\
\mathrm{min}_{0,j}(0,y) & \to & 0 & \\
\mathrm{min}_{i+1,j+1}(\mathrm{s}_i(x),\mathrm{s}_j(y)) & \to & \mathrm{s}_j(\mathrm{min}_{i,j}(x,y)) & \text{if } i >= j \\
\mathrm{min}_{i+1,j+1}(\mathrm{s}_i(x),\mathrm{s}_j(y)) & \to & \mathrm{s}_i(\mathrm{min}_{i,j}(x,y)) & \text{if } i < j \\
\mathrm{max}_{i,0}(x,0) & \to & x & \\
\mathrm{max}_{0,j}(0,y) & \to & y & \\
\mathrm{max}_{i+1,j+1}(\mathrm{s}_i(x),\mathrm{s}_j(y)) & \to & \mathrm{s}_i(\mathrm{max}_{i,j}(x,y)) & \text{if } i >= j \\
\mathrm{max}_{i+1,j+1}(\mathrm{s}_i(x),\mathrm{s}_j(y)) & \to & \mathrm{s}_j(\mathrm{max}_{i,j}(x,y)) & \text{if } i < j \\
x -_{i,0} 0 & \to & x & \\
\mathrm{s}_i(x) -_{i+1,j+1} \mathrm{s}_j(y)) & \to & x -_{i,j} y & \\
\mathrm{gcd}_{i+1,j+1}(\mathrm{s}_i(x),\mathrm{s}_j(y)) & \to & \mathrm{gcd}_{i-j,j+1}(\mathrm{s}_i(\mathrm{max}_{i,j}(x,y)) -_{i+1,j+1} & \text{if } i >= j \\
 & & \mathrm{s}_j(\mathrm{min}_{i,j}(x,y)), \mathrm{min}_{i,j}(x,y))) & \\
\mathrm{gcd}_{i+1,j+1}(\mathrm{s}_i(x),\mathrm{s}_j(y)) & \to & \mathrm{gcd}_{j-i,i+1}(\mathrm{s}_j(\mathrm{max}_{i,j}(x,y)) -_{j+1,i+1} & \text{if } i < j \\
 & & \mathrm{s}_i(\mathrm{min}_{i,j}(x,y)), \mathrm{min}_{i,j}(x,y))) &
\end{array}
$$

# Semantic labelling in a nutshell

- Fix a poset $(A, \geq)$.
- Fix weakly monotonic interpretations for all function symbols: $[f] : A^n \to A$
- such that $\forall \ell \to r \in R, \alpha : \mathcal{V} \to A$:
  - $[\ell, \alpha] \geq [r, \alpha]$ (quasi-model) or
  - $[\ell, \alpha] = [r, \alpha]$ (model).
- Define labelling function as:

$$\begin{aligned} \mathrm{lab}(x, \alpha) &= x, \\ \mathrm{lab}(f(t_1, \ldots, t_n), \alpha) &= f_{[t_1,\alpha],\ldots,[t_n,\alpha]}(\mathrm{lab}(t_1, \alpha), \ldots, \mathrm{lab}(t_n, \alpha)) \end{aligned}$$

- The labelled system $\overline{R}$ is defined as: $\mathrm{lab}(\ell, \alpha) \to \mathrm{lab}(r, \alpha)$, $\forall \alpha : \mathcal{V} \to A, \ell \to r \in R$.
- Prove termination of $\overline{R}$.

# Semantic labelling in a nutshell

- Fix a poset $(A, \geq)$.
- Fix weakly monotonic interpretations for all function symbols: $[f] : A^n \to A$
- such that $\forall \ell \to r \in R, \alpha : \mathcal{V} \to A$:
  - $[\ell, \alpha] \geq [r, \alpha]$ (quasi-model) or
  - $[\ell, \alpha] = [r, \alpha]$ (model).
- Define labelling function as:

$$\begin{aligned} \mathrm{lab}(x, \alpha) &= x, \\ \mathrm{lab}(f(t_1, \ldots, t_n), \alpha) &= f_{[t_1, \alpha], \ldots, [t_n, \alpha]}(\mathrm{lab}(t_1, \alpha), \ldots, \mathrm{lab}(t_n, \alpha)) \end{aligned}$$

- The labelled system $\overline{R}$ is defined as: $\mathrm{lab}(\ell, \alpha) \to \mathrm{lab}(r, \alpha)$, $\forall \alpha : \mathcal{V} \to A, \ell \to r \in R$.
- Prove termination of $\overline{R}$.

# Semantic labelling in a nutshell

- Fix a poset $(A, \geq)$.
- Fix weakly monotonic interpretations for all function symbols: $[f] : A^n \to A$
- such that $\forall \ell \to r \in R, \alpha : \mathcal{V} \to A$:
  - $[\ell, \alpha] \geq [r, \alpha]$ (quasi-model) or
  - $[\ell, \alpha] = [r, \alpha]$ (model).
- Define labelling function as:

$$\begin{aligned} \mathrm{lab}(x, \alpha) &= x, \\ \mathrm{lab}(f(t_1, \ldots, t_n), \alpha) &= f_{[t_1,\alpha],\ldots,[t_n,\alpha]}(\mathrm{lab}(t_1, \alpha), \ldots, \mathrm{lab}(t_n, \alpha)) \end{aligned}$$

- The labelled system $\overline{R}$ is defined as: $\mathrm{lab}(\ell, \alpha) \to \mathrm{lab}(r, \alpha),\ \forall \alpha : \mathcal{V} \to A, \ell \to r \in R$.
- Prove termination of $\overline{R}$.

# Semantic labelling in a nutshell

- Fix a poset $(A, \geq)$.
- Fix weakly monotonic interpretations for all function symbols: $[f] : A^n \to A$
- such that $\forall \ell \to r \in R, \alpha : \mathcal{V} \to A$:
  - $[\ell, \alpha] \geq [r, \alpha]$ (quasi-model) or
  - $[\ell, \alpha] = [r, \alpha]$ (model).
- Define labelling function as:

$$\begin{aligned} \mathrm{lab}(x, \alpha) &= x, \\ \mathrm{lab}(f(t_1, \ldots, t_n), \alpha) &= f_{[t_1,\alpha],\ldots,[t_n,\alpha]}(\mathrm{lab}(t_1, \alpha), \ldots, \mathrm{lab}(t_n, \alpha)) \end{aligned}$$

- The labelled system $\overline{R}$ is defined as: $\mathrm{lab}(\ell, \alpha) \to \mathrm{lab}(r, \alpha)$, $\forall \alpha : \mathcal{V} \to A, \ell \to r \in R$.
- Prove termination of $\overline{R}$.

# Semantic labelling in a nutshell

- Fix a poset $(A, \geq)$.
- Fix weakly monotonic interpretations for all function symbols: $[f] : A^n \to A$
- such that $\forall \ell \to r \in R, \alpha : \mathcal{V} \to A$:
  - $[\ell, \alpha] \geq [r, \alpha]$ (quasi-model) or
  - $[\ell, \alpha] = [r, \alpha]$ (model).
- Define labelling function as:

$$\begin{aligned} \mathrm{lab}(x, \alpha) &= x, \\ \mathrm{lab}(f(t_1, \ldots, t_n), \alpha) &= f_{[t_1,\alpha],\ldots,[t_n,\alpha]}(\mathrm{lab}(t_1, \alpha), \ldots, \mathrm{lab}(t_n, \alpha)) \end{aligned}$$

- The labelled system $\overline{R}$ is defined as: $\mathrm{lab}(\ell, \alpha) \to \mathrm{lab}(r, \alpha)$, $\forall \alpha : \mathcal{V} \to A, \ell \to r \in R$.
- Prove termination of $\overline{R}$.

# Semantic labelling in a nutshell

- Fix a poset $(A, \geq)$.
- Fix weakly monotonic interpretations for all function symbols: $[f] : A^n \to A$
- such that $\forall \ell \to r \in R, \alpha : \mathcal{V} \to A$:
  - $[\ell, \alpha] \geq [r, \alpha]$ (quasi-model) or
  - $[\ell, \alpha] = [r, \alpha]$ (model).
- Define labelling function as:

$$\begin{aligned} \mathrm{lab}(x, \alpha) &= x, \\ \mathrm{lab}(f(t_1, \ldots, t_n), \alpha) &= f_{[t_1,\alpha],\ldots,[t_n,\alpha]}(\mathrm{lab}(t_1, \alpha), \ldots, \mathrm{lab}(t_n, \alpha)) \end{aligned}$$

- The labelled system $\overline{R}$ is defined as: $\mathrm{lab}(\ell, \alpha) \to \mathrm{lab}(r, \alpha)$, $\forall \alpha : \mathcal{V} \to A, \ell \to r \in R$.
- Prove termination of $\overline{R}$.

