On New Dependency Pair Method for Proving Termination of Higher-Order Rewrite Systems

Masahiko Sakai[1] and Keiichirou Kusakari[2]

[1] Department of Information Engineering, Nagoya University,
Furo-cho, Chikusa-ku, Nagoya 464-8603, Japan
sakai@nuie.nagoya-u.ac.jp
[2] Research Institute of Electrical Communication, Tohoku University,
Katahira, Aoba-ku, Sendai, 980-8577, Japan
kusakari@nue.riec.tohoku.ac.jp

Abstract. This paper explores how to extend the dependency pair technique for proving termination of higher-order rewrite systems. We introduced a new notion of dependency forest and show that the termination property of higher-order rewrite systems R can be checked by the non-existence of an infinite R-chain, if R is non-duplicating or non-nested. One benefit is we can exclude any term whose top symbol is higher order variable from the second item of dependency pair.

1 Introduction

Higher-order rewrite rules are used in functional programming, logic programming, and theorem proving. Automatic proving of the termination property is especially required for theorem provers. Several orderings for higher-order terms have been investigated by extending recursive path orderings for proving simple termination of term rewriting systems [12, 11, 9, 8]. Among them, Jouannaud and Rubio's ordering has a simple definition by using type information. Iwami, Sakai and Toyama [7] extended the improved recursive decomposition ordering to the higher-order case by using Jouannaud's and Rubio's technique.

There is the dependency pair technique [1–3] for proving termination of term rewriting systems. It is useful, because it gives us a mechanical support for proving non-simple termination. Sakai, Watanabe, Sakabe [10] approaches how to apply dependency pair method to proving termination of higher-order rewrite systems [13]. Since it requires a reduction quasi-ordering having subterm property, we can not use argument filtering method, that is the weak point unfortunately.

This paper introduces the notion of dependency forest and studies the dependency pair technique on non-duplicating or non-nested higher-order rewrite systems. No longer subterm property is needed in exchange for giving up the completeness. The other benefit is that we can exclude any term whose top symbol is higher order variable from the second item of dependency pair.

2 Preliminary concepts

We assume the readers are familiar with the basic concepts and notations of term rewriting systems [5] and typed lambda calculi [4].

Given a set S of *basic types* (or *sorts*), the set τ_S of types is generated from S by the constructor $\to$ for *functional types*, that is, τ_S is the smallest set such that

$$\tau_S \supseteq S$$
$$\tau_S \supseteq \{\alpha \to \beta \mid \alpha, \beta \in \tau_S\}$$

Types that are not basic are called higher-order types. We use α, β to denote types. For a type β in form of $\alpha_1 \to \cdots \to \alpha_n \to \alpha$ where $n \geq 0$ and α is a basic type, the *output type* of β, denoted by $O(\beta)$, is α.

Let V_α be a set of *variables* of type α and $V = \bigcup_{\alpha \in \tau_S} V_\alpha$. Let C_α be a set of *constants* (or *function symbols*) of type α and $C = \bigcup_{\alpha \in \tau_S} C_\alpha$. We assume $V \cap C = \emptyset$, and $V_\alpha \cap V_\beta = \emptyset$ and $C_\alpha \cap C_\beta = \emptyset$ if $\alpha \neq \beta$. We use V_h to stand for the set of higher-order variables.

Constants are denoted by c, d, e, f and g. We use a to denote constants or variables.

The set T_α of *simply typed λ-terms of type α* is the smallest set satisfying the following:

$$T_\alpha \supseteq V_\alpha \cup C_\alpha$$
$$T_\alpha \supseteq \{(st) \mid s \in T_{\alpha' \to \alpha}, t \in T_{\alpha'}\}$$
$$T_\alpha \supseteq \{(\lambda x.s) \mid x \in V_{\beta'}, s \in T_\beta, \alpha = \beta' \to \beta\}$$

We write $t : \alpha$ to stand for $t \in T_\alpha$. Let $T = \bigcup_{\alpha \in \tau_S} T_\alpha$. We call a simply typed λ-term t a *term*. We use l, q, r, s, t, u and v for terms. We use $FV(t)$ for the set of free variables of t and $BV(t)$ for the set of bound variables of t. Let $Var(t) = FV(t) \cup BV(t)$. We assume for convenience that bound variables in a term are all different, and are disjoint from free variables. We use F, G, H, X, and Y for free variables and x, y and z for bound variables. We use $\equiv$ to denote syntactic equality on terms.

