The Dependency Pair Technique
Proving Termination of Term Rewrite Systems

Hannes Saffrich

University of Freiburg
Department of Computer Science

June 30, 2015
Introduction

- Term Rewrite Systems (TRSs) are a turing complete formalism for modeling both programs and programming languages.

- Proving the termination of a TRS corresponds either
  - to proving termination of a particular program, or
  - to proving termination of all programs in a programming language.

- Unfortunately, proving termination of arbitrary TRSs is undecidable (Halting Problem).

- The Dependency Pair Technique describes an algorithm which for a given TRS either
  - constructs a proof of termination, or
  - constructs a proof of non-termination, or
  - gives up.

- It can be extended to incorporate other termination analyses.
Overview

- **Part I:** Term Rewrite Systems
  - Formal definition
  - Termination property
  - Example

- **Part II:** The Dependency Pair Technique
  - Alternative Termination Property
  - Proving this property automatically
  - Covering Section 1 and 2 of the original paper.

- **Part III:** Summary
Part I

Term Rewrite Systems
Term Rewrite Systems

Introduction

- Term Rewrite System (TRS) $\mathcal{T}$ consists of
  - a set of terms $\mathcal{T}$
  - a binary relation between terms $\rightarrow \subseteq \mathcal{T} \times \mathcal{T}$
- $t_1 \rightarrow t_2$ iff $t_1$ is rewritten to $t_2$ by 1 computation step, i.e.
  
  
  \[
  1 + (2 + 3) \rightarrow 1 + 5 \quad 1 + 5 \rightarrow 6
  \]

- $t_1 \rightarrow^* t_2$ iff $t_1$ is rewritten to $t_2$ by $n$ computation steps, i.e.

  
  \[
  1 + (2 + 3) \rightarrow^* 6 \quad 6 \rightarrow^* 6
  \]

- $\mathcal{T}$ is non-terminating for $t_1$ iff $t_1$ can be rewritten infinitely often

  
  \[
  \forall t_2. \ (t_1 \rightarrow^* t_2 \implies \exists t_3. \ t_2 \rightarrow t_3)
  \]

- $\mathcal{T}$ is terminating iff it contains no non-terminating term.
Term Rewrite Systems

Terms

- The terms $T$ of a TRS are trees built from
  - a set of function symbols $\mathcal{F}$
  - a set of variables $\mathcal{V}$
- Example: Arithmetic on Natural Numbers
  - $\mathcal{F} = \{ +^2, \cdot^2, s^1, 0^0 \}$
  - $\mathcal{V} = \{ x, y, z, \ldots \}$
  - Arity fixes number of sub-terms (function arguments)
  - String notation: $+(s(0), \cdot(x, 0))$
Term Rewrite Systems

Substitution

- A substitution $\sigma : \mathcal{V} \rightarrow T$ maps variables to terms
- We write $\{x \mapsto t\}$ for the substitution

$$\{x \mapsto t\} (y) = \begin{cases} t & \text{if } x = y \\ y & \text{otherwise} \end{cases}$$

- We write $\sigma [t]$ for substituting all variables in term $t$ by $\sigma$
- i.e. for $\sigma = \{x \mapsto 0, y \mapsto (x, z)\}$ it holds that

$$\sigma \begin{bmatrix} + \\
\text{x} & \text{y} \end{bmatrix} = 0 \begin{bmatrix} + \\
\text{x} & \text{z} \end{bmatrix}$$
Term Rewrite Systems

Semantics

▶ Rewrite relation $\rightarrow \subseteq T \times T$
▶ Derived from a set of rewrite rules $t_1 \xrightarrow{R} t_2 \in T \times T$
▶ Example: Arithmetic on Natural Numbers

$+(0, y) \xrightarrow{R} y$ (BASE+) 
$+(s(x), y) \xrightarrow{R} s(+(x, y))$ (REC+)

$\cdot(0, y) \xrightarrow{R} 0$ (BASE·) 
$(s(x), y) \xrightarrow{R} +(\cdot(x, y), y)$ (REC·)

▶ Closed under substitutions
  ▶ if $t_1 \rightarrow t_2$ then $\forall \sigma. \sigma[t_1] \rightarrow \sigma[t_2]$, i.e.
  ▶ if $+(0, y) \rightarrow y$ then $+(0, \cdot(0, 0)) \rightarrow \cdot(0, 0)$

▶ Closed under contexts
  ▶ if $t_1 \rightarrow t_2$ then $\{x \mapsto t_1\}[t] \rightarrow \{x \mapsto t_2\}[t]$, i.e.
  ▶ if $+(0, y) \rightarrow y$ then $s(+(0, y)) \rightarrow s(y)$
Symbol Classification