# Semantic labelling in a nutshell

- Fix a poset $(A, \geq)$.
- Fix weakly monotonic interpretations for all function symbols: $[f] : A^n \to A$
- such that $\forall \ell \to r \in R, \alpha : \mathcal{V} \to A$:
  - $[\ell, \alpha] \geq [r, \alpha]$ (quasi-model) or
  - $[\ell, \alpha] = [r, \alpha]$ (model).
- Define labelling function as:

$$\begin{aligned} \mathrm{lab}(x, \alpha) &= x, \\ \mathrm{lab}(f(t_1, \ldots, t_n), \alpha) &= f_{[t_1,\alpha],\ldots,[t_n,\alpha]}(\mathrm{lab}(t_1, \alpha), \ldots, \mathrm{lab}(t_n, \alpha)) \end{aligned}$$

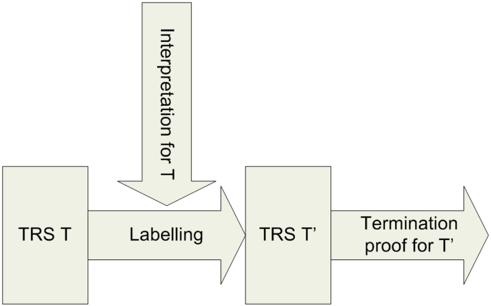
- The labelled system $\overline{R}$ is defined as: $\mathrm{lab}(\ell, \alpha) \to \mathrm{lab}(r, \alpha)$, $\forall \alpha : \mathcal{V} \to A, \ell \to r \in R$.
- Prove termination of $\overline{R}$.

# Semantic labelling in a nutshell

- Fix a poset $(A, \geq)$.
- Fix weakly monotonic interpretations for all function symbols: $[f] : A^n \to A$
- such that $\forall \ell \to r \in R, \alpha : \mathcal{V} \to A$:
  - $[\ell, \alpha] \geq [r, \alpha]$ (quasi-model) or
  - $[\ell, \alpha] = [r, \alpha]$ (model).
- Define labelling function as:

$$\begin{aligned} \mathrm{lab}(x, \alpha) &= x, \\ \mathrm{lab}(f(t_1, \ldots, t_n), \alpha) &= f_{[t_1,\alpha],\ldots,[t_n,\alpha]}(\mathrm{lab}(t_1, \alpha), \ldots, \mathrm{lab}(t_n, \alpha)) \end{aligned}$$

- The labelled system $\overline{R}$ is defined as: $\mathrm{lab}(\ell, \alpha) \to \mathrm{lab}(r, \alpha),\ \forall \alpha : \mathcal{V} \to A, \ell \to r \in R$.
- Prove termination of $\overline{R}$.




**Theorem**

*Given interpretation that is a model R is terminating iff $\overline{R}$ is terminating.*

# Semantic labelling with natural numbers - challenges

So far semantic labelling was used with $A = \{0, 1\}$.
What if we want to use it with $A = \mathbb{N}$?

- Labelled system is infinite (infinitely many rules, infinite signature)
- We need to be able to represent and manipulate it.
- This requires adoption of existing methods.
- We may also want to use non-polynomial functions min and max as interpretations.
- This requires performing case-analysis
- and complicates the corresponding polynomial comparison problem.

# Semantic labelling with natural numbers - challenges

So far semantic labelling was used with $A = \{0, 1\}$.
What if we want to use it with $A = \mathbb{N}$?

- Labelled system is infinite (infinitely many rules, infinite signature)
- We need to be able to represent and manipulate it.
- This requires adoption of existing methods.
- We may also want to use non-polynomial functions min and max as interpretations.
- This requires performing case-analysis
- and complicates the corresponding polynomial comparison problem.

# Semantic labelling with natural numbers - challenges

So far semantic labelling was used with $A = \{0, 1\}$.
What if we want to use it with $A = \mathbb{N}$?

- Labelled system is infinite (infinitely many rules, infinite signature)
- We need to be able to represent and manipulate it.
- This requires adoption of existing methods.
- We may also want to use non-polynomial functions min and max as interpretations.
- This requires performing case-analysis
- and complicates the corresponding polynomial comparison problem.

# Semantic labelling with natural numbers - challenges

So far semantic labelling was used with $A = \{0, 1\}$.
What if we want to use it with $A = \mathbb{N}$?

- Labelled system is infinite (infinitely many rules, infinite signature)
- We need to be able to represent and manipulate it.
- This requires adoption of existing methods.
- We may also want to use non-polynomial functions min and max as interpretations.
- This requires performing case-analysis
- and complicates the corresponding polynomial comparison problem.

# Semantic labelling with natural numbers - challenges

So far semantic labelling was used with $A = \{0, 1\}$.
What if we want to use it with $A = \mathbb{N}$?

- Labelled system is infinite (infinitely many rules, infinite signature)
- We need to be able to represent and manipulate it.
- This requires adoption of existing methods.
- We may also want to use non-polynomial functions min and max as interpretations.
- This requires performing case-analysis
- and complicates the corresponding polynomial comparison problem.

# Semantic labelling with natural numbers - challenges

So far semantic labelling was used with $A = \{0, 1\}$.
What if we want to use it with $A = \mathbb{N}$?

- Labelled system is infinite (infinitely many rules, infinite signature)
- We need to be able to represent and manipulate it.
- This requires adoption of existing methods.
- We may also want to use non-polynomial functions min and max as interpretations.
- This requires performing case-analysis
- and complicates the corresponding polynomial comparison problem.

# Semantic labelling with natural numbers - challenges

So far semantic labelling was used with $A = \{0, 1\}$.
What if we want to use it with $A = \mathbb{N}$?

- Labelled system is infinite (infinitely many rules, infinite signature)
- We need to be able to represent and manipulate it.
- This requires adoption of existing methods.
- We may also want to use non-polynomial functions min and max as interpretations.
- This requires performing case-analysis
- and complicates the corresponding polynomial comparison problem.

# Semantic labelling with natural numbers - motivation

**Example (SUBST)**

$$
\begin{array}{clcl}
(1) & \lambda(x) \circ y & \rightarrow & \lambda(x \circ (1 \cdot (y \circ \uparrow))) \\
(2) & (x \cdot y) \circ z & \rightarrow & (x \circ z) \cdot (y \circ z) \\
(3) & (x \circ y) \circ z & \rightarrow & x \circ (y \circ z) \\
(4) & \mathrm{id} \circ x & \rightarrow & x \\
(5) & 1 \circ \mathrm{id} & \rightarrow & 1 \\
(6) & \uparrow \circ\, \mathrm{id} & \rightarrow & \uparrow \\
(7) & 1 \circ (x \cdot y) & \rightarrow & x \\
(8) & \uparrow \circ\, (x \cdot y) & \rightarrow & y
\end{array}
$$

- This system describes the process of substitution in combinatory categorical logic.
- Showing its termination is difficult...
- ... but can be accomplished relatively easily using semantic labelling with natural numbers.

# Semantic labelling with natural numbers - motivation

## Example (SUBST)

$$
\begin{array}{llcl}
(1) & \lambda(x) \circ y & \longrightarrow & \lambda(x \circ (1 \cdot (y \circ \uparrow))) \\
(2) & (x \cdot y) \circ z & \longrightarrow & (x \circ z) \cdot (y \circ z) \\
(3) & (x \circ y) \circ z & \longrightarrow & x \circ (y \circ z) \\
(4) & \mathrm{id} \circ x & \longrightarrow & x \\
(5) & 1 \circ \mathrm{id} & \longrightarrow & 1 \\
(6) & \uparrow \circ\, \mathrm{id} & \longrightarrow & \uparrow \\
(7) & 1 \circ (x \cdot y) & \longrightarrow & x \\
(8) & \uparrow \circ\, (x \cdot y) & \longrightarrow & y
\end{array}
$$

- This system describes the process of substitution in combinatory categorical logic.
- Showing its termination is difficult...
- ... but can be accomplished relatively easily using semantic labelling with natural numbers.

# Semantic labelling with natural numbers - motivation

## Example (SUBST)

$$
\begin{array}{lrcl}
(1) & \lambda(x) \circ y & \to & \lambda(x \circ (1 \cdot (y \circ \uparrow))) \\
(2) & (x \cdot y) \circ z & \to & (x \circ z) \cdot (y \circ z) \\
(3) & (x \circ y) \circ z & \to & x \circ (y \circ z) \\
(4) & \mathrm{id} \circ x & \to & x \\
(5) & 1 \circ \mathrm{id} & \to & 1 \\
(6) & \uparrow \circ\, \mathrm{id} & \to & \uparrow \\
(7) & 1 \circ (x \cdot y) & \to & x \\
(8) & \uparrow \circ\, (x \cdot y) & \to & y
\end{array}
$$

- This system describes the process of substitution in combinatory categorical logic.
- Showing its termination is difficult...
- ... but can be accomplished relatively easily using semantic labelling with natural numbers.

