Elimination Transformations for Associative-Commutative Rewriting Systems

KUSAKARI Keiichirou[1], **NAKAMURA Masaki**[2], **TOYAMA Yoshihito**[3]

[1]Graduate School of Information Science, Nagoya University,
Furo-cho, Chikusa-ku, Nagoya, 464-8601 Japan.
kusakari@is.nagoya-u.ac.jp

[2]School of Information Science, Japan Advanced Institute of Science and Technology,
Tatsunokuchi, Ishikawa, 923-1292, Japan.
masaki-n@jaist.ac.jp

[3]Research Institute of Electrical Communication, Tohoku University,
Katahira, Aoba-ku, Sendai, 980-8577, Japan.
toyama@nue.riec.tohoku.ac.jp

Abstract. To simplify the task of proving termination and AC-termination of term rewriting systems, elimination transformations have been vigorously studied since the 1990's. Dummy elimination, distribution elimination, general dummy elimination and improved general dummy elimination are examples of elimination transformations. In this paper we clarify the essence of elimination transformations based on the notion of dependency pairs. We first present a theorem that gives a general and essential property for elimination transformations, making them sound with AC-termination. Based on the theorem, we design an elimination transformation called the argument filtering transformation. Next, we clarify the relation among various elimination transformations by comparing them with a corresponding restricted argument filtering transformation. Finally, we compare the AC-dependency pair method with the argument filtering transformation.

Keyword. (AC-)termination, (AC-)dependency pair, argument filtering, elimination transformation

1 Introduction

Term Rewriting Systems (TRSs) are models for computation in which terms are reduced by using a set of directed equations [4, 19, 28, 30]. They are used to represent abstract interpreters of functional programming languages and to model formal manipulation systems used in such applications as program optimization, program verification, and automatic theorem proving. Termination and AC-termination (i.e. the termination of TRSs with associative and commutative equations) are two of the most fundamental properties of TRSs. Even though termination and AC-termination are undecidable properties, several methods for proving termination and AC-termination have been developed [28, 30]. Several transformation methods have also been proposed to simplify the task of proving termination and AC-termination to which these methods cannot be directly applied.

Elimination transformations, which have been actively studied since the 1990's, try to transform a given TRS into a TRS whose termination and AC-termination are easier to prove than the original one. Dummy elimination [8], distribution elimination [25, 35], general dummy elimination [9], and improved general dummy elimination [27] are examples of elimination transformations. In particular, dummy elimination [10] and distribution elimination [29] are still sound for AC-termination, if the TRS to be transformed has only one AC-function symbol to be eliminated.

On the other hand, Arts and Giesl proposed the notion of dependency pairs [2, 3]. Two extensions of dependency pairs to TRSs with AC-function symbols, called the AC-dependency pair, were independently developed by Kusakari and Toyama [22] and by Marché and Urbain [23]. Giesl and Kapur also

Parts of this work were done while K. Kusakari was completing his PhD thesis: "Termination, AC-Termination and Dependence Pairs of Term Rewriting Systems", at Japan Advanced Institute of Science and Technology, School of Information Science (March, 2000). A preliminary version of parts of this article appeared in K. Kusakari, M. Nakamura, Y. Toyama, Argument filtering transformation, Proc. of Int. Conf. on Principle and Practice of Declarative Programming, LNCS 1702 (Springer-Verlag, 1999) pp.47–61.

extended the notion to TRSs with equations, which include AC-equations [13]. Using the notion of AC-dependency pairs, we can easily show the AC-termination property of TRSs to which traditional techniques cannot be applied. To show the (AC-)termination by the (AC-)dependency pair method, the notion of (AC-)reduction pair plays an essential role. The argument filtering method, which helps us locate an appropriate reduction pair, was proposed by Arts and Giesl [2, 3]; this method was also extended to TRSs with AC-function symbols [21, 23].

In this paper we investigate the relationship between the dependency pair method and elimination transformations. Our key observation is that the essence of elimination transformations can be explained by the notion of dependency pairs. We first show a theorem that presents a general and essential property for elimination transformations, making them sound with AC-termination. Indeed, we present remarkably simple proofs for the soundness of these elimination transformations based on this observation, although the original proofs presented in [8, 9, 10, 25, 27, 29, 35] are treated as different methods. This observation also leads to an elimination transformation called the argument filtering transformation, which is not only more powerful than all the other elimination transformations but also is especially useful to clarify the essential relationship hidden behind these transformations. The main contributions of this paper are:

(1) We show that dependency pairs with the argument filtering method in a uniform framework can clearly explain why various elimination transformations work well. Although this result gives a property of AC-termination, we can use dependency pairs instead of AC-dependency pairs. This approach helps in analyzing effectively all the elimination transformations.

(2) An elimination transformation called the argument filtering transformation is proposed. Unlike other elimination transformations, it has been carefully designed to remove all unnecessary rewrite rules, so it is the most powerful elimination transformation.

(3) We clarify the relationship among various elimination transformations by comparing them with a corresponding restricted argument filtering transformation. For example, dummy elimination can be seen as a restricted argument filtering transformation in which argument filtering always removes all arguments, and distribution elimination restricts argument filtering by collapsing into one.

(4) We show that all elimination transformations soundly prove not only termination but also AC-termination. Although the existing results for the soundness of AC-termination of dummy elimination [10] and distribution elimination [29] can be applied only to TRSs that have exactly one AC-function symbol to be eliminated, our method has no such restriction.

(5) We show that the argument filtering transformation and AC-dependency pair method cannot be compared. There exists a TRS such that it is difficult for the AC-dependency pair method to prove the AC-termination, but an argument filtering transformation can transform it into a simple terminating TRS, and vice versa.

The remainder of this paper is organized as follows. The next section provides the preliminaries needed below, and in Section 3, we review the AC-dependency pair method [21, 22]. In Section 4, we use these results to show a general and essential property for elimination transformations that are sound for AC-termination. In Section 5, we propose the argument filtering transformation and show its soundness for AC-termination. In Section 6, we compare various elimination transformations with the argument filtering transformation and give simple proofs of their soundness for AC-termination. Section 7 contains a comparison of the argument filtering transformation with the AC-dependency pair method. Finally, we summarize our results and outline future works in Section 8.

2 Preliminaries

We assume that the reader is familiar with notions of term rewriting systems [4].

A *signature* Σ is a finite set of function symbols, where each $f \in \Sigma$ is associated with a non-negative integer n, written as $arity(f)$. A set V is an enumerable set of variables with $\Sigma \cap V = \emptyset$. The set of *terms* constructed from Σ and V is written as $T(\Sigma, V)$. Identity of terms is denoted by $\equiv$. The set of variables occurring in a term t is denoted by $Var(t)$. A term t is *linear* if every variable in t occurs only once. The set Σ_{AC} of *associative-commutative function symbols*, which have fixed arity 2, is a subset of Σ. The binary relation $\underset{AC}{\sim}$ is the congruence relation generated by $f(f(x,y),z) =_A f(x,f(y,z))$ and

$f(x,y) =_C f(y,x)$ for all $f \in \Sigma_{AC}$. A *position* of a term is a sequence of positive integers. The root position is denoted by the empty sequence ε. The prefix order $\prec$ on term positions is defined by $p \prec q$ iff $pw = q$ for some w $(\neq \varepsilon)$.

A *substitution* $\theta : V \to T(\Sigma, V)$ is a mapping. A substitution over terms is defined as a homomorphic extension. We write $t\theta$ instead of $\theta(t)$. A *context* $C[\,]_p$ is a term with the occurrence of a special constant $\Box$, called a *hole*, at position p. $C[t]_p$ denotes the result of replacing the hole with t, and we sometimes omit p as $C[t]$ if no confusion arises. A term s is called a *subterm* of t if $t \equiv C[s]$ for some context $C[\,]$; s is a *proper subterm* if $s \not\equiv t$. $(t)_p$ denotes the symbol at position p in t, and $t|_p$ denotes the subterm of t at position p.

A *rewrite rule* is a pair of terms, written as $l \to r$, with $l \notin V$ and $Var(l) \supseteq Var(r)$. A *term rewriting system* (TRS) is a finite set of rules. A TRS R is said to be *right-linear* if r is linear for each $l \to r \in R$. The set of *defined symbols* in R is $DF(R) = \{(l)_\varepsilon \mid l \to r \in R\}$. The *reduction relation* $\underset{R}{\to}^p$ and the *AC-reduction relation* $\underset{R/AC}{\to}^p$ are defined as follows:

$$
\begin{aligned}
s \underset{R}{\to}^p t \quad &\overset{\text{def}}{\iff} \quad s \equiv C[l\theta]_p \wedge t \equiv C[r\theta]_p \\
&\qquad\quad \text{for some } l \to r \in R,\ \theta \text{ and } C[\,]_p \\
s \underset{R/AC}{\to}^p t \quad &\overset{\text{def}}{\iff} \quad s \underset{AC}{\sim} C[l\theta]_p \wedge t \equiv C[r\theta]_p \\
&\qquad\quad \text{for some } l \to r \in R,\ \theta \text{ and } C[\,]_p
\end{aligned}
$$

Note that $\underset{R/AC}{\to} = \underset{AC}{\sim} \cdot \underset{R}{\to}$. We often omit the subscripts p, R and R/AC whenever no confusion arises. The transitive-reflexive closure of a binary relation $\to$ is denoted by $\overset{*}{\to}$. The transitive closure of a binary relation $\to$ is denoted by $\overset{+}{\to}$. A TRS R is *terminating* if there exists no infinite reduction sequence such that $t_0 \underset{R}{\to} t_1 \underset{R}{\to} t_2 \underset{R}{\to} \cdots$, and *AC-terminating* if there exists no infinite AC-reduction sequence such that $t_0 \underset{R/AC}{\to} t_1 \underset{R/AC}{\to} t_2 \underset{R/AC}{\to} \cdots$.

