

SMT Techniques and Solvers in Automated Termination Analysis

Carsten Fuhs

Birkbeck, University of London

19th August 2016

Danish Static Analysis Symposium (DANSAS)
Odense, Denmark

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.

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.

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!

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 x > 0:  
    x = x - 1
```

Termination analysis, in the era of automation

Question: Does program P terminate?

Termination analysis, in the era of automation

Question: Does program P terminate?

Approach:

Encode termination proof template to logical constraint φ , ask SMT solver

Termination analysis, in the era of automation

Question: Does program P terminate?

Approach:

Encode termination proof template to logical constraint φ , ask SMT solver

→ **SMT** = **SAT**isfiability **M**odulo **T**heories, solve constraints like

$$4ab - 7b^2 > 1 \quad \vee \quad 3a + c \geq b^3$$

Termination analysis, in the era of automation

Question: Does program P terminate?

Approach:

Encode termination proof template to logical constraint φ , ask SMT solver

→ **SMT = SAT**isfiability **M**odulo **T**heories, solve constraints like

$$4ab - 7b^2 > 1 \quad \vee \quad 3a + c \geq b^3$$

Answer:

Termination analysis, in the era of automation

Question: Does program P terminate?

Approach:

Encode termination proof template to logical constraint φ , ask SMT solver

→ **SMT** = **SAT**isfiability **M**odulo **T**heories, solve constraints like

$$4ab - 7b^2 > 1 \quad \vee \quad 3a + c \geq b^3$$

Answer:

① φ **satisfiable**, model M :

⇒ P terminating, M fills in the gaps in the termination proof

Termination analysis, in the era of automation

Question: Does program P terminate?

Approach:

Encode termination proof template to logical constraint φ , ask SMT solver

→ **SMT = SAT**isfiability **M**odulo **T**heories, solve constraints like

$$4ab - 7b^2 > 1 \quad \vee \quad 3a + c \geq b^3$$

Answer:

- 1 φ **satisfiable**, model M :
⇒ P terminating, M fills in the gaps in the termination proof
- 2 φ **unsatisfiable**:
⇒ termination status of P unknown
⇒ try a different template (proof technique)

Termination analysis, in the era of automation

Question: Does program P terminate?

Approach:

Encode termination proof template to logical constraint φ , ask SMT solver

→ **SMT** = **SAT**isfiability **M**odulo **T**heories, solve constraints like

$$4ab - 7b^2 > 1 \quad \vee \quad 3a + c \geq b^3$$

Answer:

- 1 φ **satisfiable**, model M :
⇒ P terminating, M fills in the gaps in the termination proof
- 2 φ **unsatisfiable**:
⇒ termination status of P unknown
⇒ try a different template (proof technique)

In practice:

- Encode only a proof **step** at a time
→ 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
⇒ Termination of TRS implies termination of P

Why care about termination of term rewriting?

- Termination needed by theorem provers
- Translate program P with inductive data structures (trees) to TRS
⇒ Termination of TRS implies termination of P
 - Logic programming: **Prolog**
[van Raamsdonk, *ICLP '97*; Schneider-Kamp et al, *TOCL '09*;
Giesl et al, *PPDP '12*]

Why care about termination of term rewriting?

- Termination needed by theorem provers
- Translate program P with inductive data structures (trees) to TRS
⇒ Termination of TRS implies termination of P
 - Logic programming: **Prolog**
[van Raamsdonk, *ICLP '97*; Schneider-Kamp et al, *TOCL '09*;
Giesl et al, *PPDP '12*]
 - (Lazy) functional programming: **Haskell** [Giesl et al, *TOPLAS '11*]

Why care about termination of term rewriting?

- Termination needed by theorem provers
- Translate program P with inductive data structures (trees) to TRS
⇒ Termination of TRS implies termination of P
 - Logic programming: **Prolog**
[van Raamsdonk, *ICLP '97*; Schneider-Kamp et al, *TOCL '09*;
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)

$$\mathcal{R} = \left\{ \begin{array}{ll} \text{minus}(x, 0) & \rightarrow x \\ \text{minus}(s(x), s(y)) & \rightarrow \text{minus}(x, y) \\ \text{quot}(0, s(y)) & \rightarrow 0 \\ \text{quot}(s(x), s(y)) & \rightarrow s(\text{quot}(\text{minus}(x, y), s(y))) \end{array} \right.$$

