Higher Order Dependency Pairs With Argument Filterings

Cynthia Kop and Femke van Raamsdonk

Vrije Universiteit, Department of Theoretical Computer Science

Abstract. We present a termination method for left-linear Higher-order Rewrite Systems (HRSs) that are algebraic using a higher-order generalization of dependency pairs with argument filterings.

1 Introduction

An important method to (automatically) prove termination of first-order term rewriting is the dependency pair framework by Arts and Giesl [3]. This approach transforms a term rewriting system into a set of ordering constraints, to be satisfied by a well-founded ordering. This ordering, which is not required to be monotonic, can be further simplified using for example argument filterings.

Extending the dependency pair method to higher-order rewriting turns out to be difficult. A very natural extension is defined in [6], but the simplicity comes at a price: the dependency ordering must satisfy the *subterm property.* This property, which requires a term to be greater than its subterms, makes definition and use of argument filterings problematic. Moreover, well-foundedness of the relation is no longer equivalent to termination of the system.

Other extensions crucially rely on some restriction of the higher-order aspect, either by disallowing abstractions altogether (as is done in Simply Typed Term Rewriting Systems [1]) or by placing limitations on the rules; such restrictions are right-linear or non-nested [5], or plain-function-passing [4]. The present work continues on this line of research by restricting attention to *algebraic and left-linear HRSs.* Left-hand sides of algebraic HRSs only contain abstractions in a very simple form; right-hand sides are not restricted. This yields a natural class, which contains for instance functional programs in most common languages.

2 Algebraic Rules

We assume the reader is familiar with Nipkow's pattern HRSs. We will consider left-linear and *algebraic* HRSs, where left-hand sides have a basic form.

Definition 1. *A term is* simple *if it is either (the eta-long form of) a variable, or has the form $f(s_1, \dots, s_n)$ with $f \in \mathcal{F}$ and all s_i simple. A rewrite rule $l \to r$ is* algebraic *if l is simple. An HRS is algebraic if all its rewrite rules are algebraic.*

Every simple term is a pattern, but not every pattern is a simple term. As examples, consider the non-simple terms $\lambda x.\, \mathtt{o}$, $\lambda x.\, \mathtt{s}(x)$, $\lambda x.\, Z$ and $X(\mathtt{o})$. The rules $\mathtt{map}(\lambda x.\, F(x), \mathtt{cons}(H, T)) \to \mathtt{cons}(F(H), \mathtt{map}(\lambda x.\, F(x), T))$ and $\mathtt{up}(X) \to \mathtt{map}(\lambda x.\, s(x), X)$ are algebraic; note that this notion only concerns the left-hand side of a rule.

Algebraic HRSs clearly form a restriction of the usual HRSs. One might compare them to simply typed term rewriting systems (STTRSs) [1,2], but unlike STTRSs, abstractions are permitted in terms. To note the difference, consider the algebraic system LAMBDA with function symbols $@ : o \Rightarrow o \Rightarrow o$ and $\Lambda : (o \Rightarrow o) \Rightarrow o$, and single rule $@(\Lambda(\lambda x.\, Z(x)), Y) \to Z(Y)$. In an STTRS this system would be terminating, as every rule makes the term shorter. As an HRS, this system implements untyped lambda-calculus, and is evidently not terminating!

All rewrite systems where the left-hand sides are η-equivalent to abstraction-free terms can be transformed into an algebraic system, without affecting termination. As this includes the HRS-representation of functional programs in any of the common languages, the restriction does not seem excessive.

The reason to consider algebraic HRSs is that in combination with left-linearity steps which take place below an abstraction can be postponed.

Definition 2. *A step $s \to t$ is* algebraic *if it does not occur inside an abstraction: either $s = l\gamma$, $t = r\gamma$, $l \to r \in \mathcal{R}$ or $s = f(s_1, \dots, s_i, \dots, s_n)$, $t = f(s_1, \dots, s_i', \dots, s_n)$ and $s_i \to s_i'$ by an algebraic step.*

Lemma 1. *In an algebraic and left-linear HRS non-algebraic steps can be postponed: if $s \to^* t$ there exists q such that $s \to^* q$ algebraicly and $q \to^* t$ non-algebraicly.*

From now on, we consider a fixed algebraic and left-linear HRS.

3 Dependency Pairs, Chains and Orderings

The basic idea of dependency pairs is to identify *minimal non-terminating* terms. To this end we combine the rewrite relation with a subterm relation for HRSs.

