Article Submitted to Journal of Symbolic Computation

Modular and Incremental Proofs of AC-Termination

Claude Marché[1] and Xavier Urbain[1]

[1]*PCRI — LRI, CNRS UMR 8623 — LogiCal Project, INRIA Futurs*
Bât. 490, Université Paris-Sud, Centre d'Orsay
91405 Orsay Cedex, France
E-mail: {marche,urbain}@lri.fr

Abstract

Termination is a non-modular property of rewriting systems, thus it is a difficult task to discover termination proofs for rewriting systems of a large number of rules. Recently, new modular and incremental termination criteria, suitable for automation, were proposed, using an approach based on notions of *termination under non-deterministic collapse* and *dependency pairs*, which apply to *hierarchical combinations* of rewriting systems. We extend this approach and corresponding results to the important case of rewriting modulo associativity and commutativity.

1. Introduction

In the context of algebraic specifications and theorem proving—among other applications—term rewriting systems (TRS) are widely used. In such applications, one usually seeks termination of the considered TRS. However, proving (with any level of automation) termination of a term rewriting system still remains a difficult task, especially when the system has many rules. This problem is due to the fact that termination is not a modular property, in the sense that the union of terminating TRSs does not necessarily terminate, even when assuming strong properties, such as not sharing any symbol as shown by the famous Toyama's counter-example [43]. In other words, a divide and conquer strategy cannot be applied directly.

To obtain positive modularity results, two kinds of restrictions have been considered in the literature. The first one focuses on the class of allowed rules, by requiring syntactic conditions (left-linearity, non-collapsing rules, constructor-based left-hand sides, etc.) or semantic conditions (such as local confluence) [13,

This work is partially supported by the "ATIP-STIC *CiME* du CNRS"

15, 25, 35]. These are very strong restrictions in practice. Hence, so as to deal with TRSs coming from practical applications, we do not consider this kind of approach here.

The second kind of restriction considers stronger notions of termination. The starting point of this approach is the result of Kurihara and Ohuchi [26] in 1990, which states that *simple termination* (see Def. 2.1) is modular for disjoint unions of TRSs. This has been extended to constructor-sharing unions [19, 27] (indeed for finitely branching systems only, which is not a restriction in practice) and more generally to union of composable systems [24, 37]. This last notion of unions appears naturally in TRSs defined hierarchically, thus it is well suited for practice.

Nevertheless, simple termination is a quite strong notion of termination, and many systems occurring in practice are not simply terminating. In 1997, the dependency pairs technique proposed by Arts and Giesl [1, 2] provided a powerful way to discover termination proofs automatically, even for non-simply terminating systems.

Another notion of termination, less restrictive than simple termination, has been introduced by Gramlich in 1991 [19]: *termination under non-deterministic collapse*, also called *Collapse-Extended termination* by Ohlebusch [36] and abbreviated as CE-termination. As for simple termination, CE-termination has been shown to be a modular property for disjoint unions [19, 37], for constructor-sharing unions of finitely branching systems [19], and for union of composable finitely branching systems [28].

In 2001, Urbain [44, 45, 46] put together the notions of dependency pairs and CE-termination, and was able to provide, firstly a new simple proof of the modularity of CE-termination for union of composable systems, and secondly a method for proving termination of hierarchical TRSs in an *incremental* fashion, this method being moreover well-suited for automation. The ability to prove termination incrementally is very important in contexts where rewrite rules are used to develop algebraic specifications structured into libraries and modules, for instance in OBJ [18] or Maude [7] environments.

However, in practice, it is common to use operations with an associative and commutative (AC for short) behaviour. But a direct expression of AC by means of rules (for instance, $x + y \to y + x$ when $+$ is AC) is unpracticable since these rules do not terminate. In order to preserve the termination property, rewriting over AC equivalence classes, namely rewriting *modulo AC*, is used.

Rewriting modulo AC complicates termination proofs significantly. In particular, the orders involved in proofs have to be *AC-compatible*. For years, the problem of finding suitable orderings has been widely studied [4, 5, 10, 22, 23, 41] but all propositions lead to *simplification orderings*, which are too restrictive in practice.

Yet, as far as we know, there exist no modularity results on AC-termination of unions of AC-terminating TRSs. Indeed, the proof of modularity of CE-termination for composable systems in [28] is quite involved, and extending it to rewriting modulo AC is probably a very hard task. On the other hand, the simple proof given by Urbain seems much easier to extend. The first step to achieve this is to consider dependency pairs techniques for AC-rewriting. Indeed, several recent approaches extend the dependency pairs approach to the AC-case [29, 31, 34], which enhances feasibility of automating termination proofs of AC-rewriting. Giesl and Kapur [17], proposed an extension of the dependency pairs technique to rewriting modulo an arbitrary theory (nevertheless satisfying some properties), and the specialisation of their work to AC provides again another variant of the AC-dependency pairs notion.

So, in this paper, we propose a combination of AC-dependency pairs techniques, CE-termination, and hierarchical combinations so as to obtain, firstly, new modularity results on CE-termination modulo AC, and secondly, powerful and incremental methods for proving AC-termination of TRSs of large size, suitable for automation.

This paper is organised as follows. In Section 2, we give our notations and some preliminaries, and recall known results: firstly on modular termination for standard TRSs, secondly on dependency pairs criteria for termination modulo AC. In the latter part, we introduce the notion of abstract AC-dependency pairs, in order to cover all variants of AC-dependency pairs known in the literature [17, 29, 31, 34]. Section 3 is devoted to the proofs of our main modularity results. Theorem 3.3 is the main theoretical result: CE-termination modulo AC is modular for unions of finitely-branching, composable systems. Corollaries 3.2 and 3.3 provide a new method for proving AC-termination in an incremental way, suitable for automation.

Our results are implemented in the termination tool of the CiME system [8] and two complete examples are studied in Section 4. The provided termination proofs were found by CiME in a completely automated way.

2. Notations & preliminaries

We suppose that the reader is familiar with the basic notions of rewriting and termination, especially with the dependency pairs approach; we refer to surveys [3, 14] for details and to Arts & Giesl [2] regarding dependency pairs. We give our notations hereafter.

The root position in a term is denoted by Λ, the symbol at root position in a term t is denoted by $\Lambda(t)$, $t|_p$ denotes the subterm of t at position p. We use postfix notation for substitution applications.

A *term rewriting system* (TRS for short) over a signature $\mathcal{F}$ and a set of variables X is a set $R(\mathcal{F}, X)$ (we shall write $R(\mathcal{F})$ if there is no ambiguity on variables, R if there is no ambiguity on variables and signature) of *rewrite rules* written $l \to r$. Signature $\mathcal{F}$ may be split in two disjoint sets called respectively the set of *defined symbols* and the set of *constructors* of R where defined symbols are the ones which occur at least once as root symbol of a left-hand side of some rule and constructors are the others.