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

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

Example (Division)

$$\mathcal{R} = \left\{ \begin{array}{ll} \text{minus}(x, 0) & \rightarrow x \\ \text{minus}(s(x), s(y)) & \rightarrow \text{minus}(x, y) \\ \text{quot}(0, s(y)) & \rightarrow 0 \\ \text{quot}(s(x), s(y)) & \rightarrow s(\text{quot}(\text{minus}(x, y), s(y))) \end{array} \right.$$

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

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

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

Example (Division)

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

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

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

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

Show termination using Dependency Pairs

Example (Division)

$$\mathcal{R} = \left\{ \begin{array}{ll} \text{minus}(x, 0) & \rightarrow x \\ \text{minus}(s(x), s(y)) & \rightarrow \text{minus}(x, y) \\ \text{quot}(0, s(y)) & \rightarrow 0 \\ \text{quot}(s(x), s(y)) & \rightarrow s(\text{quot}(\text{minus}(x, y), s(y))) \end{array} \right.$$

Dependency Pairs [Arts, Giesl, TCS '00]

Example (Division)

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

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

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})

Example (Division)

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

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

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):

Example (Division)

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

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

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 $\mathcal{DP} \neq \emptyset$:

Example (Division)

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

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

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 $\mathcal{DP} \cup \mathcal{R} \subseteq \succ$

Example (Division)

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

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

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 $s \rightarrow t$ with $s \succ t$ from \mathcal{DP}

Example (Division)

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

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

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 $s \rightarrow t$ 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]
→ ranking functions for rewriting

Example

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

Polynomial interpretations

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

Example

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

Use $[\cdot]$ with

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

Polynomial interpretations

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

Example

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

Use $[\cdot]$ with

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

Extend to terms:

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

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

Example (Constraints for Division)

$$\mathcal{R} = \left\{ \begin{array}{ll} \text{minus}(x, 0) & \lambda \gamma \quad x \\ \text{minus}(s(x), s(y)) & \lambda \lambda \lambda \gamma \quad \text{minus}(x, y) \\ \text{quot}(0, s(y)) & \lambda \lambda \lambda \gamma \quad 0 \\ \text{quot}(s(x), s(y)) & \lambda \gamma \quad s(\text{quot}(\text{minus}(x, y), s(y))) \end{array} \right.$$

$$\mathcal{DP} = \left\{ \begin{array}{ll} \text{minus}^\#(s(x), s(y)) & (\lambda) \gamma \quad \text{minus}^\#(x, y) \\ \text{quot}^\#(s(x), s(y)) & (\lambda) \lambda \gamma \quad \text{minus}^\#(x, y) \\ \text{quot}^\#(s(x), s(y)) & (\lambda) \gamma \lambda \gamma \quad \text{quot}^\#(\text{minus}(x, y), s(y)) \end{array} \right.$$

Example (Constraints for Division)

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

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

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

