Argument Filterings and Usable Rules for Simply Typed Dependency Pairs*

Takahito Aoto[1] and Toshiyuki Yamada[2]

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

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

Abstract. Simply typed term rewriting (Yamada, 2001) is a framework of higher-order term rewriting without bound variables based on Lisp-like syntax. The dependency pair method for the framework has been obtained by extending the first-order dependency pair method and subterm criterion in (Aoto & Yamada, 2005). In this paper, we incorporate termination criteria using reduction pairs and related refinements into the simply typed dependency pair framework using recursive path orderings for S-expression rewriting systems (Toyama, 2008). In particular, we incorporate the usable rules criterion with respect to argument filterings, which is a key ingredient to prove the termination in a modular way. The proposed technique has been implemented in a termination prover and an experimental result is reported.

1 Introduction

Simply typed term rewriting [22] is a framework of higher-order term rewriting without bound variables based on *Lisp-like* syntax (e.g. $(\mathsf{map}\ F\ (\mathsf{cons}\ 0\ \mathsf{nil}))$) equipped with simple types (e.g. $\mathsf{map} : (\mathrm{N} \to \mathrm{N}) \times \mathrm{L} \to \mathrm{L}$). Its untyped version has been called *S-expression rewriting* in [19]. To integrate these names, we refer to our framework as *simply typed S-expression rewriting* in this paper. Termination proof techniques for the (simply typed) S-expression rewriting system ((ST)SRS, for short) have been investigated in [1, 3, 19, 20, 22].

The dependency pair method [5] is a powerful termination proof technique for first-order term rewriting systems which virtually all modern termination provers are based on. The authors incorporated the dependency pair framework to the STSRSs in [3], which gave a characterization of minimal non-terminating simply typed S-expressions and extended the notions of dependency pairs and (estimated) dependency graphs into the simply typed framework. They also extended the subterm criterion [10] of first-order dependency pairs and introduced the head instantiation technique to make the simply typed dependency pair method effective even in the presence of function variables.

* An extended abstract [4] of a preliminary version of this paper has been appeared in the proceedings of HOR 2007.

In this paper, we incorporate termination criteria using reduction pairs and related refinements into the simply typed dependency pair framework. For this, we use the recursive path ordering for S-expression rewriting systems [19, 20]. We extend the notions of argument filterings [5] and usable rules [10, 9] to the case of the simply typed framework. In particular, we incorporate the usable rules criterion with respect to argument filterings, which is a key ingredient to prove termination in a modular way. The proposed technique has been implemented in a termination prover and an experimental result is reported.

Related works and the contribution of the paper. Dependency pair methods for another similar framework of binder-free simply typed term rewriting based on *ML-like* syntax (the *applicative style simply typed framework*) have been studied by Kusakari, Sakai and others [13–17]. Although there are subtle differences between our approach and theirs, the main contributions of the paper and relations with results in [13–17] are summarized as follows.

- *Argument filtering in the simply typed setting based on the stability condition:* Argument filtering has been incorporated to the simply typed setting in the applicative style by Kusakari [13]. However, his method has some limitations that selective filtering for higher-order variables and rewrite rules of higher-order types are not considered. Extensions for such cases are not straightforward as the naive extension makes the technique unsound. This paper solves these issues by introducing a stability condition.
- *Simply typed usable rules with respect to argument filterings:* Usable rules without argument filterings [10] have been incorporated to simply typed setting in our preliminary version [4]. This paper also incorporates usable rules with respect to argument filterings [9]. Meanwhile, for the applicative style simply typed framework, usable rules without argument filterings appeared in [17] and very recently the ones with respect to argument filterings appeared in [16]. Unlike usable rules in [16, 17], our usable rules need not to be closed with an (unusual) extra propagation rule.
- *Implementation in a termination prover and experiments:* In contrast to the termination proving in first-order term rewriting, few implementation of a termination prover are known for the higher-order setting. An implementation of extensions presented in the preceding clauses in a termination prover is reported for the first time in this paper.

2 Preliminaries

In this section, we briefly recall the terminology and the notations of simply typed S-expression rewriting (simply typed term rewriting in [22]).

A *simple type* is either the *base type* o or a *function type* $\tau_1 \times \cdots \times \tau_n \to \tau_0$. The set of all simple types is denoted by ST. For the sake of simplicity, we consider only a 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, while the sets of all constants and all variables are denoted by Σ and V. The set $\mathrm{S}(\Sigma, V)^\tau$ of *simply typed S-expressions* of type τ is defined as follows: (i) $\Sigma^\tau \cup V^\tau \subseteq \mathrm{S}(\Sigma, V)^\tau$, (ii) if $t_0 \in \mathrm{S}(\Sigma, V)^{\tau_1 \times \cdots \times \tau_n \to \tau}$ and $t_i \in \mathrm{S}(\Sigma, V)^{\tau_i}$ for all $i \in \{1, \ldots, n\}$ then $(t_0\ t_1\ \cdots\ t_n) \in \mathrm{S}(\Sigma, V)^\tau$. The set of all simply typed S-expressions is denoted by $\mathrm{S}(\Sigma, V)^{\mathrm{ST}}$. The type of a simply typed S-expression t is denoted by $\mathrm{type}(t)$. By dropping type information, we obtain the set $\mathrm{S}(\Sigma, V)$ of *untyped* S-expressions. We call a simply typed/untyped S-expression just an S-expression when we do not care whether it is simply typed or not. The set of variables in an S-expression t is denoted by $\mathrm{V}(t)$. The *head symbol* of an S-expression is defined as follows: $\mathrm{head}(a) = a$ for $a \in \Sigma \cup V$; $\mathrm{head}((t_0\ t_1 \cdots t_n)) = \mathrm{head}(t_0)$.

A simply typed *context* of type τ is a simply typed S-expression that contains one special symbol $\Box^\tau$, called the *hole*, of type τ. The simply typed S-expression obtained by replacing the hole in a simply typed context C of type τ with a simply typed S-expression 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. The notion of context for untyped S-expressions is defined similarly. An S-expression s is a *subexpression* of an S-expression t (denoted by $s \trianglelefteq t$) if $C[s] = t$ for some context C, and is a *proper* subexpression (denoted by $s \triangleleft t$) when $s \neq t$ holds in addition. A simply typed *substitution* is a mapping $\sigma : V \to \mathrm{S}(\Sigma, V)^{\mathrm{ST}}$ such that $\mathrm{type}(x) = \mathrm{type}(\sigma(x))$ for all $x \in V$. An *instance* of a simply typed S-expression t is written as $t\sigma$. The notion of substitution for untyped S-expressions is defined similarly.

Every *simply typed rewrite rule* $l \to r$ must satisfy the following conditions: (1) $\mathrm{type}(l) = \mathrm{type}(r)$, (2) $\mathrm{head}(l) \in \Sigma$, and (3) $\mathrm{V}(r) \subseteq \mathrm{V}(l)$. Let $\mathcal{R} = \langle \Sigma, R \rangle$ be a *simply typed S-expression rewriting system* (STSRS, for short). The set Σ_{d} of *defined symbols* of $\mathcal{R}$ is defined by $\Sigma_{\mathrm{d}} = \{\mathrm{head}(l) \mid l \to r \in R\}$. For *untyped rewrite rule* $l \to r$, we omit the conditions (1) and (2) and impose (2′) $l \notin V$ [19, 20]. An *S-expression rewriting system* (SRS, for short) is a pair $\mathcal{R} = \langle \Sigma, R \rangle$ where Σ is a set of untyped constants and R is a set of untyped rewrite rules. The *rewrite relation* induced by (ST)SRS $\mathcal{R}$ is denoted by $\to_{\mathcal{R}}$; its reflexive closure, transitive closure, and reflexive transitive closure are denoted by $\to_{\mathcal{R}}^{=}$, $\to_{\mathcal{R}}^{+}$, and $\to_{\mathcal{R}}^{*}$, respectively.

The *root rewrite step* $\xrightarrow{\mathrm{r}}_{\mathcal{R}}$ is defined as $s \xrightarrow{\mathrm{r}}_{\mathcal{R}} t$ if $s = l\sigma$ and $t = r\sigma$ for some rewrite rule $l \to r$ and some substitution σ. The *head rewrite step* $\xrightarrow{\mathrm{h}}_{\mathcal{R}}$ is defined recursively as follows: $s \xrightarrow{\mathrm{h}}_{\mathcal{R}} t$ if (1) $s \xrightarrow{\mathrm{r}}_{\mathcal{R}} t$ or (2) $s = (s_0\ u_1\ \cdots\ u_n)$, $t = (t_0\ u_1\ \cdots\ u_n)$, and $s_0 \xrightarrow{\mathrm{h}}_{\mathcal{R}} t_0$. The non-head rewrite step is defined by $\xrightarrow{\mathrm{nh}}_{\mathcal{R}} = \to_{\mathcal{R}} \setminus \xrightarrow{\mathrm{h}}_{\mathcal{R}}$. An *argument context* is a context whose head symbol is the hole, more precisely, κ is an argument context of type τ if (1) $\kappa = \Box^\tau$, or (2) $\kappa = (\kappa'\ t_1 \cdots t_n)$ for some argument context κ' of a function type and some S-expressions $t_1, \ldots, t_n$ of appropriate types. Then $s \xrightarrow{\mathrm{h}}_{\mathcal{R}} t$ holds 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 $t = \kappa[r\sigma]$.

