Dependency Pairs for Simply Typed Term Rewriting

Takahito Aoto[1] and Toshiyuki Yamada[2]

[1] Research Institute of Electrical Communication, Tohoku University, Japan
aoto@nue.riec.tohoku.ac.jp

[2] Faculty of Engineering, Mie University, Japan
toshi@cs.info.mie-u.ac.jp

Abstract. Simply typed term rewriting proposed by Yamada (RTA, 2001) is a framework of higher-order term rewriting without bound variables. In this paper, the dependency pair method of first-order term rewriting introduced by Arts and Giesl (TCS, 2000) is extended in order to show termination of simply typed term rewriting systems. Basic concepts such as dependency pairs and estimated dependency graph in the simply typed term rewriting framework are clarified. The subterm criterion introduced by Hirokawa and Middeldorp (RTA, 2004) is successfully extended to the case where terms of function type are allowed. Finally, an experimental result for a collection of simply typed term rewriting systems is presented. Our method is compared with the direct application of the first-order dependency pair method to a first-order encoding of simply typed term rewriting systems.

1 Introduction

Simply typed term rewriting, proposed by Yamada [20], is a framework of higher-order term rewriting. Equational specifications using higher-order functions, like functional programs, are naturally expressed in this framework. In contrast to the usual higher-order term rewriting [11, 12, 15], simply typed term rewriting dispenses with bound variables. In this respect, this framework reflects limited higher-order features. On the other hand, it is succinct and theoretically much easier to deal with.

The dependency pair method, introduced by Arts and Giesl [3], is a technique to prove the termination of (first-order) term rewriting systems. In contrast to simplification orders, such as the recursive path ordering, this method can also handle termination proofs of non-simply terminating term rewriting systems. Various improvements of the technique have been proposed [4–6, 10, 16, 19]. Implementation techniques as well as experimental results of termination provers based on the dependency pair method are reported in [6–9]. Extensions of the method to the higher-order case have been considered in [13, 17, 18].

In this paper, the dependency pair method of first-order term rewriting is extended to the case of simply typed term rewriting. We first study properties of rewrite sequences over minimal non-terminating simply typed terms on which all the dependency pair techniques are based (Section 3). Then basic concepts such as the dependency pairs (Section 4) and the estimated dependency graph (Section 5) in the simply typed term rewriting framework are clarified.

Simply typed terms may have variables as head symbols because variables of function types are allowed. Hence we need to introduce a technique to deal with dependency pairs with head variables. It turns out that head variables in dependency pairs can always be instantiated for the termination proof. As a result, the subterm criterion, introduced by Hirokawa and Middeldorp [10], is successfully incorporated into the simply typed term rewriting framework (Section 6).

Finally, an experimental result for a collection of simply typed term rewriting systems is presented. The effectiveness of our method is supported by a comparison with the direct application of the first-order dependency pair method to a first-order encoding of simply typed term rewriting systems (Section 7).

2 Preliminaries

In this section, we briefly recall the terminology and the notations of simply typed term rewriting which were introduced in [20]. Some additional definitions needed in this paper are also given here.

A *simple type* is either the *base type* o or a *function type* $\tau_1 \times \cdots \times \tau_n \to \tau_0$. When clear, simple type is abbreviated as *type*. For the sake of simplicity, we consider only the single base type, although all the results in this paper can be extended to the case of multiple base types. The sets of *constants* and *variables* of type τ are denoted by Σ^τ and V^τ, respectively. Let $\Sigma = \bigcup_\tau \Sigma^\tau$ and $V = \bigcup_\tau V^\tau$. The set of *simply typed terms* of type τ over Σ and V is denoted by $\mathrm{T}(\Sigma, V)^\tau$. We define $\mathrm{T}(\Sigma, V) = \bigcup_\tau \mathrm{T}(\Sigma, V)^\tau$. The *head symbol* of a simply typed term t is written as $\mathrm{head}(t)$.

A *context* of type τ is a simply typed term that contains one special symbol $\Box^\tau$, called the *hole*, of type τ. The term obtained by replacing the hole in a context C of type τ with a term t of the same type τ is denoted by $C[t]$. A context of the form $\Box^\tau$ is said to be *empty*. We omit the type of a hole when it is not important. A term s is a *subterm* of a term t (denoted by $s \trianglelefteq t$) if $C[s] = t$ for some context C, and is a *proper* subterm (denoted by $s \triangleleft t$) when $s \neq t$ holds in addition. For a *substitution* $\sigma : V \to \mathrm{T}(\Sigma, V)$, an *instance* $\sigma(t)$ of a term t is also written as $t\sigma$. We say s *matches* t and write $s \precsim t$ if t is an instance of s. Clearly, the relation $\precsim$ is a quasi-order. We note that when s and t do not have variables in common, the existence of a term u such that $s \precsim u \succsim t$ is equivalent to the unifiability of s and t. We use $\mathrm{mgu}(s, t)$ to denote an arbitrary partial function to compute a most general unifier of s and t.

Every *simply typed rewrite rule* $l \to r$ must satisfy the following conditions: (1) l and r have the same type, (2) $\mathrm{head}(l) \in \Sigma$, and (3) $\mathrm{V}(r) \subseteq \mathrm{V}(l)$. The type of a rewrite rule $l \to r$ is defined as that of l and r. Let $\mathcal{R} = \langle \Sigma, R \rangle$ be a *simply typed term rewriting system* (STTRS, for short). The *rewrite relation* induced by

$\mathcal{R}$ is denoted by $\to_\mathcal{R}$. The reflexive closure, transitive closure, reflexive transitive closure of $\to_\mathcal{R}$ are written as $\to^=_\mathcal{R}$, $\to^+_\mathcal{R}$, $\to^*_\mathcal{R}$, respectively. When $\mathcal{R}$ is obvious from the context, we omit the subscript. A constant $f \in \Sigma$ is a *defined symbol* of an STTRS $\mathcal{R} = \langle \Sigma, R \rangle$ if $f = \mathrm{head}(l)$ for some $l \to r \in R$, otherwise f is a *constructor symbol*. The sets of defined and constructor symbols are denoted by Σ_d and Σ_c, respectively.

Example 1 (simply typed term rewriting). Let $\mathcal{R} = \langle \Sigma, R \rangle$ be an STTRS where $\Sigma = \{\ \mathsf{0}^\mathrm{o}, \mathsf{s}^{\mathrm{o}\to\mathrm{o}}, [\,]^\mathrm{o}, :^{\mathrm{o}\times\mathrm{o}\to\mathrm{o}}, +^{\mathrm{o}\to\mathrm{o}\to\mathrm{o}}, \mathsf{fold}^{(\mathrm{o}\to\mathrm{o}\to\mathrm{o})\times\mathrm{o}\to\mathrm{o}\to\mathrm{o}}, \mathsf{sum}^{\mathrm{o}\to\mathrm{o}}\ \}$, and

$$R = \left\{ \begin{array}{lcl} (+\ \mathsf{0})\ y & \to & y \\ (+\ (\mathsf{s}\ x))\ y & \to & \mathsf{s}\ ((+\ x)\ y) \\ (\mathsf{fold}\ F\ x)\ [\,] & \to & x \\ (\mathsf{fold}\ F\ x)\ (:\ y\ ys) & \to & (F\ y)\ ((\mathsf{fold}\ F\ x)\ ys) \\ \mathsf{sum} & \to & \mathsf{fold}\ +\ \mathsf{0} \end{array} \right\}.$$

The function sum computes the sum of all elements of a list. Here is an example of a rewrite sequence of $\mathcal{R}$:

$$\begin{array}{lll} \mathsf{sum}\ (:\ (\mathsf{s}\ \mathsf{0})\ [\,]) & \to_\mathcal{R} & (\mathsf{fold}\ +\ \mathsf{0})\ (:\ (\mathsf{s}\ \mathsf{0})\ [\,]) \\ & \to_\mathcal{R} & (+\ (\mathsf{s}\ \mathsf{0}))\ ((\mathsf{fold}\ +\ \mathsf{0})\ [\,]) \\ & \to_\mathcal{R} & (+\ (\mathsf{s}\ \mathsf{0}))\ \mathsf{0} \\ & \to_\mathcal{R} & \mathsf{s}\ ((+\ \mathsf{0})\ \mathsf{0}) \\ & \to_\mathcal{R} & \mathsf{s}\ \mathsf{0}. \end{array}$$

We have $\Sigma_\mathrm{d} = \{\ +, \mathsf{fold}, \mathsf{sum}\ \}$ and $\Sigma_\mathrm{c} = \{\ \mathsf{0}, \mathsf{s}, [\,], :\ \}$.