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

$$\begin{aligned} [\text{quot}](x_1, x_2) &= x_1 + x_2 \\ [\text{minus}](x_1, x_2) &= x_1 \\ [s](x_1) &= x_1 + 1 \end{aligned}$$

↪ order solves all constraints

Example (Constraints for Division)

$$\mathcal{R} = \left\{ \begin{array}{l} \text{minus}(x, 0) \quad \lambda \quad x \\ \text{minus}(s(x), s(y)) \quad \lambda \lambda \quad \text{minus}(x, y) \\ \text{quot}(0, s(y)) \quad \lambda \lambda \lambda \quad 0 \\ \text{quot}(s(x), s(y)) \quad \lambda \lambda \lambda \lambda \quad s(\text{quot}(\text{minus}(x, y), s(y))) \end{array} \right.$$

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

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

$[\text{quot}^\#](x_1, x_2) = x_1 + x_2$	$[\text{quot}](x_1, x_2) = x_1 + x_2$
$[\text{minus}^\#](x_1, x_2) = x_1$	$[\text{minus}](x_1, x_2) = x_1$
$[0] = 0$	$[s](x_1) = x_1 + 1$

- ↪ order solves all constraints
- ↪ $\mathcal{DP} = \emptyset$
- ↪ **termination** of division algorithm **proved**



Task: Solve

$$\text{minus}(s(x), s(y)) \rightsquigarrow \text{minus}(x, y)$$

Task: Solve $\text{minus}(s(x), s(y)) \approx \text{minus}(x, y)$

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

$$[\text{minus}](x, y) = a_m + b_m x + c_m y, \quad [s](x) = a_s + b_s x$$

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

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

$$[\text{minus}](x, y) = a_m + b_m x + c_m y, \quad [s](x) = a_s + b_s x$$

- 2 From term constraint to polynomial constraint:

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

Here: $\forall x, y. (a_s b_m + a_s c_m) + (b_s b_m - b_m) x + (b_s c_m - c_m) y \geq 0$

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

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

$$[\text{minus}](x, y) = a_m + b_m x + c_m y, \quad [s](x) = a_s + b_s x$$

- 2 From term constraint to polynomial constraint:

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

Here: $\forall x, y. (a_s b_m + a_s c_m) + (b_s b_m - b_m) x + (b_s c_m - c_m) y \geq 0$

- 3 Eliminate $\forall x, y$ by **absolute positiveness criterion**

[Hong, Jakuš, JAR '98]:

Here: $a_s b_m + a_s c_m \geq 0 \wedge b_s b_m - b_m \geq 0 \wedge b_s c_m - c_m \geq 0$

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

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

$$[\text{minus}](x, y) = a_m + b_m x + c_m y, \quad [s](x) = a_s + b_s x$$

- 2 From term constraint to polynomial constraint:

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

Here: $\forall x, y. (a_s b_m + a_s c_m) + (b_s b_m - b_m) x + (b_s c_m - c_m) y \geq 0$

- 3 Eliminate $\forall x, y$ by **absolute positiveness criterion**

[Hong, Jakuš, JAR '98]:

Here: $a_s b_m + a_s c_m \geq 0 \wedge b_s b_m - b_m \geq 0 \wedge b_s c_m - c_m \geq 0$

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

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

$$[\text{minus}](x, y) = a_m + b_m x + c_m y, \quad [s](x) = a_s + b_s x$$

- 2 From term constraint to polynomial constraint:

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

Here: $\forall x, y. (a_s b_m + a_s c_m) + (b_s b_m - b_m) x + (b_s c_m - c_m) y \geq 0$

- 3 Eliminate $\forall x, y$ by **absolute positiveness criterion**

[Hong, Jakuš, JAR '98]:

Here: $a_s b_m + a_s c_m \geq 0 \wedge b_s b_m - b_m \geq 0 \wedge b_s c_m - c_m \geq 0$

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

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

$$[\text{minus}](x, y) = a_m + b_m x + c_m y, \quad [s](x) = a_s + b_s x$$

- 2 From term constraint to polynomial constraint:

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

Here: $\forall x, y. (a_s b_m + a_s c_m) + (b_s b_m - b_m) x + (b_s c_m - c_m) y \geq 0$

- 3 Eliminate $\forall x, y$ by **absolute positiveness criterion**

[Hong, Jakuš, JAR '98]:

Here: $a_s b_m + a_s c_m \geq 0 \wedge b_s b_m - b_m \geq 0 \wedge b_s c_m - c_m \geq 0$

Non-linear constraints, even for **linear** interpretations

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

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

$$[\text{minus}](x, y) = a_m + b_m x + c_m y, \quad [s](x) = a_s + b_s x$$