A binary relation $>$ is a *strict order* if $>$ is transitive and irreflexive. A binary relation $\geq$ is a *partial order* if $\geq$ is reflexive, transitive and antisymmetric. A binary relation $\gtrsim$ is a *quasi-order* if $\gtrsim$ is transitive and reflexive. The *strict part* of a quasi-order $\gtrsim$ is defined as $\gtrsim \setminus \lesssim$. The *equivalence part* of a quasi-order $\gtrsim$ is defined as $\gtrsim \cap \lesssim$. A *reduction order* $>$ is a well-founded strict order on terms that is monotonic and stable. Here $>$ is *monotonic* iff $s > t \Rightarrow C[s] > C[t]$, and *stable* iff $s > t \Rightarrow s\theta > t\theta$. An *AC-reduction order* $>$ is a reduction order that is *AC-compatible*, i.e., $s \underset{AC}{\sim} s' > t \Rightarrow s > t$.

Proposition 2.1 A TRS R is terminating iff there exists a reduction order $>$ that satisfies $l > r$ for all $l \to r \in R$. A TRS R is AC-terminating iff there exists an AC-reduction order $>$ that satisfies $l > r$ for all $l \to r \in R$.

A *simplification order* $>$ is a reduction order with the subterm property, i.e., $C[t] > t$ for all t and non-empty context $C[\,]$. A TRS R is said to be *simply terminating* if there exists a simplification order $>$ such that $l > r$ for all $l \to r \in R$, and said to be *simply AC-terminating* if there exists a AC-compatible simplification order $>$ such that $l > r$ for all $l \to r \in R$. We define the embedding TRS Emb as follows:

$$Emb = \{f(x_1, \ldots, x_n) \to x_i \mid f \in \Sigma,\ 1 \leq i \leq n = arity(f)\}$$

Proposition 2.2 [20] A TRS R is simply (AC-)terminating iff $R \cup Emb$ is (AC-)terminating.

Although the proof in the literature [20] does not consider AC-symbols, it also works for AC-termination.

3 AC-Dependency Pairs and Argument Filtering Method

In this section, we review notions and results for unmarked and marked AC-dependency pairs in [22] and the argument filtering method in [21], needed later on. We will use the notion of unmarked AC-dependency pairs for analysis of transformation techniques in the next section. The notion of marked AC-dependency pairs will be only applied in the comparison between transformation techniques and the AC-dependency pair method in Section 7.

3.1 Unmarked AC-Dependency Pairs

Definition 3.1 A position p in a term t is said to be a *head position* if $p = \varepsilon$ or $(t)_q = (t)_p \in \Sigma_{AC}$ for all $q \prec p$. We denote by $O_{hd}(t)$ the set of all head positions in a term t. A term t is said to be a *head subterm* of a term s, denoted by $s \trianglerighteq_{hd} t$, if $s \underset{AC}{\sim} C[t]_p$ and $p \in O_{hd}(C[t]_p)$ for some context $C[\,]$. A *body reduction* $s \underset{bd}{\to} t$ is defined by $s \underset{R/AC}{\to}{}^{p} t$ and $p \notin O_{hd}(s)$.

Definition 3.2 Let R be a TRS. We define the set of *unmarked dependency pairs* $DP(R)$ and the set of *AC-extended unmarked dependency pairs* $DP_{ex}(R)$ as follows:

$$\begin{aligned} DP(R) &= \{\langle u, v\rangle \mid u \to C[v] \in R, (v)_\varepsilon \in DF(R)\}, \\ DP_{ex}(R) &= \{\langle f(l,z), f(r,z)\rangle \mid l \to r \in R, (l)_\varepsilon = f \in \Sigma_{AC}\}, \end{aligned}$$

where z is a fresh variable. The *unmarked AC-dependency pairs* $DP_{AC}(R)$ is defined by the union of the unmarked dependency pairs and the AC-extended ones, i.e.,

$$DP_{AC}(R) = DP(R) \cup DP_{ex}(R).$$

Example 3.3 Let $R_1 = \{add(x,0) \to x,\ add(x,s(y)) \to s(add(x,y))\}$ and $\Sigma_{AC} = \{add\}$. Then we obtain $DP(R_1)$, $DP_{ex}(R_1)$ and $DP_{AC}(R_1)$ as follows:

$$\begin{aligned} DP(R_1) &= \{\ \langle add(x,s(y)), add(x,y)\rangle \\ DP_{ex}(R_1) &= \begin{cases} \langle add(add(x,0),z), add(x,z)\rangle \\ \langle add(add(x,s(y)),z), add(s(add(x,y)),z)\rangle \end{cases} \\ DP_{AC}(R_1) &= DP(R_1) \cup DP_{ex}(R_1) \end{aligned}$$

Definition 3.4 A sequence $\langle u_0, v_0\rangle\langle u_1, v_1\rangle\langle u_2, v_2\rangle \cdots$ of unmarked AC-dependency pairs is an *unmarked AC-dependency chain* of R if there exist AC-terminating substitutions[1] θ_i $(i = 0, 1, 2, \ldots)$ such that

$$v_i\theta_i \underset{bd}{\overset{*}{\to}} \trianglerighteq_{hd} u_{i+1}\theta_{i+1} \qquad \text{for all } i.$$

Proposition 3.5 [22] The following properties are equivalent[2].

- A TRS R is AC-terminating.
- There exists no infinite unmarked AC-dependency chain of R.

Next we give the notion of an AC-reduction pair, which is a pair of binary relations for rewrite rules and AC-dependency pairs. As an AC-reduction order guarantees the absence of infinite rewrite sequences, an AC-reduction pair guarantees the absence of infinite AC-dependency chains.

Definition 3.6 A pair $(\gtrsim, >')$ of binary relations on terms is said to be an *AC-reduction pair* if

- $\gtrsim$ is AC-compatible, i.e., $s \underset{AC}{\sim} t \Rightarrow s \gtrsim t$,
- $\gtrsim$ is monotonic and stable,
- $>'$ is stable and well-founded, and
- $\gtrsim \cdot >' \subseteq >'$ or $>' \cdot \gtrsim \subseteq >'$.

An AC-reduction pair $(\gtrsim, >')$ has the *AC-deletion property* if

- $f(f(x,y),z) \gtrsim f(x,y)$ or $f(f(x,y),z) >' f(x,y)$ for all $f \in \Sigma_{AC}$.

Proposition 3.7 [21] The following properties are equivalent.

- A TRS R is AC-terminating.
- There exists an AC-reduction pair $(\gtrsim, >)$ with the AC-deletion property such that $\gtrsim \supseteq R$ and $> \supseteq DP_{AC}(R)$.

[1] A substitution θ is AC-terminating if $x\theta$ is AC-terminating for all $x \in V$.

[2] Although the AC-dependency chain in [22] is defined by substitutions θ_i without the AC-terminating restriction, the proof used AC-terminating substitutions.

3.2 Marked AC-Dependency Pairs

In the dependency pair method, the marking technique works effectively for proving (AC-)termination. Proposition 3.7 holds even if the head function symbols of dependency pairs are marked. Marked dependency pairs are more powerful for proving (AC-)termination than unmarked ones since we can use marked function symbols and unmarked ones separately.

Definition 3.8 The *marked symbol* of a symbol f is denoted by $f^\#$. The *marked term* of a term t, denoted by $t^\#$, is the result of replacing each symbol f at head positions with the marked symbol $f^\#$. We regard $f^\#$ as an AC-symbol for any $f \in \Sigma_{AC}$, i.e., $\Sigma_{AC} := \Sigma_{AC} \cup \{f^\# \mid f \in \Sigma_{AC}\}$.

The *dependency pairs* $DP^\#(R)$, the *AC-extended dependency pairs* $DP^\#_{ex}(R)$ and the *AC-dependency pairs* $DP^\#_{AC}(R)$ are obtained by marking all head positions in $DP(R)$, $DP_{ex}(R)$ and $DP_{AC}(R)$, respectively.

Example 3.9 We consider the TRS R_1 in Example 3.3. Then we obtain $DP^\#_{AC}(R_1)$ as follows:

$$DP^\#_{AC}(R_1) \quad = \quad \left\{ \begin{array}{l} \langle add^\#(x, s(y)), add^\#(x, y)\rangle \\ \langle add^\#(add^\#(x, 0), z), add^\#(x, z)\rangle \\ \langle add^\#(add^\#(x, s(y)), z), add^\#(s(add(x, y)), z)\rangle \end{array} \right.$$

Definition 3.10 An AC-reduction pair $(\gtrsim, >')$ satisfies the *AC-marked condition* if

- $f^\#(f(x, y), z) \sim f^\#(f^\#(x, y), z) \quad$ for all $f \in \Sigma_{AC}$,

where $\sim$ is the equivalence part of $\gtrsim$.

Proposition 3.11 [21] The following properties are equivalent.

- A TRS R is AC-terminating.
- There exists an AC-reduction pair $(\gtrsim, >)$ with the AC-deletion property and the AC-marked condition such that $\gtrsim \supseteq R$ and $> \supseteq DP^\#_{AC}(R)$.

3.3 Argument Filtering Method

The argument filtering method allows us to make an AC-reduction pair from an arbitrary AC-reduction order. In this subsection, we introduce the argument filtering method.

Definition 3.12 An *argument filtering function* π is a function such that $\pi(f)$ is either a positive integer i or a list of positive integers $[i_1, \ldots, i_m]$ where those integers $i, i_1, \ldots, i_m$ are not more than $arity(f)$ (possibly duplications). We can naturally extend $\pi : T(\Sigma, V) \to T(\Sigma^\pi, V)$ over terms as follows:

$$\begin{array}{rcll} \pi(x) & = & x & \text{if } x \in V \\ \pi(f(t_1, \ldots, t_n)) & = & \pi(t_i) & \text{if } \pi(f) = i \\ \pi(f(t_1, \ldots, t_n)) & = & f(\pi(t_{i_1}), \ldots, \pi(t_{i_m})) & \text{if } \pi(f) = [i_1, \ldots, i_m] \end{array}$$

where Σ^π has all function symbols of Σ but the arity of each function symbol is changed into m if $\pi(f) = [i_1, \ldots, i_m]$. An argument filtering function π satisfies the *AC-condition* if $\pi(f)$ is either $[\,]$ or $[1, 2]$ for any AC-symbol f. We define AC-function symbols after argument filtering by $\Sigma^\pi_{AC} = \{f \in \Sigma_{AC} \mid \pi(f) = [1, 2]\}$.

We hereafter assume that if $\pi(f)$ is not defined explicitly then it is intended to be $[1, \ldots, arity(f)]$.