- The outermost symbol of a term is called its *root symbol*.
- Root symbols of a rewrite rule’s left side are called *defined*.
  - In our example those are $\text{Def}_R = \{+, \cdot\}$
  - Defined symbols represent functions, because terms having them as root symbol may be reduced by the rewriting relation.
- Symbols which are not defined, are called *constructors*.
  - Constructors represent values of recursive datatypes.
  - In our example those are $\text{Cons}_R = \{0, s\}$ and we use them to build terms representing arbitrary natural numbers, i.e. $s(s(0))$ for 2.
- The rewrite rules define the functions of the defined function symbols by pattern matching on constructors.

\[
\begin{align*}
+(0, y) & \xrightarrow{R} y & (\text{BASE}+) \\
\cdot(0, y) & \xrightarrow{R} 0 & (\text{BASE} \cdot)
\end{align*}
\]
\[
\begin{align*}
+(s(x), y) & \xrightarrow{R} s(+ (x, y)) & (\text{REC}+) \\
\cdot(s(x), y) & \xrightarrow{R} + (\cdot (x, y), y) & (\text{REC} \cdot)
\end{align*}
\]
PART II
The Dependency Pair Technique
Dependency Pair Technique
Termination revisited

- A term is barely non-terminating iff it is non-terminating, but all its subterms are terminating
- A non-terminating term $t$ has a barely non-terminating subterm $u$
  - start with $u := t$
  - either all subterms of $u$ are terminating
  - or we can choose a non-terminating subterm as new $u$
- When rewriting $u$, eventually a rule $l \xrightarrow{R} r$ rewrites the whole term
  $$u = f(u_1, \ldots, u_n) \xrightarrow{^*} f(v_1, \ldots, v_n) = \sigma \{ l \} \xrightarrow{\sigma} \sigma \{ r \}$$
  - $\sigma \{ r \}$ is reached in a finite amount of steps, hence
  - $\sigma \{ r \}$ is non-terminating and we can repeat this infinitely often
Dependency Pair Technique

Dependency Pairs & Chains

\[ t \sqsubseteq f(u_1, \ldots, u_n) \xrightarrow{\ast} f(v_1, \ldots, v_n) = \sigma [l] \rightarrow \sigma [r] \]
\[ \sqsubseteq f'(u'_1, \ldots, u'_n) \xrightarrow{\ast} \ldots \]

- A dependency pair combines the top-level rule \( l \xrightarrow{R} r \) with the subsequent choice of the barely non-terminating subterm of \( \sigma [r] \).
- The dependency pairs of a TRS \( \mathcal{T} \) are

\[
\text{DP}(\mathcal{T}) = \left\{ l \xrightarrow{DP} r' \mid l \xrightarrow{R} r \in \mathcal{T}, r \sqsubseteq r', \text{root}(r') \in \text{Def}_R \right\}
\]

- A chain is a sequence of dependency pairs \( l_1 \xrightarrow{DP} r_1, l_2 \xrightarrow{DP} r_2, \ldots \), such that \( \forall i. \exists \sigma. \sigma [r_i] \xrightarrow{\ast} \sigma [l_{i+1}] \).
- A TRS terminates if it has no infinite chains.
Example: Arithmetic on Natural Numbers

\[ + (0, y) \xrightarrow{R} y \quad \text{(BASE+)} \quad +(s(x), y) \xrightarrow{R} s(+ (x, y)) \quad \text{(REC+)} \]

\[ \cdot (0, y) \xrightarrow{R} 0 \quad \text{(BASE\cdot)} \quad \cdot (s(x), y) \xrightarrow{R} + (\cdot (x, y), y) \quad \text{(REC\cdot)} \]

The dependency pairs for this TRS are

\[ +(s(x), y) \xrightarrow{DP} +(x, y) \quad \cdot (s(x), y) \xrightarrow{DP} \cdot (x, y) \quad \cdot (s(x), y) \xrightarrow{DP} + (\cdot (x, y), y) \]

Repeating the first rule 2 times is a finite chain

\[ +(s(x), y) \xrightarrow{DP} +(x, y) \quad +(s(x), y) \xrightarrow{DP} +(x, y) \]

because for \( \sigma_1 = \{ x \mapsto s(0), y \mapsto 0 \} \) and \( \sigma_2 = \{ x \mapsto 0, y \mapsto 0 \} \)

\[ +(s(s(0)), 0) \xrightarrow{\gamma^*} +(s(0), 0) \xrightarrow{s} + (0, 0) \]
A dependency problem $P = \langle T, D \rangle$ consists of
- a TRS $T$
- a set of dependency pairs $D$

$P$ is solved by proving the absence or existence of infinite chains of $D$ in $T$.

Solving $\langle T, DP(T) \rangle$ answers whether $T$ is terminating.

