Argument Filterings and Usable Rules
in Higher-Order Rewrite Systems

SUZUKI Sho†, KUSAKARI Keiichirou†,
Frédéric Blanqui‡

† Graduate School of Information Science, Nagoya University
‡ INRIA, France

The static dependency pair method is a method for proving the termination of higher-order rewrite systems à la Nipkow. It combines the dependency pair method introduced for first-order rewrite systems with the notion of strong computability introduced for typed λ-calculi. Argument filterings and usable rules are two important methods of the dependency pair framework used by current state-of-the-art first-order automated termination provers. In this presentation, we extend the class of higher-order systems on which the static dependency pair method can be applied. Then, we extend argument filterings and usable rules to higher-order rewriting, hence providing the basis for a powerful automated termination prover for higher-order rewrite systems.

1 Introduction

Various extensions of term rewriting systems (TRSs) [29] for handling functional variables and abstractions have been proposed [13, 22, 11, 23, 15]. In this paper, we consider higher-order rewrite systems (HRSs) [22], that is, rewriting on variables and abstractions have been proposed [13, 22, 11, 23, 15]. In this paper, we provide a new definition of PFP that significantly enlarges the class of systems on which the method can be applied. It is based on the notion of accessibility introduced in [3] and extended to HRSs in [2].

For the HRS $R_{\text{sum}} \cup R_{\text{len}}$, the static dependency pair method returns the following two components:

$$\begin{cases} 
\text{foldl}(\lambda xy.F(x, y), X, \text{cons}(Y, L)) \rightarrow \text{foldl}(\lambda xy.F(x, y), F(X, Y), L) \\
\text{add}(s(X), Y) \rightarrow \text{add}(s(add(X, Y)), 0, L)
\end{cases}$$

The static dependency pair method proves the termination of the HRS $R_{\text{sum}} \cup R_{\text{len}}$ by showing the non-loopingness of each component.

In order to show the non-loopingness of a component, the notion of reduction pair is often used. Roughly speaking, it consists in finding a well-founded quasi-ordering in which the component rules are strictly decreasing and all the original rules are non-increasing.

Argument filterings, which consist in removing some arguments of some functions, provide a way to generate reduction pairs. First introduced for TRSs [1], it has been extended to STRSs [15, 18]. In this paper, we extend it to HRSs.

In order to reduce the number of constraints required for showing the non-loopingness of a component, the notion of usable rules is also very important.
Indeed, a finer analysis of sequences of function calls show that not all original rules need to be taken into account when trying to prove the termination of a component. This analysis was first conducted for TRSs [7, 10] and has been extended to STRSs [27, 18]. In this paper, we extend it to HRSs.

All together, this paper provides a strong theoretical basis for the development of an automatic termination prover for HRSs, by extending to HRSs some successful techniques used by modern state-of-the-art first-order termination provers like for instance [8, 10].

The remainder of this paper is organized as follows. Section 2 introduces HRSs. Section 3 presents the static dependency pair method and extend the class of systems on which it can be applied. In Section 4, we extend the argument filtering method to HRSs. In Section 5, we extend the notion of usable rules on HRSs. Concluding remarks are given in Section 6.

2 Preliminaries

In this section, we introduce the basic notions for HRSs according to [22, 21].

The set \( S \) of simple types is generated from the set \( \mathcal{B} \) of basic types by the type constructor \( \rightarrow \). A functional or higher-order type is a simple type of the form \( \alpha \rightarrow \beta \). We denote by \( \triangleright_s \) the strict subterm relation on types.

A preterm is generated from an infinite set of typed variables \( \mathcal{V} \) and a set of typed function symbols \( \mathcal{Y} \) by \( \lambda \)-abstraction and \( \lambda \)-application. The set of typed preterms is denoted with \( \mathcal{T}^{pre} \).

A set \( \mathcal{T} \) of simply-typed terms is defined as \( \{ t \mid t \in \mathcal{T}^{pre} \} \). The unique type of a term \( t \) is denoted by \( \text{type}(t) \). We write \( \mathcal{V}_n \) (resp. \( \mathcal{T}_n \)) as the set of variables (resp. terms) of type \( \alpha \). The equivalence of terms is denoted by \( \equiv \). The set of free variables in a term \( t \) is denoted by \( \mathcal{FV}(t) \). We assume for convenience that bound variables in a term are all different, and are disjoint from free variables. In general, a term \( t \) is of the form \( \lambda x_1 \ldots x_m \cdot a_1 \ldots a_n \) where \( a \in \mathcal{S} \cup \mathcal{V} \). We abbreviate this by \( \lambda a_{m,n} \cdot a(t_n) \).

For a term \( t \equiv \lambda a_{m,n} \cdot a(t_n) \), the symbol \( a \), denoted by \( \text{top}(t) \), is the top symbol of \( t \), and the set \( \{ t_n \} \), denoted by \( \text{args}(t) \), is the arguments of \( t \). We define the set \( \text{Sub}(t) \) of subterms of \( t \) by \( \{ t \} \cup \text{Sub}(s) \) if \( t \equiv \lambda x \cdot s \), and \( \{ t \} \cup \bigcup_{i=1}^{n} \text{Sub}(t_i) \) if \( t \equiv a(t_n) \). We use \( t \triangleright_{\text{sub}} s \) to represent \( s \in \text{Sub}(t) \), and define \( t \triangleright_{\text{sub}} s \) by \( t \triangleright_{\text{sub}} s \) and \( t \not\equiv s \). The set \( \text{Pos}(t) \) of positions in \( t \) is the set of strings over positive integers inductively defined as \( \text{Pos}(\lambda x \cdot t) = \{ \varepsilon \} \cup \{ | p \mid p \in \text{Pos}(t) \} \) and \( \text{Pos}(a(t_n)) = \{ \varepsilon \} \cup \bigcup_{i=1}^{n} \{ i p \mid p \in \text{Pos}(t_i) \} \). The prefix order \( \prec \) on positions is defined by \( p \prec q \iff p w = q \) for some \( w \neq \varepsilon \). The subterm of \( t \) at position \( p \) is denoted by \( t_p \).

A term containing a unique occurrence of the special constant \( \square \) of type \( \alpha \) is called a context, denoted by \( C[\square] \). We use \( C[t] \) for the term obtained from \( C[\square] \) by replacing \( \square \) with \( t \). A substitution \( \theta \) is a mapping from variables to terms such that \( \theta(X) \) has the type of \( X \) for each variable \( X \). We define \( \text{dom}(\theta) = \{ X \mid X \not\equiv \theta(X) \} \) and assume that \( \text{dom}(\theta) \) is always finite. A substitution \( \theta \) is naturally extended to a mapping from terms to terms. We use \( \theta(t) \) instead of \( \theta(t) \) in the remainder of the paper. A substitution \( \theta \) is said to be a variable permutation if \( \forall X \in \text{dom}(\theta), \exists Y \in \text{dom}(\theta), \theta(X) \equiv Y \downarrow \) and \( \theta(X) \equiv \theta(Y) \Rightarrow X = Y \) hold.