A TRS R defines a rewrite relation $\to_R$ the following way: $s \to_R t$ if there is a position p such that $s|_p = l\sigma$ and $t = s[r\sigma]_p$ for a rule $l \to r \in R$ and a substitution σ. We say then that $s|_p$ is a *redex* and that *s reduces to t at position p with $l \to r$* or, if the rule of R is not relevant, *with R*, respectively denoted $s \xrightarrow[l\to r]{p} t$ and $s \xrightarrow[R]{p} t$.

A TRS R is said to be *finitely branching* if all terms have finitely many reducts by $\xrightarrow[R]{}$. Note that finite TRSs are finitely branching.

A term is *strongly normalisable* (SN) for R if it cannot reduce infinitely by $\xrightarrow[R]{}$. A substitution σ is SN whenever $x\sigma$ is SN for each variable x.

2.1. Termination notions and modularity results

Definition 2.1: *A TRS $R(\mathcal{F})$ is* terminating *if any term is SN.*

It is simply terminating *if the union of $\xrightarrow[R]{}$ and the subterm relation is terminating, or equivalently, if $R \cup \Pi_{\mathcal{F}}$ is terminating, where $\Pi_{\mathcal{F}}$ is the set of all projections rules*

$$f(x_1, \ldots, x_{i-1}, x_i, x_{i+1}, \ldots, x_n) \to x_i$$

for each symbol f of signature $\mathcal{F}$, of arity n, and each i between 1 and n.

It is CE-terminating *[19, 36] if $R \cup \Pi$ is terminating where*

$$\Pi = \left\{ \begin{array}{ll} (\pi_1) & \pi(x, y) \to x \\ (\pi_2) & \pi(x, y) \to y \end{array} \right.$$

for a fresh symbol π.

Simple termination implies CE-termination, which implies termination, but the converses do not hold. In practice, it is common to use terminating TRSs that are not simply terminating [1], however it is very uncommon to use terminating TRSs which are not CE-terminating. This is why we focus on CE-termination and do not consider this restriction as a limitation.

Definition 2.2: *Let $R_1(\mathcal{F}_1)$ and $R_2(\mathcal{F}_2)$ be two TRSs. R_1 and R_2 are said to be* disjoint *if $\mathcal{F}_1 \cap \mathcal{F}_2 = \emptyset$, their union is then called a* disjoint union. *They are said to be* constructor-sharing *if each symbol of $\mathcal{F}_1 \cap \mathcal{F}_2$ is a constructor for both systems, their union is then called a* constructor-sharing union. *R_1 and R_2 are said to be* composable *if for each symbol f of $\mathcal{F}_1 \cap \mathcal{F}_2$, rules whose head symbol is f in R_1 and R_2 are identical. Note that disjoint unions are particular cases of constructor-sharing unions, which themselves are particular unions of composable systems.*

A property P is said to be modular *for a given kind of union if whenever R_1 and R_2 satisfy P, their union satisfies P.*

CE-termination behaves nicely with reference to unions of TRSs, as shown by the following proposition.

PROPOSITION 2.1: *CE-termination is a modular property for unions of*

- *disjoint TRSs (Gramlich, Ohlebusch) [19, 36],*
- *finitely branching, constructor sharing TRSs (Gramlich) [19],*
- *finitely branching, composable TRSs (Kurihara & Ohuchi) [28].*

CE-termination is not modular for constructor sharing unions with non-finitely branching TRSs, as shown by Ohlebusch [36]. Hence we restrict our study to the finitely branching case.

In 2001, Urbain [44, 45, 46] reformulated these results using the notion of rewriting modules, a notion which corresponds naturally to the notion of modules in algebraic specifications systems.

DEFINITION 2.3: *Let R_0 be a term rewriting system over a signature $\mathcal{F}_0$. A* module *extending $R_0(\mathcal{F}_0)$ is a pair $[\mathcal{F}_1 \mid R_1]$ such that:*

1. *$\mathcal{F}_0 \cap \mathcal{F}_1 = \emptyset$ (signatures are disjoint);*
2. *R_1 is a term rewriting system over $\mathcal{F}_0 \cup \mathcal{F}_1$;*
3. *for all $l \to r \in R_1$, $\Lambda(l) \in \mathcal{F}_1$.*

$R_0 \cup R_1$ is a TRS over $\mathcal{F}_0 \cup \mathcal{F}_1$ called the hierarchical extension *of $R_0(\mathcal{F}_0)$* with module *$[\mathcal{F}_1 \mid R_1]$. We write such an extension:*

$$R_0(\mathcal{F}_0) \longleftarrow [\mathcal{F}_1 \mid R_1]$$

We say that another module $[\mathcal{F}_2 \mid R_2]$ extends $R_0(\mathcal{F}_0)$ independently *of module $[\mathcal{F}_1 \mid R_1]$ if $\mathcal{F}_1 \cap \mathcal{F}_2 = \emptyset$, $R_0(\mathcal{F}_0) \longleftarrow [\mathcal{F}_1 \mid R_1]$ and $R_0(\mathcal{F}_0) \longleftarrow [\mathcal{F}_2 \mid R_2]$.*

The only condition we put on extensions is that new rules must have a new symbol at the root position of their left-hand side. Thus, module extensions subsume notions of heavily constrained hierarchical extensions such as *constructor based extensions* [13] and *proper extensions* [24, 25].

There is a strong relationship between hierarchical extensions and the notion of composable systems, as shown by the following straightforward proposition.

PROPOSITION 2.2: *Two TRSs $R_1(\mathcal{F}_1)$ and $R_2(\mathcal{F}_2)$ are composable if and only if $R'_1 = \{l \to r \in R_1 \mid \Lambda(l) \in \mathcal{F}_1 - \mathcal{F}_2\}$ and $R'_2 = \{l \to r \in R_2 \mid \Lambda(l) \in \mathcal{F}_2 - \mathcal{F}_1\}$ extend $R_0 = \{l \to r \in R_1 \cup R_2 \mid \Lambda(l) \in \mathcal{F}_1 \cap \mathcal{F}_2\}$ (therefore necessarily independently).*

A TRS may be built by successive module extensions, resulting in a modules hierarchy. But hierarchies describing a given TRS are not unique; they can be refined until a hierarchy of *minimal modules* is reached: a TRS can actually be automatically decomposed into a unique hierarchy of modules which cannot be split up themselves into a hierarchy of non-empty modules [46].

For termination purposes, considering minimal modules amounts to dealing with fewer constraints, hence to making the proof discovery easier. However, it is not always necessary, and certain modules hierarchies can exploited as they are given, such as libraries in algebraic specification systems.

Proving a property P (such as termination) for a given TRS R *incrementally*, consists in proving property P step by step for each module of the hierarchy defined by R, using an *incrementality property* saying that whenever $R_0(\mathcal{F}_0) \longleftarrow [\mathcal{F}_1 \mid R_1]$, R_0 satisfies P, and (possibly) some additional conditions, then $R_0 \cup R_1$ satisfies P. Corollary 1 in [46] and Corollary 3.1 in this paper are examples of such a theorem.

