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

Masahiko Sakai[1] and Keiichirou Kusakari[1]

Graduate School of Information Science, Nagoya University.

Abstract. This paper explores how to extend the dependency pair technique for proving termination of higher-order rewrite systems. In the first order case, the termination of term rewriting systems are proved by showing the non-existence of an infinite R-chain of the dependency pairs. However, the termination and the non-existence of an infinite R-chain do not coincide in the higher-order case. We introduce a new notion of dependency forest that characterize infinite reductions and infinite R-chains, and show that the termination property of higher-order rewrite systems R can be checked by showing the non-existence of an infinite R-chain, if R is strongly linear or non-nested.

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 properties of term rewriting systems [18, 17, 11, 9, 10]. On the other hand, in order to prove the termination of typed λ-calculus, the notion of computability was introduced by Tait [22] and Girard [7]. Based on computability instead of simplification orders, Jouannaud and Rubio [12] and Raamsdonk [20] introduced recursive path orders in higher-order rewriting systems.

The dependency pair technique [2–4] has been developed for proving termination of term rewriting systems. It is useful because it gives us a mechanical support for proving non-simple termination by using known reduction orderings to show simple termination. For example, consider the following term rewriting systems that is not simple terminating:

$$R_1 = \begin{cases} f(X, s(Y)) \rightarrow f(X, Y) \\ g(s(X), Y) \rightarrow g(f(X, Y), Y). \end{cases}$$

where the capital letters indicate free variables. By the dependency pair technique, the termination is shown by finding a reduction quasi-ordering $\succeq$ satisfying the following constraints:

$$\begin{array}{l} f(X,s(Y)) \succeq f(X,Y) \\ g(s(X),Y) \succeq g(f(X,Y),Y) \\ f^{\#}(X,s(Y)) \succ f^{\#}(X,Y) \\ g^{\#}(s(X),Y) \succ g^{\#}(f(X,Y),Y) \\ g^{\#}(s(X),Y) \succ f^{\#}(X,Y), \end{array}$$

where $f^{\#}$ and $g^{\#}$ are freshly introduced function symbols. The great point is that the ordering $\succ$ need not be monotonic:

$$s \succ t \Rightarrow f(\ldots,s,\ldots) \succ f(\ldots,t,\ldots)$$

Hence, it is enough for proving the termination to find a reduction quasi-ordering $\succeq'$ satisfying the following constraints obtained by replacing $f(t,u)$ by t, called an argument filtering method.

$$\begin{array}{l} X \succeq' X \\ g(s(X),Y) \succeq' g(X,Y) \\ f^{\#}(X,s(Y)) \succ' f^{\#}(X,Y) \\ g^{\#}(s(X),Y) \succ' g^{\#}(X,Y) \\ g^{\#}(s(X),Y) \succ' f^{\#}(X,Y). \end{array}$$

Kusakari extended the dependency pair method to higher-order systems that do not support λ-abstraction [14, 15]. For the higher-order system including λ-abstraction introduced by Nipkow [19], Sakai, Watanabe and Sakabe studied how to apply the dependency pair method, and clarified an essential difficulty when the system has λ-abstraction [21]. The difficulty is that the ordering must have the subterm property:

$$C[s] \succeq s \text{ for any term } s \text{ and context } C.$$

This means that we cannot use the powerful technique, the argument filtering method, which is designed by eliminating unnecessary subterms. For example, in order to show the termination of the following higher-order rewriting system

$$R_2 = \begin{cases} f(\lambda x.F(x), s(X)) \\ \qquad \to f(\lambda x.F(a), f(\lambda x.F(x), X)), \end{cases}$$

we must find a reduction quasi-ordering $\succeq$ having subterm property satisfying the following constraints:

$$\begin{array}{l} f(\lambda x.F(x), s(X)) \succeq f(\lambda x.F(a), f(\lambda x.F(x), X)) \\ f^{\#}(\lambda x.F(x), s(X)) \succ f^{\#}(\lambda x.F(a), f(\lambda x.F(x), X)) \\ f^{\#}(\lambda x.F(x), s(X)) \succ f^{\#}(\lambda x.F(x), X) \\ f^{\#}(\lambda x.F(x), s(X)) \succ F(a) \\ f^{\#}(\lambda x.F(x), s(X)) \succ F(c_x) \\ f(t,u) \succeq f^{\#}(t,u) \text{ for all terms } t \text{ and } u, \end{array}$$

where c_x is an fresh constant symbol. Unfortunately, we cannot use the argument filtering method because the argument filtering breaks the subterm property. Thus, we fail to proof the termination of R_2.

This paper introduces the notion of dependency forest and try to remove the requirement of subterm property for the quasi-ordering explained above. In results, we give a theorem that characterize the condition in which the dependency pair technique works without requiring the subterm property. We also give two kind of sufficient conditions, strongly linear and non-nested.

2 Preliminary Concepts

We assume the readers are familiar with the basic concepts and notations of term rewriting systems [6] and typed lambda calculus [5].

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

$$\begin{aligned} &\tau_S \supseteq S \\ &\tau_S \supseteq \{\alpha \to \beta \mid \alpha, \beta \in \tau_S\} \end{aligned}$$

Types that are not basic are called *higher-order types*. We use α, β to denote types.

Let V_α be a set of *variables* of a type α and $V = \bigcup_{\alpha \in \tau_S} V_\alpha$. Let $\mathcal{F}_\alpha$ be a set of *constants* (or *function symbols*) of a type α and $\mathcal{F} = \bigcup_{\alpha \in \tau_S} \mathcal{F}_\alpha$. We assume $V \cap \mathcal{F} = \emptyset$, and $V_\alpha \cap V_\beta = \emptyset$ and $\mathcal{F}_\alpha \cap \mathcal{F}_\beta = \emptyset$ 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 for a constant or a variable.

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

$$\begin{aligned} &T_\alpha \supseteq V_\alpha \cup \mathcal{F}_\alpha \\ &T_\alpha \supseteq \{(st) \mid s \in T_{\alpha' \to \alpha}, t \in T_{\alpha'}\} \\ &T_\alpha \supseteq \{(\lambda x.s) \mid x \in V_{\beta'}, s \in T_\beta, \alpha = \beta' \to \beta\} \end{aligned}$$

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 a *term*. We use l, r, s, t, u, v and w for terms. We use $FV(t)$ for the set of free variables in t and $BV(t)$ for the set of bound variables in t. Let $Var(t) = FV(t) \cup BV(t)$. We say t is *closed* if it contains no free variables. We assume for convenience that bound variables in a term are all different, and are disjoint from free variables. We use F, G, H, L, X, and Y for free variables and x, y and z for bound variables unless it is known to be free or bound from other conditions.

A term containing a special constant $\Box_\alpha$ of basic type α is called a *context* denoted by $C_\alpha[\,]$. We use $C_\alpha[t]$ for the term obtained from $C_\alpha[\,]$ by replacing $\Box_\alpha$ with $t : \alpha$. Types are sometimes omitted in case this causes no confusion.

We will borrow from the λ-calculus the notions of α-equivalence, β-reduction and η-reduction. We use $\equiv$ to denote α-equality on terms. The term $C[t] \equiv C[(\cdots((a\,t_1)t_2)\cdots t_n)]$ is *η-expanded* to $C[\lambda x.(t\,x)]$ if t is not of basic types and it creates no β-redexes. 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. It is known that every term has a unique normalized term [1].

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 a 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. We use $\overline{Z}$ for the complement $V - Z$ of Z. For any substitution σ, the mapping $\tilde{\sigma} : T \to T$ is defined as follows:

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

Note that the α-conversion of t is possibly needed before applying $\tilde{\sigma}$ to t in case of $Var(\sigma) \cap BV(t) \neq \emptyset$. Instead of $\tilde{\sigma}(t)$, we write $t\tilde{\sigma}$ or even $t\sigma$ by identifying σ and $\tilde{\sigma}$. A substitution σ is said to be *normalized* if $X\sigma$ is a normalized term for all $X \in Dom(\sigma)$.

