SMT Techniques and Solvers in Automated Termination Analysis

Carsten Fuhs

Birkbeck, University of London

2nd July 2016

14th Workshop on SAT Modulo Theories (SMT)
Coimbra, Portugal

Why analyze termination?

Why analyze termination?

1. **Program**: produces result

Why analyze termination?

1. **Program**: produces result
2. **Input handler**: system reacts

Why analyze termination?

1. **Program**: produces result
2. **Input handler**: system reacts
3. **Mathematical proof**: the induction is valid

Why analyze termination?

1. **Program**: produces result
2. **Input handler**: system reacts
3. **Mathematical proof**: the induction is valid
4. **Biological process**: reaches a stable state

Why analyze termination?

1. **Program**: produces result
2. **Input handler**: system reacts
3. **Mathematical proof**: the induction is valid
4. **Biological process**: reaches a stable state

Variations of the same problem:

- 2 special case of 1
- 3 can be interpreted as 1
- 4 probabilistic version of 1

The bad news

Theorem (Turing 1936)

The question if a given program terminates on a fixed input is undecidable.

The bad news

Theorem (Turing 1936)

The question if a given program terminates on a fixed input is undecidable.

- We want to solve the (harder) question if a given program terminates on **all** inputs.

The bad news

Theorem (Turing 1936)

The question if a given program terminates on a fixed input is undecidable.

- We want to solve the (harder) question if a given program terminates on **all** inputs.
- That's not even semi-decidable!

The bad news

Theorem (Turing 1936)

The question if a given program terminates on a fixed input is undecidable.

- We want to solve the (harder) question if a given program terminates on **all** inputs.
- That's not even semi-decidable!
- But, fear not ...

Termination analysis, classically

Turing 1949

Finally the checker has to verify that the process comes to an end. Here again he should be assisted by the programmer giving a further definite assertion to be verified. This may take the form of a quantity which is asserted to decrease continually and vanish when the machine stops.

“Finally the checker has to verify that the process comes to an end. [...] This may take the form of a quantity which is asserted to decrease continually and vanish when the machine stops.”

Termination analysis, classically

Turing 1949

Finally the checker has to verify that the process comes to an end. Here again he should be assisted by the programmer giving a further definite assertion to be verified. This may take the form of a quantity which is asserted to decrease continually and vanish when the machine stops.

"Finally the checker has to verify that the process comes to an end. [...] This may take the form of a quantity which is asserted to decrease continually and vanish when the machine stops."

1. Find **ranking function** f ("quantity")

Termination analysis, classically

Turing 1949

Finally the checker has to verify that the process comes to an end. Here again he should be assisted by the programmer giving a further definite assertion to be verified. This may take the form of a quantity which is asserted to decrease continually and vanish when the machine stops.

"Finally the checker has to verify that the process comes to an end. [...] This may take the form of a quantity which is asserted to decrease continually and vanish when the machine stops."

1. Find **ranking function** f ("quantity")
2. Prove f to have a **lower bound** ("vanish when the machine stops")

Termination analysis, classically

Turing 1949

Finally the checker has to verify that the process comes to an end. Here again he should be assisted by the programmer giving a further definite assertion to be verified. This may take the form of a quantity which is asserted to decrease continually and vanish when the machine stops.

"Finally the checker has to verify that the process comes to an end. [...] This may take the form of a quantity which is asserted to decrease continually and vanish when the machine stops."

1. Find **ranking function** f ("quantity")
2. Prove f to have a **lower bound** ("vanish when the machine stops")
3. Prove that f **decreases** over time

Termination analysis, classically

Turing 1949

Finally the checker has to verify that the process comes to an end. Here again he should be assisted by the programmer giving a further definite assertion to be verified. This may take the form of a quantity which is asserted to decrease continually and vanish when the machine stops.

"Finally the checker has to verify that the process comes to an end. [...] This may take the form of a quantity which is asserted to decrease continually and vanish when the machine stops."

1. Find **ranking function** f ("quantity")
2. Prove f to have a **lower bound** ("vanish when the machine stops")
3. Prove that f **decreases** over time

Example (Termination can be simple)

while $\mathsf{x} > 0$:
    $\mathsf{x} = \mathsf{x} - 1$

Termination analysis, in the era of SMT solvers

Question: Does program P terminate?

Termination analysis, in the era of SMT solvers

Question: Does program P terminate?

Approach:
Encode termination proof template to SMT formula φ, ask SMT solver

Termination analysis, in the era of SMT solvers

Question: Does program P terminate?

Approach:
Encode termination proof template to SMT formula φ, ask SMT solver

Answer:

Termination analysis, in the era of SMT solvers

Question: Does program P terminate?

Approach:
Encode termination proof template to SMT formula φ, ask SMT solver

Answer:

1. φ satisfiable, model M:
   $\Rightarrow P$ terminating, M fills in the gaps in the termination proof

Termination analysis, in the era of SMT solvers

Question: Does program P terminate?

Approach:
Encode termination proof template to SMT formula φ, ask SMT solver

Answer:

1. φ satisfiable, model M:
   $\Rightarrow$ P terminating, M fills in the gaps in the termination proof
2. φ unsatisfiable:
   $\Rightarrow$ termination status of P unknown
   $\Rightarrow$ try a different template (proof technique)

Termination analysis, in the era of SMT solvers

Question: Does program P terminate?

Approach:
Encode termination proof template to SMT formula φ, ask SMT solver

Answer:

1. φ satisfiable, model M:
   $\Rightarrow$ P terminating, M fills in the gaps in the termination proof
2. φ unsatisfiable:
   $\Rightarrow$ termination status of P unknown
   $\Rightarrow$ try a different template (proof technique)

In practice:

- Encode only a proof **step** at a time
  $\rightarrow$ try to prove only **part** of the program terminating
- **Repeat** until the whole program is proved terminating

The rest of this talk

Termination proving in two parallel worlds

1. Term Rewrite Systems (TRSs)
2. Imperative Programs

1. Term Rewrite Systems (TRSs)

2. Imperative Programs

What's Term Rewriting?

What's Term Rewriting?

Syntactic approach for reasoning in equational first-order logic

What's Term Rewriting?

Syntactic approach for reasoning in equational first-order logic

Core functional programming language without many restrictions (and features) of "real" FP:

What's Term Rewriting?

Syntactic approach for reasoning in equational first-order logic

Core functional programming language without many restrictions (and features) of "real" FP:

- first-order (usually)
- no fixed evaluation strategy
- no fixed order of rules to apply (Haskell: top to bottom)
- untyped
- no pre-defined data structures (integers, arrays, ...)

Why care about termination of term rewriting?

- Termination needed by theorem provers

Why care about termination of term rewriting?

- Termination needed by theorem provers

- Translate program P with inductive data structures (trees) to TRS
  $\Rightarrow$ Termination of TRS implies termination of P
  - Logic programming: Prolog [Giesl et al, *PPDP '12*]
  - (Lazy) functional programming: Haskell [Giesl et al, *TOPLAS '11*]
  - Object-oriented programming: Java [Otto et al, *RTA '10*]

Example (Division)

$$\mathcal{R} = \left\{ \begin{array}{rcl} \mathsf{minus}(x, 0) & \to & x \\ \mathsf{minus}(\mathsf{s}(x), \mathsf{s}(y)) & \to & \mathsf{minus}(x, y) \\ \mathsf{quot}(0, \mathsf{s}(y)) & \to & 0 \\ \mathsf{quot}(\mathsf{s}(x), \mathsf{s}(y)) & \to & \mathsf{s}(\mathsf{quot}(\mathsf{minus}(x, y), \mathsf{s}(y))) \end{array} \right.$$

Term rewriting: Evaluate terms by applying rules from $\mathcal{R}$

$$\mathsf{minus}(\mathsf{s}(\mathsf{s}(0)), \mathsf{s}(0)) \to_{\mathcal{R}} \mathsf{minus}(\mathsf{s}(0), 0) \to_{\mathcal{R}} \mathsf{s}(0)$$

Example (Division)

$$\mathcal{R} = \begin{cases} \mathsf{minus}(x, 0) & \to & x \\ \mathsf{minus}(\mathsf{s}(x), \mathsf{s}(y)) & \to & \mathsf{minus}(x, y) \\ \mathsf{quot}(0, \mathsf{s}(y)) & \to & 0 \\ \mathsf{quot}(\mathsf{s}(x), \mathsf{s}(y)) & \to & \mathsf{s}(\mathsf{quot}(\mathsf{minus}(x, y), \mathsf{s}(y))) \end{cases}$$

Term rewriting: Evaluate terms by applying rules from $\mathcal{R}$

$$\mathsf{minus}(\mathsf{s}(\mathsf{s}(0)), \mathsf{s}(0)) \to_{\mathcal{R}} \mathsf{minus}(\mathsf{s}(0), 0) \to_{\mathcal{R}} \mathsf{s}(0)$$

Termination: No infinite evaluation sequences $t_1 \to_{\mathcal{R}} t_2 \to_{\mathcal{R}} t_3 \to_{\mathcal{R}} \dots$

Example (Division)

$$\mathcal{R} = \begin{cases} \mathsf{minus}(x, \mathsf{0}) & \rightarrow & x \\ \mathsf{minus}(\mathsf{s}(x), \mathsf{s}(y)) & \rightarrow & \mathsf{minus}(x, y) \\ \mathsf{quot}(\mathsf{0}, \mathsf{s}(y)) & \rightarrow & \mathsf{0} \\ \mathsf{quot}(\mathsf{s}(x), \mathsf{s}(y)) & \rightarrow & \mathsf{s}(\mathsf{quot}(\mathsf{minus}(x, y), \mathsf{s}(y))) \end{cases}$$

Term rewriting: Evaluate terms by applying rules from $\mathcal{R}$

$$\mathsf{minus}(\mathsf{s}(\mathsf{s}(\mathsf{0})), \mathsf{s}(\mathsf{0})) \rightarrow_{\mathcal{R}} \mathsf{minus}(\mathsf{s}(\mathsf{0}), \mathsf{0}) \rightarrow_{\mathcal{R}} \mathsf{s}(\mathsf{0})$$

