1 Higher-order dependency pairs

Frédéric Blanqui

LORIA*, Campus Scientifique, BP 239, 54506 Vandoeuvre-lès-Nancy, France

Abstract. Arts and Giesl proved that the termination of a first-order rewrite system can be reduced to the study of its "dependency pairs". We extend these results to rewrite systems on simply typed λ-terms by using Tait's computability technique.

1.1 Introduction

Let $\mathcal{F}$ be a set of function symbols, $\mathcal{X}$ be a set of variables and $\mathcal{R}$ be a set of rewrite rules over the set $\mathcal{T}(\mathcal{F},\mathcal{X})$ of first-order terms. Let $\mathcal{D}$ be the set of symbols occuring at the top of a rule left hand-side and $\mathcal{C} = \mathcal{F} \setminus \mathcal{D}$. The set $\mathcal{DP}(\mathcal{R})$ of *dependency pairs* of $\mathcal{R}$ is the set of pairs (l,t) such that l is the left hand-side of a rule $l \to r \in \mathcal{R}$ and t is a subterm of r headed by some symbol $f \in \mathcal{D}$. The term t represents a potential recursive call. The chain relation is $\to_{\mathrm{C}} = \to^*_{\mathcal{R}i} \to_{\mathcal{DP}h}$, where $\to^*_{\mathcal{R}i}$ is the reflexive and transitive closure of the restriction of $\to_{\mathcal{R}}$ to non-top positions and $\to_{\mathcal{DP}h}$ is the restriction of $\to_{\mathcal{DP}}$ to top positions. Arts and Giesl prove in [1] that $\to_{\mathcal{R}}$ is strongly normalizing (SN) (or terminating, well-founded) iff the chain relation so is. Moreover, $\to_{\mathrm{C}}$ is terminating if there is a weak reduction ordering $>$ such that $\mathcal{R} \subseteq \geq$ and $\mathcal{DP}(\mathcal{R}) \subseteq >$ (only dependency pairs need to strictly decrease).

We would like to extend these results to higher-order rewriting. There are several approaches to higher-order rewriting. In Higher-order Rewrite Systems (HRSs) [7], terms and rules are simply typed λ-terms in β-normal η-long form, left hand-sides are patterns à la Miller and matching is modulo $\beta\eta$. An extension of dependency pairs for HRSs is studied in [10,9]. In Combinatory Reduction Systems (CRSs) [6], terms are λ-terms, rules are λ-terms with meta-variables, left hand-sides are patterns à la Miller and matching uses α-conversion and some variable occur-checks. The relation between the two kinds of rewriting is studied in [12]. It appears that the matching algorithms are similar and that, in HRSs, one does more β-reductions after having applied the matching substitution. But, in both cases, β-reduction is used at the meta-level for normalizing right hand-sides after the application of the matching substitution. So, a third more atomic approach is to have no meta-level β-reduction and add β-reduction at the object level. This is the approach that we consider in this paper.

So, we assume given a set $\mathcal{R}$ of rewrite rules made of simply typed λ-terms and study the termination of $\to_\beta \cup \to_{\mathcal{R}}$ when using CRS-like matching. This

* UMR 7503 CNRS-INPL-INRIA-Nancy2-UHP

clearly implies the termination of $\to_{\mathcal{R}}$ in the corresponding CRS or HRS. Another advantage of this approach is that we can rely on Tait's technique for proving termination [11,3]. This paper explores its use with dependency pairs. This is in contrast with [10,9].

In Tait's technique, to each type T, one associates a set $[\![T]\!]$ of terms of type T. Terms of $[\![T]\!]$ are said *computable*. Before giving some properties of computable terms, let us introduce a few definitions. The sets $\mathrm{Pos}^+(T)$ and $\mathrm{Pos}^-(T)$ of *positive and negative positions* in T are defined as follows:

- $\mathrm{Pos}^+(B) = \{\varepsilon\}$ and $\mathrm{Pos}^-(B) = \emptyset$ if B is a base type,
- $\mathrm{Pos}^\delta(T \Rightarrow U) = 1 \cdot \mathrm{Pos}^{-\delta}(T) \cup 2 \cdot \mathrm{Pos}^\delta(U)$.