# Semantic labelling with natural numbers - motivation

## Example (SUBST)

$$
\begin{array}{lrcl}
(1) & \lambda(x) \circ y & \to & \lambda(x \circ (1 \cdot (y \circ \uparrow))) \\
(2) & (x \cdot y) \circ z & \to & (x \circ z) \cdot (y \circ z) \\
(3) & (x \circ y) \circ z & \to & x \circ (y \circ z) \\
(4) & \mathrm{id} \circ x & \to & x \\
(5) & 1 \circ \mathrm{id} & \to & 1 \\
(6) & \uparrow \circ\, \mathrm{id} & \to & \uparrow \\
(7) & 1 \circ (x \cdot y) & \to & x \\
(8) & \uparrow \circ\, (x \cdot y) & \to & y
\end{array}
$$

- This system describes the process of substitution in combinatory categorical logic.
- Showing its termination is difficult...
- ... but can be accomplished relatively easily using semantic labelling with natural numbers.

# Outline

1. Project overview
2. Recursive path ordering for infinite labelled systems
   - Semantic labelling
   - Recursive path ordering (RPO)
   - RPO for infinite systems
   - Automation of RPO for infinite systems

# RPO - definition

## Definition

Given well-founded ordering on function symbols $\succ$, called a precedence and a status function $\tau$ we define RPO as:

$$s = f(s_1, \ldots, s_n) \succ_{RPO} t \iff$$

(1) $\exists_{1 \leq i \leq n} \,.\, s_i \succeq_{RPO} t$, or

(2) $t = g(t_1, \ldots, t_m)$, $f \succ g$ and $\forall_{1 \leq i \leq m} \,.\, s \succ_{RPO} t_i$, or

(3) $t = g(t_1, \ldots, t_m)$, $f = g$, $\forall_{1 \leq i \leq m} \,.\, s \succ_{RPO} t_i$ and $\langle s_1, \ldots, s_n \rangle \succ_{RPO}^{\tau(f)} \langle t_1, \ldots, t_m \rangle$.

## Theorem

*If $\ell \succ_{RPO} r$ for every rule $\ell \to r$ of a TRS $R$, then $R$ is terminating.*

# RPO - definition

## Definition

Given well-founded ordering on function symbols $\succ$, called a precedence and a status function $\tau$ we define RPO as:

$s = f(s_1, \ldots, s_n) \succ_{RPO} t \iff$

(1) $\exists_{1 \leq i \leq n} \, . \, s_i \succeq_{RPO} t$, or

(2) $t = g(t_1, \ldots, t_m)$, $f \succ g$ and $\forall_{1 \leq i \leq m} \, . \, s \succ_{RPO} t_i$, or

(3) $t = g(t_1, \ldots, t_m)$, $f = g$, $\forall_{1 \leq i \leq m} \, . \, s \succ_{RPO} t_i$ and $\langle s_1, \ldots, s_n \rangle \succ_{RPO}^{\tau(f)} \langle t_1, \ldots, t_m \rangle$.

## Theorem

*If $\ell \succ_{RPO} r$ for every rule $\ell \to r$ of a TRS $R$, then $R$ is terminating.*

# RPO in a nutshell

- Orient every rule of a given TRS with RPO (choices).
- While doing so collect requirements on a precedence $\succ$.
- Check whether $\succ$ is well-founded.

Checking whether $\succ$ is well-founded coincides with checking whether the corresponding graph is acyclic.

# RPO in a nutshell

- Orient every rule of a given TRS with RPO (choices).
- While doing so collect requirements on a precedence $\succ$.
- Check whether $\succ$ is well-founded.

Checking whether $\succ$ is well-founded coincides with checking whether the corresponding graph is acyclic.

# RPO in a nutshell

- Orient every rule of a given TRS with RPO (choices).
- While doing so collect requirements on a precedence $\succ$.
- Check whether $\succ$ is well-founded.

Checking whether $\succ$ is well-founded coincides with checking whether the corresponding graph is acyclic.

# RPO in a nutshell

- Orient every rule of a given TRS with RPO (choices).
- While doing so collect requirements on a precedence $\succ$.
- Check whether $\succ$ is well-founded.

Checking whether $\succ$ is well-founded coincides with checking whether the corresponding graph is acyclic.

# Example of RPO proof

**Example (Plus)**

$$\begin{array}{rcl} 0 + y & \to & y \\ s(x) + y & \to & s(x + y) \end{array}$$

**Definition**

$s = f(s_1, \ldots, s_n) \succ_{RPO} t \iff$
(1) $\exists_{1 \le i \le n} \, . \, s_i \succeq_{RPO} t$, or
(2) $t = g(t_1, \ldots, t_m)$, $f \succ g$ and $\forall_{1 \le i \le m} \, . \, s \succ_{RPO} t_i$, or
(3) $t = g(t_1, \ldots, t_m)$, $f = g$, $\forall_{1 \le i \le m} \, . \, s \succ_{RPO} t_i$ and $\langle s_1, \ldots, s_n \rangle \succ_{RPO}^{\tau(f)} \langle t_1, \ldots, t_m \rangle$.

# Example of RPO proof

## Example (Plus)

$$\begin{array}{rcl} 0 + y & \to & y \\ 0 + y & \succ_{RPO} & y \\ \mathsf{s}(x) + y & \to & \mathsf{s}(x + y) \end{array}$$

## Definition

$s = f(s_1, \ldots, s_n) \succ_{RPO} t \iff$

(1) $\exists_{1 \le i \le n} \,.\, s_i \succeq_{RPO} t$, or

(2) $t = g(t_1, \ldots, t_m)$, $f \succ g$ and $\forall_{1 \le i \le m} \,.\, s \succ_{RPO} t_i$, or

(3) $t = g(t_1, \ldots, t_m)$, $f = g$, $\forall_{1 \le i \le m} \,.\, s \succ_{RPO} t_i$ and $\langle s_1, \ldots, s_n \rangle \succ_{RPO}^{\tau(f)} \langle t_1, \ldots, t_m \rangle$.

# Example of RPO proof

## Example (Plus)

$$
\begin{array}{rcll}
0 + y & \to & y & \\
0 + y & \succ_{RPO} & y & \\
s(x) + y & \to & s(x+y) & \\
s(x) + y & \succ_{RPO} & s(x+y) & (+ \succ s)
\end{array}
$$

## Definition

$s = f(s_1, \ldots, s_n) \succ_{RPO} t \iff$

(1) $\exists_{1 \le i \le n} \,.\, s_i \succeq_{RPO} t$, or

(2) $t = g(t_1, \ldots, t_m)$, $f \succ g$ and $\forall_{1 \le i \le m} \,.\, s \succ_{RPO} t_i$, or

(3) $t = g(t_1, \ldots, t_m)$, $f = g$, $\forall_{1 \le i \le m} \,.\, s \succ_{RPO} t_i$ and $\langle s_1, \ldots, s_n \rangle \succ_{RPO}^{\tau(f)} \langle t_1, \ldots, t_m \rangle$.

technische universiteit eindhoven

# Example of RPO proof

## Example (Plus)

$$
\begin{array}{rcll}
0 + y & \to & y & \\
0 + y & \succ_{RPO} & y & \\
s(x) + y & \to & s(x + y) & \\
s(x) + y & \succ_{RPO} & s(x + y) & (+ \succ s) \\
s(x) + y & \succ_{RPO} & x + y & (\tau(+) = Lex_{LR})
\end{array}
$$

## Definition

$s = f(s_1, \ldots, s_n) \succ_{RPO} t \iff$

(1) $\exists_{1 \le i \le n} \,.\, s_i \succeq_{RPO} t$, or

(2) $t = g(t_1, \ldots, t_m)$, $f \succ g$ and $\forall_{1 \le i \le m} \,.\, s \succ_{RPO} t_i$, or

(3) $t = g(t_1, \ldots, t_m)$, $f = g$, $\forall_{1 \le i \le m} \,.\, s \succ_{RPO} t_i$ and $\langle s_1, \ldots, s_n \rangle \succ_{RPO}^{\tau(f)} \langle t_1, \ldots, t_m \rangle$.

# Example of RPO proof

## Example (Plus)

$$
\begin{array}{rcll}
0 + y & \to & y & \\
0 + y & \succ_{RPO} & y & \\
s(x) + y & \to & s(x+y) & \\
s(x) + y & \succ_{RPO} & s(x+y) & (+ \succ s) \\
s(x) + y & \succ_{RPO} & x + y & (\tau(+) = Lex_{LR}) \\
s(x) & \succ_{RPO} & x &
\end{array}
$$

## Definition

$s = f(s_1, \ldots, s_n) \succ_{RPO} t \iff$

(1) $\exists_{1 \le i \le n} \,.\, s_i \succeq_{RPO} t$, or

(2) $t = g(t_1, \ldots, t_m)$, $f \succ g$ and $\forall_{1 \le i \le m} \,.\, s \succ_{RPO} t_i$, or

(3) $t = g(t_1, \ldots, t_m)$, $f = g$, $\forall_{1 \le i \le m} \,.\, s \succ_{RPO} t_i$ and $\langle s_1, \ldots, s_n \rangle \succ_{RPO}^{\tau(f)} \langle t_1, \ldots, t_m \rangle$.

