社団法人 電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
IEICE Technical Report
SS2007-60(2008-3)

単純型付き項書き換え系における静的依存対法とその周辺

草刈圭一朗† 酒井正彦†

† 名古屋大学大学院情報科学研究科
〒 464-8603 名古屋市千種区不老町
E-mail: †{kusakari,sakai}@is.nagoya-u.ac.jp

あらまし 我々が提案した関数プログラムの強力な停止性証明法である静的依存対法は一般には適用できないため取り扱うプログラムに一定の制限を課す必要がある．このような制限として我々は直接関数渡しと呼ばれる性質を提案した．本論文ではより適用範囲の広い関数渡しの安全条件を提案し，このクラスで静的依存対法が健全であることを示す．また，依存対法で停止性を証明する際には，引数切り落とし法や実効規則が重要となる．本論文では，既存の引数切り落とし法と異なり型の構造を破壊しない引数切り落とし法も与える．さらに，実効規則の既存の成果を拡張して引数切り落とし法と組合せた実効規則の概念を与える．

キーワード 単純型付き項書き換え系，停止性，静的依存対，引数切り落とし法，実効規則．

Static Dependency Pair Method for Simply-Typed Term Rewriting and Related Techniques

KUSAKARI Keiichirou † and SAKAI Masahiko †

† Nagoya University, Graduate School of Information Science
Furo-cho, Chikusa-ku, Nagoya, 464–8601 Japan
E-mail: †{kusakari,sakai}@is.nagoya-u.ac.jp

Abstract We proposed a static dependency pair method, which can effectively prove termination of functional programs. Since the method is not applicable in general, we proposed plain function-passing as a restriction. In this paper, we refine the method. Firstly we propose the notion of safely function-passing, which relax the restriction of plain function-passing. Next we improve the argument filtering method, which support dependency pair methods by generating a reduction pair from a given reduction order. Our argument filtering method does not destroy type structure unlike existing method. Hence our method can effectively apply reduction orders which make use of type information. Finally we combine argument filtering method and usable rules, which reduce the number of constraints.

Key words Simply-Typed Term Rewriting, Termination, Static Dependency Pair, Argument Filtering, Usable Rule.

1. Introduction

Simply-typed term-rewriting systems (STRSs), introduced in [4], is a computational model that provides operational semantics for functional programs and directly handles higher-order functions. In order to prove termination of STRSs we proposed a static dependency pair method using static recursive structure analysis based on definition dependency [6]. The key idea of the method is to analyze a recursive structure from the viewpoint of strong computability, and observation that the analysis result, called recursive components, correspond to intuitive static recursive structure. Since the method is not applicable to general STRSs, we proposed the notion of plain function-passing in which the method work well. In this paper, we expand the range in application of static dependency pair method by introducing the notion of safely function-passing.

When proving by the static dependency pair methods, we need to show the non-loopingness of each static recursion components. Then we use the reduction pair or the subterm

criterion [2], [6]. The argument filtering method generates a reduction pair from a given reduction order. The method was introduced in TRSs [1], and extended to STRSs [4]. However the method does not work well in general STRSs and may destroy the well-typedness of terms. In [4], we brought out the fact that the method work well in left-firmness STRSs, that is, any variable of left-hand sides occurs at a leaf position. On the other hand, destroying the well-typedness remarkably complicates the application of the argument filtering method to reduction order which make use of type information [5]. In this paper, we improve the argument filtering method in STRSs, which never destroys the well-typedness.

In order to reduce the number of constraints when proving the non-loopingness of recursion components, the notion of usable rules was introduced in TRSs [2], [9]. We extended the notion onto STRSs [7]. In first-order TRSs, we know that usable rules can be strengthened by combining the argument filtering method [9]. In this paper, we introduce usable rules combined with the argument filtering method in STRSs.

2. Preliminaries

Untyped term rewriting systems (UTRSs) were introduced by removing arity constraints from first-order term rewriting systems and simply-typed term rewriting systems (STRSs) were introduced as UTRSs with simple-type constraints [4].

We assume that the reader is familiar with notions of term rewriting systems [8], and of simply-typed term rewriting systems [4], [6]. In this paper, notations are according to the literature [6].

The set $T(\Sigma, \mathcal{V})$ of *(untyped) terms* generated from a set Σ of function symbols and a set $\mathcal{V}$ of variables with $\Sigma \cap \mathcal{V} = \emptyset$ is the smallest set such that $a[t_1, \ldots, t_n] \in T(\Sigma, \mathcal{V})$ whenever $a \in \Sigma \cup \mathcal{V}$ and $t_1, \ldots, t_n \in T(\Sigma, \mathcal{V})$.

A set of *basic types* is denoted by $\mathcal{B}$. The set $\mathcal{S}$ of *simple types* (with product types) is generated from $\mathcal{B}$ by type constructors $\to$ and $\times$, that is, $\mathcal{S} ::= \mathcal{B} \mid (\mathcal{S}_1 \to \mathcal{S}_2) \mid (\mathcal{S}_1 \times \cdots \times \mathcal{S}_n)$. A simple type α is said to be a *suffix* of a simple type β, denoted by $\alpha \sqsubseteq \beta$, if β has a form $\alpha_1 \to \cdots \to \alpha_n \to \alpha$ for some $\alpha_1, \ldots, \alpha_n$. A *typing function* τ is a function from $\mathcal{V} \cup (\Sigma \setminus \{\mathtt{tp}\})$ to $\mathcal{S}$, and it is naturally extended to terms. A term $t \in T(\Sigma, \mathcal{V})$ is said to be *simply typed* if t has a simple-type, that is, $\tau(t)$ is defined. A term t which has a simple-type α is often denoted by t^α. We denote the set of all simply-typed terms by $T_\tau(\Sigma, \mathcal{V})$.