3 Chains of minimal non-terminating terms

In this section, we study properties of minimal non-terminating terms, on which all the dependency pair techniques are based. Basic notions and properties of infinite rewrite sequences are lifted to the case of simply typed term rewriting. Since root symbols in first-order term rewriting correspond to head symbols in simply typed term rewriting, we now regard head rewrite steps instead of root rewrite steps.

Definition 1 (root and head rewrite steps). We write $s \xrightarrow{\mathrm{r}} t$ when $s = l\sigma$ and $t = r\sigma$ for some rewrite rule $l \to r$ and some substitution σ. The relation $\xrightarrow{\mathrm{r}}$ is called the *root rewrite step*. The *head rewrite step* $\xrightarrow{\mathrm{h}}$ is defined recursively as follows: $s \xrightarrow{\mathrm{h}} t$ if (1) $s \xrightarrow{\mathrm{r}} t$, or (2) $s = (s_0\ u_1\ \cdots\ u_n)$, $t = (t_0\ u_1\ \cdots\ u_n)$, and $s_0 \xrightarrow{\mathrm{h}} t_0$. The non-root and non-head rewrite steps are defined by $\xrightarrow{\mathrm{nr}} = \to \setminus \xrightarrow{\mathrm{r}}$ and $\xrightarrow{\mathrm{nh}} = \to \setminus \xrightarrow{\mathrm{h}}$, respectively.

By definition, $\xrightarrow{\mathrm{r}} \subseteq \xrightarrow{\mathrm{h}}$ and thus $\xrightarrow{\mathrm{nh}} \subseteq \xrightarrow{\mathrm{nr}}$ holds. If there are no terms of function types which can be rewriten, then the reverse inclusions also hold.

Example 2 (root and head rewrite step). Let $\mathcal{R}$ be the STTRS in Example 1. The following rewrite sequence exemplifies the use of the relations defined above:

$$\begin{aligned}
\underline{\mathsf{sum}}\ (:\ (\mathsf{s}\ 0)\ [\,]) &\xrightarrow{\mathrm{h}} \underline{(\mathsf{fold}\ +\ 0)\ (:\ (\mathsf{s}\ 0)\ [\,])}\\
&\xrightarrow{\mathrm{r}} (+\ (\mathsf{s}\ 0))\ \underline{((\mathsf{fold}\ +\ 0)\ [\,])}\\
&\xrightarrow{\mathrm{nh}} \underline{(+\ (\mathsf{s}\ 0))\ 0}\\
&\xrightarrow{\mathrm{r}} \mathsf{s}\ \underline{((+\ 0)\ 0)}\\
&\xrightarrow{\mathrm{nh}} \mathsf{s}\ 0.
\end{aligned}$$

We note that $\mathsf{sum}\ (:\ (\mathsf{s}\ 0)\ [\,]) \xrightarrow{\mathrm{nr}} (\mathsf{fold}\ +\ 0)\ (:\ (\mathsf{s}\ 0)\ [\,])$.

We say a term is *terminating* if there is no infinite rewrite sequence starting from that term, otherwise *non-terminating*. We denote the set of non-terminating terms by $\mathrm{NT}(\mathcal{R})$. The set of minimal (with respect to the subterm relation $\trianglelefteq$) non-terminating terms is denoted by $\mathrm{NT}_{\min}(\mathcal{R})$. We may omit the parameter $\mathcal{R}$ of $\mathrm{NT}(\mathcal{R})$ and $\mathrm{NT}_{\min}(\mathcal{R})$ when it is clear from the context. We say an STTRS $\mathcal{R}$ is terminating if all terms are terminating. Since the subterm relation is well-founded, non-termination implies the existence of a minimal non-terminating term.

Lemma 1 (chain of minimal non-terminating terms). Let $s_0 \in \mathrm{NT}_{\min}$. (1) There exist $s_1, \ldots, s_n, t \in \mathrm{NT}_{\min}$ $(n \geq 0)$ and $s_{n+1} \in \mathrm{NT}$ such that $s_0 \xrightarrow{\mathrm{nr}} s_1 \xrightarrow{\mathrm{nr}} \cdots \xrightarrow{\mathrm{nr}} s_n \xrightarrow{\mathrm{r}} s_{n+1} \trianglerighteq t$. (2) There exist $s_1, \ldots, s_n, t \in \mathrm{NT}_{\min}$ $(n \geq 0)$ and $s_{n+1} \in \mathrm{NT}$ such that $s_0 \xrightarrow{\mathrm{nh}} s_1 \xrightarrow{\mathrm{nh}} \cdots \xrightarrow{\mathrm{nh}} s_n \xrightarrow{\mathrm{h}} s_{n+1} \trianglerighteq t$, and $s_{n+1} = t$ whenever $s_n \xrightarrow{\mathrm{nr}} s_{n+1}$.

Proof. (1) The proof proceeds in the same way as the first-order case [3]. (2) By (1), there exist $s_1, \ldots, s_m, u \in \mathrm{NT}_{\min}$ $(m \geq 0)$ and $s_{m+1} \in \mathrm{NT}$ such that $s_0 \xrightarrow{\mathrm{nr}} s_1 \xrightarrow{\mathrm{nr}} \cdots \xrightarrow{\mathrm{nr}} s_m \xrightarrow{\mathrm{r}} s_{m+1} \trianglerighteq u$. Since $\xrightarrow{\mathrm{r}} \subseteq \xrightarrow{\mathrm{h}}$, there exists at least one $i \in \{0, \ldots, m\}$ such that $s_i \xrightarrow{\mathrm{h}} s_{i+1}$. Let n be the smallest such i. We take $t = u$ if $n = m$, and take $t = s_{n+1}$ otherwise. $\square$

We are now going to give a more detailed description of Lemma 1 (2). For that purpose, we need to know some properties of head and non-head rewrite steps. We first introduce a notion of argument context in order to characterize a head rewrite step as a rewrite step accompanied by an argument context. We next introduce a notion of argument sequence in order to deal with properties of non-head rewrite steps.

Definition 2 (argument context). An *argument context*[1] is a context whose head symbol is the hole, more precisely, κ is an argument context of type τ if (1) $\kappa = \square^{\tau}$, or (2) $\kappa = (\kappa'\ t_1 \cdots t_n)$ for some argument context κ' of a function type and some terms $t_1, \ldots, t_n$ of appropriate types.

[1] A similar notion is called a *suffix context* in [13].