Even more useful in practice are results combining both modularity and incrementality: for the case of two modules R_1 and R_2 extending R_0 independently, such a result allows to prove P for the union $R_0 \cup R_1 \cup R_2$ knowing P for $R_0 \cup R_1$ and additional conditions over R_2 and R_0 *but not* over R_1. Theorem 2 in [46] and Theorem 3.2 in this paper are examples of such a result. From this kind of result, one may derive both modularity results such as Theorem 3.3, and useful criteria for automation such as Corollary 3.3. Ohlebusch [38, 39] also gave such kind of results for hierarchical termination of (non-AC) TRSs.

2.2. Reduction orderings

Termination is usually proven with the help of *reduction orderings* [12] or quasi-orderings with dependency pairs. We briefly recall what we need. An *ordering pair* (or a *reduction pair* [30]) is a pair $(\succsim, \succ)$ of relations over $T(\mathcal{F}, X)$ such that: 1) $\succsim$ is a quasi-ordering, i.e. reflexive and transitive; 2) $\succ \subseteq \succsim - \preceq$ is a strict ordering, i.e. irreflexive and transitive. Our conditions fulfil the requirements of the very general definition of Kusakari & al [30]. In particular, 1) and 2) imply $\succ \cdot \succsim \subseteq \succ$ and $\succsim \cdot \succ \subseteq \succ$.

With reference to the use we make of them—for all comparisons on terms—we will refer to these pairs as *term orderings*.

A term ordering is said to be *well-founded* if there is no infinite strictly decreasing sequence $t_1 \succ t_2 \succ \cdots$; *stable* if both $\succ$ and $\succsim$ are stable under substitutions; *weakly* (resp. *strictly*) *monotonic* if for all terms t_1 and t_2, for all $f \in \mathcal{F}$, if $t_1 \succsim$ (resp. $\succ$) t_2 then $f(\ldots, t_1, \ldots) \succsim$ (resp. $\succ$) $f(\ldots, t_2, \ldots)$. A term ordering $(\succsim, \succ)$ is called a *weak (resp. strict) reduction ordering* if it is well-founded, stable and weakly (resp. strictly) monotonic; a *simplification ordering* if it is stable, monotonic and has the subterm property w.r.t. $\succsim$.

A term ordering $(\succsim, \succ)$ over $T(\mathcal{F}, X)$ is said to be π-*expandable* [44, 45] if there is a reduction ordering $(\succsim', \succ')$ over $T(\mathcal{F} \cup \{\pi\}, X)$ such that:

- $(\succsim', \succ')$ restricted to $T(\mathcal{F}, X)$ is exactly $(\succsim, \succ)$;
- $\pi(x, y) \succsim' x$ and $\pi(x, y) \succsim' y$ for arbitrary variables x and y.

Any simplification ordering is π-expandable [44, 45]. In particular, orderings commonly used in automated proofs—such as RPO and orderings induced by polynomial interpretations—are π-expandable. Lexicographical compositions of

π-expandable orderings as well as their use with *Recursive Program Schemes* [2] lead to π-expandable orderings [44, 45].

π-expandable ordering may be used for proving (CE-) termination, as shown by the following proposition.

PROPOSITION 2.3: *A term rewriting system R is terminating if and only if there is strictly monotonic reduction ordering such that $R \subseteq \succ$ (i.e for each rule $l \to r$ of R, $l \succ r$).*

R is simply terminating if and only if there is strictly monotonic simplification ordering such that $R \subseteq \succ$.

R is CE-terminating if and only if there is strictly monotonic π-expandable ordering such that $R \subseteq \succ$.

2.3. Termination of rewriting modulo AC

An *AC-signature* is a signature $\mathcal{F}$ where the set of binary symbols has two distinguished disjoint subsets $\mathcal{F}_{\mathrm{AC}}$ and $\mathcal{F}_{\mathrm{C}}$ which are the set of *Associative-Commutative* symbols and *Commutative* symbols respectively. The symbols that are neither in $\mathcal{F}_{\mathrm{AC}}$ nor in $\mathcal{F}_{\mathrm{C}}$ are called *free*.

EXAMPLE 2.1: *An AC-signature of natural numbers in Peano notation is given by $\mathcal{F} = \{0, true, false, s, +, \times, =, \geq\}$, $\mathcal{F}_{AC} = \{+, \times\}$, and $\mathcal{F}_C = \{=\}$.*

We call *equality modulo AC*, and denote $=_{\mathrm{AC}}$, the congruence on $T(\mathcal{F}, X)$ generated by commutativity axioms $f(x,y) = f(y,x)$ for each $f \in \mathcal{F}_{\mathrm{C}} \cup \mathcal{F}_{\mathrm{AC}}$, and associativity axioms $f(f(x,y),z) = f(x, f(y,z))$ for each $f \in \mathcal{F}_{\mathrm{AC}}$.

2.3.1. Varyadic terms

The set of *varyadic* terms is the set of terms over $\mathcal{F}$ and X but where AC-symbols may occur at any arity greater than or equal to 2.

The $=_{\mathrm{AC}}$ congruence generalised to varyadic terms is generated by commutativity axioms $f(x,y) = f(y,x)$ for each $f \in \mathcal{F}_{\mathrm{C}}$; permutation axioms $f(x_1, \ldots, x_n) = f(x_{\rho(1)}, \ldots, x_{\rho(n)})$ for each $f \in \mathcal{F}_{\mathrm{AC}}$, $n \geq 2$, ρ permutation of $[1, \ldots, n]$; and associativity axioms $f(x_1, \ldots, x_{i-1}, f(y_1, \ldots, y_k), x_{i+1}, \ldots, x_n) = f(x_1, \ldots, x_{i-1}, y_1, \ldots, y_k, x_{i+1}, \ldots, x_n)$ for each $f \in \mathcal{F}_{\mathrm{AC}}$, $n \geq 2$, $1 \leq i \leq n$ and $k \geq 2$. It is clear that the restriction of the latter definition of the congruence $=_{\mathrm{AC}}$ to non-varyadic terms coincides with its former definition.

The set of *flat* terms is the subset of varyadic terms in which AC-symbols never occur directly below themselves. In other words, $f(t_1, \ldots, t_n)$ is flat if $t_1, \ldots, t_n$ are flat and either $f \notin \mathcal{F}_{\mathrm{AC}}$ or $\Lambda(t_i) \neq f$ for all i. The set of *flattening rules* consists of the following rewrite rules

$$f(x_1, \ldots, x_{i-1}, f(y_1, \ldots, y_k), x_{i+1}, \ldots, x_n) \to \\ f(x_1, \ldots, x_{i-1}, y_1, \ldots, y_k, x_{i+1}, \ldots, x_n)$$

for each AC-symbol f, $n \geq 2$, $1 \leq i \leq n$ and $k \geq 2$. For each varyadic term t, t has a unique normal form with reference to flattening rules, which is a flat term, the *flattened form* of t, denoted $\overline{t}$.