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

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

An example of rewrite sequence is $(\mathsf{sum}\ (:\ (\mathsf{s}\ \mathsf{0})\ [\,])) \to_{\mathcal{R}} ((\mathsf{fold}\ +\ \mathsf{0})\ (:\ (\mathsf{s}\ \mathsf{0})\ [\,])) \to_{\mathcal{R}} ((+\ (\mathsf{s}\ \mathsf{0}))\ ((\mathsf{fold}\ +\ \mathsf{0})\ [\,])) \to_{\mathcal{R}} ((+\ (\mathsf{s}\ \mathsf{0}))\ \mathsf{0}) \to_{\mathcal{R}} (\mathsf{s}\ ((+\ \mathsf{0})\ \mathsf{0})) \to_{\mathcal{R}} (\mathsf{s}\ \mathsf{0})$.

The definition of simply typed dependency pairs is given as follows.

Definition 1 (dependency pairs [3]). A *simply typed dependency pair* (*untyped dependency pair*) is a pair of simply typed (resp. untyped) S-expressions. The set $\mathrm{DP}(\mathcal{R})$ of all *dependency pairs* of an STSRS $\mathcal{R} = \langle \Sigma, R \rangle$ is the set of simply typed dependency pairs defined as follows:

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

where the *argument expansion* $\mathrm{Exp}(l \to r)$ is defined 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 $x_1, \ldots, x_n$ are distinct fresh variables of appropriate types. A (simply typed or untyped) dependency pair $\langle l, r \rangle$ is also written as $l \rightarrowtail r$.

Example 2 (dependency pairs). Let $\mathcal{R}$ be an STSRS in Example 1. Then $\mathrm{DP}(\mathcal{R}) =$

$$\left\{ \begin{array}{llll} ((+\ (\mathsf{s}\ x))\ y) & \rightarrowtail ((+\ x)\ y) & \mathsf{sum} & \rightarrowtail (\mathsf{fold}\ +\ \mathsf{0}) \\ ((+\ (\mathsf{s}\ x))\ y) & \rightarrowtail (+\ x) & \mathsf{sum} & \rightarrowtail \mathsf{fold} \\ ((\mathsf{fold}\ F\ x)\ (:\ y\ ys)) & \rightarrowtail ((F\ y)\ ((\mathsf{fold}\ F\ x)\ ys)) & & \\ ((\mathsf{fold}\ F\ x)\ (:\ y\ ys)) & \rightarrowtail (F\ y) & \mathsf{sum} & \rightarrowtail + \\ ((\mathsf{fold}\ F\ x)\ (:\ y\ ys)) & \rightarrowtail ((\mathsf{fold}\ F\ x)\ ys) & (\mathsf{sum}\ xs) & \rightarrowtail ((\mathsf{fold}\ +\ \mathsf{0})\ xs) \end{array} \right\}.$$

In the simply typed framework, a root rewrite step using a dependency pair is not in general type-preserving and is distinguished from the rewrite relation.

Definition 2 (dependency relation [3]). Let $D \subseteq \mathrm{DP}(\mathcal{R})$. The *dependency relation* $\rightarrowtail_D$ is defined as follows: $s \rightarrowtail_D t$ if there exist a dependency pair $l \rightarrowtail r \in D$ and a simply typed substitution σ such that $s = l\sigma$, $t = r\sigma$, and $\mathrm{head}(t) \in \Sigma_{\mathrm{d}}$.

We say an S-expression s is *terminating* if there is no infinite rewrite sequence starting from s, otherwise *non-terminating*. We denote the set of non-terminating S-expressions by $\mathrm{NT}(\mathcal{R})$. The set of minimal (w.r.t. the subexpression relation $\trianglelefteq$) non-terminating S-expressions is denoted by $\mathrm{NT}_{\min}(\mathcal{R})$. Let $\mathcal{R}$ be an STSRS and D a set of simply typed dependency pairs of $\mathcal{R}$. A *dependency chain* of D is an

infinite sequence $t_0, t_1, \ldots$ on $\mathrm{NT}_{\min}(\mathcal{R}) \cap \mathrm{S}(\Sigma, V)^{\mathrm{ST}}$ such that $t_i \xrightarrow{\mathrm{nh}*}_{\mathcal{R}} \cdot \rightarrowtail_D t_{i+1}$ for all $i \geq 0$. For a set of dependency pairs D, we define $\mathbf{DC}(D)$ to be the set of all subsets of D that admit any dependency chain. We may omit the parameters $\mathcal{R}$ and D when they are not important.

Theorem 1 (termination by dependency chains [3]). An STSRS $\mathcal{R}$ is terminating if and only if $\mathbf{DC}(\mathrm{DP}(\mathcal{R})) = \emptyset$.

A relation $\gtrsim$ on a set of S-expressions is a *rewrite quasi-order* if it is a quasi-order that is closed under substitutions (i.e., $s \gtrsim t$ implies $s\sigma \gtrsim t\sigma$ for any s, t, σ) and closed under contexts (i.e., $s \gtrsim t$ implies $C[s] \gtrsim C[t]$ for any s, t, C). A pair $\langle \gtrsim, \succ \rangle$ of relations on S-expressions is a *reduction pair* if (1) $\gtrsim$ is a rewrite quasi-order, (2) $\succ$ is closed under substitutions, and (3) there is no infinite sequence $t_0 \gtrsim \cdot \succ t_1 \gtrsim \cdot \succ t_2 \cdots$.

Definition 3 (preservation of dependency chain). Let $\mathcal{R} = \langle \Sigma, R \rangle$ be an STSRS, D a set of simply typed dependency pairs of $\mathcal{R}$, $\varphi : \mathrm{S}(\Sigma, V)^{\mathrm{ST}} \to \mathrm{S}(\Sigma', V)$, R', D' sets of untyped rewrite rules on $\mathrm{S}(\Sigma', V)$, and $\psi : D \to D'$. A quadruple $\langle \varphi, R', D', \psi \rangle$ *preserves dependency chains* of D if (1) $s \xrightarrow{\mathrm{nh}*}_{\mathcal{R}} \cdot \rightarrowtail_D t$ implies $\varphi(s) \to^*_{R'} \cdot \xrightarrow{\mathrm{r}}_{D'} \varphi(t)$, for all $s, t \in \mathrm{NT}_{\min}(\mathcal{R})$, and (2) for all $s, t \in \mathrm{S}(\Sigma, V)^{\mathrm{ST}}$ and $d \in D$, $s \rightarrowtail_{\{d\}} t$ implies $\varphi(s) \xrightarrow{\mathrm{r}}_{\{\psi(d)\}} \varphi(t)$.

Lemma 1 (preservation of dependency chain). Let $\mathcal{R} = \langle \Sigma, R \rangle$ be an STSRS and D a set of simply typed dependency pairs of $\mathcal{R}$, $\langle \gtrsim, \succ \rangle$ a reduction pair, $\langle \varphi, R', D', \psi \rangle$ a quadruple that preserves dependency chains of D such that $R' \subseteq \gtrsim$ and $D' \subseteq \gtrsim$. If $D \setminus D_\succ$ admits no dependency chains where $D_\succ = \{d \in D \mid \psi(d) \subseteq \succ\}$, then D also admits no dependency chains.

Most of modern termination provers for first-order term rewriting employ the DP framework [9]. Below we formulate a DP framework in the simply typed setting. We first formulate the notion of DP problems in a way similar to the first-order case.

Definition 4 (DP problems). A *DP problem* is a pair $\langle D, \mathcal{R} \rangle$ where $\mathcal{R}$ is an STSRS and D is a set of simply typed dependency pairs. A DP problem $\langle D, \mathcal{R} \rangle$ is *finite* if $\mathbf{DC}(D) = \emptyset$; it is *infinite* otherwise.

Using these notions, Theorem 1 is reformulated as follows.

Theorem 2 (termination by a DP problem). An STSRS $\mathcal{R}$ is terminating iff the DP problem $\langle \mathrm{DP}(\mathcal{R}), \mathcal{R} \rangle$ is finite.

The notion of DP processors for first-order DP problems [9] can be incorporated in a straightforward way.

Definition 5 (DP processors). A *DP processor* Φ on a set P of DP problems is a function from P to the set of finite sets of DP problems. A DP processor Φ on P is *sound* if, for any $p \in P$, the finiteness of all elements of $\Phi(p)$ implies that of the DP problem p.

In contrast to the first-order case, the head symbol of the rhs of a dependency pair may be a variable. Based on the *head instantiation* technique [3], however, it suffices to handle dependency pairs whose head symbols of rhs's are defined constants. Such dependency pairs are said to be *head-instantiated*. To be more precise, the head instantiation yields a sound DP processor Φ on the set of simply typed DP problems such that any $\langle D', \mathcal{R}' \rangle \in \Phi(\langle D, \mathcal{R} \rangle)$, $\mathcal{R}' = \mathcal{R}$ and D' is a set of head instantiated simply typed dependency pairs. Similarly, all techniques in [3] are reformulated in the term of DP processors, as in the first-order case [9].