A term containing special constants $\Box_{\alpha_1}, \ldots, \Box_{\alpha_n}$ is called a *context* denoted by $C_{\alpha_1,\ldots,\alpha_n}[\ ,\ldots,\ ]$. We use $C_{\alpha_1,\ldots,\alpha_n}[t_1,\ldots,t_n]$ for the term obtained from $C_{\alpha_1,\ldots,\alpha_n}[\ ,\ldots,\ ]$ by replacing $\Box_{\alpha_1},\ldots,\Box_{\alpha_n}$ with $t_1 : \alpha_1, \ldots, t_n : \alpha_n$ in left to right order. Types are sometimes omitted in case this causes no confusion.

Let $(\cdots(a\,t_1)\cdots t_n)$ be a term of a basic type. Then, it is denoted by $a(t_1,\ldots,t_n)$.

We will borrow from the λ-calculus the notions of α-equivalence, β-reduction and η-reduction. We do not distinguish terms that are α-equivalent. The term $t \equiv (\cdots((a\,t_1)t_2)\cdots t_n)$ is *η-expanded* to $\lambda x.(t\,x)$ if t is not of a basic type. We say t is *η-long β-normal form* (or *normalized*) if it is a normal form with respect to both β-reduction and η-expansion. We use $t{\downarrow}$ for the η-long β-normal form of t. The simply typed λ-calculus is confluent and terminating with respect to β-reduction, and with respect to β-reduction and η-expansion as well.

A substitution σ is a mapping $V \to T$ such that the type of $\sigma(X)$ is the same as the type of X. We define $Dom(\sigma) = \{X \mid X \not\equiv \sigma(X)\}$ and $Var(\sigma) = \bigcup_{X \in Dom(\sigma)} Var(\sigma(X))$. We sometimes use $[X_1 \mapsto t_1, \ldots, X_n \mapsto t_n]$ to denote a substitution σ such that $Dom(\sigma) = \{X_1, \ldots, X_n\}$ and $\sigma(X_i) \equiv t_i$ for all i. The restriction σ_Z of substitution σ for $Z \subseteq V$ is defined as follows:

$$\sigma_Z(X) \equiv \begin{cases} \sigma(X) & \text{if } X \in Z \\ X & \text{if } X \notin Z \end{cases}$$

We sometimes say σ is an extension of σ_Z. Any substitution σ is extended to a mapping $\overline{\sigma} : T \to T$ as follows:

$$\begin{array}{ll} \overline{\sigma}(X) & \equiv \sigma(X) \\ \overline{\sigma}(c) & \equiv c \\ \overline{\sigma}(s\,t) & \equiv (\overline{\sigma}(s)\,\overline{\sigma}(t)) \\ \overline{\sigma}(\lambda x.t) & \equiv \lambda x.(\overline{\sigma}_{Dom(\sigma)-\{x\}}(t)) \text{ if } x \notin Var(\sigma) \end{array}$$

Note that α-conversion of t is possibly needed before applying $\overline{\sigma}$ to t in case of $Var(\sigma) \cap BV(t) \neq \emptyset$. Instead of $\overline{\sigma}(t)$, we write $t\overline{\sigma}$ or even $t\sigma$ by identifying σ and $\overline{\sigma}$.

It is known that normalized terms are of the form $\lambda x_1 \cdots x_m.\ a(t_1, \ldots, t_n)$ for some $m, n \geq 0$, $a \in C \cup V$ and terms $t_1, \ldots, t_n$ in η-long β-normal form themselves. Note that $a(t_1, \ldots, t_n)$ is of basic type. For an η-long β-normal form t of the form $a(u_1, \ldots, u_n)$, we write $top(t)$ for a.

A *higher-order rewrite system* (HRS) is a finite set of rewrite rules $R = \{l_i \to r_i : \alpha_i\}$, where l_i and r_i are normalized terms having the same basic type α_i. Given an HRS R, a normalized term s is *reduced* to a term t, written $s \to_R t$, $s \to_{l \to r} t$ or simply $s \to t$, if $s \equiv C[l\sigma\downarrow]$ and $t \equiv C[r\sigma\downarrow]$ for some context $C[\,]$, substitution σ and rule $l \to r \in R$. If $C[\,] \equiv \Box_\beta$ for $s : \beta$, it is written $s \stackrel{\Lambda}{\to} t$; otherwise it is written $s \stackrel{>\Lambda}{\to} t$. Note that t is also normalized if $s \to t$ [9].

We denote by $\stackrel{*}{\to}$ the reflexive transitive closure of the reduction relation $\to$. If there is an infinite reduction sequence $v \equiv v_0 \to v_1 \to \cdots$ from v, we say v has an infinite reduction sequence; otherwise we say v is terminating. If there exists no v that has an infinite reduction sequence, we say $\to$ is *terminating*. We also say that an HRS R is terminating if $\to_R$ is terminating.

