Argument Filterings and Usable Rules in Higher-Order Rewrite Systems*

Sho Suzuki[1], Keiichirou Kusakari[1], and Frédéric Blanqui[2]

[1] Graduate School of Information Science, Nagoya University, Japan
[2] INRIA, France

Abstract. 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 paper, 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) [28] for handling functional variables and abstractions have been proposed [12,21,10,22,14]. In this paper, we consider higher-order rewrite systems (HRSs) [21], that is, rewriting on β-normal η-long simply-typed λ-terms using higher-order matching.

For example, the typical higher-order function *foldl* can be defined by the following HRS:

$$R_{\mathrm{foldl}} = \begin{cases} \mathrm{foldl}(\lambda xy.F(x,y), X, \mathrm{nil}) \to X \\ \mathrm{foldl}(\lambda xy.F(x,y), X, \mathrm{cons}(Y,L)) \to \mathrm{foldl}(\lambda xy.F(x,y), F(X,Y), L) \end{cases}$$

Then, the functions *sum* and *len*, computing the sum of the elements and the number of elements respectively, can be defined by the following HRSs:

$$R_{\mathrm{sum}} = R_{\mathrm{foldl}} \cup \begin{cases} \mathrm{add}(0,Y) \to Y \\ \mathrm{add}(\mathrm{s}(X),Y) \to \mathrm{s}(\mathrm{add}(X,Y)) \\ \mathrm{sum}(L) \to \mathrm{foldl}(\lambda xy.\mathrm{add}(x,y), 0, L) \end{cases}$$

$$R_{\mathrm{len}} = R_{\mathrm{foldl}} \cup \left\{ \mathrm{len}(L) \to \mathrm{foldl}(\lambda xy.\mathrm{s}(x), 0, L) \right.$$

The static dependency pair method is a method for proving the termination of higher-order rewrite systems. It combines the dependency pair method

* This research was partially supported by MEXT KAKENHI #20500008.

introduced for first-order rewrite systems [1] with Tait and Girard's notion of strong computability introduced for typed λ-calculi [8]. It was first introduced for simply-typed term rewriting systems (STRSs) [16] and then extended to HRSs [18]. The static dependency pair method consists in showing the non-loopingness of each static recursion component independently, the set of static recursion components being computed through some static analysis of the possible sequences of function calls.

This method applies only to plain function-passing (PFP) systems. 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 $R_{\mathrm{sum}} \cup R_{\mathrm{len}}$, the static dependency pair method returns the following two components:

$$\left\{ \mathrm{foldl}^\sharp(\lambda xy.F(x,y), X, \mathrm{cons}(Y,L)) \to \mathrm{foldl}^\sharp(\lambda xy.F(x,y), F(X,Y), L) \right\}$$
$$\left\{ \mathrm{add}^\sharp(\mathrm{s}(X), Y) \to \mathrm{add}^\sharp(X,Y) \right\}$$

The static dependency pair method proves the termination of the HRS $R_{\mathrm{sum}} \cup R_{\mathrm{len}}$ 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 [14,17]. 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 [6,9] and has been extended to STRSs [26,17]. In this paper, we extend it to HRSs.

All together, this paper provides a strong theoretical basis for the development of an automated 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 [7,9].

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 [21,20].

The set $\mathcal{S}$ 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 $\rhd_s$ the strict subterm relation on types.

A *preterm* is generated from an infinite set of type variables $\mathcal{V}$ and a set of typed function symbols Σ disjoint from $\mathcal{V}$ by λ-abstraction and λ-application. The set of typed preterms is denoted with $\mathcal{T}^{pre}$. We denote by $t\!\downarrow$ the η-long β-normal form of a simply-typed preterm t. The set $\mathcal{T}$ of *(simply-typed) terms* is defined as $\{t\!\downarrow \mid t \in \mathcal{T}^{pre}\}$. The unique type of a term t is denoted by *type*(t). We write $\mathcal{V}_\alpha$ (resp. $\mathcal{T}_\alpha$) as the set of variables (resp. terms) of type α, The α-equivalence of terms is denoted by $\equiv$. The set of free variables in a term t is denoted by $FV(t)$. In general, a term t is of the form $\lambda x_1 \dots x_m.at_1 \dots t_n$ where $a \in \Sigma \cup \mathcal{V}$. We abbreviate this by $\lambda\overline{x_m}.a(\overline{t_n})$. For a term $t \equiv \lambda\overline{x_m}.a(\overline{t_n})$, the symbol a, denoted by *top*(t), is the *top symbol* of t, and the set $\{\overline{t_n}\}$, denoted by *args*(t), is the *arguments* of t. We define the set $Sub(t)$ of *subterms* of t by $\{t\} \cup Sub(s)$ if $t \equiv \lambda x.s$, and $\{t\} \cup \bigcup_{i=1}^{n} Sub(t_i)$ if $t \equiv a(\overline{t_n})$. We use $t \unrhd_{sub} s$ to represent $s \in Sub(t)$, and define $t \rhd_{sub} s$ by $t \unrhd_{sub} s$ and $t \not\equiv s$. The set $Pos(t)$ of *positions* in a term t is the set of strings over positive integers inductively defined as $Pos(\lambda x.t) = \{\varepsilon\} \cup \{1p \mid p \in Pos(t)\}$ and $Pos(a(\overline{t_n})) = \{\varepsilon\} \cup \bigcup_{i=1}^{n}\{ip \mid p \in Pos(t_i)\}$. The *prefix order* $\prec$ on positions is defined by $p \prec q$ iff $pw = q$ for some $w \neq \varepsilon$. The subterm of t at position p is denoted by $t|_p$. The size $|t|$ of t is defined as the cardinality of $Pos(t)$.

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

Following [20], a *higher-order rewrite rule* is a pair (l, r) of terms, denoted by $l \rightarrow r$, such that *top*$(l) \in \Sigma$, *type*$(l) =$ *type*$(r) \in \mathcal{B}$ and $FV(l) \supseteq FV(r)$. Since, by definition, terms are in η-long form, function symbols are always applied to the same (maximal) number of arguments. Considering non-η-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* $\underset{R}{\rightarrow}$ of an HRS R is defined by $s \underset{R}{\rightarrow} t$ iff $s \equiv C[l\theta\!\downarrow]$ and $t \equiv C[r\theta\!\downarrow]$ for some rewrite rule $l \rightarrow r \in R$, context $C[\,]$ and substitution θ. The transitive and reflexive-transitive closures of $\underset{R}{\rightarrow}$ are denoted by $\underset{R}{\overset{+}{\rightarrow}}$ and $\underset{R}{\overset{*}{\rightarrow}}$, respectively. An HRS R is said to be *finitely branching* if $\{t' \mid t \underset{R}{\rightarrow} t'\}$ 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 $>$ on terms is a *reduction order* if $>$ is closed under substitution and context. An HRS R is terminating iff $R \subseteq >$ for some reduction order $>$.

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 $type(t) \in \mathcal{B}$, and $\forall u \in \mathcal{T}_\alpha.(SC(R,u) \Rightarrow SC(R,(tu){\downarrow}))$ if $type(t) = \alpha \to \beta$. We also define the set $\mathcal{T}_{SC}^{args}(R) = \{t \mid \forall u \in args(t).SC(R,u)\}$.

3 Improved Static Dependency Pair Method

In this section, we introduce the static dependency pair method for plain function-passing (PFP) HRSs [18] but extend the class of PFP systems by redefining the notion of safe subterms by using the notion of accessibility introduced in [2].