Definition 6 (DP tree [9]). A *DP tree* over the DP processors Φ_i $(i \in I)$ is a finite tree such that (1) each leaf node is labeled with a finite DP problem, and (2) each internal node is labeled with a DP problem p such that (a) there is a DP processors Φ_i on a domain that includes p and (b) its children nodes are precisely those labeled with each element from $\Phi_i(p)$.

The following is a corollary of this definition and Theorem 2, which is used as the basis of a design of termination provers.

Theorem 3 (termination by a DP tree). An STSRS $\mathcal{R}$ is terminating if there exists a DP tree over sound DP processors such that root node is labeled with the DP problem $\langle \mathrm{DP}(\mathcal{R}), \mathcal{R} \rangle$.

The next theorem is an immediate consequence of Lemma 1, which is a basis of all main results presented in this paper.

Theorem 4 (termination by reduction pairs). Let $\mathcal{R} = \langle \Sigma, R \rangle$ be an STSRS, P a set of simply typed DP problems. Suppose Φ is a DP processor on P given by $\Phi(\langle D, \mathcal{R} \rangle) = \{\langle D \backslash D_{\succ}, \mathcal{R} \rangle\}$ if there exists a reduction pair $\langle \gtrsim, \succ \rangle$ such that some quadruple $\langle \varphi, R', D', \psi \rangle$ preserves dependency chains of D, $R' \subseteq \gtrsim$, $D' \subseteq \gtrsim$ where $D_{\succ} = \{d \in D \mid \psi(d) \subseteq \succ\}$, and $\Phi(\langle D, \mathcal{R} \rangle) = \{\langle D, \mathcal{R} \rangle\}$ otherwise. Then Φ is sound.

3 Argument filterings

In the first-order case, an argument filtering associates each function symbol with argument positions to be selected. In the simply typed case, not only the head symbol but the depth of its occurrence in an S-expression needs to be considered additionally. For example, we may want to give different filterings for the same head symbol f in $(f\ x\ y)$ and $((f\ x\ y)\ z)$, occurring at different depths.

We first formulate the domain of argument filtering functions.

Definition 7 (depth and filtering domain). The *depth* of a simple type τ is defined as follows: $\mathrm{depth}(\mathrm{o}) = 0$; $\mathrm{depth}(\tau_1 \times \cdots \times \tau_n \to \tau_0) = \mathrm{depth}(\tau_0) + 1$. For a set X of simply typed constants and simply typed variables, the *filtering domain* is defined by $\mathrm{FDom}(X) = \{\langle a, n \rangle \mid a \in X,\ 0 \leq n < \mathrm{depth}(\mathrm{type}(a))\}$.

By definition, $\mathrm{depth}(\tau) > 0$ for any function type τ.

Example 3 (filtering domain). In Example 1, we have $\mathrm{FDom}(\Sigma \cup \{F^{\mathrm{o}\to\mathrm{o}\to\mathrm{o}}\}) = \{\langle \mathsf{s}, 0\rangle, \langle :, 0\rangle, \langle \mathsf{sum}, 0\rangle, \langle +, 0\rangle, \langle +, 1\rangle, \langle F, 0\rangle, \langle F, 1\rangle, \langle \mathsf{fold}, 0\rangle, \langle \mathsf{fold}, 1\rangle\}$.

Definition 8 (recursive extraction of range type). For each $n \leq \mathrm{depth}(\tau)$, $\tau \lfloor n$ is defined as follows: $\tau \lfloor 0 = \tau$; $(\tau_1 \times \cdots \times \tau_m \to \tau_0) \lfloor (n+1) = \tau_0 \lfloor n$.

Example 4 (recursive extraction of range type). We have $((\mathrm{o} \to \mathrm{o}) \times (\mathrm{o} \to \mathrm{o}) \to \mathrm{o} \to \mathrm{o}) \lfloor 2 = (\mathrm{o} \to \mathrm{o}) \lfloor 1 = \mathrm{o}$.

Lemma 2 (property of depth). For each $\tau \in \mathrm{ST}$ and $n \leq \mathrm{depth}(\tau)$, (1) $\tau \lfloor n$ is the base type iff $\mathrm{depth}(\tau) = n$, (2) $\tau \lfloor n$ is a function type iff $\mathrm{depth}(\tau) > n$.

Proof. By induction on n. □

Thus, for each $\langle a, n\rangle \in \mathrm{FDom}(X)$, $\mathrm{type}(a) \lfloor n$ is a function type. So, we define $\mathrm{ArgPos}(a, n) = \{0, 1, \cdots, m\}$ when $\mathrm{type}(a) \lfloor n = \tau_1 \times \cdots \times \tau_m \to \tau_0$.

Now we are ready to give the definition of argument filtering function.

Definition 9 (argument filtering). Let X be a set of simply typed constants and simply typed variables. A function $\pi : \mathrm{FDom}(X) \to \mathrm{List}(\mathbb{N}) \cup \mathbb{N}$ is an *argument filtering* for X if, for each $\langle a, n\rangle \in \mathrm{FDom}(X)$, either $\pi(a, n) = [i_1, \ldots, i_k]$ for some $i_1, \cdots, i_k \in \mathrm{ArgPos}(a, n)$ with $i_1 < \cdots < i_k$ or $\pi(a, n) \in \mathrm{ArgPos}(a, n)$. Note that if $k = 0$ then $[i_1, \ldots, i_k]$ is the empty list.

In order to select argument positions of a simply typed S-expression t specified by an argument filtering π, a natural number (together with a symbol) needs to be designated. A notion of head depth is introduced for this purpose.

Definition 10 (head depth). The *head depth* of a simply typed S-expression is defined as follows: $\mathrm{hdep}(a) = 0$ if a is a constant or a variable; $\mathrm{hdep}((t_0\ t_1 \cdots t_n)) = \mathrm{hdep}(t_0) + 1$.

Lemma 3 (property of head depth). Let s be a simply typed S-expression and $\tau = \mathrm{type}(\mathrm{head}(s))$. Then (1) $\tau \lfloor \mathrm{hdep}(s) = \mathrm{type}(s)$; (2) s has the base type iff $\mathrm{depth}(\tau) = \mathrm{hdep}(s)$; (3) s has a function type iff $\mathrm{depth}(\tau) > \mathrm{hdep}(s)$.

Proof. (1) By induction on s. (2)–(3) Use (1) and Lemma 2. □

Thus, if $\mathrm{head}(s) \in X$ and s has a function type τ, then $\mathrm{hdep}(s) < \mathrm{depth}(\tau)$, and thus $\langle \mathrm{head}(s), \mathrm{hdep}(s)\rangle \in \mathrm{FDom}(X)$. The *head pair* of a simply typed S-expression t is defined by $\mathrm{hpair}(t) = \langle \mathrm{head}(t), \mathrm{hdep}(t)\rangle$. Note that a non-head rewrite step preserves both the head symbol and the head depth. Hence $s \overset{\mathrm{nh}}{\to} t$ implies $\mathrm{hpair}(s) = \mathrm{hpair}(t)$.

An argument filtering recursively selects the designated subexpressions.

Definition 11 (application of argument filtering). Let π be an argument filtering. For each simply typed S-expression t, the untyped S-expression $\pi(t)$ is defined as follows: (1) $\pi(a) = a$ if a is a constant or a variable; (2) $\pi((t_0\ t_1 \cdots t_n)) = (\pi(t_{i_1})\ \cdots\ \pi(t_{i_k}))$ if $\pi(\mathrm{hpair}(t_0)) = [i_1, \ldots, i_k]$; (3) $\pi((t_0\ t_1 \cdots t_n)) = \pi(t_i)$ if $\pi(\mathrm{hpair}(t_0)) = i$.

Example 5 (application of argument filtering). We apply various argument filtering functions to a fixed simply typed S-expression $t = ((\mathsf{fold}\ F\ x)\ y)$. If $\pi(\mathsf{fold}, 1)$ is $[\,]$, $[1]$ or 1, then $\pi(t)$ is $()$, (y), or y, respectively. Consider the case $\pi(\mathsf{fold}, 1) = 0$. If $\pi(\mathsf{fold}, 0)$ is $[0, 1, 2]$ or 1, $\pi(t)$ is $(\mathsf{fold}\ F\ x)$ or F, respectively. Let $\pi(\mathsf{fold}, 1) = [0, 1]$. If $\pi(\mathsf{fold}, 0)$ is $[\,]$, $[0, 1]$, or 0, then $\pi(t)$ is $(()\ y)$, $((\mathsf{fold}\ F)\ y)$, or $(\mathsf{fold}\ y)$, respectively.

Argument filtering can be soundly used for termination proofs provided that the filtering preserves dependency chains. However, by the presence of function variables and rewrite rules of function type, it does not always preserve dependency chains as demonstrated in the examples below.