A *simply-typed rule* is a pair (l, r) of simply-typed terms, denoted by $l \to r$, such that $root(l) \in \Sigma \setminus \{\mathtt{tp}\}$, $Var(l) \supseteq Var(r)$ and $\tau(l) = \tau(r)$. A *simply-typed term rewriting system* (STRS) is an abstract reduction system $\langle \mathcal{T}_\tau(\Sigma, \mathcal{V}), \xrightarrow[R]{} \rangle$. We often denote an STRS $\langle \mathcal{T}_\tau(\Sigma, \mathcal{V}), \xrightarrow[R]{} \rangle$ by R. For each STRS R, we define $\mathcal{T}_{SN}(R) = \{t \mid SN(R, t)\}$, $\mathcal{T}_{\neg SN}(R) = \mathcal{T}_\tau(\Sigma, \mathcal{V}) \setminus \mathcal{T}_{SN}(R)$, and $\mathcal{T}^{args}_{SN}(R) = \{t \mid \forall u \in args(t).SN(R, u)\}$.

Let R be an STRS and $l \to r \in R$ such that $\tau(l) = \alpha_1 \to \cdots \to \alpha_n \to \alpha$ and $\alpha \in \mathcal{S}_{nfun}$. The set $(l \to r)^{ex}$ of the *expansion forms* of a rule $l \to r$ is defined as $\{l \to r, l[z_1] \to r[z_1], \ldots, l[z_1, \ldots, z_n] \to r[z_1, \ldots, z_n]\}$, where $z_1^{\alpha_1}, \ldots, z_n^{\alpha_n}$ are fresh variables. We also define $R^{ex} = \bigcup_{l \to r \in R} (l \to r)^{ex}$. The rule $(l \to r)^{ex\uparrow}$ of the *full expansion form* of $l \to r$ is defined as $l[z_1, \ldots, z_n] \to r[z_1, \ldots, z_n]$, where $z_1^{\alpha_1}, \ldots, z_n^{\alpha_n}$ are fresh variables. We also define $R^{ex\uparrow} = \{(l \to r)^{ex\uparrow} \mid l \to r \in R\}$.

Proposition 2.1 Let R be an STRS. If $s \xrightarrow[R]{} t$ then there exist a rule $l \to r \in R^{ex}$, a leaf-context $C[\,]$ and substitution θ such that $s \equiv C[l\theta]$ and $t \equiv C[r\theta]$.

A term t is said to be *finite branching* in an STRS R if $\{t' \mid t \xrightarrow[R]{} t'\}$ is finite. An STRS R is said to be *finite branching* if any term is finite branching in R.

All root symbols of the left-hand sides of rules in an STRS R, denoted by $\mathcal{D}_R$, are called *defined*, whereas all other function symbols, denoted by $\mathcal{C}_R$, are called *constructors*. For each $f \in \mathcal{D}_R$, we define a new function symbol $f^\sharp$, called the *marked-symbol* of f. For each $t \equiv a[t_1, \ldots, t_n]$, we define the *marked term* $t^\sharp$ by $a^\sharp[t_1, \ldots, t_n]$ if $a \in \mathcal{D}_R$; otherwise $t^\sharp \equiv t$.

3. Static Dependency Pair Method

3.1 Safely Function-Passing

Definition 3.1 (Peeling Order) A well-founded strict order $\succ$ is said to be a *peeling order* if the following properties hold:

- $t \succ t_i$ for all $t^{\alpha_1 \to \alpha_2}$ and $t_i^{\alpha_i}$ $(i = 1, 2)$.
- $s \succ t \Rightarrow s\theta \succ t\theta$

For any peeling order $\succ$, term t and set A of types, we define $Sub^{\succ}_A(t)$ as the smallest set satisfying the following properties:

- $args(t) \subseteq Sub^{\succ}_A(t)$
- if $u \equiv a[u_1, \ldots, u_n] \in Sub^{\succ}_A(t)$, $a \in \mathcal{C}_R$, $\tau(u) \in A$ and $u \succ u_i$ then $u_i \in Sub^{\succ}_A(t)$

Definition 3.2 (Safely Function-Passing) An STRS R is said to be a *safely function-passing* with respect to a peeling order $\succ$ if there exists a set PT of non-functional types such that any $l \to r \in R$ and $v \in Sub(r)$ satisfy the following properties:

- if $root(v) \in \mathcal{V}_{fun}$ then there exists $u \in Sub^{\succ}_{PT}(l)$ such that $u \sqsubseteq v$, and
- if $v \in \mathcal{V}_{nfun}$ and $\tau(v) \in T$ then $v \in Sub^{\succ}_{PT}(l)$ holds.

The set PT is said to be a *peeling types*, and a safely function-passing STRS is often shortly denoted by SFP-STRS.

3.2 Strong Computability

Definition 3.3 (Strong Computability) Let R be an SFP-STRS with a peeling order $\succ$ and a peeling set PT. A term t is said to be *strongly computable* in R, if $SC(R,t)$ holds, which is inductively defined with respect to $\succ$ as follows:

- in case of $\tau(t) \in \mathcal{S}_{nfun} \setminus PT$, $SC(R,t)$ is defined as $SN(R,t)$,
- in case of $\tau(t) \in PT$, $SC(R,t)$ is defined as $SN(R,t)$ and $SC(R,u)$ for any $u \in \bigcup\{args(t') \mid t \xrightarrow[R]{*} t', root(t') \in \mathcal{C}_R\}$ such that $t \succ u$, and
- in case of $\tau(t) = \alpha \to \beta$, $SC(R,t)$ is defined as $SC(R,u) \Rightarrow SC(R,t[u])$ for any u^α.