Example 3 (argument context). $\kappa = ((\Box^{o\to(o\to o)}\ \mathsf{0})\ (\mathsf{s}\ \mathsf{0}))$ is an argument context. Then we have $\kappa[+] = ((+\ \mathsf{0})\ (\mathsf{s}\ \mathsf{0}))$. Neither $((+\ \Box^{o})\ (\mathsf{s}\ \mathsf{0}))$ nor $((+\ \mathsf{0})\ \Box^{o})$ is an argument context.

Lemma 2 (characterization of head rewrite steps). $s \xrightarrow{\mathrm{h}} t$ if and only if there exist a rewrite rule $l \to r$, a substitution σ, and an argument context κ such that $s = \kappa[l\sigma]$ and $t = \kappa[r\sigma]$.

Proof. ($\Rightarrow$) By induction on the definition of $\xrightarrow{\mathrm{h}}$. ($\Leftarrow$) By induction on κ. $\Box$

Lemma 3 (preservation of head symbols). (1) For any argument context κ, $\mathrm{head}(\kappa[t]) = \mathrm{head}(t)$. (2) If $s \xrightarrow{\mathrm{h}} t$ then $\mathrm{head}(s) \in \Sigma_{\mathrm{d}}$. (3) If $s \xrightarrow{\mathrm{nh}} t$ then $\mathrm{head}(s) = \mathrm{head}(t)$. (4) If $s \to t$ and $\mathrm{head}(s) \notin \Sigma_{\mathrm{d}}$ then $s \xrightarrow{\mathrm{nh}} t$ and $\mathrm{head}(t) \notin \Sigma_{\mathrm{d}}$.

Proof. (1) By induction on κ. (2) By (1) and Lemma 2. (3) By induction on s. (4) Follows immediately from (2) and (3). $\Box$

Definition 3 (argument sequence). An *argument sequence* $\mathrm{Arg}(t)$ of a simply typed term t is the empty sequence when $t \in V \cup \Sigma$; and is $\mathrm{Arg}(t_0), t_1, \ldots, t_n$ when $t = (t_0\ t_1 \cdots t_n)$.

We write the length and the i-th element of a sequence X by $|X|$ and $X_{|i}$.

Example 4 (argument sequence). Let $t = ((\mathsf{fold}\ +\ (\mathsf{s}\ \mathsf{0}))\ (:\ (\mathsf{s}\ \mathsf{0})\ [\,]))$. The argument sequence of t is $+, (\mathsf{s}\ \mathsf{0}), (:\ (\mathsf{s}\ \mathsf{0})\ [\,])$. We also have $|\mathrm{Arg}(t)| = 3$ and $\mathrm{Arg}(t)_{|2} = (\mathsf{s}\ \mathsf{0})$.

Lemma 4 (argument sequence of non-head rewrite steps). Suppose $s \xrightarrow{\mathrm{nh}} t$. (1) $|\mathrm{Arg}(s)| = |\mathrm{Arg}(t)| > 0$. (2) Let $|\mathrm{Arg}(s)| = |\mathrm{Arg}(t)| = n$. Then there exists some $i \in \{1, \ldots, n\}$ such that $\mathrm{Arg}(s)_{|i} \to \mathrm{Arg}(t)_{|i}$ and $\mathrm{Arg}(s)_{|j} = \mathrm{Arg}(t)_{|j}$ for all $j \in \{1, \ldots, n\} \setminus \{i\}$.

Proof. (1) The contraposition can be shown by induction on s. (2) By induction on s using (1). $\Box$

Lemma 5 (head symbols of minimal non-terminating terms). If $s \in \mathrm{NT}_{\mathrm{min}}$ then $\mathrm{head}(s) \in \Sigma_{\mathrm{d}}$.

Proof. Suppose $s \in \mathrm{NT}_{\mathrm{min}}$. Then there exists an infinite rewrite sequence $s = s_0 \to s_1 \to \cdots$. For a proof by contradiction, suppose $\mathrm{head}(s) \notin \Sigma_{\mathrm{d}}$. By repeatedly applying Lemma 3 (4), we know $s_0 \xrightarrow{\mathrm{nh}} s_1 \xrightarrow{\mathrm{nh}} s_2 \xrightarrow{\mathrm{nh}} \cdots$. By Lemma 4 (1), all $\mathrm{Arg}(s_k)$ have the same length for $k \geq 0$; so, let $n = |\mathrm{Arg}(s_k)|$. Then by Lemma 4 (2), for each $k \geq 0$ there exists $i \in \{1, \ldots, n\}$ such that $\mathrm{Arg}(s_k)_{|i} \to \mathrm{Arg}(s_{k+1})_{|i}$ and $\mathrm{Arg}(s_k)_{|j} = \mathrm{Arg}(s_{k+1})_{|j}$ for all $j \in \{1, \ldots, n\} \setminus \{i\}$. Thus there exists an index i such that $\mathrm{Arg}(s_0)_{|i} \to^{=} \mathrm{Arg}(s_1)_{|i} \to^{=} \cdots$ is an infinite rewrite sequence, contradicting the minimality of s. $\Box$

Lemma 6 (chain of minimal non-terminating terms in detail). If $s \in \mathrm{NT}_{\mathrm{min}}$ then there exist a rewrite rule $l \to r$, an argument context κ, a substitution σ, and simply typed terms $r' \trianglelefteq r$ and $t \in \mathrm{NT}_{\mathrm{min}}$ satisfying the following properties: (1) $s \xrightarrow{\mathrm{nh}}{}^{*} \kappa[l\sigma] \xrightarrow{\mathrm{h}} \kappa[r\sigma] \trianglerighteq t$, (2) if κ is empty then $l \not\trianglerighteq r'$ and $t = r'\sigma$ hold, otherwise $r = r'$ and $t = \kappa[r\sigma]$ hold, (3) $\mathrm{head}(r'\sigma) \in \Sigma_{\mathrm{d}}$.

Proof. By Lemma 1, there exist terms $s', t \in \mathrm{NT}_{\mathrm{min}}$ and $u \in \mathrm{NT}$ such that $s \xrightarrow{\mathrm{nh}}{}^{*} s' \xrightarrow{\mathrm{h}} u \trianglerighteq t$, and $u = t$ whenever $s' \xrightarrow{\mathrm{nr}} u$. In order to prove (1) and (2), first suppose $s' \xrightarrow{\mathrm{nr}} u$. By Lemma 2, there exist a rewrite rule $l \to r$, an argument context κ, and a substitution σ such that $s' = \kappa[l\sigma]$ and $u = \kappa[r\sigma]$. Thus (1) holds. By the definition of $\xrightarrow{\mathrm{nr}}$, κ is nonempty. Since $u = t$, (2) is satisfied by taking $r' = r$. Next, suppose $s' \xrightarrow{\mathrm{r}} u$. Then, there exist a rewrite rule $l \to r$, a substitution σ such that $s' = l\sigma$ and $u = r\sigma$. Hence (1) is satisfied by taking κ to be an empty context. No subterm of $\sigma(x)$ can be t for any $x \in \mathrm{V}(r)$, for otherwise, t would be a proper subterm of $l\sigma$ by $\mathrm{V}(r) \subseteq \mathrm{V}(l)$ and $l \notin V$, which contradicts the minimality of $l\sigma(= s')$. Hence $t = r'\sigma$ for some $r' \trianglelefteq r$. Moreover $l \not\trianglerighteq r'$ holds, for otherwise, we would obtain $l\sigma \triangleright r'\sigma = t$, contradicting again the minimality of $l\sigma$. This completes the proof of (2). Finally, (3) is obtained from (2) by using Lemma 5 and Lemma 3 (1). □

4 Dependency pairs

As shown in Lemma 6, every minimal non-terminating sequence contains infinitely many rewrite steps each accompanied by an argument context. To treat instances of a rewrite rule and instances of a rewrite rule within an argument context in a uniform way, we introduce a notion of argument expansion of a rewrite rule.