Definition 3.1 (Stable subterms). *The* stable subterms *of t are $SSub(t) = SSub_{FV(t)}(t)$ where $SSub_X(t) = \{t\} \cup SSub'_X(t)$, $SSub'_X(\lambda x.s) = SSub_X(s)$, $SSub'_X(a(\overline{t_n})) = \bigcup_{i=1}^n SSub_X(t_i)$ if $a \notin X$, and $SSub'_X(t) = \emptyset$ otherwise.*

Lemma 3.1. *(1) $SSub(t) \subseteq Sub(t)$. (2) If $u \in SSub(t)$ and $dom(\theta) \subseteq FV(t)$, then $u\theta{\downarrow} \in SSub(t\theta{\downarrow})$. (3) If $u \in Sub(t)$ and $t \in SN$, then $u \in SN$.*

Definition 3.2 (Safe subterms - New definition). *The set of* safe subterms *of a term l is $safe(l) = \bigcup_{l' \in args(l)} \{t{\downarrow} \mid t \in Acc(l'), FV(t) \subseteq FV(l')\}$ where $t \in Acc(l')$ (t is* accessible *in l') if either:*

1. *$t = l'$ or $t \in SSub(l')$, $type(t) \in \mathcal{B}$ and $FV(t) \subseteq FV(l')$,*
2. *$\lambda x.t \in Acc(l')$ and $x \notin FV(l')$,*
3. *$t(x{\downarrow}) \in Acc(l')$ and $x \notin FV(t) \cup FV(l')$,*
4. *$f(\overline{t_n}) \in Acc(l')$, $t_i = \lambda\overline{x_k}.t$, $type(t) \in \mathcal{B}$ and $\{\overline{x_k}\} \cap FV(t) = \emptyset$,*
5. *$x(\overline{t_n}) \in Acc(l')$, $t_i = t$ and $x \notin FV(\overline{t_n}) \cup FV(l')$.*

Strictly speaking, $safe(l)$ may not be included in $Sub(l)$ and, because of (3), accessible terms are β-normal preterms not necessarily in η-long form.

Definition 3.3 (Plain Function-Passing [18]). *An HRS R is* plain function-passing *(PFP) if for any $l \to r \in R$ and $Z(\overline{r_n}) \in Sub(r)$ such that $Z \in FV(r)$, there exists $k \le n$ such that $Z(\overline{r_k}){\downarrow} \in safe(l)$.*

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

The definition of safeness given in [18] corresponds to case (1). This new definition therefore includes much more terms, mainly higher-order patterns [19]. 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:

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

Indeed, $l' = \lambda x.\sin(Fx) \in Acc(l')$ by (1), $\sin(Fx) \in Acc(l')$ by (2), $Fx \in Acc(l')$ by (2) and $F \in Acc(l')$ by (3). Therefore, $safe(l) = \{l', \lambda x.Fx, y\}$. With the previous definition, we had $safe(l) = \{l', y\}$ only.

For the results presented in [18] to still hold, it suffices to check that this new definition of safeness still preserves computability (Lemma 4.3 in [18]). This can be shown by following the proof of Lemma 10 in [2].

Lemma 3.2. *Let R be an HRS, $l \to r \in R$, θ be a substitution. Then $l\theta\!\downarrow \in \mathcal{T}_{SC}^{args}(R)$ implies $SC(R, s\theta\!\downarrow)$ for any $s \in \mathit{safe}(l)$.*

Proof. We first prove that $t\theta\!\downarrow$ is computable whenever $t \in \mathit{Acc}(l')$, $l'\theta\!\downarrow$ is computable and $x\theta$ is computable whenever $x \in FV(t) \setminus FV(l')$.

1. Immediate if $t = l'$. Otherwise, since $l'\theta\!\downarrow$ is computable, $l'\theta\!\downarrow$ is strongly normalizing. Since $FV(t) \subseteq FV(l')$, we can assume that $\mathit{dom}(\theta) \subseteq FV(l')$ wlog. Hence, by Lemma 3.1, $t\theta\!\downarrow \in \mathit{Sub}(l'\theta\!\downarrow)$ and $t\theta\!\downarrow$ is SN. Therefore, since $\mathit{type}(t) \in \mathcal{B}$, $t\theta\!\downarrow$ is computable.
2. and 3. By definition of computability.
4. Since, on base types, computability is equivalent to SN and $\{\overline{x_k}\} \cap FV(t) = \emptyset$.
5. Since projections are computable.

Let now $u \in \mathit{safe}(l)$. We have $u \equiv t\!\downarrow$ for some $t \in \mathit{Acc}(l')$ and $l' \in \mathit{args}(l)$ with $FV(t) \subseteq FV(l')$. The term $l'\theta\!\downarrow$ is computable since $l\theta\!\downarrow \in \mathcal{T}_{SC}^{args}(R)$. Since $FV(t) \subseteq FV(l')$, there is no $x \in FV(t) \setminus FV(l')$. Therefore, $u\theta\!\downarrow \equiv t\theta\!\downarrow$ is 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. 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 [18].

Definition 3.4 (Static dependency pair [18]). *Let R be an HRS. All top symbols of the left-hand sides of rewrite rules, denoted by $\mathcal{D}_R$, are called* defined symbols*, whereas all other function symbols, denoted by $\mathcal{C}_R$, are* constructors.

We define the marked *term $t^\sharp$ by $f^\sharp(\overline{t_n})$ if t has the form $f(\overline{t_n})$ with $f \in \mathcal{D}_R$; otherwise $t^\sharp \equiv t$. Then, let $\mathcal{D}_R^\sharp = \{f^\sharp \mid f \in \mathcal{D}_R\}$.*

We also define the set of candidate subterms *as follows: $\mathit{Cand}(\lambda\overline{x_m}.a(\overline{t_n})) = \{\lambda\overline{x_m}.a(\overline{t_n})\} \cup \bigcup_{i=1}^{n} \mathit{Cand}(\lambda\overline{x_m}.t_i)$.*

Now, a pair $\langle\, l^\sharp,\ a^\sharp(\overline{r_n})\,\rangle$, denoted by $l^\sharp \to a^\sharp(\overline{r_n})$, is said to be a static dependency pair *in R if there exists $l \to r \in R$ such that $\lambda\overline{x_m}.a(\overline{r_n}) \in \mathit{Cand}(r)$, $a \in \mathcal{D}_R$, and $a(\overline{r_k})\!\downarrow \notin \mathit{safe}(l)$ for all $k \leq n$. We denote by $SDP(R)$ the set of static dependency pairs in R.*

Example 3.1. Let R_{ave} be the following PFP-HRS:

$$R_{\mathrm{ave}} = R_{\mathrm{sum}} \cup R_{\mathrm{len}} \cup \begin{cases} \mathrm{sub}(X, 0) \to X \\ \mathrm{sub}(0, Y) \to 0 \\ \mathrm{sub}(\mathrm{s}(X), \mathrm{s}(Y)) \to \mathrm{sub}(X, Y) \\ \mathrm{div}(0, \mathrm{s}(Y)) \to 0 \\ \mathrm{div}(\mathrm{s}(X), \mathrm{s}(Y)) \to \mathrm{s}(\mathrm{div}(\mathrm{sub}(X, Y), \mathrm{s}(Y))) \\ \mathrm{ave}(L) \to \mathrm{div}(\mathrm{sum}(L), \mathrm{len}(L)) \end{cases}$$