For each SFP-STRS R, we define $\mathcal{T}_{SC}(R) = \{t \mid SC(R,t)\}$, $\mathcal{T}_{\neg SC}(R) = \mathcal{T}_\tau(\Sigma,\mathcal{V}) \setminus \mathcal{T}_{SC}(R)$, and $\mathcal{T}_{SC}^{args}(R) = \{t \mid \forall u \in args(t).SC(R,u)\}$.

Lemma 3.4 For any SFP-STRS R, the following properties hold:

(1) For any strongly computable terms $t^{\alpha_1\to\cdots\to\alpha_n\to\alpha}$ and $u_i^{\alpha_i}$ $(i=1,\ldots,n)$, we have $SC(R,t[u_1,\ldots,u_n])$.

(2) For any non-strongly computable term $t^{\alpha_1\to\cdots\to\alpha_n\to\alpha}$, there exist strongly computable terms $u_i^{\alpha_i}$ $(1 \leqq i \leqq n)$ such that $\neg SC(R,t[u_1,\ldots,u_n])$.

(3) Any variable z^α is strongly computable, for all $\alpha \in \mathcal{S}$.

(4) $SC(R,t^\alpha) \Rightarrow SN(R,t^\alpha)$, for all $\alpha \in \mathcal{S}$.

(5) $SC(R,t) \wedge t \xrightarrow{*} t' \Rightarrow SC(R,t')$ for all t and t'.

Proof. The proofs of (1), (2), and (5) are analogous to the proof of Lemma 4.3(4), Lemma 4.2, and Lemma 4.3(3) in [6]. We prove (3) and (4) by induction on α with respect to $\succ$.

The case $\alpha \in \mathcal{S}_{nfun} \setminus PT$ is trivial. In case of $\alpha \in PT$, $SN(R,t)$ and $SN(R,z)$ are trivial, and hence $SC(R,z)$ is also trivial because of $z \xrightarrow{*} t' \Rightarrow z \equiv t'$. Suppose that $\alpha = \alpha_1 \to \alpha_2$.

From the induction hypothesis, an arbitrary variable $z_1^{\alpha_1}$ is strongly computable. Thus $SC(R,t[z_1])$ holds. From the induction hypothesis, $t[z_1]$ is terminating, hence so is t.

Assume that z is not strongly computable. Let $\alpha_2 = \alpha_2' \to \cdots \to \alpha_n' \to \beta$ with $\beta \in \mathcal{S}_{nfun}$. From (2), there exist strongly computable terms $u_1^{\alpha_1},\ldots,u_n^{\alpha_n}$ such that $z[u_1,\ldots,u_n]$ is not strongly computable. From the induction hypothesis, each u_i is terminating. Hence $z[u_1,\ldots,u_n]$ is terminating. Since $z[u_1,\ldots,u_n]$ is not strongly computable and its type β is a non-functional type, we have $\beta \in PT$ and there exist terms u' and u such that $z[u_1,\ldots,u_n] \xrightarrow{*} u'$, $root(u') \in \mathcal{C}_R$, $u \in args(u')$, $z[u_1,\ldots,u_n] \succ u$, and u is not strongly computable. It is a contradiction because $root(l) \notin \mathcal{V}$ for all $l \to r \in R$. □

Lemma 3.5 Let R be an SFP-STRS, $l \to r \in R^{ex\uparrow}$, θ be a substitution such that $l\theta \in \mathcal{T}_{SC}^{args}(R)$. Then $SC(R,v\theta)$ holds for any $v \in Sub(r)$ such that $root(v) \in \mathcal{V}$ and $v\theta \in \mathcal{T}_{SC}^{args}(R)$.

Proof. Then there exists $l' \to r' \in R$ such that $l \equiv l'[z_1,\ldots,z_n]$ and $r \equiv r'[z_1,\ldots,z_n]$.

Suppose that $v \equiv z_i$ for some i. Then $SC(R,v\theta)$ follows from $z_i\theta \in args(l\theta)$.

Suppose that $v \in Sub(r)$ and $\tau(root(v)) \in \mathcal{S}_{nfun} \setminus PT$. Then $v \in Var(r) \subseteqq Var(l)$. Since $l\theta >_{sub} v\theta$ and $l\theta \in \mathcal{T}_{SN}^{args}(R)$ follows from Lemma 3.4(4), we have $SN(R,v\theta)$. Hence $SC(R,v\theta)$ holds.

Suppose that $v \in Sub(r)$ and either $\tau(v) \in PT$ or $root(v) \in \mathcal{V}_{fun}$. From the definition of peeling types, there exist a term u such that $u \in Sub_{PT}^{\succ}(l)$ and $u \sqsubseteq v$. Since any peeling order $\succ$ is closed under substitutions, we have $u\theta \in Sub_{PT}^{\succ}(l\theta)$. From the definitions of $Sub_{PT}^{\succ}$ and SC, we have $SC(R,u\theta)$ because of $l\theta \in \mathcal{T}_{SC}^{args}(R)$. Hence $SC(R,v\theta)$ follows from $v\theta \in \mathcal{T}_{SC}^{args}(R)$ and Lemma 3.4(1). □

3.3 Static Dependency Pair Method

In this subsection, we present a static dependency pair method for SFP-STRSs.

