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 \( F \) be a set of function symbols, \( X \) be a set of variables and \( R \) be a set of rewrite rules over the set \( \mathcal{T}(F, X) \) of first-order terms. Let \( D \) be the set of symbols occurring at the top of a rule left hand-side and \( C = F \setminus D \). The set \( DP(R) \) of dependency pairs of \( R \) is the set of pairs \((l, t)\) such that \( l \) is the left hand-side of a rule \( l \rightarrow r \in R \) and \( t \) is a subterm of \( r \) headed by some symbol \( f \in D \). The term \( t \) represents a potential recursive call. The chain relation is \( \rightarrow_C := \rightarrow_R^* \cup \rightarrow_{DP} \), where \( \rightarrow_R^* \) is the reflexive and transitive closure of the restriction of \( \rightarrow_R \) to non-top positions and \( \rightarrow_{DP} \) is the restriction of \( \rightarrow_{DP} \) to top positions. Arts and Giesl prove in [1] that \( \rightarrow_R \) is strongly normalizing (SN) (or terminating, well-founded) iff the chain relation so is. Moreover, \( \rightarrow_C \) is terminating if there is a weak reduction ordering \( > \) such that \( R \subseteq \geq \) and \( DP(R) \subseteq > \) (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 \( \lambda \)-terms in \( \beta \)-normal \( \eta \)-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 \( \lambda \)-terms, rules are \( \lambda \)-terms with meta-variables, left hand-sides are patterns à la Miller and matching uses \( \alpha \)-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 \( \beta \)-reductions after having applied the matching substitution. But, in both cases, \( \beta \)-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 \( \beta \)-reduction and add \( \beta \)-reduction at the object level. This is the approach that we consider in this paper.

So, we assume given a set \( R \) of rewrite rules made of simply typed \( \lambda \)-terms and study the termination of \( \rightarrow_\beta \cup \rightarrow_R \) when using CRS-like matching. This
clearly implies the termination of $\rightarrow_R$ in the corresponding CRS or HRS. Another advantage of this approach is that we can rely on Tait’s technique for proving termination [11,13]. 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 $\llbracket T \rrbracket$ of terms of type $T$. Terms of $\llbracket T \rrbracket$ are said computable. Before giving some properties of computable terms, let us introduce a few definitions. The sets $\text{Pos}^+(T)$ and $\text{Pos}^-(T)$ of positive and negative positions in $T$ are defined as follows:

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

We use $T$ to denote a sequence of types $T_1, \ldots, T_n$ of length $|T| = n$. The $i$-th argument of a function symbol $f : T \Rightarrow B$ is accessible if $B$ occurs only positively in $T_i$. Let $\text{Acc}(f)$ be the set of indexes of the accessible arguments of $f$. A base type $B$ is basic if, for all $f : T \Rightarrow B$ and $i \in \text{Acc}(f)$, $T_i$ is a base type. After [34], 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 $\rightarrow_\beta \cup R$.
2. A term of basic type is computable if it is SN wrt $\rightarrow_\beta \cup R$.
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 $ft$ is computable if all its reducts wrt $\rightarrow_\beta \cup R$ are computable.
6. If $ft$ is computable then, for all $i \in \text{Acc}(f)$, $t_i$ is computable.
7. If $t$ contains no $f \in D$ and $\sigma$ is computable, then $t\sigma$ is computable.
8. Every term is computable whenever every $f \in D$ is computable.

### 1.2 Admissible rules

An important property of the first-order case is that, given a term $t$, a substitution $\sigma$ 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 $f l \rightarrow r$ is admissible if $\text{FV}(r) \subseteq \text{PCC}(l)$, 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 $f l \rightarrow r$ is admissible, $\text{dom}(\sigma) \subseteq \text{FV}(l)$ and $l\sigma$ is computable, then $\sigma|_{\text{FV}(r)}$ is computable.

**Proof.** We prove by induction that, for all $u \in \text{PCC}(t)$ and computable substitution $\theta$ such that $\text{dom}(\theta) \subseteq \text{FV}(u) \setminus \text{FV}(t)$, $u\sigma\theta$ is computable.
Fig. 1.1. Pattern Computability Closure [2]

| Case | Rule |
|------|------|
| (arg) | $t_i \in \text{PCC}(t)$ |
| (acc) | $gu \in \text{PCC}(t)$, $i \in \text{Acc}(g)$, $u_i \in \text{PCC}(t)$ |
| (lam) | $\lambda yu \in \text{PCC}(t)$, $y \notin \text{FV}(t)$, $u \in \text{PCC}(t)$ |
| (app-left) | $uy \in \text{PCC}(t)$, $y \notin \text{FV}(t) \cup \text{FV}(u)$, $u \in \text{PCC}(t)$ |
| (app-right) | $y^U \Rightarrow T \Rightarrow U$, $u \in \text{PCC}(t)$, $y \notin \text{FV}(t) \cup \text{FV}(u)$ |

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

(acc) By induction hypothesis, $gu\sigma$ is computable. Thus, by property (0), $u_i\sigma$ is computable.

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

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

(app-right) Let $v = \lambda xU \cdot T x$ and $\theta' = \theta \cup \{y \mapsto v\}$. By (3), $v$ is computable. Since $\text{dom}(\theta) \subseteq \text{FV}(u) \setminus \text{FV}(t)$ and $y \notin \text{FV}(t)$, $\text{dom}(\theta') \subseteq \text{FV}(\lambda yu) \setminus \text{FV}(t)$. Thus, by induction hypothesis, $(yu)\sigma\theta' = vu\sigma\theta'$ is computable. Since $y \notin \text{FV}(u)$, $u\sigma\theta' = u\sigma\theta$. 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 $\text{FAP}(t)$ of full application positions of a term $t$ and the level of a term $t$ are defined as follows:

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

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

- $\text{level}(f t_1 \ldots t_n) = 1 + \max\{\text{level}(t_i) \mid 1 \leq i \leq n\}$
Theorem 5

An higher-order reduction pair is two relations \( (\beta, \beta_C) \) such that:
- \( \beta \) is well-founded and stable by substitution,
- \( \beta_C \) is a reflexive and transitive rewrite relation containing \( \beta \),
- \( \beta \circ (\geq) \subseteq \geq \).