Definition 4 (argument expansion). The *argument expansion* $\mathrm{Exp}(l \to r)$ of a rewrite rule $l \to r$ is defined recursively as follows: if $l \to r$ is of base type then $\mathrm{Exp}(l \to r) = \emptyset$, otherwise $\mathrm{Exp}(l \to r) = \{l' \to r'\} \cup \mathrm{Exp}(l' \to r')$ where $l' = (l\ x_1\ \cdots\ x_n)$, $r' = (r\ x_1\ \cdots\ x_n)$, and $x_1, \cdots, x_n$ are distinct fresh variables of appropriate types.

We are now ready to give an extended definition of dependency pairs.

Definition 5 (dependency pairs). The set $\mathrm{DP}(\mathcal{R})$ of *dependency pairs* of an STTRS $\mathcal{R} = \langle \Sigma, R \rangle$ is defined as follows:

$$\begin{aligned}\mathrm{DP}(\mathcal{R}) = {} & \{\langle l\,, r' \rangle \mid l \to r \in R,\ r' \trianglelefteq r,\ r' \not\trianglelefteq l,\ \mathrm{head}(r') \in \Sigma_{\mathrm{d}} \cup V\} \\ & \cup \{\langle l', r' \rangle \in \mathrm{Exp}(l \to r) \mid l \to r \in R,\ \mathrm{head}(r) \in \Sigma_{\mathrm{d}} \cup V\}\end{aligned}$$

A dependency pair $\langle l, r \rangle$ is also written as $l \rightarrowtail r$. We abbreviate $\mathrm{DP}(\mathcal{R})$ as DP when $\mathcal{R}$ is clear from the context.

Note that head variables are allowed in the right-hand sides of dependency pairs. If no right-hand side of the rewrite rules in $\mathcal{R}$ has a head variable of a function type, then the first component of the definition of $\mathrm{DP}(\mathcal{R})$ is the same as the first-order dependency pairs. Moreover, if all the rewrite rules are of base type, then the second component is empty.

Example 5 (dependency pairs). Let $\mathcal{R}$ be the STTRS of Example 1. The set of dependency pairs of $\mathcal{R}$ is as follows:

$$\mathrm{DP}(\mathcal{R}) = \left\{ \begin{array}{lll} (1) & (+\ (\mathsf{s}\ x))\ y & \rightarrowtail\ (+\ x)\ y \\ (2) & (+\ (\mathsf{s}\ x))\ y & \rightarrowtail\ +\ x \\ (3) & (\mathsf{fold}\ F\ x)\ (:\ y\ ys) & \rightarrowtail\ (F\ y)\ ((\mathsf{fold}\ F\ x)\ ys) \\ (4) & (\mathsf{fold}\ F\ x)\ (:\ y\ ys) & \rightarrowtail\ F\ y \\ (5) & (\mathsf{fold}\ F\ x)\ (:\ y\ ys) & \rightarrowtail\ (\mathsf{fold}\ F\ x)\ ys \\ (6) & \mathsf{sum} & \rightarrowtail\ \mathsf{fold} + \mathsf{0} \\ (7) & \mathsf{sum} & \rightarrowtail\ \mathsf{fold} \\ (8) & \mathsf{sum} & \rightarrowtail\ + \\ (9) & \mathsf{sum}\ x & \rightarrowtail\ (\mathsf{fold} + \mathsf{0})\ x \end{array} \right\}.$$

Neither $+\ \mathsf{0}\ y \rightarrowtail y$ nor $(+\ (\mathsf{s}\ x))\ y \rightarrowtail +$ is a dependency pair of $\mathcal{R}$ because each right-hand side is a proper subterm of the corresponding left-hand side.

We introduce a new relation which expresses a root rewrite step using a dependency pair, because it is not in general type-preserving.

Definition 6 (dependency relation). The *dependency relation* $\rightarrowtail_D$ of a set of dependency pairs D is defined as follows: $s \rightarrowtail_D t$ if there exist a dependency pair $l \to r \in D$ and a substitution σ such that $s = l\sigma$, $t = r\sigma$ and $\mathrm{head}(t) \in \Sigma_{\mathrm{d}}$.

Lemma 7 (chain of minimal non-terminating terms via dependency relation). If $s \in \mathrm{NT}_{\min}$ then $s \overset{\mathrm{nh}}{\to}{}^{*} \cdot \rightarrowtail_{\mathrm{DP}} t$ for some $t \in \mathrm{NT}_{\min}$.

Proof. By Lemma 6 and the definition of dependency pairs. □

The next lemma gives a characterization of termination in terms of the dependency relation.

Lemma 8 (termination by dependency pairs). An STTRS is terminating if and only if the relation $\overset{\mathrm{nh}}{\to}{}^{*} \cdot \rightarrowtail_{\mathrm{DP}}$ on $\mathrm{NT}_{\min}$ is terminating.

Proof. ($\Rightarrow$) The contraposition can be shown by using the inclusions $\overset{\mathrm{nh}}{\to}{}^{*} \cdot \rightarrowtail_{\mathrm{DP}} \subseteq \to^{+} \cdot \trianglerighteq$ and $\trianglerighteq \cdot \to^{+} \subseteq \to^{+} \cdot \trianglerighteq$. ($\Leftarrow$) The contraposition can be shown by using Lemma 7 and the existence of a (minimal) non-terminating term. □

Definition 7 (dependency chain). Let D be a set of dependency pairs of an STTRS $\mathcal{R}$. A *dependency chain* of D is an infinite sequence $t_0, t_1, \ldots$ on $\mathrm{NT}_{\min}(\mathcal{R})$ such that $t_i \overset{\mathrm{nh}}{\to}{}^{*} \cdot \rightarrowtail_D t_{i+1}$ for all $i \geq 0$. The family of all sets of dependency pairs that admit dependency chain is denoted by $\mathbf{DC}(\mathcal{R})$. We define $\mathbf{DC}_{\min}(\mathcal{R}) = \{D \in \mathbf{DC}(\mathcal{R}) \mid D' \notin \mathbf{DC}(\mathcal{R})$ for any $D' \subsetneq D\ \}$. As usual, we omit the parameter $\mathcal{R}$ when it is clear from the context.

We now come to the simply typed version of Theorem 6 of [3].

Theorem 1 (termination by dependency chains). An STTRS is terminating if and only if $\mathbf{DC} = \emptyset$.

Proof. By Lemma 8 and the definition of a dependency chain. $\square$

Corollary 1 (termination by $\mathbf{DC}_{\min}$). Let $\mathcal{R}$ be an STTRS such that DP is finite. Then $\mathcal{R}$ is terminating if and only if $\mathbf{DC}_{\min} = \emptyset$.

Proof. ($\Rightarrow$) By Theorem 1 and the inclusion $\mathbf{DC}_{\min} \subseteq \mathbf{DC}$. ($\Leftarrow$) By Theorem 1 and the well-foundedness of the inclusion relation on the family of finite sets. $\square$

The following property of $\mathbf{DC}_{\min}$ is often used in the subsequent sections.

