SMT Techniques and Solvers in Automated Termination Analysis

Carsten Fuhs

Birkbeck, University of London

2nd July 2016

14th Workshop on SAT Modulo Theories (SMT)
Coimbra, Portugal
Why analyze termination?

Program: produces result
Input handler: system reacts
Mathematical proof: the induction is valid
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
Why analyze termination?

1. **Program**: produces result

Variations of the same problem:
- special case of
- can be interpreted as
- probabilistic version of
Why analyze termination?

1. **Program**: produces result
2. **Input handler**: system reacts
Why analyze termination?

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

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

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

Variations of the same problem:

2. special case of 1
3. can be interpreted as 1
4. probabilistic version of 1
The bad news

Theorem (Turing 1936)

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

Theorem (Turing 1936)

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

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

Theorem (Turing 1936)

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

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

Theorem (Turing 1936)

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

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

\[
\text{while } x > 0: x = x - 1
\]
Termination analysis, classically

Turing 1949

“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”)

Example (Termination can be simple)

```plaintext
while $x > 0$:
    $x = x - 1$
```
Termination analysis, classically

Turing 1949

“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. [...] 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 $x > 0$:
    $x = x - 1$
```
Question: Does program $P$ terminate?
Question: Does program $P$ terminate?

Approach:
Encode termination proof template to SMT formula $\varphi$, ask SMT solver
Termination analysis, in the era of SMT solvers

**Question:** Does program $P$ terminate?

**Approach:**
Encode termination proof template to SMT formula $\varphi$, ask SMT solver

**Answer:**
Question: Does program $P$ terminate?

Approach:
Encode termination proof template to SMT formula $\varphi$, ask SMT solver

Answer:

1. $\varphi$ satisfiable, model $M$:
   $\Rightarrow P$ terminating, $M$ fills in the gaps in the termination proof
Question: Does program $P$ terminate?

Approach:
Encode termination proof template to SMT formula $\varphi$, ask SMT solver

Answer:

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

2. $\varphi$ unsatisfiable:
   $\Rightarrow$ termination status of $P$ unknown
   $\Rightarrow$ try a different template (proof technique)
Termination analysis, in the era of SMT solvers

Question: Does program $P$ terminate?

Approach:
Encode termination proof template to SMT formula $\varphi$, ask SMT solver

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

2. $\varphi$ 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
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?

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, ...
What’s Term Rewriting?

Syntactic approach for reasoning in equational first-order logic
What’s Term Rewriting?

Syntactic approach for reasoning in equational first-order logic

Core functional programming language without many restrictions (and features) of “real” FP:
What’s Term Rewriting?

Syntactic approach for reasoning in equational first-order logic

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

- first-order (usually)
- no fixed evaluation strategy
- no fixed order of rules to apply (Haskell: top to bottom)
- untyped
- no pre-defined data structures (integers, arrays, . . .)
Why care about termination of term rewriting?

- Termination needed by theorem provers
Why care about termination of term rewriting?

- Termination needed by theorem provers

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

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

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

\[
\text{minus}(s(s(0)), s(0)) \rightarrow_{\mathcal{R}} \text{minus}(s(0), 0) \rightarrow_{\mathcal{R}} s(0)