Following [21], a higher-order rewrite rule is a pair \( (l, r) \) of terms, denoted by \( l \rightarrow r \), such that \( \text{top}(l) \in \mathcal{S} \), \( \text{type}(l) = \text{type}(r) \in \mathcal{B} \) and \( \text{FV}(l) \supseteq \text{FV}(r) \). Since, by definition, terms are in \( \eta \)-long form, function symbols are always applied to the same (maximal) number of arguments. Considering non-\( \eta \)-normal terms or rules of functional type is outside the scope of this paper. An HRS is a set of higher-order rewrite rules. The reduction relation \( \rightarrow \) of an HRS \( R \) is defined by \( s \rightarrow t \) iff \( s \equiv C[\theta][t] \) and \( t \equiv C[\eta][\theta][t] \) for some rewrite rule \( l \rightarrow r \in R \), context \( C[\theta] \) and substitution \( \theta \). The transitive and reflexive-transitive closures of \( \rightarrow \) are denoted by \( \rightarrow^* \) and \( \rightarrow^+= \), respectively. An HRS \( R \) is said to be finitely branching if \( \{ t' \mid t \rightarrow^+= t' \} \) is a finite set for any term \( t \).

A term \( t \) is said to be terminating or strongly normalizing for an HRS \( R \), denoted by \( SN(R, t) \), if there is no infinite rewrite sequence of \( R \) starting from \( t \). We write \( SN(R) \) if \( SN(R, t) \) holds for any term \( t \). A well-founded relation \( \succ \) on terms is a reduction order if \( \succ \) is closed under substitution and context. We notice that an HRS \( R \) is terminating iff \( R \nsubseteq \succ \) for some reduction order \( \succ \).

A term \( t \) is said to be strongly computable in an HRS \( R \) if \( SC(R, t) \) holds, which is inductively defined on simple types as follows: \( SN(R, t) \) if \( \text{type}(t) \in \mathcal{B} \), and \( \forall \theta \in \mathcal{T}_n, (SC(R, u) \vDash SC(R, (tu))) \) if \( \text{type}(t) = \alpha \rightarrow \beta \). We also define the set \( \mathcal{T}^{\text{arg}^*}(R) = \{ t \mid \forall \theta \in \text{args}(t). SC(R, u) \} \).

Finally, we introduce the proposition required for later proof.

Proposition 2.1 [21] If \( s \rightarrow^+= \alpha \cdot t \) and \( \theta \rightarrow^+= \theta' \) (i.e. \( \forall x \in \mathcal{V}. x \theta \rightarrow^+= x \theta' \)) then \( s \theta \rightarrow^+= \alpha \cdot t \theta' \).

3 Improved Static Dependency Pair Method

In this section, we introduce the static dependency pair method for plain function-passing (PFP) HRSs [19] but extend the class of PFP systems.
The method in [19] applies only to PFP systems. From a technical viewpoint, we have noticed that the unclosedness of strong computability with respect to the subterm relation is the reason why the method is not applicable to every HRS. Hence we can extend the applicable class for the method if more strongly computable subterms can be acquired. From the same motivation, Blanqui introduced the notion of accessibility to design a higher-order path ordering [2]. By using the notion of accessibility, we provide a new definition of PFP that enlarges the class of systems on which the method can be applied.

**Definition 3.1 (Stable subterms)** The stable subterms of $t$ are $\text{SSub}(t) = \text{SSub}_{FV(t)}(t)$ where $\text{SSub}_X(t) = \{l \mid \text{SSub}(l) = \text{SSub}_X(l), \text{SSub}_{X}(\lambda x.s) = \text{SSub}_X(s), \text{SSub}_X(a(l_0)) = \bigcup_{i=1}^{n} \text{SSub}_X(t_i)$ if $a \notin X$, and $\text{SSub}_X(t) = \emptyset$ otherwise.

**Lemma 3.2** (1) $\text{SSub}(t) \subseteq \text{Sub}(t)$. (2) If $u \in \text{SSub}(t)$ and $\text{dom}(\theta) \subseteq \text{FV}(t)$, then $u\theta \in \text{SSub}(t\theta)$.

**Definition 3.3 (Safe subterms - New definition)** The set of safe subterms of a term $l$ is $\text{safe}(l) = \bigcup_{i \in \text{args}(l)} \{t \mid t \in \text{Acc}(l'), \text{FV}(t) \subseteq \text{FV}(l')\}$ where $t \in \text{Acc}(l')$ (t is accessible in $l'$) if either:

1. $t = l'$,
2. $\lambda x.t \in \text{Acc}(l')$ and $x \notin \text{FV}(l')$,
3. $t(x) \in \text{Acc}(l')$ and $x \notin \text{FV}(t) \cup \text{FV}(l')$,
4. $l(l_0) \in \text{Acc}(l')$, $t_i = \lambda x.t$, $\text{type}(t) \in \text{B}$ and $\{x\} \cap \text{FV}(t) = \emptyset$,
5. $l(l_0) \in \text{Acc}(t)$, $t_i = t$ and $x \notin \text{FV}(l)$.

Strictly speaking, $\text{safe}(l)$ may not be included in $\text{Sub}(l)$ and, because of (3), accessible terms are $\beta$-normal preterms not necessarily in $\eta$-normal form.

**Definition 3.4 (Plain Function-Passing [19])** An HRS $R$ is plain function-passing (PFP) if for any $l \rightarrow r \in R$ and $Z(l_0) \in \text{Sub}(r)$ such that $Z \in \text{FV}(r)$, there exists $k \leq n$ such that $Z(l_0) \in \text{safe}(l)$.

For example, the HRS $R_{\text{foldl}}$ displayed in the introduction is PFP, because $\text{safe}(\text{foldl}(\lambda xy.F(x,y), X, \text{cons}(Y,L))) \equiv \{\lambda xy.F(x,y), X, \text{cons}(Y,L)\}$ and $F' \equiv \lambda xy.F(x,y) \in \text{safe}(\text{foldl}(\lambda xy.F(x,y), X, \text{cons}(Y,L)))$.

The definition of safeness given in [19] corresponds to cases ((0)) and ((1)). This new definition therefore includes much more terms, mainly higher-order patterns [20]. This greatly increases the class of rules that can be handled and the applicability of the method since it reduces the number of dependency pairs.

For instance, the new definition allows us to handle the following rule:

$$D(\lambda x.\sin(Fx)y) \rightarrow D(\lambda x.Fx)y \times \cos(Fy)$$

Indeed, $l' = \lambda x.\sin(Fx) \in \text{Acc}(l')$ by ((0)), $\sin(Fx) \in \text{Acc}(l')$ by ((2)), $F \in \text{Acc}(l')$ by ((4)) and $F' \in \text{Acc}(l')$ by ((3)). Therefore, $\text{safe}(l) = \{l', \lambda x.Fx, y\}$.

With the previous definition, we had $\text{safe}(l) = \{l', y\}$ only.

Also, the new definition allows us to handle the following rule:

$$\forall(\lambda x.(Px \land Qx)) \rightarrow \forall(\lambda x.Px) \land \forall(\lambda x.Qx)$$

Indeed, $l' = \lambda x.(Px \land Qx) \in \text{Acc}(l')$ by ((0)), $Px \land Qx \in \text{Acc}(l')$ by ((2)), $P, Q \in \text{Acc}(l')$ by ((4)), and $P, Q \in \text{Acc}(l')$ by ((3)). Therefore, $\text{safe}(l) = \{l', \lambda x.Px, \lambda x.Qx\}$. With the previous definition, we had $\text{safe}(l) = \{l'\}$ only.