Definition 3.13 Let $>$ be a strict order. We define the *AC-extension* $\gtrsim_{AC}$ by $\gtrsim_{AC} = (> \cup \underset{AC}{\sim})^*$, $s \gtrsim^{sub}_{AC} t$ by $s \gtrsim_{AC} C[t]$ for some $C[\,]$, and $\gtrsim^{sub}_{AC}$ by the strict part of $\gtrsim^{sub}_{AC}$.

Definition 3.14 Let $>$ be a strict order and π an argument filtering function. We define $s \gtrsim_\pi t$ by $\pi(s) \gtrsim_{AC} \pi(t)$, and $s >_\pi t$ by $\pi(s) \gtrsim^{sub}_{AC} \pi(t)$.

Proposition 3.15 [21, 23][3] If $>$ is an AC-reduction order and π is an argument filtering function with the AC-condition then $(\gtrsim_\pi, >_\pi)$ is an AC-reduction pair with the AC-deletion property.

[3] In [23], Marché and Urbain discussed the framework of recursive program schemes which includes the argument filtering method. We slightly improved it by the notion of AC-reduction pairs [21].

4 Soundness Condition for the Elimination Transformations

In this section, using dependency pairs and the argument filtering method, we clarify a general and essential property for elimination transformations to be sound with respect to AC-termination. Here the soundness of a transformation means that the AC-termination of a transformed TRS guarantees that of a given original TRS.

Definition 4.1 We define the including relation $\sqsubseteq$ as follows:

$$R \sqsubseteq R' \iff^{\text{def}} \forall l \to r \in R. \exists C[\,]. \; l \to C[r] \in R'$$

We notice that if there exists a simply (AC-)terminating TRS R' and $R \sqsubseteq R'$ then the TRS R is simply (AC-)terminating.

Theorem 4.2 Let R be a TRS, R' an AC-terminating TRS and π an argument filtering function with the AC-condition. If $\pi(R) \subseteq R'$ and $\pi(DP(R)) \sqsubseteq R'$ then R is AC-terminating.

Taking R as a given TRS and R' as a transformed TRS in an elimination transformation, this simple theorem can uniformly explain why elimination transformations work well with respect to AC-termination. This fact is very interesting because in the original literature the soundness of these elimination transformations was proved by different methods.

Note that the above theorem can be proved straightforwardly if R' includes AC-dependency pairs, i.e. $\pi(DP_{AC}(R)) \sqsubseteq R'$; however, we do not use AC-dependency pairs even though they are a property of AC-termination. This theorem insists that only ordinary dependency pairs, which do not include AC-extended dependency pairs, are sufficient to show the AC-termination of a given TRS. We will prove it by showing that no infinite chain consisting of AC-extended dependency pairs exists under the assumption of $\pi(R) \subseteq R'$, $\pi(DP(R)) \sqsubseteq R'$ and AC-terminating R'. The use of $DP(R)$ helps in effectively analyzing all the elimination transformations. In Sections 5 and 6, we will explain how this theorem simplifies the requirements in elimination transformations through the argument filtering transformation.

Proof. We assume that R is not AC-terminating. From Proposition 3.5, there exists an infinite unmarked AC-dependency chain

$$\langle u_0, v_0\rangle\langle u_1, v_1\rangle\langle u_2, v_2\rangle \cdots$$

and AC-terminating substitutions θ'_i $(i = 0, 1, 2, \ldots)$ such that

$$v_i\theta'_i \underset{bd}{\overset{*}{\to}} \trianglerighteq_{hd} u_{i+1}\theta'_{i+1}$$

for all i.

We define $>$ as $\underset{R'/AC}{\overset{+}{\to}}$. The AC-termination of R' ensures that $>$ is an AC-reduction order. From $\pi(R) \subseteq R'$, it follows that $R \subseteq \gtrsim_\pi$, hence $\underset{bd}{\overset{*}{\to}} \subseteq \gtrsim_\pi$. The inclusion $DP(R) \subseteq >_\pi$ follows from $\pi(DP(R)) \sqsubseteq R'$. From $\pi(R) \subseteq R'$ and the AC-condition of π, for any $\langle f(l,z), f(r,z)\rangle \in DP_{ex}(R)$, we have $f(l,z) \gtrsim_\pi f(r,z)$ and either $\pi(f) = [\,]$ or $\pi(f) = [1,2]$.

From the well-foundedness of $>_\pi$, there exist a number k, an AC-symbol f and $l_i \to r_i \in R$ $(i = k, k+1, \ldots)$ such that $\pi(f) = [\,]$ and $\langle u_i, v_i\rangle = \langle f(l_i, z_i), f(r_i, z_i)\rangle$ for all $i \geq k$. Hence, letting $\theta_i = \theta'_{k+i}$, we obtain the following infinite decreasing sequence:

$$f(l_0,z_0)\theta_0 \underset{R/AC}{\to} f(r_0,z_0)\theta_0 \underset{bd}{\overset{*}{\to}} \trianglerighteq_{hd} f(l_1,z_1)\theta_1 \underset{R/AC}{\to} f(r_1,z_1)\theta_1 \underset{bd}{\overset{*}{\to}} \trianglerighteq_{hd} \cdots$$

where $\pi(f) = [\,]$, $l_i \to r_i \in R$, $z_i \notin Var(l_i)$, and each θ_i is an AC-terminating substitution. Without loss of generality, for each i we suppose that

$$f(r_i,z_i)\theta_i \underset{bd}{\overset{*}{\to}} f(t_i,t'_i) \trianglerighteq_{hd} f(l_{i+1},z_{i+1})\theta_{i+1} \to f(r_{i+1},z_{i+1})\theta_{i+1}$$

such that $r_i\theta_i \overset{*}{\to} t_i$ and $z_i\theta_i \overset{*}{\to} t'_i$.

For any t, we define the multiset $B'_f(t)$ as follows:

$$B'_f(t) = \begin{cases} \{t\} & \text{if } (t)_\varepsilon \neq f \\ \{t|_p \mid p \notin O_{hd}(t),\ \forall q \prec p.\ q \in O_{hd}(t)\} & \text{if } (t)_\varepsilon = f \end{cases}$$

We also define $B_f(t)$ by $|B'_f(t)|$.

Here, if $r_i\theta_i \underset{R/AC}{\overset{*}{\to}} t$ with $(t)_\varepsilon = f$, then it is a contradiction with $f \equiv \pi(l_i\theta_i) \underset{R'/AC}{\to} \pi(r_i\theta_i) \underset{R'/AC}{\overset{*}{\to}} \pi(t) \equiv f$ and the AC-termination of R'. Thus each $r_i\theta_i$ cannot be reduced to any term with the root symbol f. Hence we obtain the following equality:

$$B_f(r_i\theta_i) = B_f(t_i) = 1 \qquad \cdots (i)$$

From the definition of B'_f and $\trianglerighteq_{hd}$, $B'_f(f(t_i, t'_i)) \supseteq B'_f(f(l_{i+1}, z_{i+1})\theta_{i+1})$. Hence we obtain the following inequality:

$$B_f(f(t_i, t'_i)) \geq B_f(f(l_{i+1}, z_{i+1})\theta_{i+1}) \qquad \cdots (ii)$$

Since f is an AC-symbol, $arity(f) = 2$. Hence $B_f(l_{i+1}\theta_{i+1}) \geq 2$. From $B_f(r_{i+1}\theta_{i+1}) = 1$, it follows that

$$B_f(f(l_{i+1}, z_{i+1})\theta_{i+1}) > B_f(f(r_{i+1}, z_{i+1})\theta_{i+1}) \qquad \cdots (iii)$$

Let $n_i = B_f(t'_i) - B_f(z_i\theta_i)$. From $z_i\theta_i \overset{*}{\to} t'_i$ and $f(r_i, z_i\theta_i) \underset{bd}{\overset{*}{\to}} f(t_i, t'_i)$, it follows that

$$\Sigma\{B_f(t) \mid z_i\theta_i \overset{*}{\to} t\} \geq B_f(t'_i) > n_i \geq 0$$

Here, we combine facts (i), (ii) and (iii):

$$\begin{aligned} B_f(f(r_i, z_i)\theta_i) + n_i &= (1 + B_f(z_i\theta_i)) + (B_f(t'_i) - B_f(z_i\theta_i)) \\ &= 1 + B_f(t'_i) \\ &= B_f(f(t_i, t'_i)) \\ &\geq B_f(f(l_{i+1}, z_{i+1})\theta_{i+1}) \\ &> B_f(f(r_{i+1}, z_{i+1})\theta_{i+1}) \end{aligned}$$

Since each t_i cannot be reduced to any term with the root symbol f, either $t_i \underset{AC}{\sim} z_{i+1}\theta_{i+1}$ or $t'_i \trianglerighteq_{hd} z_{i+1}\theta_{i+1}$. If $t_i \underset{AC}{\sim} z_{i+1}\theta_{i+1}$ for some i, then $B_f(t) = 2$ for all t such that $f(r_{i+1}, z_{i+1})\theta_{i+1} \overset{*}{\to} t$. It is a contradiction with $f(r_{i+1}, z_{i+1})\theta_{i+1} \overset{*}{\to} \trianglerighteq_{hd} f(l_{i+2}, z_{i+2})\theta_{i+2}$ and $f(l_{i+2}, z_{i+2})\theta_{i+2} \geq 3$. Suppose that $t'_i \trianglerighteq_{hd} z_{i+1}\theta_{i+1}$ for each i. Then $z_i\theta_i \overset{*}{\to} \trianglerighteq_{hd} z_{i+1}\theta_{i+1}$.