We use $\boldsymbol{T}$ to denote a sequence of types $T_1, \ldots, T_n$ of length $|\boldsymbol{T}| = n$. The i-th argument of a function symbol $f : \boldsymbol{T} \Rightarrow B$ is *accessible* if B occurs only positively in T_i. Let $\mathrm{Acc}(f)$ be the set of indexes of the accessible arguments of f. A base type B is *basic* if, for all $f : \boldsymbol{T} \Rightarrow B$ and $i \in \mathrm{Acc}(f)$, T_i is a base type. After [3,4], given a relation R, *computability wrt* R can be defined so that the following properties are satisfied:

(1) A computable term is strongly normalizable wrt $\to_\beta \cup R$.
(2) A term of basic type is computable if it is SN wrt $\to_\beta \cup R$.
(3) A term $v^{T \Rightarrow U}$ is computable if, for all t^T computable, vt is computable.
(4) If t is computable then every reduct of t is computable.
(5) A term $f\boldsymbol{t}$ is computable if all its reducts wrt $\to_\beta \cup R$ are computable.
(6) If $f\boldsymbol{t}$ is computable then, for all $i \in \mathrm{Acc}(f)$, t_i is computable.
(7) If t contains no $f \in \mathcal{D}$ and σ is computable, then $t\sigma$ is computable.
(8) Every term is computable whenever every $f \in \mathcal{D}$ is computable.

1.2 Admissible rules

An important property of the first-order case is that, given a term t, a substitution σ and a variable $x \in \mathcal{V}(t)$, $x\sigma$ is strongly normalizable whenever $t\sigma$ so is. This is not always true in the higher-order case. So, we need to introduce some restrictions on rules to keep this property.

Definition 1 (Admissible rules) A rule $f\boldsymbol{l} \to r$ is *admissible* if $\mathrm{FV}(r) \subseteq \mathrm{PCC}(\boldsymbol{l})$, where PCC is defined in Figure 1.1.

The Pattern Computability Closure (PCC) is called accessibility in [2]. It includes most usual higher-order patterns [8].

Lemma 2 If $f\boldsymbol{l} \to r$ is admissible, $\mathrm{dom}(\sigma) \subseteq \mathrm{FV}(\boldsymbol{l})$ and $\boldsymbol{l}\sigma$ is computable, then $\sigma|_{\mathrm{FV}(r)}$ is computable.

Proof. We prove by induction that, for all $u \in \mathrm{PCC}(\boldsymbol{t})$ and computable substitution θ such that $\mathrm{dom}(\theta) \subseteq \mathrm{FV}(u) \setminus \mathrm{FV}(\boldsymbol{t})$, $u\sigma\theta$ is computable.

Fig. 1.1. Pattern Computability Closure [2]

$$\text{(arg)}\quad t_i \in \mathrm{PCC}(\boldsymbol{t})$$

$$\text{(acc)}\quad \frac{g\boldsymbol{u} \in \mathrm{PCC}(\boldsymbol{t}) \quad i \in \mathrm{Acc}(g)}{u_i \in \mathrm{PCC}(\boldsymbol{t})}$$

$$\text{(lam)}\quad \frac{\lambda y u \in \mathrm{PCC}(\boldsymbol{t}) \quad y \notin \mathrm{FV}(\boldsymbol{t})}{u \in \mathrm{PCC}(\boldsymbol{t})}$$

$$\text{(app-left)}\quad \frac{uy \in \mathrm{PCC}(\boldsymbol{t}) \quad y \notin \mathrm{FV}(\boldsymbol{t}) \cup \mathrm{FV}(u)}{u \in \mathrm{PCC}(\boldsymbol{t})}$$

$$\text{(app-right)}\quad \frac{y^{U \Rightarrow \boldsymbol{T} \Rightarrow U} u \in \mathrm{PCC}(\boldsymbol{t}) \quad y \notin \mathrm{FV}(\boldsymbol{t}) \cup \mathrm{FV}(u)}{u \in \mathrm{PCC}(\boldsymbol{t})}$$

(arg) Since $\mathrm{dom}(\theta) = \emptyset$, $l_i\sigma\theta = l_i\sigma$ is computable by assumption.

(acc) By induction hypothesis, $g\boldsymbol{u}\sigma$ is computable. Thus, by property (6), $u_i\sigma$ is computable.