Lemma 9 (characterization of $\mathbf{DC}_{\min}$). Let D be a set of dependency pairs. $D \in \mathbf{DC}_{\min}$ if and only if $D \in \mathbf{DC}$ and for any dependency chain $t_0, t_1, \ldots$ of D and any $l \rightarrowtail r \in D$ there exist infinitely many i such that $t_i \xrightarrow{\text{nh}}{}^{*} \cdot \rightarrowtail_{\{l \rightarrowtail r\}} t_{i+1}$.

Proof. ($\Rightarrow$) Suppose $D \in \mathbf{DC}_{\min}$. If some dependency pair $l \rightarrowtail r \in D$ were used only finitely many times, then we would have $D \setminus \{l \rightarrowtail r\} \in \mathbf{DC}$, which contradicts the minimality of D. ($\Leftarrow$) For a proof by contradiction, suppose there exists $D' \subsetneq D$ such that $D' \in \mathbf{DC}$. Then there exists a dependency chain of D', which is also a dependency chain of D since $D' \subseteq D$. No dependency pairs in $D \setminus D'$ are used in this chain. Hence any $l \rightarrowtail r \in D \setminus D'$ can be used to obtain the desired contradiction. $\square$

5 Dependency graph

The dependency graph is useful to find sets of dependency pairs from which dependency chains may arise. Any dependency chain corresponds to a path of the dependency graph. One can prove absence of dependency chains by showing that for every cycle in the graph there is no dependency chain corresponding to that cycle.

Definition 8 (dependency graph). Let $\mathcal{R}$ be an STTRS and D be a set of dependency pairs of $\mathcal{R}$. The *dependency graph* $\mathrm{DG}_{\mathcal{R}}(D)$ is the directed graph whose set of vertices is D in which $\langle l \rightarrowtail r, l' \rightarrowtail r' \rangle$ is an edge if and only if there exist substitutions σ, σ' such that $r\sigma \xrightarrow{\text{nh}}{}^{*} l'\sigma'$.

Since the edges of a dependency graph are not computable in general, its computable estimation needs to be considered. The most basic estimation in the first-order setting is the one using REN $\circ$ CAP and unifiability [3]. We are now going to extend this estimation for the simply typed framework.

Recall that in simply typed term rewriting, there may be a term consisting of a head symbol and partially applied arguments which cannot be rewritten until the remaining arguments are applied. To estimate such partially applied terms, we introduce a notion of pattern.

Definition 9 (defined and undefined pattern). The *pattern* pat(t) of a term t is the term obtained from t by replacing its arguments with fresh variables, recursively defined as follows: (1) $\mathrm{pat}(t) = t$ if $t \in \Sigma \cup V$, and (2) $\mathrm{pat}(t) = (\mathrm{pat}(t_0)\ x_1\ \cdots x_n)$ if $t = (t_0\ t_1 \cdots t_n)$ where $x_1, \ldots, x_n$ are distinct and fresh variables of appropriate types. A *subpattern* of t is defined as follows: (1) $\mathrm{pat}(t)$ is a subpattern of t, (2) if $t = (t_0\ t_1 \cdots t_n)$ then any subpattern of t_0 is a subpattern of t. A term t has a *defined* pattern if either (1) $\mathrm{head}(t) \in V$ or (2) $\mathrm{head}(t) \in \Sigma_{\mathrm{d}}$ and there exists a subpattern p of t such that $p \precsim l$ for some $l \to r \in R$; otherwise, it has an *undefined* pattern.

Example 6 (defined and undefined patterns). Let $\mathcal{R}$ be the STTRS in Example 1. Let $s = ((\mathsf{fold} + ((+\ x)\ \mathsf{0}))\ [\,])$. Then s has a pattern $((\mathsf{fold}\ F\ y)\ z)$; its subpatterns are $((\mathsf{fold}\ F\ y)\ z)$, $(\mathsf{fold}\ F\ y)$, and fold. The term t has a defined pattern, while $(\mathsf{fold} + ((+\ x)\ \mathsf{0}))$ has an undefined pattern.

Lemma 10 (properties of patterns). Let s and t be terms. (1) $\mathrm{pat}(s) \precsim s$. (2) If $s = \kappa[t]$ for some argument context κ and p is a subpattern of t then p is a subpattern of s. (3) If $\mathrm{head}(s) \in \Sigma$, then, for any substitution σ, $\mathrm{pat}(s) = \mathrm{pat}(s\sigma)$ (up to the renaming of fresh variables).

Proof. (1) By induction on s. (2) By induction on κ. (3) By induction on s. ☐

Lemma 11 (properties of defined and undefined patterns). (1) If $s = (s_0\ s_1 \cdots s_n)$ has an undefined pattern then s_0 has an undefined pattern. (2) If s has an undefined pattern, then for any substitution σ, $s\sigma$ has an undefined pattern. (3) For any rewrite rule $l \to r$ and any substitution σ, $l\sigma$ has a defined pattern. (4) If s has a defined pattern and $C[s]$ has an undefined pattern then $C[x]$ has an undefined pattern where $x \in V$.

An approximation of a term is now defined by using the notion of pattern.

Definition 10 (approximation approx**).** Define $\mathrm{approx}(t) = \mathrm{REN}(\mathrm{CAP}(t))$. Here, CAP replaces every outermost subterm in any argument which have a defined pattern by a fresh variable of appropriate type. REN replaces all occurrences of variables but the head occurrence[2] with distinct fresh variables.

Example 7 (approximation approx*).* Let $s = ((\mathsf{fold}\ +\ ((+\ x)\ y))\ (\mathsf{s}\ \mathsf{0}))$. Then $\mathrm{approx}(s) = ((\mathsf{fold}\ +\ z)\ (\mathsf{s}\ \mathsf{0}))$. Let $t = (F\ (G\ ((+\ x)\ y)))$. Then $\mathrm{approx}(t) = (F\ z)$.

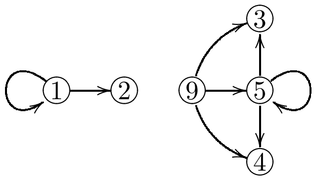
Note that our CAP keeps a subterm with a defined head symbol if none of its instance can be rewriten by a head step because of the lack of arguments. This is contrasted with the CAP for first-order terms, which replaces every subterm with a defined root symbol by a variable. Our CAP is the same as the original first-order definition when it is applied to a first-order term.

[2] This is not at all essential, but it turns out to be useful to define *head instantiation* later.

Lemma 12 (properties of CAP **and** REN**).** (1) $\mathrm{CAP}(s) \precsim s$ and $\mathrm{REN}(s) \precsim s$ for any term s. (2) $\mathrm{CAP}(s) \precsim \mathrm{CAP}(s\sigma)$ and $\mathrm{REN}(s) \precsim \mathrm{REN}(s\sigma)$ for any term s and substitution σ. (3) For any $l \to r \in R$, $x \in V$, a context C, and a substitution σ, $\mathrm{CAP}(C[l\sigma]) \precsim \mathrm{CAP}(C[x])$.

Definition 11 (estimated dependency graph). Let $\mathcal{R}$ be an STTRS and D a set of dependency pairs of $\mathcal{R}$. The *estimated dependency graph* $\mathrm{EDG}_{\mathcal{R}}(D)$ is the directed graph whose set of vertices is D in which $\langle l \rightarrowtail r, l' \rightarrowtail r' \rangle$ is an edge if and only if $\mathrm{approx}(r)$ and l' are unifiable. The estimated dependency graph $\mathrm{EDG}_{\mathcal{R}}(\mathrm{DP}(\mathcal{R}))$ is abbreviated as $\mathrm{EDG}(\mathcal{R})$.