We denote by $\equiv$ the equivalence on flat terms up to permutations of arguments of C or AC-symbols. It is clear that for any varyadic terms s and t, $s =_{\mathrm{AC}} t$ if and only if $\overline{s} \equiv \overline{t}$.

Example 2.2 (continued): *$(s(0)+x)+(s(y)+0)$ and $(x+(s(y)+s(0)))+0$ are AC-equal since their flattened forms are $s(0)+x+s(y)+0$ and $x+s(y)+s(0)+0$ which are equivalent up to permutation.*

2.3.2. Rewriting modulo AC

We focus now on definition of rewriting modulo associativity and commutativity. In fact, several slightly different variants exist; a detailed comparison may be found in [34].

Class rewriting is defined by

$$s \xrightarrow[l\to r/\mathrm{AC}]{} t,$$

if there are s' and t' such that $s =_{\mathrm{AC}} s'$, $t =_{\mathrm{AC}} t'$ and $s' \xrightarrow[l\to r]{} t'$.

This definition is algorithmically complex because deciding whether a term s rewrites to another modulo AC amounts to searching for an instance of a rule in any term AC-equivalent to s.

AC-extended rewriting, proposed by Peterson and Stickel [40], is a restricted definition where AC-equivalence is allowed only at or below the position p of a term where one wants to match a rule:

$$s \xrightarrow[\mathrm{AC}\backslash l\to r]{p} t$$

if there is a substitution σ such that $s|_p =_{\mathrm{AC}} l\sigma$ and $t = s[r\sigma]_p$.

Example 2.3 (continued): *The following TRS R*

$$\begin{array}{rcl} x+0 & \to & x \\ s(x)+s(y) & \to & s(s(x+y)) \end{array}$$

may be used to define addition in Peano arithmetics modulo AC. It is completely defined, for example $0+s(0)$ rewrites to $s(0)$ with first rule, since $0+s(0) =_{AC} s(0)+0$. A term like $s(0)+(y+s(0))$ is in normal form for $\xrightarrow[AC\backslash R]{}$ whereas it is reducible to $y+s(s(0))$ by $\xrightarrow[R/AC]{}$, since $s(0)+(y+s(0)) =_{AC} y+(s(0)+s(0))$.

There is a drawback in this restriction: rewriting is not necessarily *coherent* with AC in the sense defined by Jouannaud and Kirchner [21]:

$$s =_{\mathrm{AC}} t \text{ and } t \xrightarrow[\mathrm{AC}\backslash R]{} u$$

should imply that there exist v and w such that

$$s \xrightarrow[\mathrm{AC}\backslash R]{}{}^{\star} v =_{\mathrm{AC}} w \xleftarrow[\mathrm{AC}\backslash R]{}{}^{\star} u.$$

Indeed, there are rewrite systems R for which $\rightarrow_{\mathrm{AC}\backslash R}$ terminates whereas $\rightarrow_{R/\mathrm{AC}}$ does not [34] (for example $a+a \rightarrow (a+b)+a$). One may make any rewrite system R AC-coherent by adding to R the so-called *extended rules*. That is, for each rule $f(s,t) \rightarrow r$ where f is AC, adding the rule $f(f(s,t),x) \rightarrow f(r,x)$ where x is a new variable.

EXAMPLE 2.4 (CONTINUED): *The extended rule of the third rule of R above is $(s(x)+s(y))+z \rightarrow s(s(x+y))+z$. With this new rule, the term $s(0)+(y+s(0))$ which in normal form for $AC\backslash R$ is now reducible.*

Varyadic AC-rewriting is a variant of AC-extended rewriting that, in some sense, builds the extended rules in the rewriting relation, by considering varyadic terms. It is defined by

$$s \xrightarrow[l\rightarrow r/\overline{\mathrm{AC}}]{p} t$$

if there is a substitution σ such that either

- $s|_p =_{\mathrm{AC}} l\sigma$ and $t = s[r\sigma]_p$ or
- $l = f(l_1,\ldots,l_n)$ where f is AC and for a new variable x, $s|_p =_{\mathrm{AC}} f(l_1,\ldots,l_n,x)\sigma$ and $t = s[f(r,x)\sigma]_p/\overline{\mathrm{AC}}$.

EXAMPLE 2.5 (CONTINUED): *With varyadic rewriting, the term $s(0)+y+s(0)$ is reducible by rule $s(x)+s(y) \rightarrow s(s(x+y))$, using the second case above.*

Regarding termination, these notions of rewriting are equivalent, thanks to the following proposition.

PROPOSITION 2.4 ([17, 34]): *For any TRS R, the following statements are equivalent:*

1. *class rewriting with R is terminating;*
2. *varyadic AC-rewriting with R is terminating;*
3. *AC-extended rewriting with R augmented with its extended rules is terminating.*

In this paper, we will always consider that we have TRSs where extended rules have been added if needed, and we will use indifferently varyadic or fixed-arity rewriting. From now on, for the sake of readability, we shall denote rewriting steps by

$$s \xrightarrow[l\rightarrow r/\mathrm{AC}]{p} t$$

2.3.3. AC-termination with dependency pairs

Arts & Giesl first introduced [1] *dependency pairs* of a TRS $R(\mathcal{F})$: those are pairs of terms whose root symbols are what they called *tuple symbols*, instead of symbols from signature $\mathcal{F}$. Several authors [17, 29, 31, 34] have proposed extensions of the dependency pairs criteria to AC-rewriting, with similar but different definitions of AC-dependency pairs. For some of those generalisations, two variants of dependency pairs have been given: one for which root symbols are distinguished (like in the standard case), and one for which root symbols stay in the original signature. Their authors call these variants *marked pairs* and *non-marked pairs* respectively, and they use the special notation $f^\sharp$ or $\widehat{f}$ to denote the so-called *marked copy* of symbol f (which plays exactly the same role as the tuple symbol defined by Arts & Giesl). We shall use notation $\widehat{f}$. The variant without marks has been considered for two reasons. Firstly, dependency pairs criteria without marks are a bit simpler than with marks, because marks require to introduce additional rules so as to handle some constraints existing between f and $\widehat{f}$ when f is AC. In the following definition, these rules are called *marks handling rules*. Secondly, the constraints given by those marks handling rules usually impose that a symbol f and its marked copy $\widehat{f}$ are considered equivalent in the term ordering eventually found. Indeed, in practice, even without AC symbols, the set of constraints to solve for proving termination of a given TRS may be significantly larger when using marks than without, with a larger search space (because the signature size is doubled). Hence the time and space used to solve them can be much larger, as illustrated by the second example in Section 4.