\]
Example (Division)

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

Term rewriting: Evaluate terms by applying rules from \( R \)

\[ \text{minus}(s(s(0)), s(0)) \rightarrow_R \text{minus}(s(0), 0) \rightarrow_R s(0) \]

Termination: No infinite evaluation sequences \( t_1 \rightarrow_R t_2 \rightarrow_R t_3 \rightarrow_R \ldots \)
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 \( t_1 \rightarrow_{\mathcal{R}} t_2 \rightarrow_{\mathcal{R}} t_3 \rightarrow_{\mathcal{R}} \ldots \)

Show termination using Dependency Pairs
Example (Division)

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

Dependency Pairs [Arts, Giesl, TCS ’00]
Example (Division)

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

\[ \mathcal{DP} = \left\{ \begin{array}{ll}
\text{minus}^\#(s(x), s(y)) & \rightarrow \text{minus}^\#(x, y) \\
\text{quot}^\#(s(x), s(y)) & \rightarrow \text{minus}^\#(x, y) \\
\text{quot}^\#(s(x), s(y)) & \rightarrow \text{quot}^\#(\text{minus}(x, y), s(y))
\end{array} \right. \]

Dependency Pairs [Arts, Giesl, TCS ’00]

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

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

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

Dependency Pairs [Arts, Giesl, TCS ’00]

- For TRS \( \mathcal{R} \) build dependency pairs \( \mathcal{DP} \) \((\sim \text{function calls})\)
- Show: No \( \infty \) 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)

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

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

Dependency Pairs [Arts, Giesl, TCS ’00]

- For TRS \( \mathcal{R} \) build dependency pairs \( \mathcal{DP} \) (\( \sim \) function calls)
- Show: No \( \infty \) call sequence with \( \mathcal{DP} \) (eval of \( \mathcal{DP} \)'s args via \( \mathcal{R} \))
- Dependency Pair Framework [Giesl et al, JAR ’06] (simplified): 
  \hspace{1cm} \textbf{while} \ \mathcal{DP} \neq \emptyset :
Example (Division)

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

\[
\mathcal{DP} = \begin{cases} 
\text{minus}^\#(s(x), s(y)) & \sim \text{minus}^\#(x, y) \\
\text{quot}^\#(s(x), s(y)) & \sim \text{minus}^\#(x, y) \\
\text{quot}^\#(s(x), s(y)) & \sim \text{quot}^\#(\text{minus}(x, y), s(y))
\end{cases}
\]

Dependency Pairs [Arts, Giesl, TCS ’00]

- For TRS \(\mathcal{R}\) build dependency pairs \(\mathcal{DP}\) \((\sim\) function calls\)
- Show: No \(\infty\) 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)

\[ R = \begin{cases} 
\text{minus}(x, 0) & \sim x \\
\text{minus}(s(x), s(y)) & \sim \text{minus}(x, y) \\
\text{quot}(0, s(y)) & \sim 0 \\
\text{quot}(s(x), s(y)) & \sim s(\text{quot}(\text{minus}(x, y), s(y))) 
\end{cases} \]

\[ DP = \begin{cases} 
\text{minus}^\#(s(x), s(y)) & \sim \text{minus}^\#(x, y) \\
\text{quot}^\#(s(x), s(y)) & \sim \text{minus}^\#(x, y) \\
\text{quot}^\#(s(x), s(y)) & \sim \text{quot}^\#(\text{minus}(x, y), s(y)) 
\end{cases} \]

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

- For TRS \( R \) build dependency pairs \( DP \) (~ function calls)
- Show: No \( \infty \) call sequence with \( DP \) (eval of \( DP \)'s args via \( R \))
- Dependency Pair Framework [Giesl et al, *JAR ’06*] (simplified):
  - while \( DP \neq \emptyset \):
    - find well-founded order \( \succ \) with \( DP \cup R \subseteq \succ \)
    - delete \( s \rightarrow t \) with \( s \succ t \) from \( DP \)
### Example (Division)

\[
R = \begin{cases} 
\text{minus}(x, 0) & x \\
\text{minus}(s(x), s(y)) & \text{minus}(x, y) \\
\text{quot}(0, s(y)) & 0 \\
\text{quot}(s(x), s(y)) & s(\text{quot}(\text{minus}(x, y), s(y))) 
\end{cases}
\]

\[
DP = \begin{cases} 
\text{minus}^\#(s(x), s(y)) & \text{minus}^\#(x, y) \\
\text{quot}^\#(s(x), s(y)) & \text{minus}^\#(x, y) \\
\text{quot}^\#(s(x), s(y)) & \text{quot}^\#(\text{minus}(x, y), s(y)) 
\end{cases}
\]

### Dependency Pairs [Arts, Giesl, TCS ’00]

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

Get $\succeq$ via polynomial interpretations $\lfloor \cdot \rfloor$ over $\mathbb{N}$ [Lankford ’79] → ranking functions for rewriting

Example

$$\text{minus}(s(x), s(y)) \succeq \text{minus}(x, y)$$
Polynomial interpretations

