Program Analysis Seminar

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 T
  - a binary relation between terms $\longrightarrow \subseteq T \times T$
- $t_1 \longrightarrow t_2$ iff t_1 is rewritten to t_2 by 1 computation step, i.e.

$$1 + (2 + 3) \longrightarrow 1 + 5 \qquad\qquad 1 + 5 \longrightarrow 6$$

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

$$1 + (2 + 3) \longrightarrow^* 6 \qquad\qquad 6 \longrightarrow^* 6$$

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

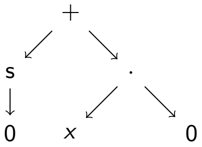
$$\forall t_2.\ (t_1 \longrightarrow^* t_2 \implies \exists t_3.\ t_2 \longrightarrow t_3)$$

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

Term Rewrite Systems

Terms

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



Term Rewrite Systems

Substitution

- A *substitution* $\sigma : \mathcal{V} \to T$ 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 σ
- i.e. for $\sigma = \{x \mapsto 0, y \mapsto \cdot(x, z)\}$ it holds that

$$\sigma \left[\begin{tikzcd}[column sep=small] & + \arrow[dl] \arrow[dr] & \\ x & & y \end{tikzcd} \right] = \begin{tikzcd}[column sep=small] & + \arrow[dl] \arrow[dr] & & \\ 0 & & \cdot \arrow[dl] \arrow[dr] & \\ & x & & z \end{tikzcd}$$

Term Rewrite Systems

Semantics

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

$$+(0, y) \xrightarrow{R} y \quad (\textsc{Base}+) \qquad +(\mathsf{s}(x), y) \xrightarrow{R} \mathsf{s}(+(x, y)) \quad (\textsc{Rec}+)$$

$$\cdot(0, y) \xrightarrow{R} 0 \quad (\textsc{Base}\cdot) \qquad \cdot(\mathsf{s}(x), y) \xrightarrow{R} +(\cdot(x, y), y) \quad (\textsc{Rec}\cdot)$$

- Closed under substitutions
  - if $t_1 \longrightarrow t_2$ then $\forall \sigma.\ \sigma[t_1] \longrightarrow \sigma[t_2]$, i.e.
  - if $+(0, y) \longrightarrow y$ then $+(0, \cdot(0, 0)) \longrightarrow \cdot(0, 0)$
- Closed under contexts
  - if $t_1 \longrightarrow t_2$ then $\{x \mapsto t_1\}[t] \longrightarrow \{x \mapsto t_2\}[t]$, i.e.
  - if $+(0, y) \longrightarrow y$ then $\mathsf{s}(+(0, y)) \longrightarrow \mathsf{s}(y)$

Term Rewrite Systems

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 $\mathrm{Def}_{\mathcal{R}} = \{+, \cdot\}$
  - 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 $\mathrm{Cons}_{\mathcal{R}} = \{0, \mathsf{s}\}$ 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.

$$+(0, y) \xrightarrow{R} y \quad (\textsc{Base}+) \qquad +(\mathsf{s}(x), y) \xrightarrow{R} \mathsf{s}(+(x, y)) \quad (\textsc{Rec}+)$$

$$\cdot(0, y) \xrightarrow{R} 0 \quad (\textsc{Base}\cdot) \qquad \cdot(\mathsf{s}(x), y) \xrightarrow{R} +(\cdot(x, y), y) \quad (\textsc{Rec}\cdot)$$

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

$$u = f(u_1, \ldots, u_n) \longrightarrow_s^* f(v_1, \ldots, v_n) = \sigma[l] \longrightarrow \sigma[r]$$

- $\sigma[r]$ 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

$$
\begin{aligned}
t \sqsupseteq f(u_1, \ldots, u_n) &\longrightarrow_s^* f(v_1, \ldots, v_n) = \sigma\,[l] \longrightarrow \sigma\,[r] \\
&\sqsupseteq f'(u'_1, \ldots, u'_n) \longrightarrow_s^* \ldots
\end{aligned}
$$

- 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

$$\mathrm{DP}(\mathcal{T}) = \left\{ l \xrightarrow{DP} r' \;\middle|\; l \xrightarrow{R} r \in \mathcal{T},\ r \sqsupseteq r',\ \mathrm{root}(r') \in \mathrm{Def}_{\mathcal{R}} \right\}$$

- A chain is a sequence of dependency pairs $l_1 \xrightarrow{DP} r_1, l_2 \xrightarrow{DP} r_2, \ldots$, such that $\forall i.\ \exists \sigma.\ \sigma\,[r_i] \longrightarrow_s^* \sigma\,[l_{i+1}]$.
- A TRS terminates if it has no infinite chains.

Dependency Pair Technique

Dependency Pairs & Chains

- Example: Arithmetic on Natural Numbers

$$+(0, y) \xrightarrow{R} y \quad (\textsc{Base}+) \qquad +(\mathsf{s}(x), y) \xrightarrow{R} \mathsf{s}(+(x, y)) \quad (\textsc{Rec}+)$$

$$\cdot(0, y) \xrightarrow{R} 0 \quad (\textsc{Base}\cdot) \qquad \cdot(\mathsf{s}(x), y) \xrightarrow{R} +(\cdot(x, y), y) \quad (\textsc{Rec}\cdot)$$

- The dependency pairs for this TRS are

$$+(\mathsf{s}(x), y) \xrightarrow{DP} +(x, y) \qquad \cdot(\mathsf{s}(x), y) \xrightarrow{DP} \cdot(x, y) \qquad \cdot(\mathsf{s}(x), y) \xrightarrow{DP} +(\cdot(x, y), y)$$