Definition 3. *The* subterm relation*, notation $\trianglerighteq$, is generated by the clauses:*

1. *$s \trianglerighteq s$,*
2. *$a(s_1, \dots, s_n) \trianglerighteq t$ if $s_i \trianglerighteq t$ for some i, for $a \in \mathcal{V} \cup \mathcal{F}$,*
3. *$\lambda \boldsymbol{x}.\, s \trianglerighteq t$ if $s[\boldsymbol{x} := \boldsymbol{c}] \trianglerighteq t$.*

Here, $c_\sigma : \sigma$ is a new symbol for each type σ. The substitution $[\boldsymbol{x} := \boldsymbol{c}]$ maps every variable $x_i : \sigma$ in $\{\boldsymbol{x}\}$ to c_σ.

The subterms of $f(\lambda x.\, X(x))$ are $f(\lambda x.\, X(x))$, $\lambda x.\, X(x)$, $X(c)$, and c. We have just a single constant c_σ for every type σ; this doesn't cause problems due to left-linearity. The relation $\trianglerighteq$ has all the nice properties of the normal subterm relation; in particular $\to_{\mathcal{R}} \cdot \trianglerighteq$ is terminating if and only if $\to_{\mathcal{R}}$ is terminating.

Let $\mathcal{F}^{\#}$ denote $\mathcal{F}$ extended with for every defined symbol f its marked version $f^{\#}$ with the same arity. Let $\mathcal{F}_c = \mathcal{F} \cup C$, where C contains the fresh constants c_σ. Let $\mathcal{F}_c^{\#} = \mathcal{F}^{\#} \cup C$. We define $s^{\#}$ as $f^{\#}(s_1, \dots, s_n)$ if $s = f(s_1, \dots, s_n)$ with f defined and $s^{\#} = s$ otherwise. Now we are ready to define dependency pairs.

Definition 4. *Given a rewrite rule $l \to r$, the pair of terms $l^{\#} \rightsquigarrow p^{\#}$ is a* dependency pair *for $l \to r$ if $p \trianglelefteq r$ and either p is headed by a defined symbol, or $p = X(s_1, \dots, s_n)$ with X a free variable in r and $n > 0$. The set of dependency pairs of $\mathcal{R}$ is denoted by $\mathsf{DP}(\mathcal{R})$, or by DP if $\mathcal{R}$ is clear from the context.*

For example, in the HRS $\{\mathtt{f}(\lambda x.\, Z(x)) \to Z(\mathtt{g}(\mathtt{a})), \mathtt{g}(x) \to \mathtt{h}(x)\}$, the dependency pairs are: $\mathtt{f}^{\#}(\lambda x.\, Z(x)) \rightsquigarrow Z(\mathtt{g}(\mathtt{a}))$ and $\mathtt{f}^{\#}(\lambda x.\, Z(x)) \rightsquigarrow \mathtt{g}^{\#}(\mathtt{a})$.

Dependency pairs find their use in the notion of a dependency chain: a sequence of dependency pairs with certain properties.

Definition 5. *A* dependency chain *for $\mathcal{R}$ is a sequence $[(l_i, p_i, t_i, \gamma_i) \mid i \in A]$ with either $A = \mathbb{N}$ or $A = \{1, \dots, N\}$ for some N such that, for all i:*

- *$l_i \rightsquigarrow p_i \in \mathsf{DP}(\mathcal{R})$,*
- *If p_i is headed by $f^{\#}$ for some f, then $t_i = p_i\gamma_i$,*
- *If p_i is headed by a variable, then there is q such that $t_i = q^{\#}$ and $p_i\gamma_i \trianglerighteq q$, but not $q \trianglelefteq \gamma_i(x)$ for any x, nor $q \trianglelefteq s\gamma_i$ for any $s \triangleleft p_i$,*
- *$t_i \to^*_{\mathcal{R},in} l_{i+1}\gamma_{i+1}$.*

The dependency chain is safe *if always $t_i \to^*_{\mathcal{R},in} l_{i+1}\gamma_{i+1}$ by only algebraic steps.*

Theorem 1. *An algebraic and left-linear HRS is terminating if it has no infinite safe dependency chain.*

The proof is straightforward; the *safe* condition holds because non-algebraic reductions can be postponed. Note that the condition in the theorem is sufficient, not required. There are terminating HRSs which admit an infinite (safe) dependency chain. This is because a non-terminating subterm may be beta-reduced away immediately.

To prove non-existence of a safe dependency chain we generate a set of inequalities from $\mathsf{DP}(\mathcal{R})$. If these inequalities can be satisfied by a so-called *quasi-monotonic ordering pair*, termination follows.