# Outline

1. Project overview

2. Recursive path ordering for infinite labelled systems
    - Semantic labelling
    - Recursive path ordering (RPO)
    - RPO for infinite systems
    - Automation of RPO for infinite systems

# In the remainder we fix $A = \mathbb{N}$.

We need to restrict to precedences such that:

- they can be captured in finite description,
- are "powerful",
- efficient checking for well-foundedness is possible.

**Definition (Precedence description)**

A *precedence description* consists of:

- for every $f \in \Sigma$ of arity $n$, $\phi_f : \mathbb{N}^n \to \mathbb{N}$ and
- $\text{pd} : \Sigma \times \Sigma \to \{\bot, >, \geq, \top\}$.

This gives rise to a precedence $\succ$: $f_{k_1,\ldots,k_n} \succ g_{l_1,\ldots,l_m}$ iff:

$$\begin{array}{l} \text{pd}(f, g) = \top \vee \\ (\text{pd}(f, g) = \geq \ \wedge \phi_f(k_1, \ldots, k_n) \geq \phi_g(l_1, \ldots, l_m)) \vee \\ (\text{pd}(f, g) = > \ \wedge \phi_f(k_1, \ldots, k_n) > \phi_g(l_1, \ldots, l_m)). \end{array}$$

# In the remainder we fix $A = \mathbb{N}$.

We need to restrict to precedences such that:

- they can be captured in finite description,
- are "powerful",
- efficient checking for well-foundedness is possible.

**Definition (Precedence description)**

A precedence description consists of:

- for every $f \in \Sigma$ of arity $n$, $\phi_f : \mathbb{N}^n \to \mathbb{N}$ and
- $\mathrm{pd} : \Sigma \times \Sigma \to \{\perp, >, \geq, \top\}$.

This gives rise to a precedence $\succ$: $f_{k_1,\ldots,k_n} \succ g_{l_1,\ldots,l_m}$ iff:

$$\begin{aligned}
&\mathrm{pd}(f,g) = \top \lor \\
&(\mathrm{pd}(f,g) = \geq \ \land \phi_f(k_1,\ldots,k_n) \geq \phi_g(l_1,\ldots,l_m)) \lor \\
&(\mathrm{pd}(f,g) = > \ \land \phi_f(k_1,\ldots,k_n) > \phi_g(l_1,\ldots,l_m)).
\end{aligned}$$

# In the remainder we fix $A = \mathbb{N}$.

We need to restrict to precedences such that:

- they can be captured in finite description,
- are "powerful",
- efficient checking for well-foundedness is possible.

**Definition (Precedence description)**

A precedence description consists of:

- for every $f \in \Sigma$ of arity $n$, $\phi_f : \mathbb{N}^n \to \mathbb{N}$ and
- $\mathrm{pd} : \Sigma \times \Sigma \to \{\bot, >, \geq, \top\}$.

This gives rise to a precedence $\succ$: $f_{k_1,\ldots,k_n} \succ g_{l_1,\ldots,l_m}$ iff:

$$\begin{aligned} &\mathrm{pd}(f,g) = \top \lor \\ &(\mathrm{pd}(f,g) = \geq \ \land \phi_f(k_1,\ldots,k_n) \geq \phi_g(l_1,\ldots,l_m)) \lor \\ &(\mathrm{pd}(f,g) = > \ \land \phi_f(k_1,\ldots,k_n) > \phi_g(l_1,\ldots,l_m)). \end{aligned}$$

# In the remainder we fix $A = \mathbb{N}$.

We need to restrict to precedences such that:

- they can be captured in finite description,
- are "powerful",
- efficient checking for well-foundedness is possible.

**Definition (Precedence description)**

A precedence description consists of:

- for every $f \in \Sigma$ of arity $n$, $\phi_f : \mathbb{N}^n \to \mathbb{N}$ and
- $\mathrm{pd} : \Sigma \times \Sigma \to \{\perp, >, \geq, \top\}$.

This gives rise to a precedence $\succ$: $f_{k_1,\ldots,k_n} \succ g_{l_1,\ldots,l_m}$ iff:

$$\begin{aligned} &\mathrm{pd}(f,g) = \top \,\vee \\ &(\mathrm{pd}(f,g) = \geq\ \wedge \phi_f(k_1,\ldots,k_n) \geq \phi_g(l_1,\ldots,l_m)) \,\vee \\ &(\mathrm{pd}(f,g) = >\ \wedge \phi_f(k_1,\ldots,k_n) > \phi_g(l_1,\ldots,l_m)). \end{aligned}$$

# In the remainder we fix $A = \mathbb{N}$.

We need to restrict to precedences such that:

- they can be captured in finite description,
- are "powerful",
- efficient checking for well-foundedness is possible.

**Definition (Precedence description)**

A precedence description consists of:

- for every $f \in \Sigma$ of arity $n$, $\phi_f : \mathbb{N}^n \to \mathbb{N}$ and
- $\mathrm{pd} : \Sigma \times \Sigma \to \{\bot, >, \geq, \top\}$.

This gives rise to a precedence $\succ$: $f_{k_1,\ldots,k_n} \succ g_{l_1,\ldots,l_m}$ iff:

$$\begin{gathered} \mathrm{pd}(f,g) = \top \vee \\ (\mathrm{pd}(f,g) = \geq \ \wedge \phi_f(k_1,\ldots,k_n) \geq \phi_g(l_1,\ldots,l_m)) \vee \\ (\mathrm{pd}(f,g) = > \ \wedge \phi_f(k_1,\ldots,k_n) > \phi_g(l_1,\ldots,l_m)). \end{gathered}$$

technische universiteit eindhoven

In the remainder we fix $A = \mathbb{N}$.

We need to restrict to precedences such that:

- they can be captured in finite description,
- are "powerful",
- efficient checking for well-foundedness is possible.

## Definition (Precedence description)

A precedence description consists of:

- for every $f \in \Sigma$ of arity $n$, $\phi_f : \mathbb{N}^n \to \mathbb{N}$ and
- $\text{pd} : \Sigma \times \Sigma \to \{\bot, >, \geq, \top\}$.

This gives rise to a precedence $\succ$: $f_{k_1,\ldots,k_n} \succ g_{l_1,\ldots,l_m}$ iff:

$$\begin{gathered}\text{pd}(f,g) = \top \vee \\ (\text{pd}(f,g) = \geq \ \wedge \phi_f(k_1,\ldots,k_n) \geq \phi_g(l_1,\ldots,l_m)) \vee \\ (\text{pd}(f,g) = > \ \wedge \phi_f(k_1,\ldots,k_n) > \phi_g(l_1,\ldots,l_m)).\end{gathered}$$

technische universiteit eindhoven

In the remainder we fix $A = \mathbb{N}$.

We need to restrict to precedences such that:

- they can be captured in finite description,
- are "powerful",
- efficient checking for well-foundedness is possible.

## Definition (Precedence description)

A precedence description consists of:

- for every $f \in \Sigma$ of arity $n$, $\phi_f : \mathbb{N}^n \to \mathbb{N}$ and
- $\mathrm{pd} : \Sigma \times \Sigma \to \{\perp, >, \geq, \top\}$.

This gives rise to a precedence $\succ$: $f_{k_1,\ldots,k_n} \succ g_{l_1,\ldots,l_m}$ iff:

$$\begin{aligned}
&\mathrm{pd}(f,g) = \top \vee \\
&(\mathrm{pd}(f,g) = \geq \ \wedge \phi_f(k_1,\ldots,k_n) \geq \phi_g(l_1,\ldots,l_m)) \vee \\
&(\mathrm{pd}(f,g) = > \ \wedge \phi_f(k_1,\ldots,k_n) > \phi_g(l_1,\ldots,l_m)).
\end{aligned}$$

## In the remainder we fix $A = \mathbb{N}$.

We need to restrict to precedences such that:

- they can be captured in finite description,
- are "powerful",
- efficient checking for well-foundedness is possible.

### Definition (Precedence description)

A precedence description consists of:

- for every $f \in \Sigma$ of arity $n$, $\phi_f : \mathbb{N}^n \to \mathbb{N}$ and
- $\mathrm{pd} : \Sigma \times \Sigma \to \{\perp, >, \geq, \top\}$.