- Repeating the first rule 2 times is a finite chain

$$+(\mathsf{s}(x), y) \xrightarrow{DP} +(x, y)\ , \qquad +(\mathsf{s}(x), y) \xrightarrow{DP} +(x, y)$$

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

$$+(\mathsf{s}(\mathsf{s}(0)), 0) \longrightarrow +(\mathsf{s}(0), 0) \longrightarrow_s^* \qquad +(\mathsf{s}(0), 0) \longrightarrow +(0, 0)$$

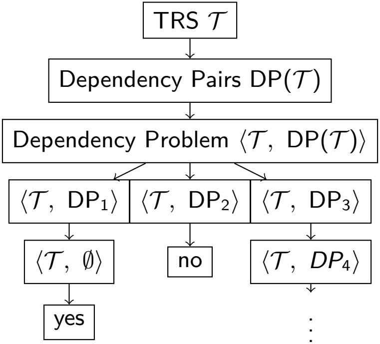
Dependency Pair Technique

Dependency Problems

- A *dependency problem* $P = \langle \mathcal{T},\ D \rangle$ consists of
  - a TRS $\mathcal{T}$
  - a set of dependency pairs D
- P is *solved* by proving the absence or existence of infinite chains of D in $\mathcal{T}$.
- Solving $\langle \mathcal{T},\ \mathrm{DP}(\mathcal{T}) \rangle$ answers whether $\mathcal{T}$ 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 DP($\mathcal{T}$)
- Start with problem $\langle \mathcal{T},\ \mathrm{DP}(\mathcal{T}) \rangle$
- Split with dependency processors
- Repeat until solved or timeout
- $\mathrm{DP}_i = \emptyset \rightarrow$ no chains $\rightarrow$ terminates
- Dependency processors as extension mechanism

Dependency Pair Technique

Dependency Graph Processor

- Input: dependency problem $\langle \mathcal{T},\ \mathrm{DP} \rangle$
- 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 $\mathcal{T}$
- Output: $\{\mathcal{T}\} \times \mathrm{SCCs}(G)$
- 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 $\langle \mathcal{T},\ \mathrm{DP}(\mathcal{T}) \rangle$ for our example

$$+(\mathsf{s}(x), y) \xrightarrow{DP} +(x, y) \qquad \cdot(\mathsf{s}(x), y) \xrightarrow{DP} \cdot(x, y) \qquad \cdot(\mathsf{s}(x), y) \xrightarrow{DP} +(\cdot(x, y), y)$$

- Dependency graph for our example

$$\begin{tikzcd}
\boxed{\cdot(\mathsf{s}(x), y) \xrightarrow{DP} \cdot(x, y)} \arrow[loop left] \arrow[dr] & & \boxed{+(\mathsf{s}(x), y) \xrightarrow{DP} +(x, y)} \arrow[loop right] \\
& \boxed{\cdot(\mathsf{s}(x), y) \xrightarrow{DP} +(\cdot(x, y), y)} \arrow[ur] &
\end{tikzcd}$$

- 2 output problems

$$\left\langle \mathcal{T},\ \left\{ \cdot(\mathsf{s}(x), y) \xrightarrow{DP} \cdot(x, y) \right\} \right\rangle \quad \left\langle \mathcal{T},\ \left\{ +(\mathsf{s}(x), y) \xrightarrow{DP} +(x, y) \right\} \right\rangle$$

Dependency Pair Technique

Reduction Pair Processor

- Idea: well-founded order $\succ$ on terms implies termination
- Forall 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.

Dependency Pair Technique

Reduction Pair Processor

- We can do better!
- We want to show that there is no infinite chain for a DP problem
  - no infinite sequence $l_1 \xrightarrow{DP} r_1, l_2 \xrightarrow{DP} r_2, \ldots$ with $\forall i.\ \exists \sigma.\ \sigma\,[r_i] \longrightarrow_s^* \sigma\,[l_{i+1}]$.
- Only the dependency pairs have to make the terms smaller wrt. $\succ$
- For TRS rules it suffices to not make the terms bigger, as $\sigma\,[r_i] \longrightarrow_s^* \sigma\,[l_{i+1}]$ terminates anyway.
- Hence, for the absence of infinite chains, it suffices to show
  - $l \succ r$ for all dependency pairs $l \xrightarrow{DP} r$
  - $l \succcurlyeq r$ for all TRS rules $l \xrightarrow{R} r$

Dependency Pair Technique

Reduction Pair Processor

- In our example we have two dependency problems left

$$\left\langle \mathcal{T},\ \left\{ \cdot(\mathsf{s}(x), y) \xrightarrow{DP} \cdot(x, y) \right\} \right\rangle \quad \left\langle \mathcal{T},\ \left\{ +(\mathsf{s}(x), y) \xrightarrow{DP} +(x, y) \right\} \right\rangle$$

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

$$\begin{aligned} +(\mathsf{s}(x), y) &\succ_1 +(x, y) \\ +(0, y) &\succcurlyeq_i y \\ \cdot(0, y) &\succcurlyeq_i 0 \end{aligned} \qquad \begin{aligned} \cdot(\mathsf{s}(x), y) &\succ_2 \cdot(x, y) \\ +(\mathsf{s}(x), y) &\succcurlyeq_i \mathsf{s}(+(x, y)) \\ \cdot(\mathsf{s}(x), y) &\succcurlyeq_i +(\cdot(x, y), y) \end{aligned}$$

- The lexicographic order with $0 < \mathsf{s} < + < \cdot$
  - 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