For the results presented in [19] to still hold, it suffices to check that this new definition of safeness still preserves strong computability (Lemma 4.3 in [19]). This can be shown by following the proof of Lemma 10 in [2].

**Lemma 3.5** Let $R$ be an HRS and $l \rightarrow r \in R$. Then $l_0 \downarrow \in T_{\text{SC}}(R)$ implies $SC(R, \theta_1)$ for any $t \in \text{safe}(l)$ and substitution $\theta$.

**Proof.** We first prove that $t_0 \downarrow$ is strongly computable whenever $t \in \text{Acc}(l')$, $l' \downarrow$ is strongly computable, and $xt$ is strongly computable for any $x \notin \text{FV}(t) \setminus \text{FV}(l')$.

Wlog we can assume that $\text{dom}(\theta) \subseteq \text{FV}(t)$. We prove the claim by induction on the definition of Acc.

0. Immediate.

1. Since $l' \downarrow$ is strongly computable, $l' \downarrow$ is strongly normalizing. By Lemma 3.2, $\theta_1 \in \text{Sub}(l' \downarrow)$ and $\theta_1 \downarrow$ is SN. Therefore, since $\text{type}(t) \in \text{B}$, $\theta_1 \downarrow$ is strongly computable.

2. By definition of computability.
(3) We have type(t) = α → β. So, let u ∈ T_n strongly computable and θ = θ ∪ {x → u} (x ∉ dom(θ) since x ∉ FV(t)). Since x ∉ FV(t), we have (θ ∪ {x → u})θ = (t(x))θ. By IH, (t(x))θ is strongly computable. Therefore, θθ is strongly computable.

(4) Since strong computability on base types is equivalent to SN and F V(FV(t)) = ∅.

(5) The term p_i = λy_i.y_i can easily be proved strongly computable. Then, let θ = θ ∪ {x → p_i} (x ∉ dom(θ) since x ∉ FV(FV(t))). Since x ∉ FV(t), we have (p_i)θ = p_iθ. By induction hypothesis, (p_i)θ is strongly computable. Therefore, θθ is strongly computable.

Let now u ∈ safe(t). We have u = t_l for some t ∈ Acc(l') and l' ∈ args(l) with FV(t) ⊆ FV(l'). The term l'θ is strongly computable since θθ ∈ T_{SC}^SC(R). Since FV(t) ⊆ FV(l'), there is no x ∈ FV(t) \ FV(l'). Therefore, uθ is strongly computable.

This definition of safeness can be further improved (in case (4)) by using more complex interpretations for base types than just the set of strongly normalizing terms, but this requires to check more properties[5]. We leave this for future work.

We now recall the definitions of static dependency pair, static recursion component and reduction pair, and the basic theorems concerning these notions, including the subterm criterion [19].

Definition 3.6 (Static dependency pair [19]) Let R be an HRS. All top symbols of the left-hand sides of rewrite rules, denoted by D_R, are called defined symbols.

We define the marked term lθ by fθ(T_n) if t has the form f(T_n) with f ∈ D_R; otherwise lθ = l. Then, let D_R^2 = {fθ | f ∈ D_R}.

We also define the set of candidate subterms as follows: Cand(λx_m.a(T_n)) = {λx_m.a(T_n)} ∪ \sum_{i=1}^{m} Cand(λx_m.t_i).

Now, a pair {l, a(T_n)}), denoted by lθ → aθ(T_n), is said to be a static dependency pair in R if there exists l → r ∈ R such that λx_m.a(T_n) ∈ Cand(r), a ∈ D_R, and a(T_n) ∉ safe(l) for all k ≤ n. We denote by SDP(R) the set of static dependency pairs in R.

Example 3.7 Let R_{ave} be the following PFP-HRS:

\[
\begin{align*}
\text{sub}(X,0) & \rightarrow X \\
\text{sub}(0,Y) & \rightarrow 0 \\
\text{sub}(s(X),s(Y)) & \rightarrow \text{sub}(X,Y) \\
\text{div}(0,s(Y)) & \rightarrow 0 \\
\text{div}(s(X),s(Y)) & \rightarrow s(\text{div}(\text{sub}(X,Y),s(Y))) \\
\text{ave}(L) & \rightarrow \text{div}(\text{sum}(L),\text{len}(L))
\end{align*}
\]

Then, the set SDP(R_{ave}) consists of the following eleven pairs:

\[
\begin{align*}
\text{fold}^2(λxy.F(x,y), X, \text{cons}(Y,L)) & \rightarrow \text{fold}^2(λxy.F(x,y), F(X,Y), L) \\
\text{add}^2(s(x),Y) & \rightarrow \text{add}^2(x,Y) \\
\text{sum}^2(L) & \rightarrow \text{fold}^2(λxy.\text{add}(x,y),0,L) \\
\text{sub}^2(s(x),s(Y)) & \rightarrow \text{sub}^2(x,Y) \\
\text{div}^2(s(X),s(Y)) & \rightarrow \text{div}^2(\text{sub}(X,Y),s(Y)) \\
\text{div}^2(s(x),s(Y)) & \rightarrow \text{sub}^2(x,Y) \\
\text{len}^2(L) & \rightarrow \text{fold}^2(λxy.s(x),0,L) \\
\text{ave}^2(L) & \rightarrow \text{div}^2(\text{sum}(L),\text{len}(L)) \\
\text{ave}^2(L) & \rightarrow \text{sum}(L) \\
\text{ave}^2(L) & \rightarrow \text{len}(L)
\end{align*}
\]

Definition 3.8 (Static dependency chain [19]) Let R be an HRS. A sequence u_0 → v_0, u_1 → v_1, ..., of static dependency pairs is a static dependency chain in R if there exist θ_0, θ_1, ..., such that v_iθ_i → u_{i+1}θ_{i+1} and u_iθ_i, v_iθ_i ∈ T_{SC}^SC(R) for all i.

Note that, for all i, u_iθ_i and v_iθ_i are terminating, since strong computability implies termination.

Proposition 3.9 [19] Let R be a PFP-HRS. If there exists no infinite static dependency chain then R is terminating.

Proof. By using Lemma 3.5 instead of Lemma 4.3 in [19], the proof of the correspondence theorem (Theorem 5.23 in [19]) still holds.

Definition 3.10 (Static recursion component [19]) Let R be an HRS. The static dependency graph of R is the directed graph in which nodes are SDP(R)
and there exists an arc from \( u^x \rightarrow v^x \) to \( u^y \rightarrow v^y \) if the sequence \( u^x \rightarrow v^x \), \( u^y \rightarrow v^y \) is a static dependency chain.

A static recursion component is a set of nodes in a strongly connected subgraph of the static dependency graph of \( R \). We denote by \( SRC(R) \) the set of static recursion components of \( R \).

A static recursion component \( C \) is non-looping if there exists no infinite static dependency chain in which only pairs in \( C \) occur and every \( u^x \rightarrow v^y \in C \) occurs infinitely many times.

**Proposition 3.11** [19] Let \( R \) be a PFP-HRS such that there exists no infinite path in the static dependency graph. If all static recursion components are non-looping, then \( R \) is terminating.