In this paper, we want to design criteria for incremental proofs of termination which will work with any of those AC-dependency pairs notions. To achieve this, we define now an abstract notion of AC-dependency pairs, with some required properties which are satisfied by all of the former AC-dependency pairs notions. In order to handle both the marked and the non-marked cases, we denote by $\widehat{\mathcal{F}}$ the extended set of symbols in use, which can be either equal to $\mathcal{F}$ for the non-marked case, or equal to $\mathcal{F} \cup \{\widehat{f} \mid f \text{ defined}\}$ for the marked case.

Definition 2.4: *An* abstract AC-dependency pairs definition *is a function which, given a TRS R over a signature $\mathcal{F} = D \cup C$ where D are defined symbols of R and C are constructors, gives a set of pairs of terms denoted $\langle u, v\rangle$ on signature $D \cup \widehat{D} \cup C$, and an additional rewrite relation $\rightarrow_{mh}$ called* marks handling*, satisfying:*

i. For any dependency pair $\langle u, v\rangle$, $\Lambda(u)$ and $\Lambda(v)$ are in $\widehat{D}$. Moreover, there is a rewrite rule $l \to r$ in R such that $\Lambda(u) = \widehat{\Lambda(l)}$ (resp. $\Lambda(u) = \Lambda(l)$ for non-marked case) and $\Lambda(v) = \widehat{f}$ (resp. f) for some f occurring in r.

ii. $\rightarrow_{mh}$ is generated by a finite set of rewrite rules R_{mh} such that for each $l \to r$ in R_{mh}, there exists one AC-symbol f such that $\Lambda(l) = \Lambda(r) = \widehat{f}$ and strict subterms of l and r may contain f, $\widehat{f}$ and variables only. Note that *this system is not required to terminate, and is necessarily empty if we do not use marks.*

iii. *R is SN iff there is no infinite sequence of pairs $\langle u_1, v_1\rangle, \langle u_2, v_2\rangle, \ldots$ together with a substitution σ such that for all i:*

$$v_i\sigma(\rightarrow^{\neq\Lambda}_{R/AC} \cup \rightarrow_{mh})^\star u_{i+1}\sigma$$

The superscript $\neq \Lambda$ means that no rewriting (nor AC-step) occurs at root position. Moreover, that sequence may be chosen minimal*, that is for which σ is SN. Such a sequence is called a dependency chain. We assume that variables in those pairs are renamed so that pairs are variable-disjoint, and consequently, σ has an infinite domain.*

In the remaining of the paper we will refer to these conditions as abstract properties (i), (ii) and (iii). It can be easily checked that properties (i) and (ii) are satisfied by existing concrete AC-dependency pairs techniques [17, 29, 31, 34] (property (iii) is the main result of each corresponding papers):

- Kusakari & Toyama [31] consider fixed-arity terms. With their variant without marks, $\rightarrow_{mh}$ is empty; and with marks, $\rightarrow_{mh}$ is generated by $\widehat{f}(\widehat{f}(x,y),z) \leftrightarrow \widehat{f}(f(x,y),z)$ and $\widehat{f}(\widehat{f}(x,y),z) \rightarrow \widehat{f}(x,y)$ for each AC-symbol f.
- Marché & Urbain [34] consider flat terms. In their variant without marks, $\rightarrow_{mh}$ is empty. Their variant with marks appears to be unsound in the sense that (iii) is not satisfied, so we do not consider it. (See [29] for a counter-example.)
- Kusakari, Marché & Urbain [29] consider flat terms and marks, $\rightarrow_{mh}$ is the reduction relation generated by $\widehat{f}(x,y,z) \rightarrow \widehat{f}(f(x,y),z)$ for each AC-symbol f.
- Giesl & Kapur [17] (when specialised to the AC case) consider fixed-arity terms and marks, $\rightarrow_{mh}$ is generated by $\widehat{f}(f(x,y),z) \leftrightarrow \widehat{f}(x,f(y,z))$ and $\widehat{f}(x,y) \leftrightarrow \widehat{f}(y,x)$ for each AC-symbol f.

Note that in our implementation C*i*ME, we use by default a small variant which is none of the above: we consider flat terms, and we use marks for non-AC symbols but no marks for AC symbols. In other words, we have $\widehat{D} = \{\widehat{f} \mid f \in D - \mathcal{F}_{AC}\} \cup \{f \mid f \in D \cap \mathcal{F}_{AC}\}$. Properties (i), (ii) and (iii) above are also satisfied with this variant, with $\rightarrow_{mh}$ being empty. Indeed, as soon as it is chosen not to use marks for AC symbols, then all the variants above become equivalent. This choice seems to be a good compromise, keeping the power of having marked symbols for non AC-symbols and avoiding complexity induced by marked AC-symbols.

2.3.4. AC and marks compatibility

For any of these variants, we have the following criterion [17, 29, 31, 34], well-suited for automation, extending the one for the non-AC case [1, 2].

PROPOSITION 2.5: *If R is an AC-TRS, if $(\succsim, \succ)$ is a weakly monotonic reduction ordering such that:*

- *for any dependency pair $\langle u, v\rangle$ of R, $u \succ v$,*
- *for any rule $l \to r$ of R, $l \succsim r$,*
- *$\succsim$ is AC-compatible with each non-marked AC-symbol,*
- *for any rule $l \to r$ of $\to_{mh}$, $l \succsim r$,*

then R is terminating.

The third and fourth conditions above will be referred as the *AC and marks compatibility* of the ordering, in particular in Section 3.4 where we will show how to automate our incremental criteria. Note that if one does not use marks for AC symbols, then AC and marks compatibility is simply the usual AC-compatibility.

Other criteria exist, usually more powerful, based on the so-called *approximated dependency graphs*, allowing less restrictive constraints than the first two conditions above, but still requiring the last two. We do not need such improvements in this paper, because they are orthogonal to our results, but the implementation we use for examples of Section 4 uses them.

3. Incremental termination proofs for AC-rewriting

We assume now given a notion of AC-dependency pairs satisfying the abstract properties given in Section 2.3.3. All TRSs we consider now are assumed finitely branching.

3.1. AC-dependency pairs of modules

For the AC case, we keep the same definition of modules as for the standard case (Def. 2.3), the important point here is that only one symbol is involved in each AC-step. Hence, there is no permutation of signatures inside terms: AC-steps *introduce no dependency* between modules.

DEFINITION 3.1: *Let us consider a hierarchical extension $R_1(\mathcal{F}_1) \longleftarrow [\mathcal{F}_2 \mid R_2]$.*

The set of AC-dependency pairs of module $[\mathcal{F}_2 \mid R_2]$, *denoted* $\mathrm{MDP}([\mathcal{F}_2 \mid R_2])$, *is the set of AC-dependency pairs of TRS R_2 over signature $\mathcal{F}_1 \cup \mathcal{F}_2$, that is where all symbols from $\mathcal{F}_1$ are considered as constructors.*

Notice that $\mathrm{MDP}([\mathcal{F}_2 \mid R_2])$ is always a subset of the set of standard dependency pairs of $R_1 \cup R_2$, and the pairs of $R_1 \cup R_2$ that are not in $\mathrm{MDP}([\mathcal{F}_2 \mid R_2])$ are pairs that cannot belong to a cycle of the dependency graph of $R_1 \cup R_2$. So indeed, using the notion of dependency pair of a module prevents the consideration of some of the pairs which would have been discarded later if dependency graphs were used, but without even building any graph.