Definition 3.6 Let R be an SFP-STRS. For each $l \to r \in R^{ex\uparrow}$ and $a[r_1,\ldots,r_m] \in Sub(r)$ such that

- $a \in \mathcal{D}_R$,
- there exists no $u \in Sub_{PT}^{\succ}(l)$ such that $u \sqsubseteq a[r_1,\ldots,r_m]$, and
- there exists no $u \in Sub(l) \setminus \{l\}$ such that $u \equiv a[r_1,\ldots,r_m]$ and $\tau(u) \in \mathcal{S}_{nfun} \setminus PT$,

we define a *static dependency pair* of R as $l^\sharp \to a^\sharp[r_1,\ldots,r_m,z_1,\ldots,z_n]$, where $\tau(a^\sharp[r_1,\ldots,r_m,z_1,\ldots,z_n]) \in \mathcal{S}_{nfun}$ and $z_1,\ldots,z_n$ are fresh variables. We denote by $SDP(R)$ the set of all static dependency pairs of R.

Definition 3.7 Let R be an SFP-STRS. A sequence $u_0^\sharp \to v_0^\sharp, u_1^\sharp \to v_1^\sharp, \cdots$ of static dependency pairs of R is said to be a *static dependency chain* of R if there exist $\theta_0,\theta_1,\ldots$ such that $u_i^\sharp\theta_i, v_i^\sharp\theta_i \in \mathcal{T}_{SC}^{args}(R)$ and $v_i^\sharp\theta_i \xrightarrow{*} u_{i+1}^\sharp\theta_{i+1}$ for any i. A *static dependency graph* of R is a directed graph, in which nodes are $SDP(R)$ and there exists an arc from $u_0^\sharp \to v_0^\sharp$ to $u_1^\sharp \to v_1^\sharp$ if $u_0^\sharp \to v_0^\sharp, u_1^\sharp \to v_1^\sharp$ is a static dependency chain.

Definition 3.8 A (maximal) *static recursion component* of R is a set of nodes in a (maximal) strongly connected subgraph of static dependency graph. We denote by $SRC(R)$ the set of all static recursion component of R.

A static recursion component $C \in SRC(R)$ is said to be *non-looping* if there exists no infinite static dependency chain $u_0^\sharp \to v_0^\sharp, u_1^\sharp \to v_1^\sharp, \cdots$ such that $u_i^\sharp \to v_i^\sharp \in C$ for all i and

every $u^\sharp \to v^\sharp \in C$ occurs infinitely many times.

Lemma 3.9 If an SFP-STRS R is not terminating then $\mathcal{T}_{nfun}(\Sigma,\mathcal{V}) \cap \mathcal{T}_{\neg SC}(R) \cap \mathcal{T}_{SC}^{args}(R) \neq \emptyset$.

The proof is analogous to the proof of Lemma 4.15 in [6].

Lemma 3.10 Let R be an SFP-STRS. For any $t \in \mathcal{T}_{nfun}(\Sigma,\mathcal{V}) \cap \mathcal{T}_{\neg SC}(R) \cap \mathcal{T}_{SC}^{args}(R)$, there exist $l^\sharp \to v^\sharp \in SDP(R)$ and θ such that $t^\sharp \xrightarrow{*} l^\sharp\theta$ and $l\theta, v\theta \in \mathcal{T}_{nfun}(\Sigma,\mathcal{V}) \cap \mathcal{T}_{\neg SC}(R) \cap \mathcal{T}_{SC}^{args}(R)$.

Proof. Let $t \in \mathcal{T}_{nfun}(\Sigma,\mathcal{V}) \cap \mathcal{T}_{\neg SC}(R) \cap \mathcal{T}_{SC}^{args}(R)$. Then $t \in \mathcal{T}_{SN}^{args}(R)$ follows from $t \in \mathcal{T}_{SC}^{args}(R)$ and Lemma 3.4(4).

- Consider the case that $t \notin \mathcal{T}_{SN}(R)$. Since $t \in \mathcal{T}_{SN}^{args}(R) \cap \mathcal{T}_{nfun}(R)$, there exist $l \to r \in R^{ex\uparrow}$ and θ' such that $t^\sharp \xrightarrow{*} l^\sharp\theta'$, $\neg SN(R, l\theta')$ and $\neg SN(R, r\theta')$. Hence $\neg SC(R, l\theta')$ and $\neg SC(R, r\theta')$ follows from Lemma 3.4(4).
- Consider the case that $t \in \mathcal{T}_{SN}(R)$. Since $t \in \mathcal{T}_{\neg SC}(R) \cap \mathcal{T}_{nfun}(R)$, we have $\tau(t) \in PT$ and there exist terms t' and $t'' \in args(t')$ such that $t \xrightarrow{*} t'$, $root(t') \in \mathcal{C}_R$, $t \succ t''$ and $t'' \in \mathcal{T}_{\neg SC}(R)$. Assume that $root(t) \in \mathcal{C}_R$. From $t \in \mathcal{T}_{\neg SC}(R) \cap \mathcal{T}_{SN}(R) \cap \mathcal{T}_{nfun}(R)$, we have $\tau(t) \in PT$. Then $SC(R,t)$ follows from $t \in \mathcal{T}_{SN}(R) \cap \mathcal{T}_{SC}^{args}(R)$, $root(t) \in \mathcal{C}_R$ and Lemma 3.4(5). It is a contradiction. Hence $root(t) \notin \mathcal{C}_R$. Thus there exist $l \to r \in R^{ex\uparrow}$ and θ' such that $t^\sharp \xrightarrow{*} l^\sharp\theta'$ and $l\theta' \to r\theta' \xrightarrow{*} t'$. Since t'' is not strongly computable, so is t'. From Lemma 3.4(5), we have $\neg SC(R, l\theta')$ and $\neg SC(R, r\theta')$. In both cases above, we have $\{v' \in Sub(r) \mid \neg SC(R, v'\theta')\} \neq \emptyset$ because $r \in Sub(r)$ and $\neg SC(R, r\theta')$. Let $v' \equiv a[r_1, \ldots, r_m]$ be a minimal size term in this set. Then $SC(R, r_i\theta')$ holds for every i. From Lemma 3.4(2), there exist $v_1, \ldots, v_k \in \mathcal{T}_{SC}(R)$ such that $\tau(v'\theta'[v_1, \ldots, v_k]) \in \mathcal{S}_{nfun}$ and $\neg SC(v'\theta'[v_1, \ldots, v_k])$. Here $v'\theta'[v_1, \ldots, v_k] \in \mathcal{T}_{SC}^{args}(R)$.