- 2 From term constraint to polynomial constraint:

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

Here: $\forall x, y. (a_s b_m + a_s c_m) + (b_s b_m - b_m) x + (b_s c_m - c_m) y \geq 0$

- 3 Eliminate $\forall x, y$ by **absolute positiveness criterion**

[Hong, Jakuš, JAR '98]:

Here: $a_s b_m + a_s c_m \geq 0 \wedge b_s b_m - b_m \geq 0 \wedge b_s c_m - c_m \geq 0$

Non-linear constraints, even for **linear** interpretations

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

\rightsquigarrow **Prove termination** of given term rewrite system

- Polynomials with **negative coefficients** and **max-operator**
[Hirokawa, Middeldorp, *IC '07*; Fuhs et al, *SAT '07*, *RTA '08*]
 - can model behavior of functions more closely:
[pred](x_1) = $\max(x_1 - 1, 0)$
 - automation via encoding to non-linear constraints, more complex Boolean structure

- Polynomials with **negative coefficients** and **max-operator** [Hirokawa, Middeldorp, *IC '07*; Fuhs et al, *SAT '07*, *RTA '08*]
 - can model behavior of functions more closely:
 $[\text{pred}](x_1) = \max(x_1 - 1, 0)$
 - automation via encoding to non-linear constraints, 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*]

- Polynomials with **negative coefficients** and **max-operator** [Hirokawa, Middeldorp, *IC '07*; Fuhs et al, *SAT '07, RTA '08*]
 - can model behavior of functions more closely:
 $[\text{pred}](x_1) = \max(x_1 - 1, 0)$
 - automation via encoding to non-linear constraints, 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*]
 - linear interpretation to vectors over \mathbb{N}^k , coefficients are matrices
 - useful for deeply nested terms
 - automation: constraints with more complex atoms
 - several flavors: plus-times-semiring, max-plus-semiring [Koprowski, Waldmann, *Acta Cyb. '09*], ...

- Polynomials with **negative coefficients** and **max-operator** [Hirokawa, Middeldorp, *IC '07*; Fuhs et al, *SAT '07, RTA '08*]
 - can model behavior of functions more closely:
 $[\text{pred}](x_1) = \max(x_1 - 1, 0)$
 - automation via encoding to non-linear constraints, 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*]
 - linear interpretation to vectors over \mathbb{N}^k , coefficients are matrices
 - useful for deeply nested terms
 - automation: constraints with more complex atoms
 - several flavors: plus-times-semiring, max-plus-semiring [Koprowski, Waldmann, *Acta Cyb. '09*], ...
- ...

- Proving **non**-termination (an infinite run is possible) [Giesl, Thiemann, Schneider-Kamp, *FroCoS '05*; Payet, *TCS '08*; Zankl et al, *SOFSEM '10*; ...]

Further extensions

- Proving **non**-termination (an infinite run is possible) [Giesl, Thiemann, Schneider-Kamp, *FroCoS '05*; Payet, *TCS '08*; Zankl et al, *SOFSEM '10*; ...]
- **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

- Proving **non**-termination (an infinite run is possible) [Giesl, Thiemann, Schneider-Kamp, *FroCoS '05*; Payet, *TCS '08*; Zankl et al, *SOFSEM '10*; ...]
- **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 (Quantifier-Free Non-linear Integer Arithmetic)

Year	Winner
2009	Barcellogic-QF_NIA
2010	MiniSmt
2011	AProVE
2012	<i>no QF_NIA</i>
2013	<i>no SMT-COMP</i>
2014	AProVE
2015	AProVE
2016	Yices

SMT solvers *from* termination analysis

Annual SMT-COMP, division QF_NIA (Quantifier-Free Non-linear Integer Arithmetic)

Year	Winner
2009	Barcellogic-QF_NIA
2010	MiniSmt (spin-off of $T_T T_2$)
2011	AProVE
2012	<i>no QF_NIA</i>
2013	<i>no SMT-COMP</i>
2014	AProVE
2015	AProVE
2016	Yices

⇒ Termination provers can also be successful SMT solvers!

SMT solvers *from* termination analysis

Annual SMT-COMP, division QF_NIA (Quantifier-Free Non-linear Integer Arithmetic)

Year	Winner
2009	Barcellogic-QF_NIA
2010	MiniSmt (spin-off of T _T T ₂)
2011	AProVE
2012	<i>no QF_NIA</i>
2013	<i>no SMT-COMP</i>
2014	AProVE
2015	AProVE
2016	Yices

⇒ **Termination provers** can also be successful SMT solvers!

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

1 Term Rewrite Systems (TRSs)

2 Imperative Programs

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  $x \geq 0$ :  
    while  $x \neq 0$ :  
         $x = x - 1$ 
```