Example 8 (estimated dependency graph). Let $\mathcal{R}$ be the STTRS of Example 1. The dependency pairs of $\mathcal{R}$ are presented in Example 5. The estimated dependency graph $\mathrm{EDG}(\mathcal{R})$ is shown below.



Lemma 13 (soundness of estimation). For any terms s, t and substitutions σ, τ, if $s\sigma \overset{\mathrm{nh}}{\to}{}^{*} t\tau$ then $\mathrm{approx}(s)$ and t are unifiable.

Proof. Let $u \to_{\mathcal{R}} v$ with $s, t \in \mathrm{T}(\Sigma, V)$. Then $u = C[l\sigma]$ and $C[r\sigma] = v$ for some rewrite rule $l \to r$, substitution σ and a context C such that $\mathrm{head}(C) \in \Sigma_{\mathrm{d}} \cup V$. Then by (2) and (3) of Lemma 12, $\mathrm{approx}(u) \precsim \mathrm{approx}(v)$. Thus, by induction on the length of the reduction, it easily follows $s\sigma \overset{\mathrm{nh}}{\to}{}^{*} t\tau$ implies $\mathrm{approx}(s\sigma) \precsim \mathrm{approx}(t\tau)$. Then by (1) and (2) of Lemma 12, $\mathrm{approx}(s) \precsim \mathrm{approx}(s\sigma) \precsim \mathrm{approx}(t\tau) \precsim t\tau \succsim t$. Since variables in $\mathrm{approx}(s)$ and t are disjoint, $\mathrm{approx}(s)$ and t are unifiable. $\square$

Theorem 2 (approximation by estimated dependency graph). Let D be a finite set of dependency pairs of an STTRS $\mathcal{R}$. If $D' \subseteq D$ and $D' \in \mathbf{DC}_{\min}(\mathcal{R})$, then for any vertices $u, v \in D'$ there exists a nonempty path in D' from u to v.

Proof. By Lemma 9, Lemma 13, and the definition of an estimated dependency graph. $\square$

6 Subterm criterion and head instantiation

Simple projections and the subterm criterion have been introduced by Hirokawa and Middeldorp [10] to show the absence of dependency chains. We first extend these notions to the case of simply typed term rewriting. A simple projection maps a term to one of its arguments according to its head symbol. In the simply typed framework, a simple projection may not be well-defined because of partially applied arguments and head variables. For this reason, we have to speak of *feasible* simple projections. The head instantiation technique is introduced in order to handle dependency pairs with head variables.

Definition 12 (feasible simple projection). A *simple projection* is a mapping which maps each defined symbol to a positive integer. A simple projection π is *feasible* for a term t if (1) $\mathrm{head}(t) \in \Sigma_{\mathrm{d}}$ and (2) $\pi(\mathrm{head}(t)) \leq |\mathrm{Arg}(t)|$. In that case, we define $\pi(t) = \mathrm{Arg}(t)_{|\pi(\mathrm{head}(t))}$. A simple projection π is feasible for a set D of dependency pairs if it is feasible for l and r for any $l \rightarrowtail r \in D$. In that case, we define $\pi(D) = \{\pi(l) \rightarrowtail \pi(r) \mid l \rightarrowtail r \in D\}$.

Example 9 (feasible simple projection). Let π_1 be a simple projection satisfying $\pi_1(+) = 1$. Then π_1 is feasible for $((+\ (\mathsf{s}\ \mathsf{0}))\ \mathsf{0})$ and for $(+\ (\mathsf{s}\ \mathsf{0}))$: we have $\pi_1((+\ (\mathsf{s}\ \mathsf{0}))\ \mathsf{0}) = \pi_1(+\ (\mathsf{s}\ \mathsf{0})) = (\mathsf{s}\ \mathsf{0})$. Let π_2 be a simple projection satisfying $\pi_2(+) = 2$. Then π_2 is feasible for $((+\ (\mathsf{s}\ \mathsf{0}))\ \mathsf{0})$ but not for $(+\ (\mathsf{s}\ \mathsf{0}))$: the latter lacks the second argument.

Definition 13 (subterm criterion). Let D be a set of dependency pairs. We say that D satisfies the *subterm criterion* if there exists a simple projection π such that π is feasible for D, $\pi(D) \subseteq \trianglerighteq$, and $\pi(D) \cap \rhd \neq \emptyset$.

Theorem 3 (absence of dependency chains by subterm criterion). If a set of dependency pairs D satisfies the subterm criterion, then $D \notin \mathbf{DC}_{\mathrm{min}}$.

Proof. The proof proceeds in the same way as for first-order TRSs (see [10]). □

Example 10 (termination by subterm criterion). Let $\mathcal{R}$ be the STTRS of Example 1. $\mathrm{DP}(\mathcal{R})$ and $\mathrm{EDG}(\mathcal{R})$ are presented in Examples 5 and 8, respectively. Let π be a simple projection satisfying $\pi(+) = 1$ and $\pi(\mathsf{fold}) = 3$. Then $\pi((1)), \pi((5)) \in \rhd$ and hence $\{(1)\}, \{(5)\} \notin \mathbf{DC}_{\mathrm{min}}$. Therefore $\mathcal{R}$ is terminating.

A simple projection is infeasible for a set of dependency pairs if some right-hand side has a variable as its head symbol. To deal with such a case, we introduce a head instantiation technique. Some instantiation techniques for first-order dependency pairs are discussed in [4, 6, 7].

Definition 14 (head instantiation). Let $l \rightarrowtail r$ and $l' \rightarrowtail r'$ be dependency pairs such that $\mathrm{head}(r) \in V$. Moreover, we assume that $l \rightarrowtail r$ and $l' \rightarrowtail r'$ do not share any variables in common (after renaming). Suppose that $\mathrm{approx}(r)$ and l' are unifiable and let $\sigma = \mathrm{mgu}(\mathrm{approx}(r), l')$. The *head substitution* of $l \rightarrowtail r$ for $l' \rightarrowtail r'$ is defined as $\{F \mapsto \mathrm{pat}(\sigma(F))\}$ where $F = \mathrm{head}(r) = \mathrm{head}(\mathrm{approx}(r))$. The set $\mathrm{HS}_D(l \rightarrowtail r)$ of head substitutions of $l \rightarrowtail r$ for a set D of dependency pairs is the singleton set of the identity substitution if $\mathrm{head}(r) \in \Sigma_{\mathrm{d}}$ and is the set $\{\sigma \mid \sigma \text{ is the head substitution of } l \rightarrowtail r \text{ for some } l' \rightarrowtail r' \in D\}$ if $\mathrm{head}(r) \in V$. The *head instantiation* $\mathrm{HI}(D)$ of a set D of dependency pairs is defined by:

$$\mathrm{HI}(D) = \bigcup_{l \rightarrowtail r \in D} \{l\sigma \rightarrowtail r\sigma \mid \sigma \in \mathrm{HS}_D(l \rightarrowtail r)\}.$$

Theorem 4 (soundness and completeness of head instantiation). Let D be a set of dependency pairs. Then $\mathrm{HI}(D) \in \mathbf{DC}$ if and only if $D \in \mathbf{DC}$.

Proof. (⇒) By the inclusion $\rightarrowtail_{\mathrm{HI}(D)} \subseteq \rightarrowtail_D$. (⇐) By the fact that $s \rightarrowtail_D t \overset{\mathrm{nh}}{\rightarrow}{}^{*} \cdot \rightarrowtail_D u$ implies $s \rightarrowtail_{\mathrm{HI}(D)} t$.