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

$$\left\{\begin{array}{rcl}
\mathrm{foldl}^\sharp(\lambda xy.F(x,y), X, \mathrm{cons}(Y,L)) & \to & \mathrm{foldl}^\sharp(\lambda xy.F(x,y), F(X,Y), L)\\
\mathrm{add}^\sharp(\mathrm{s}(X),Y) & \to & \mathrm{add}^\sharp(X,Y)\\
\mathrm{sum}^\sharp(L) & \to & \mathrm{foldl}^\sharp(\lambda xy.\mathrm{add}(x,y), 0, L)\\
\mathrm{sum}^\sharp(L) & \to & \mathrm{add}^\sharp(x,y)\\
\mathrm{sub}^\sharp(\mathrm{s}(X),\mathrm{s}(Y)) & \to & \mathrm{sub}^\sharp(X,Y)\\
\mathrm{div}^\sharp(\mathrm{s}(X),\mathrm{s}(Y)) & \to & \mathrm{div}^\sharp(\mathrm{sub}(X,Y),\mathrm{s}(Y))\\
\mathrm{div}^\sharp(\mathrm{s}(X),\mathrm{s}(Y)) & \to & \mathrm{sub}^\sharp(X,Y)\\
\mathrm{len}^\sharp(L) & \to & \mathrm{foldl}^\sharp(\lambda xy.\mathrm{s}(x), 0, L)\\
\mathrm{ave}^\sharp(L) & \to & \mathrm{div}^\sharp(\mathrm{sum}(L),\mathrm{len}(L))\\
\mathrm{ave}^\sharp(L) & \to & \mathrm{sum}^\sharp(L)\\
\mathrm{ave}^\sharp(L) & \to & \mathrm{len}^\sharp(L)
\end{array}\right.$$

Definition 3.5 (Static dependency chain [18]). *Let R be an HRS. A sequence $u_0^\sharp \to v_0^\sharp, u_1^\sharp \to v_1^\sharp, \ldots$ of static dependency pairs is a* static dependency chain *in R if there exist $\theta_0, \theta_1, \ldots$ such that $v_i^\sharp\theta_i{\downarrow} \xrightarrow[R]{*} u_{i+1}^\sharp\theta_{i+1}{\downarrow}$ and $u_i\theta_i{\downarrow}, v_i\theta_i{\downarrow} \in \mathcal{T}_{SC}^{args}(R)$ for all i.*

Note that, for all i, $u_i^\sharp\theta_i$ and $v_i^\sharp\theta_i$ are terminating, since strong computability implies termination.

Proposition 3.1. *[18] Let R be a PFP-HRS. If there exists no infinite static dependency chain then R is terminating.*

Definition 3.6 (Static recursion component [18]). *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 $u^\sharp \to v^\sharp$ to $u'^\sharp \to v'^\sharp$ if the sequence $u^\sharp \to v^\sharp$, $u'^\sharp \to v'^\sharp$ 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 $u^\sharp \to v^\sharp \in C$ occurs infinitely many times.*

Proposition 3.2. *[18] 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 3.2. For the PFP-HRS R_{ave} in Example 3.1, the static dependency graph of R_{ave} is shown in Fig. 1. Then the set $SRC(R_{\mathrm{ave}})$ consists of the following four static recursion components:

$$\begin{array}{l}
\{\,\mathrm{foldl}^\sharp(\lambda xy.F(x,y), X, \mathrm{cons}(Y,L)) \to \mathrm{foldl}^\sharp(\lambda xy.F(x,y), F(X,Y), L)\,\}\\
\{\,\mathrm{add}^\sharp(\mathrm{s}(X),Y) \to \mathrm{add}^\sharp(X,Y)\,\}\\
\{\,\mathrm{sub}^\sharp(\mathrm{s}(X),\mathrm{s}(Y)) \to \mathrm{sub}^\sharp(X,Y)\,\}\\
\{\,\mathrm{div}^\sharp(\mathrm{s}(X),\mathrm{s}(Y)) \to \mathrm{div}^\sharp(\mathrm{sub}(X,Y),\mathrm{s}(Y))\,\}
\end{array}$$

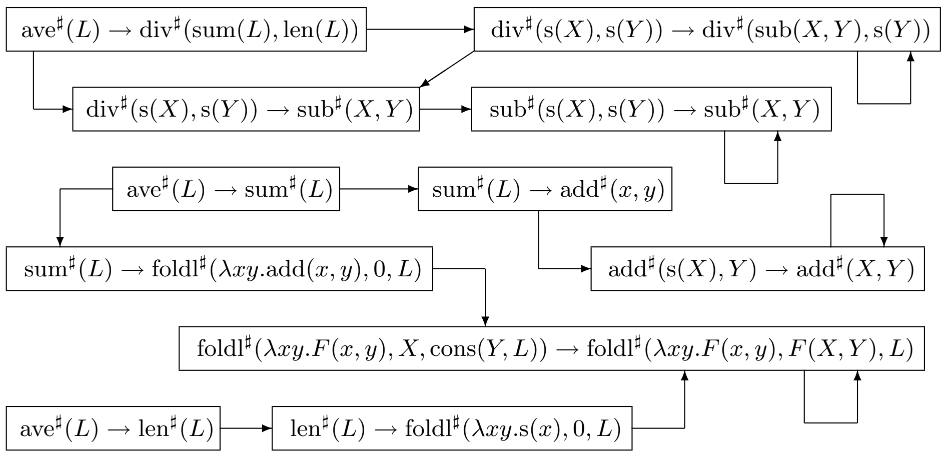
$\mathrm{ave}^\sharp(L) \to \mathrm{div}^\sharp(\mathrm{sum}(L), \mathrm{len}(L))$
$\mathrm{div}^\sharp(\mathrm{s}(X), \mathrm{s}(Y)) \to \mathrm{div}^\sharp(\mathrm{sub}(X,Y), \mathrm{s}(Y))$
$\mathrm{div}^\sharp(\mathrm{s}(X), \mathrm{s}(Y)) \to \mathrm{sub}^\sharp(X,Y)$
$\mathrm{sub}^\sharp(\mathrm{s}(X), \mathrm{s}(Y)) \to \mathrm{sub}^\sharp(X,Y)$
$\mathrm{ave}^\sharp(L) \to \mathrm{sum}^\sharp(L)$
$\mathrm{sum}^\sharp(L) \to \mathrm{add}^\sharp(x,y)$
$\mathrm{sum}^\sharp(L) \to \mathrm{foldl}^\sharp(\lambda xy.\mathrm{add}(x,y), 0, L)$
$\mathrm{add}^\sharp(\mathrm{s}(X), Y) \to \mathrm{add}^\sharp(X,Y)$
$\mathrm{foldl}^\sharp(\lambda xy.F(x,y), X, \mathrm{cons}(Y,L)) \to \mathrm{foldl}^\sharp(\lambda xy.F(x,y), F(X,Y), L)$
$\mathrm{ave}^\sharp(L) \to \mathrm{len}^\sharp(L)$
$\mathrm{len}^\sharp(L) \to \mathrm{foldl}^\sharp(\lambda xy.\mathrm{s}(x), 0, L)$

Fig. 1. The static dependency graph of R_{ave}

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 [9], and then extended to STRSs [16] and HRSs [18]. Reduction pairs [15] are an abstraction of the notion of weak-reduction order [1].

Definition 3.7 (Subterm criterion [18]). *Let R be an HRS and $C \in SRC(R)$. We say that C satisfies the* subterm criterion *if there exists a function π from $\mathcal{D}_R^\sharp$ to non-empty sequences of positive integers such that:*

- $u|_{\pi(top(u^\sharp))} \rhd_{sub} v|_{\pi(top(v^\sharp))}$ *for some $u^\sharp \to v^\sharp \in C$,*
- *and the following conditions hold for every $u^\sharp \to v^\sharp \in C$:*
  - $u|_{\pi(top(u^\sharp))} \unrhd_{sub} v|_{\pi(top(v^\sharp))}$,
  - $\forall p \prec \pi(top(u^\sharp)).top(u|_p) \notin FV(u)$,
  - *and* $\forall q \prec \pi(top(v^\sharp)).q = \varepsilon \vee top(v|_q) \notin FV(v) \cup \mathcal{D}_R$.