PROPOSITION 3.1: *For each pair* $\langle u, v\rangle$ *of* $\mathrm{MDP}([\mathcal{F}_2 \mid R_2])$*, root symbols of* u *and* v *are in* $\widehat{\mathcal{F}_2}$.

Proof: Since root symbols of R_2 rules lhs are in $\mathcal{F}_2$, defined symbols of R_2 are all in $\mathcal{F}_2$. Hence, from abstract property (i), we know that root symbols of terms in module AC-dependency pairs are in $\widehat{\mathcal{F}_2}$. □

Note that abstract property (i) is crucial for this proposition. If it was not true, AC-dependency pairs would not respect the hierarchical structure of the rules, hence we would have no hope of finding modularity results. In the following, we will not need abstract property (i) anymore, only the proposition above.

DEFINITION 3.2: *A* relative AC-dependency chain of a module $[\mathcal{F} \mid R]$ over a TRS S *(with* $R \subseteq S$*) is a sequence of pairs* $\ldots\langle s_j, t_j\rangle\ldots$ *(*$\langle s_j, t_j\rangle \in \mathrm{MDP}([\mathcal{F} \mid R])$*) together with a substitution* σ *such that for all i:*

$$t_i\sigma(\xrightarrow[S/AC]{\neq\Lambda} \cup \rightarrow_{mh})^{\star} s_{i+1}\sigma$$

holds. This chain is said to be minimal *if* σ *is* S*-strongly normalisable.*

We are now able to extend Urbain's incrementality results [45, 46] to the AC-case. The methodology is the same: we firstly prove a key result, which will be used three times to prove the main theorems.

3.2. Key result for the AC-case

PROPOSITION 3.2: *Let* S_1 *and* S_2 *be AC-TRSs over signature* $\mathcal{F}_1$. *Let* S_3 *be an AC-TRS over* $\mathcal{F}_1 \cup \mathcal{F}_2$ *such that:*

- $\mathcal{F}_1 \cap \mathcal{F}_2 = \emptyset$;
- *for each* $l \rightarrow r \in S_3$, $\Lambda(l) \in \mathcal{F}_2$.

If there is an infinite minimal relative chain of $[\mathcal{F}_1 \mid S_2]$ *over* $S_1 \cup S_2 \cup S_3$*, then there is an infinite relative chain of* $[\mathcal{F}_1 \mid S_2]$ *over* $S_1 \cup S_2 \cup \Pi$ *with the same sequence of pairs but with a new substitution and new rewriting steps.*

To achieve this proof, we introduce an interpretation of varyadic AC-terms, and firstly establish several basic lemmata.

DEFINITION 3.3: *Let us denote* $S = S_1 \cup S_2 \cup S_3$. *Interpretation* $I : T(\widehat{\mathcal{F}_1} \cup \mathcal{F}_2, X) \to T(\widehat{\mathcal{F}_1} \cup \{\pi : 2\} \cup \{\perp : 0\}, X)$ *is defined as follows:*

$$\begin{array}{rcl} I(x) & = & x \text{ if } x \in X, \\ I(f(t_1 \ldots t_n)) & = & \left\{ \begin{array}{l} f(I(t_1) \ldots I(t_n)) \text{ if } f \in \widehat{\mathcal{F}_1}, \\ \mathrm{Comb}(\mathrm{Red}(f(t_1 \ldots t_n))) \text{ if } f \in \mathcal{F}_2, \end{array} \right. \end{array}$$

where

$$\begin{array}{rcl} \mathrm{Red}(t) & = & \{I(t') /\ t \xrightarrow[S_1 \cup S_2 \cup S_3/AC]{} t'\}, \\ \mathrm{Comb}(\emptyset) & = & \perp, \\ \mathrm{Comb}(\{a\} \cup E) & = & \pi(a, \mathrm{Comb}(E)). \end{array}$$

This interpretation is essentially the same as for the standard case, except that we have AC-rewriting in Red. Since rewriting by $S_1 \cup S_2 \cup S_3/\mathrm{AC}$ may be used recursively in that definition, $I(t)$ may not be defined as a finite term for any t. Hence, we establish below a sufficient condition on t for $I(t)$ to be defined.

Given a substitution σ, by $I(\sigma)$ we denote the substitution σ' such that $x\sigma' = I(x\sigma)$ for any variable x.

LEMMA 3.1: *For any term t for which all subterms headed by a symbol in $\mathcal{F}_2$ are strongly normalisable by $S_1 \cup S_2 \cup S_3$ modulo AC, $I(t)$ is well-defined. For all $t \in T(\widehat{\mathcal{F}_1}, X)$ and all substitutions σ strongly normalisable by $S_1 \cup S_2 \cup S_3$ modulo AC, $I(t\sigma) = tI(\sigma)$. For all $t_1, \ldots, t_n$ in $T(\widehat{\mathcal{F}_1} \cup \mathcal{F}_2, X)$ strongly normalisable by $S_1 \cup S_2 \cup S_3$ modulo AC and for each context C over $\mathcal{F}_1$ with n holes, $I(C[t_1, \ldots, t_n]) = C[I(t_1), \ldots, I(t_n)]$.*

Proof of these facts are the same as for the standard case [46], using simple inductions, and without need of our abstract properties. Note that the first fact requires that TRSs are finitely branching.

LEMMA 3.2: *For all s and t of $T(\widehat{\mathcal{F}_1} \cup \mathcal{F}_2, X)$, such that $I(s)$ and $I(t)$ are well-defined:*

$$\textit{If } s \xrightarrow[S_1 \cup S_2/AC]{p} t \textit{ then } I(s) \xrightarrow[S_1 \cup S_2 \cup \Pi/AC]{}^{+} I(t).$$

Moreover, if $p \neq \Lambda$ and $\Lambda(s) \in \widehat{\mathcal{F}_1}$ then $I(s) \xrightarrow[S_1 \cup S_2 \cup \Pi/AC]{\neq \Lambda}^{+} I(t)$.

$$\textit{If } s \xrightarrow[S_3/AC]{p} t \textit{ then } I(s) \xrightarrow[\Pi/AC]{}^{+} I(t).$$

Moreover, if $\Lambda(s) \in \widehat{\mathcal{F}_1}$ then $I(s) \xrightarrow[\Pi/AC]{\neq \Lambda}^{+} I(t)$.

Again, proof of these facts are the same as for the standard case [46], using previous lemma, and without need of our abstract properties.

We are now ready to prove Proposition 3.2. Let $\langle u_1, v_1\rangle, \langle u_2, v_2\rangle, \ldots$ be a minimal AC-dependency chain of $[\mathcal{F}_1 \mid S_2]$ over $S_1 \cup S_2 \cup S_3$ with a substitution σ. Let σ' be the substitution such that for each x, $x\sigma' = I(x\sigma)$.

