SMT-Based Techniques in Automated Termination Analysis

Carsten Fuhs

Birkbeck, University of London

6$^{\text{th}}$ September 2016

15$^{\text{th}}$ Workshop on Termination
Obergurgl, Austria

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

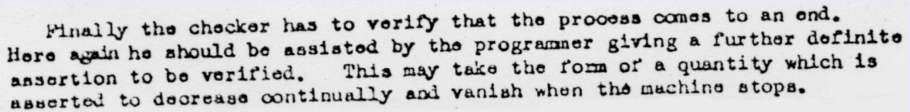
Finally the checker has to verify that the process comes to an end. Here again he should be assisted by the programmer giving a further definite assertion to be verified. This may take the form of a quantity which is asserted to decrease continually and vanish when the machine stops.

"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
$\rightarrow$ **SMT = SAT**isfiability **M**odulo **T**heories, solve constraints like

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

Termination analysis, in the era of automation

Question: Does program P terminate?

Approach:
Encode termination proof template to logical constraint φ, ask SMT solver
$\rightarrow$ **SMT** = **SAT**isfiability **M**odulo **T**heories, solve constraints like

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

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
$\rightarrow$ **SMT** = **SAT**isfiability **M**odulo **T**heories, solve constraints like

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

Answer:

1. φ satisfiable, model M:
   $\Rightarrow$ 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
$\rightarrow$ **SMT** = **SAT**isfiability **M**odulo **T**heories, solve constraints like

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

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 automation

Question: Does program P terminate?

Approach:
Encode termination proof template to logical constraint φ, ask SMT solver
$\rightarrow$ **SMT** = **SAT**isfiability **M**odulo **T**heories, solve constraints like

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

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, ...)

Example (Division)

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

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

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

Example (Division)

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

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

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

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

Example (Division)

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

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

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

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

Example (Division)

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

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)

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

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

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

- For TRS $\mathcal{R}$ build dependency pairs $\mathcal{DP}$ (∼ 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)

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

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

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)

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

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

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

- For TRS $\mathcal{R}$ build dependency pairs $\mathcal{DP}$ (∼ 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 '75]
$\rightarrow$ ranking functions for rewriting

Example

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

Polynomial interpretations

Get $\succ$ via polynomial interpretations $[\,\cdot\,]$ over $\mathbb{N}$ [Lankford '75]

$\to$ ranking functions for rewriting

Example

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

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 '75]
$\to$ 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:

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

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

Example (Constraints for Division)

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

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

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

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

$\curvearrowright$ order solves all constraints

Example (Constraints for Division)

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

$$\mathcal{DP} = \left\{ \vphantom{\begin{array}{c} a \\ b \\ c \end{array}} \right.$$

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

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

$\curvearrowright$ order solves all constraints

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

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

Remark

Polynomial interpretations play several roles for program analysis:

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

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

$\curvearrowright$ order solves all constraints

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

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

Remark

Polynomial interpretations play several roles for program analysis:

- Ranking function: $[\mathsf{quot}^\sharp]$ and $[\mathsf{minus}^\sharp]$

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

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

$\curvearrowright$ order solves all constraints

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

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

Remark

Polynomial interpretations play several roles for program analysis:

- Ranking function: $[\mathsf{quot}^\sharp]$ and $[\mathsf{minus}^\sharp]$
- Summary: $[\mathsf{quot}]$ and $[\mathsf{minus}]$

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

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

$\curvearrowright$ order solves all constraints

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

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

Remark

Polynomial interpretations play several roles for program analysis:

- Ranking function: $[\mathsf{quot}^\sharp]$ and $[\mathsf{minus}^\sharp]$
- Summary: $[\mathsf{quot}]$ and $[\mathsf{minus}]$
- Abstraction: $[\mathsf{0}]$ and $[\mathsf{s}]$

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

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

$\curvearrowright$ order solves all constraints
$\curvearrowright$ $\mathcal{DP} = \emptyset$
$\curvearrowright$ termination of division algorithm proved $\square$

Automation

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

Automation

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

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

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

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:

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

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

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

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

2. From term constraint to polynomial constraint:

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

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

3. Eliminate quantifiers $\forall x, y$ 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 quantifiers $\forall x, y$ 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 $\qquad \mathsf{minus}(\mathsf{s}(x), \mathsf{s}(y)) \succsim \mathsf{minus}(x, y)$

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 quantifiers $\forall x, y$ 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 quantifiers $\forall x, y$ 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, even for linear interpretations

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

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

Non-linear constraints, 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*]
  - 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

Extensions

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

Extensions

- 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*], ...

Extensions

- 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*], ...
- ...

Recipe

Automate your own weakly monotone interpretations

1. Pick suitable well-founded algebra $(A, \geq, >)$ and operations on A

Recipe

Automate your own weakly monotone interpretations

1. Pick suitable well-founded algebra $(A, \geq, >)$ and operations on A
2. Use templates for interpretations $[\,f\,]$

Recipe

Automate your own weakly monotone interpretations

1. Pick suitable well-founded algebra $(A, \geq, >)$ and operations on A
2. Use templates for interpretations $[\, f \,]$
3. Get arithmetic inequalities

$$s \succ t \quad \curvearrowright \quad \forall \vec{x}.\ [s] > [t]$$

Recipe

Automate your own weakly monotone interpretations

1. Pick suitable well-founded algebra $(A, \geq, >)$ and operations on A
2. Use templates for interpretations $[\,f\,]$
3. Get arithmetic inequalities

$$s \succ t \quad \curvearrowright \quad \forall \vec{x}.\; [s] > [t]$$

4. Use sound quantifier elimination to remove $\forall \vec{x}$

Recipe

Automate your own weakly monotone interpretations

1. Pick suitable well-founded algebra $(A, \geq, >)$ and operations on A
2. Use templates for interpretations $[\, f \,]$
3. Get arithmetic inequalities

$$s \succ t \quad \curvearrowright \quad \forall \vec{x}.\ [s] > [t]$$

4. Use sound quantifier elimination to remove $\forall \vec{x}$
5. Feed quantifier-free SMT formula to suitable SMT solver
   (or encode to a SAT problem)