Since $z_0\theta_0$ is AC-terminating, $\Sigma\{B_f(t) \mid z_0\theta_0 \overset{*}{\to} C[t]\}$ is finite. Hence $\Sigma_{i=0}^{\infty} n_i$ is also finite, because of

$$\Sigma\{B_f(t) \mid z_0\theta_0 \overset{*}{\to} C[t]\} \geq \Sigma_{i=0}^{\infty} \Sigma\{B_f(t) \mid z_i\theta_i \overset{*}{\to} t\} > \Sigma_{i=0}^{\infty} n_i.$$

Thus, there exists a natural number k such that $n_i = 0$ for all $i \geq k$. Therefore it is a contradiction with

$$B_f(f(r_k, z_k)\theta_k) > B_f(f(r_{k+1}, z_{k+1})\theta_{k+1}) > B_f(f(r_{k+2}, z_{k+2})\theta_{k+2}) > \cdots$$

□

5 Argument Filtering Transformation

In this section, we propose an elimination transformation called the argument filtering transformation (AFT). Although the AFT can be regarded as a new method to prove the AC-termination, the main purpose is to demonstrate that the existing elimination transformations can be explained by the notion of argument filtering. Two kinds of AFT are shown. The former is denoted by AFT_π, where parameter π is an argument filtering function, and the latter is denoted by $AFT_\pi^{\vec{C}_i}$, where parameter $\vec{C}_i$ is a sequence of contexts. The AFT is directly designed based on Theorem 4.2, which is the essence of elimination transformations.

For an argument filtering function π, the argument filtering function $\overline{\pi}$ is defined as follows: $\overline{\pi}(f) = [i]$ if $\pi(f)$ is an integer i, and $\overline{\pi}(f) = \pi(f)$ if $\pi(f)$ is a list.

Definition 5.1 (Argument Filtering Transformation)
Let π be an argument filtering function. The *argument filtering transformation* (AFT_π) is defined as follows:

$$dec_\pi(t) = \begin{cases} \emptyset & \text{if} \quad t \in V \\ \bigcup_{i \notin \overline{\pi}(f)} \{t_i\} \ \cup \ \bigcup_{i=1}^{n} dec_\pi(t_i) & \text{if} \quad t \equiv f(t_1, \ldots, t_n) \end{cases}$$

$$pick_\pi(T) = \{t \in T \mid \overline{\pi}(t) \text{ includes some defined symbols of } R\}$$

$$AFT_\pi(R) = \pi(R) \cup \{\pi(l) \to \pi(r') \mid l \to r \in R,\ r' \in pick_\pi(dec_\pi(r))\}$$

The main part $\pi(R)$ of a transformed TRS $AFT_\pi(R)$ guarantees that the first condition $\pi(R) \subseteq AFT_\pi(R)$ of Theorem 4.2 is satisfied. Functions dec_π and $pick_\pi$ are designed for satisfying the second condition $\pi(DP(R)) \sqsubseteq AFT_\pi(R)$ of Theorem 4.2. The function dec_π collects the subterms deleted by π. Since only terms having defined symbols are needed for the inclusion $\pi(DP(R)) \sqsubseteq AFT_\pi(R)$, only such terms are picked up by $pick_\pi$. We provide an example of the AFT.

Example 5.2 Let R_2 be a TRS, $\Sigma_{AC} = \{h\}$ and π an argument filtering function such that

$$R_2 = \begin{cases} f(x, f(x, x)) \to f(h(g(0, 1, 2), 3), g'(f(4, 5), 6)) \\ 4 \to 1 \\ 5 \to 1 \end{cases}$$

and $\pi(h) = [\,]$, $\pi(g) = [1, 3]$, and $\pi(g') = 2$. Note that $DF(R_2) = \{f, 4, 5\}$. We denote r instead of $f(h(g(0, 1, 2), 3), g'(f(4, 5), 6))$. Then,

$$\begin{aligned} \pi(r) &= f(h, 6) \\ dec_\pi(r) &= \{g(0, 1, 2),\ 1,\ 3,\ f(4, 5)\} \\ pick_\pi(dec_\pi(r)) &= \{f(4, 5)\} \\ \pi(R_2) &= \{f(x, f(x, x)) \to f(h, 6),\ 4 \to 1,\ 5 \to 1\} \\ AFT_\pi(R_2) &= \pi(R_2) \cup \{f(x, f(x, x)) \to f(4, 5)\} \end{aligned}$$

The termination of $AFT_\pi(R_2)$ is easily proved by the recursive path order [7]. Since $\Sigma^\pi_{AC} = \emptyset$, $AFT_\pi(R_2)$ is trivially AC-terminating. Thus, R_2 is terminating and AC-terminating, if the argument filtering transformation is sound.

To show the soundness of AFT_π, we first show that dec_π takes all deleted subterms that are needed for the inclusion $\pi(DP(R)) \sqsubseteq AFT_\pi(R)$; that is, for a term s and its subterm t, if $\pi(t)$ is not a subterm of $\pi(s)$ then a term having $\pi(t)$ as a subterm is in $\pi(dec_\pi(t))$.

Lemma 5.3 Let $C[\,]$ be a context and t a term. Then, there exists a context $D[\,]$ such that $D[\pi(t)] \in \pi(dec_\pi(C[t]))$ or $D[\pi(t)] \equiv \pi(C[t])$.

Proof. We prove the claim by induction on the structure of $C[\,]$. In the case $C[\,] \equiv \Box$, it is trivial. Suppose that $C[\,] \equiv f(\ldots, t_{i-1}, C'[\,], t_{i+1}, \ldots)$. From the induction hypothesis, there exists a context $D'[\,]$ such that $D'[\pi(t)] \in \pi(dec_\pi(C'[t]))$ or $D'[\pi(t)] \equiv \pi(C'[t])$. In the former case, it follows that $D'[\pi(t)] \in \pi(dec_\pi(C'[t])) \subseteq \pi(dec_\pi(C[t]))$. In the latter case, if $i = \pi(f)$ or $i \in \pi(f)$ then trivial. Otherwise, $D'[\pi(t)] \equiv \pi(C'[t]) \in \pi(dec_\pi(C[t]))$ from the definition of dec_π. □

Theorem 5.4 If $AFT_\pi(R)$ is AC-terminating and π satisfies the AC-condition then R is AC-terminating.

Proof. From the definition, $\pi(R) \subseteq AFT_\pi(R)$. Let $\langle u, v\rangle \in DP(R)$. From the definition of DP, there exists a rule $u \to C[v] \in R$. From Lemma 5.3, there exists a context $D[\,]$ such that $D[\pi(v)] \in \pi(dec_\pi(C[v]))$ or $D[\pi(v)] \equiv \pi(C[v])$. In the former case, from the definition of DP and $\overline{\pi}$, $(\overline{\pi}(v))_\varepsilon$ is a defined symbol. Thus, $D[\pi(v)] \in \pi(pick_\pi(dec_\pi(C[v])))$. Therefore, it follows that $\pi(u) \to D[\pi(v)] \in AFT_\pi(R)$. In the latter case, it follows that $\pi(u) \to D[\pi(v)] \in \pi(R) \subseteq AFT_\pi(R)$. From Theorem 4.2, R is AC-terminating. □

The following two corollaries are obtained as special cases of this theorem, i.e., $\Sigma^\pi_{AC} = \emptyset$ and $\Sigma_{AC} = \emptyset$, respectively.

Corollary 5.5 If $AFT_\pi(R)$ is terminating and $\pi(f) = [\,]$ for each AC-symbol f then R is AC-terminating.

Corollary 5.6 If $AFT_\pi(R)$ is terminating then R is terminating.

From the proof of Theorem 5.4, we can see that the second argument $\{\pi(l) \to \pi(r') \mid l \to r \in R,\ r' \in pick_\pi(dec_\pi(r))\}$ of the definition of the argument filtering transformation AFT_π is used only to keep information of dependency pairs. Thus, introducing redundancy context does not destroy the soundness of the argument filtering transformation. Therefore, we can define another argument filtering transformation $AFT_\pi^{\vec{C}_i}(R)$ as

$$AFT_\pi^{\vec{C}_i}(R) = \{l_1 \to C_1[r_1], \ldots, l_n \to C_n[r_n]\}$$

where $AFT_\pi(R) = \{l_1 \to r_1, \ldots, l_n \to r_n\}$ and $\vec{C}_i$ denotes the list of contexts $C_1[\,], C_2[\,], \ldots, C_n[\,]$. The following theorem about simple termination trivially holds:

Theorem 5.7 If $AFT_\pi^{\vec{C}_i}(R)$ is simply (AC-)terminating, $AFT_\pi(R)$ is simply (AC-)terminating.

We also obtain the following corollary from Theorem 5.4 and Corollaries 5.5 and 5.6:

Corollary 5.8

1. If $AFT_\pi^{\vec{C}_i}(R)$ is AC-terminating and π satisfies the AC-condition then R is AC-terminating.
2. If $AFT_\pi^{\vec{C}_i}(R)$ is terminating and $\pi(f) = [\,]$ for any AC-symbols f then R is AC-terminating.
3. If $AFT_\pi^{\vec{C}_i}(R)$ is terminating then R is terminating.

Efficient choice of π aside, AFT_π may be used for automated termination proofs since the number of π to be considered for the set of function symbols of an input TRS is finite. While the number of argument filterings is finite, it is exponential. Hence, to automate the argument filtering transformation one needs a method to efficiently search for a suitable argument filtering. For the dependency pair approach, such methods were developed in [16, 18]. We feel that these methods can also work well for the argument filtering transformation, but the same cannot be said for $AFT_\pi^{\vec{C}_i}$ because there are infinitely many $\vec{C}_i$. We introduced $AFT_\pi^{\vec{C}_i}$ for comparison with some existing elimination transformations. Although we can provide a way to choose a suitable $\vec{C}_i$ for each elimination transformation, it is not important for the comparison. Since one of the simplest ways to compare termination of two TRSs is to show an inclusion relation of them, such as $R_1 \subseteq R_2$, we want to point out that for each existing elimination transformation E, our $AFT_\pi(R)$ is a subset of $E(R)$ for some π. Unfortunately there exists E such that $AFT_\pi(R)$ is not included in $E(R)$ for any π because of redundancy contexts of $E(R)$. For the extending argument filtering transformation $AFT_\pi^{\vec{C}_i}$, we can say that $AFT_\pi^{\vec{C}_i}(R) \subseteq E(R)$ for some π and $\vec{C}_i$. This is the only reason we introduced $AFT_\pi^{\vec{C}_i}$. If $E(R)$ is simply terminating, so is $AFT_\pi(R)$ from Theorem 5.7 whenever $AFT_\pi^{\vec{C}_i}(R) \subseteq E(R)$. Since the elimination transformations have been designed to transform non-simply terminating TRSs into simply terminating TRSs, it can be said that AFT_π is a more powerful transformation method than E if $AFT_\pi^{\vec{C}_i} \subseteq E(R)$.