Termination: No infinite evaluation sequences $t_1 \rightarrow_{\mathcal{R}} t_2 \rightarrow_{\mathcal{R}} t_3 \rightarrow_{\mathcal{R}} \ldots$
Show termination using Dependency Pairs

Example (Division)

$$\mathcal{R} = \begin{cases} \mathsf{minus}(x, \mathsf{0}) & \to & x \\ \mathsf{minus}(\mathsf{s}(x), \mathsf{s}(y)) & \to & \mathsf{minus}(x, y) \\ \mathsf{quot}(\mathsf{0}, \mathsf{s}(y)) & \to & \mathsf{0} \\ \mathsf{quot}(\mathsf{s}(x), \mathsf{s}(y)) & \to & \mathsf{s}(\mathsf{quot}(\mathsf{minus}(x, y), \mathsf{s}(y))) \end{cases}$$

Dependency Pairs [Arts, Giesl, *TCS '00*]

Example (Division)

$$\mathcal{R} \;=\; \left\{ \begin{array}{rcl} \mathsf{minus}(x, 0) & \to & x \\ \mathsf{minus}(\mathsf{s}(x), \mathsf{s}(y)) & \to & \mathsf{minus}(x, y) \\ \mathsf{quot}(0, \mathsf{s}(y)) & \to & 0 \\ \mathsf{quot}(\mathsf{s}(x), \mathsf{s}(y)) & \to & \mathsf{s}(\mathsf{quot}(\mathsf{minus}(x, y), \mathsf{s}(y))) \end{array} \right.$$

$$\mathcal{DP} \;=\; \left\{ \begin{array}{rcl} \mathsf{minus}^\sharp(\mathsf{s}(x), \mathsf{s}(y)) & \to & \mathsf{minus}^\sharp(x, y) \\ \mathsf{quot}^\sharp(\mathsf{s}(x), \mathsf{s}(y)) & \to & \mathsf{minus}^\sharp(x, y) \\ \mathsf{quot}^\sharp(\mathsf{s}(x), \mathsf{s}(y)) & \to & \mathsf{quot}^\sharp(\mathsf{minus}(x, y), \mathsf{s}(y)) \end{array} \right.$$

Dependency Pairs [Arts, Giesl, *TCS '00*]

- For TRS $\mathcal{R}$ build dependency pairs $\mathcal{DP}$ ($\sim$ function calls)
- Show: No ∞ call sequence with $\mathcal{DP}$ (eval of $\mathcal{DP}$'s args via $\mathcal{R}$)

Example (Division)

$$\mathcal{R} = \left\{ \begin{array}{rcl} \mathsf{minus}(x, 0) & \to & x \\ \mathsf{minus}(\mathsf{s}(x), \mathsf{s}(y)) & \to & \mathsf{minus}(x, y) \\ \mathsf{quot}(0, \mathsf{s}(y)) & \to & 0 \\ \mathsf{quot}(\mathsf{s}(x), \mathsf{s}(y)) & \to & \mathsf{s}(\mathsf{quot}(\mathsf{minus}(x, y), \mathsf{s}(y))) \end{array} \right.$$

$$\mathcal{DP} = \left\{ \begin{array}{rcl} \mathsf{minus}^\sharp(\mathsf{s}(x), \mathsf{s}(y)) & \to & \mathsf{minus}^\sharp(x, y) \\ \mathsf{quot}^\sharp(\mathsf{s}(x), \mathsf{s}(y)) & \to & \mathsf{minus}^\sharp(x, y) \\ \mathsf{quot}^\sharp(\mathsf{s}(x), \mathsf{s}(y)) & \to & \mathsf{quot}^\sharp(\mathsf{minus}(x, y), \mathsf{s}(y)) \end{array} \right.$$

Dependency Pairs [Arts, Giesl, *TCS '00*]

- For TRS $\mathcal{R}$ build dependency pairs $\mathcal{DP}$ ($\sim$ function calls)
- Show: No ∞ call sequence with $\mathcal{DP}$ (eval of $\mathcal{DP}$'s args via $\mathcal{R}$)
- Dependency Pair Framework [Giesl et al, *JAR '06*] (simplified):

Example (Division)

$$\mathcal{R} = \left\{ \begin{array}{rcl} \mathsf{minus}(x, \mathsf{0}) & \to & x \\ \mathsf{minus}(\mathsf{s}(x), \mathsf{s}(y)) & \to & \mathsf{minus}(x, y) \\ \mathsf{quot}(\mathsf{0}, \mathsf{s}(y)) & \to & \mathsf{0} \\ \mathsf{quot}(\mathsf{s}(x), \mathsf{s}(y)) & \to & \mathsf{s}(\mathsf{quot}(\mathsf{minus}(x, y), \mathsf{s}(y))) \end{array} \right.$$

$$\mathcal{DP} = \left\{ \begin{array}{rcl} \mathsf{minus}^\sharp(\mathsf{s}(x), \mathsf{s}(y)) & \to & \mathsf{minus}^\sharp(x, y) \\ \mathsf{quot}^\sharp(\mathsf{s}(x), \mathsf{s}(y)) & \to & \mathsf{minus}^\sharp(x, y) \\ \mathsf{quot}^\sharp(\mathsf{s}(x), \mathsf{s}(y)) & \to & \mathsf{quot}^\sharp(\mathsf{minus}(x, y), \mathsf{s}(y)) \end{array} \right.$$

Dependency Pairs [Arts, Giesl, *TCS '00*]

- For TRS $\mathcal{R}$ build dependency pairs $\mathcal{DP}$ ($\sim$ function calls)
- Show: No ∞ call sequence with $\mathcal{DP}$ (eval of $\mathcal{DP}$'s args via $\mathcal{R}$)
- Dependency Pair Framework [Giesl et al, *JAR '06*] (simplified):
  while $\mathcal{DP} \neq \emptyset$ **:**

Example (Division)

$$\mathcal{R} = \left\{ \begin{array}{rcl} \mathsf{minus}(x, \mathsf{0}) & \succsim & x \\ \mathsf{minus}(\mathsf{s}(x), \mathsf{s}(y)) & \succsim & \mathsf{minus}(x, y) \\ \mathsf{quot}(\mathsf{0}, \mathsf{s}(y)) & \succsim & \mathsf{0} \\ \mathsf{quot}(\mathsf{s}(x), \mathsf{s}(y)) & \succsim & \mathsf{s}(\mathsf{quot}(\mathsf{minus}(x, y), \mathsf{s}(y))) \end{array} \right.$$

$$\mathcal{DP} = \left\{ \begin{array}{rcl} \mathsf{minus}^\sharp(\mathsf{s}(x), \mathsf{s}(y)) & \succsim & \mathsf{minus}^\sharp(x, y) \\ \mathsf{quot}^\sharp(\mathsf{s}(x), \mathsf{s}(y)) & \succsim & \mathsf{minus}^\sharp(x, y) \\ \mathsf{quot}^\sharp(\mathsf{s}(x), \mathsf{s}(y)) & \succsim & \mathsf{quot}^\sharp(\mathsf{minus}(x, y), \mathsf{s}(y)) \end{array} \right.$$

Dependency Pairs [Arts, Giesl, *TCS '00*]

- For TRS $\mathcal{R}$ build dependency pairs $\mathcal{DP}$ ($\sim$ function calls)
- Show: No ∞ call sequence with $\mathcal{DP}$ (eval of $\mathcal{DP}$'s args via $\mathcal{R}$)
- Dependency Pair Framework [Giesl et al, *JAR '06*] (simplified):
  while $\mathcal{DP} \neq \emptyset$ **:**
  - find well-founded order $\succ$ with $\mathcal{DP} \cup \mathcal{R} \subseteq \succsim$

Example (Division)

$$\mathcal{R} = \begin{cases} \mathsf{minus}(x, \mathsf{0}) & \succsim & x \\ \mathsf{minus}(\mathsf{s}(x), \mathsf{s}(y)) & \succsim & \mathsf{minus}(x, y) \\ \mathsf{quot}(\mathsf{0}, \mathsf{s}(y)) & \succsim & \mathsf{0} \\ \mathsf{quot}(\mathsf{s}(x), \mathsf{s}(y)) & \succsim & \mathsf{s}(\mathsf{quot}(\mathsf{minus}(x, y), \mathsf{s}(y))) \end{cases}$$

$$\mathcal{DP} = \begin{cases} \mathsf{minus}^\sharp(\mathsf{s}(x), \mathsf{s}(y)) & \overset{\succ}{(\sim)} & \mathsf{minus}^\sharp(x, y) \\ \mathsf{quot}^\sharp(\mathsf{s}(x), \mathsf{s}(y)) & \overset{\succ}{(\sim)} & \mathsf{minus}^\sharp(x, y) \\ \mathsf{quot}^\sharp(\mathsf{s}(x), \mathsf{s}(y)) & \overset{\succ}{(\sim)} & \mathsf{quot}^\sharp(\mathsf{minus}(x, y), \mathsf{s}(y)) \end{cases}$$

Dependency Pairs [Arts, Giesl, *TCS '00*]

- For TRS $\mathcal{R}$ build dependency pairs $\mathcal{DP}$ ($\sim$ function calls)
- Show: No ∞ call sequence with $\mathcal{DP}$ (eval of $\mathcal{DP}$'s args via $\mathcal{R}$)
- Dependency Pair Framework [Giesl et al, *JAR '06*] (simplified):
  while $\mathcal{DP} \neq \emptyset$:
  - find well-founded order $\succ$ with $\mathcal{DP} \cup \mathcal{R} \subseteq \succsim$
  - delete $s \to t$ with $s \succ t$ from $\mathcal{DP}$