  Now let $v \equiv a[r_1, \ldots, r_m, z_1, \ldots, z_k]$ for fresh variables $z_1, \ldots, z_k$ and we define $\theta(x)$ by v_i if $x = z_i$ $(i = 1, \ldots, k)$; otherwise by $\theta'(x)$. Then we have $l\theta = l\theta'$ and $v\theta = v'\theta'[v_1, \ldots, v_k]$. Since $l\theta \in \mathcal{T}_{SC}^{args}(R)$ follows from $t \in \mathcal{T}_{SC}^{args}(R)$ and Lemma 3.4(5), we have $l\theta \in \mathcal{T}_{nfun}(\Sigma,\mathcal{V}) \cap \mathcal{T}_{\neg SC}(R) \cap \mathcal{T}_{SC}^{args}(R)$. Because $v\theta \in \mathcal{T}_{nfun}(\Sigma,\mathcal{V}) \cap \mathcal{T}_{\neg SC}(R) \cap \mathcal{T}_{SC}^{args}(R)$ also holds, it suffices to show that $l^\sharp \to v^\sharp \in SDP(R)$. We prove this by contradiction. Assume that $l^\sharp \to v^\sharp \notin SDP(R)$. Let $l \equiv l'[z'_1, \ldots, z'_p]$ and $r \equiv r'[z'_1, \ldots, z'_p]$ such that $l' \to r' \in R$ and $z'_1, \ldots, z'_p$ are fresh variables.
- Assume that $a \in \mathcal{V}$. Then $SC(R, v\theta)$ follows from Lemma 3.5. This is a contradiction.
- Assume that $a \in \mathcal{C}_R$. Since $r_1\theta, \ldots, r_m\theta, v_1, \ldots, v_k$ are terminating from Lemma 3.4(4), $v\theta$ is terminating. Since $v\theta \in \mathcal{T}_{nfun}(\Sigma,\mathcal{V}) \cap \mathcal{T}_{\neg SC}(R)$, we have $\tau(v\theta) \in PT$ and there exist terms u' and $u'' \in args(u')$ such that $v\theta \xrightarrow{*} u'$, $root(u') \in \mathcal{C}_R$, $v\theta \succ u''$ and $u'' \in \mathcal{T}_{\neg SC}(R)$. Since $root(v\theta) = a \in \mathcal{C}_R$ and $v\theta \in \mathcal{T}_{SC}^{args}(R)$, $u' \in \mathcal{T}_{SC}^{args}(R)$ follows from Lemma 3.4(5). This is a contradiction.
- Assume that $a \in \mathcal{D}_R$ and there exists $u \in Sub_{PT}^{\succ}(l)$ such that $u \sqsubseteq a[r_1, \ldots, r_m]$. Then $SC(R, a[r_1, \ldots, r_m]\theta)$ is showed as similar to the proof of Lemma 3.5. Hence $SC(R, v\theta)$ follows from Lemma 3.4(1). This is a contradiction.
- Assume that $a \in \mathcal{D}_R$ and there exists $u \in Sub(l) \setminus \{l\}$ such that $u \equiv a[r_1, \ldots, r_m]$ and $\tau(u) \in \mathcal{S}_{nfun} \setminus PT$. Then $u \equiv v$ follows from $\tau(u) \in \mathcal{S}_{nfun}$. Since $l\theta \in \mathcal{T}_{SC}^{args}(R)$, $l\theta \in \mathcal{T}_{SN}^{args}(R)$ follows from Lemma 3.4(4), and hence $v\theta$ is terminating. Since $\tau(v\theta) \in \mathcal{S}_{nfun} \setminus PT$, $v\theta$ is strongly computable. This is a contradiction. □

We obtain the fundamental theorem of the static dependency pair method.

Theorem 3.11 Let R be an SFP-STRS. If there exists no infinite static dependency chain then R is terminating.

Note that the ($\Leftarrow$)-part does not hold in the theorem. The static dependency pair method therefore has a theoretical limitation.

Corollary 3.12 Let R be an SFP-STRS such that there exists no infinite path in the static dependency graph. If all recursion components in $SRC(R)$ are non-looping then R is terminating.

3.4 Non-loopingness of Recursion Components

Firstly we introduce the notion of (semi-)reduction pairs according to the literature [6]. The notion of reduction pairs was introduced in [3], which is a slightly abstraction of weak-reduction order [1]. The notion of semi-reduction pairs was essentially introduced in [4].

Definition 3.13 A pair $(\gtrsim, >)$ of a quasi-order $\gtrsim$ and a strict order $>$ is said to be a *semi-reduction pair* if $\gtrsim$ is closed under leaf-contexts and substitutions, $>$ is well-founded and closed under substitution, and either $\gtrsim \cdot > \subseteq >$ or $> \cdot \gtrsim \subseteq >$ holds. A semi-reduction pair $(\gtrsim, >)$ is said to be a *reduction pair* if $\gtrsim$ is closed under contexts.

Although the path order based on strong computability in [5] generates reduction pairs, the path order based on the simplification order in [4] does not generate reduction pairs and only generates semi-reduction pairs.