Every normalized term can be represented by the form $\lambda x_1 \cdots x_m. (\cdots(at_1)\cdots t_n)$ where $m, n \geq 0$, $a \in \mathcal{F} \cup V$ and $(\cdots(at_1)\cdots t_n)$ is of basic types. In this paper, we represent this term t by $\lambda x_1 \cdots x_m.a(t_1, \ldots, t_n)$. The *top symbol* of t is defined as $top(t) = a$.

Let t be a normalized term. We say t is a *pattern* if $top(t) \in \mathcal{F}$, free variables in t are linear and the η-normal forms of $u_1, \ldots, u_n$ are different bound variables for any subterm $F(u_1, \ldots, u_n)$ of t such that $F \in FV(t)$ [16]. Let α be a basic type, $l : \alpha$ be a pattern and $r : \alpha$ be a normalized term such that $FV(l) \supseteq FV(r)$. Then, $l \to r$ is called a *higher-order rewrite rule (with type α)*. A *higher-order rewrite system* (HRS) is a finite set of higher-order rewrite rules. Given an HRS R, a normalized term s is *reduced* to a term t, written $s \to_R t$, $s \to_{l \to r, \sigma} t$ 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 $C[\;] \equiv \Box$, it is written $s \xrightarrow{\Lambda} t$; otherwise it is written $s \xrightarrow{>\Lambda} t$. Note that t is also normalized if $s \to t$ [19].

We denote by $\xrightarrow{*}$ the reflexive transitive closure of a relation $\to$. If there is no infinite sequence $v \equiv v_0 \to v_1 \to \cdots$ from v, we say v is *terminating* (with respect to $\to$). If every v is terminating with respect to $\to$, 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 monotone* 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 monotone. Note that $\succeq$ always needs to be preserved under α-conversion since we do not distinguish α-equivalent terms

3 Dependency Pair and Forest

We extend the notion of dependency pairs [2–4] for proving termination of TRSs to higher-order rewrite systems.

We use the ordinary subterm relation, while the reference [21] uses a special subterm relation [1] . For easy treatment against the name collision between a free variable and a bound variable, we assume that each free variable originated from a bound variable is fresh. For example, $f(Y, Y)$ is a subterm of $\lambda x.f(x, x)$.

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 $s \not\equiv t$.

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 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 [21] 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.

[1] In the reference [21], $f(c_x, c_x)$ is a subterm of $\lambda x.\ f(x, x)$, where c_x is a fresh constant.

Example 1. Consider the following HRS:

$$R_3 = \begin{cases} map(\lambda x.F(x), nil) \to nil, \\ map(\lambda x.F(x), cons(X, L)) \\ \qquad \to cons(F(X), map(\lambda x.F(x), L)) \end{cases}$$

Then, we have only one dependency pair:

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

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 $t_i\sigma_i\!\downarrow \xrightarrow{*} s_{i+1}\sigma_{i+1}\!\downarrow$ holds for all $i = 1, \ldots, n-1$.*

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 $g, h, i \in \mathcal{F}_{\alpha\to\alpha}$, $f \in \mathcal{F}_{(\alpha\to\alpha)\to\alpha}$, $F \in V_{\alpha\to\alpha}$, $X \in V_\alpha$ and basic type α:

$$R_4 \ = \ \begin{cases} f(\lambda x.F(x)) \to F(a), \\ g(a) \to f(\lambda x.i(x)), \\ i(X) \to h(g(X)). \end{cases}$$

We have three dependency pairs $\langle g^{\#}(a), f^{\#}(\lambda x.i(x))\rangle$, $\langle g^{\#}(a), i^{\#}(Y)\rangle$, $\langle i^{\#}(X), g^{\#}(X)\rangle$. We have an infinite reduction sequence:

$$\begin{aligned} g(a) &\to_{R_4} f(\lambda x.i(x)) \to_{R_4} i(a) \to_{R_4} h(g(a)) \\ &\to_{R_4} h(f(\lambda x.i(x))) \to_{R_4} \cdots \end{aligned}$$

and an infinite R_4-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 an infinite reduction sequence for soundness of the dependency pair method. However, the construction method of the first order case is not applicable to the infinite reduction sequence in Example 2.

Definition 4. *A non-terminating term u in η-long β-normal form is said to be minimal if any proper subterm of u is terminating.*

Note that minimal non-terminating terms are with a basic type since the types of rewrite rules are basic. We also note that a term has at least one minimal non-terminating subterm if it is not terminating.

We say a substitution σ is *terminating*, if $F\sigma\!\downarrow$ is terminating for any variables F.

Lemma 1. *Let r and u be normalized terms and σ be a terminating substitution such that $Var(\sigma_{Var(r)})$ and $BV(r)$ are disjoint. If u is minimal non-terminating and $r\sigma\!\downarrow\, \trianglerighteq u$, then the following (a) or (b) holds for some v such that $r \trianglerighteq v$:*

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

Definition 5. *Consider the following infinite sequence:*

$$\begin{aligned}
&u_1 \stackrel{>\Lambda}{\to} \cdots \stackrel{>\Lambda}{\to} u_{k_1} \stackrel{\Lambda}{\to} v_1 \\
&\trianglerighteq u_{k_1+1} \stackrel{>\Lambda}{\to} \cdots \stackrel{>\Lambda}{\to} u_{k_2} \stackrel{\Lambda}{\to} v_2 \\
&\trianglerighteq u_{k_2+1} \cdots ,
\end{aligned}$$

where $0 \leq k_1 < k_2 < \cdots$. We say the infinite sequence is minimal non-terminating, if u_i is minimal non-terminating for every i.

Proposition 1. *If an HRS R is not terminating, there exists a minimal non-terminating sequence.*

Example 3. A minimal non-terminating sequence of R_4 is

$$\begin{aligned}
&g(a) \stackrel{\Lambda}{\to}_{R_4} f(\lambda x.i(x)) \\
&\trianglerighteq f(\lambda x.i(x)) \stackrel{\Lambda}{\to}_{R_4} i(a) \\
&\trianglerighteq i(a) \stackrel{\Lambda}{\to}_{R_4} h(g(a)) \\
&\triangleright\, g(a) \stackrel{\Lambda}{\to}_{R_4} f(\lambda x.i(x)) \\
&\trianglerighteq f(\lambda x.i(x)) \stackrel{\Lambda}{\to}_{R_4} i(a) \\
&\quad \vdots
\end{aligned}$$

We need to introduce the notion of descendants (residuals) that traces the redex occurrences[8, 13]. Here, we will give an intuitive explanation of higher-order version of descendants in order to concentrate on the essence of the dependency forest. The precise definition[13] is found in Sect. 4. Let's consider a reduction $A : C[l\sigma\!\downarrow] \to C[r\sigma\!\downarrow]$. The *descendants* of an occurrence p in $C[l\sigma\!\downarrow]$ with respect to A are

(1) p if p is in $C[\ ]$,
(2) none if p corresponds to the non-variable occurrence of l,
(3) the occurrences corresponding to $F\sigma$ in $C[r\sigma\!\downarrow]$ if p is in $F\sigma$ for $Var(l)$

as shown in Fig. 1. This notions can be extended naturally to reduction sequences and minimal non-terminating sequences.