**Example 3.12** For the PFP-HRS \( R_{ave} \) in Example 3.7, the static dependency graph of \( R_{ave} \) is shown in Fig. 1. Then the set \( SRC(R_{ave}) \) consists of the following four static recursion components:

\[
\begin{align*}
\text{foldl}^2(\lambda xy.F(x, y), X, \text{cons}(Y, L)) & \rightarrow \text{foldl}^2(\lambda xy.F(x, y), F(X, Y), L) \\
\text{add}^2(s(X), Y) & \rightarrow \text{add}^2(X, Y) \\
\text{sub}^2(s(X), s(Y)) & \rightarrow \text{sub}^2(X, Y) \\
\text{div}^2(s(X), s(Y)) & \rightarrow \text{div}^2(\text{sub}(X, Y), s(Y))
\end{align*}
\]

In order to prove the non-loopingness of components, the notions of subterm criterion and reduction pair have been proposed. The subterm criterion was introduced on TRSs [10], and then extended to STRSs [17] and HRSs [19]. Reduction pairs [16] are an abstraction of the notion of weak-reduction order [1].

**Definition 3.13 (Subterm criterion [19])** Let \( R \) be an HRS and \( C \in SRC(R) \). We say that \( C \) satisfies the subterm criterion if there exists a function \( \pi \) from \( D^+_R \) to non-empty sequences of positive integers such that:

- \( u|\pi(\text{top}(u))] >_{\text{sub}} v|\pi(\text{top}(v))] \) for some \( u^x \rightarrow v^y \in C \),
- and the following conditions hold for every \( u^x \rightarrow v^y \in C \):
  - \( u|\pi(\text{top}(u))] \geq_{\text{sub}} v|\pi(\text{top}(v))] \),
  - \( \forall p < \pi(\text{top}(u^x)).\text{top}(u^y) \notin FV(u) \),
  - and \( \forall q < \pi(\text{top}(v^y)).q = \varepsilon \lor \text{top}(v^q) \notin FV(v) \cup D_R \).

**Definition 3.14 (Reduction pair, Weak reduction order [1, 16])** A pair \((\succeq, >)\) of relations is a reduction pair if \( \succeq \) and \( > \) satisfy the following properties:

- \( > \) is well-founded and closed under substitutions,
- \( \succeq \) is closed under contexts and substitutions,
- and \( \succeq \cdot > \subseteq > \lor > \cdot \succeq \subseteq > \).

In particular, \( \succeq \) is a weak reduction order if \( \succeq \cap > \neq \emptyset \).

**Proposition 3.15** [19] Let \( R \) be a PFP-HRS such that there exists no infinite path in the static dependency graph. Then, \( C \in SRC(R) \) is non-looping if \( C \) satisfies one of the following properties:

- \( C \) satisfies the subterm criterion.
- There is a reduction pair \((\succeq, >)\) such that \( R \subseteq \succeq, C \subseteq \succeq \cup > \) and \( C \cap > \neq \emptyset \).

**Example 3.16** Let \( \pi(\text{foldl}^2) = 3 \) and \( \pi(\text{add}^2) = \pi(\text{sub}^2) = 1 \). Then, every static recursion component \( C \) except the one for \( \text{div} \) (cf. Example 3.12) satisfies the
subterm criterion in the underlined positions below. Hence, these static recursion components are non-looping:

\[
\begin{align*}
\text{fold}^2(\lambda x y. F(x, y), X, \text{cons}(Y, L)) &\rightarrow \text{fold}^2(\lambda x y. F(x, y), F(X, Y), L) \\
\text{add}^2(s(X), Y) &\rightarrow \text{add}^2(X, Y) \\
\text{sub}^2(s(X), s(Y)) &\rightarrow \text{sub}^2(X, Y)
\end{align*}
\]

4 Argument Filterings

An argument filtering generates a weak reduction order from an arbitrary reduction order. The method was first proposed on TRSs [1], and then extended to STRSs [15, 18]. Since this extension has the problem that this method may destroy the well-typedness of terms, Kusakari and Sakai improved the method so that the well-typedness is never destroyed [18]. In this section, we expand this technique to HRSs.

**Definition 4.1** An argument filtering function is a function \( \pi \) such that, for every \( f \in \Sigma \) of type \( \alpha_1 \rightarrow \cdots \rightarrow \alpha_n \rightarrow \beta \) with \( \beta \in \mathcal{B} \), \( \pi(f) \) is either a positive integer \( i \leq n \) if \( \alpha_i = \beta \), or a list of positive integers \( [i_1, \ldots, i_k] \) with \( i_1, \ldots, i_k \leq n \). Then, we extend the function \( \pi \) to terms by taking:

\[
\pi(\lambda x_m. a(t_n)) \equiv \begin{cases} 
\lambda x_m. \pi(t_i) & \text{if } a \in \Sigma \text{ and } \alpha(a) = i \\
\lambda x_m. \pi(\{t_i\}) & \text{if } a \in \Sigma \text{ and } \alpha(a) = [i_1, \ldots, i_k] \\
\lambda x_m. \pi(t_i) & \text{if } a \in \mathcal{V}
\end{cases}
\]

Given an argument filtering \( \pi \) and a binary relation \( > \), we define \( s \geq_\pi t \) by \( \pi(s) > \pi(t) \) or \( \pi(s) \equiv \pi(t) \) and \( s \geq_\pi t \) by \( \pi(s) > \pi(t) \). We also define the substitution \( \theta_\pi \) by \( \theta_\pi(x) = \pi(\theta(x)) \). Finally, we define the typing function \( \text{type}_\pi \) after argument filtering as \( \text{type}_\pi(a) = \alpha_i \rightarrow \cdots \rightarrow \alpha_k \rightarrow \beta \) if \( a \in \Sigma \), \( \text{type}(a) = [i_1, \ldots, i_k] \), \( \text{type}(a) = \alpha_1 \rightarrow \cdots \alpha_n \rightarrow \beta \) and \( \beta \in \mathcal{B} \); otherwise \( \text{type}_\pi(a) = \text{type}(a) \).

In the examples, except stated otherwise, \( \pi(f) = [1, \ldots, n] \) if \( \text{type}(f) = \alpha_1 \rightarrow \cdots \alpha_n \rightarrow \beta \) and \( \beta \in \mathcal{B} \) (no argument is removed).

For instance, if \( \pi(\text{sub}) = [1] \) then \( \pi(\text{div}^2(\text{sub}(X, Y), s(Y))) \equiv \text{div}^2(\text{sub}(X), s(Y)) \).

Note that our argument filtering method never destroys the well-typedness, which is easily proved by induction on terms.

**Theorem 4.2** For any argument filtering \( \pi \) and term \( t \in \mathcal{T} \), \( \pi(t) \) is well-typed under the typing function \( \text{type}_\pi \) and \( \text{type}_\pi(\pi(t)) = \text{type}(t) \).

In the following, we prove the soundness of the argument filtering method as a generating method of weak reduction orders. To this end, we first prove a lemma required for showing that \( > \pi \) and \( \geq_\pi \) are closed under substitution.

**Lemma 4.3** \( \pi(\theta \pi) \equiv \pi(\theta) \theta_\pi \).

Proof. We proceed by induction on preterm \( t \) ordered with \( \pi(\theta \pi) \cup \theta_\pi \).

- In case of \( t = \lambda x.u \): Since \( \theta \theta_\pi u \), we have \( \pi(u) \theta_\pi \equiv \pi(u) \theta_\pi \) from the induction hypothesis. Hence we have: \( \pi((\lambda x.u)\theta) \equiv \lambda x.(\pi(u)\theta) \).