Definition 3.8 (Reduction pair, Weak reduction order [1,15]). *A pair $(\gtrsim, >)$ of relations is a* reduction pair *if $\gtrsim$ and $>$ satisfy the following properties:*

- *$>$ is well-founded and closed under substitutions,*
- *$\gtrsim$ is closed under contexts and substitutions,*
- *and $\gtrsim \cdot > \subseteq >$ or $> \cdot \gtrsim \subseteq >$.*

In particular, $\gtrsim$ is a weak reduction order *if $(\gtrsim, \gtrsim \setminus \lesssim)$ is a reduction pair.*

Proposition 3.3. *[18] 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 $(\gtrsim, >)$ such that $R \subseteq \gtrsim$, $C \subseteq \gtrsim \cup >$ and $C \cap > \neq \emptyset$.*

Example 3.3. Let $\pi(\mathrm{foldl}^\sharp) = 3$ and $\pi(\mathrm{add}^\sharp) = \pi(\mathrm{sub}^\sharp) = 1$. Then, every static recursion component C except the one for div (cf. Example 3.2) satisfies the subterm criterion in the underlined positions below. Hence, these static recursion components are non-looping.

$$\left\{\mathrm{foldl}^\sharp(\lambda xy.F(x,y), X, \underline{\mathrm{cons}(Y,L)}) \to \mathrm{foldl}^\sharp(\lambda xy.F(x,y), F(X,Y), \underline{L})\right\}$$
$$\left\{\mathrm{add}^\sharp(\underline{\mathrm{s}(X)}, Y) \to \mathrm{add}^\sharp(\underline{X}, Y)\right\} \quad \left\{\mathrm{sub}^\sharp(\underline{\mathrm{s}(X)}, \mathrm{s}(Y)) \to \mathrm{sub}^\sharp(\underline{X}, Y)\right\}$$

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 [14,17]. In this section, we expand this technique to HRSs.

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

$$\pi(\lambda\overline{x_m}.a(\overline{t_n})) \equiv \begin{cases} \lambda\overline{x_m}.\pi(t_i) & \text{if } a \in \Sigma \text{ and } \pi(a) = i \\ \lambda\overline{x_m}.a(\pi(t_{i_1}), \ldots, \pi(t_{i_k})) & \text{if } a \in \Sigma \text{ and } \pi(a) = [i_1, \ldots, i_k] \\ \lambda\overline{x_m}.a(\pi(t_1), \ldots, \pi(t_n)) & \text{if } a \in \mathcal{V} \end{cases}$$

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

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

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

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

Theorem 4.1. *For any argument filtering π and term $t \in \mathcal{T}$, $\pi(t)$ is well-typed under the typing function $type_\pi$ and $type_\pi(\pi(t)) = type(t)$.*

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 $\gtrsim_\pi$ are closed under substitution.

Lemma 4.1. $\pi(t\theta\downarrow) \equiv \pi(t)\theta_\pi\downarrow$.

Proof. We proceed by induction on preterm $t\theta$ ordered with $\xrightarrow[\beta]{} \cup \rhd_{sub}$. We only show the case $t \equiv X(\overline{t_n})$, $X \in \mathcal{V}$ and $n > 0$. The other cases are immediate. Since $type(X) = type(X\theta)$, we have $X\theta \equiv \lambda\overline{y_n}.a(\overline{u_k})$. For each i, since

$t\theta \rhd_{sub} t_i\theta$, we have $\pi(t_i\theta\downarrow) \equiv \pi(t_i)\theta_\pi\downarrow$ from the induction hypothesis. Here we abbreviate $\{1, \ldots, n\}$ by $\overline{n}$. Since $t\theta \equiv (\lambda\overline{y_n}.a(\overline{u_k}))(\overline{t_n\theta}) \xrightarrow[\beta]{+} a(\overline{u_k})\{y_i := t_i\theta\downarrow \mid i \in \overline{n}\}$, we have $\pi(a(\overline{u_k})\{y_i := t_i\theta\downarrow \mid i \in \overline{n}\}\downarrow) \equiv \pi(a(\overline{u_k}))\{y_i := \pi(t_i\theta\downarrow) \mid i \in \overline{n}\}\downarrow$ from the induction hypothesis. Hence we have $\pi(X(\overline{t_n})\theta\downarrow) \equiv \pi((\lambda\overline{y_n}.a(\overline{u_k}))(\overline{t_n\theta\downarrow})\downarrow) \equiv \pi(a(\overline{u_k})\{y_i := t_i\theta\downarrow \mid i \in \overline{n}\}\downarrow) \equiv \pi(a(\overline{u_k}))\{y_i := \pi(t_i\theta\downarrow) \mid i \in \overline{n}\}\downarrow \equiv \pi(a(\overline{u_k}))\{y_i := \pi(t_i)\theta_\pi\downarrow \mid i \in \overline{n}\}\downarrow \equiv (\lambda\overline{y_n}.\pi(a(\overline{u_k})))(\overline{\pi(t_n)\theta_\pi\downarrow})\downarrow \equiv \pi(\lambda\overline{y_n}.a(\overline{u_k}))(\overline{\pi(t_n)\theta_\pi\downarrow})\downarrow \equiv X(\overline{\pi(t_n)})\theta_\pi\downarrow \equiv \pi(X(\overline{t_n}))\theta_\pi\downarrow$. □

Theorem 4.2. *For any reduction order $>$ and argument filtering function π, $\gtrsim_\pi$ is a weak reduction order.*

Proof. It is easily shown that $s \gtrsim_\pi t \Rightarrow C[s] \gtrsim_\pi C[t]$ by induction on $C[\,]$. From Lemma 4.1, we have $s \gtrsim_\pi t \Rightarrow \pi(s) \geq \pi(t) \Rightarrow \pi(s)\theta_\pi\downarrow \geq \pi(t)\theta_\pi\downarrow \Rightarrow \pi(s\theta\downarrow) \geq \pi(t\theta\downarrow) \Rightarrow s\theta\downarrow \gtrsim_\pi t\theta\downarrow$, and $s >_\pi t \Rightarrow \pi(s) > \pi(t) \Rightarrow \pi(s)\theta_\pi\downarrow > \pi(t)\theta_\pi\downarrow \Rightarrow \pi(s\theta\downarrow) > \pi(t\theta\downarrow) \Rightarrow s\theta\downarrow >_\pi t\theta\downarrow$. Remaining properties are routine. □

Example 4.1. Consider the PFP-HRS R_{ave} in Example 3.1. Every static recursion component except $\{\text{div}^\sharp(\text{s}(X), \text{s}(Y)) \to \text{div}^\sharp(\text{sub}(X, Y), \text{s}(Y))\}$ is non-looping (cf. Example 3.3). We can prove its non-loopingness with the argument filtering method, by taking $\pi(\text{sub}) = \pi(\text{div}^\sharp) = [1]$, and the normal higher-order reduction ordering $>^n_{rhorpo}$, written $(>_{rhorpo})_n$ in [11] defined by:

- a neutralization level $\mathcal{L}^j_f = 0$ 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 [11] not to be confused with the argument filtering method) by taking $\mathcal{A}^j_f = \emptyset$ for all f and j (again, these parameters are relevant for functional arguments only),
- a precedence $s_{new} >_{\Sigma_{new}} sub_{new}$ (a symbol f_{new} with $f \in \Sigma$ is a new symbol introduced by the definition of $>^n_{rhorpo}$ in [11], with the same type as f since neutralization levels are null),
- a multiset (or lexicographic) status for $\text{div}^\sharp_{new}$,
- 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 $\pi(\text{div}^\sharp(\text{s}(X), \text{s}(Y))) \equiv \text{div}^\sharp(\text{s}(X)) >^n_{rhorpo} \text{div}^\sharp(\text{sub}(X)) \equiv \pi(\text{div}^\sharp(\text{sub}(X, Y), \text{s}(Y)))$, and $R_{\text{div}} \subseteq (\geq^n_{rhorpo})_\pi$. For instance, $\text{div}^\sharp(\text{s}(X)) >^n_{rhorpo} \text{div}^\sharp(\text{sub}(X))$ since $NF(\text{div}^\sharp(\text{s}(X)))\downarrow_\beta >_{rhorpo} NF(\text{div}^\sharp(\text{sub}(X)))\downarrow_\beta$ and, because $\mathcal{L}^j_f = 0$ and $\mathcal{A}^j_f = \emptyset$, $NF(ft_1 \ldots t_n) = f_{new}NF(t_1) \ldots NF(t_n)$. From Proposition 3.3, the static recursion component for div is non-looping, and R_{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 [6,9] and then extended to STRSs [26,17]. 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 heap ::= leaf | node(nat, heap, heap) and the PFP-HRS R_{heap} defined by the following rules:

$$\begin{cases}
\text{add}(0, Y) \to Y, \quad \text{add}(\text{s}(X), Y) \to \text{s}(\text{add}(X, Y)) \\
\text{map}(\lambda x.F(x), \text{nil}) \to \text{nil} \\
\text{map}(\lambda x.F(x), \text{cons}(X, L)) \to \text{cons}(F(X), \text{map}(\lambda x.F(x).L)) \\
\text{merge}(H, \text{leaf}) \to H, \quad \text{merge}(\text{leaf}, H) \to H \\
\text{merge}(\text{node}(X_1, H_{11}, H_{12}), \text{node}(X_2, H_{21}, H_{22})) \\
\qquad\quad \to \text{node}(X_1, H_{11}, \text{merge}(H_{12}, \text{node}(X_2, H_{21}, H_{22})) \\
\text{merge}(\text{node}(X_1, H_{11}, H_{12}), \text{node}(X_2, H_{21}, H_{22})) \\
\qquad\quad \to \text{node}(X_2, \text{merge}(\text{node}(X_1, H_{11}, H_{12}), H_{21}), H_{22}) \\
\text{foldT}(\lambda xyz.F(x, y, z), X, \text{leaf}) \to X \\
\text{foldT}(\lambda xyz.F(x, y, z), X, \text{node}(Y, H_1, H_2)) \\
\qquad\quad \to F(X, \text{foldT}(\lambda xyz.F(x, y, z), X, H_1), \text{foldT}(\lambda xyz.F(x, y, z), X, H_2)) \\
\text{sumT}(H) \to \text{foldT}(\lambda xyz.\text{add}(x, \text{add}(y, z)), 0, H) \\
\text{hd}(\text{nil}) \to \text{leaf}, \quad \text{hd}(\text{cons}(X, L)) \to X \\
\text{l2t}(\text{nil}) \to \text{nil}, \quad \text{l2t}(\text{cons}(H, \text{nil})) \to \text{cons}(H, \text{nil}) \\
\text{l2t}(\text{cons}(H_1, \text{cons}(H_2, L))) \to \text{l2t}(\text{cons}(\text{merge}(H_1, H_2), \text{l2t}(L))) \\
\text{list2heap}(L) \to \text{hd}(\text{l2t}(\text{map}(\lambda x.\text{node}(x, \text{leaf}, \text{leaf}), L)))
\end{cases}$$

The static recursion components for foldT consists of

$$\{\text{foldT}^\sharp(\lambda xyz.F(x, y, z), X, \text{node}(Y, H_1, H_2)) \to \text{foldT}(\lambda xyz.F(x, y, z), X, H_i)\}$$

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 l2t are non-looping:

$$\begin{array}{l}
\left\{ \text{l2t}^\sharp(\text{cons}(H_1, \text{cons}(H_2, L))) \to \text{l2t}^\sharp(\text{cons}(\text{merge}(H_1, H_2), \text{l2t}(L))) \ \cdots (1) \right\} \\
\left\{ \text{l2t}^\sharp(\text{cons}(H_1, \text{cons}(H_2, L))) \to \text{l2t}^\sharp(L) \ \cdots (2) \right\} \\
\{(1), (2)\}
\end{array}$$

The component $\{(2)\}$ satisfies the subterm criterion. By taking $\pi(\text{cons}) = [2]$ and $\pi(\text{l2t}) = \pi(\text{l2t}^\sharp) = 1$, we can orient the static dependency pairs (1) and (2) by using the normal higher-order recursive path ordering [11]:

$$\begin{array}{l}
\pi(\text{l2t}^\sharp(\text{cons}(H_1, \text{cons}(H_2, L)))) \\
\qquad \equiv \text{cons}(\text{cons}(L)) >^n_{rhorpo} \text{cons}(L) \equiv \pi(\text{l2t}^\sharp(\text{cons}(\text{merge}(H_1, H_2), \text{l2t}(L)))) \\
\pi(\text{l2t}^\sharp(\text{cons}(H_1, \text{cons}(H_2, L)))) \equiv \text{cons}(\text{cons}(L)) >^n_{rhorpo} L \equiv \pi(\text{l2t}^\sharp(L))
\end{array}$$

However, in contrast to Example 4.1, the non-loopingness of $\{(1)\}$ and $\{(1), (2)\}$ cannot be shown with the previous techniques. Indeed, we cannot solve the constraint $R_{\text{heap}} \subseteq \gtrsim$. More precisely, we cannot orient the rule for hd, because $\pi(\text{hd}(\text{cons}(X, L))) \equiv \text{hd}(\text{cons}(L))$ 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 hd for showing the non-loopingness of l2t.

Definition 5.1 (Usable rules). *We denote $f >_{\mathrm{def}} g$ if g is a defined symbol and there is some $l \to r \in R$ such that $top(l) = f$ and g occurs in r.*

We define the set $\mathcal{U}(t)$ of usable rules of a term t as follows. If, for every $X(\overline{t_n}) \in Sub(t)$, $\overline{t_n}$ are distinct bound variables, then $\mathcal{U}(t) = \{l \to r \in R \mid f >^*_{\mathrm{def}} top(l)$ for some $f \in \mathcal{D}_R$ occurs in $t\}$. Otherwise, $\mathcal{U}(t) = R$. The usable rules of a static recursion component C is $\mathcal{U}(C) = \bigcup\{\mathcal{U}(v^\sharp) \mid u^\sharp \to v^\sharp \in C\}$.

For each $\alpha \in \mathcal{B}$, we associate the new function symbols $\bot_\alpha$ and c_α with $type(\bot_\alpha) = \alpha$ and $type(\mathrm{c}_\alpha) = \alpha \to \alpha \to \alpha$. We define the HRS C_e as $C_e = \{\mathrm{c}_\alpha(x_1, x_2) \to x_i \mid \alpha \in \mathcal{B}, i = 1, 2\}$.

Hereafter we omit the index α whenever no confusion arises.

When we show the non-loopingness of a static recursion component using a reduction pair, Proposition 3.3 requires showing that $R \subseteq \gtrsim$. The non-loopingness is not guaranteed by simply replacing R with $\mathcal{U}(C)$. We can supplement the gap with the HRS C_e.