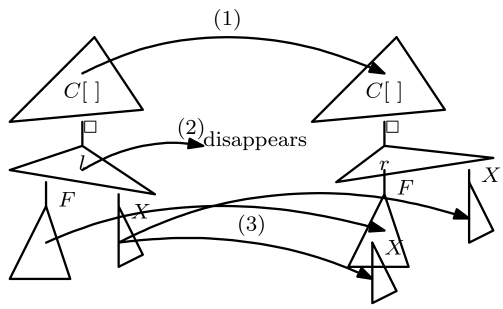

Fig. 1. Descendants

Definition 6. *Given a minimal non-terminating sequence:*

$$
\begin{aligned}
& u_1 \overset{>\Lambda}{\rightarrow}_{e_1,\sigma_1} \cdots \overset{>\Lambda}{\rightarrow}_{e_{k_1-1},\sigma_{k_1-1}} u_{k_1} \\
& \overset{\Lambda}{\rightarrow}_{e_{k_1},\sigma_{k_1}} v_1 \\
& \trianglerighteq u_{k_1+1} \overset{>\Lambda}{\rightarrow}_{e_{k_1+1},\sigma_{k_1+1}} \cdots \overset{>\Lambda}{\rightarrow}_{e_{k_2-1},\sigma_{k_2-1}} u_{k_2} \\
& \overset{\Lambda}{\rightarrow}_{e_{k_2},\sigma_{k_2}} v_2 \\
& \quad \vdots,
\end{aligned}
$$

where each e_i denotes a rule $l_i \rightarrow r_i$ and we can assume that $Dom(\sigma_i) = FV(l_i)$ without loss of generality. We define a dependency forest $\langle N, E\rangle$, where N is a node set whose elements are triples of a natural number, a term and a flag (Λ or $>\Lambda$), and E is a set of edges labeled by either a dependency pair or a substitution.
(Step 1) Let $N := \{\langle 1, t, p\rangle \mid u_1 \trianglerighteq t, top(t) \in D\}$ and $E := \emptyset$, where p is Λ if $u_1 \equiv t$; otherwise p is $>\Lambda$.
(Step 2) Do the following for each i in increasing order from 1:

i) In case of $u_i \equiv C[l_i\sigma_i\!\downarrow] \overset{>\Lambda}{\rightarrow} u_{i+1} \equiv C[r_i\sigma_i\!\downarrow]$, do the followings for each dependency pair $\langle l_i^{\#}, t^{\#}\rangle$:
 - *Add a node $\langle i+1, t\sigma_i\!\downarrow, >\Lambda\rangle$ and an edge with the label $\langle l_i^{\#}, t^{\#}\rangle$ from the node $\langle j+1, t', >\Lambda\rangle$ to $\langle i+1, t\sigma\!\downarrow, >\Lambda\rangle$, where $\langle j+1, t', >\Lambda\rangle$ is a node for the greatest j such that $j < i$ and the occurrence of $l_i\sigma_i\!\downarrow$ in u_i is a descendant of t' in u_{j+1}.*

ii) In case of $u_i \equiv l_i\sigma_i\!\downarrow \overset{\Lambda}{\rightarrow} v_m \equiv r_i\sigma_i\!\downarrow$ and $i = k_m$, either (a) or (b) of Lemma 1 holds for some w such that $r_i \trianglerighteq w$ by Lemma 1.
 - *ii-i) For the case (a), add a node $\langle i+1, t\sigma_i\!\downarrow, p\rangle$ and an edge with the label $\langle l_i^{\#}, t^{\#}\rangle$ from the node $\langle k_{m-1}+1, u_{k_{m-1}+1}, \Lambda\rangle$ to $\langle i+1, t\sigma_i\!\downarrow, p\rangle$ for each dependency pair $\langle l_i^{\#}, t^{\#}\rangle$ such that $w \trianglerighteq t$, where $k_0 = 0$, and p is Λ if $w \equiv t$; otherwise p is $>\Lambda$.*

ii-ii) *For the case (b), add a node $\langle i+1, u_{i+1}, \Lambda\rangle$ and an edge with the label θ from the node $\langle j, t', {>}\Lambda\rangle$ to $\langle i+1, u_{i+1}, \Lambda\rangle$, where j is the greatest number such that $j < i$ and the occurrence ε of u_{i+1} is an descendant of t' in u_j and θ is a substitution such that $t'\theta{\downarrow} \equiv u_{i+1}$.*

(Step 3) For every node O such that the flag part of O is ${>}\Lambda$ and any flag parts of reachable nodes from O are ${>}\Lambda$, remove O and edges connected to O.

Lemma 2. *The first item of any root nodes of dependency forests is 1.*

Note that we have infinite nodes having flag Λ. Hence, dependency forests still have infinite nodes after nodes removal in Step 3 of the definition.

If there exists an infinite path in a dependency forest, we can construct a sequence of dependency pairs from the path. Then, we can show that the sequence is R-chain from the construction of the dependency forest. Moreover, the R-chain is infinite because we have no successive edges labeled by a substitution in every path.

Example 4. Consider the HRS R_4 in Example 2. The dependency forest of the minimal non-terminating sequence in Example 3 is shown in Fig. 2(a), where nodes whose third items are Λ and $>\Lambda$ are drawn by solid lines and dashed lines, respectively. From the labels of the infinite path

$$\langle 1, g(a), \Lambda\rangle\ \langle 2, i(Y), {>}\Lambda\rangle\ \langle 3, i(a), \Lambda\rangle\ \langle 4, g(a), \Lambda\rangle\ \cdots$$

in the dependency forest, we can construct the following infinite R_4-chain:

$$\begin{array}{l}\langle g^{\#}(a), i^{\#}(Y)\rangle\\ \langle i^{\#}(X), g^{\#}(X)\rangle\\ \langle g^{\#}(a), i^{\#}(Y)\rangle\\ \langle i^{\#}(X), g^{\#}(X)\rangle\\ \qquad\vdots\end{array}$$

The following example shows that the necessity of the case i) of Step 2 in the definition of dependency forests.

Example 5. Consider the R_4 in Example 2. We have the following minimal non-terminating sequence different from that in Example 3.

$$\begin{array}{l} g(a) \xrightarrow{\Lambda}_{R_4} f(\lambda x.i(x)) \\ \trianglerighteq f(\lambda x.i(x)) \xrightarrow{>\Lambda}_{R_4} f(\lambda x.h(g(x))) \xrightarrow{\Lambda}_{R_4} h(g(a)) \\ \triangleright\ g(a) \xrightarrow{\Lambda}_{R_4} f(\lambda x.i(x)) \\ \qquad\vdots \end{array}$$

The dependency forest of this sequence is shown in Fig. 2(b). The infinite R-chain constructed from the dependency forest is the same as that in Example 4.

If we remove the case i) of Step 2 from the definition of dependency forests, the node $\langle 3, g(Y), {>}\Lambda\rangle$ disappears and we have no infinite path.

The following is a characterization lemma.

Lemma 3. *Let R be an HRS in the class that dependency forests are finite branching. Then, the non-existence of infinite R-chains implies the termination of R.*

Proof. Assume R is not terminating. Then, we have a minimal non-terminating sequence by Proposition 1. The dependency forest has infinite nodes with finite root nodes by Lemma 2. Since it is also finite branching, we have an infinite path from König's Lemma that finite branching infinite tress have an infinite path. From the construction of the forest, an infinite R-chain is obtained from the infinite path. $\square$

4 Finite-Branchingness of Dependency Forest

In this section, we show sufficient conditions that guarantee the finite branchingness of the dependency forests.

4.1 Descendant

This subsection shows the precise definition of the descendants developed in [13].