- In case of \( t = f(t_1, \ldots, t_n) \), \( f \in \Sigma \), and \( \pi(f) = i \): Since \( \theta \theta_\pi t_i \), we have \( \pi(t_i) \theta_\pi \equiv \pi(t_i) \theta_\pi \) from the induction hypothesis. Hence we have: \( \pi(f(t_1, \ldots, t_n)) \equiv \pi(f(t_1, \ldots, t_n)) \).

- In case of \( t = f(f(t_n)) \), \( f \in \Sigma \), and \( \pi(f) = i \): Suppose that \( t'_i \equiv \perp \) if \( i \notin \pi(f) \); otherwise \( t'_i \equiv \pi(t_i) \), and \( t'_i \equiv \perp \) if \( i \notin \pi(f) \); otherwise \( t'_i \equiv \pi(t_i) \). For each \( i \), since \( \theta \theta_\pi t_i \), we have \( \pi(t_i) \theta_\pi \equiv \pi(t_i) \theta_\pi \) from the induction hypothesis. Then \( t'_i \equiv t'_i \theta_\pi \) holds for each \( i \). Hence we have: \( \pi(f(f(t_n))) \equiv f(f(t_n)) \).

- In case of \( t = X \in \mathcal{V} \): Obvious from the definition of \( \theta_\pi \).

- In case of \( t = X(t_n) \), \( X \in \mathcal{V} \) and \( n > 0 \): Since \( \text{type}(X) = \text{type}(X) \), we have \( X \theta \equiv \lambda x_m. a(\pi(\theta)) \). For each \( i \), since \( \theta \theta_\pi t_i \), we have \( \pi(t_i) \theta_\pi \equiv \pi(t_i) \theta_\pi \) from the induction hypothesis. Since \( \theta \theta \equiv \lambda x_m. a(\pi(\theta)) \), \( a(\pi(\theta)) \{ y_i := t_i | i \in \pi(\theta) \} \) holds for each \( i \). Hence we have: \( \pi(X(t_n)\theta_\pi) \equiv \pi((\lambda x_m. a(\pi(\theta)))(t_n)) \).

Note that the corresponding lemma in STRSs is \( \pi(\theta \pi) \geq \pi(\theta) \theta_\pi \) where \( \geq \) is a given binary relation [18]. This is the technical reason why the argument filtering method on STRSs can apply to only left-firmness (left-hand side variables occurs at leaf positions only) STRSs [15, 18]. This difference originates the fact that STRSs allow partial application (ex. \( \text{fold} \), \( \text{fold} \) \( F \) \( X) \) but HRSs does not.
Theorem 4.4 For any reduction order $>\pi$ and argument filtering function $\pi$, $\geq_{\pi}$ is a weak reduction order.

**Proof.** It is easily shown that $s \geq_{\pi} t \Rightarrow C[s] \geq_{\pi} C[t]$ by induction on $C[\cdot]$. From Lemma 4.3, we have $s \geq_{\pi} t \Rightarrow \pi(s) \geq \pi(t)$, $\pi(s)\theta_{\downarrow} \geq \pi(t)\theta_{\downarrow} \Rightarrow \pi(s\theta_{\downarrow}) \geq \pi(t\theta_{\downarrow})$, and $s > \pi t \Rightarrow \pi(s) > \pi(t) \Rightarrow \pi(s)\theta_{\downarrow} > \pi(t)\theta_{\downarrow} \Rightarrow \pi(s\theta_{\downarrow}) > \pi(t\theta_{\downarrow}) \Rightarrow s\theta_{\downarrow} >_{\pi} t\theta_{\downarrow}$. Remaining properties are routine. □

Example 4.5 Consider the PFP-HRS $R_{\text{ave}}$ in Example 3.7. Every static recursion component except $\{\text{div}(s(X), s(Y)) \rightarrow \text{div}(s(Y), s(Y))\}$ is non-looping (cf. Example 3.16). We can prove its non-loopingness with the argument filtering method by taking $\pi(\text{sub}) = \pi(\text{div}) = [1]$, and the normal higher-order reduction ordering $\succ_{\text{rhorpo}}$, written $(\succ_{\text{rhorpo}})_n$ in [12] defined by:

- a neutralization level $\mathcal{L}_j = 0$ for all symbol $f \in \Sigma$ and argument position $j$ (in fact, these parameters are relevant for functional arguments only),
- filtering out all arguments (a notion introduced in [12] not to be confused with the argument filtering method) by taking $\mathcal{A}_j = \emptyset$ for all $f$ and $j$ (again, these parameters are relevant for functional arguments only),
- a precedence $s_{\text{new}} \succ_{\text{new}, \text{subnew}}$ (a symbol $f_{\text{new}}$ with $f \in \Sigma$ is a new symbol introduced by the definition of $\succ_{\text{rhorpo}}$ in [12], with the same type as $f$ since neutralization levels are null),
- a multiset (or lexicographic) status for $\text{div}^2_{\text{new}}$,
- a quasi-ordering on types reduced to the equality (the strict part is well-founded since it is empty, and equality preserves functional types).

Then we have $\pi(\text{div}^2(s(X), s(Y))) \equiv \text{div}^2(s(X)) \succ_{\text{rhorpo}} \text{div}^2(s(Y)) \equiv \pi(\text{div}^2(s(X), s(Y)))$, and $R_{\text{div}} \subseteq (\succ_{\text{rhorpo}})_\pi$. For instance, $\text{div}^2(s(X)) \succ_{\text{rhorpo}} \text{div}^2(\text{sub}(X))$ since $F N(\text{div}^2(s(X)))_{\beta} \succ_{\text{rhorpo}} F N(\text{div}^2(\text{sub}(X)))_{\beta}$ and, because $\mathcal{L}_j = 0$ and $\mathcal{A}_j = \emptyset$, $F N(f_{\text{new}} s_{\text{new}} f_{\text{new}} t_{\text{new}}) = f_{\text{new}} F N(f t_{1} \ldots t_{n}) = F N(t_{1}) \ldots F N(t_{n})$. From Proposition 3.15, the static recursion component for div is non-looping, and $R_{\text{div}}$ is terminating.

5 Usable Rules

In order to reduce the number of constraints required for showing the non-loopingness of a component, the notion of usable rules is widely used. This notion was introduced on TRSs [7, 10] and then extended to STRSs [27, 18]. In this section, we extend it to HRSs.

To illustrate the interest of this notion, we start with some example.