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

\textbf{Example}

\[ \text{minus}(s(x), s(y)) \preceq \text{minus}(x, y) \]

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] \(
\rightarrow\) 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:

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

\(\succ\) boils down to \(\succ\) over \(\mathbb{N}\)
Example (Constraints for Division)

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

\[ \mathcal{DP} = \begin{cases} 
\text{minus}^\#(s(x), s(y)) & \rightarrow \text{minus}^\#(x, y) \\
\text{quot}^\#(s(x), s(y)) & \rightarrow \text{minus}^\#(x, y) \\
\text{quot}^\#(s(x), s(y)) & \rightarrow \text{quot}^\#(\text{minus}(x, y), s(y)) 
\end{cases} \]
Example (Constraints for Division)

\[ R = \begin{cases} 
\text{minus}(x, 0) & x \\
\text{minus}(s(x), s(y)) & \text{minus}(x, y) \\
\text{quot}(0, s(y)) & 0 \\
\text{quot}(s(x), s(y)) & s(\text{quot}(\text{minus}(x, y), s(y))) 
\end{cases} \]

\[ DP = \begin{cases} 
\text{minus}^\#(s(x), s(y)) & \succ \text{minus}^\#(x, y) \\
\text{quot}^\#(s(x), s(y)) & \succ \text{minus}^\#(x, y) \\
\text{quot}^\#(s(x), s(y)) & \succ \text{quot}^\#(\text{minus}(x, y), s(y)) 
\end{cases} \]

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

\[
\begin{align*}
[\text{quot}^\#](x_1, x_2) &= x_1 + x_2 \\
[\text{minus}^\#](x_1, x_2) &= x_1 \\
[0] &= 0 \\
[\text{quot}](x_1, x_2) &= x_1 + x_2 \\
[\text{minus}](x_1, x_2) &= x_1 \\
[s](x_1) &= x_1 + 1
\end{align*}
\]

\(\succ\) order solves all constraints
Example (Constraints for Division)

\[
R = \begin{cases}
\text{minus}(x, 0) & x \\
\text{minus}(s(x), s(y)) & \text{minus}(x, y) \\
\text{quot}(0, s(y)) & 0 \\
\text{quot}(s(x), s(y)) & s(\text{quot}(\text{minus}(x, y), s(y)))
\end{cases}
\]

\[
\mathcal{DP} = \begin{cases}
\end{cases}
\]

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

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

\(\bowtie\) order solves all constraints

\(\bowtie\) \(\mathcal{DP} = \emptyset\)

\(\bowtie\) termination of division algorithm proved
Task: Solve \( \text{minus}(s(x), s(y)) \preceq \text{minus}(x, y) \)
Task: Solve \( \text{minus}(s(x), s(y)) \preceq \text{minus}(x, y) \)

1. Fix a degree, use polynomial 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 \( \text{minus}(s(x), s(y)) \supseteq \text{minus}(x, y) \)

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:
   \[
   s \supseteq t \rightsquigarrow [s] \geq [t]
   \]
   Here: \( \forall x, y. \quad (a_s b_m + a_s c_m) + (b_s b_m - b_m) x + (b_s c_m - c_m) y \geq 0 \)
Automation

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

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:
   \[
   s \preceq t \Leftrightarrow [s] \geq [t]
   \]

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

3. Eliminate \( \forall x, y \) by absolute positiveness criterion
   [Hong, Jakuš, JAR ’98]:
   Here:
   \[
   a_s b_m + a_s c_m \geq 0 \land b_s b_m - b_m \geq 0 \land b_s c_m - c_m \geq 0
   \]
Task: Solve \( \text{minus}(s(x), s(y)) \preceq \text{minus}(x, y) \)

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:

\[
s \gtrsim t \leadsto [s] \geq [t]
\]

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: \( a_s b_m + a_s c_m \geq 0 \land b_s b_m - b_m \geq 0 \land b_s c_m - c_m \geq 0 \)
Task: Solve \( \text{minus}(s(x), s(y)) \succcurlyeq \text{minus}(x, y) \)

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:

\[
s \succcurlyeq t \bowtie [s] \geq [t]
\]

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: \( a_s \, b_m + a_s \, c_m \geq 0 \land b_s \, b_m - b_m \geq 0 \land b_s \, c_m - c_m \geq 0 \)
Task: Solve $\text{minus}(s(x), s(y)) \preceq \text{minus}(x, y)$

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:

   \[ s \preceq t \leadsto [s] \geq [t] \]

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

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

   [Hong, Jakuš, JAR ’98]:

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

   **Non-linear constraints (QF_NIA)**, even for **linear** interpretations
Task: Solve \( \text{minus}(s(x), s(y)) \preceq \text{minus}(x, y) \)

1. Fix a degree, use pol. interpretation with \textit{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:
   \[
   s \preceq t \iff [s] \geq [t]
   \]
   Here: \( \forall x, y. \quad (a_s b_m + a_s c_m) + (b_s b_m - b_m) x + (b_s c_m - c_m) y \geq 0 \)

3. Eliminate \( \forall x, y \) by \textit{absolute positiveness criterion} [Hong, Jakuš, JAR ’98]:
   Here: \( a_s b_m + a_s c_m \geq 0 \land b_s b_m - b_m \geq 0 \land b_s c_m - c_m \geq 0 \)

   \textit{Non-linear constraints} (QF_NIA), even for \textit{linear} interpretations

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

Prove termination of given term rewrite system
Extensions

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

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

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

- **Polynomials over $\mathbb{Q}^+$ and $\mathbb{R}^+$** [Lucas, *RAIRO ’05*]
  - non-integer coefficients increase proving power

- **Matrix interpretations** [Endrullis, Waldmann, Zantema, *JAR ’08*]
  - interpretation to vectors over $\mathbb{N}^k$, coefficients are matrices
  - useful for deeply nested terms
  - QF_NIA instances with more complex atoms
Extensions

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

- **Polynomials over \(\mathbb{Q}^+\) and \(\mathbb{R}^+\)** [Lucas, RAIRO ’05]
  - non-integer coefficients increase proving power
  - SMT-based automation [Fuhs et al, AISC ’08; Zankl, Middeldorp, LPAR ’10; Borralleras et al, JAR ’12]

- **Matrix interpretations** [Endrullis, Waldmann, Zantema, JAR ’08]
  - interpretation to vectors over \(\mathbb{N}^k\), coefficients are matrices
  - useful for deeply nested terms
  - QF_NIA instances with more complex atoms

- **“Arctic” matrices** on the max-plus semiring on \(\mathbb{N}\) or \(\mathbb{Z}\) (instead of plus-times) [Koprowski, Waldmann, Acta Cyb. ’09]
  - very useful for deeply nested terms
  - can be encoded to QF_LIA, but (unary!) bit-blasting seems to be faster in practice [Codish, Fekete, Fuhs, Giesl, Waldmann, SMT ’12]
Polynomials with negative coefficients and max-operator [Hirokawa, Middeldorp, IC ’07; Fuhs et al, SAT ’07, RTA ’08]
- models behavior of functions more closely
- automation via SMT for QF_NIA, more complex Boolean structure

Polynomials over \( \mathbb{Q}^+ \) and \( \mathbb{R}^+ \) [Lucas, RAIRO ’05]
- non-integer coefficients increase proving power
- SMT-based automation [Fuhs et al, AISC ’08; Zankl, Middeldorp, LPAR ’10; Borralleras et al, JAR ’12]

Matrix interpretations [Endrullis, Waldmann, Zantema, JAR ’08]
- interpretation to vectors over \( \mathbb{N}^k \), coefficients are matrices
- useful for deeply nested terms
- QF_NIA instances with more complex atoms

“Arctic” matrices on the max-plus semiring on \( \mathbb{N} \) or \( \mathbb{Z} \) (instead of plus-times) [Koprowski, Waldmann, Acta Cyb. ’09]
- very useful for deeply nested terms
- can be encoded to QF_LIA, but (unary!) bit-blasting seems to be faster in practice [Codish, Fekete, Fuhs, Giesl, Waldmann, SMT ’12]
Example (bits)

\[ \mathcal{R} = \begin{cases} 
  \text{half}(0) & \rightarrow 0 \\
  \text{half}(s(0)) & \rightarrow 0 \\
  \text{half}(s(s(x))) & \rightarrow s(\text{half}(x)) \\
  \text{bits}(0) & \rightarrow 0 \\
  \text{bits}(s(x)) & \rightarrow s(\text{bits}(\text{half}(s(x)))) \\
\end{cases} \]

Classic polynomials cannot solve \( \text{bits}(\text{half}(s(x))) \succ \text{bits}(\text{half}(s(s(x)))) \).

Remedy: 
\[ \begin{align*}
  \text{bits}(s(x)) &= s(\text{bits}(\text{half}(s(x)))) \\
  \text{half}(s(s(x))) &= \text{half}(s(s(x))) \\
  \text{half}(s(s(s(x)))) &= \text{half}(s(s(s(x)))) \\
\end{align*} \]

But: Then \( \succ \) not well founded any more: \( 0 \succ \text{half}(0) \succ \text{half}(\text{half}(0)) \succ \ldots \)

\[ \Rightarrow \text{Solution (Hirokawa, Middeldorp, IC '07)}: \]
\[ \begin{align*}
  \text{half}(x_1) &= \max(x_1 - 1, 0) \\
  \text{half}(s(x)) &= \max((x + 1) - 1, 0) = x \text{ for } x \leq 15/25
\end{align*} \]
Example (bits)

\[ R = \begin{cases} 
\text{half}(0) & \rightarrow 0 \\
\text{half}(s(0)) & \rightarrow 0 \\
\text{half}(s(s(x))) & \rightarrow s(\text{half}(x)) 
\end{cases} \]

\[ D_P = \begin{cases} 
\text{half}^\#(s(s(x))) & \rightarrow \text{half}^\#(x) \\