Example 11 (Termination by head instantiation). Let $\mathcal{R} = \langle \Sigma, R \rangle$ be an STTRS where $\Sigma = \{ \ [\,]^{\mathrm{o}},\ \ :^{\mathrm{o}\times\mathrm{o}\to\mathrm{o}},\ \ \mathsf{map}^{(\mathrm{o}\to\mathrm{o})\times\mathrm{o}\to\mathrm{o}},\ \ \mathsf{pair}^{\mathrm{o}\times\mathrm{o}\to\mathrm{o}},\ \ \mathsf{append}^{\mathrm{o}\times\mathrm{o}\to\mathrm{o}},\ \mathsf{curry}^{(\mathrm{o}\times\mathrm{o}\to\mathrm{o})\to\mathrm{o}\to\mathrm{o}},\ \mathsf{cartprod}^{\mathrm{o}\times\mathrm{o}\to\mathrm{o}}\ \}$, and

$$R = \left\{ \begin{array}{lll} \mathsf{append}\ [\,]\ xs & \to & xs \\ \mathsf{append}\ (:\ x\ xs)\ ys & \to & :\ x\ (\mathsf{append}\ xs\ ys) \\ \mathsf{map}\ F\ [\,] & \to & [\,] \\ \mathsf{map}\ F\ (:\ x\ xs) & \to & :\ (F\ x)\ (\mathsf{map}\ F\ xs) \\ ((\mathsf{curry}\ G)\ x)\ y & \to & G\ x\ y \\ \mathsf{cartprod}\ [\,]\ ys & \to & [\,] \\ \mathsf{cartprod}\ (:\ x\ xs)\ ys & & \\ \multicolumn{3}{r}{\to\ \ \mathsf{append}\ (\mathsf{map}\ ((\mathsf{curry}\ \mathsf{pair})\ x)\ ys)\ (\mathsf{cartprod}\ xs\ ys)} \end{array} \right\}.$$

The function $\mathsf{cartprod}$ computes the cartesian product of two lists. For example, $\mathsf{cartprod}\ [1,2]\ [3,4] = [(1,3),(1,4),(2,3),(2,4)]$. We have $\mathrm{DP}(\mathcal{R}) =$

$$\left\{ \begin{array}{llll} (1) & \mathsf{append}\ (:\ x\ xs)\ ys & \rightarrowtail & \mathsf{append}\ xs\ ys \\ (2) & \mathsf{map}\ F\ (:\ x\ xs) & \rightarrowtail & F\ x \\ (3) & \mathsf{map}\ F\ (:\ x\ xs) & \rightarrowtail & \mathsf{map}\ F\ xs \\ (4) & ((\mathsf{curry}\ G)\ x)\ y & \rightarrowtail & G\ x\ y \\ (5) & \mathsf{cartprod}\ (:\ x\ xs)\ ys & & \\ & \multicolumn{3}{r}{\rightarrowtail\ \ \mathsf{append}\ (\mathsf{map}\ ((\mathsf{curry}\ \mathsf{pair})\ x)\ ys)\ (\mathsf{cartprod}\ xs\ ys)} \\ (6) & \mathsf{cartprod}\ (:\ x\ xs)\ ys & \rightarrowtail & \mathsf{append} \\ (7) & \mathsf{cartprod}\ (:\ x\ xs)\ ys & \rightarrowtail & \mathsf{map}\ ((\mathsf{curry}\ \mathsf{pair})\ x)\ ys \\ (8) & \mathsf{cartprod}\ (:\ x\ xs)\ ys & \rightarrowtail & \mathsf{map} \\ (9) & \mathsf{cartprod}\ (:\ x\ xs)\ ys & \rightarrowtail & (\mathsf{curry}\ \mathsf{pair})\ x \\ (10) & \mathsf{cartprod}\ (:\ x\ xs)\ ys & \rightarrowtail & \mathsf{curry}\ \mathsf{pair} \\ (11) & \mathsf{cartprod}\ (:\ x\ xs)\ ys & \rightarrowtail & \mathsf{curry} \\ (12) & \mathsf{cartprod}\ (:\ x\ xs)\ ys & \rightarrowtail & \mathsf{cartprod}\ xs\ ys \end{array} \right\}.$$

The strongly connected components of $\mathrm{EDG}(\mathcal{R})$ are $\{(1)\}$ and $\{\ (2), (3), (4), (7), (12)\ \}$. It is easy to prove $\{(1)\} \notin \mathbf{DC}_{\min}$ by using a simple projection. Let $D = \{(2),(3),(4),(7),(12)\}$. Then $\mathrm{HI}(D) =$

$$\left\{ \begin{array}{llll} (2)^{\{F \mapsto ((\mathsf{curry}\ G)\ y)\}} & \mathsf{map}\ ((\mathsf{curry}\ G)\ y)\ (:\ x\ xs) & \rightarrowtail & ((\mathsf{curry}\ G)\ y)\ x \\ (3) & \mathsf{map}\ F\ (:\ x\ xs) & \rightarrowtail & \mathsf{map}\ F\ xs \\ (4)^{\{G \mapsto \mathsf{cartprod}\}} & ((\mathsf{curry}\ \mathsf{cartprod})\ x)\ y & \rightarrowtail & \mathsf{cartprod}\ x\ y \\ (7) & \mathsf{cartprod}\ (:\ x\ xs)\ ys & \rightarrowtail & \mathsf{map}\ ((\mathsf{curry}\ \mathsf{pair})\ x)\ ys \\ (12) & \mathsf{cartprod}\ (:\ x\ xs)\ ys & \rightarrowtail & \mathsf{cartprod}\ xs\ ys \end{array} \right\}.$$

We now show $\mathrm{HI}(D) \notin \mathbf{DC}$ by using the recursive SCC algorithm of [9]. Take a simple projection π as $\pi(\mathsf{curry}) = 3$, $\pi(\mathsf{map}) = \pi(\mathsf{cartprod}) = 2$. It is easy to check $\{\pi((2)^{\{F \mapsto ((\mathsf{curry}\ G)\ y)\}}),\ \pi((3))\} \subseteq \rhd$ and $\{\pi((4)^{\{G \mapsto \mathsf{cartprod}\}}),\ \pi((7)),\ \pi((12))\} \subseteq \unrhd$. Thus it suffices to show $D' = \{\pi((4)^{\{G \mapsto \mathsf{cartprod}\}}), (7), (12)\} \notin \mathbf{DC}$. The only cycle in $\mathrm{EDG}(D')$ is $\{(12)\}$. By taking a simple projection π as $\pi(\mathsf{cartprod}) = 1$, we know $D' \notin \mathbf{DC}$. Hence $\mathrm{HI}(D) \notin \mathbf{DC}$. Therefore $\mathcal{R}$ is terminating.

7 Experiments

Based on the techniques presented in this paper, we implemented a termination prover for STTRSs written in the functional programming language Haskell. We have tested 125 examples of STTRSs (of which 122 examples are terminating). They include typical higher-order functions such as fold, map, rec, filter, etc. of various simple types. For example, fold of types $(o{\times}o{\to}o){\times}o{\times}o{\to}o$, $(o{\times}o{\to}o){\times}o{\to}o{\to}o$, $(o{\to}o{\to}o){\times}o{\to}o{\to}o$, and $(o{\to}o{\to}o){\to}o{\to}o{\to}o$. The table below is the number of STTRSs categorized by the number of functions, i.e. defined symbols. (Mutually recursive functions are counted as one function.)

number of functions	1	2	3	4	5	6	7	8
number of programs	40	21	28	15	7	5	6	3

In our first experiment, we count the number of STTRSs $\mathcal{R}$ whose termination can be inferred by checking that $\mathrm{EDG}(\mathcal{R})$ is acyclic. In the second experiment, termination proof using subterm criterion is tried additionally; and in the third, both subterm criterion and head instantiation are used.