The strict part $\succ$ of a quasi-ordering $\succeq$ is defined as $s \succ t \iff s \succeq t \wedge t \not\succeq s$. We also write $s \sim t$ for $s \succeq t \wedge t \succeq s$. An ordering $\succ$ on T is said to be *well-founded* if it does not admit an infinite sequence $t_1 \succ t_2 \succ \cdots$ of elements t_1, t_2, $\ldots \in T$. A quasi-ordering $\succeq$ is *closed under substitutions* if $s \succeq t \Rightarrow s\sigma\downarrow \succeq t\sigma\downarrow$ and $s \succ t \Rightarrow s\sigma\downarrow \succ t\sigma\downarrow$ for all substitutions σ. A quasi-ordering $\succeq$ is *weakly closed under contexts* if $s \succeq t \Rightarrow f(\ldots, s, \ldots) \succeq f(\ldots, t, \ldots)$ for all function symbols f. A quasi-ordering is called a *reduction quasi-ordering* if it is well-founded, closed under substitutions and weakly closed under contexts.

3 Dependency Pairs of HRSs

We extend the notion of dependency pairs [1–3] for proving termination of TRSs to higher-order rewrite systems.

We use ordinary subterm relation, while the reference [10] a special subterm relation. For easy treatment of name collision between a free variable and a bound variable, we assume that each free variable originated from a bound variable is fresh.

Definition 1. *Let s be a normalized term. A term t is a subterm of s, denoted by $s \trianglerighteq t$, if*

(a) $s \equiv t$, *or*
(b) $s \equiv \lambda x.s'$ *and* $s'[x \mapsto X] \trianglerighteq t$ *where* X *is a fresh variable, or*
(c) $s \equiv a(u_1, \ldots, u_n)$ *and* $u_i \trianglerighteq t$ *for some* $i \in \{1, \ldots, n\}$.

We say t is a proper subterm of s, denoted by $s \triangleright t$, if $s \trianglerighteq t$ and $s \not\equiv t$.

The following proposition is obviously obtained.

Proposition 1. *Let R be an HRS. Then, $(\trianglerighteq \circ \to_R) \subseteq (\to_R \circ \trianglerighteq)$, where $\circ$ denotes the composition of relations.*

We say f is a defined symbol if $f = top(l)$ for some rule $l \to r$ and let $D = \{top(l) \mid l \to r \in R\}$ and $D^\# = \{f^\# \mid f \in D\}$ where $f^\#$ is a fresh symbol obtained by marking f in D. We define $s^\# \equiv f^\#(t_1, \ldots, t_n)$ if $s \equiv f(t_1, \ldots, t_n)$ and $f \in D$; otherwise $s^\# \equiv s$.

Dependency pairs and R-chain are defined the same as the first order case, while in the reference [10] a dependency pair of a rule $l \to r$ is $\langle l^\#, t^\# \rangle$, where t is a subterm of r such that $top(t) \in D \cup FV_h$.

Definition 2. *The set $DP_{l \to r}$ of dependency pairs of a rule $l \to r$ is defined as follows:*

$$DP_{l \to r} = \{\langle l^\#, t^\# \rangle \mid r \trianglerighteq t, top(t) \in D\}$$

DP_R denotes the collection of all dependency pairs of rules in HRS R.

Example 1. Consider the following HRS:

$$R_1 = \begin{cases} map(\lambda x.F(x), nil) \to nil, \\ map(\lambda x.F(x), cons(X, XS)) \\ \qquad \to cons(F(X), map(\lambda x.F(x), XS)) \end{cases}$$

Then, we have only one dependency pair:

$$\langle map^\#(\lambda x.F(x), cons(X, XS)),\ map^\#(\lambda x.F(x), XS) \rangle.$$

Definition 3. *Let $\langle s_1, t_1\rangle \cdots \langle s_n, t_n\rangle$ be a (possibly infinite) sequence of dependency pairs for an HRS R. It is called an R-chain if there exist substitutions $\sigma_1, \ldots, \sigma_n$ such that the following conditions holds for all $i = 1, \ldots, n-1$:*

$$top(t_i) \in D^{\#} \text{ and } t_i\sigma_i{\downarrow} \xrightarrow{*} s_{i+1}\sigma_{i+1}{\downarrow}$$

Note that we use a substitution σ_i for each dependency pair $\langle s_i, t_i\rangle$ in the definition of the R-chains, although the original definition uses only one substitution. The reason is only for presentation convenience.