6 Elimination Transformations

Elimination transformations were actively studied in the 1990's [8, 9, 10, 25, 27, 29, 35]. In this section, we present remarkable simple proofs for the soundness of existing elimination transformations: the dummy elimination [8, 10], the distribution elimination [25], the general dummy elimination [9], and the improved general dummy elimination [27]. For each elimination transformation, we introduce its definition and properties about soundness and show that it is an instance of the argument filtering transformation. Furthermore, we demonstrate that not only the argument filtering transformation includes all the elimination transformations but also that there exists a non-simply terminating TRS that no elimination transformation can simplify but the argument filtering transformation can transform into a simply terminating TRS.

6.1 Dummy Elimination

The dummy elimination, introduced by Ferreira and Zantema in [8], is the simplest elimination transformation. Moreover, Ferreira showed that the dummy elimination is also sound with respect to AC-termination if only one AC-function symbol is eliminated [10]. In this subsection, we explain that the argument filtering transformation is a proper extension of the dummy elimination, and that the dummy elimination is also sound with respect to AC-termination for an arbitrary number of AC-function symbols.

Definition 6.1 [8](Dummy Elimination) Let e be a function symbol, called an *eliminated symbol*. The *dummy elimination* (DE_e) is defined as follows:

$$\begin{cases} cap_e(x) &= x \\ cap_e(e(t_1,\ldots,t_n)) &= \diamond \\ cap_e(f(t_1,\ldots,t_n)) &= f(cap_e(t_1),\ldots,cap_e(t_n)) \quad \text{if} \quad f \neq e \end{cases}$$

$$\begin{cases} dec_e(x) &= \emptyset \\ dec_e(e(t_1,\ldots,t_n)) &= \bigcup_{i=1}^{n}(\{cap_e(t_i)\} \cup dec_e(t_i)) \\ dec_e(f(t_1,\ldots,t_n)) &= \bigcup_{i=1}^{n} dec_e(t_i) \quad \text{if} \quad f \neq e \end{cases}$$

$$DE_e(R) = \{cap_e(l) \to r' \mid l \to r \in R,\ r' \in \{cap_e(r)\} \cup dec_e(r)\}$$

Example 6.2 Let $t \equiv f(e(0, g(1, e(2,3))), 4)$. Then, $cap_e(t) = f(\diamond, 4)$ and $dec_e(t) = \{0,\ 2,\ 3,\ g(1,\diamond)\}$.

We introduce known results related with the dummy elimination for termination [8] and AC-termination [10].

Proposition 6.3

(a) [8] If $DE_e(R)$ is terminating then R is terminating.

(b) [10] If $DE_e(R)$ is terminating and e is only AC-function symbol then R is AC-terminating.

In the following, we show that the dummy elimination can be expressed by the argument filtering transformation with $\pi(e) = [\,]$. In the case of $\pi(e) = [\,]$, it is noticed that all terms $e(\ldots)$ with the root symbol e are transformed into a constant e by π, i.e. $\pi(e(\ldots)) \equiv e$. We suppose that the constant $\diamond$ occurring in $DE_e(R)$ is identified with the constant e in $AFT_\pi(R)$ without loss of generality.

Lemma 6.4 For $\pi(e) = [\,]$, $AFT_\pi(R) \subseteq DE_e(R)$.

Proof. Let $\pi(e) = [\,]$. Trivially $\pi = cap_e$ and $\pi(pick_\pi(dec_\pi(t))) \subseteq \pi(dec_\pi(t))$. We firstly prove $\pi(dec_\pi(t)) = dec_e(t)$ by induction on t. The case $t \in V$ is trivial. Suppose that $t \equiv f(t_1,\ldots,t_n)$. In the case $f \neq e$,

$$\begin{aligned} \pi(dec_\pi(f(t_1,\ldots,t_n))) &= \bigcup_{i=1}^{n} \pi(dec_\pi(t_i)) \\ &= \bigcup_{i=1}^{n} dec_e(t_i) \\ &= dec_e(f(t_1,\ldots,t_n)) \end{aligned}$$

In the case $f = e$,

$$\begin{aligned} \pi(dec_\pi(e(t_1,\ldots,t_n))) &= \bigcup_{i=1}^{n}(\{\pi(t_i)\} \cup \pi(dec_\pi(t_i))) \\ &= \bigcup_{i=1}^{n}(\{cap_e(t_i)\} \cup dec_e(t_i)) \\ &= dec_e(e(t_1,\ldots,t_n)). \end{aligned}$$

Next we assume $l \to r \in AFT_\pi(R)$ is an arbitrary rule in $AFT_\pi(R)$. From the definition of AFT_π, there exists $l' \to r' \in R$ such that $l \equiv \pi(l')$ and $r \in \pi(pick_\pi(dec_\pi(r')))$. From the definition of DE_e, there exists $cap_e(l') \to r'' \in DE_e(R)$ for each $r'' \in \{cap_e(r')\} \cup dec_e(r')$. Since $\pi(t) \equiv cap_e(t)$ and $\pi(pick_\pi(dec_\pi(t))) \subseteq dec_e(t)$ for each t, the rule $l \to r$ also exists in $DE_e(R)$. □

Theorem 6.5 Let $\pi(e) = [\,]$. If $DE_e(R)$ is AC-terminating or simply AC-terminating then so is $AFT_\pi(R)$, respectively.

Proof. From Lemma 6.4. □

This theorem means that the argument filtering transformation is a proper extension of the dummy elimination. Hence we obtain the following corollary from Theorems 5.4 and 6.5.

Corollary 6.6 If $DE_e(R)$ is AC-terminating then R is AC-terminating.

We notice that known results (a) and (b) in Proposition 6.3 are special cases $\Sigma_{AC} = \emptyset$ and $\Sigma_{AC} = \{e\}$ of Corollary 6.6, respectively.

6.2 Distribution Elimination

The distribution elimination, introduced by Zantema in [35], is an exceptional elimination transformation: it is not sound with respect to termination and AC-termination in general. The distribution elimination requires the right-linearity of transformed TRS. After then, Ohsaki, Middeldorp, and Giesl showed that the distribution elimination is also sound with respect to AC-termination if the eliminated symbol is only one AC-function symbol [29]. On the other hand, Middeldorp, Ohsaki, and Zantema gave another restriction without right-linearity to ensure the soundness of termination [25].

Definition 6.7 [35](Distribution Elimination) Let e be an eliminated symbol with $arity(e) > 0$. A rule $l \to r$ is said to be a *distribution rule* for e if $l \equiv C[e(x_1, \ldots, x_n)]$ and $r \equiv e(C[x_1], \ldots, C[x_n])$ for some pairwise different variables $x_1, \ldots, x_n$ and some non-empty context $C[\,]$ in which e does not occur. The *distribution elimination* (DIS_e) is defined as follows:

$$E_e(t) = \begin{cases} \{t\} & \text{if} \quad t \in V \\ \bigcup_{i=1}^{n} E_e(t_i) & \text{if} \quad t \equiv e(t_1, \ldots, t_n) \\ \{f(s_1, \ldots, s_n) \mid s_i \in E_e(t_i)\} & \text{if} \quad t \equiv f(t_1, \ldots, t_n) \\ & \quad \text{with } f \neq e \end{cases}$$

$$DIS_e(R) = \{l \to r' \mid l \to r \in R \text{ is not a distribution rule for } e, r' \in E_e(r)\}$$

Example 6.8 Let $t \equiv f(e(0, g(1, e(2, 3))), 4)$.
Then, $E_e(t) = \{f(0, 4),\ f(g(1, 2), 4),\ f(g(1, 3), 4)\}$.

We introduce known results related to the distribution elimination for termination [35] and AC-termination [29] with the restriction of right-linearity, and termination without right-linearity [25].

Proposition 6.9

- Suppose that each rule $l \to r \in R$ is a distribution rule or a rule in which the eliminated symbol e does not occur in l.
  - **(a)** [35] If $DIS_e(R)$ is terminating and right-linear then R is terminating.
  - **(b)** [29] If $DIS_e(R)$ is terminating, right-linear and e is only AC-symbol (i.e. $\Sigma_{AC} = \{e\}$) then R is AC-terminating.
- Suppose that the eliminated symbol e does not occur in l for each rule $l \to r \in R$.
  - **(c)** [25] If $DIS_e(R)$ is terminating then R is terminating.

In the following, we explain how the distribution elimination is expressed as a restricted argument filtering transformation, and generalize the known result (c) in this proposition to handle AC-function symbols. The corresponding argument filtering is $\pi(e) = i$ for some $i = 1, \ldots, arity(e)$.

Lemma 6.10 Suppose that the eliminated symbol e does not occur in l for all $l \to r \in R$. For any $i \in \{1, \ldots, arity(e)\}$, if $\pi(e) = i$, then $AFT_\pi^{\vec{C}_i}(R) \subseteq DIS_e(R)$ for some $\vec{C}_i$.

Proof. From the definition of AFT_π, for any $l_i \to r_i \in AFT_\pi(R)$ there exists a rule $l_i \to C'[r'] \in R$ such that $r \equiv \pi(r')$. It is easily proved by induction on $C'[\,]$ that for any r' and $C'[\,]$, there exists a context $C[\,]$ such that $C[\pi(r')] \in E_e(C'[r'])$. Hence there exists $l_i \to C_i[r_i] \in DIS_e(R)$. For these C_i we get $AFT_\pi^{\vec{C}_i}(R) \subseteq DIS_e(R)$. □

Theorem 6.11 Suppose that the eliminated symbol e does not occur in l for all $l \to r \in R$. We choose $\pi(e) = i$ for some $i = 1, \ldots, arity(e)$. If $DIS_e(R)$ is AC-terminating or terminating then so is $AFT_\pi^{\vec{C}_i}(R)$ for some $\vec{C}_i$, respectively. If $DIS_e(R)$ is simply AC-terminating or simply terminating then so is $AFT_\pi(R)$, respectively.

Proof. From Lemma 6.10 and Theorem 5.7. □

From Corollary 5.8 and Theorem 6.11, we obtain the following corollary, which is a generalization of the known result (c) in Proposition 6.9.