Proposition 3.14 [6] Let R be an STRS and C be a static recursion component. If there exists a reduction pair (resp. semi-reduction pair) $(\gtrsim, >)$ satisfying the following conditions then C is non-looping.

(i) $R \subseteq \gtrsim$ (resp. $R^{ex} \subseteq \gtrsim$),
(ii) $C \subseteq \gtrsim$ and $C \cap > \neq \emptyset$, and

We next introduced the subterm criterion [6] which are slightly improvements of the criterion in [2]. Although the original definition of the codomain of π (see the following definition) in [2] allows only positive integers, the following definition allows stipulates sequences of positive integers.

Definition 3.15 Let R be an SFP-STRS and $C \in SRC(R)$. We say that C satisfies the *subterm criterion* if for any $u^\sharp \to v^\sharp \in C$, and there exists a function π from $\mathcal{D}_R$ to non-empty sequences of positive integers such that

(α) $u|_{\pi(root(u))} >_{esub} v|_{\pi(root(v))}$ for some $u^\sharp \to v^\sharp \in C$, and

(β) the following conditions hold for any $u^\sharp \to v^\sharp \in C$:

- $u|_{\pi(root(u))} \geq_{esub} v|_{\pi(root(v))}$,
- $(u)_p \notin \mathcal{V}$ for all $p \prec \pi(root(u))$, and
- $q \neq \varepsilon \Rightarrow (v)_q \in \mathcal{C}_R$ for all $q \prec \pi(root(v))$.

Proposition 3.16 [6] Let R be an STRS and C be a static recursion component. If C satisfies the subterm criterion, then C is non-looping.

4. Argument Filtering Method

The argument filtering method, which is designed by eliminating unnecessary arguments, generate reduction pair from given reduction order. The method firstly introduced on first-order TRSs by Arts and Giesl [1]. After that the method was extended to STRSs by Kusakari [4]. However the method in STRSs may destroy the well-typedness of terms, which remarkably complicates the application of the argument filtering method to reduction order which make use of type information [5]. In this section we improve the argument filtering method which never destroys the well-typedness.

Definition 4.1 An argument filtering function is a function π such that for any $f \in \Sigma$, $\pi(f)$ is a list of positive integers $[i_1, \ldots, i_k]$ with $i_1 < \cdots < i_k \leqq n$, where $\tau(f) = \alpha_1 \to \cdots \alpha_n \to \beta$ and $\beta \in \mathcal{S}_{nfun}$. We can extend π over terms as $\pi(a[t_1, \ldots, t_n]) = a[t'_1, \ldots, t'_n]$, where each t'_i is defined as $t'_i \equiv \bot^{\alpha_i}$ if $a \in \Sigma$ and $i \notin \pi(a)$; otherwise $t'_i \equiv \pi(t_i)$.

For given argument filtering function π and binary relation $>$, we define $s \gtrsim^\pi t$ by $\pi(s) \geq \pi(t)$. We hereafter assume that if $\pi(F)$ is not defined explicitly then it is intended to be $[1, \ldots, n]$, where $\tau(f) = \alpha_1 \to \cdots \alpha_n \to \beta$ and $\beta \in \mathcal{S}_{nfun}$.

Unfortunately, it was indicated in [4] that $\gtrsim_\pi$ is not closed under substitutions. Hence the notion of left-firmness was introduced [4].

Definition 4.2 A term t is said to be *firmness* if any variable occurs at a leaf position. A pair (s, t) of term is said to be *left-firmness*, denoted by $LF(s, t)$, if s is firmness.

Theorem 4.3 For any (semi-)reduction order $>$ with $\bot$-condition, the pair $(\gtrsim_\pi, >_\pi)$ is a (semi-)reduction pair under the predicate LF.

Proof. Let $t \equiv a[t_1, \ldots, t_n]$. It suffices to show that $\pi(t)\theta_\pi \geq \pi(t\theta)$. We prove it by induction on $|t|$. From the induction hypothesis, $\pi(t_i)\theta_\pi \geq \pi(t_i\theta)$ for any i.

In case of $a \in \Sigma$, we suppose that $t'_i \equiv \bot$ if $a \in \Sigma$ and $i \notin \pi(a)$; otherwise $t'_i \equiv \pi(t_i)$, and $t''_i \equiv \pi(t_i\theta)$ if $i \in \pi(a)$; otherwise $t''_i \equiv \bot$. Then we have $t'_i\theta_\pi \geq t''_i$, and hence $\pi(t)\theta_\pi \equiv a[t'_1\theta_\pi, \ldots, t'_n\theta_\pi] \geq a[t''_1, \ldots, t''_n] \equiv \pi(t\theta)$.

In case of $a \in \mathcal{V}$ and $root(\theta(a)) \in \mathcal{V}$, we have $\pi(t)\theta_\pi \equiv \theta_\pi(a)[\pi(t_1)\theta_\pi, \ldots, \pi(t_n)\theta_\pi] \geq \theta_\pi(a)[\pi(t_1\theta), \ldots, \pi(t_n\theta)] \equiv \pi(\theta(a)[t_1\theta, \ldots, t_n\theta]) \equiv \pi(t\theta)$.

In case of $a \in \mathcal{V}$ and $root(\theta(a)) \in \Sigma$, we suppose that $\theta(a) = a'[u_1, \ldots, u_k]$ and $t''_i \equiv \pi(t_i\theta)$ if $i + k \in \pi(a')$; otherwise $t''_i \equiv \bot$. Then we have $\pi(t_i)\theta \geq t''_i$, and hence $\pi(t)\theta_\pi \equiv \theta_\pi(a)[\pi(t_1)\theta_\pi, \ldots, \pi(t_n)\theta_\pi] \geq \theta_\pi(a)[t''_1, \ldots, t''_n] \equiv \pi(\theta(a)[t_1\theta, \ldots, t_n\theta]) \equiv \pi(t\theta)$. □

5. Usable Rules

In this section, we strengthen usable rules in STRSs [7], by combining with the argument filtering method.