Example 2. Consider the following HRS with $g, h, i \in C_{\alpha\to\alpha}$, $f \in C_{(\alpha\to\alpha)\to\alpha}$, $F \in V_{\alpha\to\alpha}$, $X \in V_\alpha$ and basic type α:

$$R_2 \;=\; \begin{cases} f(\lambda x.F(x)) \to F(a), \\ g(a) \to f(\lambda x.i(x)), \\ i(X) \to h(g(X)) \end{cases}$$

The dependency pairs are

$$\langle g^{\#}(a), f^{\#}(\lambda x.i(x))\rangle, \langle g^{\#}(a), i^{\#}(Y)\rangle, \langle i^{\#}(X), g^{\#}(X)\rangle.$$

We have an infinite reduction sequence:

$$g(a) \to_{R_2} f(\lambda x.i(x)) \to_{R_2} i(a) \to_{R_2} h(g(a)) \to_{R_2} h(f(\lambda x.i(x))) \to_{R_2} \cdots$$

and an infinite R_2-chain $\langle g^{\#}(a), i^{\#}(Y)\rangle\ \langle i^{\#}(X), g^{\#}(X)\rangle\ \langle g^{\#}(a), i^{\#}(Y)\rangle \cdots$ with $Y\sigma_1 \equiv a$, $X\sigma_2 \equiv a$, $Y\sigma_3 \equiv a, \cdots$.

We have to show how to construct an infinite R-chain from non-terminating HRS for soundness of the dependency pair method. It is not easy because Example 2 shows that the infinite reduction sequence does not correspond to the R_2-chain directly.

Definition 4. *A term u in η-long β-normal form is said to be essential if*

(a) u has an infinite reduction sequence, and
(b) any proper subterm of u has no infinite reduction sequence.

Note that the type of an essential term is basic since we assume the type of both sides of rewrite rules are basic. We also note that a term has at least one essential subterm if it has an infinite reduction sequence.

We say a substitution σ is *terminating*, if $F\sigma{\downarrow}$ is terminating for all variables F.

Lemma 1. *Let r and u be in η-long β-normal form and σ be a terminating substitution and $Var(\sigma_{Var(r)})$ and $BV(r)$ are disjoint. If u is essential and $r\sigma{\downarrow} \trianglerighteq u$, then the following (a) or (b) holds for some q such that $r \trianglerighteq q$:*

(a) $top(q) = top(u) \in D$ and $q\sigma'{\downarrow} \equiv u$ for some extension σ' of σ

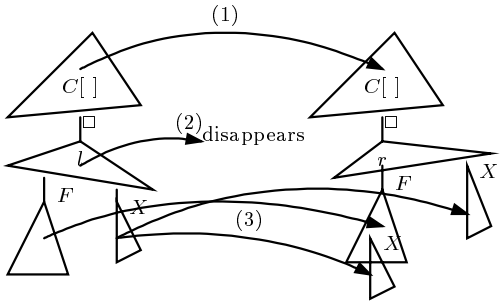

Fig. 1. A pictorial explanation of descendants

(b) top(q) is a higher-order variable, $q\sigma\downarrow \trianglerighteq u$ and $q'\sigma'\downarrow \not\equiv u$ for any proper subterm q' of q and extension σ' of σ.

Definition 5. *Consider the following infinite sequence:*

$$u_1 \overset{>\Lambda}{\to} \cdots \overset{>\Lambda}{\to} u_{k_1} \overset{\Lambda}{\to} v_1 \trianglerighteq u_{k_1+1} \overset{>\Lambda}{\to} \cdots \overset{>\Lambda}{\to} u_{k_2} \overset{\Lambda}{\to} v_2 \trianglerighteq \cdots,$$

where $0 \le k_1 < k_2 < \cdots$. We say it is essential, if u_i is essential for every i.

Proposition 2. *If an HRS R is not terminating, there exists an infinite essential sequence.*

Example 3. An infinite essential sequence of R_2 is

$$\begin{aligned}
&g(a) \overset{\Lambda}{\to}_{R_2} f(\lambda x.i(x)) \\
&\trianglerighteq f(\lambda x.i(x)) \overset{\Lambda}{\to}_{R_2} i(a) \\
&\trianglerighteq i(a) \overset{\Lambda}{\to}_{R_2} h(g(a)) \\
&\triangleright g(a) \overset{\Lambda}{\to}_{R_2} f(\lambda x.i(x)) \\
&\trianglerighteq f(\lambda x.i(x)) \overset{\Lambda}{\to}_{R_2} i(a) \\
&\quad\vdots
\end{aligned}$$

We need to introduce the notion of descendants (residuals)[6]. We will give an intuitive explanation of higher-order version of descendants. Let's consider a reduction $A : C[l\sigma\downarrow] \to C[r\sigma\downarrow]$. The descendants of a occurrence p in $C[l\sigma\downarrow]$ with respect to A are (1) p if p is in $C[\ ]$, (2) empty if p corresponds to the non-variable occurrence of l, (3) the occurrences corresponding to $F\sigma$ in $C[l\sigma\downarrow]$ if p is in $F\sigma$ for $Var(l)$ (Figure 1). This notions can be extended naturally to reduction sequences and essential sequences.