**Example 5.1** We consider the data type $\text{heap} ::= \text{leaf} | \text{node}(\text{nat}, \text{heap}, \text{heap})$ and the PFP-HRS $R_{\text{heap}}$ defined by the following rules:

- $\text{add}(0, Y) \rightarrow Y$
- $\text{add}(s(X), Y) \rightarrow s(\text{add}(X, Y))$
- $\text{map}(\lambda x. F(x), \text{nil}) \rightarrow \text{nil}$
- $\text{map}(\lambda x. F(x), \text{cons}(X, L)) \rightarrow \text{cons}(F(X), \text{map}(\lambda x. F(x). L))$
- $\text{merge}(H, \text{leaf}) \rightarrow H$
- $\text{merge}(\text{leaf}, H) \rightarrow H$
- $\text{merge}(\text{node}(X_1, H_{11}, H_{12}), \text{node}(X_2, H_{21}, H_{22}))$
  $\rightarrow \text{node}(X_1, H_{11}, \text{merge}(H_{12}, \text{node}(X_2, H_{21}, H_{22})))$
- $\text{merge}(\text{node}(X_1, H_{11}, H_{12}), \text{node}(X_2, H_{21}, H_{22}))$
  $\rightarrow \text{node}(X_2, \text{merge}(\text{node}(X_1, H_{11}, H_{12}), H_{21}), H_{22})$

$\text{foldT}(\lambda xyz. F(x, y, z), X, \text{leaf}) \rightarrow X$

$\text{foldT}(\lambda xyz. F(x, y, z), X, \text{node}(Y, H_{11}, H_{12}))$
  $\rightarrow F(X, \text{foldT}(\lambda xyz. F(x, y, z), X, H_{11}), \text{foldT}(\lambda xyz. F(x, y, z), X, H_{12}))$
  $\rightarrow \text{sumT}(H)$
  $\rightarrow \text{foldT}(\lambda xyz. \text{add}(x, \text{add}(y, z)), 0, H)$
- $\text{hd}(\text{nil}) \rightarrow \text{leaf}$
- $\text{hd}(\text{cons}(X, L)) \rightarrow X$
- $\text{l2t}(\text{nil}) \rightarrow \text{nil}$
- $\text{l2t}(\text{cons}(H, \text{nil})) \rightarrow \text{cons}(H, \text{nil})$
- $\text{l2t}(\text{cons}(H_{1}, \text{cons}(H_{2}, L))) \rightarrow \text{cons}(\text{merge}(H_{1}, H_{2}), \text{l2t}(L))$
- $\text{list2heap}(L) \rightarrow \text{hd}(\text{l2t}(\text{map}(\lambda x. \text{node}(x, \text{leaf}, \text{leaf}), L)))$

The static recursion components for $\text{foldT}$ consists of

$$\{\text{foldT}^2(\lambda xyz. F(x, y, z), X, \text{node}(Y, H_{11}, H_{12})) \rightarrow \text{foldT}(\lambda xyz. F(x, y, z), X, H_{11})\}$$

for $i = 1, 2$, and their union. By taking $\pi(\text{foldT}) = 3$, these components satisfy the subterm criterion. The static recursion components for add, map and merge.
also satisfy the subterm criterion. Hence it suffices to show that the following three static recursion components for \( l_2 t \) are non-looping:

\[
\begin{align*}
\{ & l_2 t^2 (\text{cons}(H_1, \text{cons}(H_2, L))) \rightarrow l_2 t^2 (\text{merge}(H_1, H_2), l_2 t(L)) \} \quad (1) \\
\{ & l_2 t^2 (\text{cons}(H_1, \text{cons}(H_2, L))) \rightarrow l_2 t^2 (L) \} \quad (2) \\
\{ & (1), (2) \}
\end{align*}
\]

The component \{(2)\} satisfies the subterm criterion. By taking \( \pi(\text{cons}) = [2] \) and \( \pi(\text{cons}(L)) = \pi(\text{cons}(L)) = 1 \), we can orient the static dependency pairs \{(1)\} and \{(2)\} by using the normal higher-order recursive path ordering [12]:

\[
\begin{align*}
\pi(\text{cons}(H_1, \text{cons}(H_2, L))) & \equiv \pi(\text{cons}(L)) >^n_{\text{horpo}} \pi(\text{cons}(H_1, H_2), l_2 t(L))) \\
\pi(\text{cons}(H_1, \text{cons}(H_2, L))) & \equiv \pi(\text{cons}(L)) >^n_{\text{horpo}} L \equiv \pi(\text{cons}(L))
\end{align*}
\]

However, in contrast to Example 4.5, the non-loopingness of \{(1)\} and \{(1), (2)\} cannot be shown with the previous techniques. Indeed, we cannot solve the constraint \( R_{\text{heap}} \subseteq \geq \). More precisely, we cannot orient the rule for \( \text{hd} \), because \( \pi(\text{hd}(X, L)) \equiv \pi(\text{hd}(L)) \) does not contain the variable \( X \) occurring in the right-hand side.

The notion of usable rule solves this problem, that is, it allows us to ignore the rewrite rule for \( \text{hd} \) for showing the non-loopingness of \( l_2 t \).

**Definition 5.2 (Usable rules)** We denote \( f >^a e g \) if \( g \) is a defined symbol and there is some \( l \rightarrow r \in R \) such that \( \text{top}(l) = f \) and \( g \) occurs in \( r \).

We define the set \( U(t) \) of usable rules of a term \( t \) as follows. If, for every \( X(\overline{\alpha}) \in \text{Sub}(t), \overline{\alpha} \) are distinct bound variables, then \( U(t) = \{ l \rightarrow r \in R \mid f >^a e \text{top}(l) \} \). Otherwise, \( U(t) = R \). The usable rules of a static recursion component \( C \) is \( U(C) = \bigsqcup \{ U(t^2) \mid t^2 \rightarrow t^3 \in C \} \).

For each \( \alpha \in B \), we associate the new function symbols \( \bot_\alpha \) and \( c_\alpha \) with type \( \bot_\alpha = \alpha \) and \( \text{type}(c_\alpha) = \alpha \rightarrow \alpha \rightarrow \alpha \). We define the HRS \( C_e \) as \( C_e = \{ c_\alpha(x_1, x_2) \rightarrow x_1 \mid \alpha \in B, i = 1, 2 \} \).

Hereafter we omit the index \( \alpha \) whenever no confusion arises.

When we show the non-loopingness of a static recursion component using a reduction pair, Proposition 3.15 requires showing that \( R \subseteq \geq \). The non-loopingness is not guaranteed by simply replacing \( R \) with \( U(C) \). We can supplement the gap with the HRS \( C_e \).

**Theorem 5.3** Let \( R \) be a finitely-branching PFP-HRS. Then \( C \in SRC(R) \) is non-looping if there exists a reduction pair \( (\geq, >) \) such that \( U(C) \cup C_e \subseteq \geq, C \subseteq \geq \cup > \), and \( C \cap > \neq \emptyset \).

The proof of this theorem will be given at the end of this section.

**Example 5.4** We show the termination of the PFP-HRS \( R_{\text{heap}} \) in Example 5.1. We have to show the non-loopingness of the components \{(1)\} and \{(1), (2)\}. To this end, it suffices to show that the constraint \( U(\{(1), (2)\}) \subseteq C_e \leq \geq \) can be solved (instead of \( R_{\text{heap}} \subseteq \geq \)). The usable rules of \{(1), (2)\} are:

\[
\begin{align*}
\text{merge}(H, \text{leaf}) & \rightarrow H \\
\text{merge}(\text{leaf}, H) & \rightarrow H \\
\text{merge}(\text{node}(X_1, H_{11}, H_{12}), \text{node}(X_2, H_{21}, H_{22})) & \rightarrow \text{node}(X_1, H_{11}, \text{merge}(H_{12}, \text{node}(X_2, H_{21}, H_{22}))) \\
\text{merge}(\text{node}(X_1, H_{11}, H_{12}), \text{node}(X_2, H_{21}, H_{22})) & \rightarrow \text{node}(X_2, \text{merge}(\text{node}(X_1, H_{11}, H_{12}), H_{21}), H_{22}) \\
l_2 t(\text{nil}) & \rightarrow \text{nil} \\
l_2 t(\text{cons}(H, \text{nil})) & \rightarrow \text{cons}(H, \text{nil}) \\
l_2 t(\text{cons}(H_1, \text{cons}(H_2, L))) & \rightarrow l_2 t(\text{cons}(\text{merge}(H_1, H_2), l_2 t(L)))
\end{align*}
\]

The weak reduction order \( >^n_{\text{horpo}} \) orient the rules. Since \( C_e \subseteq (>^n_{\text{horpo}}, \pi) \), we conclude that \( R_{\text{heap}} \) is terminating.