Does this program terminate?

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

Example (Imperative program)

```
 $\ell_0$ :  if  $x \geq 0$ :  
 $\ell_1$ :      while  $x \neq 0$ :  
 $\ell_2$ :           $x = x - 1$ 
```

Does this program terminate?

Example (Equivalent translation to transition system)

```
 $\ell_0(x) \rightarrow \ell_1(x) \quad [x \geq 0]$   
 $\ell_1(x) \rightarrow \ell_2(x) \quad [x \neq 0]$   
 $\ell_2(x) \rightarrow \ell_1(x - 1)$   
 $\ell_1(x) \rightarrow \ell_3(x) \quad [x == 0]$ 
```

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

Example (Imperative program)

```
 $\ell_0$ :   if  $x \geq 0$ :  
 $\ell_1$ :       while  $x \neq 0$ :  
 $\ell_2$ :            $x = x - 1$ 
```

Does this program terminate?

Example (Equivalent translation to transition system)

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

Oh no! $\ell_1(-1) \rightarrow \ell_2(-1) \rightarrow \ell_1(-2) \rightarrow \ell_2(-2) \rightarrow \ell_1(-3) \rightarrow \dots$

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)

$$\begin{aligned}l_0(x) &\rightarrow l_1(x) && [x \geq 0] \\l_1(x) &\rightarrow l_2(x) && [x \neq 0] \\l_2(x) &\rightarrow l_1(x - 1) \\l_1(x) &\rightarrow l_3(x) && [x == 0]\end{aligned}$$

Oh no! $l_1(-1) \rightarrow l_2(-1) \rightarrow l_1(-2) \rightarrow l_2(-2) \rightarrow l_1(-3) \rightarrow \dots$

⇒ **Restrict initial states** to $l_0(z)$ 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)

$$\begin{array}{lll} l_0(x) & \rightarrow & l_1(x) \quad [x \geq 0] \\ l_1(x) & \rightarrow & l_2(x) \quad [x \neq 0] \\ l_2(x) & \rightarrow & l_1(x - 1) \\ l_1(x) & \rightarrow & l_3(x) \quad [x == 0] \end{array}$$

Oh no! $l_1(-1) \rightarrow l_2(-1) \rightarrow l_1(-2) \rightarrow l_2(-2) \rightarrow l_1(-3) \rightarrow \dots$

⇒ **Restrict initial states** to $l_0(z)$ for $z \in \mathbb{Z}$

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

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)

$$\begin{array}{lll} l_0(x) & \rightarrow & l_1(x) \quad [x \geq 0] \\ l_1(x) & \rightarrow & l_2(x) \quad [x \neq 0 \wedge x \geq 0] \\ l_2(x) & \rightarrow & l_1(x - 1) \quad [x \geq 0] \\ l_1(x) & \rightarrow & l_3(x) \quad [x == 0 \wedge x \geq 0] \end{array}$$