Given a infinite essential sequence

$$\begin{aligned}
&u_1 \overset{>\Lambda}{\to}_{l_1\to r_1} u_2 \overset{>\Lambda}{\to}_{l_2\to r_2} \cdots \overset{>\Lambda}{\to}_{l_{k_1-1}\to r_{k_1-1}} u_{k_1} \overset{\Lambda}{\to}_{l_{k_1}\to r_{k_1}} v_1 \\
&\trianglerighteq u_{k_1+1} \overset{>\Lambda}{\to}_{l_{k_1+1}\to r_{k_1+1}} u_{k_1+2} \overset{>\Lambda}{\to}_{l_{k_1+2}\to r_{k_1+2}} \cdots \overset{>\Lambda}{\to}_{l_{k_2-1}\to r_{k_2-1}} u_{k_2} \overset{\Lambda}{\to}_{l_{k_2}\to r_{k_2}} v_2 \\
&\quad\vdots,
\end{aligned}$$

we define a *dependency forest* $\langle N, E\rangle$ each of which nodes is a tuple of a natural number and a term and each of which edges is labeled by either a dependency pair or a substitution.

1. Let $N := \{\langle 1, t\rangle \mid u_1 \trianglerighteq t, top(t) \in D\}$ and $E := \emptyset$.
2. Do the following for each u_i in increasing order from $i = 1$:
   (a) In case of $u_i \equiv C[l_i\sigma\downarrow]$ and $u_{i+1} \equiv C[r_i\sigma\downarrow]$ for non-empty context $C[\ ]$, we can assume that $Dom(\sigma) = Var(l_i)$ without loss of generality. From essentiality the substitution σ is terminating. Add a node $\langle i+1, t\sigma\downarrow\rangle$ for each t such that $r_i \trianglerighteq t$ and $top(t) \in D$. Do the followings for each t:
   If there exists j such that $j < i$, the occurrence of $l_i\sigma\downarrow$ in u_i is a descendant of $t'\sigma'\downarrow$ in u_j where $r_{j-1} \trianglerighteq t'$ and $top(t') \in D$, then add an edge with the label $\langle l^\#, t^\#\rangle$ from the node $\langle j, t'\sigma'\downarrow\rangle$ to $\langle i+1, t\sigma\downarrow\rangle$. Otherwise, the occurrence of $l_i\sigma\downarrow$ in u_i is an descendant of t' in u_1 such that $top(t') \in D$. Then, add an edge with the label $\langle l^\#, t^\#\rangle$ from the node $\langle 1, t'\rangle$ to $\langle i+1, t\sigma\downarrow\rangle$.
   (b) In case of $u_i \equiv l_i\sigma\downarrow$, $v_m \equiv r_i\sigma\downarrow$ and $i = k_m$, the substitution σ is terminating. We can assume the domain of σ is $Var(l_i)$. By Lemma 1, either (a) or (b) of Lemma 1 holds for some q such that $r \trianglerighteq q$. Firstly, add a node $\langle i+1, t\sigma\downarrow\rangle$ for each t such that $q \trianglerighteq t$ and $top(t) \in D$. Also add an edge with the label $\langle l_i^\#, t^\#\rangle$ from the node $\langle i, u_i\rangle$ to $\langle i+1, t\sigma\downarrow\rangle$ for each t. Moreover, in case of (a) do the following.
   Add a node $\langle i+1, u_{i+1}\rangle$. If there exists j such that $j < i$ and the occurrence of u_{i+1} in v_l is an descendant of $t'\sigma'\downarrow$ in u_j where $r_{j-1} \trianglerighteq t'$ and $top(t') \in D$, then $t'\sigma'\downarrow\theta \equiv u_{i+1}$ for some θ whose domain is a subset of $BV(r_{j-1})$. Hence, add an edge with the label θ from the node $\langle j, t'\sigma'\downarrow\rangle$ to $\langle i+1, u_{i+1}\rangle$. Otherwise, the occurrence of u_{i+1} in v_l is an descendant of t' in u_1 where $top(t') \in D$ and $t'\theta \equiv u_{i+1}$ for some θ whose domain is a subset of $BV(u_1)$. Hence, add an edge with the label θ from the node $\langle 1, t'\rangle$ to $\langle i+1, u_{i+1}\rangle$.

Example 4. Consider the HRS R_2 in Example 2. The dependency forest of the infinite essential sequence in Example 3 is shown in Figure 2.

Lemma 2. *The root nodes of a dependency forest are* $\{\langle 1, t\rangle \mid \langle 1, t\rangle \in N\}$.