In the rest of this section, we present a proof of Theorem 5.3. We assume that \( R \) is a finitely-branching PFP-HRS, \( C \) is a static recursion component of \( R \), and \( \Delta = \{ \text{top}(l) \mid l \rightarrow r \in R \setminus U(C) \} \).

The key idea of the proof is to use the following interpretation \( I \).

Thanks to the Well-ordering theorem, we assume that every non-empty set of terms \( T \) has a least element \( \text{least}(T) \).

**Definition 5.5** For a terminating term \( t \in T_a, I(t) \) is defined as follows:

\[
I(t) \equiv \begin{cases}
\lambda x. I(t') & \text{if } t \equiv \lambda x. t' \\
\text{c}_\alpha(a(t_{\alpha})) & \text{if } t \equiv a(t_{\alpha}) \text{ and } \alpha \notin \Delta \\
\text{c}_\alpha(a(t_{\alpha}), \text{Red}_\alpha(I(t') \mid t \rightarrow^* t')) & \text{if } t \equiv a(t_{\alpha}) \text{ and } \alpha \in \Delta
\end{cases}
\]

Here, for each \( \alpha \in B \), \( \text{Red}_\alpha(T) \) is defined as \( \bot_\alpha \) if \( T = \emptyset \); otherwise \( \text{c}_\alpha(u, \text{Red}_\alpha(T \setminus \{ u \}) \mid u) \) where \( u \equiv \text{least}(T) \). We also define \( \theta^t \) by \( \theta^t(x) \equiv I(\theta(x)) \) for a terminating substitution \( \theta \).
The interpretation $I$ is inductively defined on terminating terms with respect to $\triangleright_{\text{sub}} \cup \frac{\alpha}{\beta}$, which is well-founded on terminating terms. Moreover, the set \{\(I(t') \mid t \xrightarrow{\frac{\alpha}{\beta}} t'\)\} is finite because $R$ is finitely branching. Hence, the above definition of $I$ is well-defined. As for argument filterings (Theorem 4.2), this interpretation never destroys well-typedness.

**Theorem 5.6** For any terminating $t$, $I(t)$ is well-typed and $\text{type}(I(t)) = \text{type}(t)$.

**Proof.** It can be easily proved by induction on $t$ ordered by $\triangleright_{\text{sub}} \cup \frac{\alpha}{\beta}$.

**Lemma 5.7** Let $t$ be a term and $\theta$ be a substitution such that $\theta t^1$ is terminating. Then, $I(\theta t^1) \xrightarrow{\frac{\alpha}{\beta}} I(t)t^1 \xrightarrow{\frac{\alpha}{\beta}} \theta t^1$.

**Proof.** We prove the claim by induction on $\{(\text{type}(x) \mid x \in \text{dom}(\theta))\}$ ordered by the lexicographic combination of the multisets extension $\triangleright_{\text{sub}} \cup \frac{\alpha}{\beta}$ of $\triangleright_{\text{sub}}$ and $\triangleright_{\text{sub}} \cup \frac{\alpha}{\beta}$.

- In case of $t = \lambda x.\theta'$. Since $t \xrightarrow{\frac{\alpha}{\beta}} t'$, we have $I(\lambda x.\theta') \xrightarrow{\frac{\alpha}{\beta}} I(t') \xrightarrow{\frac{\alpha}{\beta}} \theta' t^1$ from the induction hypothesis. Hence we have: $I(\lambda x.\theta') \xrightarrow{\frac{\alpha}{\beta}} \lambda x.\theta t^1 \equiv \lambda x.\lambda x.\theta t^1 \equiv I(\lambda x.\theta')t^1 \equiv I(\lambda x.\theta')t^1 \xrightarrow{\frac{\alpha}{\beta}} \lambda x.\theta t^1 \equiv \lambda x.\lambda x.\theta t^1$.

- In case of $t = \lambda x.\theta'$. Since $t \xrightarrow{\frac{\alpha}{\beta}} t'$, we have $I(\lambda x.\theta') \xrightarrow{\frac{\alpha}{\beta}} I(t') \xrightarrow{\frac{\alpha}{\beta}} \theta t^1$ from the induction hypothesis. Hence we have: $I(\lambda x.\theta') \xrightarrow{\frac{\alpha}{\beta}} \lambda x.\theta t^1 \equiv \lambda x.\lambda x.\theta t^1 \equiv I(\lambda x.\theta')t^1 \equiv I(\lambda x.\theta')t^1 \xrightarrow{\frac{\alpha}{\beta}} \lambda x.\theta t^1 \equiv \lambda x.\lambda x.\theta t^1$.

- In case of $t \equiv \lambda x.\theta'$. Since $t \xrightarrow{\frac{\alpha}{\beta}} t'$, we have $I(\lambda x.\theta') \xrightarrow{\frac{\alpha}{\beta}} I(t') \xrightarrow{\frac{\alpha}{\beta}} \theta t^1$ from the induction hypothesis. Hence we have: $I(\lambda x.\theta') \xrightarrow{\frac{\alpha}{\beta}} \lambda x.\theta t^1 \equiv \lambda x.\lambda x.\theta t^1 \equiv I(\lambda x.\theta')t^1 \equiv I(\lambda x.\theta')t^1 \xrightarrow{\frac{\alpha}{\beta}} \lambda x.\theta t^1 \equiv \lambda x.\lambda x.\theta t^1$.

For the proof of Theorem 5.3, it is enough to show that $I(\theta t^1) \xrightarrow{\frac{\alpha}{\beta}} \theta t^1$. In fact, the corresponding lemma for STRSs was the claim [27]. However, the proof of the previous lemma required the stronger claim $I(\theta t^1) \xrightarrow{\frac{\alpha}{\beta}} I(t)t^1 \xrightarrow{\frac{\alpha}{\beta}} \theta t^1$ for applying the induction hypothesis.

**Lemma 5.8** Let $t$ be a term and $\theta$ be a permutation such that $\theta t^1$ is terminating. Then, $I(\theta t^1) \equiv I(t)t^1$.

**Proof.** We prove the claim by induction on $t$ ordered by $\triangleright_{\text{sub}} \cup \frac{\alpha}{\beta}$.

- In case of $t \equiv \lambda x.\theta'$. Since $t \xrightarrow{\frac{\alpha}{\beta}} t'$, we have $I(\lambda x.\theta') \equiv I(t)t^1$ from the induction hypothesis. Hence we have: $I(\lambda x.\theta') \equiv I(\lambda x.\theta') \equiv \lambda x.\lambda x.\theta t^1 \equiv \lambda x.\lambda x.\theta t^1$.