The occurrence of a normalized term is based on the form of $\lambda x_1 \cdots x_m.a(t_1, \ldots, t_n)$. In order to simplify the definition of descendants, the same representation of occurrence is assigned to $\lambda x.t$ and t in a term $\cdots \lambda x.t \cdots$. In this section, we abbreviate $\lambda x_1 \cdots x_m$ as $\lambda \boldsymbol{x}$. An *occurrence* of a normalized term is a sequence of natural numbers. We use p and q for occurrences. The set of occurrences of $t \equiv \lambda \boldsymbol{x}.a(t_1, \ldots, t_n)$ is defined by $Occ(t) = \{\varepsilon\} \cup \{ip \mid 1 \leq i \leq n, p \in Occ(t_i)\}$. Let p and q be occurrences. We write $p \leq q$ if $pp' = q$ for some occurrence p'. Moreover we write $p < q$ if $p' \neq \varepsilon$. We say p and q are *disjoint* if $p \not\leq q$ and $p \not\geq q$. The *subterm at the occurrence* p is represented as follows:

$$(\lambda \boldsymbol{x}.a(t_1, \ldots, t_n))|_p \equiv \begin{cases} a(t_1, \ldots, t_n) & \text{if } p = \varepsilon \\ t_i|_q & \text{if } p = iq. \end{cases}$$

$Occ_V(t)$ indicates the set of occurrences $p \in Occ(t)$ such that $top(t|_p)$ is a free variable in a normalized term t. $t[u]_p$ represents the term obtained from a normalized term t by replacing $t|_p$ with a normalized term u having the same basic type as $t|_p$. This is defined as follows:

$$\begin{aligned} &(\lambda \boldsymbol{x}.a(t_1, \ldots, t_n))[u]_p \\ &\equiv \begin{cases} \lambda \boldsymbol{x}.u & \text{if } p = \varepsilon \\ \lambda \boldsymbol{x}.a(\ldots, t_i[u]_q, \ldots) & \text{if } p = iq. \end{cases} \end{aligned}$$

In the following, we sometimes refer the reduction sequence $A : t_0 \to t_1 \to \cdots \to t_n$ by the attached label A. The definition of descendants of redexes are complicated because the occurrences of redexes move considerably by β-reductions taken in the reduction as the following example shows.

Example 6. Consider the following HRS R_8,

$$R_8 = \begin{cases} apply(\lambda x.F(x), X) \to F(X) \\ \qquad\qquad a \to b, \end{cases}$$

and a reduction $A_1 : t \equiv apply(\lambda x.f(g(x), x), a) \to f(g(a), a) \equiv s$. The descendants of a redex a on occurrence 2 of t are occurrences 2 and 11 of s as shown in Fig. 3.

In order to follow the occurrences of redexes correctly, the mutually recursive functions PV and PT is used, each of which returns a set of occurrences. The function $PV(t, \sigma, F, p)$ returns the set of the corresponding occurrences of $t\sigma\!\downarrow$ to $(F\sigma)|_p$. The function $PT(t, \sigma, p)$ returns the set of the corresponding occurrences of $t\sigma\!\downarrow$ to $t|_p$. In the previous example, we have $PV(F(X), \sigma, X, \varepsilon) = \{11, 2\}$ where $\sigma = \{F \mapsto \lambda x.f(g(x), x), X \mapsto a\}$. This shows that occurrences of a introduced by σ appears on the occurrences 11 and 2 of $F(X)\sigma\!\downarrow = f(g(a), a)$.

Definition 7. *Let t be a normalized term, σ be a normalized substitution and F be a variable. The function PV is defined as follows for an occurrence $p \in Occ(F\sigma)$.*

$$PV(t, \sigma, F, p) =$$
$$\begin{cases}
\{p\} & \mathit{if}\ t \equiv F & (1) \\
\bigcup_i \{iq \mid q \in PV(t_i, \sigma, F, p)\} & & (2) \\
 & \mathit{if}\ t \equiv a(t_1, \ldots, t_n),\ n > 0\ \mathit{and} & \\
 & \quad a \in \mathcal{F} \cup \overline{Dom(\sigma)} & \\
PV(t', \sigma_{\overline{\{x_1, \ldots, x_n\}}}, F, p) & & (3) \\
 & \mathit{if}\ t \equiv \lambda x_1 \cdots x_n.t',\ n > 0\ \mathit{and} & \\
 & \quad F \notin \{x_1 \cdots x_n\} & \\
\bigcup_i PV(t', \sigma', y_i, PV(t_i, \sigma, F, p)) & & (4) \\
 & \mathit{if}\ t \equiv G(t_1, \ldots, t_n),\ n > 0, & \\
 & \quad G \in Dom(\sigma)\ \mathit{and}\ F \neq G & \\
 & \mathit{where}\ G\sigma \equiv \lambda y_1 \ldots y_n.t' & \\
 & \mathit{s.t.}\ \sigma' = \{y_1 \mapsto t_1\sigma\!\downarrow, \ldots, y_n \mapsto t_n\sigma\!\downarrow\} & \\
(\bigcup_i PV(t', \sigma', y_i, PV(t_i, \sigma, F, p))) \cup PT(t', \sigma', p) & & (5) \\
 & \mathit{if}\ t \equiv F(t_1, \ldots, t_n),\ n > 0\ \mathit{and} & \\
 & \quad F \in Dom(\sigma) & \\
 & \mathit{where}\ F\sigma \equiv \lambda y_1 \ldots y_n.t' & \\
 & \mathit{s.t.}\ \sigma' = \{y_1 \mapsto t_1\sigma\!\downarrow, \ldots, y_n \mapsto t_n\sigma\!\downarrow\} & \\
\emptyset & \mathit{if}\ t \equiv G \neq F\ \mathit{or}\ t \in \mathcal{F} & (6)
\end{cases}$$

where $PV(t, \sigma, F, P)$ denotes $\bigcup_{p \in P} PV(t, \sigma, F, p)$ for a set P of occurrences.

Definition 8. *Let t be a normalized term, σ be a normalized substitution. The function PT is defined as follows for an occurrence $p \in Occ(t)$.*

$$PT(t,\sigma,p) = \begin{cases} \{\varepsilon\} & \textit{if } p = \varepsilon \quad (1) \\ \{iq \mid q \in PT(t_i,\sigma,p')\} & (2) \\ & \textit{if } p = ip', t \equiv a(t_1,\ldots,t_n),\ n > 0 \\ & \quad \textit{and } a \in \mathcal{F} \cup \overline{Dom(\sigma)} \\ PT(t',\sigma_{\overline{\{x_1,\ldots,x_n\}}},p) & (3) \\ & \textit{if } p \neq \varepsilon, t \equiv \lambda x_1 \cdots x_n.t' \textit{ and } n > 0 \\ PV(t',\sigma',y_i,PT(t_i,\sigma,p')) & (4) \\ & \textit{if } p = ip', t \equiv G(t_1,\ldots,t_n),\ n > 0 \\ & \quad \textit{and } G \in Dom(\sigma) \\ & \textit{where } G\sigma \equiv \lambda y_1 \ldots y_n.t' \\ & \textit{s.t. } \sigma' = \{y_1 \mapsto t_1\sigma\downarrow,\ldots,y_n \mapsto t_n\sigma\downarrow\} \end{cases}$$

Definition 9. *Let $A : s[l\sigma\downarrow]_p \to_{l\to r} s[r\sigma\downarrow]_p$ be a reduction for a rewrite rule $l \to r \in R$, a substitution σ, a term s and occurrence p in s. Then, the set of descendants of q in s by A is defined as follows:*

$$q\backslash A = \begin{cases} \{q\} & \textit{if } q \mid p \textit{ or } q \prec p \\ \{pp_3 \mid p_3 \in PV(r,\sigma,top(l|_{p_1}),p_2)\} & \\ & \textit{if } q = pp_1p_2 \textit{ and } p_1 \in Occ_V(l) \\ \emptyset & \textit{otherwise.} \end{cases}$$

For normalized terms s and t such that $A : s[t]_p \trianglerighteq t$, the descendants is defined simply as

$$q\backslash A = \begin{cases} \{p'\} & \text{if } q = pp' \\ \emptyset & \text{otherwise.} \end{cases}$$

For a reduction sequence $B : s\ (\to \cup \trianglerighteq)^*\ t$, the set $q\backslash B$ of descendants is naturally extended.

4.2 Sufficient Conditions

This subsection shows conditions that guarantees the finite branchingness of the dependency forests (Lemma 4).