We say a term t is linear if every variable of t occurs only once. For example, linear terms are $f(X, Y)$, $f(X, \lambda yz.g(y, z))$, while non-linear terms are $f(X, X)$, $f(\lambda x.f(x, x))$. We say an HRS is non-duplicating if r is linear for every rule $l \to r$.

We say a term t is nested if there is a variable F in the η-normal form of t such that at least one of the argument of F contains a variable. For example, nested terms are $F(X)$, $f(\lambda yz.y(z))$, while non-nested terms are $f(F(a), X)$, $f(\lambda yz.g(y, z))$, $f(\lambda z.X(z))$. We say an HRS is non-nested if r is non-nested for every rule $l \to r$.

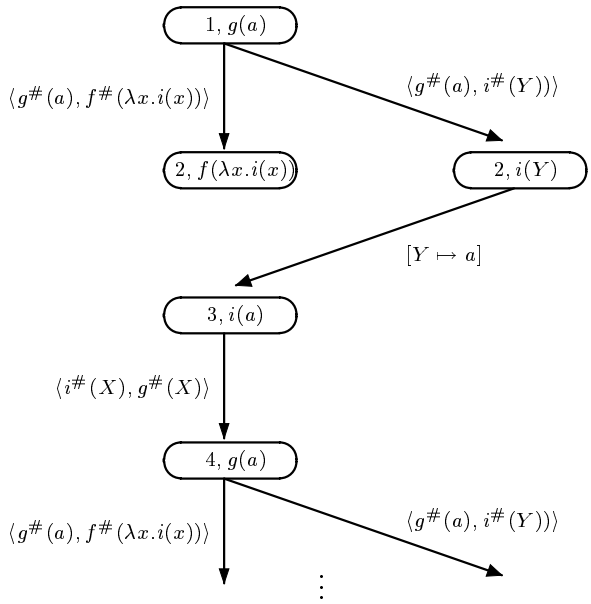

Fig. 2. The dependency forest of the infinite essential sequence in Example 3

Lemma 3. *Let R be a non-duplicating or non-nested HRS. For every infinite essential sequence, the number of reduced descendants of a occurrence are finite. Therefore, the dependency forest of the sequence is finite branching, that is, the number of children of every node is finite.*

Example 5. Consider the following non-duplicating HRS:

$$R_3 = \begin{cases} f(X, \lambda x.F(x)) \to F(X), \\ g(a,a) \to f(a, \lambda x.g(b,x)) \\ b \to a \end{cases}$$

and the infinite essential sequence

$$\begin{aligned} & f(b, \lambda x.g(x,x)) \to g(b,b) \\ \trianglerighteq\, & g(b,b) \to g(a,b) \to g(a,a) \to f(a, \lambda x.g(b,x)) \\ \trianglerighteq\, & f(a, \lambda x.g(b,x)) \to g(b,a) \\ \trianglerighteq\, & g(b,a) \to g(a,a) \to f(a, \lambda x.g(b,x)) \\ \trianglerighteq\, & f(a, \lambda x.g(b,x)) \to g(b,a) \\ & \vdots \end{aligned}$$

The number of reductions on the descendants of b is two, i.e., $g(b,b) \to g(a,b) \to g(a,a)$. The dependency forest of the essential sequence is shown in Figure 3.

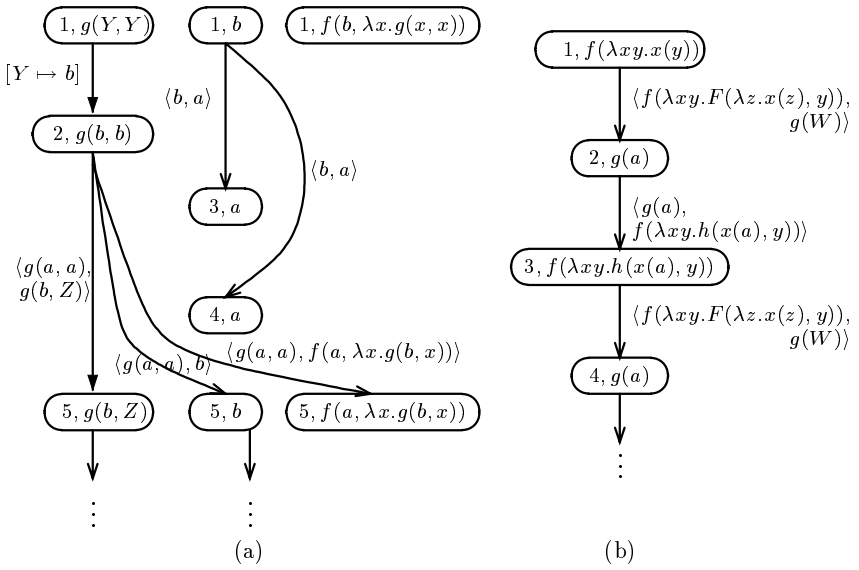

Fig. 3. The dependency forest of the infinite essential sequence in Example 5 and **??**