This gives rise to a precedence $\succ$: $f_{k_1,\ldots,k_n} \succ g_{l_1,\ldots,l_m}$ iff:

$$\begin{array}{l} \mathrm{pd}(f,g) = \top \vee \\ (\mathrm{pd}(f,g) = \geq \ \wedge \phi_f(k_1,\ldots,k_n) \geq \phi_g(l_1,\ldots,l_m)) \vee \\ (\mathrm{pd}(f,g) = > \ \wedge \phi_f(k_1,\ldots,k_n) > \phi_g(l_1,\ldots,l_m)). \end{array}$$

## In the remainder we fix $A = \mathbb{N}$.

We need to restrict to precedences such that:

- they can be captured in finite description,
- are "powerful",
- efficient checking for well-foundedness is possible.

### Definition (Precedence description)

A precedence description consists of:

- for every $f \in \Sigma$ of arity $n$, $\phi_f : \mathbb{N}^n \to \mathbb{N}$ and
- $\mathrm{pd} : \Sigma \times \Sigma \to \{\bot, >, \geq, \top\}$.

This gives rise to a precedence $\succ$: $f_{k_1,\ldots,k_n} \succ g_{l_1,\ldots,l_m}$ iff:

$$\begin{array}{l} \mathrm{pd}(f,g) = \top \vee \\ (\mathrm{pd}(f,g) = \geq \ \wedge \phi_f(k_1,\ldots,k_n) \geq \phi_g(l_1,\ldots,l_m)) \vee \\ (\mathrm{pd}(f,g) = > \ \wedge \phi_f(k_1,\ldots,k_n) > \phi_g(l_1,\ldots,l_m)). \end{array}$$

## Example

Let us consider $\phi$ functions to be:

- Always identity (id) for unary symbols,
- For binary symbols one of the following: summation ($+$), left projection ($\pi_1$) or right projection ($\pi_2$).

Then to have $f_{i+1,j+1} \succ g_{i+2,j}$ one of the following precedence descriptions is possible:

- $\mathrm{pd}(f, g) = \top$,
- $\phi(f) = +,\ \phi(g) = +,\ \mathrm{pd}(f, g) = \geq,$,
- $\phi(f) = +,\ \phi(g) = \pi_1,\ \mathrm{pd}(f, g) = \geq$,
- $\phi(f) = +,\ \phi(g) = \pi_2,\ \mathrm{pd}(f, g) = > / \geq$,
- $\phi(f) = \pi_2,\ \phi(g) = \pi_2,\ \mathrm{pd}(f, g) = > / \geq$.

## Example

Let us consider $\phi$ functions to be:

- Always identity (id) for unary symbols,
- For binary symbols one of the following: summation ($+$), left projection ($\pi_1$) or right projection ($\pi_2$).

Then to have $f_{i+1,j+1} \succ g_{i+2,j}$ one of the following precedence descriptions is possible:

- $\mathrm{pd}(f,g) = \top$,
- $\phi(f) = +$, $\phi(g) = +$, $\mathrm{pd}(f,g) = \geq$,,
- $\phi(f) = +$, $\phi(g) = \pi_1$, $\mathrm{pd}(f,g) = \geq$,
- $\phi(f) = +$, $\phi(g) = \pi_2$, $\mathrm{pd}(f,g) = > / \geq$,
- $\phi(f) = \pi_2$, $\phi(g) = \pi_2$, $\mathrm{pd}(f,g) = > / \geq$.

## Example

Let us consider $\phi$ functions to be:

- Always identity (id) for unary symbols,
- For binary symbols one of the following: summation ($+$), left projection ($\pi_1$) or right projection ($\pi_2$).

Then to have $f_{i+1,j+1} \succ g_{i+2,j}$ one of the following precedence descriptions is possible:

- $\mathrm{pd}(f, g) = \top$,
- $\phi(f) = +,\ \phi(g) = +,\ \mathrm{pd}(f, g) = \geq,$,
- $\phi(f) = +,\ \phi(g) = \pi_1,\ \mathrm{pd}(f, g) = \geq,$
- $\phi(f) = +,\ \phi(g) = \pi_2,\ \mathrm{pd}(f, g) = > / \geq,$
- $\phi(f) = \pi_2,\ \phi(g) = \pi_2,\ \mathrm{pd}(f, g) = > / \geq.$

## Example

Let us consider $\phi$ functions to be:

- Always identity (id) for unary symbols,
- For binary symbols one of the following: summation ($+$), left projection ($\pi_1$) or right projection ($\pi_2$).

Then to have $f_{i+1,j+1} \succ g_{i+2,j}$ one of the following precedence descriptions is possible:

- $\mathrm{pd}(f, g) = \top$,
- $\phi(f) = +,\ \phi(g) = +,\ \mathrm{pd}(f, g) = \geq,$
- $\phi(f) = +,\ \phi(g) = \pi_1,\ \mathrm{pd}(f, g) = \geq,$
- $\phi(f) = +,\ \phi(g) = \pi_2,\ \mathrm{pd}(f, g) = > / \geq,$
- $\phi(f) = \pi_2,\ \phi(g) = \pi_2,\ \mathrm{pd}(f, g) = > / \geq.$

## Example

Let us consider $\phi$ functions to be:

- Always identity (id) for unary symbols,
- For binary symbols one of the following: summation ($+$), left projection ($\pi_1$) or right projection ($\pi_2$).

Then to have $f_{i+1,j+1} \succ g_{i+2,j}$ one of the following precedence descriptions is possible:

- $\mathrm{pd}(f,g) = \top$,
- $\phi(f) = +,\ \phi(g) = +,\ \mathrm{pd}(f,g) = \geq,$,
- $\phi(f) = +,\ \phi(g) = \pi_1,\ \mathrm{pd}(f,g) = \geq,$
- $\phi(f) = +,\ \phi(g) = \pi_2,\ \mathrm{pd}(f,g) = > / \geq,$
- $\phi(f) = \pi_2,\ \phi(g) = \pi_2,\ \mathrm{pd}(f,g) = > / \geq.$

technische universiteit eindhoven

## Example

Let us consider $\phi$ functions to be:

- Always identity (id) for unary symbols,
- For binary symbols one of the following: summation ($+$), left projection ($\pi_1$) or right projection ($\pi_2$).

Then to have $f_{i+1,j+1} \succ g_{i+2,j}$ one of the following precedence descriptions is possible:

- $\mathrm{pd}(f, g) = \top$,
- $\phi(f) = +,\ \phi(g) = +,\ \mathrm{pd}(f, g) = \geq,$,
- $\phi(f) = +,\ \phi(g) = \pi_1,\ \mathrm{pd}(f, g) = \geq,$
- $\phi(f) = +,\ \phi(g) = \pi_2,\ \mathrm{pd}(f, g) = > / \geq,$
- $\phi(f) = \pi_2,\ \phi(g) = \pi_2,\ \mathrm{pd}(f, g) = > / \geq.$

## Example

Let us consider $\phi$ functions to be:

- Always identity (id) for unary symbols,
- For binary symbols one of the following: summation ($+$), left projection ($\pi_1$) or right projection ($\pi_2$).

Then to have $f_{i+1,j+1} \succ g_{i+2,j}$ one of the following precedence descriptions is possible:

- $\mathrm{pd}(f, g) = \top$,
- $\phi(f) = +, \ \phi(g) = +, \ \mathrm{pd}(f, g) = \geq,$,
- $\phi(f) = +, \ \phi(g) = \pi_1, \ \mathrm{pd}(f, g) = \geq,$
- $\phi(f) = +, \ \phi(g) = \pi_2, \ \mathrm{pd}(f, g) = > / \geq,$
- $\phi(f) = \pi_2, \ \phi(g) = \pi_2, \ \mathrm{pd}(f, g) = > / \geq.$

## Definition (Precedence graph)

pd gives rise to a precedence graph having $\Sigma$ as its node set, and having connections between all $f, g \in \Sigma$, $\text{pd}(f, g) \neq \bot$:

| $\text{pd}(f, g)$ | Edge type | Notation |
|---|---|---|
| $\top$ | Unconditional | $\Longrightarrow$ |
| $\geq$ | Non-strict | $\dashrightarrow$ |
| $>$ | Strict | $\longrightarrow$ |

## Theorem

*A precedence description* pd *gives rise to a well-founded precedence* $\succ$ *if every cycle in the corresponding precedence graph:*

(1) *contains no unconditional edge, and*
(2) *contains at least one strict edge.*

## Definition (Precedence graph)

pd gives rise to a precedence graph having $\Sigma$ as its node set, and having connections between all $f, g \in \Sigma$, $\text{pd}(f, g) \neq \bot$:

| $\text{pd}(f, g)$ | Edge type | Notation |
|---|---|---|
| $\top$ | Unconditional | $\Longrightarrow$ |
| $\geq$ | Non-strict | $\dashrightarrow$ |
| $>$ | Strict | $\longrightarrow$ |

## Theorem

*A precedence description* pd *gives rise to a well-founded precedence $\succ$ if every cycle in the corresponding precedence graph:*

(1) *contains no unconditional edge, and*

(2) *contains at least one strict edge.*

# Outline

1. Project overview
2. Recursive path ordering for infinite labelled systems
   - Semantic labelling
   - Recursive path ordering (RPO)
   - RPO for infinite systems
   - Automation of RPO for infinite systems

# Implementing RPO for infinite systems

- Approach: Fix some finite domains for $\phi$ functions.
- Problem: we do not know in advance what $\phi$ function are appropriate for given symbols.
- Solution: let us localize this choice and make it part of the search procedure.