Corollary 6.12 Suppose that the eliminated symbol e is not AC-symbol and the symbol e does not occur in l for each rule $l \to r \in R$.
If $DIS_e(R)$ is AC-terminating then R is AC-terminating.

The assumption $e \notin \Sigma_{AC}$ is needed since Theorem 5.4 requires the AC-condition of π but $\pi(e) = i$ for some $i = 1, \ldots, arity(e)$ in Theorem 6.11. Hence e should be non-AC-symbol. We notice that the known result (c) in Proposition 6.9 is a special case $\Sigma_{AC} = \emptyset$ of corollary 6.12.

Unfortunately, Corollary 6.12 cannot handle known results (a) and (b) in Proposition 6.9 related to the distribution elimination for right-linear TRSs with at most one AC-symbol. The reason is that the distribution transformation deletes some rules, and hence delete some dependency pairs. In [21], readers can see that we extended the notion of the argument filtering method onto AC-multiset, and discussed why the distribution elimination with the right-linearity works well.

6.3 General Dummy Elimination

To prove termination, Ferreira introduced the general dummy elimination by placing the dummy elimination and the distribution elimination together [9].

Definition 6.13 [9](General Dummy Elimination) For any $f \in \Sigma$, an *f-status* τ satisfies $\tau(f) = (\emptyset, 0)$ or (J, j) with $j \in J \subseteq \{1, \ldots, arity(f)\}$. Let e be an eliminated symbol and $\tau(e) = (I, i)$. The *general dummy elimination* (GDE_e) is defined as follows:

$$cap_i(t) = \begin{cases} t & \text{if} \quad t \in V \\ f(cap_i(t_1), \ldots, cap_i(t_n)) & \text{if} \quad t \equiv f(t_1, \ldots, t_n) \wedge f \neq e \\ cap_i(t_i) & \text{if} \quad t \equiv e(t_1, \ldots, t_n) \wedge i \neq 0 \\ \diamond & \text{if} \quad t \equiv e(t_1, \ldots, t_n) \wedge i = 0 \end{cases}$$

$$E_i(t) = \begin{cases} \{t\} & \text{if} \quad t \in V \\ \{f(s_1, \ldots, s_n) \mid s_j \in E_i(t_j)\} & \text{if} \quad t \equiv f(t_1, \ldots, t_n) \wedge f \neq e \\ E(t_i) & \text{if} \quad t \equiv e(t_1, \ldots, t_n) \end{cases}$$

$$E(t) = \begin{cases} \{t\} & \text{if} \quad t \in V \\ \{cap_0(t)\} & \text{if} \quad I = \emptyset \\ \bigcup_{j \in I} E_j(t) & \text{if} \quad I \neq \emptyset \end{cases}$$

$$dec(t) = \begin{cases} \emptyset & \text{if} \quad t \in V \\ \bigcup_{j=1}^n dec(t_j) & \text{if} \quad t \equiv f(t_1, \ldots, t_n) \wedge f \neq e \\ \bigcup_{j=1}^n dec(t_j) \cup \bigcup_{j \notin I} E(t_j) & \text{if} \quad t \equiv e(t_1, \ldots, t_n) \end{cases}$$

$$GDE_e(R) = \{cap_i(l) \to r' \mid l \to r \in R,\ r' \in E(r) \cup dec(r)\}$$

The general dummy elimination was extended to the improved general dummy elimination ($IGDE$) by removing several unnecessary rewrite rules [27]. It corresponds to $pick_\pi$ in the AFT.

Definition 6.14 [27](Improved General Dummy Elimination) The functions τ, cap_i, E and dec are the same as that of the general dummy elimination. Let $\tau(e) = (I, i)$. The *improved general dummy*

elimination ($IGDE_e$) is defined as $IGDE_e(R) = GDE_e(R)$ if $e \in DF(R)$; otherwise,

$$\begin{array}{rcl} E'(t) & = & \{s \in E(t) \mid \ s \text{ includes some defined symbols of } R\} \\ dec'(t) & = & \{s \in dec(t) \mid \ s \text{ includes some defined symbols of } R\} \\ IGDE_e(R) & = & \{cap_i(l) \to r' \mid l \to r \in R,\ r' \in \{cap_i(r)\} \cup E'(r) \cup dec'(r)\} \end{array}$$

Example 6.15 Let $t \equiv f(0, e(f(1, e(2, 3, 4)), 5, 6))$ and $\tau(e) = (\{1, 3\}, 1)$. Then, we have $E(t) = \{f(0, 6),\ f(0, f(1, 2)),\ f(0, f(1, 4))\}$ and $dec(t) = \{5,\ 3\}$. If $DF(R) = \{f, 5\}$ then $E'(t) = E(t)$ and $dec'(t) = \{5\}$.

We introduce known results in [9, 27] related to the general dummy elimination and the improved general dummy elimination.

Proposition 6.16 [9, 27] If $GDE_e(R)$ (or $IGDE_e(R)$) is terminating then R is terminating.

In this subsection, we show that the general dummy elimination and the improved general dummy elimination also work well for proving AC-termination, and provide a remarkable simple proof for its soundness. In the following, we explain how the (improved) general dummy elimination can also be expressed as a restricted argument filtering transformation where $\pi(e) = [\,]$ if $\tau(e) = (\emptyset, 0)$, or $\pi(e) = i$ if $\tau(e) = (I, i)$ $(i \neq 0)$. Here we also regard e as $\diamond$ when $\pi(e) = [\,]$.

Lemma 6.17 Let $\tau(e) = (I, i)$. We take $\pi(e) = [\,]$ if $i = 0$, or $\pi(e) = i$ if $i \neq 0$. Then $AFT_\pi^{\vec{C}_i}(R) \subseteq IGDE_e(R)$ for some $\vec{C}_i$.

Proof. In the case $\tau(e) = (\emptyset, 0)$, since $GDE_e = DE_e(R)$ holds trivially, $AFT_\pi^{\vec{C}_i}(R) \subseteq GDE_e(R)$ holds where each C_i is the trivial context $\Box$, i.e. $AFT_\pi^{\vec{C}_i} = AFT_\pi$, from Lemma 6.4. Each rule in $GDE_e(R) \setminus IGDE_e(R)$ is not included in $AFT_\pi^{\vec{C}_i}(R)$ since $pick_\pi$ should not pick up such rule. Hence $AFT_\pi(R) \subseteq GDE_e(R)$.

Consider the case $\tau(e) = (I, i)$ with $i \neq 0$. For any $l_j \to r_j \in AFT_\pi(R)$ there exists a rule $l' \to C'[r'] \in R$ with $l_j \equiv \pi(l')$ and $r_j \equiv \pi(r')$ from the definition of AFT_π. It is easily proved by induction on $C'[\,]$ that for any r' and $C'[\,]$, there exists a context $C[\,]$ such that $C[\pi(r')] \in dec(C'[r']) \cup E(C'[r'])$. Since $\pi(l') \equiv cap_i(l')(\equiv l_j)$ holds too, there exists $l_j \to C_j[r_j] \in GDE_e(R)$. For these $\vec{C}_j$ we get $AFT_\pi^{\vec{C}_j}(R) \subseteq GDE_e(R)$. From the same reason in the above case, no rule in $GDE_e(R) \setminus IGDE_e(R)$ is included in $AFT_\pi^{\vec{C}_i}(R)$. Hence $AFT_\pi^{\vec{C}_j}(R) \subseteq IGDE_e(R)$. $\Box$

Theorem 6.18 Suppose that $\tau(e) = (I, i)$. If $GDE_e(R)$ (or $IGDE_e(R)$) is AC-terminating or terminating then so is $AFT_\pi^{\vec{C}_i}(R)$, respectively. If $GDE_e(R)$ (or $IGDE_e(R)$) is simply AC-terminating or simply terminating then so is $AFT_\pi(R)$, respectively.

Proof. From Lemma 6.17 and Theorem 5.7. $\Box$

From Corollary 5.8 and Theorem 6.18, we obtain the following corollary, which means that the (improved) general dummy elimination also works well for proving AC-termination.

Corollary 6.19 Let $\tau(e) = (I, i)$ and either $e \notin \Sigma_{AC}$ or $i = 0$.
If $GDE_e(R)$ (or $IGDE_e(R)$) is AC-terminating then R is AC-terminating.

Note that the assumption "$e \notin \Sigma_{AC}$ or $i = 0$" guarantees the argument filtering function π to satisfy the AC-condition. We notice that Proposition 6.16 is a special case $\Sigma_{AC} = \emptyset$ of this corollary.

Example 6.20 Consider the following TRS with $\Sigma_{AC} = \{h\}$.

$$R_3 = \left\{ \begin{array}{rcl} f(f(x)) & \to & f(g(f(x), x)) \\ f(f(x)) & \to & f(h(f(x), f(x))) \\ f(x) & \to & x \\ g(x, y) & \to & y \\ h(x, x) & \to & g(x, 0) \end{array} \right.$$

Let $\pi(g) = [2]$ and $\pi(h) = [\,]$. Then,

$$AFT_\pi(R_3) = \left\{ \begin{array}{rcl} f(f(x)) & \to & f(g(x)) \\ f(f(x)) & \to & f(x) \\ f(f(x)) & \to & f(h) \\ f(f(x)) & \to & x \\ g(y) & \to & y \\ h & \to & g(0) \end{array} \right.$$

The termination of $AFT_\pi(R_3)$ is easily proved by the recursive path order with the precedence $f \rhd h \rhd g \rhd 0$ [7]. From Corollary 5.5, R_3 is AC-terminating. The improved general dummy elimination cannot transform it into a simply terminating one. It means that $IGDE_f(R_3)$, $IGDE_g(R_3)$ and $IGDE_h(R_3)$ are not terminating for any status τ. The dummy elimination, the distribution elimination and the general dummy elimination cannot, too. Note that the AC-termination of R_3 is not easily proved, because R_3 is not simply AC-terminating.

7 Comparison of Argument Filtering Transformation with AC-Dependency Pair Method

We have unified all the elimination transformations by the argument filtering transformation, which is based on notions of dependency pairs and argument filterings. This naturally leads to the question of: where is the argument filtering transformation more useful? Where is the AC-dependency pair method more useful? In this section, we study this relationship.