We define $Sub_\pi(a[t_1, \ldots, t_n])$ as $\{a[t_1, \ldots, t_n]\} \cup \bigcup_{i \in I} Sub_\pi(t_i)$ where $I = \pi(a)$ if $a \in \Sigma$; otherwise $I = \{1, \ldots, n\}$, and $Sub^{int}_{\mathcal{V},\pi}(t)$ as $\{t' \in Sub_\pi(t) \mid root(t') \in \mathcal{V},\ args(t') \neq [\,]\}$.

Definition 5.1 For each pair $\langle u, v\rangle$ of terms, the subset $\mathcal{U}'(\langle u, v\rangle, \pi)$ of STRS R is defined by $l \to r \in \mathcal{U}'(\langle u, v\rangle, \pi)$ iff one of the following condition hold:

(1) $root(v') = root(l)$ and $\tau(v') \sqsubseteq \tau(l)$ for some $v' \in Sub_\pi(v)$,

(2) $\tau(root(v')) \sqsubseteq \tau(root(l))$ and $\tau(v') \sqsubseteq \tau(l)$ for some $v' \in Sub^{int}_{\mathcal{V},\pi}(v)$, or

(3) $\tau(root(u')) \sqsubseteq \tau(l)$ for some $u' \in Sub^{int}_{\mathcal{V},\pi}(u)$ with $root(u') \in Var(v)$.

We define the set $\mathcal{U}(\langle u, v\rangle, \pi)$ by the smallest set satisfying $\mathcal{U}'(\langle u, v\rangle, \pi) \subseteq \mathcal{U}(\langle u, v\rangle, \pi)$ and $\mathcal{U}(\langle l, r\rangle, \pi) \subseteq \mathcal{U}(\langle u, v\rangle, \pi)$ whenever $l' \to r' \in \mathcal{U}(\langle u, v\rangle, \pi)$ and $l \to r \in (l' \to r')^{ex}$. For each set C of pairs of terms, we define *usable rules w.r.t. argument filtering* $\mathcal{U}(C, \pi) = \bigcup_{\langle u,v\rangle \in C} \mathcal{U}(\langle u, v\rangle, \pi)$.

In the following, we assume that R is a finitely branching STRS, C is pairs of marked terms, and $t \in \Delta$ iff $root(t) = root(l)$ and $\tau(t) \sqsubseteq \tau(l)$ for some $l \to r \in R \backslash \mathcal{U}(C, \pi)$

Lemma 5.2 For each $l \to r \in C \cup \mathcal{U}(C,\pi)^{ex}$ and θ, the following properties hold:

(1) $v\theta \notin \Delta$ for all $v \in Sub_\pi(r)$ with $root(v) \in \Sigma$,

(2) $v\theta \notin \Delta$ for all $v \in Sub^{int}_{\mathcal{V},\pi}(r)$, and

(3) $root(u)\theta \notin \Delta$ for all $u \in Sub^{int}_{\mathcal{V},\pi}(l)$ with $root(u) \in Var(r)$.

Each (i) is a directly consequence of Definition 5.1 (i).

Definition 5.3 We suppose that for each $\alpha \in \mathcal{S}$, $\bot_\alpha$ and $\mathtt{c}_\alpha$ are fresh function symbol with $\tau(\bot_\alpha) = \alpha$ and $\tau(\mathtt{c}_\alpha) = \alpha \to \alpha \to \alpha$. The interpretation I_π is a mapping from terminating terms in $\mathcal{T}_\tau(\Sigma,\mathcal{V})$ to terms in $\mathcal{T}_\tau(\Sigma \cup \bigcup_{\alpha\in\mathcal{S}}\{\bot_\alpha,\mathtt{c}_\alpha\},\mathcal{V})$; for each $t^\alpha \equiv a[t_1^{\alpha_1},\ldots,t_n^{\alpha_n}]$, $I_\pi(t)$ is defined as follows:

$$I_\pi(t) = \begin{cases} a[t'_1,\ldots,t'_n] & \text{if } t \notin \Delta \\ \mathtt{c}_\alpha[a[t'_1,\ldots,t'_n], Red_\alpha(\{I_\pi(t') \mid t \xrightarrow[R]{} t'\})] & \text{if } t \in \Delta \end{cases}$$

where $t'_i \equiv I_\pi(t_i)$ if $a \in \Sigma$ and $i \in \pi(a)$; otherwise $t'_i \equiv \bot_{\alpha_i}$, and

$$Red_\alpha(T) = \begin{cases} \bot_\alpha & \text{if } T = \emptyset \\ \mathtt{c}_\alpha[least(T), Red(T \setminus \{t\})] & \text{if } T \neq \emptyset \end{cases}$$

Thanks to the well-ordering theorem, we assume an arbitrary but fixed well-order on $\mathcal{T}_\tau(\Sigma,\mathcal{V})$. For each terminating substitution θ, we define θ_{I_π} by $\theta_{I_\pi}(x) = I_\pi(\theta(x))$ for each $x \in \mathcal{V}$.

The following two lemmas can be proved by structual induction.