Example (Division)

$$\mathcal{R} = \left\{ \begin{array}{rcl} \mathsf{minus}(x, \mathsf{0}) & \succsim & x \\ \mathsf{minus}(\mathsf{s}(x), \mathsf{s}(y)) & \succsim & \mathsf{minus}(x, y) \\ \mathsf{quot}(\mathsf{0}, \mathsf{s}(y)) & \succsim & \mathsf{0} \\ \mathsf{quot}(\mathsf{s}(x), \mathsf{s}(y)) & \succsim & \mathsf{s}(\mathsf{quot}(\mathsf{minus}(x, y), \mathsf{s}(y))) \end{array} \right.$$

$$\mathcal{DP} = \left\{ \begin{array}{rcl} \mathsf{minus}^\sharp(\mathsf{s}(x), \mathsf{s}(y)) & \underset{(\sim)}{\succ} & \mathsf{minus}^\sharp(x, y) \\ \mathsf{quot}^\sharp(\mathsf{s}(x), \mathsf{s}(y)) & \underset{(\sim)}{\succ} & \mathsf{minus}^\sharp(x, y) \\ \mathsf{quot}^\sharp(\mathsf{s}(x), \mathsf{s}(y)) & \underset{(\sim)}{\succ} & \mathsf{quot}^\sharp(\mathsf{minus}(x, y), \mathsf{s}(y)) \end{array} \right.$$

Dependency Pairs [Arts, Giesl, *TCS '00*]

- For TRS $\mathcal{R}$ build dependency pairs $\mathcal{DP}$ ($\sim$ function calls)
- Show: No ∞ call sequence with $\mathcal{DP}$ (eval of $\mathcal{DP}$'s args via $\mathcal{R}$)
- Dependency Pair Framework [Giesl et al, *JAR '06*] (simplified):
  while $\mathcal{DP} \neq \emptyset$ **:**
  - find well-founded order $\succ$ with $\mathcal{DP} \cup \mathcal{R} \subseteq \succsim$
  - delete $s \to t$ with $s \succ t$ from $\mathcal{DP}$
- Find $\succ$ automatically and efficiently

Polynomial interpretations

Get $\succ$ via polynomial interpretations $[\,\cdot\,]$ over $\mathbb{N}$ [Lankford '79]

$\rightarrow$ ranking functions for rewriting

Example

$$\mathsf{minus}(\mathsf{s}(x), \mathsf{s}(y)) \;\; \gtrsim \;\; \mathsf{minus}(x, y)$$

Polynomial interpretations

Get $\succ$ via polynomial interpretations $[\,\cdot\,]$ over $\mathbb{N}$ [Lankford '79]

$\rightarrow$ ranking functions for rewriting

Example

$$\mathsf{minus}(\mathsf{s}(x), \mathsf{s}(y)) \;\succsim\; \mathsf{minus}(x, y)$$

Use $[\,\cdot\,]$ with

- $[\mathsf{minus}](x_1, x_2) = x_1$
- $[\mathsf{s}](x_1) = x_1 + 1$

Polynomial interpretations

Get $\succ$ via polynomial interpretations $[\,\cdot\,]$ over $\mathbb{N}$ [Lankford '79]
$\rightarrow$ ranking functions for rewriting

Example

$$\forall x, y. \quad x + 1 \;=\; [\mathsf{minus}(\mathsf{s}(x), \mathsf{s}(y))] \;\geq\; [\mathsf{minus}(x, y)] \;=\; x$$

Use $[\,\cdot\,]$ with

- $[\mathsf{minus}](x_1, x_2) = x_1$
- $[\mathsf{s}](x_1) = x_1 + 1$

Extend to terms:

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

$\succ$ boils down to $>$ over $\mathbb{N}$

Example (Constraints for Division)

$$\mathcal{R} = \left\{ \begin{array}{rcl} \mathsf{minus}(x, \mathsf{0}) & \succsim & x \\ \mathsf{minus}(\mathsf{s}(x), \mathsf{s}(y)) & \succsim & \mathsf{minus}(x, y) \\ \mathsf{quot}(\mathsf{0}, \mathsf{s}(y)) & \succsim & \mathsf{0} \\ \mathsf{quot}(\mathsf{s}(x), \mathsf{s}(y)) & \succsim & \mathsf{s}(\mathsf{quot}(\mathsf{minus}(x, y), \mathsf{s}(y))) \end{array} \right.$$

$$\mathcal{DP} = \left\{ \begin{array}{rcl} \mathsf{minus}^\sharp(\mathsf{s}(x), \mathsf{s}(y)) & \underset{(\sim)}{\succ} & \mathsf{minus}^\sharp(x, y) \\ \mathsf{quot}^\sharp(\mathsf{s}(x), \mathsf{s}(y)) & \underset{(\sim)}{\succ} & \mathsf{minus}^\sharp(x, y) \\ \mathsf{quot}^\sharp(\mathsf{s}(x), \mathsf{s}(y)) & \underset{(\sim)}{\succ} & \mathsf{quot}^\sharp(\mathsf{minus}(x, y), \mathsf{s}(y)) \end{array} \right.$$

Example (Constraints for Division)

$$\mathcal{R} = \begin{cases} \mathsf{minus}(x, 0) & \succsim & x \\ \mathsf{minus}(\mathsf{s}(x), \mathsf{s}(y)) & \succsim & \mathsf{minus}(x, y) \\ \mathsf{quot}(0, \mathsf{s}(y)) & \succsim & 0 \\ \mathsf{quot}(\mathsf{s}(x), \mathsf{s}(y)) & \succsim & \mathsf{s}(\mathsf{quot}(\mathsf{minus}(x, y), \mathsf{s}(y))) \end{cases}$$

$$\mathcal{DP} = \begin{cases} \mathsf{minus}^\sharp(\mathsf{s}(x), \mathsf{s}(y)) & \succ & \mathsf{minus}^\sharp(x, y) \\ \mathsf{quot}^\sharp(\mathsf{s}(x), \mathsf{s}(y)) & \succ & \mathsf{minus}^\sharp(x, y) \\ \mathsf{quot}^\sharp(\mathsf{s}(x), \mathsf{s}(y)) & \succ & \mathsf{quot}^\sharp(\mathsf{minus}(x, y), \mathsf{s}(y)) \end{cases}$$

Use interpretation $[\,\cdot\,]$ over $\mathbb{N}$ with

$$\begin{aligned} [\mathsf{quot}^\sharp](x_1, x_2) &= x_1 + x_2 & \qquad [\mathsf{quot}](x_1, x_2) &= x_1 + x_2 \\ [\mathsf{minus}^\sharp](x_1, x_2) &= x_1 & [\mathsf{minus}](x_1, x_2) &= x_1 \\ [0] &= 0 & [\mathsf{s}](x_1) &= x_1 + 1 \end{aligned}$$

$\curvearrowright$ order solves all constraints

Example (Constraints for Division)

$$\mathcal{R} = \left\{ \begin{array}{rcl} \mathsf{minus}(x, \mathsf{0}) & \succsim & x \\ \mathsf{minus}(\mathsf{s}(x), \mathsf{s}(y)) & \succsim & \mathsf{minus}(x, y) \\ \mathsf{quot}(\mathsf{0}, \mathsf{s}(y)) & \succsim & \mathsf{0} \\ \mathsf{quot}(\mathsf{s}(x), \mathsf{s}(y)) & \succsim & \mathsf{s}(\mathsf{quot}(\mathsf{minus}(x, y), \mathsf{s}(y))) \end{array} \right.$$

$$\mathcal{DP} = \left\{ \right.$$

Use interpretation $[\,\cdot\,]$ over $\mathbb{N}$ with

$$\begin{array}{rcl} [\mathsf{quot}^\sharp](x_1, x_2) & = & x_1 + x_2 \\ [\mathsf{minus}^\sharp](x_1, x_2) & = & x_1 \\ [\mathsf{0}] & = & 0 \end{array} \qquad \begin{array}{rcl} [\mathsf{quot}](x_1, x_2) & = & x_1 + x_2 \\ [\mathsf{minus}](x_1, x_2) & = & x_1 \\ [\mathsf{s}](x_1) & = & x_1 + 1 \end{array}$$

$\curvearrowright$ order solves all constraints

$\curvearrowright$ $\mathcal{DP} = \emptyset$

$\curvearrowright$ termination of division algorithm proved $\square$

Automation

Task: Solve $\quad \mathsf{minus}(\mathsf{s}(x), \mathsf{s}(y)) \succsim \mathsf{minus}(x, y)$

Automation

Task: Solve $\quad \mathsf{minus}(\mathsf{s}(x), \mathsf{s}(y)) \gtrsim \mathsf{minus}(x, y)$

1. Fix a degree, use pol. interpretation with parametric coefficients:

$$[\mathsf{minus}](x, y) = a_{\mathsf{m}} + b_{\mathsf{m}}\, x + c_{\mathsf{m}}\, y, \quad [\mathsf{s}](x) = a_{\mathsf{s}} + b_{\mathsf{s}}\, x$$

Automation

Task: Solve $\qquad \mathsf{minus}(\mathsf{s}(x), \mathsf{s}(y)) \gtrsim \mathsf{minus}(x, y)$

1. Fix a degree, use pol. interpretation with parametric coefficients:

$$[\mathsf{minus}](x, y) = a_{\mathsf{m}} + b_{\mathsf{m}}\, x + c_{\mathsf{m}}\, y, \quad [\mathsf{s}](x) = a_{\mathsf{s}} + b_{\mathsf{s}}\, x$$

2. From term constraint to polynomial constraint:

$$s \gtrsim t \curvearrowright [s] \geq [t]$$

Here: $\quad \forall x, y. \quad (a_{\mathsf{s}}\, b_{\mathsf{m}} + a_{\mathsf{s}}\, c_{\mathsf{m}}) + (b_{\mathsf{s}}\, b_{\mathsf{m}} - b_{\mathsf{m}})\, x + (b_{\mathsf{s}}\, c_{\mathsf{m}} - c_{\mathsf{m}})\, y \geq 0$

Automation

Task: Solve $\mathsf{minus}(\mathsf{s}(x), \mathsf{s}(y)) \gtrsim \mathsf{minus}(x, y)$

1. Fix a degree, use pol. interpretation with parametric coefficients:

$$[\mathsf{minus}](x, y) = a_{\mathsf{m}} + b_{\mathsf{m}}\, x + c_{\mathsf{m}}\, y, \quad [\mathsf{s}](x) = a_{\mathsf{s}} + b_{\mathsf{s}}\, x$$