By minimality of the considered chain, substitution σ is strongly normalisable, hence we know by Lemma 3.1 that terms instantiated by σ' are all finite, and moreover $I(u_i\sigma)$ and $I(v_i\sigma)$ are well-defined for all i, since u_i and v_i only contain symbols from $\hat{\mathcal{F}}_1$.

We show that $\langle u_1, v_1\rangle, \langle u_2, v_2\rangle, \ldots$ with σ' is an AC-chain of $[\mathcal{F}_1 \mid S_2]$ over $S_1 \cup S_2 \cup \Pi$.

In that purpose, we have to prove for each i:

$$v_i\sigma' \left(\xrightarrow[S_1 \cup S_2 \cup \Pi/\mathrm{AC}]{\neq\Lambda} \cup \to_{mh} \right)^\star u_{i+1}\sigma'.$$

We know

$$v_i\sigma \left(\xrightarrow[S_1 \cup S_2 \cup S_3/\mathrm{AC}]{\neq\Lambda} \cup \to_{mh} \right)^\star u_{i+1}\sigma.$$

Let us consider a step $s \xrightarrow[S_1 \cup S_2 \cup S_3/\mathrm{AC}]{p} t$ from that reduction. Since

$$\Lambda(s) = \Lambda(t) = \Lambda(v_i) = \Lambda(u_{i+1}) \in \widehat{\mathcal{F}_1},$$

by one of the two cases of Lemma 3.2 we get

$$I(s) \xrightarrow[S_1 \cup S_2 \cup \Pi/\mathrm{AC}]{\neq\Lambda}{}^\star I(t).$$

Finally, let us consider a step $s \to_{mh} t$ from that reduction. Since $\to_{mh}$ reduction only applies to terms headed by a marked symbol, and involves only symbols from $\mathcal{F}_1 \cup \widehat{\mathcal{F}_1}$ (abstract property (ii)), such a reduction is necessarily completely in the top part of s, hence we conclude by Lemma 3.1 that $I(s) \to_{mh} I(t)$.

Now, we may actually build the desired sequence by juxtaposition of steps so as to obtain

$$I(v_i\sigma)(\xrightarrow[S_1 \cup S_2 \cup \Pi/\mathrm{AC}]{\neq\Lambda} \cup \to_{mh})^\star I(u_{i+1}\sigma).$$

Since $I(v_i\sigma) = v_i\sigma'$ and $I(u_{i+1}\sigma) = u_{i+1}\sigma'$ we conclude using Lemma 3.1.

This ends the proof of Proposition 3.2.

3.3. Main modularity results

With the help of Proposition 3.2 we may prove the following results regarding (respectively) extensions with one module and extension with several independent modules.

Theorem 3.1: *Let* $[\mathcal{F}_1 \mid R_1] \longleftarrow [\mathcal{F}_2 \mid R_2]$ *be a hierarchical extension of* $R_1(\mathcal{F}_1)$ *where* $\mathcal{F}_1 \cup \mathcal{F}_2$ *contains AC-symbols. If*

1. R_1 *is CE-terminating modulo AC, and*
2. *there is no infinite relative AC-chain of* $[\mathcal{F}_2 \mid R_2]$ *over* $R_1 \cup R_2$,

then $R_1 \cup R_2$ *is terminating modulo AC.*

Proof: By contradiction. If $R_1 \cup R_2$ is not terminating, then by abstract property (iii) there is an infinite (non relative) AC-dependency chain of $R_1 \cup R_2$. We are going to show that:

- either there is an infinite relative AC-dependency chain of $[\mathcal{F}_2 \mid R_2]$ over $R_1 \cup R_2$, thus contradicting the second premise,
- or R_1 is not CE-terminating modulo AC, thus contradicting the first premise.

For that purpose, we use an analysis of which pairs may follow another, as for the dependency graph analysis: if $\langle u, v\rangle$ is a non-modular pair of $R_1 \cup R_2$, then either:

① $\Lambda(u) \in \widehat{\mathcal{F}_1}$, hence $\Lambda(v) \in \widehat{\mathcal{F}_1}$ from the AC-modules structure, so $\langle u, v\rangle$ is a modular pair of $[\mathcal{F}_1 \mid R_1]$;

② $\Lambda(u)$ and $\Lambda(v) \in \widehat{\mathcal{F}_2}$, hence $\langle u, v\rangle$ is a modular pair of $[\mathcal{F}_2 \mid R_2]$;

③ $\Lambda(u) \in \mathcal{F}_2$ and $\Lambda(v) \in \widehat{\mathcal{F}_1}$.

For any consecutive pairs $\langle u_1, v_1\rangle$, $\langle u_2, v_2\rangle$ in an AC-dependency chain, we have a substitution σ such that:

$$v_1\sigma(\xrightarrow[R_1 \cup R_2/\mathrm{AC}]{\neq\Lambda} \cup \rightarrow_{mh})^\star u_2\sigma.$$

Thanks to abstract property (ii), $\rightarrow_{mh}$ preserves the root symbol, so we know that $\Lambda(v_1) = \Lambda(u_2)$. Thus, we know that in an AC-dependency chain, pairs ② and ③ may follow pairs ② only. Similarly, pairs ① may follow pairs ① or ③ only.

Hence we may encounter three cases: the AC-dependency chain we consider consists of:

1. pairs ② only, that is pairs of module $[\mathcal{F}_2 \mid R_2]$ or
2. pairs ① only, that is pairs of module $[\mathcal{F}_1 \mid R_1]$ or
3. pairs ② in finite number (possibly zero) followed by only one pair ③ *then* by an infinite number of pairs ①.

- First case: An infinite relative AC-dependency chain of pairs of $[\mathcal{F}_2 \mid R_2]$ over $R_1 \cup R_2$ contradicts the second premise.

- Second & third cases: An infinite relative AC-chain of $[\mathcal{F}_1 \mid R_1]$ over $R_1 \cup R_2$ occurs in both cases. Applying Prop. 3.2 with $R_1 = S_1 = S_2$ and $R_2 = S_3$, from any infinite AC-dependency chain of $[\mathcal{F}_1 \mid R_1]$ over $R_1 \cup R_2$, we can build a corresponding (infinite) AC-chain of $[\mathcal{F}_1 \mid R_1]$ over $R_1 \cup \Pi$ which is an an infinite AC-chain of $[\mathcal{F}_1 \cup \{\pi : 2\} \mid R_1 \cup \Pi]$ over $R_1 \cup \Pi$. That chain is an infinite AC-dependency chain of $R_1 \cup \Pi$, thus contradicting hypothesis 'R_1 CE-terminating modulo AC'.