Lemma 5.4 Let $l \to r \in C \cup \mathcal{U}(C,\pi)^{ex}$ and θ be a substitution such that $l\theta$ is terminating. We define σ as $\sigma(x) = u$ if $x \notin Var(r)$ and $\theta^{I_\pi}(x)$ has a form $\mathtt{c}[u,T]$; otherwise $\sigma(x) = \theta_{I_\pi}(x)$. Then we have $I_\pi(l\theta) \xrightarrow[Ce]{*} \pi(l)\sigma$,

Lemma 5.5 Let $l \to r \in C \cup \mathcal{U}(C)^{ex}$ and θ be a substitution such that $r\theta$ is terminating. Then we have $I_\pi(r\theta) \equiv \pi(r)\theta_{I_\pi}$.

Lemma 5.6 If $s \xrightarrow[R]{} t$ and s is terminating then $I_\pi(s) \xrightarrow[\mathcal{U}(C,\pi)\cup Ce]{*} I_\pi(t)$.

Proof. From Proposition 2.1, there exist a rule $l \to r \in R^{ex}$, a leaf-context $C[\,]$ and substitution θ such that $s \equiv C[l\theta]$ and $t \equiv C[r\theta]$. We prove the claim by induction on $C[\,]$. Thanks that $C[\,]$ is a leaf-context, it suffices to show the following three cases.

- Suppose that $s \notin \Delta$ and $C[\,] \equiv \Box$. Then $l \to r \in \mathcal{U}(C,\pi)^{ex}$. We define the substitution σ as similar to Lemma 5.4. From Lemma 5.4 and 5.5, we have $I_\pi(l\theta) \xrightarrow[Ce]{*} l\sigma \xrightarrow[\mathcal{U}(C)]{} r\sigma \equiv r\theta_{I_\pi} \equiv I_\pi(r\theta)$.
- Suppose that $s \notin \Delta$, $C[\,] \equiv a[\ldots,u_{i-1},C'[\,],u_{i+1},\ldots]$, $a \in \Sigma$ and $i \notin \pi(a)$. Then $t \notin \Delta$ and hence $I_\pi(C[l\theta]) \equiv a[\ldots,\bot,\ldots] \equiv I_\pi(C[r\theta])$.
- Suppose that $s \notin \Delta$, $C[\,] \equiv a[\ldots,u_{i-1},C'[\,],u_{i+1},\ldots]$, and either $a \in \mathcal{V}$ or $a \in \Sigma$ and $i \in \pi(a)$. Then $t \notin \Delta$ and hence $I_\pi(C[l\theta]) \equiv a[\ldots,I_\pi(C'[l\theta]),\ldots] \xrightarrow[\mathcal{U}(C,\pi)\cup Ce]{*} a[\ldots,I_\pi(C'[r\theta]),\ldots] \equiv I_\pi(C[r\theta])$.
- Suppose that $s \in \Delta$. Then $I_\pi(C[l\theta]) \xrightarrow[Ce]{} Red(\{I_\pi(v) \mid C[l\theta] \xrightarrow[R]{} v\}) \xrightarrow[Ce]{+} I_\pi(C[r\theta])$. □

Based on Corollary 3.12 and Proposition 3.14, we obtain the method to prove termination of SFP-STRSs.

Theorem 5.7 Let R be a left-firmness and finitely branching SFP-STRS such that there exists no infinite path in the static dependency graph. If for each $C \in SRC(R)$ there exist an argument filtering function π and a reduction order (resp. semi-reduction order) $>$ such that $C \cap > \neq \emptyset$ and $\mathcal{U}(C,\pi) \cup Ce \cup C \subseteq \gtrsim_\pi$ (resp. $(\mathcal{U}(C,\pi) \cup Ce)^{ex} \cup C \subseteq \gtrsim_\pi$), then R is terminating.

Acknowledgements

This research was partially supported by MEXT.KAKENHI #18500011, and by the Kayamori Foundation of Informational Science Advancement.

References

[1] Arts,T., Giesl,J.: Termination of Term Rewriting Using Dependency Pairs. *Theoretical Computer Science*, Vol.236, pp.133–178, 2000.

[2] Hirokawa,N., Middeldorp,A.: Dependency Pairs Revisited. In: *Proc. of the 15th Int. Conf. on Rewriting Techniques and Applications*, LNCS 3091 (RTA04), pp.249–268, 2004.

[3] Kusakari,K., Nakamura,M., Toyama,Y.: Argument Filtering Transformation. In *Proc. of Int. Conf. on Principles and Practice of Declarative Programming*, LNCS 1702 (PPDP'99), pp.47–61, 1999.

[4] Kusakari,K.: 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.

[5] Kusakari,K.: Higher-Order Path Orders based on Computability. *IEICE Transactions on Information and Systems*, Vol.E87-D, No.2, pp.352–359, 2004.

[6] Kusakari,K., Sakai,M.: Enhancing Dependency Pair Method using Strong Computability in Simply-Typed Term Rewriting Systems, *Applicable Algebra in Engineering, Communication and Computing*, Vol.18, No.5, pp.407–431, 2007.

[7] Sakurai,T., Kusakari,K., Sakai,M., Sakabe,T., Nishida,N.: Usable Rules and Labeling Product-Typed Terms for Dependency Pair Method in Simply-Typed Term Rewriting Systems. *IEICE Transactions on Information and Systems*, Vol.J90-D, No.4, pp.978–989, 2007.

[8] Terese: Term Rewriting Systems. Cambridge Tracts in Theoretical Computer Science, Vol.55, Cambridge University Press, 2003.

[9] Thiemann,R., Giesl,J., Schneider-Kamp,P.: Improved Modular Termination Proofs Using Dependency Pairs. In: *Proc. of the 2nd Int. Joint Conf. on Automated Reasoning*, LNAI 3097 (IJCAR2004), pp.75–90, 2004.