2. From term constraint to polynomial constraint:

$$s \gtrsim t \curvearrowright [s] \geq [t]$$

Here: $\forall x, y. \quad (a_{\mathsf{s}}\, b_{\mathsf{m}} + a_{\mathsf{s}}\, c_{\mathsf{m}}) + (b_{\mathsf{s}}\, b_{\mathsf{m}} - b_{\mathsf{m}})\, x + (b_{\mathsf{s}}\, c_{\mathsf{m}} - c_{\mathsf{m}})\, y \geq 0$

3. Eliminate $\forall x, y$ by absolute positiveness criterion [Hong, Jakuš, *JAR '98*]:

Here: $a_{\mathsf{s}}\, b_{\mathsf{m}} + a_{\mathsf{s}}\, c_{\mathsf{m}} \geq 0 \ \wedge \ b_{\mathsf{s}}\, b_{\mathsf{m}} - b_{\mathsf{m}} \geq 0 \ \wedge \ b_{\mathsf{s}}\, c_{\mathsf{m}} - c_{\mathsf{m}} \geq 0$

Automation

Task: Solve $\quad \mathsf{minus}(\mathsf{s}(x), \mathsf{s}(y)) \succsim \mathsf{minus}(x, y)$

1. Fix a degree, use pol. interpretation with parametric coefficients:

$$[\mathsf{minus}](x, y) = a_{\mathsf{m}} + b_{\mathsf{m}}\, x + c_{\mathsf{m}}\, y, \quad [\mathsf{s}](x) = a_{\mathsf{s}} + b_{\mathsf{s}}\, x$$

2. From term constraint to polynomial constraint:

$$s \succsim t \;\curvearrowright\; [s] \geq [t]$$

Here: $\quad \forall x, y. \quad (a_{\mathsf{s}}\, b_{\mathsf{m}} + a_{\mathsf{s}}\, c_{\mathsf{m}}) + (b_{\mathsf{s}}\, b_{\mathsf{m}} - b_{\mathsf{m}})\, x + (b_{\mathsf{s}}\, c_{\mathsf{m}} - c_{\mathsf{m}})\, y \geq 0$

3. Eliminate $\forall x, y$ by absolute positiveness criterion [Hong, Jakuš, *JAR '98*]:

Here: $\quad a_{\mathsf{s}}\, b_{\mathsf{m}} + a_{\mathsf{s}}\, c_{\mathsf{m}} \geq 0 \;\wedge\; b_{\mathsf{s}}\, b_{\mathsf{m}} - b_{\mathsf{m}} \geq 0 \;\wedge\; b_{\mathsf{s}}\, c_{\mathsf{m}} - c_{\mathsf{m}} \geq 0$

Automation

Task: Solve $\quad \mathsf{minus}(\mathsf{s}(x), \mathsf{s}(y)) \succsim \mathsf{minus}(x, y)$

1. Fix a degree, use pol. interpretation with parametric coefficients:

$$[\mathsf{minus}](x,y) = a_{\mathsf{m}} + b_{\mathsf{m}}\,x + c_{\mathsf{m}}\,y, \quad [\mathsf{s}](x) = a_{\mathsf{s}} + b_{\mathsf{s}}\,x$$

2. From term constraint to polynomial constraint:

$$s \succsim t \;\curvearrowright\; [s] \geq [t]$$

Here: $\quad \forall x, y. \quad (a_{\mathsf{s}}\,b_{\mathsf{m}} + a_{\mathsf{s}}\,c_{\mathsf{m}}) + (b_{\mathsf{s}}\,b_{\mathsf{m}} - b_{\mathsf{m}})\,x + (b_{\mathsf{s}}\,c_{\mathsf{m}} - c_{\mathsf{m}})\,y \geq 0$

3. Eliminate $\forall x, y$ by absolute positiveness criterion [Hong, Jakuš, *JAR '98*]:

Here: $\quad a_{\mathsf{s}}\,b_{\mathsf{m}} + a_{\mathsf{s}}\,c_{\mathsf{m}} \geq 0 \;\wedge\; b_{\mathsf{s}}\,b_{\mathsf{m}} - b_{\mathsf{m}} \geq 0 \;\wedge\; b_{\mathsf{s}}\,c_{\mathsf{m}} - c_{\mathsf{m}} \geq 0$

Automation

Task: Solve $\quad \mathsf{minus}(\mathsf{s}(x), \mathsf{s}(y)) \succsim \mathsf{minus}(x, y)$

1. Fix a degree, use pol. interpretation with parametric coefficients:

$$[\mathsf{minus}](x, y) = a_{\mathsf{m}} + b_{\mathsf{m}}\, x + c_{\mathsf{m}}\, y, \quad [\mathsf{s}](x) = a_{\mathsf{s}} + b_{\mathsf{s}}\, x$$

2. From term constraint to polynomial constraint:

$$s \succsim t \curvearrowright [s] \geq [t]$$

Here: $\quad \forall x, y. \quad (a_{\mathsf{s}}\, b_{\mathsf{m}} + a_{\mathsf{s}}\, c_{\mathsf{m}}) + (b_{\mathsf{s}}\, b_{\mathsf{m}} - b_{\mathsf{m}})\, x + (b_{\mathsf{s}}\, c_{\mathsf{m}} - c_{\mathsf{m}})\, y \geq 0$

3. Eliminate $\forall x, y$ by absolute positiveness criterion [Hong, Jakuš, *JAR '98*]:

Here: $\quad a_{\mathsf{s}}\, b_{\mathsf{m}} + a_{\mathsf{s}}\, c_{\mathsf{m}} \geq 0 \ \wedge\ b_{\mathsf{s}}\, b_{\mathsf{m}} - b_{\mathsf{m}} \geq 0 \ \wedge\ b_{\mathsf{s}}\, c_{\mathsf{m}} - c_{\mathsf{m}} \geq 0$

Non-linear constraints (QF_NIA), even for linear interpretations

Automation

Task: Solve $\quad \mathsf{minus}(\mathsf{s}(x), \mathsf{s}(y)) \succsim \mathsf{minus}(x, y)$

1. Fix a degree, use pol. interpretation with parametric coefficients:

$$[\mathsf{minus}](x, y) = a_{\mathsf{m}} + b_{\mathsf{m}}\, x + c_{\mathsf{m}}\, y, \quad [\mathsf{s}](x) = a_{\mathsf{s}} + b_{\mathsf{s}}\, x$$

2. From term constraint to polynomial constraint:

$$s \succsim t \curvearrowright [s] \geq [t]$$

Here: $\quad \forall x, y. \quad (a_{\mathsf{s}}\, b_{\mathsf{m}} + a_{\mathsf{s}}\, c_{\mathsf{m}}) + (b_{\mathsf{s}}\, b_{\mathsf{m}} - b_{\mathsf{m}})\, x + (b_{\mathsf{s}}\, c_{\mathsf{m}} - c_{\mathsf{m}})\, y \geq 0$

3. Eliminate $\forall x, y$ by absolute positiveness criterion [Hong, Jakuš, *JAR '98*]:

Here: $\quad a_{\mathsf{s}}\, b_{\mathsf{m}} + a_{\mathsf{s}}\, c_{\mathsf{m}} \geq 0 \ \wedge\ b_{\mathsf{s}}\, b_{\mathsf{m}} - b_{\mathsf{m}} \geq 0 \ \wedge\ b_{\mathsf{s}}\, c_{\mathsf{m}} - c_{\mathsf{m}} \geq 0$

Non-linear constraints (QF_NIA), even for linear interpretations

Task: Show satisfiability of non-linear constraints over $\mathbb{N}$

$\curvearrowright$ Prove termination of given term rewrite system

Extensions

- Polynomials with negative coefficients and max-operator [Hirokawa, Middeldorp, *IC '07*; Fuhs et al, *SAT '07, RTA '08*]
  - models behavior of functions more closely
  - automation via SMT for QF_NIA, more complex Boolean structure

Extensions

- Polynomials with negative coefficients and max-operator [Hirokawa, Middeldorp, *IC '07*; Fuhs et al, *SAT '07, RTA '08*]
  - models behavior of functions more closely
  - automation via SMT for QF_NIA, more complex Boolean structure
- Polynomials over $\mathbb{Q}^+$ and $\mathbb{R}^+$ [Lucas, *RAIRO '05*]
  - non-integer coefficients increase proving power
  - SMT-based automation [Fuhs et al, *AISC '08*; Zankl, Middeldorp, *LPAR '10*; Borralleras et al, *JAR '12*]

Extensions

- Polynomials with negative coefficients and max-operator [Hirokawa, Middeldorp, *IC '07*; Fuhs et al, *SAT '07, RTA '08*]
  - models behavior of functions more closely
  - automation via SMT for QF_NIA, more complex Boolean structure
- Polynomials over $\mathbb{Q}^+$ and $\mathbb{R}^+$ [Lucas, *RAIRO '05*]
  - non-integer coefficients increase proving power
  - SMT-based automation [Fuhs et al, *AISC '08*; Zankl, Middeldorp, *LPAR '10*; Borralleras et al, *JAR '12*]
- Matrix interpretations [Endrullis, Waldmann, Zantema, *JAR '08*]
  - interpretation to vectors over $\mathbb{N}^k$, coefficients are matrices
  - useful for deeply nested terms
  - QF_NIA instances with more complex atoms

Extensions

- Polynomials with negative coefficients and max-operator [Hirokawa, Middeldorp, *IC '07*; Fuhs et al, *SAT '07, RTA '08*]
  - models behavior of functions more closely
  - automation via SMT for QF_NIA, more complex Boolean structure
- Polynomials over $\mathbb{Q}^+$ and $\mathbb{R}^+$ [Lucas, *RAIRO '05*]
  - non-integer coefficients increase proving power
  - SMT-based automation [Fuhs et al, *AISC '08*; Zankl, Middeldorp, *LPAR '10*; Borralleras et al, *JAR '12*]
- Matrix interpretations [Endrullis, Waldmann, Zantema, *JAR '08*]
  - interpretation to vectors over $\mathbb{N}^k$, coefficients are matrices
  - useful for deeply nested terms
  - QF_NIA instances with more complex atoms
- "Arctic" matrices on the max-plus semiring on $\mathbb{N}$ or $\mathbb{Z}$ (instead of plus-times) [Koprowski, Waldmann, *Acta Cyb. '09*]
  - very useful for deeply nested terms
  - can be encoded to QF_**L**IA, but (unary!) bit-blasting seems to be faster in practice [Codish, Fekete, Fuhs, Giesl, Waldmann, *SMT '12*]