Definition 6. *A* quasi-monotonic ordering pair *is a pair $(>, \geq)$ of an ordering and a quasi-ordering, comparing base-type terms of equal types) such that:*

1. *$>$ and $\geq$* combine*: $> \cdot \geq$ is contained in $>$,*
2. *if s_i has base type and $s_i \geq s_i'$, then $f(s_1, \dots, s_i, \dots, s_n) \geq f(s_1, \dots, s_i', \dots, s_n)$*
3. *if s is simple then $s > t$ implies $s\gamma > t\gamma$ and $s \geq t$ implies $s\gamma \geq t\gamma$*

We say a quasi-monotonic ordering pair $(>, \geq)$ is well-founded if $>$ is well-founded. Comparing the definition for an ordering pair to similar definitions in the literature, requirements 2 and 3 are weakened versions from having the pair quasi-monotonic and closed under substitution.

Definition 7. *A* dependency ordering *for* DP *and* $\mathcal{R}$ *is a quasi-monotonic ordering pair* $(>, \geq)$ *satisfying the following inequalities:*

1. $l > p$ *for all dependency pairs* $l \rightsquigarrow p \in \mathsf{DP}$,
2. $l \geq r$ *for all rules* $l \to r \in \mathcal{R}$,
3. $l\gamma > t^{\#}$ *for all pairs* $l \rightsquigarrow p \in \mathsf{DP}$ *with* p *headed by a variable* X, *substitution* γ *and* $t \trianglelefteq p\gamma$ *such that* t *is headed by a defined symbol and neither* $t \trianglelefteq \gamma(X)$ *or* $t \trianglelefteq s\gamma$ *for any direct subterm* s *of* p.

The first two of these requirements are usual for dependency orderings; requirement 3 replaces the *subterm property* in [6]. Requirement 3 always holds if $\trianglerighteq$ is contained in $\geq$ (since then $l\gamma > p\gamma \geq t$), but there may be other ways.

Theorem 2. $\mathsf{DP}(\mathcal{R})$ *and* $\mathcal{R}$ *admit a well-founded dependency ordering if and only if there is no infinite safe dependency chain over* $\mathcal{R}$.

4 Argument Filterings

The idea behind argument filterings is to simplify the constraints on the rewrite rules and the dependency pairs by omitting some arguments of function symbols.

An argument filtering is defined as a partial function that maps a symbol $f : \sigma_1 \Rightarrow \ldots \Rightarrow \sigma_n \Rightarrow b$ in $\mathcal{F}^{\#}$ to either some $i \in \{1, \ldots, n\}$ (the collapsing case), or to a sequence $[i_1, \ldots, i_k]$ with $1 \leq i_1 < i_2 < \ldots < i_k \leq n$. In the collapsing case σ_i should have output-type b. Note that both unmarked and marked symbols are filtered, but the symbols c_σ, used to replace bound variables, are not.

Given an argument filtering A, let $\mathcal{F}_A$ contain all symbols in $\mathcal{F}_c^{\#}$, and additionally for every function symbol $f : \sigma_1 \Rightarrow \ldots \Rightarrow \sigma_n \Rightarrow b$ in $\mathcal{F}^{\#}$ with $A(f) = [i_1, \ldots, i_k]$ a fresh filtered function symbol $f_A : \sigma_{i_1} \Rightarrow \ldots \Rightarrow \sigma_{i_k} \Rightarrow b$. In the argument filtering of a term a symbol is not filtered if one of its arguments contains a bound variable (clause 2), and in the collapsing case the fresh constants c are used.

Definition 8. *Given an argument filtering* A, *the term filtering* $\mathtt{fil}$ *is defined as* $\mathtt{fil}(s) = \mathtt{fil}_\emptyset(s)$, *where the auxiliary mapping* $\mathtt{fil}_X$ *is defined by the clauses:*

1. $\mathtt{fil}_X(\lambda y.\, s) = \lambda y.\, \mathtt{fil}_{X \cup \{y\}}(s)$,
2. $\mathtt{fil}_X(a(s_1, \ldots, s_n)) = a(\mathtt{fil}_X(s_1), \ldots, \mathtt{fil}_X(s_n))$
if $a \notin \mathrm{Dom}(A)$ *or* $X \cap FV(a(s_1, \ldots, s_n)) \neq \emptyset$,
3. $\mathtt{fil}_X(a(s_1, \ldots, s_n)) = a^{\#}(\mathtt{fil}(s_{i_1}), \ldots, \mathtt{fil}(s_{i_k}))$
if $a \in \mathrm{Dom}(A)$, $X \cap FV(a(s_1, \ldots, s_n)) = \emptyset$ *and* $A(a) = [i_1, \ldots, i_k]$,
4. $\mathtt{fil}_X(a(s_1, \ldots, \lambda \boldsymbol{x}.\, s_i, \ldots, s_n)) = \mathtt{fil}_{\{\boldsymbol{x}\}}(s_i)[\boldsymbol{x} := \boldsymbol{c}]$
if $a \in \mathrm{Dom}(A)$, $X \cap FV(a(s_1, \ldots, s_n)) = \emptyset$ *and* $A(a) = i$ *(*$\boldsymbol{x}$ *may be empty).*