In the conditions of Theorem 4, \( \beta_C \) terminates if \( \mathcal{D} \subseteq \geq \) and \( \mathcal{D} \mathcal{P} \subseteq \geq \).

Proof. By (1), this is so if every term is computable wrt \( \beta_C \). By (3), this is so if every \( f^{T \Rightarrow B} \in \mathcal{D} \) is computable. By (4), this is so if, for all

\[ \text{FAP}\left(f^{1} \ldots \cdot t_{n}\right) = \{\varepsilon\} \cup \bigcup_{i=1}^{n} 1^{n-i} \cdot \text{FAP}(t_{i}) \]

If \( t \neq f^{1} \ldots \cdot t_{n} \) with \( f \in \mathcal{D} \), then \( \text{FAP}(tu) = 1 \cdot \text{FAP}(t) \cup 2 \cdot \text{FAP}(u) \) and \( \text{level}(tu) = \max\{\text{level}(t), \text{level}(u)\} \).

Definition 3 (Dependency pairs)

The set of dependency pairs is \( \mathcal{D} \mathcal{P} = \{l \rightarrow r \mid l \rightarrow r \in \mathcal{D}, p \in \text{FAP}(r)\} \). The chain relation is \( \rightarrow_{\mathcal{C}} = \rightarrow_{\mathcal{R}_{i}} \mathcal{D} \mathcal{P}_{h} \), where \( \rightarrow_{\mathcal{R}_{i}} \) is the restriction of \( \rightarrow_{\mathcal{R}} \) to non-top positions, and \( \mathcal{D} \mathcal{P}_{h} \) is the restriction of \( \rightarrow_{\mathcal{D} \mathcal{P}} \) to top positions.

If, for all \( l \rightarrow r \in \mathcal{D} \mathcal{P}, \text{FV}(r) \subseteq \text{FV}(l) \), we have \( \rightarrow_{\mathcal{C}} \subseteq \rightarrow_{\mathcal{R}} \). Hence, \( \rightarrow_{\beta C} \) is terminating whenever \( \rightarrow_{\mathcal{R}} \) so is. We now prove the converse:

Theorem 4

Assume that, for all \( l \rightarrow r \in \mathcal{D} \) and \( p \in \text{FAP}(r), \text{FV}(r|_{p}) \subseteq \text{FV}(r) \) and \( r|_{p} \) has the type of \( l \) (*). Then, \( \rightarrow_{\beta \mathcal{C}} \) is terminating if \( \rightarrow_{\beta \mathcal{R}} \) so is.

Proof. By (1), this is so if every term is computable wrt \( \rightarrow_{\mathcal{R}} \). By (3), this is so if every \( f^{T \Rightarrow B} \in \mathcal{D} \) is computable. By (4), this is so if, for all \( \lambda \mathcal{N} \mathcal{U}(\mathcal{F} \mathcal{N} \mathcal{x}) \).

The condition on free variables is an important restriction since it is not satisfied by function calls with bound variables like in \( (\lim \mathcal{F}) + x \rightarrow \lim \lambda \mathcal{N} \mathcal{U}(\mathcal{F} \mathcal{N} \mathcal{x}) \).
\( t : T \) computable, \( ft \) is computable. We prove it by induction on \((ft, t)\) with \((\alpha, (\rightarrow_{\beta_{\text{R}}} \alpha)_{\text{lex}})_{\text{lex}}\) as well-founded ordering (H1). Indeed, by (1) and Theorem 4, \( t \) are strongly normalizable wrt \( \rightarrow_{\beta_{\text{R}}} \). By (3), it suffices to prove that every reduct of \( ft \) is computable. If \( t \rightarrow_{\beta_{\text{R}}} t' \) then, by (H1), \( ft' \) is computable since, by (1), \( t' \) are computable and \( (ft') \subseteq (ft) \) since \( \rightarrow_{\beta_{\text{R}}} \subseteq \geq \) and \( \geq \circ \geq \subseteq \geq \). Now, assume that there is \( ft \rightarrow r \in DP \) and \( \sigma \) such that \( t = l \sigma \). Since rules are admissible, by Lemma 2, \( \sigma' = \sigma|_{\text{FV}(r)} \) is computable. Since \( DP \subseteq \geq \) and \( \geq \) is stable by substitution, \( ft > r \sigma' \). Thus, by (H1), \( r \sigma' \) is computable. \end{proof}

An example of reduction pair can be given by using the higher-order recursive path ordering \( >_{\text{horpo}} \). Take \( > = (\rightarrow_{\beta} \cup >_{\text{horpo}})^+ \) and \( \geq = (\rightarrow_{\beta} \cup >_{\text{horpo}})^* \). The study of these two relations has to be done. However, \( >_{\text{horpo}} \) 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 \( >_{\text{whorco}} \), whose monotonic closure strictly contains \( >_{\text{horpo}} \). Moreover, \( >_{\text{whorco}} \) is transitive, which is not the case of \( >_{\text{horpo}} \). It would therefore be interesting to look for reduction pairs built from \( >_{\text{whorco}} \).

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.