- In case of $t \equiv \lambda x.\theta'$. Since $t \xrightarrow{\frac{\alpha}{\beta}} t'$, we have $I(\lambda x.\theta') \equiv I(t)t^1$ from the induction hypothesis. Hence we have: $I(\lambda x.\theta') \equiv I(\lambda x.\theta') \equiv \lambda x.\lambda x.\theta t^1 \equiv \lambda x.\lambda x.\theta t^1$.

- In case of $t \equiv \lambda x.\theta'$. Since $t \xrightarrow{\frac{\alpha}{\beta}} t'$, we have $I(\lambda x.\theta') \equiv I(t)t^1$ from the induction hypothesis. Hence we have: $I(\lambda x.\theta') \equiv I(\lambda x.\theta') \equiv \lambda x.\lambda x.\theta t^1 \equiv \lambda x.\lambda x.\theta t^1$.
such that $t \xrightarrow{\nu(C)} t''$, we have $I(t''\theta) \equiv I(t')\theta^{|}$ from the induction hypothesis. Since $\theta$ is a permutation, we have $\{I(t') \mid \theta| \to t'\} = \{I(t') \} t \to t'\}$. Hence we have: $I(f_T(t')\theta) \equiv I(f_T(t')\theta)^{|}$ \text{c}(f(T(t')\theta)), \text{Red}(\{I(t') \mid \theta| \to t'\}) \equiv c(f(T(t')\theta)), \text{Red}(\{I(t') \mid \theta| \to t'\}) \equiv c(f(T(t')\theta)) \equiv I(f_T(t')\theta)^{|} \equiv I(f_T(t')\theta)^{|} \equiv I(f_T(t')\theta)^{|}$.

Lemma 5.5 Let $l \to r \in C \cup U(C)$ and $\theta$ be a substitution such that $r\theta^{|}$ is terminating. Then, $I(r\theta) \equiv r\theta^{|}$.

Proof. We show the stronger property $I(t\theta) \equiv t\theta^{|}$ for any $t \to r \in C \cup U(C)$ and $t \in \text{Sub}(r)$. We prove the claim by induction on $t$. Note that we have no case that $t \equiv f_T(t)$ and $f \in \Delta$.

- In case of $t \equiv \lambda x.t':$ Since $t \xrightarrow{\delta} t'$, we have $I(t\theta) \equiv t\theta^{|}$ from the induction hypothesis. Hence we have: $I(\lambda x.t') \equiv \lambda x.I(t') \equiv \lambda x.t\theta^{|} \equiv \lambda x.t\theta^{|}$.

- In case of $t \equiv a(T_n)$ and $\theta \not\equiv \Delta \cup \text{dom}(\theta):$ For each $i$, since $t \xrightarrow{\delta} a_i$, we have $I(t) \equiv a_i\theta^{|}$ from the induction hypothesis. Hence we have: $I(a(T_n)\theta) \equiv I(a_i\theta) \equiv a_i\theta^{|}$.

- In case of $t \equiv X(T_n)$ and $X \in \text{dom}(\theta):$ Since $\text{type}(X) = \text{type}(X)$, we have $X \equiv \lambda y_1 a(T_n).$ For each $i$, since $t \xrightarrow{\delta} a_i$, we have $I(a_i\theta) \equiv a_i\theta^{|}$ from the induction hypothesis. If $t_1, \ldots, t_n$ are mutually distinct bound variables, then $\{y_i \equiv t_i\theta^{|} \mid i \in \pi\}$ is a permutation, and hence it follows from Lemma 5.8 that $I(T_n) \equiv \lambda y_1 a(T_n).$ Otherwise, $I(T_n) \equiv \lambda y_1 a(T_n)$. This contradicts the well-foundedness of $\Rightarrow$.

Lemma 5.10 If $s \xrightarrow{\nu(C)} t$ and $s$ is terminating, then $I(s) \equiv I(t)$.  

Proof. From $s \xrightarrow{\nu(C)} t$, there exists a rule $l \to r \in R$, a context $E[\theta]$ and a substitution $\sigma$ such that $s \equiv E[\theta]$ and $t \equiv E[\sigma\theta]$. We prove the claim by induction on $|E|$. 

- In case of $E[\theta] \equiv \square$ and $l \to r \in U(C)$: From Lemma 5.7 and 5.9, we have: $I(s) \equiv I(\theta) \equiv \theta^{|} \equiv \theta^{|} \equiv I(r\theta) \equiv I(t)$.

- In case of $E[l] \equiv \lambda x.E'[\theta]$ and $t \equiv E[\theta]$: From Lemma 5.7 and 5.9, we have: $I(s) \equiv I(\theta) \equiv \theta^{|} \equiv \theta^{|} \equiv I(r\theta) \equiv I(t)$.

Finally, we give the proof of the main theorem for usable rules: 

Proof of Theorem 5.3. Assume that static dependency pairs in $\mathcal{C}$ generate an infinite chain $\nu_0 \to \nu_1 \to \nu_2 \to \ldots$, in which every $\nu \to \nu^*$ in $\mathcal{C}$ occurs infinitely many times. Then there exist $\theta_0, \theta_1, \theta_2, \ldots$ such that for each $i$, $\nu_i \theta_i \equiv \nu_{i+1} \theta_{i+1}$. Let $i$ be an arbitrary number. From Lemma 5.7, 5.9 and 5.10, we have: $\nu_i \theta_i \equiv \nu_{i+1} \theta_{i+1}$. This contradicts the well-foundedness of $\Rightarrow$.

6 Conclusion

By using the notion of accessibility [3, 2], we extended in an important way the class of systems to which the static dependency pair method [19] can be applied. We then extended to HRSs some methods initially developed for TRSs: arguments filtering [1] and usable rules [7, 10]. So, together with the subterm criterion for HRSs [19] and the normal higher-order recursive path ordering [12], this paper provides a strong theoretical basis for the development of an efficient automated termination proving for HRSs, since all these methods have been shown quite successful in the termination competition on TRSs [30] and are indeed the basis of current state-of-the-art termination provers for TRSs [8, 10]. We now plan to implement all these techniques, all the more so since some competition on the termination of higher-order rewrite systems is under consideration [24]. Currently, HORPO is the only technique for higher-order rewrite systems that has been implemented [25]. One could also build over [14, 28, 6] to provide certificates for these techniques in the case of HRSs.

However, there are still some theoretical problems. Currently, the static dependency pair method does not handle function definitions involving data type
constructors with functional arguments in a satisfactory way like, for instance, the rule $\text{Sum} 5$ of Van de Pol’s formulation of $\mu$CRL [31]:

$$\Sigma(\lambda d. P d) \circ X \rightarrow \Sigma((\lambda d. P d) \circ X)$$

The first reason is that these arguments are not safe (Definition 3.3). This can be fixed by considering a more complex interpretations for base types [2]. The second reason is that it gives rise to the static dependency pair $\Sigma(\lambda d. P d) \circ X \rightarrow P d \circ X$ the right-hand side of which contains a variable $d$ not occurring in the left-hand side. And, currently, no technique can prove the non-loopingness of this static recursion component, a problem occurring also in [4].

Acknowledgments We would like to thank the anonymous referees for their helpful comments.

This research was partially supported by MEXT KAKENHI #20500008.

References