For example, let $\mathcal{F} = \{f : o \Rightarrow o \Rightarrow o, a : o\}$ and $A(f) = []$. Then $\mathtt{fil}_\emptyset(\lambda x.\, f(x, f(a, a))) = \lambda x.\, \mathtt{fil}_{\{x\}}(f(x, f(a, a))) = \lambda x.\, f(\mathtt{fil}_{\{x\}}(x), \mathtt{fil}_{\{x\}}(f(a, a))) = \lambda x.\, f(x, f_A)$. In a filtered term a symbol might both occur in normal and filtered form. The filtered and unfiltered symbols should be considered as different symbols.

To each ordering pair $(\succ, \succeq)$ over $\mathcal{T}(\mathcal{F}_A)$, we associate a pair $(>, \geq)$ over $\mathcal{T}(\mathcal{F}^{\#})$ in the natural way: $s > t$ iff $\mathtt{fil}(s) \succ \mathtt{fil}(t)$, $s \geq t$ iff $\mathtt{fil}(s) \succeq \mathtt{fil}(t)$.

Theorem 3. *Let $(\succ, \succeq)$ be a quasi-monotonic ordering pair on $\mathcal{T}(\mathcal{F}_A)$ which satisfies the following requirements:*

1. *$\succ$ contains the superterm relation $\rhd$ (but doesn't have to be monotonic),*
2. *always $f(x_1, \ldots, x_n) \succeq \mathtt{fil}(f(x_1, \ldots, x_n))$ if all x_i are (free) variables, and also $f(x_1, \ldots, x_n) \succeq \mathtt{fil}(f^{\#}(x_1, \ldots, x_n))$ for defined symbols f,*
3. *$\mathtt{fil}(l) \succ \mathtt{fil}(p)$ for all $l \rightsquigarrow p \in \mathsf{DP}(\mathcal{R})$,*
4. *$\mathtt{fil}(l) \succeq \mathtt{fil}(r)$ for all $l \to r \in \mathcal{R}$.*

Then the associated pair is a dependency filtering for $\mathcal{R}$; well-founded iff $\succ$ is.

That is, to prove termination of $\mathcal{R}$ it suffices to find a well-founded ordering pair over the filtered rules, satisfying the requirements in the theorem.

5 Concluding Remarks

In this paper we have defined a new generalization of dependency pairs with argument filterings for left-linear HRSs that are algebraic. algebraic. For left-linear first-order TRSs, our definition of dependency pairs coincides with the original one. Also, a higher-order generalization of argument filterings is given. The work has been done for HRSs, but the results are equally viable for different styles of higher-order rewriting, as long as the rules are presented in eta-long form. The results have been implemented in a tool, WANDA, which will participate in the termination competition of 2010.

For future work, we aim to further investigate dependency graphs and less standard filterings (for instance replacing a term $f(s_1, s_m)$ by $s_1 \cdot s_2$). We also intend to look into non-eta-normal algebraic systems, without transforming them.

References

1. T. Aoto and Y. Yamada. Dependency pairs for simply typed term rewriting. In J. Giesl, editor, *Proceedings of the 6th International Conference on Rewriting Techniques and Applications (RTA 2005)*, volume 3467 of *LNCS*, pages 120–134, Nara, Japan, April 2005. Springer Verlag.
2. T. Aoto and Y. Yamada. Argument filterings and usable rules for simply typed dependency pairs (extended abstract). In *Proceedings of the 4th International Workshop on Higher-Order Rewriting (HOR 2007)*, pages 21–27, Paris, France, June 2007. Workshop proceedings.
3. T. Arts and J. Giesl. Termination of term rewriting using dependency pairs. *Theoretical Computer Science*, 236:133–178, 2000.
4. K. Kusakari, Y. Isogai, M. Sakai, and F. Blanqui. Static dependency pair method based on strong computability for higher-order rewrite systems. *IEICE Transactions on Information and Systems*, 92(10):2007–2015, 2009.
5. M. Sakai and K. Kusakari. On dependency pair method for proving termination of higher-order rewrite systems. *IEICE Transactions on Information and Systems*, E88-D(3):583–593, 2005.
6. Y. Sakai, M. Watanabe and T. Sakabe. An extension of the dependency pair method for proving termination of higher-order rewrite systems. *IEICE Transactions on Information and Systems*, E84-D(8):1025–1032, 2001.