(lam) Let $\theta' = \theta|_{\mathrm{dom}(\theta)\setminus\{y\}}$. Wlog, we can assume that $y \notin \mathrm{codom}(\sigma\theta)$. Hence, $(\lambda yu)\sigma\theta' = \lambda yu\sigma\theta'$. Now, since $\mathrm{dom}(\theta) \subseteq \mathrm{FV}(u)\setminus\mathrm{FV}(\boldsymbol{t})$, $\mathrm{dom}(\theta') \subseteq \mathrm{FV}(\lambda yu) \setminus \mathrm{FV}(\boldsymbol{t})$. Thus, by induction hypothesis, $\lambda yu\sigma\theta'$ is computable. Since $y\theta$ is computable, by (3), $(\lambda yu\sigma\theta')y\theta$ is computable and, by (4), $u\sigma\theta'\{y \mapsto y\theta\}$ is computable. Finally, since $y \notin \mathrm{dom}(\sigma\theta') \cup \mathrm{codom}(\sigma\theta')$, $u\sigma\theta'\{y \mapsto y\theta\} = u\sigma\theta$.

(app-left) Let $v : T_y$ computable and $\theta' = \theta \cup \{y \mapsto v\}$. Since $\mathrm{dom}(\theta) \subseteq \mathrm{FV}(u) \setminus \mathrm{FV}(\boldsymbol{t})$ and $y \notin \mathrm{FV}(\boldsymbol{t})$, $\mathrm{dom}(\theta') = \mathrm{dom}(\theta) \cup \{y\} \subseteq \mathrm{FV}(uy) \setminus \mathrm{FV}(\boldsymbol{t})$. Thus, by induction hypothesis, $(uy)\sigma\theta' = u\sigma\theta' v$ is computable. Since $y \notin \mathrm{FV}(u)$, $u\sigma\theta' = u\sigma\theta$. Thus, $u\sigma\theta$ is computable.

(app-right) Let $v = \lambda x^U \lambda \boldsymbol{y}^{\boldsymbol{T}} x$ and $\theta' = \theta \cup \{y \mapsto v\}$. By (3), v is computable. Since $\mathrm{dom}(\theta) \subseteq \mathrm{FV}(u) \setminus \mathrm{FV}(\boldsymbol{t})$ and $y \notin \mathrm{FV}(\boldsymbol{t})$, $\mathrm{dom}(\theta') \subseteq \mathrm{FV}(yu) \setminus \mathrm{FV}(\boldsymbol{t})$. Thus, by induction hypothesis, $(yu)\sigma\theta' = vu\sigma\theta'$ is computable. Since $y \notin \mathrm{FV}(u)$, $u\sigma\theta' = u\sigma\theta$. Thus, by (4), $u\sigma\theta$ is computable. □

1.3 Higher-order dependency pairs

In the following, we assume given a set $\mathcal{R}$ of admissible rules. The sets $\mathrm{FAP}(t)$ of *full application positions* of a term t and the *level* of a term t are defined as follows:

- $\mathrm{FAP}(x) = \emptyset$ and $\mathrm{level}(x) = 0$
- $\mathrm{FAP}(\lambda xt) = 1 \cdot \mathrm{FAP}(t)$ and $\mathrm{level}(\lambda xt) = \mathrm{level}(t)$

If $f \in \mathcal{D}$ then:

- $\mathrm{level}(ft_1 \ldots t_n) = 1 + max\{\mathrm{level}(t_i) \mid 1 \le i \le n\}$

– $\mathrm{FAP}(ft_1 \dots t_n) = \{\varepsilon\} \cup \bigcup_{i=1}^n 1^{n-i}2 \cdot \mathrm{FAP}(t_i)$

If $t \neq ft_1 \dots t_n$ with $f \in \mathcal{D}$, then $\mathrm{FAP}(tu) = 1 \cdot \mathrm{FAP}(t) \cup 2 \cdot \mathrm{FAP}(u)$ and $\mathrm{level}(tu) = max\{\mathrm{level}(t), \mathrm{level}(u)\}$.

Definition 3 (Dependency pairs) The set of *dependency pairs* is $\mathcal{DP} = \{l \to r|_p \mid l \to r \in \mathcal{R},\, p \in \mathrm{FAP}(r)\}$. The *chain relation* is $\to_{\mathrm{C}} = \to_{\mathcal{R}i}^* \to_{\mathcal{DP}h}$, where $\to_{\mathcal{R}i}$ is the restriction of $\to_{\mathcal{R}}$ to non-top positions, and $\to_{\mathcal{DP}h}$ is the restriction of $\to_{\mathcal{DP}}$ to top positions.