Example 6 (unsound filtering). Let $\mathcal{R}_1 = \langle \Sigma_1, R_1 \rangle$ be an STSRS where $\Sigma_1 = \{\ \mathsf{0}^{\mathsf{o}}, \mathsf{f}^{\mathsf{o}\to\mathsf{o}}, \mathsf{s}^{\mathsf{o}\to\mathsf{o}}\ \}$ and $R_1 = \{\ (\mathsf{f}\ (F\ x)) \to (\mathsf{f}\ (\mathsf{s}\ x))\ \}$. Its dependency pair $\langle \mathsf{f}\ (F\ x),\ \mathsf{f}\ (\mathsf{s}\ x)\rangle$ admits a dependency chain $\mathsf{f}\ (\mathsf{s}\ x),\ \mathsf{f}\ (\mathsf{s}\ x),\ \cdots$. This chain is not preserved by an argument filtering π such that $\pi(\mathsf{s}, 0) = 1$ and $\pi(\mathsf{f}, 0) = \pi(F, 0) = [0, 1]$, because the filtered dependency pair $\langle \mathsf{f}\ (F\ x),\ \mathsf{f}\ x\rangle$ does not admit the filtered chain $\mathsf{f}\ x,\ \mathsf{f}\ x,\ \cdots$.

Let $\mathcal{R}_2 = \langle \Sigma_2, R_2 \rangle$ be an STSRS where $\Sigma_2 = \{\ \mathsf{f}^{\mathsf{o}\to\mathsf{o}}, \mathsf{g}^{\mathsf{o}\to\mathsf{o}}, \mathsf{h}^{\mathsf{o}\to\mathsf{o}}\ \}$ and $R_2 = \{\ (\mathsf{f}\ (\mathsf{h}\ x)) \to (\mathsf{f}\ (\mathsf{g}\ x)), \mathsf{g} \to \mathsf{h}\ \}$. The dependency pair $\langle \mathsf{f}\ (\mathsf{h}\ x),\ \mathsf{f}\ (\mathsf{g}\ x)\rangle$ in combination with the rewrite rule $\mathsf{g} \to \mathsf{h}$ admits a dependency chain $\mathsf{f}\ (\mathsf{h}\ x), \mathsf{f}\ (\mathsf{h}\ x), \cdots$. This chain is not preserved by an argument filtering π such that $\pi(\mathsf{f}, 0) = 1$, $\pi(\mathsf{g}, 0) = [\,]$, and $\pi(\mathsf{h}, 0) = [1]$, because the filtered dependency pair $(x) \rightarrowtail ()$ does not admit the filtered chain $(x), (x), \cdots$. Note that the rule $\mathsf{g} \to \mathsf{h}$ can not be applied after filtering.

The first example suggests that filtering functions should consistently select the same argument positions from both an expression with a function variable and its instance. The second example suggests that filtering functions should consistently select the same argument positions in a subexpression when its head is rewritten by a rule of function type. The former suggestion is closely related to the extraction of the substitution part from filtered S-expressions. Notions of stabilization type and stable filtering functions are needed to show this.

Definition 12 (stabilization type). Let π be an argument filtering. For any simply typed S-expression t, the set $\mathrm{Stab}(t) \subseteq \mathrm{ST}$ of *stabilization types* of t is defined as follows:

$$\mathrm{Stab}(t) = \begin{cases} \emptyset & \text{if } t \text{ is a constant or a variable} \\ \{\mathrm{type}(t_0) \mid \mathrm{head}(t_0) \in V\} \cup \bigcup_{j=1}^{k} \mathrm{Stab}(t_{i_j}) & \\ \qquad \text{if } t = (t_0\ t_1 \cdots t_n) \text{ and } \pi(\mathrm{hpair}(t_0)) = [i_1, \ldots, i_k] & \\ \{\mathrm{type}(t_0) \mid \mathrm{head}(t_0) \in V\} \cup \mathrm{Stab}(t_i) & \\ \qquad \text{if } t = (t_0\ t_1 \cdots t_n) \text{ and } \pi(\mathrm{hpair}(t_0)) = i \end{cases}$$

Similarly to the first-order case, marking symbols of dependency pairs is useful to distinguish defined head symbols from other symbols. We define $\Sigma^\sharp = \Sigma \cup \{a^\sharp \mid a \in \Sigma_{\mathrm{d}}\}$ where $a^\sharp$ is a new constant having the same type as a. For

a simply typed S-expression t such that $\mathrm{head}(t) \in \Sigma_{\mathrm{d}}$, define $t^\sharp$ recursively as follows: $t^\sharp = a^\sharp$ if $t = a \in \Sigma_{\mathrm{d}}$; $t^\sharp = (t_0^\sharp\ t_1 \cdots t_n)$ if $t = (t_0\ t_1 \cdots t_n)$. We also define $\mathrm{S}^\sharp(\Sigma, V)^{\mathrm{ST}} = \mathrm{S}(\Sigma, V)^{\mathrm{ST}} \cup \{t^\sharp \mid t \in \mathrm{S}(\Sigma, V)^{\mathrm{ST}}, \mathrm{head}(t) \in \Sigma_{\mathrm{d}}\}$.

Definition 13 (stability). Let π be an argument filtering for $\Sigma^\sharp \cup V$ and T a set of simple types. We say π is *stable* on T if for any $\langle a, n\rangle, \langle b, m\rangle \in \mathrm{FDom}(\Sigma \cup V)$ and any $\tau \in T$, $\mathrm{type}(a) \lfloor n = \mathrm{type}(b) \lfloor m = \tau$ implies $\pi(a, n) = \pi(b, m)$. Note that restrictions are imposed on argument positions only for unmarked symbols.

For a simply typed substitution σ and an argument filtering π, the substitution σ_π is defined by $\sigma_\pi(x) = \pi(\sigma(x))$.

Lemma 4 (extraction of substitution). Let $t \in \mathrm{S}^\sharp(\Sigma, V)^{\mathrm{ST}}$ and $\sigma : V \to \mathrm{S}(\Sigma, V)^{\mathrm{ST}}$ be a simply typed substitution. Let π be an argument filtering for $\Sigma^\sharp \cup V$. If π is stable on $\mathrm{Stab}(t)$ then $\pi(t\sigma) = \pi(t)\sigma_\pi$.

Proof. By induction on t. Use Lemma 3. □

Definition 14 (stability w.r.t. rules). Let π be an argument filtering.

1. π is *stable* w.r.t. a set R of simply typed rewrite rules if for any rewrite rule $l \to r \in R$, π is stable on $\mathrm{Stab}(l) \cup \mathrm{Stab}(r)$ and if the rule is of function type τ then π is stable on $\{\tau, \ldots, \tau \lfloor (\mathrm{depth}(\tau) - 1)\}$.
2. π is *stable* w.r.t. a set D of simply typed head-instantiated dependency pairs if π is stable on $\mathrm{Stab}(l^\sharp) \cup \mathrm{Stab}(r^\sharp)$ for every dependency pair $l \rightarrowtail r \in D$.

Example 7 (stability w.r.t. rules). Let $\mathcal{R} = \langle \Sigma, R\rangle$ be the STSRS in Example 1. An argument filtering π such that $\pi(\mathsf{s}, 0) = \pi(\mathsf{sum}, 0) = \pi(+, 1) = \pi(\mathsf{fold}, 1) = [1]$, $\pi(F, n) = [1]$ for all $F \in V^\tau$ and $n \in \mathbb{N}$ such that $\tau \lfloor n = \mathrm{o} \to \mathrm{o}$ is stable w.r.t. R. Note that for this argument filtering π, we have $\mathrm{Stab}(((F\ y)\ ((\mathsf{fold}\ F\ x)\ ys))) = \{\mathrm{o} \to \mathrm{o}\}$ and thus the type $\mathrm{o} \to \mathrm{o} \to \mathrm{o} \notin \bigcup_{l \to r \in R}(\mathrm{Stab}(l) \cup \mathrm{Stab}(r))$. Furthermore, since there is no rules of type $\mathrm{o} \to \mathrm{o} \to \mathrm{o}$ in R, the stability w.r.t. R holds even if we have $\pi(+, 0) \neq \pi(F^{\mathrm{o}\to\mathrm{o}\to\mathrm{o}}, 0)$.

Let π be an argument filtering, R a set of rewrite rules, and D a set of head-instantiated dependency pairs. We write $D^\sharp = \{l^\sharp \rightarrowtail r^\sharp \mid l \rightarrowtail r \in D\}$, $\pi(R) = \{\pi(l) \to \pi(r) \mid l \to r \in R\}$, and $\pi(D^\sharp) = \{\pi(l^\sharp) \to \pi(r^\sharp) \mid l \rightarrowtail r \in D\}$. An argument filtering as a transformation of a set of dependency pairs satisfies condition (2) of dependency chain preservation in Section 2.

Lemma 5 (simulation of rewrite step). Let $\mathcal{R} = \langle \Sigma, R\rangle$ be an STSRS and D a set of head-instantiated dependency pairs. Suppose that an argument filtering π is stable w.r.t. R and D. If $s, t \in \mathrm{S}(\Sigma, V)^{\mathrm{ST}}$ then (1) $s \to_R t$ implies $\pi(s) \to^{=}_{\pi(R)} \pi(t)$, (2) $s \xrightarrow{\mathrm{nh}}_R t$ implies $\pi(s^\sharp) \to^{=}_{\pi(R)} \pi(t^\sharp)$, and (3) $s \rightarrowtail_D t$ implies $\pi(s^\sharp) \xrightarrow{\mathrm{r}}_{\pi(D^\sharp)} \pi(t^\sharp)$.