\text{bits}^\#(s(x)) & \rightarrow \text{half}^\#(s(x)) \\
\text{bits}^\#(s(x)) & \rightarrow \text{bits}^\#(\text{half}(s(x))) 
\end{cases} \]

Classic polynomials cannot solve \[ \text{bits}^\#(s(s(x))) \succ \text{bits}^\#(\text{half}(s(x))) \]

Remedy: \[ [\text{bits}^\#(x)] = x, [s(x)] = x + 1, [\text{half}(x)] = x - 1 \]

But then \[ \succ \] is no longer well-founded:
\[ 0 \succ \text{half}(0) \succ \text{half}(\text{half}(0)) \succ \ldots \]

⇒ Solution [Hirokawa, Middeldorp, IC '07]:
\[ [\text{half}^\#(x_1)] = \max(x_1 - 1, 0) \]

⇒ \[ [\text{bits}^\#(s(s(x)))] = \max((x + 1) - 1, 0) = x + 1/2 \]
Example (bits)

\[ R = \begin{cases} 
  \text{half}(0) & \succsim 0 \\
  \text{half}(s(0)) & \succsim 0 \\
  \text{half}(s(s(x))) & \succsim \text{s(half}(x)) \\
\end{cases} \]

\[ DP = \begin{cases} 
  \text{half}^\#(s(s(x))) & \succsim \text{half}^\#(x) \\
  \text{bits}^\#(s(x)) & \succsim \text{half}^\#(s(x)) \\
  \text{bits}^\#(s(x)) & \succsim \text{bits}^\#(\text{half}(s(x))) \\
\end{cases} \]

Classic polynomials cannot solve \[ \text{bits}^\#(s(x)) \succ \text{bits}^\#(\text{half}(s(x))) \]

Remedy: \[ \text{bits}^\#(x) = x, \quad \text{bits}^\#(s(x)) = x + 1, \quad \text{half}^\#(s(x)) = x - 1 \]

But then \[ \succ \text{not well founded any more} \]

⇒ Solution \[ [\text{Hirokawa, Middeldorp, IC '07}]: \]

\[ \text{half}^\#(x_1) = \max(x_1 - 1, 0) \]

⇒ \[ \text{half}^\#(s(x)) = \max((x + 1) - 1, 0) = x \]
Example (bits)

\[ R = \begin{cases} 
    \text{half}(0) \preceq 0 & \text{bits}(0) \preceq 0 \\
    \text{half}(\text{s}(0)) \preceq 0 & \text{bits}(\text{s}(x)) \preceq \text{s} (\text{bits}(\text{half}(\text{s}(x)))) \\
    \text{half}(\text{s}(\text{s}(x))) \preceq \text{s} (\text{half}(x)) 
\end{cases} \]

\[ DP = \begin{cases} 
    \text{bits}^\sharp (\text{s}(x)) \succ \text{bits}^\sharp (\text{half}(\text{s}(x))) 
\end{cases} \]
Example (bits)

\[ R = \begin{cases} 
\text{half}(0) \succsim 0 & \text{bits}(0) \succsim 0 \\
\text{half}(\text{s}(0)) \succsim 0 & \text{bits}(\text{s}(x)) \succsim \text{s}(\text{bits}(\text{half}(\text{s}(x)))) \\
\text{half}(\text{s}(\text{s}(x))) \succsim \text{s}(\text{half}(x)) 
\end{cases} \]

\[ D\mathcal{P} = \begin{cases} 
\text{bits}^\#(\text{s}(x)) \succ \text{bits}^\#(\text{half}(\text{s}(x))) 
\end{cases} \]

- Classic polynomials cannot solve \( \text{bits}^\#(\text{s}(x)) \succ \text{bits}^\#(\text{half}(\text{s}(x))) \)
Example (bits)

\[ R = \begin{cases} 
\text{half}(0) \preceq 0 & \text{bits}(0) \preceq 0 \\
\text{half}(s(0)) \preceq 0 & \text{bits}(s(x)) \preceq s(\text{bits}(\text{half}(s(x)))) \\
\text{half}(s(s(x))) \preceq s(\text{half}(x)) & 
\end{cases} \]

\[ D \mathcal{P} = \begin{cases} 
\text{bits}^\#(s(x)) \succ \text{bits}^\#(\text{half}(s(x))) 
\end{cases} \]

- Classic polynomials cannot solve \( \text{bits}^\#(s(x)) \succ \text{bits}^\#(\text{half}(s(x))) \)
- Remedy: \( [\text{bits}^\#](x) = x, \; [s](x) = x + 1, \; [\text{half}](x) = x - 1 \)
Example (bits)

\[ R = \begin{cases} 
\text{half}(0) \preceq \ 0 & \text{bits}(0) \preceq \ 0 \\
\text{half}(s(0)) \preceq \ 0 & \text{bits}(s(x)) \preceq \ s(\text{bits}(\text{half}(s(x)))) \\
\text{half}(s(s(x))) \preceq \ s(\text{half}(x)) 
\end{cases} \]

\[ DP = \begin{cases} 
\text{bits}^\#(s(x)) \succ \text{bits}^\#(\text{half}(s(x))) 
\end{cases} \]

- Classic polynomials cannot solve \( \text{bits}^\#(s(x)) \succ \text{bits}^\#(\text{half}(s(x))) \)
- Remedy: \([\text{bits}^\#](x) = x, \ [s](x) = x + 1, \ [\text{half}](x) = x - 1\)
- But: Then \(\succ\) not well founded any more:
  \[ 0 \succ \text{half}(0) \succ \text{half}(\text{half}(0)) \succ \ldots \]
Example (bits)

\[ \mathcal{R} = \left\{ \begin{array}{ll}
\text{half}(0) & \preceq 0 \\
\text{half}(\text{s}(0)) & \preceq 0 \\
\text{half}(\text{s}(\text{s}(x))) & \preceq \text{s}(\text{half}(x))
\end{array} \right. \]

\[ \mathcal{DP} = \left\{ \begin{array}{ll}
\text{bits}^\#(\text{s}(x)) & \succ \text{bits}^\#(\text{half}(x))
\end{array} \right. \]

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

\[ 0 \succ \text{half}(0) \succ \text{half}(\text{half}(0)) \succ \ldots \]

\( \Rightarrow \) Solution [Hirokawa, Middeldorp, IC '07]:

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

\( \Rightarrow \) \[ [\text{half}(\text{s}(x))] = \max((x + 1) - 1, 0) = x \]
Example (bits)

\[ R = \begin{cases} \text{half}(0) \succsim 0 & \text{bits}(0) \succsim 0 \\ \text{half}(s(0)) \succsim 0 & \text{bits}(s(x)) \succsim s(\text{bits}(\text{half}(s(x)))) \\ \text{half}(s(s(x))) \succsim s(\text{half}(x)) & \end{cases} \]

\[ DP = \begin{cases} \end{cases} \]

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

\[ 0 \succ \text{half}(0) \succ \text{half}(\text{half}(0)) \succ \ldots \]

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

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

\Rightarrow \[ [\text{half}(s(x))] = \max((x + 1) - 1, 0) = x \]
Problem: Expressions like \( \max(x_1 - 1, 0) \) are no polynomials

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

For \[ s > t \], show \( s^{left} > t^{right} \)

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

For $[s] > [t]$, show $[s]^{\text{left}} > [t]^{\text{right}}$

- $[s]^{\text{left}}$ under-approximation of $[s]$
- $[t]^{\text{right}}$ over-approximation of $[t]$
- $[s]^{\text{left}}, [t]^{\text{right}}$ polynomials

Automation initially: Generate-and-test

Approx. for $\max(p, 0)$ depend on signum of constant addend of $p$

\[
[s(x)] = \max(x + 1, 0) \quad \Rightarrow \quad [s(x)]^{\text{right}} = x + 1 \\
\text{half}(x)] = \max(x - 1, 0) \quad \Rightarrow \quad [\text{half}(x)]^{\text{right}} = x
\]
Problem: Expressions like $\max(x_1 - 1, 0)$ are no polynomials

For $[s] > [t]$, show $[s]^\text{left} > [t]^\text{right}$

- $[s]^\text{left}$ under-approximation of $[s]$
- $[t]^\text{right}$ over-approximation of $[t]$
- $[s]^\text{left}, [t]^\text{right}$ polynomials

Automation initially: Generate-and-test

Approx. for $\max(p, 0)$ depend on signum of constant addend of $p$

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

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

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

. . . using side constraints

$$(b_f \geq 0 \rightarrow c_f(x) = b_f) \land (b_f < 0 \rightarrow c_f(x) = 0)$$

Boolean structure in SMT quite handy!
Path orders: based on precedences of function symbols

- Recursive Path Order [Dershowitz, TCS ’82; Codish et al, JAR ’11]
- Weighted Path Order [Yamada, Kusakari, Sakabe, SCP ’15]
Path orders: based on precedences of function symbols

- Recursive Path Order [Dershowitz, TCS ’82; Codish et al, JAR ’11]
- Weighted Path Order [Yamada, Kusakari, Sakabe, SCP ’15]
- Knuth-Bendix Order [Knuth, Bendix, CPAA ’70]
  → SMT-Encoding to QF_LIA [Zankl, Hirokawa, Middeldorp, JAR ’09]
  outperformed polynomial time algorithm [Korovin, Voronkov, IC ’03] in experiments
Path orders: based on precedences of function symbols

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

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

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

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

- **Complexity analysis** [Hirokawa, Moser, *IJCAR ’08*; Noschinski, Emmes, Giesl, *JAR ’13*]
  Can re-use termination machinery to infer and prove statements like “runtime complexity of this TRS is in $\mathcal{O}(n^3)$”
SMT solvers *from* termination analysis

Annual SMT-COMP, division QF\_NIA

<table>
<thead>
<tr>
<th>Year</th>
<th>Winner</th>
</tr>
</thead>
<tbody>
<tr>
<td>2009</td>
<td>Barcelogic-QF_NIA</td>
</tr>
<tr>
<td>2010</td>
<td>MiniSmt</td>
</tr>
<tr>
<td>2011</td>
<td>AProVE</td>
</tr>
<tr>
<td>2012</td>
<td><em>no QF_NIA</em></td>
</tr>
<tr>
<td>2013</td>
<td><em>no SMT-COMP</em></td>
</tr>
<tr>
<td>2014</td>
<td>AProVE</td>
</tr>
<tr>
<td>2015</td>
<td>AProVE</td>
</tr>
<tr>
<td>2016</td>
<td>→ <em>today, 4 pm</em></td>
</tr>
</tbody>
</table>
### SMT solvers from termination analysis

Annual SMT-COMP, division QF_NIA

<table>
<thead>
<tr>
<th>Year</th>
<th>Winner</th>
</tr>
</thead>
<tbody>
<tr>
<td>2009</td>
<td>Barcelogic-QF_NIA</td>
</tr>
<tr>
<td>2010</td>
<td>MiniSmt (spin-off of TTT2)</td>
</tr>
<tr>
<td>2011</td>
<td>AProVE</td>
</tr>
<tr>
<td>2012</td>
<td>no QF_NIA</td>
</tr>
<tr>
<td>2013</td>
<td>no SMT-COMP</td>
</tr>
<tr>
<td>2014</td>
<td>AProVE</td>
</tr>
<tr>