Basic idea of the dependency pair technique
- split dependency problems into smaller dependency problems
- such that solving the smaller problems solves the original problem
Dependency Pair Technique

Overview

Given TRS $\mathcal{T}$
- Calculate $\text{DP}(\mathcal{T})$
- Start with problem $\langle \mathcal{T}, \text{DP}(\mathcal{T}) \rangle$
- Split with dependency processors
- Repeat until solved or timeout
- $\text{DP}_i = \emptyset \rightarrow$ no chains $\rightarrow$ terminates
- Dependency processors as extension mechanism
Dependency Pair Technique

Dependency Graph Processor

- **Input:** dependency problem \( \langle T, \text{DP} \rangle \)
- **Approximates a dependency graph** \( G \)
  - **Nodes are dependency pairs** DP
  - **Edge from** \( p_1 \) **to** \( p_2 \) **iff** \( p_1, p_2 \) **is a chain of** DP **in** \( T \)
- **Output:** \( \{T\} \times \text{SCCs}(G) \)
- **Solving output problems solves input problem**
  - DP finite, but infinite chains are infinite
  - Dependency pairs have to repeat
  - \( G \) **captures potential repeating by edges**
  - Any infinite chain has at least one infinite sub-chain containing only the dependency pairs of a single SCC of \( G \)
  - Focusing on that SCC, still allows to find this infinite sub-chain
  - Covering all SCCs covers all infinite chains
Dependency Pair Technique

Dependency Graph Processor

- Dependency pairs in \( \langle T, \text{DP}(T) \rangle \) for our example

\[ +(s(x), y) \xrightarrow{\text{DP}} +(x, y) \quad \cdot(s(x), y) \xrightarrow{\text{DP}} \cdot(x, y) \quad \cdot(s(x), y) \xrightarrow{\text{DP}} +\cdot(x, y), y) \]

- Dependency graph for our example

- 2 output problems

\[ \langle T, \left\{ \cdot(s(x), y) \xrightarrow{\text{DP}} \cdot(x, y) \right\} \rangle \quad \langle T, \left\{ +(s(x), y) \xrightarrow{\text{DP}} +(x, y) \right\} \rangle \]
Dependency Pair Technique
Reduction Pair Processor

- Idea: well-founded order $\succ$ on terms implies termination
- For all $t_1$ exist only finitely many $t_2$ with $t_1 \succ t_2$
- If for any rule $l \xrightarrow{R} r$ it holds that $l \succ r$ and $\succ$ is closed under substitution and contexts, then all terms are terminating
- If each rewriting makes the term smaller and there are only a finite amount of smaller terms, we can only finitely often rewrite.
We can do better!

We want to show that there is no infinite chain for a DP problem

- no infinite sequence \( l_1 \xrightarrow{DP} r_1, l_2 \xrightarrow{DP} r_2, \ldots \)
  \( \forall i. \exists \sigma. \sigma[r_i] \xrightarrow{\sim}^* \sigma[l_{i+1}] \).

Only the dependency pairs have to make the terms smaller wrt. \( \succ \).

For TRS rules it suffices to not make the terms bigger, as

\( \sigma[r_i] \xrightarrow{S}^* \sigma[l_{i+1}] \) terminates anyway.

Hence, for the absence of infinite chains, it suffices to show

- \( l \succ r \) for all dependency pairs \( l \xrightarrow{DP} r \)
- \( l \succeq r \) for all TRS rules \( l \xrightarrow{R} r \)
Dependency Pair Technique

Reduction Pair Processor

- In our example we have two dependency problems left

\[
\langle T, \{(s(x), y) \xrightarrow{DP} (x, y)\} \rangle \quad \langle T, \{(s(x), y) \xrightarrow{DP} + (x, y)\} \rangle
\]

- We have to find an order \(\succ_i\) each, satisfying the following constraints

\[
\begin{align*}
+(s(x), y) & \succ_1 + (x, y) & \cdot(s(x), y) & \succ_2 \cdot(x, y) \\
+(0, y) & \succ_i y & +(s(x), y) & \succ_i s(+ (x, y)) \\
\cdot(0, y) & \succ_i 0 & \cdot(s(x), y) & \succ_i + (\cdot(x, y), y)
\end{align*}
\]

- The lexicographic order with \(0 < s < + < \cdot\)
  - is well-founded
  - satisfies the constraints for \(i \in \{1, 2\}\)
Part III

Summary
Summary

- Proving termination is undecidable
- The dependency pair technique can decide some instances
- This is achieved by identifying infinite chains
- Dependency problems restrict the scope in which to identify those chains
- Dependency problems are split by dependency processors
- This allows for integrating other termination analyses
- Further information in the original paper

Giesl, Thiemann, Falke.  
*Mechanizing and Improving Dependency Pairs.*  
Journal of Automated Reasoning, Vol 37, 2006
The End

Thanks for your attention!
Questions?

Giesl, Thiemann, Falke.
*Mechanizing and Improving Dependency Pairs.*
Journal of Automated Reasoning, Vol 37, 2006