Theorem 5.1. *Let R be a finitely-branching PFP-HRS. Then $C \in SRC(R)$ is non-looping if there exists a reduction pair $(\gtrsim, >)$ such that $\mathcal{U}(C) \cup C_e \subseteq \gtrsim$, $C \subseteq \gtrsim \cup >$, and $C \cap > \neq \emptyset$.*

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

Example 5.2. We show the termination of the PFP-HRS R_{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 $\mathcal{U}(\{(1), (2)\}) \cup C_e \subseteq \gtrsim$ can be solved (instead of $R_{\mathrm{heap}} \subseteq \gtrsim$). The usable rules of $\{(1), (2)\}$ are:

$$\begin{cases} \mathrm{merge}(H, \mathrm{leaf}) \to H \\ \mathrm{merge}(\mathrm{leaf}, H) \to H \\ \mathrm{merge}(\mathrm{node}(X_1, H_{11}, H_{12}), \mathrm{node}(X_2, H_{21}, H_{22})) \\ \qquad \to \mathrm{node}(X_1, H_{11}, \mathrm{merge}(H_{12}, \mathrm{node}(X_2, H_{21}, H_{22})) \\ \mathrm{merge}(\mathrm{node}(X_1, H_{11}, H_{12}), \mathrm{node}(X_2, H_{21}, H_{22})) \\ \qquad \to \mathrm{node}(X_2, \mathrm{merge}(\mathrm{node}(X_1, H_{11}, H_{12}), H_{21}), H_{22}) \\ \mathrm{l2t}(\mathrm{nil}) \to \mathrm{nil} \\ \mathrm{l2t}(\mathrm{cons}(H, \mathrm{nil})) \to \mathrm{cons}(H, \mathrm{nil}) \\ \mathrm{l2t}(\mathrm{cons}(H_1, \mathrm{cons}(H_2, L))) \to \mathrm{l2t}(\mathrm{cons}(\mathrm{merge}(H_1, H_2), \mathrm{l2t}(L))) \end{cases}$$

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

In the rest of this section, we present a proof of Theorem 5.1. We assume that R is a finitely-branching PFP-HRS, C is a static recursion component of R, and $\Delta = \{top(l) \mid l \to r \in R \setminus \mathcal{U}(C)\}$.

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 $\mathrm{least}(T)$.

Definition 5.2. *For a terminating term $t \in \mathcal{T}_\alpha$, $I(t)$ is defined as follows:*

$$I(t) \equiv \begin{cases} \lambda x.I(t') & \textit{if } t \equiv \lambda x.t' \\ a(\overline{I(t_n)}) & \textit{if } t \equiv a(\overline{t_n}) \textit{ and } a \notin \Delta \\ \mathrm{c}_\alpha(a(\overline{I(t_n)}), \mathit{Red}_\alpha(\{I(t') \mid t \xrightarrow[R\setminus\mathcal{U}(C)]{} t'\})) & \textit{if } t \equiv a(\overline{t_n}) \textit{ and } a \in \Delta \end{cases}$$

Here, for each $\alpha \in \mathcal{B}$, $\mathit{Red}_\alpha(T)$ is defined as $\bot_\alpha$ if $T = \emptyset$; otherwise $\mathrm{c}_\alpha(u, \mathit{Red}_\alpha(T \setminus \{u\}))$ where $u \equiv \mathrm{least}(T)$. We also define θ^I by $\theta^I(x) \equiv I(\theta(x))$ for a terminating substitution θ.

The interpretation I is inductively defined on terminating terms with respect to $\rhd_{sub} \cup \xrightarrow[R]{}$, which is well-founded on terminating terms. Moreover, the set $\{I(t') \mid t \xrightarrow[R]{} t'\}$ is finite because R is finitely branching. Hence, the above definition of I is well-defined. As for argument filterings (Theorem 4.1), this interpretation never destroys well-typedness.

Theorem 5.2. *For any terminating t, $I(t)$ is well-typed and $\mathit{type}(I(t)) = \mathit{type}(t)$.*

Proof. It can be easily proved by induction on t ordered by $\rhd_{sub} \cup \xrightarrow[R]{}$. □

Lemma 5.1. *Let t be a term and θ be a substitution such that $t\theta{\downarrow}$ is terminating. Then, $I(t\theta{\downarrow}) \xrightarrow[C_e]{*} I(t)\theta^I{\downarrow} \xrightarrow[C_e]{*} t\theta^I{\downarrow}$.*

Proof. We prove the claim by induction on $(\{\mathit{type}(x) \mid x \in \mathit{dom}(\theta)\}, t)$ ordered by the lexicographic combination of the multiset extension $\rhd_s^{mul}$ of $\rhd_s$, and $\rhd_{sub} \cup \xrightarrow[R]{}$. We only show the following two cases. Remaining cases are routine.

• In case of $t \equiv X(\overline{t_n})$, $X \in \mathit{dom}(\theta)$ and $n > 0$: Thanks to the general assumption $\mathit{type}(X) = \mathit{type}(X\theta)$, we let $X\theta \equiv \lambda\overline{y_n}.a(\overline{u_k})$. Since $\mathit{type}(X) = \alpha_1 \to \cdots \to \alpha_n \to \beta \rhd_s \alpha_i = \mathit{type}(y_i)$ for each i, we have

$$I(a(\overline{u_k})\{y_i := t_i\theta{\downarrow} \mid i \in \overline{n}\}{\downarrow}) \xrightarrow[C_e]{*} I(a(\overline{u_k}))\{y_i := I(t_i\theta{\downarrow}) \mid i \in \overline{n}\}{\downarrow}$$

from the induction hypothesis, where $\overline{n} = \{1, \ldots, n\}$.

For each i, since $t \rhd_{sub} t_i$, we have $I(t_i\theta{\downarrow}) \xrightarrow[C_e]{*} I(t_i)\theta^I{\downarrow} \xrightarrow[C_e]{*} t_i\theta^I{\downarrow}$ from the induction hypothesis. Hence, by Theorem 3.9 in [20] (if $s \xrightarrow[R]{*} t$ and $\theta \xrightarrow[R]{*} \theta'$ then $s\theta{\downarrow} \xrightarrow[R]{*} t\theta'{\downarrow}$), we have: $I(X(\overline{t_n})\theta{\downarrow}) \equiv I((\lambda\overline{y_n}.a(\overline{u_k}))(\overline{t_n\theta{\downarrow}}){\downarrow}) \equiv I(a(\overline{u_k})\{y_i := t_i\theta{\downarrow} \mid i \in \overline{n}\}{\downarrow}) \xrightarrow[C_e]{*} I(a(\overline{u_k}))\{y_i := I(t_i\theta{\downarrow}) \mid i \in \overline{n}\}{\downarrow} \xrightarrow[C_e]{*} I(a(\overline{u_k}))\{y_i := I(t_i)\theta^I{\downarrow} \mid i \in \overline{n}\}{\downarrow} \equiv (\lambda\overline{y_n}.I(a(\overline{u_k})))(\overline{I(t_n)\theta^I{\downarrow}}){\downarrow} \equiv X(\overline{I(t_n)})\theta^I{\downarrow} \equiv I(X(\overline{t_n}))\theta^I{\downarrow}$ and $I(X(\overline{t_n}))\theta^I{\downarrow} \equiv I(a(\overline{u_k}))\{y_i := I(t_i)\theta^I{\downarrow} \mid i \in \overline{n}\}{\downarrow} \xrightarrow[C_e]{*} I(a(\overline{u_k}))\{y_i := t_i\theta^I{\downarrow} \mid i \in \overline{n}\}{\downarrow} \equiv \lambda\overline{y_n}.I(a(\overline{u_k}))(\overline{t_n\theta^I{\downarrow}}){\downarrow} \equiv I(\lambda\overline{y_n}.a(\overline{u_k}))\ (\overline{t_n\theta^I{\downarrow}}){\downarrow} \equiv X(\overline{t_n})\theta^I{\downarrow}$.