Oh no! $l_1(-1) \rightarrow l_2(-1) \rightarrow l_1(-2) \rightarrow l_2(-2) \rightarrow l_1(-3) \rightarrow \dots$

⇒ **Restrict initial states** to $l_0(z)$ for $z \in \mathbb{Z}$

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

Proving termination with invariants

Example (Transition system with invariants)

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

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

Proving termination with invariants

Example (Transition system with invariants)

$l_0(x)$	\rightsquigarrow	$l_1(x)$	$[x \geq 0]$
$l_1(x)$	\rightsquigarrow	$l_2(x)$	$[x \neq 0 \wedge x \geq 0]$
$l_2(x)$	γ	$l_1(x - 1)$	$[x \geq 0]$
$l_1(x)$	\rightsquigarrow	$l_3(x)$	$[x == 0 \wedge x \geq 0]$

Prove termination by ranking function $[\cdot]$ with $[l_0](x) = [l_1](x) = \dots = x$

Proving termination with invariants

Example (Transition system with invariants)

$\ell_0(x)$	\rightsquigarrow	$\ell_1(x)$	$[x \geq 0]$
$\ell_1(x)$	\rightsquigarrow	$\ell_2(x)$	$[x \neq 0 \wedge x \geq 0]$
$\ell_2(x)$	\rightsquigarrow	$\ell_1(x - 1)$	$[x \geq 0]$
$\ell_1(x)$	\rightsquigarrow	$\ell_3(x)$	$[x == 0 \wedge x \geq 0]$

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

Automate search using **parametric** ranking function:

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

Proving termination with invariants

Example (Transition system with invariants)

$\ell_0(x)$	\rightsquigarrow	$\ell_1(x)$	$[x \geq 0]$
$\ell_1(x)$	\rightsquigarrow	$\ell_2(x)$	$[x \neq 0 \wedge x \geq 0]$
$\ell_2(x)$	\rightsquigarrow	$\ell_1(x - 1)$	$[x \geq 0]$
$\ell_1(x)$	\rightsquigarrow	$\ell_3(x)$	$[x == 0 \wedge x \geq 0]$

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

Automate search using **parametric** ranking function:

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

Constraints e.g.:

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

Proving termination with invariants

Example (Transition system with invariants)

$\ell_0(x)$	\rightsquigarrow	$\ell_1(x)$	$[x \geq 0]$
$\ell_1(x)$	\rightsquigarrow	$\ell_2(x)$	$[x \neq 0 \wedge x \geq 0]$
$\ell_2(x)$	\rightsquigarrow	$\ell_1(x - 1)$	$[x \geq 0]$
$\ell_1(x)$	\rightsquigarrow	$\ell_3(x)$	$[x == 0 \wedge x \geq 0]$

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

Automate search using **parametric** ranking function:

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

Constraints e.g.:

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

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

Proving termination with invariants

Example (Transition system with invariants)

$\ell_0(x)$	\rightsquigarrow	$\ell_1(x)$	$[x \geq 0]$
$\ell_1(x)$	\rightsquigarrow	$\ell_2(x)$	$[x \neq 0 \wedge x \geq 0]$
$\ell_2(x)$	\rightsquigarrow	$\ell_1(x - 1)$	$[x \geq 0]$
$\ell_1(x)$	\rightsquigarrow	$\ell_3(x)$	$[x == 0 \wedge x \geq 0]$

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

Automate search using **parametric** ranking function:

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

Constraints e.g.:

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

Use Farkas' Lemma to eliminate $\forall x$, solver for **linear** constraints 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

Searching for invariants using SMT

Termination prover needs to find invariants for programs on integers

- Statically before the translation

[Otto et al., *RTA '10*; Ströder et al, *IJCAR '14*, ...]