Extensions

- Polynomials with negative coefficients and max-operator [Hirokawa, Middeldorp, *IC '07*; Fuhs et al, *SAT '07, RTA '08*]
  - models behavior of functions more closely
  - automation via SMT for QF_NIA, more complex Boolean structure
- Polynomials over $\mathbb{Q}^+$ and $\mathbb{R}^+$ [Lucas, *RAIRO '05*]
  - non-integer coefficients increase proving power
  - SMT-based automation [Fuhs et al, *AISC '08*; Zankl, Middeldorp, *LPAR '10*; Borralleras et al, *JAR '12*]
- Matrix interpretations [Endrullis, Waldmann, Zantema, *JAR '08*]
  - interpretation to vectors over $\mathbb{N}^k$, coefficients are matrices
  - useful for deeply nested terms
  - QF_NIA instances with more complex atoms
- "Arctic" matrices on the max-plus semiring on $\mathbb{N}$ or $\mathbb{Z}$ (instead of plus-times) [Koprowski, Waldmann, *Acta Cyb. '09*]
  - very useful for deeply nested terms
  - can be encoded to QF_**L**IA, but (unary!) bit-blasting seems to be faster in practice [Codish, Fekete, Fuhs, Giesl, Waldmann, *SMT '12*]

Example (bits)

$$\mathcal{R} = \left\{ \begin{array}{rcl} \mathsf{half}(0) & \to & 0 \\ \mathsf{half}(\mathsf{s}(0)) & \to & 0 \\ \mathsf{half}(\mathsf{s}(\mathsf{s}(x))) & \to & \mathsf{s}(\mathsf{half}(x)) \end{array} \qquad \begin{array}{rcl} \mathsf{bits}(0) & \to & 0 \\ \mathsf{bits}(\mathsf{s}(x)) & \to & \mathsf{s}(\mathsf{bits}(\mathsf{half}(\mathsf{s}(x)))) \end{array} \right.$$

Example (bits)

$$\mathcal{R} = \left\{ \begin{array}{rcl} \mathsf{half}(0) & \to & 0 \\ \mathsf{half}(\mathsf{s}(0)) & \to & 0 \\ \mathsf{half}(\mathsf{s}(\mathsf{s}(x))) & \to & \mathsf{s}(\mathsf{half}(x)) \end{array} \quad \begin{array}{rcl} \mathsf{bits}(0) & \to & 0 \\ \mathsf{bits}(\mathsf{s}(x)) & \to & \mathsf{s}(\mathsf{bits}(\mathsf{half}(\mathsf{s}(x)))) \end{array} \right.$$

$$\mathcal{DP} = \left\{ \begin{array}{rcl} \mathsf{half}^\sharp(\mathsf{s}(\mathsf{s}(x))) & \to & \mathsf{half}^\sharp(x) \\ \mathsf{bits}^\sharp(\mathsf{s}(x)) & \to & \mathsf{half}^\sharp(\mathsf{s}(x)) \\ \mathsf{bits}^\sharp(\mathsf{s}(x)) & \to & \mathsf{bits}^\sharp(\mathsf{half}(\mathsf{s}(x))) \end{array} \right.$$

Example (bits)

$$\mathcal{R} = \left\{ \begin{array}{rcl} \mathsf{half}(0) & \succsim & 0 \\ \mathsf{half}(\mathsf{s}(0)) & \succsim & 0 \\ \mathsf{half}(\mathsf{s}(\mathsf{s}(x))) & \succsim & \mathsf{s}(\mathsf{half}(x)) \end{array} \right. \qquad \begin{array}{rcl} \mathsf{bits}(0) & \succsim & 0 \\ \mathsf{bits}(\mathsf{s}(x)) & \succsim & \mathsf{s}(\mathsf{bits}(\mathsf{half}(\mathsf{s}(x)))) \\ & & \end{array}$$

$$\mathcal{DP} = \left\{ \begin{array}{rcl} \mathsf{half}^\sharp(\mathsf{s}(\mathsf{s}(x))) & \underset{(\succsim)}{\succ} & \mathsf{half}^\sharp(x) \\ \mathsf{bits}^\sharp(\mathsf{s}(x)) & \underset{(\succsim)}{\succ} & \mathsf{half}^\sharp(\mathsf{s}(x)) \\ \mathsf{bits}^\sharp(\mathsf{s}(x)) & \underset{(\succsim)}{\succ} & \mathsf{bits}^\sharp(\mathsf{half}(\mathsf{s}(x))) \end{array} \right.$$

Example (bits)

$$\mathcal{R} = \left\{ \begin{array}{rcl} \mathsf{half}(0) & \succsim & 0 \\ \mathsf{half}(\mathsf{s}(0)) & \succsim & 0 \\ \mathsf{half}(\mathsf{s}(\mathsf{s}(x))) & \succsim & \mathsf{s}(\mathsf{half}(x)) \end{array} \quad \begin{array}{rcl} \mathsf{bits}(0) & \succsim & 0 \\ \mathsf{bits}(\mathsf{s}(x)) & \succsim & \mathsf{s}(\mathsf{bits}(\mathsf{half}(\mathsf{s}(x)))) \\ & & \end{array} \right.$$

$$\mathcal{DP} = \left\{ \begin{array}{rcl} & & \\ \mathsf{bits}^\sharp(\mathsf{s}(x)) & \succ & \mathsf{bits}^\sharp(\mathsf{half}(\mathsf{s}(x))) \end{array} \right.$$

Example (bits)

$$\mathcal{R} = \left\{ \begin{array}{rcl} \mathsf{half}(0) & \succsim & 0 \\ \mathsf{half}(\mathsf{s}(0)) & \succsim & 0 \\ \mathsf{half}(\mathsf{s}(\mathsf{s}(x))) & \succsim & \mathsf{s}(\mathsf{half}(x)) \end{array} \qquad \begin{array}{rcl} \mathsf{bits}(0) & \succsim & 0 \\ \mathsf{bits}(\mathsf{s}(x)) & \succsim & \mathsf{s}(\mathsf{bits}(\mathsf{half}(\mathsf{s}(x)))) \end{array} \right.$$

$$\mathcal{DP} = \left\{ \begin{array}{rcl} & & \\ \mathsf{bits}^\sharp(\mathsf{s}(x)) & \succ & \mathsf{bits}^\sharp(\mathsf{half}(\mathsf{s}(x))) \end{array} \right.$$

- Classic polynomials cannot solve $\mathsf{bits}^\sharp(\mathsf{s}(x)) \succ \mathsf{bits}^\sharp(\mathsf{half}(\mathsf{s}(x)))$

Example (bits)

$$\mathcal{R} = \left\{ \begin{array}{rcl} \mathsf{half}(0) & \succsim & 0 \\ \mathsf{half}(\mathsf{s}(0)) & \succsim & 0 \\ \mathsf{half}(\mathsf{s}(\mathsf{s}(x))) & \succsim & \mathsf{s}(\mathsf{half}(x)) \end{array} \right. \qquad \begin{array}{rcl} \mathsf{bits}(0) & \succsim & 0 \\ \mathsf{bits}(\mathsf{s}(x)) & \succsim & \mathsf{s}(\mathsf{bits}(\mathsf{half}(\mathsf{s}(x)))) \end{array}$$

$$\mathcal{DP} = \left\{ \begin{array}{rcl} & & \\ \mathsf{bits}^\sharp(\mathsf{s}(x)) & \succ & \mathsf{bits}^\sharp(\mathsf{half}(\mathsf{s}(x))) \end{array} \right.$$

- Classic polynomials cannot solve $\mathsf{bits}^\sharp(\mathsf{s}(x)) \succ \mathsf{bits}^\sharp(\mathsf{half}(\mathsf{s}(x)))$
- Remedy: $[\mathsf{bits}^\sharp](x) = x, \quad [\mathsf{s}](x) = x + 1, \quad [\mathsf{half}](x) = x - 1$

Example (bits)

$$\mathcal{R} = \left\{ \begin{array}{rcl} \mathsf{half}(0) & \succsim & 0 \\ \mathsf{half}(\mathsf{s}(0)) & \succsim & 0 \\ \mathsf{half}(\mathsf{s}(\mathsf{s}(x))) & \succsim & \mathsf{s}(\mathsf{half}(x)) \end{array} \right. \qquad \begin{array}{rcl} \mathsf{bits}(0) & \succsim & 0 \\ \mathsf{bits}(\mathsf{s}(x)) & \succsim & \mathsf{s}(\mathsf{bits}(\mathsf{half}(\mathsf{s}(x)))) \end{array}$$

$$\mathcal{DP} = \left\{ \begin{array}{rcl} \\ \\ \mathsf{bits}^\sharp(\mathsf{s}(x)) & \succ & \mathsf{bits}^\sharp(\mathsf{half}(\mathsf{s}(x))) \end{array} \right.$$

- Classic polynomials cannot solve $\mathsf{bits}^\sharp(\mathsf{s}(x)) \succ \mathsf{bits}^\sharp(\mathsf{half}(\mathsf{s}(x)))$
- Remedy: $[\mathsf{bits}^\sharp](x) = x$, $[\mathsf{s}](x) = x + 1$, $[\mathsf{half}](x) = x - 1$
- But: Then $\succ$ not well founded any more:

$$0 \succ \mathsf{half}(0) \succ \mathsf{half}(\mathsf{half}(0)) \succ \dots$$

Example (bits)

$$\mathcal{R} = \left\{ \begin{array}{rcl} \mathsf{half}(0) & \succsim & 0 \\ \mathsf{half}(\mathsf{s}(0)) & \succsim & 0 \\ \mathsf{half}(\mathsf{s}(\mathsf{s}(x))) & \succsim & \mathsf{s}(\mathsf{half}(x)) \end{array} \right. \qquad \begin{array}{rcl} \mathsf{bits}(0) & \succsim & 0 \\ \mathsf{bits}(\mathsf{s}(x)) & \succsim & \mathsf{s}(\mathsf{bits}(\mathsf{half}(\mathsf{s}(x)))) \end{array}$$

$$\mathcal{DP} = \left\{ \begin{array}{rcl} \mathsf{bits}^\sharp(\mathsf{s}(x)) & \succ & \mathsf{bits}^\sharp(\mathsf{half}(\mathsf{s}(x))) \end{array} \right.$$