We say a term t is *strongly linear* if there exists an α-equal term s each of which variable occurs only once in it. For example, $f(X,Y)$, $f(\lambda x.x, \lambda x.g(x))$ are strongly linear, while $f(X,X)$, $f(\lambda x.g(x,x))$ are not. We say an HRS is *strongly linear* if r is strongly linear for every rule $l \to r$. We say a substitution $\sigma = \{X_1 \mapsto t_1,\ldots,X_n \mapsto t_n\}$ is *strongly linear* if the term $c(t_1,\ldots,t_n)$ is strongly linear for a constant c.

Let W be a set of variables. We say a term t is *nested with respect to W* if there is a subterm $F(t_1,\ldots,t_n)$ in t such that $F \in W$ and

(a) $W \cap FV(t_i) \neq \emptyset$ for some i, or
(b) t' is nested with respect to $\{x_1, \ldots, x_m\}$ for some $t_i \equiv \lambda x_1 \cdots x_m.t'$.

Especially, we say a term t is *nested* if it is nested with respect to some free variable in $FV(t)$. For example, $F(X)$ and $F(\lambda xy.x(y))$ are nested, while $f(F(d), X)$, $f(\lambda xy.x(y))$, $f(\lambda x.F(x))$, $f(\lambda xy.F(x, y))$ and $F(\lambda xy.f(x, y))$ are not. We say an HRS is *non-nested* if r is non-nested for every rule $l \to r$.

The following is the key lemma.

Lemma 4. *Let R be an HRS and DF be a dependency forest for a minimal non-terminating sequence A.*

(a) If R is strongly linear and A begins at a strongly linear term, then DF is finite branching.
(b) If R is non-nested, then DF is finite branching.

From now, we prepare technical lemmas for proving the above lemma.

Proposition 2 ([13]). *If $F \notin (FV(t) \cap Dom(\sigma))$ then $PV(t, \sigma, F, p) = \emptyset$ for any p.*

Proposition 3. *Let t be a strongly linear term. Let σ be a strongly linear and closed substitution. Then $t\sigma\!\downarrow$ is strongly linear.*

We use $|P|$ for the number of elements of a set P.

Lemma 5. *Let t be a strongly linear term. Let σ be a strongly linear, normalized and closed substitution.*

(a) If p is an occurrence of $F\sigma$ then $|PV(t, \sigma, F, p)| \leq 1$.
(b) If p is an occurrence of t, then $|PT(t, \sigma, p)| \leq 1$.

Proof. We prove (a) and (b) simultaneous induction on the definition of PV and PT. For (a), we have six cases according to the definition of PV. We abbreviate $PV(t, \sigma, F, p)$ as P.

(PV1) Since $t \equiv F$, we have $P = \{p\}$ and the claim trivially holds.
(PV2) Let $t \equiv a(t_1, \ldots, t_n)$. we have at most one i such that t_i contains F from the linearity of t. Hence, $P = \emptyset$ or $P = PV(t_i, \sigma, F, p)$ from Proposition 2. Thus, the claim holds from the induction hypothesis.
(PV3) Let $t \equiv \lambda x_1 \cdots x_n.t'$. The claim directly holds by the induction hypothesis since t' is also strongly linear.
(PV4) Let $t \equiv G(t_1, \ldots, t_n)$ for $F \neq G$. Since we have at most one i such that t_i contains F from linearity of t, we get $P = \emptyset$ or $P = PV(t', \sigma', y_i, PV(t_i, \sigma, F, p))$ from Proposition 2 where $G\sigma \equiv \lambda y_1 \cdots y_n.t'$ and $\sigma' = \{y_1 \mapsto t_1\sigma\!\downarrow, \ldots, y_n \mapsto t_n\sigma\!\downarrow\}$. In the former case, we have done. Consider the latter case. We have that t' is strongly linear from the strong linearity of σ. We also have that σ' is strongly linear from the strong linearity of t and σ by Proposition 3. Since we have $|PV(t_i, \sigma, F, p)| \leq 1$ by the induction hypothesis, $P = \emptyset$ or $P = PV(t', \sigma', y_i, q)$ for $\{q\} \in PV(t_i, \sigma, F, p)$. Therefore, the claim holds by the induction hypothesis.

(PV5) In case of $t \equiv F(t_1, \dots, t_n)$, we have no i such that t_i contains F from linearity of t. Hence, we have $P = PT(t', \sigma', p)$ by Proposition 2 where t' and σ' are given as same as in the case (PV4). The claim holds by the induction hypothesis since t' and σ' are strongly linear.
(PV6) In this case, it is trivial.

For (b), we have four cases according to the definition of PT. We abbreviate $PT(t, \sigma, p)$ as P.

(PT1) In this case, we have $P = \{\varepsilon\}$ and the claim trivially holds.
(PT2) Let $p = ip'$ and $t \equiv a(t_1, \dots, t_n)$. Then, we have $P = \{iq \mid q \in PT(t_i, \sigma, p)\}$ and the claim holds from the induction hypothesis.
(PT3) Let $t \equiv \lambda x_1 \cdots x_n.t'$. Then, $P = PT(t', \sigma|_{\overline{x_1,\dots,x_n}}, p)$. Since t' is strongly linear, the claim holds by the induction hypothesis.
(PT4) Let $p = ip'$ and $t \equiv G(t_1, \dots, t_n)$. We have $|PT(t_i, \sigma, p')| \leq 1$ by the induction hypothesis. Thus, $P = \emptyset$ or $P = PV(t', \sigma', y_i, q)$ for $\{q\} \in PT(t_i, \sigma, p')$. Since t' and σ' are strongly linear from the strong linearity of t and σ by Proposition 3, the claim holds by the induction hypothesis. □

Lemma 6. *Let R be a strongly linear HRS, t be a strongly linear term and p be an occurrence of t. Then, $p\backslash A$ is empty or singleton for a sequence $A : t\ (\to_R \cup \trianglerighteq)^*\ t'$.*

Proof. It is enough to show the case $A : t\ (\to_R \cup \trianglerighteq)\ t'$ because t' is also strongly linear.

For $A : t \trianglerighteq t'$, it is trivial.

Let $A : t \equiv t[l\sigma\downarrow]_q \to_R t[r\sigma\downarrow]_q \equiv t'$ for some $l \to r \in R$. We can assume t is closed without loss of generality. The non-trivial case is that $p = qp_1p_2$ and $p_1 \in Occ_V(l)$. Since t is strongly linear and closed and l is a pattern, we have σ is strongly linear and closed. Hence, the lemma holds by Lemma 5(a). □

Lemma 7. *Let t be a term and σ be a substitution.*

(a) If the occurrences in $P_F \subseteq Occ(F\sigma)$ are pairwise disjoint for each $F \in Dom(\sigma)$ and t is non-nested with respect to $Dom(\sigma)$, then the occurrences in $\bigcup_{F \in Dom(\sigma)} PV(t, \sigma, F, P_F)$ are pairwise disjoint.
(b) If the occurrences in $P \subseteq Occ(t)$ are pairwise disjoint and t' is non-nested with respect to $\{y_1, \dots, y_n\}$ for every $X \in FV(t)$ where $X\sigma \equiv \lambda y_1 \cdots y_n.t'$, then the occurrences in $PT(t, \sigma, P)$ are pairwise disjoint.

Proof. We prove (a) and (b) simultaneous induction on the definition of PV and PT. Firstly, consider (a). We abbreviate $\bigcup_{F \in Dom(\sigma)} PV(t, \sigma, F, P_F)$ as Q.

(1) In case of $t \equiv F$, we have $Q = PV(F, \sigma, F, P_F) = P_F$ by Proposition 2 and the definition of PV. Hence, the claim trivially holds.
(2) In case of $t \equiv a(t_1, \dots, t_n)$ for $a \in \mathcal{F} \cup \overline{Dom(\sigma)}$, the occurrences in $\bigcup_{F \in Dom(\sigma)} PV(t_i, \sigma, F, P_F)$ are pairwise disjoint from the induction hypothesis for every i. The claim follows from $Q = \{iq \mid q \in \bigcup_{F \in Dom(\sigma)} PV(t_i, \sigma, F, P_F)\}$.