Example 6. Consider the following non-nested HRS:

$$R_4 \;=\; \begin{cases} f(\lambda xy.F(\lambda z.x(z), y)) \to h(F(\lambda z.g(z), a), F(\lambda z.g(z), a)) \\ g(a) \to f(\lambda xy.h(x(a), y)) \end{cases}$$

and the infinite essential sequence

$$\begin{aligned} & f(\lambda xy.x(y)) \to h(g(a), g(a)) \\ \trianglerighteq\, & g(a) \to f(\lambda xy.h(x(a), y)) \\ \trianglerighteq\, & f(\lambda xy.h(x(a), y)) \to h(h(g(a), a), h(g(a), a)) \\ \trianglerighteq\, & g(a) \to f(\lambda xy.h(x(a), y)) \\ & \vdots \end{aligned}$$

One duplication appears in this reduction sequence, i.e., only the first reduction duplicates the term a.

Theorem 1. *Let an HRS R be non-duplicating or non-nested. If there is no infinite R-chain, it is terminating.*

Proof. Assume R is not terminating. Then, we have an infinite essential sequence by Proposition 2. The dependency forest have finite root nodes and infinite nodes. Moreover, it is finite branching by Lemma 3. It follows from König's Theorem that there is an infinite path. From the construction of the forest, an infinite R-chain is obtained from the infinite path. □

Example 7. Consider the HRS R_2 in Example 2. We can construct the infinite R_2-chain

$$\langle g^\#(a), i^\#(Y)\rangle\langle i^\#(X), g^\#(X)\rangle\langle g^\#(a), i^\#(Y)\rangle\langle i^\#(X), g^\#(X)\rangle\cdots$$

shown in Example 2 from the infinite path

$$(1, g(a))\,(2, i(Y))\,(3, i(a))\,(3, g(a))\cdots$$

in the dependency forest in Figure 2.

Example 8. Consider the following duplicating HRS[10]:

$$R_5 \;=\; \{\; f(g(\lambda x.F(x))) \to F(g(\lambda x.h(F(x)))) \;\}$$

Although there is no dependency pair, it is not terminating, i.e., we have an infinite reduction sequence:

$$\begin{array}{l} f(g(\lambda x.f(x))) \\ \to_{R_2} f(g(\lambda x.h(f(x)))) \\ \to_{R_2} h(f(g(\lambda x.h(h(f(x)))))) \\ \to_{R_2} h(h(h(f(g(\lambda x.h(h(h(f(x)))))))))\\ \quad\vdots \end{array}$$

and the dependency forest for the essential sequence

$$\begin{array}{l} f(g(\lambda x.h(f(x)))) \stackrel{\Lambda}{\to} h(f(g(\lambda x.h(h(f(x)))))) \\ \trianglerighteq f(g(\lambda x.h(h(f(x))))) \stackrel{\Lambda}{\to} h(h(h(f(g(\lambda x.h(h(h(f(x)))))))))\\ \quad\vdots \end{array}$$

is shown in Figure 4.

4 Proving termination

We can apply the method similarly to the first-order case for proving termination of HRSs. While the reference [10] requires a reduction quasi-ordering satisfying the subterm property for proving termination, we do not need the subterm property anymore. This means that we can use the argument filtering method to construct the quasi-ordering.

Lemma 4. *Let R be an HRS. If there exists a reduction quasi-ordering $\succeq$ such that*

(a) $l \succeq r$ for all rules $l \to r \in R$, and
(b) $s \succ t$ for all dependency pairs $\langle s, t\rangle$,

then R has no infinite R-chain.