<td>2015</td>
<td>AProVE</td>
</tr>
<tr>
<td>2016</td>
<td>→ today, 4 pm</td>
</tr>
</tbody>
</table>

⇒ Termination provers can also be successful SMT solvers!
SMT solvers *from* termination analysis

Annual SMT-COMP, division QF\_NIA

<table>
<thead>
<tr>
<th>Year</th>
<th>Winner</th>
</tr>
</thead>
<tbody>
<tr>
<td>2009</td>
<td>Barcelogic-QF_NIA</td>
</tr>
<tr>
<td>2010</td>
<td>MiniSmt (spin-off of TTT2)</td>
</tr>
<tr>
<td>2011</td>
<td>AProVE</td>
</tr>
<tr>
<td>2012</td>
<td><em>no QF_NIA</em></td>
</tr>
<tr>
<td>2013</td>
<td><em>no SMT-COMP</em></td>
</tr>
<tr>
<td>2014</td>
<td>AProVE</td>
</tr>
<tr>
<td>2015</td>
<td>AProVE</td>
</tr>
<tr>
<td>2016</td>
<td>→ <em>today, 4 pm</em></td>
</tr>
</tbody>
</table>

⇒ *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 \textbf{integers} as data

**Example (Imperative program)**

\begin{verbatim}
if x \geq 0:
  while x \neq 0:
    x = x - 1
\end{verbatim}

Does this program terminate?
Papers on termination of imperative programs often about **integers** as data.

**Example (Imperative program)**

\[\ell_0: \text{if } x \geq 0:\]
\[\ell_1: \text{while } x \neq 0:\]
\[\ell_2: \quad x = x - 1\]

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 \textbf{integers} as data

Example (Imperative program)

\begin{itemize}
  \item[$\ell_0$:] if $x \geq 0$:
  \item[$\ell_1$:] while $x \neq 0$:
  \item[$\ell_2$:] $x = x - 1$
\end{itemize}

Does this program terminate?

Example (Equivalent translation to transition system)

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

Oh no! $\ell_1(-1) \rightarrow \ell_2(-1) \rightarrow \ell_1(-2) \rightarrow \ell_2(-2) \rightarrow \ell_1(-3) \rightarrow \cdots$
Papers on termination of imperative programs often about integers as data

Example (Imperative program)

\[\ell_0: \quad \textbf{if } x \geq 0: \]
\[\ell_1: \quad \textbf{while } x \neq 0: \]
\[\ell_2: \quad x = x - 1\]

Does this program terminate?

Example (Equivalent translation to transition system)

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

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

⇒ Restrict initial states to \(\ell_0(z)\) for \(z \in \mathbb{Z}\)
Papers on termination of imperative programs often about **integers** as data.

### Example (Imperative program)

<table>
<thead>
<tr>
<th>Label</th>
<th>Statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\ell_0$:</td>
<td>if $x \geq 0$:</td>
</tr>
<tr>
<td>$\ell_1$:</td>
<td>while $x \neq 0$:</td>
</tr>
<tr>
<td>$\ell_2$:</td>
<td>$x = x - 1$</td>
</tr>
</tbody>
</table>

Does this program terminate?

### Example (Equivalent translation to transition system)

<table>
<thead>
<tr>
<th>Label</th>
<th>Statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\ell_0(x)$</td>
<td>$\rightarrow \ell_1(x)$ [(x \geq 0)]</td>
</tr>
<tr>
<td>$\ell_1(x)$</td>
<td>$\rightarrow \ell_2(x)$ [(x \neq 0)]</td>
</tr>
<tr>
<td>$\ell_2(x)$</td>
<td>$\rightarrow \ell_1(x - 1)$</td>
</tr>
<tr>
<td>$\ell_1(x)$</td>
<td>$\rightarrow \ell_3(x)$ [(x == 0)]</td>
</tr>
</tbody>
</table>

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

⇒ **Restrict initial states** to $\ell_0(z)$ for $z \in \mathbb{Z}$
⇒ **Find invariant** $x \geq 0$ at $\ell_1, \ell_2$
Papers on termination of imperative programs often about \textbf{integers} as data.

Example (Imperative program)

\begin{align*}
\ell_0: & \text{ if } x \geq 0: \\
\ell_1: & \text{ while } x \neq 0: \\
\ell_2: & x = x - 1
\end{align*}

Does this program terminate?

Example (Equivalent translation to transition system)

\begin{align*}
\ell_0(x) & \rightarrow \ell_1(x) \quad [x \geq 0] \\
\ell_1(x) & \rightarrow \ell_2(x) \quad [x \neq 0 \land 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 \land x \geq 0]
\end{align*}

Oh no!

\begin{align*}
\ell_1(-1) & \rightarrow \ell_2(-1) \rightarrow \ell_1(-2) \rightarrow \ell_2(-2) \rightarrow \ell_1(-3) \rightarrow \cdots
\end{align*}

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

\Rightarrow \text{ Find invariant } x \geq 0 \text{ at } \ell_1, \ell_2
Example (Transition system with invariants)

\[
\begin{align*}
\ell_0(x) & \rightarrow \ell_1(x) \quad [x \geq 0] \\
\ell_1(x) & \rightarrow \ell_2(x) \quad [x \neq 0 \land x \geq 0] \\