**Definition (Precedence description scheme)**

Precedence description schema is a function:

$$\text{pds}(f, g) \subseteq \{\bot\} \cup (\{>, \geq\} \times \mathbb{N}^{\mathbb{N}^n} \times \mathbb{N}^{\mathbb{N}^m}) \cup \{\top\}$$

# Implementing RPO for infinite systems

- Approach: Fix some finite domains for $\phi$ functions.
- Problem: we do not know in advance what $\phi$ function are appropriate for given symbols.
- Solution: let us localize this choice and make it part of the search procedure.

**Definition (Precedence description scheme)**

Precedence description schema is a function:

$$\mathrm{pds}(f, g) \subseteq \{\bot\} \cup (\{>, \geq\} \times \mathbb{N}^{\mathbb{N}^n} \times \mathbb{N}^{\mathbb{N}^m}) \cup \{\top\}$$

# Implementing RPO for infinite systems

- Approach: Fix some finite domains for $\phi$ functions.
- Problem: we do not know in advance what $\phi$ function are appropriate for given symbols.
- Solution: let us localize this choice and make it part of the search procedure.

**Definition (Precedence description scheme)**

Precedence description schema is a function:

$$\mathrm{pds}(f, g) \subseteq \{\bot\} \cup (\{>, \geq\} \times \mathbb{N}^{\mathbb{N}^n} \times \mathbb{N}^{\mathbb{N}^m}) \cup \{\top\}$$

# Implementing RPO for infinite systems

- Approach: Fix some finite domains for $\phi$ functions.
- Problem: we do not know in advance what $\phi$ function are appropriate for given symbols.
- Solution: let us localize this choice and make it part of the search procedure.

**Definition (Precedence description scheme)**

Precedence description schema is a function:

$$\mathrm{pds}(f, g) \subseteq \{\bot\} \cup (\{>, \geq\} \times \mathbb{N}^{\mathbb{N}^n} \times \mathbb{N}^{\mathbb{N}^m}) \cup \{\top\}$$

# Implementing RPO for infinite systems

## Definition (Precedence description scheme)

Precedence description schema is a function:

$$\mathrm{pds}(f, g) \subseteq \{\bot\} \cup (\{>, \geq\} \times \mathbb{N}^{\mathbb{N}^n} \times \mathbb{N}^{\mathbb{N}^m}) \cup \{\top\}$$

## Definition (Compatibility with precedence description)

A precedence description $(\{\phi_f\}_{f \in \Sigma}, \mathrm{pd})$ is compatible with a precedence description schema pds if:

$$\forall f, g \in \Sigma \,.\, \begin{cases} \mathrm{pd}(f, g) = \top \implies \top \in \mathrm{pds}(f, g) \\ \mathrm{pd}(f, g) = \geq \implies (\geq, \phi_f, \phi_g) \in \mathrm{pds}(f, g) \\ \mathrm{pd}(f, g) = > \implies (>, \phi_f, \phi_g) \in \mathrm{pds}(f, g) \end{cases}$$

# Implementing RPO for infinite systems

## Definition (Compatibility with precedence description)

A precedence description $(\{\phi_f\}_{f \in \Sigma}, \mathrm{pd})$ is compatible with a precedence description schema pds if:

$$\forall f, g \in \Sigma \,.\, \begin{cases} \mathrm{pd}(f,g) = \top \implies \top \in \mathrm{pds}(f,g) \\ \mathrm{pd}(f,g) = \geq \implies (\geq, \phi_f, \phi_g) \in \mathrm{pds}(f,g) \\ \mathrm{pd}(f,g) = > \implies (>, \phi_f, \phi_g) \in \mathrm{pds}(f,g) \end{cases}$$

## Theorem

*Suppose at least three different $\phi$ functions are allowed. Then given a precedence description scheme* pds*, the problem of finding a precedence description compatible with it is NP-complete.*

## Definition (Algorithm)

(1) Compute SCCs in the precedence graph. Refine pds.

$$\mathrm{pds}'(f, g) := \begin{cases} \{\top\} & \text{if } SCC(f) \neq SCC(g) \\ \{s \in \mathrm{pds}(f, g) \mid s \neq \top\} & \text{otherwise} \end{cases}$$

(2) If for any pair $f$ and $g$, $\mathrm{pds}(f, g) = \emptyset$ answer NO and stop.

(3) For every function symbol $f$ compute $\mathrm{IN}_f$, $\mathrm{OUT}_f$:

(4) For every SCC:

- (4a) Compute possible label synthesis functions:
- (4b) Refine pds:
- (4c) If for any $f$ and $g$, $\mathrm{pds}'(f, g) = \emptyset$ answer NO and stop.
- (4d) If $\mathrm{pds}' \neq \mathrm{pds}$ set $\mathrm{pds} := \mathrm{pds}'$ and go to (4a), otherwise continue with (4e).
- (4e) Consider all possible precedences compatible with pds and check whether they comply with condition (2). If none does, answer NO and stop. Otherwise continue with step (4) with the next SCC or with step (5) if there are no more SCCs.

(5)

## Definition (Algorithm)

(1) Compute SCCs in the precedence graph. Refine pds.

(2) If for any pair $f$ and $g$, $\text{pds}(f, g) = \emptyset$ answer NO and stop.

(3) For every function symbol $f$ compute $\text{IN}_f$, $\text{OUT}_f$:

(4) For every SCC:

- (4a) Compute possible label synthesis functions.
- (4b) Refine pds.
- (4c) If for any $f$ and $g$, $\text{pds}'(f, g) = \emptyset$ answer NO and stop.
- (4d) If $\text{pds}' \neq \text{pds}$ set $\text{pds} := \text{pds}'$ and go to (4a), otherwise continue with (4e).
- (4e) Consider all possible precedences compatible with pds and check whether they comply with condition (2). If none does, answer NO and stop. Otherwise continue with step (4) with the next SCC or with step (5) if there are no more SCCs.

(5) Combine solutions for all SCCs to get a precedence description compatible with pds giving rise to a well-founded precedence.

technische universiteit eindhoven

## Definition (Algorithm)

(1) Compute SCCs in the precedence graph. Refine pds.

(2) If for any pair $f$ and $g$, $\text{pds}(f, g) = \emptyset$ answer NO and stop.

(3) For every function symbol $f$ compute $\text{IN}_f, \text{OUT}_f$:

$$\begin{aligned} \text{IN}_f &= \{g \mid \text{pds}(g, f) \neq \{\bot\},\ SCC(f) = SCC(g)\} \\ \text{OUT}_f &= \{g \mid \text{pds}(f, g) \neq \{\bot\},\ SCC(f) = SCC(g)\} \end{aligned}$$

(4) For every SCC:

- (4a) Compute possible label synthesis functions;
- (4b) Refine pds;
- (4c) If for any $f$ and $g$, $\text{pds}'(f, g) = \emptyset$ answer NO and stop.
- (4d) If $\text{pds}' \neq \text{pds}$ set $\text{pds} := \text{pds}'$ and go to (4a), otherwise continue with (4e).
- (4e) Consider all possible precedences compatible with pds and check whether they comply with condition (2). If none does, answer NO and stop. Otherwise continue with step (4) with the next SCC or with step (5) if there are no more SCCs.

(5)

technische universiteit eindhoven

## Definition (Algorithm)

(1) Compute SCCs in the precedence graph. Refine pds.

(2) If for any pair $f$ and $g$, $\mathrm{pds}(f, g) = \emptyset$ answer NO and stop.

(3) For every function symbol $f$ compute $\mathrm{IN}_f$, $\mathrm{OUT}_f$:

(4) For every SCC:

- (4a) Compute possible label synthesis functions:
- (4b) Refine pds:
- (4c) If for any $f$ and $g$, $\mathrm{pds}'(f, g) = \emptyset$ answer NO and stop.
- (4d) If $\mathrm{pds}' \neq \mathrm{pds}$ set $\mathrm{pds} := \mathrm{pds}'$ and go to (4a), otherwise continue with (4e).
- (4e) Consider all possible precedences compatible with pds and check whether they comply with condition (2). If none does, answer NO and stop. Otherwise continue with step (4) with the next SCC or with step (5) if there are no more SCCs.

(5) Combine solutions for all SCCs to get a precedence description compatible with pds giving rise to a well-founded precedence.

## Definition (Algorithm)

(1) Compute SCCs in the precedence graph. Refine pds.

(2) If for any pair $f$ and $g$, $\mathrm{pds}(f, g) = \emptyset$ answer NO and stop.