7.1 AC-Dependency Pairs with Simplification Orders

In this subsection, we examine the relation of a most basic framework: for a simplification order $>$, the AC-termination of R is shown as $> \supseteq AFT_\pi(R)$ (cf. Theorem 5.4 and Proposition 2.2) and as $\gtrsim_\pi \supseteq R$ and $>_\pi \supseteq DP^\#_{AC}(R)$ (cf. Propositions 3.11 and 3.15).

For ordinary termination cases without AC-symbols, as a direct consequence of Theorem 4.2 and Definition 3.14, we can see that the dependency pair method successfully proves termination whenever the argument filtering transformation successfully does so. Moreover, there exists an example for which termination can be proved by the dependency pair method but cannot be treated well by the argument filtering transformation (described below). Giesl and Middeldorp studied the relationship more precisely by also considering dependency graphs [12].

For AC-termination, the situation is different. Indeed, the argument filtering transformation and the basic AC-dependency pair method are incomparable; there exists a TRS whose AC-termination can be proved by the argument filtering transformation but cannot be proved by the AC-dependency pair method, and vice versa. One answer is that the argument filtering transformation is more useful when there exists a necessity of $\pi(f) = [\,]$ for some defined AC-symbol f; otherwise the AC-dependency pair method is more useful.

First, we consider $\pi(f) = [\,]$ for some defined AC-symbol f. Here the dependency pair method always fails because this π transforms an AC-extended dependency pair $\langle f(l,z)^\#, f(r,z)^\# \rangle$ into $\langle f^\#, f^\# \rangle$. Trivially, $f^\# \not> _\pi f^\#$ for any $>$ and π. Note that $\pi(f) = [\,]$ requires $\pi(f^\#) = [\,]$ to ensure the AC-marked condition. Hence, in these cases the argument filtering transformation is more useful than the dependency pair method. The following example is a typical case:

Example 7.1

$$R_4 = \left\{ \begin{array}{rcl} f(f(x)) & \to & f(h(f(x), f(x))) \\ f(x) & \to & x \\ h(x,x) & \to & a \end{array} \right.$$

Let $\pi(h) = [\,]$. Then,

$$AFT_\pi(R_4) = \left\{ \begin{array}{rcl} f(f(x)) & \to & f(h) \\ f(f(x)) & \to & f(x) \\ f(x) & \to & x \\ h & \to & a \end{array} \right.$$

The termination of $AFT_\pi(R_4)$ is easily proved by the recursive path order with the precedence $f \rhd h \rhd a$ [7]. Hence the AC-termination of R_4 follows from Corollary 5.5.

Next we consider $\pi(f) = [1, 2]$ for all defined AC-symbols. Here, if the argument filtering transformation successfully proves the AC-termination of TRS R by an AC-compatible simplification order, then the dependency pair method also successfully proves AC-termination by the same simplification order because $\pi(R) \cup \pi(DP(R)) \sqsubseteq AFT_\pi(R)$. Again we ask a natural question: in what cases is the AC-dependency pair method strictly more useful? To answer this, we prepare an abstract theorem.

Theorem 7.2 Suppose that $\pi(R) \cup \pi(DP(R)) \sqsubseteq R'$ and $\pi(f) = [1, 2]$ for all $f \in \Sigma_{AC} \cap DF(R)$. If $R \cup DP(R)$ is not AC-terminating, then R' is not simply AC-terminating.

Proof. Assume that R' is simply AC-terminating. We define $>$ as $\xrightarrow[R' \cup Emb/AC]{+}$. The AC-termination of R' ensures that $>$ is an AC-compatible simplification order. It is trivial that $\pi(R_D) \cup \pi(DP(R_D)) \sqsubseteq R'$ where $R_D = R \cup DP(R)$. Thus, $l \gtrsim_\pi r$ for all $l \to r \in R_D$ and $u >_\pi v$ for all $\langle u, v\rangle \in DP(R_D)$. For any extended unmarked dependency pair $\langle f(l, z), f(r, z)\rangle$, it follows that $\pi(f(l, z)) \equiv f(\pi(l), z) > f(\pi(r), z) \equiv \pi(f(r, z))$. From Proposition 3.7, R_D is AC-terminating. It is a contradiction. □

In the case of $\pi(f) = [1, 2]$ for each defined AC-symbol f, this theorem means that if $R \cup DP(R)$ is not AC-terminating, then $AFT_\pi(R)$ is not simply AC-terminating because $\pi(R) \cup \pi(DP(R)) \sqsubseteq AFT_\pi(R)$. Hence the argument filtering transformation with AC-compatible simplification orders always fails to prove AC-termination. We give the following example such that R is AC-terminating but $R \cup DP(R)$ is not.

Example 7.3 Let R_5 be the following TRS with $\Sigma_{AC} = \{h\}$.

$$R_5 = \left\{ \begin{array}{rcl} f(a) & \to & f(b) \\ b & \to & g(h(a, a), a) \\ h(x, x) & \to & x \end{array} \right. \qquad DP(R_5) = \left\{ \begin{array}{rcl} f(a) & \to & f(b) \\ f(a) & \to & b \\ b & \to & h(a, a) \end{array} \right.$$

From $f(a) \underset{R_5}{\to} f(b) \underset{DP(R_5)}{\to} f(h(a, a)) \underset{R_5}{\to} f(a)$, $R_5 \cup DP(R_5)$ is not AC-terminating. Hence, from Theorem 7.2, the argument filtering transformation with AC-compatible simplification orders always fails to prove the AC-termination. On the other hand, the AC-dependency pair method succeeds to prove the AC-termination. In fact, we suppose that $\pi(g) = [\,]$, and $>$ is the recursive path order with flattening based on the precedence $b^\# \rhd a \rhd b \rhd g$ and $f^\# \rhd b^\# \rhd h^\#$ [5]. Then $l \gtrsim_\pi r$ for all $l \to r \in R$, and $u^\# >_\pi v^\#$ for all $\langle u^\#, v^\#\rangle \in DP^\#_{AC}(R)$. From Proposition 3.7, we succeed to prove the AC-termination of R.

The AC-dependency method succeeds but the argument filtering transformation fails in this example because of the mark of function symbols. Because of the marks, a function symbol and the marked one can be used as different function symbols, such as $b^\# \rhd a \rhd b \rhd g$.

7.2 AC-Dependency Pairs with Advanced Methods

We have seen that the argument filtering transformation is more useful if there exists a necessity of $\pi(f) = [\,]$ for some defined AC-symbol f. For example, both AC-terminations of TRSs R_3 (Example 6.20) and R_4 (Example 7.1) are easily proved by the argument filtering transformation; however, these AC-terminations are difficult to be proved by using the AC-dependency pair method, as defined in Proposition 3.15.

On the other hand, Proposition 3.15 is the most basic version of the AC-dependency pair method. By using more advanced refinements, the AC-dependency pair method can be improved. Examples for such refinements are methods based on dependency graphs [1, 2, 3, 15, 26] and criteria for CE-termination for hierarchical unions [16, 24, 31, 32, 33, 34]. These methods have already been implemented in AProVE [17] and CiME [6], which to the best of our knowledge are the most powerful automatic provers for AC-termination. We can prove the AC-termination of R_3 and R_4 by the AC-dependency pair method with these advanced methods. Actually, AProVE and CiME can prove the AC-termination of R_3 and R_4 because such advanced methods remove the necessity of $\pi(h) = [\,]$. Under which condition can we remove the necessity of $\pi(h) = [\,]$ for each defined AC-symbol h? This is a very difficult question. Finally, we show a TRS in which it is very difficult to remove the necessity of $\pi(h) = [\,]$ for some defined AC-symbol h.

Example 7.4 Let R_6 be the following TRS with $\Sigma_{AC} = \{h\}$.

$$R_6 = \left\{ \begin{array}{rcl} f(f(s(x))) & \to & f(h(f(s(x)), x)) \\ f(h(x, y)) & \to & h(x, y) \\ h(x, y) & \to & g(0) \\ g(x) & \to & f(f(x)) \end{array} \right.$$

Let $\pi(h) = [\,]$. Then,

$$AFT_\pi(R_6) = \left\{ \begin{array}{rcl} f(f(s(x))) & \to & f(h) \\ f(f(s(x))) & \to & f(s(x)) \\ f(h) & \to & h \\ h & \to & g(0) \\ g(x) & \to & f(f(x)) \\ g(x) & \to & f(x) \end{array} \right.$$

The termination of $AFT_\pi(R_6)$ is easily proved by the recursive path order with the precedence $s \rhd h \rhd 0 \rhd g \rhd f$ [7]. Hence the AC-termination of R_6 follows from Corollary 5.5.

In this example, the choice $\pi(h) = [\,]$ for a defined AC-symbol h strongly supports the proof of AC-termination by the argument filtering transformation. AProVE and CiME, the prover of which is based on the AC-dependency pair method, failed to prove the AC-termination. They could not choose $\pi(h) = [\,]$ because such π interprets the AC-extended dependency pair $\langle h^\#(h^\#(x, y), z), h^\#(g(0), z)\rangle$ into $\langle h^\#, h^\#\rangle$. Note that the argument filtering transformation do not need to analyze AC-extended dependency pairs.

8 Concluding Remarks

Elimination Transformations: In Section 6, we unified all elimination transformations by comparing them with corresponding restricted argument filtering transformations. We summarize that each elimination transformation is expressed as a restricted argument filtering transformation (Fig. 1), with e denoted as the eliminated symbol.

Dummy Elimination (DE_e)

$$\pi(f) = \left\{ \begin{array}{ll} [\,] & \text{if} \quad f = e \\ [1, \ldots, arity(f)] & \text{if} \quad f \neq e \end{array} \right.$$

Distribution Elimination (DIS_e)

$$\pi(f) = \left\{ \begin{array}{ll} i & \text{if} \quad f = e \\ [1, \ldots, arity(f)] & \text{if} \quad f \neq e \end{array} \right.$$

where i is an arbitrary number such that $1 \leq i \leq arity(e)$.