\ell_2(x) & \rightarrow \ell_1(x-1) \quad [x \geq 0] \\
\ell_1(x) & \rightarrow \ell_3(x) \quad [x == 0 \land x \geq 0]
\end{align*}
\]

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

Example (Transition system with invariants)

\[\ell_0(x) \gg \ell_1(x) \quad [x \geq 0]\]
\[\ell_1(x) \gg \ell_2(x) \quad [x \neq 0 \land x \geq 0]\]
\[\ell_2(x) \gg \ell_1(x - 1) \quad [x \geq 0]\]
\[\ell_1(x) \gg \ell_3(x) \quad [x == 0 \land x \geq 0]\]

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

Example (Transition system with invariants)

\[
\begin{align*}
\ell_0(x) & \preceq \ell_1(x) & [x \geq 0] \\
\ell_1(x) & \preceq \ell_2(x) & [x \neq 0 \land x \geq 0] \\
\ell_2(x) & \preceq \ell_1(x - 1) & [x \geq 0] \\
\ell_1(x) & \preceq \ell_3(x) & [x == 0 \land x \geq 0]
\end{align*}
\]

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)**

\[ \ell_0(x) \succsim \ell_1(x) \quad [x \geq 0] \]

\[ \ell_1(x) \succsim \ell_2(x) \quad [x \neq 0 \land x \geq 0] \]

\[ \ell_2(x) \succ \ell_1(x - 1) \quad [x \geq 0] \]

\[ \ell_1(x) \succsim \ell_3(x) \quad [x == 0 \land x \geq 0] \]

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.:

\[ x \geq 0 \quad \Rightarrow \quad a_2 + b_2 \cdot x > a_1 + b_1 \cdot (x - 1) \quad \text{“decrease ...”} \]

\[ x \geq 0 \quad \Rightarrow \quad a_2 + b_2 \cdot x \geq 0 \quad \text{“... against a bound”} \]
Proving termination with invariants

Example (Transition system with invariants)

\[ \ell_0(x) \preceq \ell_1(x) \quad [x \geq 0] \]
\[ \ell_1(x) \preceq \ell_2(x) \quad [x \neq 0 \land x \geq 0] \]
\[ \ell_2(x) \succ \ell_1(x - 1) \quad [x \geq 0] \]
\[ \ell_1(x) \preceq \ell_3(x) \quad [x == 0 \land x \geq 0] \]

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.:

\[ x \geq 0 \quad \Rightarrow \quad a_2 + b_2 \cdot x > a_1 + b_1 \cdot (x - 1) \quad \text{“decrease . . .”} \]
\[ x \geq 0 \quad \Rightarrow \quad a_2 + b_2 \cdot x \geq 0 \quad \text{“. . . against a bound”} \]

Use Farkas’ Lemma to eliminate \( \forall x \), QF_LRA solver gives model for \( a_i, b_i \).
Proving termination with invariants

Example (Transition system with invariants)

\[ \ell_0(x) \succcurlyeq \ell_1(x) \quad [x \geq 0] \]
\[ \ell_1(x) \succcurlyeq \ell_2(x) \quad [x \neq 0 \land x \geq 0] \]
\[ \ell_2(x) \succcurlyeq \ell_1(x - 1) \quad [x \geq 0] \]
\[ \ell_1(x) \succcurlyeq \ell_3(x) \quad [x == 0 \land x \geq 0] \]

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.:

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

Use Farkas’ Lemma to eliminate \(\forall x\), QF_LRA solver gives model for \(a_i, b_i\).

More: [Podelski, Rybalchenko, VMCAI ’04, Alias et al, SAS ’10]
Searching for invariants using SMT

Termination prover needs to find invariants for programs on integers

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

Nowadays all SMT-based!
Extensions

- Proving non-termination (infinite run from initial states is possible)

- CTL* model checking for infinite state systems based on termination and non-termination provers
  [Cook, Khlaaf, Piterman, *CAV ’15*]

- Complexity bounds
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 \(\sim 15\) years
- Term rewriting: need to encode how to represent data structures
- Imperative programs on integers: need to consider reachability and invariants
Conclusion

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

Term rewriting: need to encode how to represent data structures

Imperative programs on integers: need to consider reachability and invariants

Since a few years cross-fertilization

Automation heavily relies on SMT solving for automation
Conclusion

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

- Term rewriting: need to encode how to represent data structures
- Imperative programs on integers: need to consider reachability and invariants
- Since a few years cross-fertilization
- Automation heavily relies on SMT solving for automation
- Needs of termination analysis have also led to better SMT solvers

Annual termCOMP:
http://termination-portal.org/wiki/Termination_Competition
Conclusion

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

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