We have also tested the direct application of the first-order dependency pair method to a first-order encoding of STTRSs. In the experiment, we used the termination prover AProVE [8] of ver.1.1c-β where the non-overlappingness check is disabled (so that it does not attempt innermost termination proof, but attempts termination proof). Our encoding is the most naive one; the encoding Θ from a simply typed term to the corresponding first-order term is defined as $\Theta((t_0\ t_1 \cdots t_n)) = \mathsf{a}_n(\Theta(t_0), \Theta(t_1), \ldots, \Theta(t_n))$, where a_n $(n > 0)$ are new function symbols of arity n.

The result is summarized in the table below.

	cycle check	+ subterm criterion	+ head instantiation	first-order encoding
success	6	56	98	67
failure/timeout(30 sec.)	119/0	69/0	27/0	55/3

This table shows that both the subterm criterion and the head instantiation are effective in simply typed dependency pair approach. Our approach proves more examples than the termination proof via the naive first-order encoding does. There are, however, 8 examples that succeed in the approach via first-order encoding but fail in our approach.

8 Concluding remarks

We presented an extension of the dependency pair method of first-order term rewriting which enlarges the scope of automatic termination proofs for simply typed term rewriting systems. We clarified basic concepts of the dependency pair method in the simply typed term rewriting framework. We incorporated the subterm criterion into the extension and introduced the head instantiation. As a result, we obtained a dependency pair method which is effectively applicable even in the presence of function variables. We compared our method with the direct application of the first-order dependency pair method to a naive first-order encoding of simply typed term rewriting systems. Comparison with a more elaborate encoding such as [1] and [2] remains as a future work.

In contrast to the subterm criterion, the argument filtering is not likely to be incorporated in a straightforward way; for, in the presence of variables of function types, argument filtering functions are not closed under substitutions. Because of this difficulty, the reduction pair and the argument filtering techniques in the simply typed term rewriting framework still need to be investigated. Extensions of other techniques of the first-order dependency pair method such as the innermost termination proof have not yet been tried.

In [17, 18], dependency pair techniques for Nipkow's HRSs [15] have been proposed. In [18] the notions of the dependency pair and the dependency graph are incorporated into HRSs, while the difficulty of using the argument filtering are reported. In [17], the argument filtering has been incorporated at the cost of limiting HRSs to be non-duplicating and non-nested.

In [13, 14], Kusakari proposed path orderings and a dependency pair method for his higher-order framework TRS_{hv}. His method is based on the reduction pair and the argument filtering techniques. Our characterization of minimal non-terminating terms is similar to those used in his paper. However, we believe that our in-depth analysis is indispensable to make our development robust. In his method, in contrast to ours, the definition of the dependency chain is modified, while the definition of the dependency pair is kept same as that of the first-order dependency pair. By this difference, it seems not obvious whether our version of the dependency graph and the subterm criterion are incorporated to his dependency pair method or not.

Finally, we remark that experiments for a numerous collection of examples have not been likely intended by these existing works on higher-order dependency pair methods. The head instantiation and estimation of dependency graphs using patterns are proposed for the first time in this paper.

Acknowledgments

The authors thank the anonymous referees for their valuable comments. This work was partially supported by grants from JSPS, Nos. 14780187 and 16700030, and from Kayamori Foundation of Informational Science Advancement.

References

1. T. Aoto and T. Yamada. Termination of simply typed term rewriting systems by translation and labelling. In *Proceedings of the 14th International Conference on Rewriting Techniques and Applications*, volume 2706 of *LNCS*, pages 380–394. Springer-Verlag, 2003.
2. T. Aoto and T. Yamada. Termination of simply-typed applicative term rewriting systems. In *Proceedings of the 2nd International Workshop on Higher-Order Rewriting*, 2004.

3. T. Arts and J. Giesl. Termination of term rewriting using dependency pairs. *Theoretical Computer Science*, 236(1–2):133–178, 2000.
4. J. Giesl and T. Arts. Verification of Erlang processes by dependency pairs. *Applicable Algebra in Engineering, Communication and Computing*, 12(1/2):39–72, 2001.
5. J. Giesl, T. Arts, and E. Ohlebusch. Modular termination proofs for rewriting using dependency pairs. *Journal of Symbolic Computation*, 34(1):21–58, 2002.
6. J. Giesl, R. Thiemann, P. Schneider-Kamp, and S. Falke. Improving dependency pairs. In *Proceedings of the 10th International Conference on Logic for Programming, Artificial Intelligence and Reasoning*, volume 2850 of *LNAI*, pages 165–179. Springer-Verlag, 2003.
7. J. Giesl, R. Thiemann, P. Schneider-Kamp, and S. Falke. Mechanizing dependency pairs. Research Report 2003–08, Aachener Informatik-Bericht, 2003.
8. J. Giesl, R. Thiemann, P. Schneider-Kamp, and S. Falke. Automated termination proofs with AProVE. In *Proceedings of the 15th International Conference on Rewriting Techniques and Applications*, volume 3091 of *LNCS*, pages 210–220. Springer-Verlag, 2004.
9. N. Hirokawa and A. Middeldorp. Automating the dependency pair method. In *Proceedings of the 19th International Conference on Automated Deduction*, volume 2741 of *LNAI*. Springer-Verlag, 2003.
10. N. Hirokawa and A. Middeldorp. Dependency pairs revisited. In *Proceedings of the 15th International Conference on Rewriting Techniques and Applications*, volume 3091 of *LNCS*, pages 249–268. Springer-Verlag, 2004.
11. J.-P. Jouannaud and M. Okada. Executable higher-order algebraic specification languages. In *Proceedings of the 6th IEEE Symposium on Logic in Computer Science*, pages 350–361. IEEE Press, 1991.
12. J. W. Klop. *Combinatory Reduction Systems*. PhD thesis, Rijksuniversiteit, Utrecht, 1980.
13. K. Kusakari. On proving termination of term rewriting systems with higher-order variables. *IPSJ Transactions on Programming*, 42(SIG 7 PRO 11):35–45, 2001.
14. K. Kusakari. Higher-order path orders based on computability. *IEICE Transactions on Information and Systems*, E87–D(2):352–359, 2003.
15. R. Mayr and T. Nipkow. Higher-order rewrite systems and their confluence. *Theoretical Computer Science*, 192(1):3–29, 1998.
16. A. Middeldorp. Approximations for strategies and termination. *Electronic Notes in Theoretical Computer Science*, 70(6):1–20, 2002.
17. M. Sakai and K. Kusakari. On new dependency pair method for proving termination of higher-order rewrite systems. In *Proceedings of the International Workshop on Rewriting in Proof and Computation*, pages 176–187, 2001.
18. M. Sakai, Y. Watanabe, and T. Sakabe. An extension of dependency pair method for proving termination of higher-order rewrite systems. *IEICE Transactions on Information and Systems*, E84–D(8):1025–1032, 2001.
19. R. Thiemann, J. Giesl, and P. Schneider-Kamp. Improved modular termination proofs using dependency pairs. In *Proceedings of the 2nd International Joint Conference on Automated Reasoning*, volume 3097 of *LNAI*, pages 75–90. Springer-Verlag, 2004.
20. T. Yamada. Confluence and termination of simply typed term rewriting systems. In *Proceedings of the 12th International Conference on Rewriting Techniques and Applications*, volume 2051 of *LNCS*, pages 338–352. Springer-Verlag, 2001.