• In case of $t \equiv f(\overline{t_n})$ and $f \in \Delta$: For each i, since $t \rhd_{sub} t_i$, we have $I(t_i\theta{\downarrow}) \xrightarrow[C_e]{*} I(t_i)\theta^I{\downarrow} \xrightarrow[C_e]{*} t_i\theta^I{\downarrow}$ from the induction hypothesis. For an arbitrary t'' such that $t \xrightarrow[R\setminus\mathcal{U}(C)]{} t''$, we have $I(t''\theta{\downarrow}) \xrightarrow[C_e]{*} I(t'')\theta^I{\downarrow} \xrightarrow[C_e]{*} t''\theta^I{\downarrow}$ from the induction hypothesis. Hence we have: $I(f(\overline{t_n})\theta{\downarrow}) \equiv \mathrm{c}(f(\overline{I(t_n\theta{\downarrow})}), \mathit{Red}(\{I(t') \mid t\theta{\downarrow} \to t'\})){\downarrow} \xrightarrow[C_e]{*} \mathrm{c}(f(\overline{I(t_n\theta{\downarrow})}), \mathit{Red}(\{I(t''\theta{\downarrow}) \mid t \to t''\})){\downarrow} \xrightarrow[C_e]{*} \mathrm{c}(f(\overline{I(t_n)\theta^I{\downarrow}}), \mathit{Red}(\{I(t'')\theta^I{\downarrow} \mid t \to t''\})){\downarrow} \equiv \mathrm{c}(f(\overline{I(t_n)}), \mathit{Red}(\{I(t'') \mid t \to t''\}))\theta^I{\downarrow} \equiv I(f(\overline{t_n}))\theta^I{\downarrow}$ and $I(f(\overline{t_n}))\theta^I{\downarrow} \equiv \mathrm{c}(f(\overline{I(t_n)\theta^I{\downarrow}}), \mathit{Red}(\{I(t'')\theta^I{\downarrow} \mid t \to t''\})){\downarrow} \xrightarrow[C_e]{} f(\overline{I(t_n)\theta^I{\downarrow}}) \xrightarrow[C_e]{*} f(\overline{t_n\theta^I{\downarrow}}) \equiv f(\overline{t_n})\theta^I{\downarrow}$. □

Lemma 5.2. *Let t be a term and θ be a permutation such that $t\theta\!\downarrow$ is terminating. Then, $I(t\theta\!\downarrow) \equiv I(t)\theta^I\!\downarrow$.*

Proof. We prove the claim by induction on t ordered by $\rhd_{sub} \cup \xrightarrow[R]{}$. We only show the following two cases. Remaining cases are routine.

• In case of $t \equiv X(\overline{t_n})$ and $X \in dom(\theta)$: Since θ is a permutation, we let $X\theta\!\downarrow \equiv X'\!\downarrow$ for a variable X'. Hence we have $I(X(\overline{t_n})\theta\!\downarrow) \equiv I(X'(\overline{t_n\theta\!\downarrow})) \equiv X'(\overline{I(t_n\theta\!\downarrow)}) \equiv X'(\overline{I(t_n)\theta^I\!\downarrow}) \equiv X(\overline{I(t_n)})\theta^I\!\downarrow \equiv I(X(\overline{t_n}))\theta^I\!\downarrow$.

• In case of $t \equiv f(\overline{t_n})$ and $f \in \varDelta$: Since θ is a permutation, we have $\{I(t') \mid t\theta\!\downarrow \to t'\} = \{I(t''\theta\!\downarrow) \mid t \to t''\}$. Hence we have: $I(f(\overline{t_n})\theta\!\downarrow) \equiv I(f(\overline{t_n\theta\!\downarrow})) \equiv \mathrm{c}(f(\overline{I(t_n\theta\!\downarrow)}), \mathit{Red}(\{I(t') \mid t\theta\!\downarrow \to t'\})) \equiv \mathrm{c}(f(\overline{I(t_n\theta\!\downarrow)}), \mathit{Red}(\{I(t''\theta\!\downarrow) \mid t \to t''\})) \equiv \mathrm{c}(f(\overline{I(t_n)\theta^I\!\downarrow}), \mathit{Red}(\{I(t'')\theta^I\!\downarrow \mid t \to t''\})) \equiv \mathrm{c}(f(\overline{I(t_n)}), \mathit{Red}(\{I(t'') \mid t \to t''\}))\theta^I\!\downarrow \equiv I(f(\overline{t_n}))\theta^I\!\downarrow$. ☐

Lemma 5.3. *Let $l \to r \in C \cup \mathcal{U}(C)$ and θ be a substitution such that $r\theta\!\downarrow$ is terminating. Then, $I(r\theta\!\downarrow) \equiv r\theta^I\!\downarrow$.*

Proof. We show the stronger property $I(t\theta\!\downarrow) \equiv t\theta^I\!\downarrow$ for any $l \to r \in C \cup \mathcal{U}(C)$ and $t \in \mathit{Sub}(r)$. We prove the claim by induction on t. Note that we have no case that $t \equiv f(\overline{t_n})$ and $f \in \varDelta$. We only show the case of $t \equiv X(\overline{t_n})$ and $X \in dom(\theta)$. Remaining cases are routine.

Since $\mathit{type}(X) = \mathit{type}(X\theta)$, we have $X\theta \equiv \lambda\overline{y_n}.a(\overline{u_k})$. If $t_1, \ldots, t_n$ are distinct bound variables, then $\{y_i := t_i\theta\!\downarrow \mid i \in \overline{n}\}$ is a permutation, and hence it follows from Lemma 5.2 that $I(X(\overline{t_n})\theta\!\downarrow) \equiv I((\lambda\overline{y_n}.a(\overline{u_k}))(\overline{t_n\theta})\!\downarrow) \equiv I(a(\overline{u_k})\{y_i := t_i\theta\!\downarrow \mid i \in \overline{n}\}\!\downarrow) \equiv I(a(\overline{u_k}))\{y_i := I(t_i\theta\!\downarrow) \mid i \in \overline{n}\}\!\downarrow \equiv I(a(\overline{u_k}))\{y_i := t_i\theta^I\!\downarrow \mid i \in \overline{n}\}\!\downarrow \equiv (\lambda\overline{y_n}.I(a(\overline{u_k})))(\overline{t_n\theta^I})\!\downarrow \equiv I(\lambda\overline{y_n}.a(\overline{u_k}))\ (\overline{t_n\theta^I})\!\downarrow \equiv X(\overline{t_n})\theta^I\!\downarrow$, where $\overline{n} = \{1, \ldots, n\}$. Otherwise, $I(X(\overline{t_n})\theta\!\downarrow) \equiv X(\overline{t_n})\theta\!\downarrow$ and $\theta = \theta^I$, because of $\varDelta = \emptyset$. ☐

Lemma 5.4. *If $s \xrightarrow[R]{} t$ and s is terminating, then $I(s) \xrightarrow[\mathcal{U}(C) \cup C_e]{+} I(t)$.*