Proof. (1) By induction on s. Use Lemma 4. (2) By induction on s. (3) By Lemma 4. □

Theorem 5 (argument filtering refinement). Let $\mathcal{R} = \langle \Sigma, R \rangle$ be an STSRS and P the set of simply typed DP problems. Suppose Φ is a DP processor on P given by $\Phi(\langle D, \mathcal{R} \rangle) = \{\langle D \backslash D_\succ, \mathcal{R} \rangle\}$ if all dependency pairs in D are head-instantiated and there exists a reduction pair $\langle \gtrsim, \succ \rangle$ on $\mathrm{S}(\Sigma^\sharp, V)$ and an argument filtering π stable w.r.t. R and D, $\pi(R) \subseteq \gtrsim$, $\pi(D^\sharp) \subseteq \gtrsim$ where $D_\succ = \{l \rightarrowtail r \in D \mid \pi(l^\sharp) \succ \pi(r^\sharp)\}$, and $\Phi(\langle D, \mathcal{R} \rangle) = \{\langle D, \mathcal{R} \rangle\}$ otherwise. Then Φ is a sound DP processor on P.

Proof. By Lemma 5, $s \xrightarrow{\mathrm{nh}}{}^* \cdot \rightarrowtail_D t$ implies $\pi(s^\sharp) \rightarrow^*_{\pi(R)} \cdot \xrightarrow{\mathrm{r}}_{\pi(D^\sharp)} \pi(t^\sharp)$ for all $s, t \in \mathrm{S}(\Sigma, V)^{\mathrm{ST}}$. Hence the conclusion follows from Theorem 4. □

Example 8 (termination proof (1)). Let $\mathcal{R}' = \langle \Sigma', R' \rangle$ be an STSRS where $\Sigma' = \{\ \mathsf{0}^{\mathrm{o}},\ \mathsf{s}^{\mathrm{o}\to\mathrm{o}},\ [\,]^{\mathrm{o}},\ :^{\mathrm{o}\times\mathrm{o}\to\mathrm{o}},\ \mathsf{map}^{(\mathrm{o}\to\mathrm{o})\times\mathrm{o}\to\mathrm{o}},\ \circ^{(\mathrm{o}\to\mathrm{o})\times(\mathrm{o}\to\mathrm{o})\to\mathrm{o}\to\mathrm{o}},\ \mathsf{twice}^{(\mathrm{o}\to\mathrm{o})\to\mathrm{o}\to\mathrm{o}}\ \}$, and $R' =$

$$\left\{ \begin{array}{ll} (\mathsf{map}\ G\ [\,]) & \to [\,] \\ (\mathsf{map}\ G\ (:\ x\ xs)) & \to (:\ (G\ x)\ (\mathsf{map}\ G\ xs)) \end{array} \right\} \quad \left\{ \begin{array}{ll} ((\circ\ G\ H)\ x) & \to (G\ (H\ x)) \\ (\mathsf{twice}\ G) & \to (\circ\ G\ G) \end{array} \right\}.$$

Then two DP problems $\langle D_1, \mathcal{R}' \rangle$, $\langle D_2, \mathcal{R}' \rangle$ are obtained from the head-instantiation and dependency graph processors from the DP problem $\langle \mathrm{DP}(\mathcal{R}), \mathcal{R} \rangle$ where

$$D_1 = \left\{\ (\mathsf{map}^\sharp\ G\ (:\ x\ xs)) \quad \rightarrowtail \quad (\mathsf{map}^\sharp\ G\ xs)\ \right\},\ \text{and}\ D_2 =$$

$$\left\{ \begin{array}{lll} ((\circ^\sharp\ (\circ\ U\ V)\ H)\ x) & \rightarrowtail ((\circ^\sharp\ U\ V)\ (H\ x)) & ((\circ^\sharp\ G\ (\circ\ U\ V))\ x) \rightarrowtail ((\circ^\sharp\ U\ V)\ x) \\ ((\circ^\sharp\ (\mathsf{twice}\ U)\ H)\ x) & \rightarrowtail ((\mathsf{twice}^\sharp\ U)\ (H\ x)) & ((\circ^\sharp\ G\ (\mathsf{twice}\ U))\ x) \rightarrowtail ((\mathsf{twice}^\sharp\ U)\ x) \\ ((\mathsf{twice}^\sharp\ G)\ x) & \rightarrowtail ((\circ^\sharp\ G\ G)\ x) & \end{array} \right\}.$$

The finiteness of $\langle D_1, \mathcal{R}' \rangle$ is shown using the subterm criterion processor [3]. We here show the finiteness of $\langle D_2, \mathcal{R}' \rangle$ based on the processor Φ given in Theorem 5. Take an argument filtering π such that $\pi(H^{\mathrm{o}\to\mathrm{o}}, 0) = \pi(\circ, 1) = \pi(\mathsf{twice}, 1) = 1$, $\pi(\mathsf{twice}, 0) = \pi(\mathsf{twice}^\sharp, 0) = \pi(\circ^\sharp, 1) = [0, 1]$, and $\pi(\mathsf{map}, 0) = \pi(:, 0) = \pi(\circ, 0) = \pi(\circ^\sharp, 0) = [0, 1, 2]$. Then π is stable w.r.t. R' and D_2, and we have $\pi(R') =$

$$\left\{ \begin{array}{lll} (\mathsf{map}\ G\ [\,]) & \to & [\,] \\ (\mathsf{map}\ G\ (:\ x\ xs)) & \to & (:\ x\ (\mathsf{map}\ G\ xs)) \end{array} \right\} \quad \left\{ \begin{array}{lll} x & \to & x \\ (\mathsf{twice}\ G) & \to & (\circ\ G\ G) \end{array} \right\}, \pi(D_2^\sharp) =$$

$$\left\{ \begin{array}{lll} ((\circ^\sharp\ (\circ\ U\ V)\ H)\ x) & \rightarrowtail ((\circ^\sharp\ U\ V)\ x) & ((\circ^\sharp\ G\ (\circ\ U\ V))\ x) \rightarrowtail ((\circ^\sharp\ U\ V)\ x) \\ ((\circ^\sharp\ (\mathsf{twice}\ U)\ H)\ x) & \rightarrowtail ((\mathsf{twice}^\sharp\ U)\ x) & ((\circ^\sharp\ G\ (\mathsf{twice}\ U))\ x) \rightarrowtail ((\mathsf{twice}^\sharp\ U)\ x) \\ ((\mathsf{twice}^\sharp\ G)\ x) & \rightarrowtail ((\circ^\sharp\ G\ G)\ x) & \end{array} \right\}.$$

Take the reduction pair $\langle \gtrsim, \succ \rangle$ based on the lexicographic path ordering for S-expressions [19] with the precedence $\mathsf{twice} > \mathsf{twice}^\sharp > \circ^\sharp$, $\mathsf{twice} > \circ > \circ^\sharp$, and $\mathsf{map} > :$. Then $\pi(R) \subseteq \gtrsim$ and $\pi(D^\sharp) \subseteq \succ$ are satisfied. Therefore $\Phi(\langle D_2, \mathcal{R}' \rangle) = \{\langle \emptyset, \mathcal{R}' \rangle\}$. Hence the DP problem $\langle D_2, \mathcal{R}' \rangle$ is finite and thus $\mathcal{R}'$ is terminating.

4 Usable rules

The termination of the STSRS $\mathcal{R} = \langle \Sigma, R \rangle$ in Example 1 can be proved using the dependency graph and the subterm criterion processors. However, the termination of combined STSRS $\mathcal{R} \cup \mathcal{R}' = \langle \Sigma \cup \Sigma', R \cup R' \rangle$ of $\mathcal{R}$ (in Example 1)

and $\mathcal{R}' = \langle \Sigma', R' \rangle$ (in Example 8) can not be shown by the processors obtained so far. The problem is that no precedence produces the lexicographic path order satisfying $\pi(D_2^\sharp) \subseteq \succsim$, $\pi(R \cup R') \subseteq \succsim$—the additional constraint $\pi(R) \subseteq \succsim$ from $\mathcal{R}$ makes the reasoning in Example 8 failed. Usable rules criterion guarantees that only rewrite rules *usable* from a set of dependency pairs need to be oriented so that the termination proof can be applied in a modular way. The usable rules criterion for the first-order dependency pairs was first introduced for the innermost termination [5] and later it is extended to the general case [9, 10].

By the presence of function variables, the usual first-order usable rules criterion [9, 10] is not directly applicable in the simply typed setting. More precisely, the usable rules criterion does not hold for the following naive extension of first-order usable rules.