(3) For every function symbol $f$ compute $\mathrm{IN}_f$, $\mathrm{OUT}_f$:

(4) For every SCC:

(4a) Compute possible label synthesis functions:

$$\mathrm{PSF}_f = \bigcap_{g \in \mathrm{OUT}_f} \{\phi_f \mid (\geq / >, \phi_f, \phi_g) \in \mathrm{pds}(f, g)\} \cap \bigcap_{g \in \mathrm{IN}_f} \{\phi_f \mid (\geq / >, \phi_g, \phi_f) \in \mathrm{pds}(g, f)\}$$

(4b) Refine pds:

(4c) If for any $f$ and $g$, $\mathrm{pds}'(f, g) = \emptyset$ answer NO and stop.

(4d) If $\mathrm{pds}' \neq \mathrm{pds}$ set $\mathrm{pds} := \mathrm{pds}'$ and go to (4a), otherwise continue with (4e).

(4e) Consider all possible precedences compatible with pds and check whether they comply with

## Definition (Algorithm)

(1) Compute SCCs in the precedence graph. Refine pds.

(2) If for any pair $f$ and $g$, $\mathrm{pds}(f, g) = \emptyset$ answer NO and stop.

(3) For every function symbol $f$ compute $\mathrm{IN}_f$, $\mathrm{OUT}_f$:

(4) For every SCC:

(4a) Compute possible label synthesis functions:

(4b) Refine pds:

$$\mathrm{pds}'(f, g) := \{(\geq / >, \phi_f, \phi_g) \in \mathrm{pds}(f, g) \mid \phi_f \in \mathrm{PSF}_f,\ \phi_g \in \mathrm{PSF}_g\}$$

(4c) If for any $f$ and $g$, $\mathrm{pds}'(f, g) = \emptyset$ answer NO and stop.

(4d) If $\mathrm{pds}' \neq \mathrm{pds}$ set $\mathrm{pds} := \mathrm{pds}'$ and go to (4a), otherwise continue with (4e).

(4e) Consider all possible precedences compatible with pds and check whether they comply with condition (2). If none does, answer NO and stop. Otherwise continue with step (4) with the next SCC or with step (5) if there are no more SCCs.

(5) Combine solutions for all SCCs to get a precedence description compatible with pds giving rise to a

## Definition (Algorithm)

(1) Compute SCCs in the precedence graph. Refine pds.

(2) If for any pair $f$ and $g$, $\text{pds}(f, g) = \emptyset$ answer NO and stop.

(3) For every function symbol $f$ compute $\text{IN}_f$, $\text{OUT}_f$:

(4) For every SCC:

- (4a) Compute possible label synthesis functions:
- (4b) Refine pds:
- (4c) If for any $f$ and $g$, $\text{pds}'(f, g) = \emptyset$ answer NO and stop.
- (4d) If $\text{pds}' \neq \text{pds}$ set $\text{pds} := \text{pds}'$ and go to (4a), otherwise continue with (4e).
- (4e) Consider all possible precedences compatible with pds and check whether they comply with condition (2). If none does, answer NO and stop. Otherwise continue with step (4) with the next SCC or with step (5) if there are no more SCCs.

(5) Combine solutions for all SCCs to get a precedence description compatible with pds giving rise to a well-founded precedence.

technische universiteit eindhoven

## Definition (Algorithm)

(1) Compute SCCs in the precedence graph. Refine pds.

(2) If for any pair $f$ and $g$, $\text{pds}(f, g) = \emptyset$ answer NO and stop.

(3) For every function symbol $f$ compute $\text{IN}_f$, $\text{OUT}_f$:

(4) For every SCC:

- (4a) Compute possible label synthesis functions:
- (4b) Refine pds:
- (4c) If for any $f$ and $g$, $\text{pds}'(f, g) = \emptyset$ answer NO and stop.
- (4d) If $\text{pds}' \neq \text{pds}$ set $\text{pds} := \text{pds}'$ and go to (4a), otherwise continue with (4e).
- (4e) Consider all possible precedences compatible with pds and check whether they comply with condition (2). If none does, answer NO and stop. Otherwise continue with step (4) with the next SCC or with step (5) if there are no more SCCs.

(5) Combine solutions for all SCCs to get a precedence description compatible with pds giving rise to a well-founded precedence.

## Definition (Algorithm)

(1) Compute SCCs in the precedence graph. Refine pds.

(2) If for any pair $f$ and $g$, $\mathrm{pds}(f, g) = \emptyset$ answer NO and stop.

(3) For every function symbol $f$ compute $\mathrm{IN}_f$, $\mathrm{OUT}_f$:

(4) For every SCC:

- (4a) Compute possible label synthesis functions:
- (4b) Refine pds:
- (4c) If for any $f$ and $g$, $\mathrm{pds}'(f, g) = \emptyset$ answer NO and stop.
- (4d) If $\mathrm{pds}' \neq \mathrm{pds}$ set $\mathrm{pds} := \mathrm{pds}'$ and go to (4a), otherwise continue with (4e).
- (4e) Consider all possible precedences compatible with pds and check whether they comply with condition (2). If none does, answer NO and stop. Otherwise continue with step (4) with the next SCC or with step (5) if there are no more SCCs.

(5) Combine solutions for all SCCs to get a precedence description compatible with pds giving rise to a well-founded precedence.

## Definition (Algorithm)

(1) Compute SCCs in the precedence graph. Refine pds.

(2) If for any pair $f$ and $g$, $\text{pds}(f, g) = \emptyset$ answer NO and stop.

(3) For every function symbol $f$ compute $\text{IN}_f$, $\text{OUT}_f$:

(4) For every SCC:

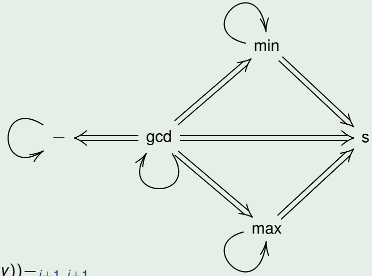
- (4a) Compute possible label synthesis functions:
- (4b) Refine pds:
- (4c) If for any $f$ and $g$, $\text{pds}'(f, g) = \emptyset$ answer NO and stop.
- (4d) If $\text{pds}' \neq \text{pds}$ set $\text{pds} := \text{pds}'$ and go to (4a), otherwise continue with (4e).
- (4e) Consider all possible precedences compatible with pds and check whether they comply with condition (2). If none does, answer NO and stop. Otherwise continue with step (4) with the next SCC or with step (5) if there are no more SCCs.

(5) Combine solutions for all SCCs to get a precedence description compatible with pds giving rise to a well-founded precedence.

technische universiteit eindhoven

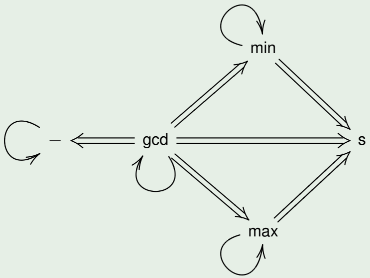
## Example (Precedence description for GCD)

$$
\begin{array}{rcl}
\mathsf{min}_{i,0}(x,0) & \to & 0 \\
\mathsf{min}_{0,j}(0,y) & \to & 0 \\
\mathsf{min}_{i+1,j+1}(\mathsf{s}_i(x),\mathsf{s}_j(y)) & \to & \mathsf{s}_j(\mathsf{min}_{i,j}(x,y)) \\
\mathsf{min}_{i+1,j+1}(\mathsf{s}_i(x),\mathsf{s}_j(y)) & \to & \mathsf{s}_i(\mathsf{min}_{i,j}(x,y)) \\
\mathsf{max}_{i,0}(x,0) & \to & x \\
\mathsf{max}_{0,j}(0,y) & \to & y \\
\mathsf{max}_{i+1,j+1}(\mathsf{s}_i(x),\mathsf{s}_j(y)) & \to & \mathsf{s}_i(\mathsf{max}_{i,j}(x,y)) \\
\mathsf{max}_{i+1,j+1}(\mathsf{s}_i(x),\mathsf{s}_j(y)) & \to & \mathsf{s}_j(\mathsf{max}_{i,j}(x,y)) \\
x -_{i,0} 0 & \to & x \\
\mathsf{s}_i(x) -_{i+1,j+1} \mathsf{s}_j(y)) & \to & x -_{i,j} y \\
\mathsf{gcd}_{i+1,j+1}(\mathsf{s}_i(x),\mathsf{s}_j(y)) & \to & \mathsf{gcd}_{i-j,j+1}(\mathsf{s}_i(\mathsf{max}_{i,j}(x,y)) -_{i+1,j+1} \\
 & & \mathsf{s}_j(\mathsf{min}_{i,j}(x,y)), \mathsf{min}_{i,j}(x,y))) \\
\mathsf{gcd}_{i+1,j+1}(\mathsf{s}_i(x),\mathsf{s}_j(y)) & \to & \mathsf{gcd}_{j-i,i+1}(\mathsf{s}_j(\mathsf{max}_{i,j}(x,y)) -_{j+1,i+1} \\
 & & \mathsf{s}_i(\mathsf{min}_{i,j}(x,y)), \mathsf{min}_{i,j}(x,y)))
\end{array}
$$




$$
\begin{aligned}
\phi_{min} &= \pi_1 \\
\phi_{max} &= \pi_1 \\
\phi_{s} &= \mathrm{id} \\
\phi_{-} &= \mathrm{id} \\
\phi_{gcd} &= +
\end{aligned}
$$