If, for all $l \to r \in \mathcal{DP}$, $\mathrm{FV}(r) \subseteq \mathrm{FV}(l)$, we have $\to_{\mathrm{C}} \subseteq \to_{\mathcal{R}}^+ \trianglerighteq$. Hence, $\to_{\beta\mathrm{C}}$ is terminating whenever $\to_{\beta\mathcal{R}}$ so is. We now prove the converse:

Theorem 4 Assume that, for all $l \to r \in \mathcal{R}$ and $p \in \mathrm{FAP}(r)$, $\mathrm{FV}(r|_p) \subseteq \mathrm{FV}(r)$ and $r|_p$ has the type of l (*). Then, $\to_{\beta\mathcal{R}}$ is terminating if $\to_{\beta\mathrm{C}}$ so is.

Proof. By (1), this is so if every term is computable wrt $\to_{\mathcal{R}}$. By (8), this is so if every $f^{\boldsymbol{T}\Rightarrow B} \in \mathcal{D}$ is computable. By (3), this is so if, for all $\boldsymbol{t} : \boldsymbol{T}$ computable, $f\boldsymbol{t}$ is computable. We prove it by induction on $(f\boldsymbol{t}, \boldsymbol{t})$ with $(\to_{\mathrm{C}}, (\to_{\beta\mathcal{R}})_{\mathrm{lex}})_{\mathrm{lex}}$ as well-founded ordering (H1). Indeed, by (1), $\boldsymbol{t}$ are strongly normalizable wrt $\to_{\beta\mathcal{R}}$. By (5), it suffices to prove that every reduct of $f\boldsymbol{t}$ is computable. If $\boldsymbol{t} \to_{\beta\mathcal{R}} \boldsymbol{t}'$ then, by (H1), $f\boldsymbol{t}'$ is computable since, by (4), $\boldsymbol{t}'$ are computable and $\to_{\beta\mathrm{C}}(f\boldsymbol{t}') = \to_{\beta\mathrm{C}}(f\boldsymbol{t})$. Now, assume that there is $f\boldsymbol{l} \to r \in \mathcal{R}$ and σ such that $\boldsymbol{t} = \boldsymbol{l}\sigma$. Since rules are admissible, by Lemma 2, $\sigma' = \sigma|_{\mathrm{FV}(r)}$ is computable. We now prove that $r\sigma'$ is computable by induction on the level n of r (H2). Let $p_1, \dots, p_k$ be the positions in r of the subterms of level $n-1$; $\boldsymbol{y}^i$ be the variables of $\mathrm{FV}(r|_{p_i}) \setminus \mathrm{FV}(r)$; $x_1, \dots, x_k$ be distinct variables not occuring in r; r' be the term obtained by replacing $r|_{p_i}$ by $x_i\boldsymbol{y}^i$ in r; and $\theta = \{x_i \mapsto \lambda\boldsymbol{y}^i r|_{p_i}\sigma'\}$. We have $\mathrm{level}(r') = 0$ and $r'\sigma'\theta \to_\beta^* r\sigma'$. If θ is computable then, by (7), $r'\sigma'\theta$ is computable and we are done. By (*), $\{\boldsymbol{y}^i\} = \emptyset$ and it suffices to prove that $r_{p_i}\sigma'$ is computable. For all $i \leq k$, $r|_{p_i}$ is of the form $g\boldsymbol{u}$ with $\mathrm{level}(u_j) < n$. By (H2), $\boldsymbol{u}\sigma'$ are computable and, since $f\boldsymbol{t} \to_{\mathrm{C}} r|_{p_i}\sigma'$, by (H1), $x_i\theta$ is computable. □

The condition on free variables is an important restriction since it is not satisfied by function calls with bound variables like in $(lim\, F) + x \to lim\, \lambda n(Fn + x)$.

Theorem 5 An higher-order reduction pair is two relations $(>, \geq)$ such that:

– $>$ is well-founded and stable by substitution,
– $\geq$ is a reflexive and transitive rewrite relation containing $\to_\beta$,
– $\geq \circ > \subseteq >$.

In the conditions of Theorem 4, $\to_{\beta\mathrm{C}}$ terminates if $\mathcal{R} \subseteq \geq$ and $\mathcal{DP} \subseteq >$.