Definition 15 (naive usable rules). A relation $\blacktriangleright_R$ on Σ_d is the smallest quasi-order such that $f \blacktriangleright_R g$ for every simply typed rewrite rule $l \to r \in R$ with $\mathrm{head}(l) = f$ and $g \in \Sigma_\mathrm{d}(r)$. For a set D of head-instantiated dependency pairs, $\mathcal{U}_R(D^\sharp) = \{\ l \to r \in R \mid f \blacktriangleright_R \mathrm{head}(l) \text{ for some } f \in \Sigma_\mathrm{d}(\mathrm{RHS}(D^\sharp))\ \}$ where $\mathrm{RHS}(D^\sharp) = \{r' \mid l' \rightarrowtail r' \in D^\sharp\}$.

Example 9 (counterexample). Let $\mathcal{R} = \langle \Sigma, R \rangle$ be an STSRS where $\Sigma = \{\ \mathsf{0}^\mathrm{o}, \mathsf{f}^{(\mathrm{o}\to\mathrm{o})\times\mathrm{o}\to\mathrm{o}}, \mathsf{g}^{\mathrm{o}\to\mathrm{o}}\ \}$ and $R = \{\ (\mathsf{f}\ F\ \mathsf{0}) \to (\mathsf{f}\ F\ (F\ \mathsf{0})), (\mathsf{g}\ \mathsf{0}) \to \mathsf{0}\ \}$. Then we have $\mathrm{DP}(\mathcal{R}) = \{\ (1)\ (\mathsf{f}\ F\ \mathsf{0}) \rightarrowtail (\mathsf{f}\ F\ (F\ \mathsf{0})), (2)\ (\mathsf{f}\ F\ \mathsf{0}) \rightarrowtail (F\ \mathsf{0})\ \}$. For $D = \{(1)\}$, there is a dependency chain $(\mathsf{f}\ \mathsf{g}\ \mathsf{0}) \rightarrowtail_D (\mathsf{f}\ \mathsf{g}\ (\mathsf{g}\ \mathsf{0})) \to_\mathcal{R} (\mathsf{f}\ \mathsf{g}\ \mathsf{0}) \rightarrowtail_D \cdots$. However $\mathcal{U}_R(D^\sharp) = \emptyset$ and thus there is no infinite dependency chain on D and $\mathcal{U}_R(D^\sharp) \cup \{(\mathsf{cons}\ x\ y) \to x, (\mathsf{cons}\ x\ y) \to y\}$.

This example suggests that the way function variables may be instantiated should be taken into consideration. In what follows, we present a usable rules refinement for simply typed dependency pairs. As in the first-order case, the notion of interpretation [21] is crucial to obtain this.

Definition 16 (interpretation). Let $\mathcal{R} = \langle \Sigma, R \rangle$ be an STSRS, $\Gamma \subseteq \Sigma \times \mathbb{N}$, π an argument filtering, $\mathsf{nil}, \mathsf{cons}$ fresh constants, and $t \in \mathrm{S}^\sharp(\Sigma, V)^\mathrm{ST}$ a terminating expression with the property that $\{s \mid t \to_\mathcal{R}^* s\}$ is finite. Then the *interpretation* $\mathrm{I}_{\Gamma,\pi}(t)$ is an S-expression defined by

$$\mathrm{I}_{\Gamma,\pi}(t) = \begin{cases} \Pi(t) & \text{if } \mathrm{hpair}(t) \notin \Gamma \\ (\mathsf{cons}\ \Pi(t)\ \mathrm{order}(\{\mathrm{I}_{\Gamma,\pi}(u) \mid t \to_\mathcal{R} u\})) & \text{if } \mathrm{hpair}(t) \in \Gamma \end{cases}$$

where

$$\begin{array}{ll} \Pi(a) = a & \\ \Pi((t_0\ t_1 \cdots t_n)) = \mathrm{I}_{\Gamma,\pi}(t_i) & \text{if } \pi(\mathrm{hpair}(t_0)) = i \\ \Pi((t_0\ t_1 \cdots t_n)) = (\mathrm{I}_{\Gamma,\pi}(t_{i_1})\ \cdots\ \mathrm{I}_{\Gamma,\pi}(t_{i_k})) & \text{if } \pi(\mathrm{hpair}(t_0)) = [i_1, \ldots, i_k] \end{array}$$

$$\mathrm{order}(T) = \begin{cases} \mathsf{nil} & \text{if } T = \emptyset \\ (\mathsf{cons}\ t\ \mathrm{order}(T \setminus \{t\})) & \text{if } t \text{ is the minimum element of } T. \end{cases}$$

Here we assume an arbitrary but fixed total order on $\mathrm{S}(\Sigma^\sharp, V)$. Our assumption on t implies that the S-expression order$(\{\mathrm{I}_{\Gamma,\pi}(u) \mid t \to_{\mathcal{R}} u\})$ is well-defined (via an inductive argument [10]). Clearly, when $\mathrm{I}_{\Gamma,\pi}(t)$ is defined, $\mathrm{I}_{\Gamma,\pi}(s)$ is defined for any subexpression s of t. We omit the subscript Γ, π when it is obvious from its context.

For a substitution σ such that $\mathrm{I}(\sigma(x))$ is well-defined for all $x \in \mathrm{Dom}(\sigma)$, we denote by σ_I a substitution defined by $\sigma_\mathrm{I}(x) = \mathrm{I}(\sigma(x))$.

Definition 17 (usable pairs w.r.t. an argument filtering). Let π be an argument filtering and $t \in \mathrm{S}^\sharp(\Sigma, V)^{\mathrm{ST}}$ a simply typed S-expression. Then the set $\mathrm{UP}_\pi(t)$ of *usable pairs* in t w.r.t. π is defined as follows:

$$\mathrm{UP}_\pi(t) = \begin{cases} \{\mathrm{hpair}(t) \mid t \in \Sigma_\mathrm{d}\} \text{ if } t \in \Sigma^\sharp \cup V \\ \{\mathrm{hpair}(t) \mid \mathrm{head}(t) \in \Sigma_\mathrm{d} \cup V\} \cup \mathrm{UP}_\pi(t_i) \\ \qquad\qquad \text{if } t = (t_0 \cdots t_n) \text{ and } \pi(\mathrm{hpair}(t_0)) = i \\ \{\mathrm{hpair}(t) \mid \mathrm{head}(t) \in \Sigma_\mathrm{d} \cup V\} \cup \bigcup_{j=1}^{k} \mathrm{UP}_\pi(t_{i_j}) \\ \qquad\qquad \text{if } t = (t_0 \cdots t_n) \text{ and } \pi(\mathrm{hpair}(t_0)) = [i_1, \ldots, i_k] \end{cases}$$

We also put $\mathrm{UP}_\pi(T) = \{\mathrm{UP}_\pi(t) \mid t \in T\}$ for any set $T \subseteq \mathrm{S}^\sharp(\Sigma, V)^{\mathrm{ST}}$.

Example 10 (usable pairs). Suppose $\pi(\mathsf{fold}, 0) = \pi(\mathsf{fold}, 1) = [0, 1]$. Then we have $\mathrm{UP}_\pi(((\mathsf{fold} + x)\ ys)) = \{\langle \mathsf{fold}, 1\rangle, \langle \mathsf{fold}, 0\rangle, \langle +, 0\rangle\}$.

Let $\mathcal{C}_\mathcal{E} = \langle \Sigma^\sharp \cup \{\mathsf{nil}, \mathsf{cons}\},\ \{(\mathsf{cons}\ x\ y) \to x, (\mathsf{cons}\ x\ y) \to y\}\rangle$ be an SRS.

Lemma 6 (extraction of substitution in interpretation). Let $\mathcal{R} = \langle \Sigma, R\rangle$ be an STSRS, $\Gamma \subseteq \Sigma_\mathrm{d} \times \mathbb{N}$, and $t \in \mathrm{S}^\sharp(\Sigma, V)^{\mathrm{ST}}$. Suppose that σ is a simply typed substitution, π is an argument filtering which is stable on $\mathrm{Stab}(t)$, and $\mathrm{I}_{\Gamma,\pi}(t\sigma)$ is well-defined. Then (1) $\mathrm{I}(t\sigma) \to^*_{\mathcal{C}_\mathcal{E}} \pi(t)\sigma_\mathrm{I}$; (2) if $\mathrm{UP}_\pi(t) \cap \Gamma = \emptyset$ and $\mathrm{type}(F) \neq \mathrm{type}(f) \downarrow k$ for any $\langle F, n\rangle \in \mathrm{UP}_\pi(t)$, $\langle f, n+k\rangle \in \Gamma$ such that $F \in V$, then $\mathrm{I}(t\sigma) = \pi(t)\sigma_\mathrm{I}$.

Proof. By induction on t. □

Definition 18 (simply typed usable rules). Let $\mathcal{R} = \langle \Sigma, R\rangle$ be an STSRS and π an argument filtering.