→ abstract interpretation [Cousot, Cousot, *POPL '77*]

Searching for invariants using SMT

Termination prover needs to find invariants for programs on integers

- Statically before the translation
[Otto et al., *RTA '10*; Ströder et al, *IJCAR '14*, ...]
→ abstract interpretation [Cousot, Cousot, *POPL '77*]
- By counterexample-based reasoning + safety prover: **Terminator**
[Cook, Podelski, Rybalchenko, *CAV '06*, *PLDI '06*]
→ prove termination of single program **runs**
→ termination argument often generalizes

Searching for invariants using SMT

Termination prover needs to find invariants for programs on integers

- Statically before the translation
[Otto et al., *RTA '10*; Ströder et al, *IJCAR '14*, ...]
→ abstract interpretation [Cousot, Cousot, *POPL '77*]
- By counterexample-based reasoning + safety prover: **Terminator**
[Cook, Podelski, Rybalchenko, *CAV '06*, *PLDI '06*]
→ prove termination of single program **runs**
→ termination argument often generalizes
- ... also cooperating with removal of terminating **rules** (as for TRSs):
T2 [Brockschmidt, Cook, Fuhs, *CAV '13*]

Searching for invariants using SMT

Termination prover needs to find invariants for programs on integers

- Statically before the translation
[Otto et al., *RTA '10*; Ströder et al, *IJCAR '14*, ...]
→ abstract interpretation [Cousot, Cousot, *POPL '77*]
- By counterexample-based reasoning + safety prover: **Terminator**
[Cook, Podelski, Rybalchenko, *CAV '06*, *PLDI '06*]
→ prove termination of single program **runs**
→ termination argument often generalizes
- ... also cooperating with removal of terminating **rules** (as for TRSs):
T2 [Brockschmidt, Cook, Fuhs, *CAV '13*]
- Using Max-SMT
[Larraz, Oliveras, Rodríguez-Carbonell, Rubio, *FMCAD '13*]

Searching for invariants using SMT

Termination prover needs to find invariants for programs on integers

- Statically before the translation
[Otto et al., *RTA '10*; Ströder et al, *IJCAR '14*, ...]
→ abstract interpretation [Cousot, Cousot, *POPL '77*]
- By counterexample-based reasoning + safety prover: **Terminator**
[Cook, Podelski, Rybalchenko, *CAV '06*, *PLDI '06*]
→ prove termination of single program **runs**
→ termination argument often generalizes
- ... also cooperating with removal of terminating **rules** (as for TRSs):
T2 [Brockschmidt, Cook, Fuhs, *CAV '13*]
- Using Max-SMT
[Larraz, Oliveras, Rodríguez-Carbonell, Rubio, *FMCAD '13*]

Nowadays all SMT-based!

- Proving **non-termination** (infinite run is possible **from initial states**)
[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*, ...]

- Proving **non-termination** (infinite run is possible **from initial states**)
[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*, ...]
- Complexity bounds
[Alias et al, *SAS '10*, Hoffmann, Shao, *JFP '15*, Brockschmidt et al, *TOPLAS '16*, ...]

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

- Proving **non-termination** (infinite run is possible **from initial states**)
[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*, ...]
- Complexity bounds
[Alias et al, *SAS '10*, Hoffmann, Shao, *JFP '15*, Brockschmidt et al, *TOPLAS '16*, ...]
- CTL* model checking for **infinite** state systems based on termination and non-termination provers
[Cook, Khlaaf, Piterman, *CAV '15*]
- Beyond sequential programs on integers:
 - structs/classes [Berdine et al, *CAV '06*; Otto et al, *RTA '10*; ...]
 - arrays (pointer arithmetic) [Ströder et al, *IJCAR '14*, ...]
 - multi-threaded programs [Cook et al, *PLDI '07*, ...]
 - ...