(3) In case of $t \equiv \lambda x_1 \cdots x_n.t'$, let $\sigma' = \sigma|_{\overline{\{x_1,\ldots,x_n\}}}$. Since we have $Q = \bigcup_{F \in Dom(\sigma')} PV(t', \sigma', F, P_F)$ and t' is non-nested with respect to $Dom(\sigma')$, the claim holds by the induction hypothesis.
(4) In case of $t \equiv G(t_1, \ldots, t_n)$ and $G \in Dom(\sigma)$. Since we have no i such that t_i contains free variables in $Dom(\sigma)$ from non-nestedness, $\bigcup_{F \in Dom(\sigma)} PV(t_i, \sigma, F, P_F) = \emptyset$ from Proposition 2. Hence, $Q = PT(t', \sigma', P_G)$, where $G\sigma = \lambda y_1 \cdots y_n.t'$ and $\sigma' = \{y_1 \mapsto t_1\sigma', \ldots, y_n \mapsto t_n\sigma'\}$. We also have that t'_i is non-nested with respect to $\{z_1, \ldots, z_m\}$ for every i, where $y_i\sigma' \equiv t_i \equiv \lambda z_1 \cdots z_m.t'_i$. Since $t_i\sigma' = t_i$ for each i, we can apply the induction hypothesis and the claim holds.
(5) In the other cases, it is trivial.

Secondly, consider (b). We abbreviate $PT(t, \sigma, P)$ as Q. In case of $\varepsilon \in P$, we have $P = \{\varepsilon\}$ since the occurrences in P are pairwise disjoint. Hence, we have $Q = \{\varepsilon\}$ and the claim holds. In case of $\varepsilon \notin P$, we have two subcases.

(1) In case of $t \equiv \lambda x_1 \cdots x_n.t'$, we have $Q = PT(t', \sigma|_{\overline{\{x_1,\ldots,x_n\}}})$. The claim holds by the induction hypothesis.
(2) In case of $t \equiv a(t_1, \ldots, t_n)$, let $P_i = \{p \mid ip \in P\}$ for each i. The occurrences in $Q'_i = PV(t_i, \sigma, P_i)$ are pairwise disjoint for each i from the induction hypothesis.
  (2-1) If $a \in \mathcal{F} \cup \overline{Dom(\sigma)}$, we have $Q = \bigcup_i \{iq \mid q \in Q'_i\}$. Hence, the claim follows.
  (2-2) If $a = G \in Dom(\sigma)$, let $G\sigma \equiv \lambda y_1 \cdots y_n.t'$. Then, we have $Q = \bigcup_i PV(t', \sigma', y_i, Q'_i)$, where $\sigma' = \{y_1 \mapsto t_1\sigma{\downarrow}, \ldots, y_n \mapsto t_n\sigma{\downarrow}\}$. Since t' is non-nested with respect to $Dom(\sigma')$, the occurrences in $Q = \bigcup_i PV(t', \sigma', y_i, Q'_i)$ are pairwise disjoint by the induction hypothesis.

□

Lemma 8. *Let R be a non-nested HRS, t be a term and P be a set of occurrences of t. If the occurrences in P are pairwise disjoint, then the occurrences in $P\backslash A$ are also pairwise disjoint for a reduction $A : t\ (\to_R \cup \trianglerighteq)^*\ t'$.*

Proof. It is enough to show the case $A : t\ (\to_R \cup \trianglerighteq)\ t'$.

For $A : t \trianglerighteq t'$, it is trivial.

Let $A : t \equiv t[l\sigma{\downarrow}]_q \to_R t[r\sigma{\downarrow}]_q \equiv t'$ for some $l \to r \in R$. Let $P_F = \{p_2 \mid qp_1p_2 \in P, F = top(l|_{p_1})\}$. Then, we have

$$\begin{aligned} P\backslash A = \{&p \in P \mid p \not\geq q\} \\ &\cup \{qp_3 \mid p_3 \in \textstyle\bigcup_{F \in Dom(\sigma)} PV(r, \sigma, F, P_F)\}. \end{aligned}$$

Since the occurrences in P_F are pairwise disjoint for each $F \in Dom(\sigma)$, the lemma holds by Lemma 7(a). □

Proof of Lemma 4.

Let R be an HRS and DF be a dependency forest for a minimal non-terminating sequence A:

$$
\begin{array}{l}
u_1 \overset{>\Lambda}{\rightarrow}_{e_1} u_2 \overset{>\Lambda}{\rightarrow}_{e_2} \cdots \overset{>\Lambda}{\rightarrow}_{e_{k_1-1}} u_{k_1} \\
\overset{\Lambda}{\rightarrow}_{e_{k_1}} v_1 \\
\trianglerighteq u_{k_1+1} \overset{>\Lambda}{\rightarrow}_{e_{k_1+1}} u_{k_1+2} \overset{>\Lambda}{\rightarrow}_{e_{k_1+2}} \cdots \overset{>\Lambda}{\rightarrow}_{e_{k_2-1}} u_{k_2} \\
\overset{\Lambda}{\rightarrow}_{e_{k_2}} v_2 \\
\quad \vdots ,
\end{array}
$$

where each e_i denotes a rule $l_i \rightarrow r_i$.

Firstly, we show that DF is finite branching if R is non-nested. Consider nodes having flag Λ, say $\langle n, t, \Lambda\rangle$. The outedges from it are added only in the case ii-i) of the definition. In this case, we have $n = k_{m-1} + 1$ for some m and outedges from it are added only when $i = k_m$. Thus no infinite outedges from these nodes.

Consider the other types of nodes $\langle n, t, >\Lambda\rangle$. Since these nodes are not removed by (Step 3) of the definition, there is a node having a flag Λ reachable from $\langle n, t, >\Lambda\rangle$.

(1) If it is directly reachable by an edge added in the case ii-ii) of the definition, the destination of the edge is $\langle i+1, u_{i+1}, \Lambda\rangle$, $n < i = k_m$ and the occurrence ε of u_{i+1} is an descendant of t in u_n. Hence, u_{i+1} is the only one descendant of t in u_n from Lemma 8. Since $u_{k_m+1} \overset{\Lambda}{\rightarrow} v_{m+1}$, the descendants of t in u_n disappears by this reduction, which means that no outedge from $\langle n, t, >\Lambda\rangle$ to nodes numbered greater than k_{m+1}. Thus, no infinite outedges from this node.
(2) Otherwise, we have a path to a node $\langle k_m + 1, u_{k_m+1}, \Lambda\rangle$ from $\langle n, t, >\Lambda\rangle$ via edges added by i) and an edge added by ii-ii). Similarly to the above case, we can show that no outedge from $\langle n, t, >\Lambda\rangle$ to nodes numbered greater than $k_m + 1$.

Secondly, we can show that DF is finite branching if R is strongly linear and A begins at a strongly linear term by using Lemma 6 instead of Lemma 8, $\square$

Example 7. Consider the following strongly linear HRS:

$$
R_5 = \begin{cases} f(X, \lambda x.F(x)) \rightarrow F(X), \\ g(a, a) \rightarrow f(a, \lambda x.g(b, x)) \\ b \rightarrow a \end{cases}
$$

and the minimal non-terminating sequence

$$
\begin{array}{l}
f(b, \lambda x.g(x, x)) \rightarrow g(b, b) \\
\trianglerighteq g(b, b) \rightarrow g(a, b) \rightarrow g(a, a) \rightarrow f(a, \lambda x.g(b, x)) \\
\trianglerighteq f(a, \lambda x.g(b, x)) \rightarrow g(b, a) \\
\trianglerighteq g(b, a) \rightarrow g(a, a) \rightarrow f(a, \lambda x.g(b, x)) \\
\trianglerighteq f(a, \lambda x.g(b, x)) \rightarrow g(b, a) \\
\quad \vdots
\end{array}
$$

The dependency forest of the minimal non-terminating sequence is shown in Fig. 4(a), where dotted nodes and edges are removed ones by (Step 3) of the definition.