1. A relation $\blacktriangleright^{\mathrm{ST}}_{R,\pi}$ on $(\Sigma_\mathrm{d} \cup V) \times \mathbb{N}$ is the smallest quasi-order that satisfies:
   (1) $\mathrm{hpair}(l) \blacktriangleright^{\mathrm{ST}}_{R,\pi} \langle a, n\rangle$ for any $l \to r \in R \cup \mathrm{Exp}(R)$ and $\langle a, n\rangle \in \mathrm{UP}_\pi(r)$ where $\mathrm{Exp}(R) = \bigcup_{l\to r\in R} \mathrm{Exp}(l \to r)$,
   (2) $\langle F, n\rangle \blacktriangleright^{\mathrm{ST}}_{R,\pi} \langle f, n+k\rangle$ for any $F \in V$, $f \in \Sigma_\mathrm{d}$ and $n \in \mathbb{N}$ such that $\mathrm{type}(F) = \mathrm{type}(f) \downarrow k$, and
   (3) $\langle f, n+1\rangle \blacktriangleright^{\mathrm{ST}}_{R,\pi} \langle f, n\rangle$ for any $f \in \Sigma_\mathrm{d}$ and $n \in \mathbb{N}$.
2. Let D be a set of head-instantiated dependency pairs. We define the set of usable rules by $\mathcal{U}^{\mathrm{ST}}_{R,\pi}(D^\sharp) = \{l \to r \in R \mid \langle a, n\rangle \blacktriangleright^{\mathrm{ST}}_{R,\pi} \mathrm{hpair}(l)$ for some $\langle a, n\rangle \in \mathrm{UP}_\pi(\mathrm{RHS}(D^\sharp))\}$.

Below we put $\mathcal{U}^{*\mathrm{ST}}_{R,\pi}(D^\sharp) = \{l \to r \in R \cup \mathrm{Exp}(R) \mid \langle a, n\rangle \blacktriangleright^{\mathrm{ST}}_{R,\pi} \mathrm{hpair}(l)$ for some $\langle a, n\rangle \in \mathrm{UP}_\pi(\mathrm{RHS}(D^\sharp))\}$. Note that $\mathcal{U}^{\mathrm{ST}}_{R,\pi}(D^\sharp) = \mathcal{U}^{*\mathrm{ST}}_{R,\pi}(D^\sharp) \cap R$.

Lemma 7 (property of usable rules). Let $\mathcal{R} = \langle \Sigma, R \rangle$ be an STSRS and D a set of head-instantiated dependency pairs, π an argument filtering. Let $U = \mathcal{U}^{\mathrm{ST}}_{R,\pi}(D^\sharp)$, $U^* = \mathcal{U}^{*\mathrm{ST}}_{R,\pi}(D^\sharp)$, $\Gamma = \{\langle f, m\rangle \mid \mathrm{hpair}(l) = \langle f, m'\rangle, m' \leq m, l \to r \in (R \cup \mathrm{Exp}(R)) \setminus U^*\}$, and $t \in \mathrm{RHS}(U^*) \cup \mathrm{RHS}(D^\sharp)$. (1) $\mathrm{UP}_\pi(t) \cap \Gamma = \emptyset$, (2) $\mathrm{type}(F) \neq \mathrm{type}(f) \lfloor k$ for any $\langle F, n\rangle \in \mathrm{UP}_\pi(t)$, $\langle f, n+k\rangle \in \Gamma$ such that $F \in V$.

Proof. Straightforward. □

In the following lemma, only Lemma 7 for the case $t \in \mathrm{RHS}(U) \cup \mathrm{RHS}(D^\sharp)$ is needed.

Lemma 8 (preservation of rewrite step). Let $\mathcal{R} = \langle \Sigma, R \rangle$ be an STSRS, D a set of head-instantiated dependency pairs, and π an argument filtering. Let $U = \mathcal{U}^{\mathrm{ST}}_{R,\pi}(D^\sharp)$ such that π is stable w.r.t. U and D, $U^* = \mathcal{U}^{*\mathrm{ST}}_{R,\pi}(D^\sharp)$, $\Gamma = \{\langle f, m\rangle \mid \mathrm{hpair}(l) = \langle f, m'\rangle, m' \leq m, l \to r \in (R \cup \mathrm{Exp}(R)) \setminus U^*\}$, and $s, t \in \mathrm{S}^\sharp(\Sigma, V)$ such that $\mathrm{I}(s)$ and $\mathrm{I}(t)$ are well-defined. Then (1) $s \to_{\mathcal{R}} t$ implies $\mathrm{I}(s) \to^*_{\mathcal{C}_\mathcal{E} \cup \pi(U)} \mathrm{I}(t)$; (2) $s \rightarrowtail_{D^\sharp} t$ implies $\mathrm{I}(s) \to^*_{\mathcal{C}_\mathcal{E}} \cdot \xrightarrow{\mathrm{r}}_{\pi(D^\sharp)} \mathrm{I}(t)$.

Proof. (1) By induction on s. Use Lemmata 6 and 7. (2) By Lemmata 6 and 7. □

We now arrive at the main result of this section.

Theorem 6 (simply typed usable rules refinement). Let $\mathcal{R} = \langle \Sigma, R \rangle$ be a finitely branching STSRS, and P a set of simply typed DP problems. Suppose Φ is a DP processor on P given by $\Phi(\langle D, \mathcal{R}\rangle) = \{\langle D \backslash D_\succ, \mathcal{R}\rangle\}$ if all dependency pairs in D are head-instantiated and there exists a reduction pair $\langle \gtrsim, \succ\rangle$ on $\mathrm{S}(\Sigma^\sharp, V)$ and an argument filtering π stable w.r.t. U and D, $C_\mathcal{E} \cup \pi(U) \subseteq \gtrsim$, $\pi(D^\sharp) \subseteq \gtrsim$ where $U = \mathcal{U}^{\mathrm{ST}}_{R,\pi}(D^\sharp)$, $D_\succ = \{u \rightarrowtail v \in D \mid \pi(u^\sharp) \succ \pi(v^\sharp)\}$, and $\Phi(\langle D, \mathcal{R}\rangle) = \{\langle D, \mathcal{R}\rangle\}$ otherwise. Then Φ is a sound DP processor on P.

Proof. Suppose $s, t \in \mathrm{NT}_{\min}(R)$ and $s \overset{\mathrm{nh}}{\to}^* \cdot \rightarrowtail_D t$. Since $s, t \in \mathrm{NT}_{\min}(R)$ and R is finitely branching, $\mathrm{I}(s^\sharp), \mathrm{I}(t^\sharp)$ are well-defined. Hence $\mathrm{I}(s^\sharp) \to^*_{\mathcal{C}_\mathcal{E} \cup \pi(U)} \cdot \xrightarrow{\mathrm{r}}_{\pi(D^\sharp)} \mathrm{I}(t^\sharp)$ by Lemma 8. Hence the conclusion follows from Theorem 4. □

Example 11 (termination proof (2)). Consider the combination $\mathcal{R} \cup \mathcal{R}' = \langle \Sigma \cup \Sigma', R \cup R'\rangle$ of the STSRS $\mathcal{R} = \langle \Sigma, R\rangle$ in Example 1 and $\mathcal{R}' = \langle \Sigma', R'\rangle$ in Example 8. Using the head-instantiation, dependency graph, subterm criterion processors, the finiteness of the DP problems other than $\langle D_2, \mathcal{R} \cup \mathcal{R}'\rangle$ is shown. We show the finiteness of the DP problem $\langle D_2, \mathcal{R} \cup \mathcal{R}'\rangle$ using the DP processor Φ given in Theorem 6. Take an argument filtering π such that $\pi(\circ^\sharp, 0) = [0, 1, 2]$, $\pi(\circ^\sharp, 1) = [0]$, $\pi(\circ, 0) = [0, 1, 2]$, $\pi(\circ, 1) = [\,]$, $\pi(\mathsf{twice}^\sharp, 0) = [0, 1]$, and $\pi(\mathsf{twice}^\sharp, 1) = [0]$. Then $U = \mathcal{U}^{\mathrm{ST}}_{R \cup R', \pi}(D_2^\sharp) = \emptyset$, π is stable w.r.t. $\emptyset$ and D_2, and

$$\pi(D_2^\sharp) = \left\{ \begin{array}{lll} ((\circ^\sharp\ (\circ\ U\ V)\ G)) & \rightarrowtail ((\circ^\sharp\ U\ V)) & ((\circ^\sharp\ G\ (\circ\ U\ V))) \rightarrowtail ((\circ^\sharp\ U\ V)) \\ ((\circ^\sharp\ (\mathsf{twice}\ U)\ G)) & \rightarrowtail ((\mathsf{twice}^\sharp\ U)) & ((\circ^\sharp\ G\ (\mathsf{twice}\ U))) \rightarrowtail ((\mathsf{twice}^\sharp\ U)) \\ ((\mathsf{twice}^\sharp\ G)) & \rightarrowtail ((\circ^\sharp\ G\ G)) & \end{array} \right\}.$$

Take the reduction pair $\langle \gtrsim, \succ\rangle$ as in Example 8. Then we have $C_\mathcal{E} \cup \pi(U) \subseteq \succ$ and $D_2^\sharp \subseteq \succ$. Thus $\Phi(\langle D_2, \mathcal{R} \cup \mathcal{R}'\rangle) = \{\langle \emptyset, \mathcal{R} \cup \mathcal{R}'\rangle\}$. Hence the DP problem $\langle D_2, \mathcal{R} \cup \mathcal{R}'\rangle$ is finite and thus we conclude that $\mathcal{R} \cup \mathcal{R}'$ is terminating.