Proof. By (1), this is so if every term is computable wrt $\to_{\mathrm{C}}$. By (8), this is so if every $f^{\boldsymbol{T}\Rightarrow B} \in \mathcal{D}$ is computable. By (3), this is so if, for all

$\boldsymbol{t} : \boldsymbol{T}$ computable, $f\boldsymbol{t}$ is computable. We prove it by induction on $(f\boldsymbol{t}, \boldsymbol{t})$ with $(>, (\rightarrow_{\beta\mathcal{R}})_{\mathrm{lex}})_{\mathrm{lex}}$ as well-founded ordering (H1). Indeed, by (1) and Theorem 4, $\boldsymbol{t}$ are strongly normalizable wrt $\rightarrow_{\beta\mathcal{R}}$. By (5), it suffices to prove that every reduct of $f\boldsymbol{t}$ is computable. If $\boldsymbol{t} \rightarrow_{\beta\mathcal{R}} \boldsymbol{t}'$ then, by (H1), $f\boldsymbol{t}'$ is computable since, by (4), $\boldsymbol{t}'$ are computable and $>(f\boldsymbol{t}') \subseteq >(f\boldsymbol{t})$ since $\rightarrow_{\beta\mathcal{R}} \subseteq \geq$ and $\geq \circ > \subseteq >$. Now, assume that there is $f\boldsymbol{l} \rightarrow r \in \mathcal{DP}$ and σ such that $\boldsymbol{t} = \boldsymbol{l}\sigma$. Since rules are admissible, by Lemma 2, $\sigma' = \sigma|_{\mathrm{FV}(r)}$ is computable. Since $\mathcal{DP} \subseteq >$ and $>$ is stable by substitution, $f\boldsymbol{t} > r\sigma'$. Thus, by (H1), $r\sigma'$ is computable. □

An example of reduction pair can be given by using the higher-order recursive path ordering $>_{\mathrm{horpo}}$ [5]. Take $> = (\rightarrow_\beta \cup >_{\mathrm{horpo}})^+$ and $\geq = (\rightarrow_\beta \cup >_{\mathrm{horpo}})^*$. The study of these two relations has to be done. However, $>_{\mathrm{horpo}}$ does not take advantage of the fact that $>$ does not need to be monotonic. Such a relation is given by the weak higher-order recursive computability ordering $>_{\mathrm{whorco}}$, whose monotonic closure strictly contains $>_{\mathrm{horpo}}$ [4]. Moreover, $>_{\mathrm{whorco}}$ is transitive, which is not the case of $>_{\mathrm{horpo}}$. It would therefore be interesting to look for reduction pairs built from $>_{\mathrm{whorco}}$.

References

1. T. Arts and J. Giesl. Termination of term rewriting using dependency pairs. *Theoretical Computer Science*, 236:133–178, 2000.
2. F. Blanqui. Termination and confluence of higher-order rewrite systems. In *Proc. of RTA'00*, LNCS 1833.
3. F. Blanqui. Definitions by rewriting in the Calculus of Constructions. *Mathematical Structures in Computer Science*, 15(1):37–92, 2005.
4. F. Blanqui. (HO)RPO revisited, 2006. Manuscript.
5. J.-P. Jouannaud and A. Rubio. The Higher-Order Recursive Path Ordering. In *Proc. of LICS'99*.
6. J. W. Klop, V. van Oostrom, and F. van Raamsdonk. Combinatory reduction systems. *Theoretical Computer Science*, 121:279–308, 1993.
7. R. Mayr and T. Nipkow. Higher-order rewrite systems and their confluence. *Theoretical Computer Science*, 192(2):3–29, 1998.
8. D. Miller. A logic programming language with lambda-abstraction, function variables, and simple unification. In *Proc. of ELP'89*, LNCS 475.
9. 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.
10. M. Sakai, Y. Watanabe, and T. Sakabe. An extension of dependency pair method for proving termination of higher-order rewrite systems. *IEICE Transactions on Information and Systems*, E84-D(8):1025–1032, 2001.
11. W. W. Tait. Intensional interpretations of functionals of finite type I. *Journal of Symbolic Logic*, 32(2):198–212, 1967.
12. V. van Oostrom and F. van Raamsdonk. Comparing Combinatory Reduction Systems and Higher-order Rewrite Systems. In *Proc. of HOA'93*, LNCS 816.