- Classic polynomials cannot solve $\mathsf{bits}^\sharp(\mathsf{s}(x)) \succ \mathsf{bits}^\sharp(\mathsf{half}(\mathsf{s}(x)))$
- Remedy: $[\mathsf{bits}^\sharp](x) = x, \quad [\mathsf{s}](x) = x + 1, \quad [\mathsf{half}](x) = x - 1$
- But: Then $\succ$ not well founded any more:

$$0 \succ \mathsf{half}(0) \succ \mathsf{half}(\mathsf{half}(0)) \succ \ldots$$

$\Rightarrow$ Solution [Hirokawa, Middeldorp, *IC '07*]:

$$[\mathsf{half}](x_1) = \max(x_1 - 1, 0)$$

$\Rightarrow$ $[\mathsf{half}(\mathsf{s}(x))] = \max((x + 1) - 1, 0) = x$

Example (bits)

$$\mathcal{R} = \left\{ \begin{array}{rcl} \mathsf{half}(0) & \succsim & 0 \\ \mathsf{half}(\mathsf{s}(0)) & \succsim & 0 \\ \mathsf{half}(\mathsf{s}(\mathsf{s}(x))) & \succsim & \mathsf{s}(\mathsf{half}(x)) \end{array} \right. \qquad \begin{array}{rcl} \mathsf{bits}(0) & \succsim & 0 \\ \mathsf{bits}(\mathsf{s}(x)) & \succsim & \mathsf{s}(\mathsf{bits}(\mathsf{half}(\mathsf{s}(x)))) \end{array}$$

$$\mathcal{DP} = \left\{ \right.$$

- Classic polynomials cannot solve $\mathsf{bits}^\sharp(\mathsf{s}(x)) \succ \mathsf{bits}^\sharp(\mathsf{half}(\mathsf{s}(x)))$
- Remedy: $[\mathsf{bits}^\sharp](x) = x$, $[\mathsf{s}](x) = x + 1$, $[\mathsf{half}](x) = x - 1$
- But: Then $\succ$ not well founded any more:

$$0 \succ \mathsf{half}(0) \succ \mathsf{half}(\mathsf{half}(0)) \succ \ldots$$

$\Rightarrow$ Solution [Hirokawa, Middeldorp, *IC '07*]:

$$[\mathsf{half}](x_1) = \max(x_1 - 1, 0)$$

$\Rightarrow$ $[\mathsf{half}(\mathsf{s}(x))] = \max((x + 1) - 1, 0) = x$

- Problem: Expressions like $\max(x_1 - 1, 0)$ are no polynomials
- For $[s] > [t]$, show

- Problem: Expressions like $\max(x_1 - 1, 0)$ are no polynomials
- For $[s] > [t]$, show $[s]^{left} > [t]^{right}$
  - $[s]^{left}$ under-approximation of $[s]$
  - $[t]^{right}$ over-approximation of $[t]$
  - $[s]^{left}, [t]^{right}$ polynomials

- Problem: Expressions like $\max(x_1 - 1, 0)$ are no polynomials
- For $[s] > [t]$, show $[s]^{left} > [t]^{right}$
  - $[s]^{left}$ under-approximation of $[s]$
  - $[t]^{right}$ over-approximation of $[t]$
  - $[s]^{left}, [t]^{right}$ polynomials
- Automation initially: **Generate-and-test**
- Approx. for $\max(p, 0)$ depend on **signum** of constant addend of p

$$\begin{array}{rclcrcl} [\mathsf{s}(x)] & = & \max(x+1,0) & \Rightarrow & [\mathsf{s}(x)]^{right} & = & x+1 \\ [\mathsf{half}(x)] & = & \max(x-1,0) & \Rightarrow & [\mathsf{half}(x)]^{right} & = & x \end{array}$$

- Problem: Expressions like $\max(x_1 - 1, 0)$ are no polynomials
- For $[s] > [t]$, show $[s]^{left} > [t]^{right}$
  - $[s]^{left}$ under-approximation of $[s]$
  - $[t]^{right}$ over-approximation of $[t]$
  - $[s]^{left}, [t]^{right}$ polynomials
- Automation initially: **Generate-and-test**
- Approx. for $\max(p, 0)$ depend on **signum** of constant addend of p

$$\begin{array}{rclcrcl} [\mathsf{s}(x)] & = & \max(x+1,0) & \Rightarrow & [\mathsf{s}(x)]^{right} & = & x+1 \\ [\mathsf{half}(x)] & = & \max(x-1,0) & \Rightarrow & [\mathsf{half}(x)]^{right} & = & x \end{array}$$

- Solution [Fuhs et al, *SAT '07*]: Encode case analysis ...

$$[\,f(x)] = \max(a_f\, x_1 + b_f, 0) \quad \Rightarrow \quad [\,f(x)]^{right} = a_f\, x_1 + c_{f(x)}$$

... using side constraints

$$(b_f \geq 0 \;\rightarrow\; c_{f(x)} = b_f) \quad \wedge \quad (b_f < 0 \;\rightarrow\; c_{f(x)} = 0)$$

- Boolean structure in SMT quite handy!

(SAT and) SMT solving for path orders

Path orders: based on precedences of function symbols

- Recursive Path Order [Dershowitz, *TCS '82*; Codish et al, *JAR '11*]
- Weighted Path Order [Yamada, Kusakari, Sakabe, *SCP '15*]

(SAT and) SMT solving for path orders

Path orders: based on precedences of function symbols

- Recursive Path Order [Dershowitz, *TCS '82*; Codish et al, *JAR '11*]
- Weighted Path Order [Yamada, Kusakari, Sakabe, *SCP '15*]
- Knuth-Bendix Order [Knuth, Bendix, *CPAA '70*]
  $\rightarrow$ SMT-Encoding to QF_LIA [Zankl, Hirokawa, Middeldorp, *JAR '09*]
  outperformed polynomial time algorithm [Korovin, Voronkov, *IC '03*]
  in experiments

(SAT and) SMT solving for path orders

Path orders: based on precedences of function symbols

- Recursive Path Order [Dershowitz, *TCS '82*; Codish et al, *JAR '11*]
- Weighted Path Order [Yamada, Kusakari, Sakabe, *SCP '15*]
- Knuth-Bendix Order [Knuth, Bendix, *CPAA '70*]
  $\rightarrow$ SMT-Encoding to QF_LIA [Zankl, Hirokawa, Middeldorp, *JAR '09*]
  outperformed polynomial time algorithm [Korovin, Voronkov, *IC '03*]
  in experiments

  Analogy: Exponential-time simplex vs. polynomial-time interior-point methods for QF_LRA?

Further extensions

- Constrained term rewriting [Fuhs et al, *RTA '09*; Kop, Nishida, *FroCoS '13*; Rocha, Meseguer, Muñoz, *WRLA '14*]
    - term rewriting with predefined operations from SMT theories, e.g. integer arithmetic, ...
    - target language for translations from programming languages

Further extensions

- Constrained term rewriting [Fuhs et al, *RTA '09*; Kop, Nishida, *FroCoS '13*; Rocha, Meseguer, Muñoz, *WRLA '14*]
  - term rewriting with predefined operations from SMT theories, e.g. integer arithmetic, ...
  - target language for translations from programming languages

- Complexity analysis [Hirokawa, Moser, *IJCAR '08*; Noschinski, Emmes, Giesl, *JAR '13*]

  Can re-use termination machinery to infer and prove statements like "runtime complexity of this TRS is in $\mathcal{O}(n^3)$"

SMT solvers *from* termination analysis

Annual SMT-COMP, division QF_NIA

Year	Winner
2009	Barcelogic-QF_NIA
2010	MiniSmt
2011	AProVE
2012	*no QF_NIA*
2013	*no SMT-COMP*
2014	AProVE
2015	AProVE
2016	$\rightarrow$ *today, 4 pm*

SMT solvers *from* termination analysis

Annual SMT-COMP, division QF_NIA

Year	Winner
2009	Barcelogic-QF_NIA
2010	MiniSmt (spin-off of $\mathsf{T_TT_2}$)
2011	AProVE
2012	*no QF_NIA*
2013	*no SMT-COMP*
2014	AProVE
2015	AProVE
2016	$\rightarrow$ *today, 4 pm*

$\Rightarrow$ Termination provers can also be successful SMT solvers!

SMT solvers *from* termination analysis

Annual SMT-COMP, division QF_NIA

Year	Winner
2009	Barcelogic-QF_NIA
2010	MiniSmt (spin-off of $\mathsf{T_TT_2}$)
2011	AProVE
2012	*no QF_NIA*
2013	*no SMT-COMP*
2014	AProVE
2015	AProVE
2016	$\rightarrow$ *today, 4 pm*

$\Rightarrow$ Termination provers can also be successful SMT solvers!

(disclaimer: Z3 participated only *hors concours* in the last years)

Papers on termination of imperative programs often about **integers** as data

Papers on termination of imperative programs often about **integers** as data

Example (Imperative program)

```
if x ≥ 0:
    while x ≠ 0:
        x = x − 1
```

Does this program terminate?

Papers on termination of imperative programs often about **integers** as data

Example (Imperative program)

ℓ_0: **if** x $\geq$ 0:
ℓ_1: **while** x $\neq$ 0:
ℓ_2: x $=$ x $-$ 1

Does this program terminate?