5 Experiments

The techniques in this paper have been implemented based on the dependency pair method for simply typed S-expression rewriting [3], in which dependency pairs, dependency graph, and the subterm criterion have been incorporated from the first-order dependency pairs and the head instantiation technique is introduced. The program consists of about 8,000 lines of code written in SML/NJ. Constraints for ordering satisfiability problems are encoded as SAT-problems and an external SAT-solver is called from the program to solve them. Most of the SAT-encoding methods of these constraints are based on [6, 18]. We employed the lexicographic path relation (comparing from left to right) for S-expressions based on quasi-precedence [19, 20] for the reduction order. The dependency graph processor and the head instantiation are included in default.

For the experiment, the 125 examples used in the experiment in [3] are employed. They consists of typical higher-order functions such as fold, map, rec, filter of various types. All tests have been performed on a PC equipped with 4 Intel Xeon processors of 2.66GHz and a memory of 7GB.

The table below summarizes our experimental result. The columns below the title 'direct' show the results of experiments via our prover. The numbers of examples where termination has been proved are on the column 'success'. The numbers of examples which timeout (in 60 seconds) are on the column 'timeout'. The total time (in seconds) needed to perform the checks for all the examples are on the column 'total'. We compare our result with those obtained by the corresponding first-order dependency pair techniques applied to a (naive) first-order encoding [3] of our examples. They are listed on the columns below the title 'first-order encoding'. We also tested termination proving of our first-order encoded examples using competitive termination provers AProVE 07 [7] and TTT2 [10] for first-order TRSs.

	direct			first-order encoding		
	success	timeout	total	success	timeout	total
reduction pairs	28	0	3.439	43	0	3.830
+ argument filtering	73	0	17.397	53	0	16.606
+ usable rules	121	0	19.145	65	0	17.808
subterm criterion	98	0	3.824	59	0	3.435
+ reduction pairs	103	0	4.183	62	0	4.390
+ argument filtering	115	0	9.420	68	0	13.480
+ usable rules	121	0	12.162	68	0	13.761
AProVE				89	34	2,336.109
TTT2				94	6	1,245.639

The table shows the effectiveness of reduction pairs, argument filtering and usable rules directly formulated in the simply typed S-expression framework.

6 Related works

Another similar framework of binder-free higher-order term rewriting is (simply-typed) *applicative* term rewriting [2, 8, 11, 12], which is called term rewriting with

higher-order variables or simply-typed term rewriting in [13–17]. Applicative terms can be seen as S-expressions on the signature whose function symbols are unary. Applications of first-order dependency pairs for such frameworks are studied in [2, 8, 11].

Dependency pairs and argument filterings for such frameworks are introduced in [13]. Besides the difference of the framework (applicative terms vs. S-expressions), the framework in [13] allows rewrite rules only of basic types. The characterization of dependency chains and dependency pairs in the presence of rewrite rules of function types are introduced by the authors in [3]. ([15] allows rewrite rules of function types but without the special treatment for dependency pairs from argument expansions as in [3].) Another notion of dependency pairs based on strong computability (SC-dependency pairs) is studied in [15–17].

The argument filtering in [13–15, 17] is formulated as a function from Σ to List($\mathbb{N}$)—therefore, there are two limitations compared to the one in this paper: (1) variables are excluded from the domain of argument filterings (e.g., they always have $\pi(((F\ s_1)\ s_2)) = ((F\ \pi(s_1))\ \pi(s_2))$ when $F \in V$), and (2) eliminating head symbols is not allowed (that is, $\pi(((f\ s_1)\ s_2)) = f$ or $\pi(((f\ s_1)\ s_2)) = (f\ \pi(s_i))$ is allowed but $\pi(((f\ s_1)\ s_2)) = (\pi(s_1)\ \pi(s_2))$, $\pi(((f\ s_1)\ s_2)) = \pi(s_i)$, etc. are not). Besides these limitations, some extra conditions on the precedence, the form of the lhs of the rewrite rules and argument filterings are needed (Corollary 7.8 in [13], Theorem 6.12 in [14])—these conditions are simplified in [16] by introducing argument filterings which, instead of eliminating subexpressions, replaces them with new constants.

Compared to the usable rules in this paper, usable rules in [16, 17] (for SC-dependency pairs) need to be closed with an (unusual) extra propagation rule $u \to (\rightarrowtail)\ v \blacktriangleright l \to r$ if $\langle F, n\rangle \in \mathrm{UP}_\pi(u)$, $F \in V(v)$, and $\mathrm{type}(l) \downarrow k = \mathrm{type}(F)$ so that they depend on lhs of the dependency pairs and rewrite rules. In contrast, our usable rules criterion conservatively extends the first-order case. Moreover, we obtain no larger set of usable rules than the one in [16, 17] for the same set of dependency pairs. Hence our criterion is more effective to prove termination.

Acknowledgments

The authors thank Jeoren Ketema, Yoshihito Toyama, Yuki Chiba, and anonymous referees for their helpful comments. This work was partially supported by a grant from JSPS (No. 20500002).

References

1. T. Aoto and T. Yamada. Termination of simply typed term rewriting systems by translation and labelling. In *Proc. of RTA 2003*, 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 *Proc. of HOR 2004*, pages 61–65, 2004.

3. T. Aoto and T. Yamada. Dependency pairs for simply typed term rewriting. In *Proc. of RTA 2005*, volume 3467 of *LNCS*, pages 120–134. Springer-Verlag, 2005.
4. T. Aoto and T. Yamada. Argument filterings and usable rules for simply typed dependency pairs (extended abstract). In *Proc. of HOR 2007*, pages 21–27, 2007.
5. T. Arts and J. Giesl. Termination of term rewriting using dependency pairs. *Theoretical Computer Science*, 236(1–2):133–178, 2000.
6. M. Codish, V. Lagoon, and P. J. Stuckey. Solving partial order constraints for LPO termination. In *Proc. of RTA 2006*, volume 4098 of *LNCS*, pages 4–18. Springer-Verlag, 2006.
7. J. Giesl, P. Schneider-Kamp, and R. Thiemann. AProVE 1.2: Automatic termination proofs in the dependency pair framework. In *Proc. of IJCAR 2006*, volume 4130 of *LNAI*, pages 281–286. Springer-Verlag, 2006.
8. J. Giesl, R. Thiemann, and P. Schneider-Kamp. Proving and disproving termination of higher-order functions. In *Proc. of FroCoS 2005*, volume 3717 of *LNAI*, pages 216–231. Springer-Verlag, 2005.
9. J. Giesl, R. Thiemann, and P. Schneider-Kamp. Mechanizing and improving dependency pairs. *Journal of Automated Reasoning*, 37(3):155–203, 2006.
10. N. Hirokawa and A. Middeldorp. Tyrolean termination tool: Techniques and features. *Information and Computation*, 205(4):474–511, 2007.
11. N. Hirokawa, A. Middeldorp, and H. Zankl. Uncurrying for termination. In *Proc. of LPAR 2008*, volume 5330 of *LNCS*, pages 667–681. Springer-Verlag, 2008.
12. R. Kennaway, J. W. Klop, R. Sleep, and F.-J. de Vries. Comparing curried and uncurried rewriting. *Journal of Symbolic Computation*, 21:57–78, 1996.
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 Trans. on Inf. & Sys.*, E87–D(2):352–359, 2004.
15. K. Kusakari and M. Sakai. Enhancing dependency pair method using strong computability in simply-typed term rewriting. *Applicable Algebra in Engineering, Communication and Computing*, 18(5):407–431, 2007.
16. K. Kusakari and M. Sakai. Static dependency pair method for simply-typed term rewriting and related techniques. *IEICE Trans. on Inf. & Sys.*, E92–D(2):235–247, 2009.
17. T. Sakurai, K. Kusakari, M. Sakai, T. Sakabe, and N. Nishida. Usable rules and labeling product-typed terms for dependency pair method in simply-typed term rewriting systems (in Japanese). *IEICE Trans. on Inf. & Sys.*, J90–D(4):978–989, 2007.
18. P. Schneider-Kamp, R. Thiemann, E. Annov, M. Codish, and J. Giesl. Proving termination using recursive path orders and SAT solving. In *Proc. of FroCoS '07*, volume 4720 of *LNAI*, pages 267–282. Springer-Verlag, 2007.
19. Y. Toyama. Termination of S-expression rewriting systems: Lexicographic path ordering for higher-order terms. In *Proc. of RTA 2004*, volume 3091 of *LNCS*, pages 40–54. Springer-Verlag, 2004.
20. Y. Toyama. Termination proof of S-expression rewriting systems with recursive path relations. In *Proc. of RTA 2008*, volume 5117 of *LNCS*, pages 381–391. Springer-Verlag, 2008.
21. X. Urbain. Modular & incremental automated termination proofs. *Journal of Automated Reasoning*, 32:315–355, 2004.
22. T. Yamada. Confluence and termination of simply typed term rewriting systems. In *Proc. of RTA 2001*, volume 2051 of *LNCS*, pages 338–352. Springer-Verlag, 2001.