(Improved) General Dummy Elimination with $\tau(e) = (I, i)$ ($(I)GDE_e$)

$$\pi(f) = \left\{ \begin{array}{ll} [\,] & \text{if} \quad f = e \wedge i = 0 \\ i & \text{if} \quad f = e \wedge i \neq 0 \\ [1, \ldots, arity(f)] & \text{if} \quad f \neq e \end{array} \right.$$

AC-dependency Pair Method and Argument Filtering Transformation: We summarize the discussion in Section 7 that compares the argument filtering transformation and the AC-dependency pair method.

When $\pi(h) = [\,]$ for some defined AC-symbol h:

The argument filtering transformation is more useful than the AC-dependency pair method. Indeed, the basic AC-dependency pair method always fails to prove the AC-termination (cf. the discussion above Example 7.1). Moreover, there exist some TRSs for which AC-termination can be proved by the argument filtering transformation, but cannot be proved by existing tools based on the AC-dependency pair method with advanced methods (cf. Example 7.4).

One advantage of the argument filtering transformation is its ability to ignore AC-extended dependency pairs to prove AC-termination (Theorem 4.2).

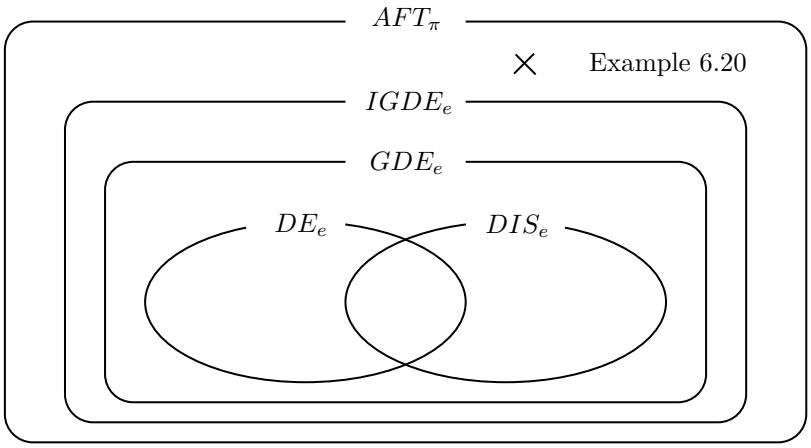

Figure 1: Relation among elimination transformations

When $\pi(h) = [1, 2]$ for each defined AC-symbol h:

The AC-dependency pair method is more useful than the argument filtering transformation. Indeed, the AC-dependency pair method successfully proves AC-termination whenever the argument filtering transformation succeeds. The argument filtering transformation cannot prove the AC-termination of R such that $R \cup DP(R)$ is not AC-terminating (cf. Example 7.3).

One advantage of the AC-dependency pair method is its ability to mark function symbols.

As seen above, if we can efficiently remove the necessity of $\pi(f) = [\,]$ for any defined AC-symbol h, then there exists no advantage for the argument filtering transformation. For example, although the AC-termination of TRSs R_3 (Example 6.20) and R_4 (Example 7.1) can easily be proved by the argument filtering transformation with $\pi(h) = [\,]$, we can also prove it by the AC-dependency pair method to remove the necessity of $\pi(h) = [\,]$ using suitable advanced methods. Actually, the AC-termination of R_3 and R_4 can be proved by AProVE and CiME. However, the AC-termination of R_6, which is more a complicated system, could not be proved by AProVE and CiME. It is future work to investigate in what cases we can remove the necessity of $\pi(h) = [\,]$ for each defined AC-symbol h, and what costs are required to remove it.

Others: Recent years have seen vigorous research on techniques for automated (AC-)termination proofs. As improvements of the dependency pair method other than dependency graphs and CE-termination for hierarchical unions, we have lexicographic argument filtering [21], modularity [15, 31], transforming dependency pairs [16], and so on. One area of future work will include comparing these improvements with the argument filtering transformation.

References

[1] T.Arts, Automatically Proving Termination and Innermost Normalization of Term Rewriting Systems, Ph.D. thesis, Utrecht University, 1997.

[2] T.Arts, J.Giesl, Automatically Proving Termination where Simplification Orderings Fail, In *Proc. of 7th Int. Joint Conf. on Theory and Practice of Software Development*, LNCS 1214 (TAPSOFT'97), pp.261–272, 1997.

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

[4] F.Baader, T.Nipkow, Term Rewriting and All That, Cambridge University Press, 1998.

[5] T.Bachmair, D.A.Plaisted, Termination Orderings for Associative-Commutative Rewriting Systems, *J.Symbolic Computation*, Vol. 1, pp.329–349, 1985.

[6] E.Contejean, C.Marché, B.Monate and X.Urbain, CiME version 2, 2000. (Available at http://cime.lri.fr/)

[7] N.Dershowitz, Orderings for Term-rewriting Systems, *Theoretical Computer Science*, Vol. 17, pp.279–301, 1982.

[8] M.Ferreira, H.Zantema, Dummy Elimination: Making Termination Easier, In *Proc. of 10th Int. Conf. on Fundamentals of Computation Theory*, LNCS 965 (FCT'95), pp.243–252, 1995.

[9] M.Ferreira, Termination of Term Rewriting, Well-foundedness, Totality and Transformations, Ph.D. thesis, Utrecht University, 1995.

[10] M.Ferreira, D.Kesner, L.Puel, Reducing AC-Termination to Termination, In *Proc. of 23rd Int. Symp. on Mathematical Foundations of Computer Science*, LNCS 1450 (MFCS'98), pp.239–247, 1998.

[11] J.Giesl, E.Ohlebusch, Pushing the Frontiers of Combining Rewrite Systems Farther Outwards, In *Proc. of 2nd Int. Workshop on Frontiers of Combining Systems* (FroCos '98), Amsterdam, The Netherlands, Studies in Logic and Computation 7, pp.141–160, Research Studies Press, John Wiley & Sons (2000).

[12] J.Giesl, A.Middeldorp, Eliminating Dummy Elimination, In *Proc. of 17th Int. Conf. on Automated Deduction*, LNAI 1831 (CADE2000), pp.309–323, 2000.

[13] J.Giesl, D.Kapur, Dependency Pairs for Equational Rewriting, In *Proc. 12th Int. Conf. on Rewriting Techniques and Applications*, LNCS 2051 (RTA'01), pp.93–107, 2001.

[14] J.Giesl, T.Arts, Verification of Erlang Processes by Dependency Pairs, *Applicable Algebra in Engineering, Communication and Computing*, 12(1,2):pp.39–72, 2001.

[15] J.Giesl, T.Arts, E.Ohlebusch, Modular Termination Proofs for Rewriting Using Dependency Pairs, *Journal of Symbolic Computation*, 34(1):pp.21–58, 2002.

[16] J.Giesl, R.Thiemann, P.Schneider-Kamp, S.Falke, Improving Dependency Pairs, In *Proc. of 10th Int. Conf. on Logic for Programming Artificial Intelligence and Reasoning*, LNAI 2850 (LPAR03), pp.165–179, 2003.

[17] J.Giesl, R.Thiemann, P.Schneider-Kamp, S.Falke, Automated Termination Proofs with AProVE, In *Proc. of 15th Int. Conf. on Rewriting Techniques and Applications*, LNCS 3091 (RTA04), pp.210–220, 2004.

[18] N.Hirokawa, A.Middeldorp Automating the Dependency Pair Method, *Information and Computation* 199(1,2), pp.172–199, 2005.

[19] J.W.Klop, Term Rewriting Systems, *Handbook of Logic in Computer Science* II, pp.1–112, Oxford University Press, 1992.

[20] M.Kurihara, A.Ohuchi, Modularity of Simple Termination of Term Rewriting Systems with Shared Constructors, *Theoretical Computer Science*, vol.103, pp.273–282, 1992.

[21] K.Kusakari, Y.Toyama, On Proving AC-Termination by Argument Filtering Method, *IPSJ Transactions on Programming* Vol.41, No.SIG 4 (PRO 7), pp.65–78, 2000.

[22] K.Kusakari, Y.Toyama, On Proving AC-Termination by AC-Dependency Pairs, *IEICE Transactions on Information and Systems*, Vol.E84-D, No.5, pp.604–612, 2001.

[23] C.Marché, X.Urbain, Termination of Associative-Commutative Rewriting by Dependency Pairs, In *Proc. of 9th Int. Conf. on Rewriting Techniques and Applications*, LNCS 1379 (RTA'98), pp.241–255, 1998.

[24] C.Marché, X.Urbain, Modular and Incremental Proofs of AC-Termination, *Journal of Symbolic Computation* 38(1),pp.873–897, 2004.

[25] A.Middeldorp, H.Ohsaki, H.Zantema, Transforming Termination by Self-Labeling, In *Proc. of 13th Int. Conf. on Automated Deduction*, LNCS 1104 (CADE-13), pp.373–387, 1996.

[26] A.Middeldorp, Approximating Dependency Graphs using Tree Automata Techniques, In *Proc. of the Int. Joint Conf. on Automated Reasoning*, LNAI 2083 (IJCAR01), pp.593–610, 2001.

[27] M.Nakamura, K.Kusakari, Y.Toyama, On Proving Termination by General Dummy Elimination, *IEICE Transactions on Information and Systems*, vol. J82-D-I, No.10, pp.1225–1231, 1999. (in Japanese)

[28] E.Ohlebusch, Advanced Topics in Term Rewriting, Springer-Verlag, 2002.

[29] H.Ohsaki, A.Middeldorp, J.Giesl, Equational Termination by Semantic Labeling, In *Proc. of 14th Annual Conf. of the European Association for Computer Science Logic*, LNCS 1862 (CSL'00), pp.457–471, 2000.

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

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

[32] X.Urbain, Automated Incremental Termination Proofs for Hierarchically Defined Term Rewriting Systems, In *Proc. of 10th Int. Joint Conf. on Automated Reasoning*, LNAI 2083 (IJCAR'01), pp.485–498, 2001.

[33] X.Urbain, Approche Incrémentale des Preuves Automatiques de Termination, Ph.D. Thesis, Université Paris-Sud, 2002.

[34] X.Urbain, Modular & Incremental Automated Termination Proofs, *Journal of Automated Reasoning* 32(4), pp.315–355, 2004.

[35] H.Zantema, Termination of Term Rewriting: Interpretation and Type Elimination, *Journal of Symbolic Computation* 17, pp.23–50, 1994.