Proof. From $s \xrightarrow[R]{} t$, there exists a rule $l \to r \in R$, a context $E[\,]$, and a substitution θ such that $s \equiv E[l\theta\!\downarrow]$ and $t \equiv E[r\theta\!\downarrow]$. We prove the claim by induction on $E[\,]$. In case of $E[\,] \equiv \Box$ and $l \to r \in \mathcal{U}(C)$: From Lemma 5.1 and 5.3, we have $I(s) \equiv I(l\theta\!\downarrow) \xrightarrow[C_e]{*} l\theta^I\!\downarrow \xrightarrow[\mathcal{U}(C)]{} r\theta^I\!\downarrow \equiv I(r\theta\!\downarrow) \equiv I(t)$. In case of $s \equiv f(\overline{s_n})$ and $f \in \varDelta$, we have $I(s) \equiv \mathrm{c}(f(\overline{I(s_n)}), \mathit{Red}(\{I(v) \mid s \to v\})) \xrightarrow[C_e]{} \mathit{Red}(\{I(v) \mid s \to v\}) \xrightarrow[C_e]{+} I(t)$. Remaining cases follows from the induction hypothesis. ☐

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

Proof of Theorem 5.1. Assume that static dependency pairs in C generate an infinite chain $u_0^\sharp \to v_0^\sharp, u_1^\sharp \to v_1^\sharp, \ldots$, in which every $u^\sharp \to v^\sharp \in C$ occurs infinitely many times. Then there exist $\theta_0, \theta_1, \theta_2, \ldots$ such that for each i, $v_i^\sharp\theta_i\!\downarrow \xrightarrow[R]{*} u_{i+1}^\sharp\theta_{i+1}\!\downarrow$. Let i be an arbitrary number. From Lemma 5.1, 5.3 and 5.4, we have: $v_i^\sharp\theta_i^I\!\downarrow \equiv I(v_i^\sharp\theta_i\!\downarrow) \xrightarrow[\mathcal{U}(C) \cup C_e]{*} I(u_{i+1}^\sharp\theta_{i+1}\!\downarrow) \xrightarrow[C_e]{*} u_{i+1}^\sharp\theta_{i+1}^I\!\downarrow$. Hence we have $v_i^\sharp\theta_i^I\!\downarrow \gtrsim u_{i+1}^\sharp\theta_{i+1}^I\!\downarrow \gtrsim v_{i+1}^\sharp\theta_{i+1}^I\!\downarrow$ from $\mathcal{U}(C) \cup C_e \subseteq \gtrsim$. Moreover, from $C \subseteq \gtrsim \cup >$ and $C \cap > \neq \emptyset$, we have $u_j^\sharp\theta_j^I\!\downarrow > v_j^\sharp\theta_j^I\!\downarrow$ for infinitely many j. This contradicts the well-foundedness of $>$. ☐

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 [18] can be applied. We then extended to HRSs some methods initially developed for TRSs: arguments filterings [1] and usable rules [6,9]. So, together with the subterm criterion for HRSs [18] and the normal higher-order recursive path ordering [11], this paper provides a strong theoretical basis for the development of an efficient automated termination provers for HRSs, since all these methods have been shown quite successful in the termination competition on TRSs [29] and are indeed the basis of current state-of-the-art termination provers for TRSs [7,9]. 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 [23]. Currently, HORPO is the only technique for higher-order rewrite systems that has been implemented [24]. One could also build over [13,27,5] 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 $Sum5$ of Van de Pol's formulation of μCRL [30]:

$$\Sigma(\lambda d.Pd) \circ X \to \Sigma(\lambda d.Pd \circ X)$$

The first reason is that these arguments are not safe (Definition 3.2). 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 $\Sigma(\lambda d.Pd) \circ X \to Pd \circ X$ 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].

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, J.-P. Jouannaud and M. Okada. Inductive-data-type Systems. *Theoretical Computer Science*, 272:41-68, 2002.
4. F. Blanqui. Higher-order dependency pairs. In *Proc. of WST'06.*
5. E. Contejean, A. Paskevich, X. Urbain, P. Courtieu, O. Pons and J. Forest. A3PAT, an approach for certified automated termination proofs. In *Proc. of PEPM'10.*
6. J. Giesl, R. Thiemann, P. Schneider-Kamp and S. Falke. Mechanizing and Improving Dependency Pairs. *Journal of Automated Reasoning*, 37(3):155-203, 2006.
7. J. Giesl, P. Schneider-Kamp and R. Thiemann. AProVE 1.2: Automatic termination proofs in the dependency pair framework. In *Proc. IJCAR'06*, LNCS 4130.
8. J.-Y. Girard, Y. Lafont and P. Taylor. *Proofs and Types.* Cambridge University Press, 1988.

9. N. Hirokawa and A. Middeldorp. Tyrolean Termination Tool: Techniques and Features. In *Information and Computation* 205(4):474-511, 2007
10. J.-P. Jouannaud and M. Okada. A computation model for executable higher-order algebraic specification languages. In *Proc. of LICS'91.*
11. J.-P. Jouannaud and A. Rubio. Higher-Order Orderings for Normal Rewriting. In *Proc. of RTA'06*, LNCS 4098.
12. J. W. Klop. *Combinatory Reduction Systems.* PhD thesis, Utrecht Universiteit, The Netherlands, 1980. Published as Mathematical Center Tract 129.
13. A. Koprowski. Certified Higher-Order Recursive Path Ordering. In *Proc. of RTA'06*, LNCS 4098. `http://color.inria.fr/`.
14. K. Kusakari. On Proving Termination of Term Rewriting Systems with Higher-Order Variables. *IPSJ Transactions on Programming*, Vol. 42, No. SIG 7 (PRO 11), p. 35-45, 2001.
15. K. Kusakari, M. Nakamura and Y. Toyama. Elimination Transformations for Associative-Commutative Rewriting Systems, *Journal of Automated Reasoning*, 37(3):205-229, 2006.
16. K. Kusakari and M. Sakai. Enhancing Dependency Pair Method using Strong Computability in Simply-Typed Term Rewriting Systems. *Applicable Algebra in Engineering, Communication and Computing*, 18(5):407-431, 2007.
17. K. Kusakari and M. Sakai. Static Dependency Pair Method for Simply-Typed Term Rewriting and Related Techniques. *IEICE Transactions on Information and Systems*, E92-D(2):235-247, 2009.
18. 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,* E92-D(10):2007-2015, 2009.
19. D. Miller. A logic programming language with lambda-abstraction, function variables, and simple unification. In *Proceedings of the International Workshop on Extensions of Logic Programming*, LNCS 475, 1989.
20. R. Mayr and N. Nipkow. Higher-Order Rewrite Systems and their Confluence. *Theoretical Computer Science*, 192(2):3-29, 1998.
21. N. Nipkow. Higher-order Critical Pairs. In *Proc. LICS'91.*
22. V. van Oostrom. *Confluence for Abstract and Higher-Order Rewriting.* PhD thesis, Vrije Universiteit Amsterdam, The Netherlands, 1994.
23. A. Rubio. `http://termination-portal.org/wiki/Higher_Order`, Feb 2010.
24. A. Rubio. A GNU-Prolog implementation of HORPO. Available on `http://www.lsi.upc.es/~albert/term.html`.
25. M. Sakai, Y. 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.
26. T. Sakurai, K. Kusakari, M. Sakai, T. Sakabe and N. Nishida. Usable Rules and Labeling Product-Typed Terms for Dependency Pair Method in Simply-Typed Term Rewriting Systems. *IEICE Transactions on Information and Systems*, J90-D(4):978-989, 2007. (In Japanese.)
27. C. Sternagel and R. Thiemann. Certification of Termination Proofs using CeTA. In *Proc. of TPHOL'09*, LNCS 5674.
28. Terese. Term Rewriting Systems. *Cambridge Tracts in Theoretical Computer Science*, Vol. 55, Cambridge University Press, 2003.
29. `http://termination-portal.org/wiki/Termination_Competition`.
30. J. van de Pol. *Termination of higher-order rewrite systems*, PhD thesis, Utrecht Universiteit, The Netherlands, 1996.