Example 8. Consider the following non-nested HRS:

$$R_6 \;=\; \begin{cases} f(\lambda xy.F(\lambda z.x(z), y)) \\ \qquad \to h(F(\lambda z.g(z), a), F(\lambda z.g(z), a)) \\ g(a) \to f(\lambda xy.h(x(a), y)) \end{cases}$$

and the minimal non-terminating sequence

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

One duplication appears in this reduction sequence, i.e., only the first reduction duplicates the term a. The dependency forest of the minimal non-terminating sequence is shown in Fig. 4(b).

Now we obtain the following theorem from lemmas 3 and 4.

Theorem 1. *Let R be an HRS R.*

(a) If R is strongly linear and there is no infinite R-chain, every strongly linear term is terminating.
(b) If R is non-nested and there is no infinite R-chain, R is terminating.

Example 9. Consider the following HRS R_7:

$$R_7 \;=\; \begin{cases} compo(\lambda x.F(x), \lambda y.G(y), Z) \to F(G(Z)), \\ apply(\lambda x.F(x), X) \to F(X) \end{cases}$$

Since R_7 has no dependency pair, there is no (infinite) R-chain. Hence, it is terminating by Theorem 1(a).

Example 10. Consider the following nested HRS[21] that is not strongly linear:

$$R_8 \;=\; \{\; f(g(\lambda x.F(x))) \to F(g(\lambda x.h(F(x)))) \;\}$$

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

$$\begin{array}{l} f(g(\lambda x.f(x))) \\ \to_{R_8} f(g(\lambda x.h(f(x)))) \\ \to_{R_8} h(f(g(\lambda x.h(h(f(x)))))) \\ \to_{R_8} h(h(h(f(g(\lambda x.h(h(h(f(x))))))))) \\ \quad \vdots \end{array}$$

and the dependency forest for the minimal non-terminating sequence

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

is shown in Fig. 5.

The following example shows that even the duplication of first-order variable is harmful.

Example 11. Consider the following HRS:

$$
R_9 = \begin{cases} i(X) \to g(X, X), \\ g(h(\lambda x.F(x)), X) \to F(X) \end{cases}
$$

Although we have only one dependency pair $\langle i^\#(X),\ g^\#(X, X)\rangle$, the following infinite reduction sequence exists;

$$
\begin{array}{l}
i(h(\lambda x.i(x))) \\
\to_{R_9} g(h(\lambda x.i(x)), h(\lambda x.i(x))) \\
\to_{R_9} i(h(\lambda x.i(x))) \\
\quad \vdots
\end{array}
$$

and the dependency forest for the minimal non-terminating sequence

$$
\begin{array}{l}
i(h(\lambda x.i(x))) \xrightarrow{\Lambda} g(h(\lambda x.i(x)), h(\lambda x.i(x))) \\
\trianglerighteq g(h(\lambda x.i(x)), h(\lambda x.i(x))) \xrightarrow{\Lambda} i(h(\lambda x.i(x))) \\
\trianglerighteq i(h(\lambda x.i(x))) \xrightarrow{\Lambda} g(h(\lambda x.i(x)), h(\lambda x.i(x))) \\
\quad \vdots
\end{array}
$$

is shown in Fig. 6.

5 Proving termination

We can apply the method similarly to the first-order case for proving termination of HRSs. While the reference [21] 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 9. *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.*

Proof. Assume we have an infinite R-chain $\langle s_1, t_1 \rangle\ \langle s_2, t_2 \rangle\ \langle s_3, t_3 \rangle \cdots$. Then there exist substitutions $\sigma_1,\ \sigma_2,\ \ldots$ 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. □

Corollary 1. *Let* R *be an HRS. If there exists a reduction quasi-ordering satisfying the conditions (a) and (b) in Lemma 9. Then,*

(a) If R *is strongly linear, every strongly linear term is terminating.*
(b) If R *is non-nested,* R *is terminating.*

Example 12. Consider the HRS R_2, which is strongly linear. From Corollary 1(a), we must find a reduction quasi-ordering $\succeq$ satisfying the following constraints:

$$\begin{array}{l} f(\lambda x.F(x), s(X)) \succeq f(\lambda x.F(a), f(\lambda x.F(x), X)) \\ f^{\#}(\lambda x.F(x), s(X)) \succ f^{\#}(\lambda x.F(a), f(\lambda x.F(x), X)) \\ f^{\#}(\lambda x.F(x), s(X)) \succ f^{\#}(\lambda x.F(x), X). \end{array}$$

By using argument filtering method, it is enough to find a reduction quasi-ordering $\succeq'$ satisfying $s(X) \succeq' X$ obtained the above conditions by replacing $f(t, u)$ by u and $f^{\#}(t, u)$ by u. Since it is easy to find such $\succeq'$, we can show that R_2 is terminating.

6 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 indicate 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.

Combining the following method with the dependency method gives more power to proving termination:

– The dependency graph refinement is 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 inverse of Lemma 3 does not satisfy. The result of this paper is applicable only if HRSs are strongly linear or non-nested unfortunately. It is strongly desirable to find weaker conditions.

Acknowledgment

We thank anonymous referees for their useful suggestions. This work is partly supported by MEXT. KAKENHI #15500007.

References

1. P.B. Andrews, Resolution in type theory, Journal of Symbolic Logic, Vol.36, No.3, pp.414–432, 1991.
2. 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.
3. T. Arts, Automatically proving termination and innermost normalization of term rewriting system Ph.D. thesis, Utrecht University, The Netherlands, 1997.
4. T. Arts and J. Giesl, Termination of term rewriting using dependency pairs, Theoretical Computer Science, Vol.236, pp.133–178, 2000.
5. H. Barendregt, Lambda calculi with types, in Handbook of Logic in Computer Science, ed.Abramsky el al., Oxford University Press, 1993.
6. F. Baader and T. Nipkow, Term rewriting and all that, Cambridge University Press, 1998.
7. J.-Y. Girard, Interprétation fonctionnelle et élimination des coupures del'arithmétique d'ordre supérieur, Ph.D. thesis, University of Paris VII, 1972.
8. 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, 1991.
9. 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.
10. M. Iwami and Y. Toyama, Simplification ordering for higher-order rewrite systems, IPSJ Trans. on Programming, Vol.40, No.SIG 4 (PRO 3), pp.1–10, 1999.
11. 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.
12. J.-P. Jouannaud, A. Rubio, The higher-order recursive path ordering, in Proc. of IEEE Symposium on Logic in Computer Science, Trento, Italy, 1999.
13. H. Kasuya, M. Sakai and K. Agusa, Descendants and head normalization of higher-order rewrite systems, in 6th International Symposium on Functional and Logic Programming, , FLOPS2002, in LNCS 2441, pp.198–211, 2002.
14. K. Kusakari, On proving termination of term rewriting systems with higher-order variables, IPSJ Trans. on Programming, Vol.42, No.SIG 7 (PRO 11), pp.35–45, 2001.
15. K. Kusakari, Higher-order path orders based on computability, IEICE Trans. on Inf. & Syst., Vol.E87-D, No.2, pp.352–359, 2004.
16. D. Miller, A logic programming language with lambda-abstraction, function variables, and simple unification, Journal of Logic and Computation, Vol.1, No.4, pp.497–536, 1991.
17. O. Lysne and J. Piris, A termination ordering for higher order rewrite systems, in Proc. 6th International Conference on Rewriting Techniques and Applications, RTA'95, in LNCS, Vol.914, pp.26–40, 1995.