Example (Equivalent translation to transition system)

$$
\begin{array}{lcll}
\ell_0(x) & \to & \ell_1(x) & [x \geq 0] \\
\ell_1(x) & \to & \ell_2(x) & [x \neq 0] \\
\ell_2(x) & \to & \ell_1(x-1) & \\
\ell_1(x) & \to & \ell_3(x) & [x == 0]
\end{array}
$$

Papers on termination of imperative programs often about **integers** as data

Example (Imperative program)

ℓ_0: **if** x $\geq$ 0:
ℓ_1: **while** x $\neq$ 0:
ℓ_2: x = x − 1

Does this program terminate?

Example (Equivalent translation to transition system)

$$\begin{array}{lcll}
\ell_0(x) & \to & \ell_1(x) & [x \geq 0] \\
\ell_1(x) & \to & \ell_2(x) & [x \neq 0] \\
\ell_2(x) & \to & \ell_1(x-1) & \\
\ell_1(x) & \to & \ell_3(x) & [x == 0]
\end{array}$$

Oh no! $\ell_1(-1) \to \ell_2(-1) \to \ell_1(-2) \to \ell_2(-2) \to \ell_1(-3) \to \cdots$

Papers on termination of imperative programs often about **integers** as data

Example (Imperative program)

ℓ_0: **if** x $\geq$ 0:
ℓ_1: **while** x $\neq$ 0:
ℓ_2: x = x $-$ 1

Does this program terminate?

Example (Equivalent translation to transition system)

$$
\begin{array}{lcll}
\ell_0(x) & \to & \ell_1(x) & [x \geq 0] \\
\ell_1(x) & \to & \ell_2(x) & [x \neq 0] \\
\ell_2(x) & \to & \ell_1(x-1) & \\
\ell_1(x) & \to & \ell_3(x) & [x == 0]
\end{array}
$$

Oh no! $\ell_1(-1) \to \ell_2(-1) \to \ell_1(-2) \to \ell_2(-2) \to \ell_1(-3) \to \cdots$

$\Rightarrow$ **Restrict initial states** to $\ell_0(z)$ for $z \in \mathbb{Z}$

Papers on termination of imperative programs often about **integers** as data

Example (Imperative program)

```
ℓ0:  if x ≥ 0:
ℓ1:      while x ≠ 0:
ℓ2:          x = x − 1
```

Does this program terminate?

Example (Equivalent translation to transition system)

$$
\begin{array}{lcll}
\ell_0(x) & \to & \ell_1(x) & [x \geq 0] \\
\ell_1(x) & \to & \ell_2(x) & [x \neq 0] \\
\ell_2(x) & \to & \ell_1(x-1) & \\
\ell_1(x) & \to & \ell_3(x) & [x == 0]
\end{array}
$$

Oh no! $\ell_1(-1) \to \ell_2(-1) \to \ell_1(-2) \to \ell_2(-2) \to \ell_1(-3) \to \cdots$

$\Rightarrow$ **Restrict initial states** to $\ell_0(z)$ for $z \in \mathbb{Z}$

$\Rightarrow$ Find **invariant** $x \geq 0$ at ℓ_1, ℓ_2

Papers on termination of imperative programs often about **integers** as data

Example (Imperative program)

ℓ_0: **if** x $\geq$ 0:
ℓ_1: **while** x $\neq$ 0:
ℓ_2: x = x − 1

Does this program terminate?

Example (Equivalent translation to transition system)

$$
\begin{array}{lcll}
\ell_0(x) & \to & \ell_1(x) & [x \geq 0] \\
\ell_1(x) & \to & \ell_2(x) & [x \neq 0 \wedge x \geq 0] \\
\ell_2(x) & \to & \ell_1(x-1) & [x \geq 0] \\
\ell_1(x) & \to & \ell_3(x) & [x == 0 \wedge x \geq 0]
\end{array}
$$

Oh no! $\ell_1(-1) \to \ell_2(-1) \to \ell_1(-2) \to \ell_2(-2) \to \ell_1(-3) \to \cdots$

⇒ **Restrict initial states** to $\ell_0(z)$ for $z \in \mathbb{Z}$

⇒ Find **invariant** $x \geq 0$ at ℓ_1, ℓ_2

Proving termination with invariants

Example (Transition system with invariants)

$$
\begin{array}{lcll}
\ell_0(x) & \to & \ell_1(x) & [x \geq 0] \\
\ell_1(x) & \to & \ell_2(x) & [x \neq 0 \wedge x \geq 0] \\
\ell_2(x) & \to & \ell_1(x-1) & [x \geq 0] \\
\ell_1(x) & \to & \ell_3(x) & [x == 0 \wedge x \geq 0]
\end{array}
$$

Prove termination by ranking function $[\,\cdot\,]$ with $[\ell_0](x) = [\ell_1](x) = \cdots = x$

Proving termination with invariants

Example (Transition system with invariants)

$$
\begin{array}{lcll}
\ell_0(x) & \succsim & \ell_1(x) & [x \geq 0] \\
\ell_1(x) & \succsim & \ell_2(x) & [x \neq 0 \wedge x \geq 0] \\
\ell_2(x) & \succ & \ell_1(x-1) & [x \geq 0] \\
\ell_1(x) & \succsim & \ell_3(x) & [x == 0 \wedge x \geq 0]
\end{array}
$$

Prove termination by ranking function $[\,\cdot\,]$ with $[\ell_0](x) = [\ell_1](x) = \cdots = x$

Proving termination with invariants

Example (Transition system with invariants)

$$\begin{array}{llll} \ell_0(x) & \succsim & \ell_1(x) & [x \geq 0] \\ \ell_1(x) & \succsim & \ell_2(x) & [x \neq 0 \wedge x \geq 0] \\ \ell_2(x) & \succ & \ell_1(x-1) & [x \geq 0] \\ \ell_1(x) & \succsim & \ell_3(x) & [x == 0 \wedge x \geq 0] \end{array}$$

Prove termination by ranking function $[\,\cdot\,]$ with $[\ell_0](x) = [\ell_1](x) = \cdots = x$

Automate search using parametric ranking function:

$$[\ell_0](x) = a_0 + b_0 \cdot x, \quad [\ell_1](x) = a_1 + b_1 \cdot x, \quad \ldots$$

Proving termination with invariants

Example (Transition system with invariants)

$$
\begin{array}{lcll}
\ell_0(x) & \succsim & \ell_1(x) & [x \geq 0] \\
\ell_1(x) & \succsim & \ell_2(x) & [x \neq 0 \land x \geq 0] \\
\ell_2(x) & \succ & \ell_1(x-1) & [x \geq 0] \\
\ell_1(x) & \succsim & \ell_3(x) & [x == 0 \land x \geq 0]
\end{array}
$$

Prove termination by ranking function $[\,\cdot\,]$ with $[\ell_0](x) = [\ell_1](x) = \cdots = x$

Automate search using parametric ranking function:

$$[\ell_0](x) = a_0 + b_0 \cdot x, \quad [\ell_1](x) = a_1 + b_1 \cdot x, \quad \ldots$$

Constraints e.g.:

$$
\begin{array}{lcll}
x \geq 0 & \Rightarrow & a_2 + b_2 \cdot x > a_1 + b_1 \cdot (x-1) & \text{“decrease \ldots”} \\
x \geq 0 & \Rightarrow & a_2 + b_2 \cdot x \geq 0 & \text{“\ldots against a bound”}
\end{array}
$$

Proving termination with invariants

Example (Transition system with invariants)

$$\begin{array}{lclcl}
\ell_0(x) & \succsim & \ell_1(x) & & [x \geq 0] \\
\ell_1(x) & \succsim & \ell_2(x) & & [x \neq 0 \wedge x \geq 0] \\
\ell_2(x) & \succ & \ell_1(x-1) & & [x \geq 0] \\
\ell_1(x) & \succsim & \ell_3(x) & & [x == 0 \wedge x \geq 0]
\end{array}$$

Prove termination by ranking function $[\cdot]$ with $[\ell_0](x) = [\ell_1](x) = \cdots = x$

Automate search using parametric ranking function:

$$[\ell_0](x) = a_0 + b_0 \cdot x, \quad [\ell_1](x) = a_1 + b_1 \cdot x, \quad \ldots$$

Constraints e.g.:

$$\begin{array}{lcll}
x \geq 0 & \Rightarrow & a_2 + b_2 \cdot x > a_1 + b_1 \cdot (x-1) & \text{“decrease \ldots”} \\
x \geq 0 & \Rightarrow & a_2 + b_2 \cdot x \geq 0 & \text{“\ldots\ against a bound”}
\end{array}$$

Use Farkas' Lemma to eliminate $\forall x$, QF_LRA solver gives model for a_i, b_i.

Proving termination with invariants

Example (Transition system with invariants)

$$
\begin{array}{lcll}
\ell_0(x) & \succsim & \ell_1(x) & [x \geq 0] \\
\ell_1(x) & \succsim & \ell_2(x) & [x \neq 0 \wedge x \geq 0] \\
\ell_2(x) & \succ & \ell_1(x-1) & [x \geq 0] \\
\ell_1(x) & \succsim & \ell_3(x) & [x == 0 \wedge x \geq 0]
\end{array}
$$

Prove termination by ranking function $[\,\cdot\,]$ with $[\ell_0](x) = [\ell_1](x) = \cdots = x$

Automate search using parametric ranking function:

$$[\ell_0](x) = a_0 + b_0 \cdot x, \quad [\ell_1](x) = a_1 + b_1 \cdot x, \quad \ldots$$

Constraints e.g.:

$$
\begin{array}{lcll}
x \geq 0 & \Rightarrow & a_2 + b_2 \cdot x > a_1 + b_1 \cdot (x-1) & \text{"decrease \ldots"} \\
x \geq 0 & \Rightarrow & a_2 + b_2 \cdot x \geq 0 & \text{"\ldots against a bound"}
\end{array}
$$

Use Farkas' Lemma to eliminate $\forall x$, QF_LRA solver gives model for a_i, b_i.
More: [Podelski, Rybalchenko, *VMCAI '04*, Alias et al, *SAS '10*]

Searching for invariants using SMT

Termination prover needs to find invariants for programs on integers

- Statically before the translation [Ströder et al, *IJCAR '14*]
- In cooperation with a safety prover [Brockschmidt, Cook, Fuhs, *CAV '13*]
- Using Max-SMT
  [Larraz, Oliveras, Rodríguez-Carbonell, Rubio, *FMCAD '13*]

Nowadays all SMT-based!