## Example




$$
\begin{aligned}
\phi_{min} &= \pi_1 \\
\phi_{max} &= \pi_1 \\
\phi_{s} &= \mathrm{id} \\
\phi_{-} &= \mathrm{id} \\
\phi_{gcd} &= +
\end{aligned}
$$

$$
\begin{array}{rcll}
\mathrm{min}_{i,j} & > & \mathrm{min}_{k,l} & \text{if } i > k \\
\mathrm{gcd}_{i,j} & > & \mathrm{gcd}_{k,l} & \text{if } i + j > k + l \\
\mathrm{min}_{i,j} & > & \mathrm{s}_{k} & \forall i, j, k \\
\mathrm{gcd}_{i,j} & > & \mathrm{min}_{k,l} & \forall i, j, k, l \\
\mathrm{max}_{i,j} & > & \mathrm{max}_{k,l} & \text{if } i > k \\
\mathrm{gcd}_{i,j} & > & \mathrm{s}_{k} & \forall i, j, k \\
\mathrm{max}_{i,j} & > & \mathrm{s}_{k} & \forall i, j, k \\
\mathrm{gcd}_{i,j} & > & \mathrm{max}_{k,l} & \forall i, j, k, l \\
-_{i,j} & > & -_{k,l} & \text{if } i > k \\
\mathrm{gcd}_{i,j} & > & -_{k,l} & \forall i, j, k, l
\end{array}
$$

## Example (Precedence for GCD)

$$
\begin{array}{rcl}
\mathrm{min}_{i,0}(x,0) & \to & 0 \\
\mathrm{min}_{0,j}(0,y) & \to & 0 \\
\mathrm{min}_{i+1,j+1}(\mathrm{s}_i(x),\mathrm{s}_j(y)) & \to & \mathrm{s}_j(\mathrm{min}_{i,j}(x,y)) \\
\mathrm{min}_{i+1,j+1}(\mathrm{s}_i(x),\mathrm{s}_j(y)) & \to & \mathrm{s}_i(\mathrm{min}_{i,j}(x,y)) \\
\mathrm{max}_{i,0}(x,0) & \to & x \\
\mathrm{max}_{0,j}(0,y) & \to & y \\
\mathrm{max}_{i+1,j+1}(\mathrm{s}_i(x),\mathrm{s}_j(y)) & \to & \mathrm{s}_i(\mathrm{max}_{i,j}(x,y)) \\
\mathrm{max}_{i+1,j+1}(\mathrm{s}_i(x),\mathrm{s}_j(y)) & \to & \mathrm{s}_j(\mathrm{max}_{i,j}(x,y)) \\
x -_{i,0} 0 & \to & x \\
\mathrm{s}_i(x) -_{i+1,j+1} \mathrm{s}_j(y)) & \to & x -_{i,j} y \\
\mathrm{gcd}_{i+1,j+1}(\mathrm{s}_i(x),\mathrm{s}_j(y)) & \to & \mathrm{gcd}_{i-j,j+1}(\mathrm{s}_i(\mathrm{max}_{i,j}(x,y)) -_{i+1,j+1} \\
 & & \mathrm{s}_j(\mathrm{min}_{i,j}(x,y)), \mathrm{min}_{i,j}(x,y))) \\
\mathrm{gcd}_{i+1,j+1}(\mathrm{s}_i(x),\mathrm{s}_j(y)) & \to & \mathrm{gcd}_{j-i,i+1}(\mathrm{s}_j(\mathrm{max}_{i,j}(x,y)) -_{j+1,i+1} \\
 & & \mathrm{s}_i(\mathrm{min}_{i,j}(x,y)), \mathrm{min}_{i,j}(x,y)))
\end{array}
$$

$$
\begin{array}{rcll}
\mathrm{min}_{i,j} & > & \mathrm{min}_{k,l} & \text{if } i > k \\
\mathrm{gcd}_{i,j} & > & \mathrm{gcd}_{k,l} & \text{if } i + j > k + l \\
\mathrm{min}_{i,j} & > & \mathrm{s}_k & \forall i,j,k \\
\mathrm{gcd}_{i,j} & > & \mathrm{min}_{k,l} & \forall i,j,k,l \\
\mathrm{max}_{i,j} & > & \mathrm{max}_{k,l} & \text{if } i > k \\
\mathrm{gcd}_{i,j} & > & \mathrm{s}_k & \forall i,j,k \\
\mathrm{max}_{i,j} & > & \mathrm{s}_k & \forall i,j,k \\
\mathrm{gcd}_{i,j} & > & \mathrm{max}_{k,l} & \forall i,j,k,l \\
-_{i,j} & > & -_{k,l} & \text{if } i > k \\
\mathrm{gcd}_{i,j} & > & -_{k,l} & \forall i,j,k,l
\end{array}
$$

# Conclusions

Conclusions:

- Semantic labelling with natural numbers is an interesting and powerful technique for proving termination.
- It can be implemented (and has been implemented in TPA).
- For some systems (SUBST, GCD) it is the only successful technique at the moment.

Further research:

- Choice of interpretation functions.
- Choice of label synthesis functions.
- Approach for case analysis,
- Comparison of polynomials in presence of side conditions.
- Extension of other techniques to infinite signatures (KBO?).

# Conclusions

Conclusions:

- Semantic labelling with natural numbers is an interesting and powerful technique for proving termination.
- It can be implemented (and has been implemented in TPA).
- For some systems (SUBST, GCD) it is the only successful technique at the moment.

Further research:

- Choice of interpretation functions.
- Choice of label synthesis functions.
- Approach for case analysis,
- Comparison of polynomials in presence of side conditions.
- Extension of other techniques to infinite signatures (KBO?).

# Conclusions

Conclusions:

- Semantic labelling with natural numbers is an interesting and powerful technique for proving termination.
- It can be implemented (and has been implemented in TPA).
- For some systems (SUBST, GCD) it is the only successful technique at the moment.

Further research:

- Choice of interpretation functions.
- Choice of label synthesis functions.
- Approach for case analysis,
- Comparison of polynomials in presence of side conditions.
- Extension of other techniques to infinite signatures (KBO?).

# Conclusions

Conclusions:

- Semantic labelling with natural numbers is an interesting and powerful technique for proving termination.
- It can be implemented (and has been implemented in TPA).
- For some systems (SUBST, GCD) it is the only successful technique at the moment.

Further research:

- Choice of interpretation functions.
- Choice of label synthesis functions.
- Approach for case analysis,
- Comparison of polynomials in presence of side conditions.
- Extension of other techniques to infinite signatures (KBO?).

# Conclusions

Conclusions:

- Semantic labelling with natural numbers is an interesting and powerful technique for proving termination.
- It can be implemented (and has been implemented in TPA).
- For some systems (SUBST, GCD) it is the only successful technique at the moment.

Further research:

- Choice of interpretation functions.
- Choice of label synthesis functions.
- Approach for case analysis,
- Comparison of polynomials in presence of side conditions.
- Extension of other techniques to infinite signatures (KBO?).

# Conclusions

Conclusions:

- Semantic labelling with natural numbers is an interesting and powerful technique for proving termination.
- It can be implemented (and has been implemented in TPA).
- For some systems (SUBST, GCD) it is the only successful technique at the moment.

Further research:

- Choice of interpretation functions.
- Choice of label synthesis functions.
- Approach for case analysis,
- Comparison of polynomials in presence of side conditions.
- Extension of other techniques to infinite signatures (KBO?).

# Conclusions

Conclusions:

- Semantic labelling with natural numbers is an interesting and powerful technique for proving termination.
- It can be implemented (and has been implemented in TPA).
- For some systems (SUBST, GCD) it is the only successful technique at the moment.

Further research:

- Choice of interpretation functions.
- Choice of label synthesis functions.
- Approach for case analysis,
- Comparison of polynomials in presence of side conditions.
- Extension of other techniques to infinite signatures (KBO?).

# Conclusions

Conclusions:

- Semantic labelling with natural numbers is an interesting and powerful technique for proving termination.
- It can be implemented (and has been implemented in TPA).
- For some systems (SUBST, GCD) it is the only successful technique at the moment.

Further research:

- Choice of interpretation functions.
- Choice of label synthesis functions.
- Approach for case analysis,
- Comparison of polynomials in presence of side conditions.
- Extension of other techniques to infinite signatures (KBO?).

Thank you for your attention.