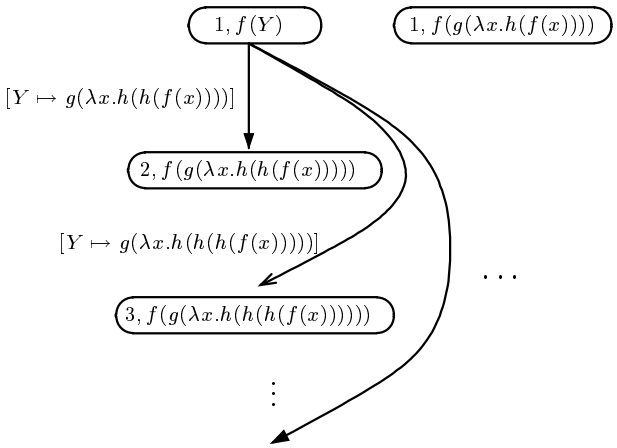

Fig. 4. The dependency forest of the infinite essential sequence in Example 8

Proof. Assume we have an infinite R-chain $\langle s_1, t_1\rangle\ \langle s_2, t_2\rangle\ \langle s_3, t_3\rangle \cdots$ from Theorem 1. Then there exist substitutions σ_1, σ_2, ... such that $t_i\sigma_i{\downarrow} \xrightarrow{*} s_{i+1}\sigma_{i+1}{\downarrow}$ for all i. We have $t_i\sigma_i{\downarrow} \succeq s_{i+1}\sigma_{i+1}{\downarrow}$ from Premise (a) and the closedness of $\succeq$ under substitutions. It follows from $s_i \succ t_i$ for all i that we have an infinite sequence $s_1\sigma_1{\downarrow} \succ s_2\sigma_2{\downarrow} \succ \cdots$, which is a contradiction. □

Example 9. Consider the following HRS R_5:

$$R_6 \ = \ \begin{cases} \mathit{composition}(\lambda x.F(x), \lambda y.G(y), Z) \to F(G(Z)), \\ \mathit{apply}(\lambda x.F(x), X) \to F(X) \end{cases}$$

Since R_6 is non-duplicating and has no dependency pair, it is terminating.

5 Discussion

By extending the dependency pair approach to the higher-order setting, one can benefit from the following features of dependency pairs:

- One need not include any subterm of right hand side whose top symbol is higher order variable to dependency pairs.
- One can make a difference between usual function symbols and marked function symbols.
- One can strip off context consists of constructors and higher-order variables around defined symbols when building dependency pairs.

Probably, combining the following method with dependency method gives more power to proving termination, although we did not show the fact in this paper;

- Argument filtering will be applicable and helpful, because we can eliminate not only certain arguments of function symbols but also certain arguments of higher order variables.
- The dependency graph refinement will be helpful in the higher-order case as well to determine that the application of certain reduction steps never leads to an infinite reduction.

However, the completeness does not satisfy any longer. Moreover, unfortunately, HRSs should be non-duplicating or non-nested to apply this method. It is strongly desirable to find weaker condition.

Acknowledgment

We thank Prof. Toshiki Sakabe for suggestions. This work is partly supported by Grants from Ministry of Education, Science and Culture of Japan #11680352 .

References

1. T. Arts and J. Giesl, Automatically proving termination where simplification orderings fail, In Proc. 22nd International Colloquium on Trees in Algebra and Programming, CAAP'97, In LNCS, vol.1214, pp.261–272, Springer-Verlag, 1997.
2. T. Arts, Automatically proving termination and innermost normalization of term rewriting system Ph.D. thesis, Utrecht University, The Netherlands, 1997.
3. T. Arts and J. Giesl, Termination of term rewriting using dependency pairs, Theoretical Computer Science, Vol.236, pp.133–178, 2000.
4. H. Barendregt, Lambda calculi with types, in Handbook of Logic in Computer Science, ed.Abramsky el al., Oxford University Press, 1993.
5. F. Baader and T. Nipkow, Term Rewriting and All That, Cambridge University Press, 1998.
6. G. Huet and J.-J. Lévy, Computations in Orthogonal Rewriting Systems, I and II, in *Computational Logic, Essays in Honor of Alan Robinson*, (The MIT Press, 1991) 396–443.
7. M. Iwami, M. Sakai, and Y. Toyama, An improved recursive decomposition ordering for higher-order rewrite systems, IEICE Trans. Inf. & Syst., Vol.E-81-D, pp.988–996, 1998.
8. M. Iwami and Y. Toyama, Simplification ordering for higher-order rewrite systems, Technical Report IS-RR-98-0024F, JAIST, 1998.
9. J.-P. Jouannaud and A. Rubio, Rewrite orderings for higher-order terms in η-long β-normal form and the recursive path ordering, Theoretical Computer Science, Vol.208, pp.33–58, 1998.
10. M. Sakai and Y. Watanabe and T. Sakabe, An Extension of Dependency Pair Method for Proving Termination of Higher-Order Rewrite Systems, IEICE Trans. on Information and Systems, Vol.E84-D, No.8, pp.1025–1032, 2001.
11. O. Lysne and J. Piris, A termination ordering for higher order rewrite systems, Proc. 6th International Conference on Rewriting Techniques and Applications, RTA'95, in LNCS, Vol.914, pp.26–40, Springer-Verlag, 1995.

12. C. Loría-Sáenz and J. Steinbach, Termination of combined (rewrite and λ-calculus) systems, Proc. 3rd International Workshop on Conditional Term Rewriting Systems, CTRS'92, in LNCS, Vol.656, pp.143–147, Springer-Verlag, 1993.
13. T. Nipkow, Higher-order critical pairs, Proc. 6th annual IEEE Symposium on Logic in Computer Science, pp.342–349, 1991.