Extensions

- Proving *non*-termination (infinite run from initial states is possible) [Gupta et al, *POPL '08*, Brockschmidt et al, *FoVeOOS '11*, Chen et al, *TACAS '14*, Larraz et al, *CAV '14*, Cook et al, *FMCAD '14*]
- CTL* model checking for *infinite* state systems based on termination and non-termination provers
  [Cook, Khlaaf, Piterman, *CAV '15*]
- Complexity bounds
  [Alias et al, *SAS '10*, Hoffmann, Shao, *JFP '15*, Brockschmidt et al, *TOPLAS '16*]

Conclusion

- Automated termination analysis for term rewriting and for imperative programs developed in parallel over the last ~ 15 years

Conclusion

- Automated termination analysis for term rewriting and for imperative programs developed in parallel over the last $\sim$ 15 years
- Term rewriting: need to encode how to represent data structures

Conclusion

- Automated termination analysis for term rewriting and for imperative programs developed in parallel over the last $\sim$ 15 years
- Term rewriting: need to encode how to represent data structures
- Imperative programs on integers: need to consider reachability and invariants

Conclusion

- Automated termination analysis for term rewriting and for imperative programs developed in parallel over the last $\sim$ 15 years
- Term rewriting: need to encode how to represent data structures
- Imperative programs on integers: need to consider reachability and invariants
- Since a few years cross-fertilization

Conclusion

- Automated termination analysis for term rewriting and for imperative programs developed in parallel over the last $\sim$ 15 years
- Term rewriting: need to encode how to represent data structures
- Imperative programs on integers: need to consider reachability and invariants
- Since a few years cross-fertilization
- Automation heavily relies on SMT solving for automation

Conclusion

- Automated termination analysis for term rewriting and for imperative programs developed in parallel over the last $\sim$ 15 years
- Term rewriting: need to encode how to represent data structures
- Imperative programs on integers: need to consider reachability and invariants
- Since a few years cross-fertilization
- Automation heavily relies on SMT solving for automation
- Needs of termination analysis have also led to better SMT solvers

Conclusion

- Automated termination analysis for term rewriting and for imperative programs developed in parallel over the last $\sim$ 15 years
- Term rewriting: need to encode how to represent data structures
- Imperative programs on integers: need to consider reachability and invariants
- Since a few years cross-fertilization
- Automation heavily relies on SMT solving for automation
- Needs of termination analysis have also led to better SMT solvers
- Annual termCOMP:
  `http://termination-portal.org/wiki/Termination_Competition`

Conclusion

- Automated termination analysis for term rewriting and for imperative programs developed in parallel over the last $\sim$ 15 years
- Term rewriting: need to encode how to represent data structures
- Imperative programs on integers: need to consider reachability and invariants
- Since a few years cross-fertilization
- Automation heavily relies on SMT solving for automation
- Needs of termination analysis have also led to better SMT solvers
- Annual termCOMP:
  `http://termination-portal.org/wiki/Termination_Competition`

Without SAT and SMT solving, push-button termination analysis would not be where it is today

References I

C. Alias, A. Darte, P. Feautrier, and L. Gonnord. Multi-dimensional rankings, program termination, and complexity bounds of flowchart programs. In *SAS '10*, pages 117–133, 2010.

T. Arts and J. Giesl. Termination of term rewriting using dependency pairs. *Theoretical Computer Science*, 236(1-2):133–178, 2000.

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

M. Brockschmidt, T. Ströder, C. Otto, and J. Giesl. Automated detection of non-termination and `NullPointerExceptions` for Java Bytecode. In *FoVeOOS '11*, pages 123–141, 2012.

M. Brockschmidt, B. Cook, and C. Fuhs. Better termination proving through cooperation. In *CAV '13*, pages 413–429, 2013.

References II

M. Brockschmidt, F. Emmes, S. Falke, C. Fuhs, and J. Giesl. Analyzing runtime and size complexity of integer programs. *ACM TOPLAS*, 2016. To appear.

H.-Y. Chen, B. Cook, C. Fuhs, K. Nimkar, and P. W. O'Hearn. Proving nontermination via safety. In *TACAS '14*, pages 156–171, 2014.

M. Codish, Y. Fekete, C. Fuhs, J. Giesl, and J. Waldmann. Exotic semiring constraints (extended abstract). In *SMT '12*, pages 87–96, 2012a.

M. Codish, J. Giesl, P. Schneider-Kamp, and R. Thiemann. SAT solving for termination proofs with recursive path orders and dependency pairs. *Journal of Automated Reasoning*, 49(1):53–93, 2012b.

References III

B. Cook, C. Fuhs, K. Nimkar, and P. W. O'Hearn. Disproving termination with overapproximation. In *FMCAD '14*, pages 67–74, 2014.

B. Cook, H. Khlaaf, and N. Piterman. On automation of CTL* verification for infinite-state systems. In *CAV '15, Part I*, pages 13–29, 2015.

N. Dershowitz. Orderings for term-rewriting systems. *Theoretical Computer Science*, 17(3):279–301, 1982.

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

C. Fuhs, J. Giesl, A. Middeldorp, P. Schneider-Kamp, R. Thiemann, and H. Zankl. SAT solving for termination analysis with polynomial interpretations. In *SAT '07*, pages 340–354, 2007.

References IV

- C. Fuhs, J. Giesl, A. Middeldorp, P. Schneider-Kamp, R. Thiemann, and H. Zankl. Maximal termination. In *RTA '08*, pages 110–125, 2008a.
- C. Fuhs, R. Navarro-Marset, C. Otto, J. Giesl, S. Lucas, and P. Schneider-Kamp. Search techniques for rational polynomial orders. In *AISC '08*, pages 109–124, 2008b.
- C. Fuhs, J. Giesl, M. Plücker, P. Schneider-Kamp, and S. Falke. Proving termination of integer term rewriting. In *RTA '09*, pages 32–47, 2009.
- J. Giesl, R. Thiemann, P. Schneider-Kamp, and S. Falke. Mechanizing and improving dependency pairs. *Journal of Automated Reasoning*, 37 (3):155–203, 2006.

References V

- J. Giesl, M. Raffelsieper, P. Schneider-Kamp, S. Swiderski, and R. Thiemann. Automated termination proofs for Haskell by term rewriting. *ACM TOPLAS*, 33(2):1–39, 2011.
- J. Giesl, T. Ströder, P. Schneider-Kamp, F. Emmes, and C. Fuhs. Symbolic evaluation graphs and term rewriting: A general methodology for analyzing logic programs. In *PPDP '12*, pages 1–12, 2012.
- A. Gupta, T. A. Henzinger, R. Majumdar, A. Rybalchenko, and R.-G. Xu. Proving non-termination. In *POPL '08*, pages 147–158, 2008.
- N. Hirokawa and A. Middeldorp. Tyrolean Termination Tool: Techniques and features. *Information and Computation*, 205(4): 474–511, 2007.
- N. Hirokawa and G. Moser. Automated complexity analysis based on the dependency pair method. In *IJCAR '08*, pages 364–379, 2008.

References VI

J. Hoffmann and Z. Shao. Type-based amortized resource analysis with integers and arrays. *Journal of Functional Programming*, 25, 2015.

H. Hong and D. Jakuš. Testing positiveness of polynomials. *Journal of Automated Reasoning*, 21(1):23–38, 1998.

D. E. Knuth and P. B. Bendix. Simple word problems in universal algebras. *Computational Problems in Abstract Algebra*, pages 263–297, 1970.

C. Kop and N. Nishida. Term rewriting with logical constraints. In *FroCoS '13*, pages 343–358, 2013.

A. Koprowski and J. Waldmann. Max/plus tree automata for termination of term rewriting. *Acta Cybernetica*, 19(2):357–392, 2009.

References VII

K. Korovin and A. Voronkov. Orienting rewrite rules with the Knuth-Bendix order. *Information and Computation*, 183(2):165–186, 2003.

D. Lankford. On proving term rewriting systems are Noetherian. Technical Report MTP-3, Louisiana Technical University, Ruston, LA, USA, 1979.

D. Larraz, A. Oliveras, E. Rodríguez-Carbonell, and A. Rubio. Proving termination of imperative programs using Max-SMT. In *FMCAD '13*, pages 218–225, 2013.

S. Lucas. Polynomials over the reals in proofs of termination: from theory to practice. *RAIRO - Theoretical Informatics and Applications*, 39(3):547–586, 2005.

References VIII

- L. Noschinski, F. Emmes, and J. Giesl. Analyzing innermost runtime complexity of term rewriting by dependency pairs. *Journal of Automated Reasoning*, 51(1):27–56, 2013.
- C. Otto, M. Brockschmidt, C. v. Essen, and J. Giesl. Automated termination analysis of Java Bytecode by term rewriting. In *RTA '10*, pages 259–276, 2010.
- A. Podelski and A. Rybalchenko. A complete method for the synthesis of linear ranking functions. In *VMCAI '04*, pages 239–251, 2004.
- C. Rocha, J. Meseguer, and C. A. Muñoz. Rewriting modulo SMT and open system analysis. In *WRLA '14*, pages 247–262, 2014.
- T. Ströder, J. Giesl, M. Brockschmidt, F. Frohn, C. Fuhs, J. Hensel, and P. Schneider-Kamp. Proving termination and memory safety for programs with pointer arithmetic. In *IJCAR '14*, pages 208–223, 2014.

References IX

- A. M. Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, 42(2):230–265, 1936.
- A. M. Turing. Checking a large routine. In *Report of a Conference on High Speed Automatic Calculating Machines*, pages 67–69, 1949.
- H. Zankl and A. Middeldorp. Satisfiability of non-linear (ir)rational arithmetic. In *LPAR (Dakar) '10*, pages 481–500, 2010.
- H. Zankl, N. Hirokawa, and A. Middeldorp. KBO orientability. *Journal of Automated Reasoning*, 43(2):173–201, 2009.