Recipe

Automate your own weakly monotone interpretations

1. Pick suitable well-founded algebra $(A, \geq, >)$ and operations on A
2. Use templates for interpretations $[\, f \,]$
3. Get arithmetic inequalities

$$s \succ t \quad \curvearrowright \quad \forall \vec{x}.\ [s] > [t]$$

4. Use sound quantifier elimination to remove $\forall \vec{x}$
5. Feed quantifier-free SMT formula to suitable SMT solver (or encode to a SAT problem)
6. Enjoy!

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 (Quantifier-Free Non-linear Integer Arithmetic)

Year	Winner
2009	Barcelogic-QF_NIA
2010	MiniSmt
2011	AProVE
2012	*no QF_NIA*
2013	*no SMT-COMP*
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	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	Yices

$\Rightarrow$ 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	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	Yices

$\Rightarrow$ 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)

$$
\begin{array}{l}
\textbf{if } x \geq 0\text{:} \\
\quad \textbf{while } x \neq 0\text{:} \\
\quad\quad x = x - 1
\end{array}
$$

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)

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

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

$\Rightarrow$ **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)

$$
\begin{array}{ll}
\ell_0\text{:} & \textbf{if } \mathsf{x} \geq 0\text{:} \\
\ell_1\text{:} & \quad \textbf{while } \mathsf{x} \neq 0\text{:} \\
\ell_2\text{:} & \quad\quad \mathsf{x} = \mathsf{x} - 1
\end{array}
$$

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)

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

Does this program terminate?

Example (Equivalent translation to transition system)

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

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

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

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

Proving termination with invariants

Example (Transition system with invariants)

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

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

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 $[\ell_0](x) = [\ell_1](x) = \cdots = x$

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:

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

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.:

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

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:

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

Constraints e.g.:

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

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

Proving termination with invariants

Example (Transition system with invariants)

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

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.:

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

Use Farkas' Lemma to eliminate $\forall x$, 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*, . . .]

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*, . . .]

- By counterexample-based reasoning + safety prover: **Terminator**
  [Cook, Podelski, Rybalchenko, *CAV '06, PLDI '06*]
  $\rightarrow$ prove termination of single program **runs**
  $\rightarrow$ 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*, ...]

- By counterexample-based reasoning + safety prover: **Terminator**
  [Cook, Podelski, Rybalchenko, *CAV '06, PLDI '06*]
  $\rightarrow$ prove termination of single program **runs**
  $\rightarrow$ 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*, ...]
- 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*, . . .]
- By counterexample-based reasoning + safety prover: **Terminator**
  [Cook, Podelski, Rybalchenko, *CAV '06, PLDI '06*]
  $\rightarrow$ prove termination of single program **runs**
  $\rightarrow$ 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!

Extensions

- 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*, ...]

Extensions

- 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*, Flores-Montoya, Hähnle, *APLAS '14*, Sinn, Zuleger, Veith, *CAV '14*, Hoffmann, Shao, *JFP '15*, Brockschmidt et al, *TOPLAS '16*, ...]

Extensions

- 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*, Flores-Montoya, Hähnle, *APLAS '14*, Sinn, Zuleger, Veith, *CAV '14*, 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*]

Extensions

- 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*, Flores-Montoya, Hähnle, *APLAS '14*, Sinn, Zuleger, Veith, *CAV '14*, 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*, ...]
  - IEEE floating-point numbers [Maurica, Mesnard, Payet, *SAC '16*]
  - ...

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 ~ 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: 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 $\sim$ 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

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
- 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:

`http://termination-portal.org`

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:

http://termination-portal.org

Behind (almost) every successful termination prover . . .

. . . there is a powerful SMT solver!

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 `NullPointerExceptions` 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.

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.

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.

A. Flores-Montoya and R. Hähnle. Resource analysis of complex programs with cost equations. In *APLAS '14*, pages 275–295, 2014.

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, P. Schneider-Kamp, and S. Falke. Mechanizing and improving dependency pairs. *Journal of Automated Reasoning*, 37 (3):155–203, 2006.
- A. Gupta, T. A. Henzinger, R. Majumdar, A. Rybalchenko, and R.-G. Xu. Proving non-termination. In *POPL '08*, pages 147–158, 2008.

References V

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. S. Lankford. Canonical algebraic simplification in computational logic. Technical Report ATP-25, University of Texas, 1975.

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.

F. Maurica, F. Mesnard, and É. Payet. Termination analysis of floating-point programs using parameterizable rational approximations. In *SAC '16*, pages 1674–1679, 2016.

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.
- M. Sinn, F. Zuleger, and H. Veith. A simple and scalable static analysis for bound analysis and amortized complexity analysis. In *CAV '14*, pages 745–761, 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.

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.