Conclusion

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

Conclusion

- Automated termination analysis for term rewriting and for imperative programs developed in parallel over the last ~ 15 years
- Term rewriting: need to encode how to represent data structures
- Imperative programs: 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: 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 ~ 15 years
- Term rewriting: need to encode how to represent data structures
- Imperative programs: 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: 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
- More information . . .

<http://termination-portal.org>

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: 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
- More information . . . e.g., on the annual Termination Competition:

http://termination-portal.org/wiki/Termination_Competition

→ next on 5/6 Sep 2016 at Workshop on Termination, Obergurgl

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: 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
- More information ... e.g., on the annual Termination Competition:

http://termination-portal.org/wiki/Termination_Competition

→ next on 5/6 Sep 2016 at Workshop on Termination, Obergurgl

Behind (almost) every successful termination prover ...

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: 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
- More information ... e.g., on the annual Termination Competition:

http://termination-portal.org/wiki/Termination_Competition

→ next on 5/6 Sep 2016 at Workshop on Termination, Obergurgl

**Behind (almost) every successful termination prover ...
... there is a powerful SMT solver!**

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.
-  J. Berdine, B. Cook, D. Distefano, and P. W. O'Hearn. Automatic termination proofs for programs with shape-shifting heaps. In *CAV '06*, pages 386–400, 2006.
-  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 `NullPointerException`s for Java Bytecode. In *FoVeOOS '11*, pages 123–141, 2012.

References II

-  M. Brockschmidt, B. Cook, and C. Fuhs. Better termination proving through cooperation. In *CAV '13*, pages 413–429, 2013.
-  M. Brockschmidt, F. Emmes, S. Falke, C. Fuhs, and J. Giesl. Analyzing runtime and size complexity of integer programs. *ACM TOPLAS*, 38(4), 2016.
-  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.
-  B. Cook, A. Podelski, and A. Rybalchenko. Terminator: Beyond safety. In *CAV '06*, pages 415–418, 2006a.
-  B. Cook, A. Podelski, and A. Rybalchenko. Termination proofs for systems code. In *PLDI '06*, pages 415–426, 2006b.
-  B. Cook, A. Podelski, and A. Rybalchenko. Proving thread termination. In *PLDI '07*, pages 320–330, 2007.

-  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.
-  P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *POPL '77*, pages 238–252, 1977.
-  J. Endrullis, J. Waldmann, and H. Zantema. Matrix interpretations for proving termination of term rewriting. *Journal of Automated Reasoning*, 40(2–3):195–220, 2008.

References IV

-  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.
-  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, and P. Schneider-Kamp. Proving and disproving termination of higher-order functions. In *FroCoS '05*, pages 216–231, 2005.
-  J. Giesl, R. Thiemann, P. Schneider-Kamp, and S. Falke. Mechanizing and improving dependency pairs. *Journal of Automated Reasoning*, 37(3):155–203, 2006.
-  J. Giesl, 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.
-  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.
-  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.

-  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.
-  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.
-  É. Payet. Loop detection in term rewriting using the eliminating unfoldings. *Theoretical Computer Science*, 403(2-3), 2008.
-  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.
-  P. Schneider-Kamp, J. Giesl, A. Serebrenik, and R. Thiemann. Automated termination proofs for logic programs by term rewriting. *ACM TOCL*, 11(1):1–52, 2009.

References IX

-  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.
-  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.
-  F. van Raamsdonk. Translating logic programs into conditional rewriting systems. In *ICLP '97*, pages 168–182, 1997.
-  H. Zankl and A. Middeldorp. Satisfiability of non-linear (ir)rational arithmetic. In *LPAR (Dakar) '10*, pages 481–500, 2010.

-  H. Zankl, C. Sternagel, D. Hofbauer, and A. Middeldorp. Finding and certifying loops. In *SOFSEM '10*, pages 755–766, 2010.