18. C. Loría-Sáenz and J. Steinbach, Termination of combined (rewrite and λ-calculus) systems, in Proc. 3rd International Workshop on Conditional Term Rewriting Systems, CTRS'92, in LNCS, Vol.656, pp.143–147, 1993.
19. T. Nipkow, Higher-order critical pairs, in Proc. 6th annual IEEE Symposium on Logic in Computer Science, pp.342–349, 1991.
20. F. Raamsdonk, On termination of higher-order rewriting, in Proc. of 12th Int. Conf. on Rewriting Techniques and Applications, in LNCS, Vol.2051, pp.261–275, 2001.
21. M. Sakai, 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.
22. W.W. Tait, Intensional interpretation of functionals of finite type, Journal of Symbolic Logic, Vol.32, pp.198–212, 1967.

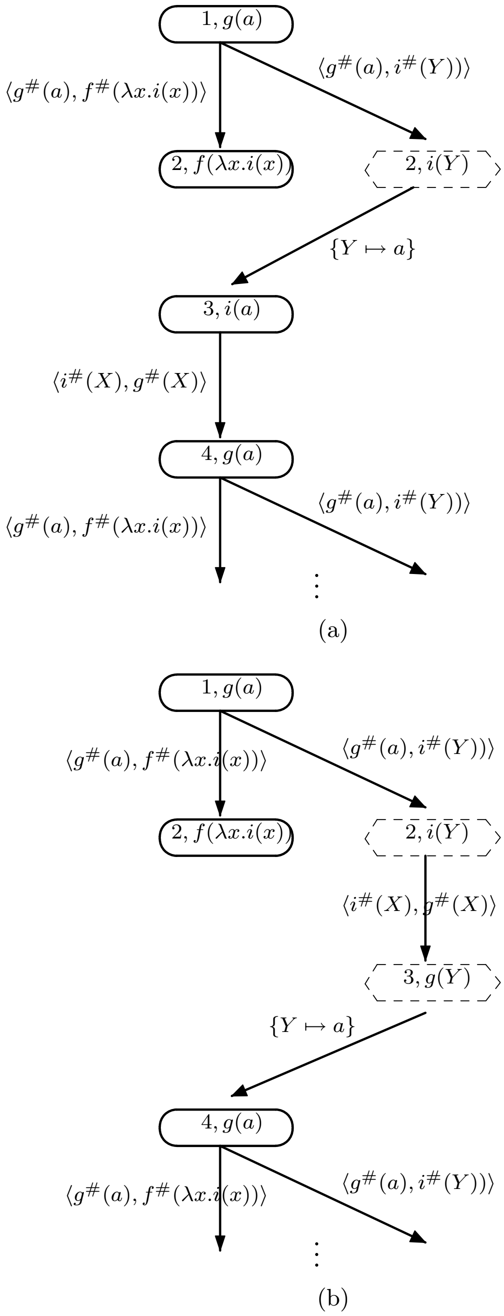

Fig. 2. The dependency forest of the minimal non-terminating sequence in Example 3 and 5 (Nodes whose third items are Λ ($>\Lambda$) are drawn by solid (dashed) lines)

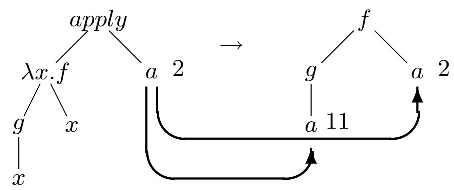

Fig. 3. Descendants

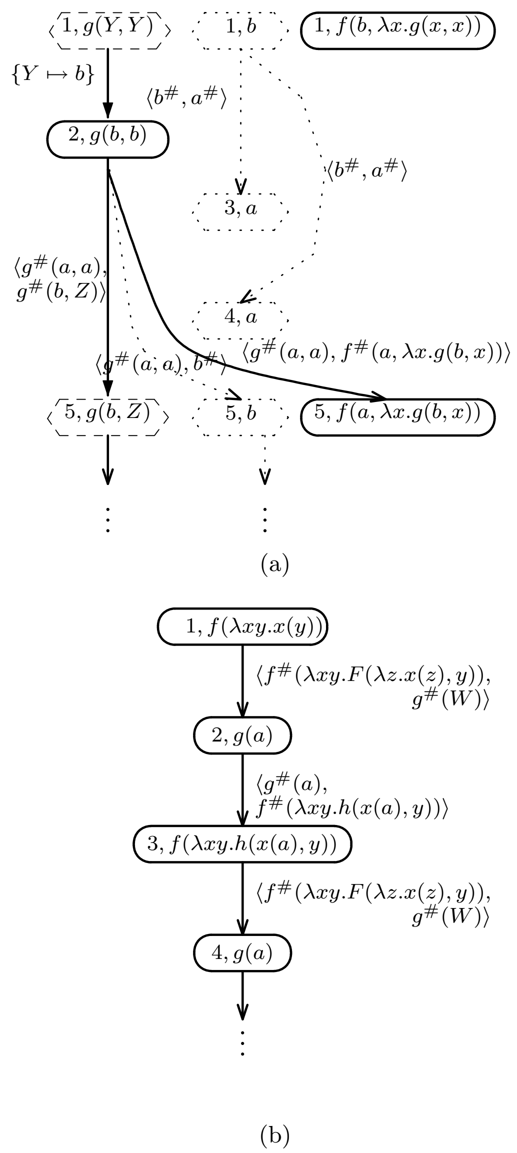

Fig. 4. The dependency forest of the minimal non-terminating sequence in Example 7 and 8 (Nodes whose third items are Λ ($>\Lambda$) are drawn by solid (dashed) lines. Nodes and edges removed from the graph are drawn by dotted lines)

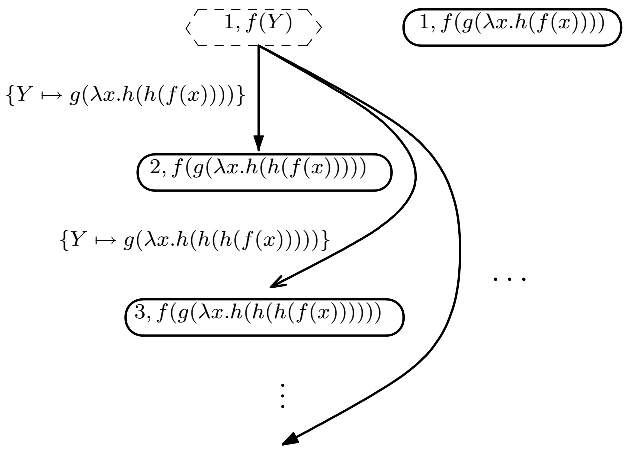

Fig. 5. The dependency forest of the minimal non-terminating sequence in Example 10

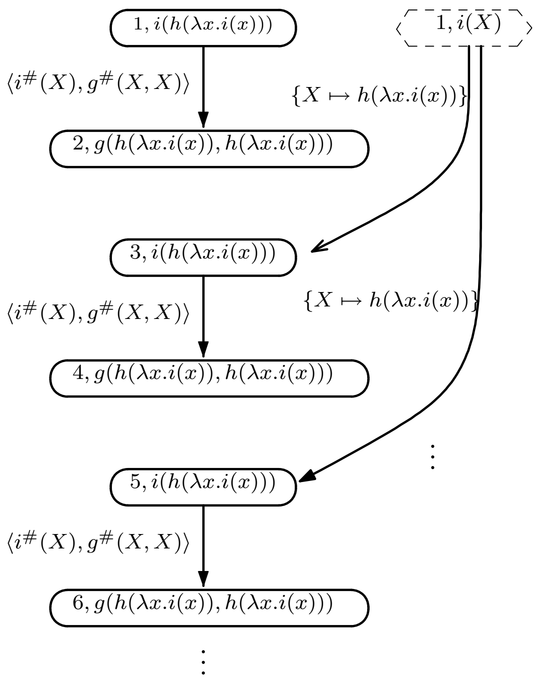

Fig. 6. The dependency forest of the minimal non-terminating sequence in Example 11