This ends the proof of Theorem 3.1. □

THEOREM 3.2: *Let $[\mathcal{F}_1 \mid R_1] \longleftarrow [\mathcal{F}_2 \mid R_2]$ be a hierarchical extension of $R_1(\mathcal{F}_1)$, let $[\mathcal{F}_3 \mid R_3]$ be a module extending R_1 independently of R_2. If*

1. *$R_1 \cup R_2$ is CE-terminating modulo AC, and*
2. *there is no infinite relative AC-chain of $[\mathcal{F}_3 \mid R_3]$ over $R_1 \cup R_3 \cup \Pi$,*

then $R_1 \cup R_2 \cup R_3$ is CE-terminating modulo AC.

Proof: Again by contradiction, let us suppose that $R_1 \cup R_2 \cup R_3$ is not CE-terminating modulo AC, that is $R = R_1 \cup R_2 \cup R_3 \cup \Pi$ is not terminating modulo AC. From abstract property (iii), we know there is an infinite AC-dependency chain for R. We are going to show that in such a case, we may conclude either on non-CE-termination of $R_1 \cup R_2$, thus contradicting the first premise, or on the existence of an infinite relative AC-dependency chain of module $[\mathcal{F}_3 \mid R_3]$ over $R_1 \cup R_3 \cup \Pi$, a contradiction to the second premise.

From the hierarchical structure of our three modules and abstract property (ii), as in the proof of Theorem 3.1, we know that AC-dependency chains of $R_1 \cup R_2 \cup R_3 \cup \Pi$ are either:

- AC-chains of $[\mathcal{F}_3 \mid R_3]$ over $R_1 \cup R_2 \cup R_3 \cup \Pi$;
- AC-chains of $[\mathcal{F}_1 \cup \mathcal{F}_2 \mid R_1 \cup R_2]$ over $R_1 \cup R_2 \cup R_3 \cup \Pi$;
- AC-chains consisting of a *finite number* of pairs of module $[\mathcal{F}_3 \mid R_3]$, followed by *only one* pair $\langle \widehat{s}, \widehat{t} \rangle$ such that $\Lambda(s) \in \mathcal{F}_3$ and $\Lambda(t) \in \mathcal{F}_1 \cup \mathcal{F}_2$, *then* by an AC-chain of $[\mathcal{F}_1 \cup \mathcal{F}_2 \mid R_1 \cup R_2]$ over $R_1 \cup R_2 \cup R_3 \cup \Pi$.

Thus, it suffices to prove finiteness of relative AC-chains of $[\mathcal{F}_3 \mid R_3]$ over R and of $[\mathcal{F}_1 \cup \mathcal{F}_2 \mid R_1 \cup R_2]$ over R.

- There is no infinite AC-chain of $[\mathcal{F}_3 \mid R_3]$ over R. Otherwise, by Prop. 3.2 with $S_1 = R_1$, $S_2 = R_3 \cup \Pi$, $S_3 = R_2$, $\mathcal{F}_1 = \mathcal{F}_1 \cup \mathcal{F}_3$ and $\mathcal{F}_2 = \mathcal{F}_2$, we would end with an infinite AC-chain of $[\mathcal{F}_3 \mid R_3]$ over $R_1 \cup R_3 \cup \Pi$. But all those are finite from the premises.
- There is no infinite AC-chain of $[\mathcal{F}_1 \cup \mathcal{F}_2 \mid R_1 \cup R_2]$ over R. Otherwise, applying Prop. 3.2 with $S_1 = \emptyset$, $S_2 = R_1 \cup R_2$ and $S_3 = R_3 \cup \Pi$, we would end with an infinite AC-chain of $[\mathcal{F}_1 \cup \mathcal{F}_2 \mid R_1 \cup R_2]$ over $R_1 \cup R_2 \cup \Pi$. But these chains are all finite since $R_1 \cup R_2$ CE-terminates modulo AC from premises.

Hence, $R_1 \cup R_2 \cup R_3$ is CE-terminating modulo AC. □

As a corollary, we extend to rewriting modulo AC a previous result by Kurihara & Ohuchi [28].

THEOREM 3.3: *CE-AC-termination is a modular property for unions of composable finitely branching TRSs.*

Proof: As noticed in Proposition 2.2, it is equivalent to show that whenever two modules R_1 and R_2 extend a TRS R_0 independently, and both $R_0 \cup R_1$ and $R_0 \cup R_2$ are CE-AC-terminating, then $R_0 \cup R_1 \cup R_2$ is CE-AC-terminating, which is a easy consequence of Theorem 3.2. □

Note that the theorem above implies in particular that

- CE-AC-termination is modular for constructor-sharing unions, thus extending to AC-rewriting the result of [19], and
- CE-AC-termination is modular for disjoint unions, thus almost (because we assume finitely branching TRSs) extending the result of [19, 36].

3.4. Applications to automated and incremental proofs of termination

In order to achieve automated incremental proofs, we need to derive corollaries of previous theorems, expressing conditions of non-existence of infinite dependency AC-chains in terms of existence of suitable reduction orderings. As for the standard dependency pairs criteria, there are basic criteria which require that all pairs must be decreasing, and enhanced criteria which use estimated dependency graphs. We give below basic criteria; extending these corollaries using estimated dependency graph is left to the reader.

COROLLARY 3.1: *Let $[\mathcal{F}_1 \mid R_1] \longleftarrow [\mathcal{F}_2 \mid R_2]$ be a hierarchical extension of $R_1(\mathcal{F}_1)$. If*

1. *R_1 is CE-terminating modulo AC, and*
2. *there is no infinite AC-chain of $[\mathcal{F}_2 \mid R_2]$ over $R_1 \cup R_2 \cup \Pi$,*

then $R_1 \cup R_2$ is CE-terminating modulo AC.

Proof: We just have to consider extension $[\mathcal{F}_1 \mid R_1] \longleftarrow [\mathcal{F}_2 \cup \{\pi : 2\} \mid R_2 \cup \Pi]$ and apply Theorem 3.1. □

Our theorems may be used in practical application with the help of π-expandable orderings that are compatible with AC and the mark handling rules of the AC-dependency pairs criterion in use, as defined in Proposition 2.5.

COROLLARY 3.2: *Let $[\mathcal{F}_1 \mid R_1] \longleftarrow [\mathcal{F}_2 \mid R_2]$ be a hierarchical extension of $R_1(\mathcal{F}_1)$. If*

1. *R_1 is CE-terminating modulo AC, and*
2. *there is a weakly monotonic (resp. weakly monotonic π-expandable) reduction ordering $(\succsim, \succ)$, AC and marks compatible, such that:*
   - $R_1 \cup R_2 \subseteq \succsim$ *and*
   - $\mathrm{MDP}([\mathcal{F}_2 \mid R_2]) \subseteq \succ$

then $R_1 \cup R_2$ is terminating (resp. CE-terminating) modulo AC.

Proof: The ordering ensures that there is no infinite relative AC-chain of $[\mathcal{F}_2 \mid R_2]$ over $R_1 \cup R_2 \cup \Pi$, we may then apply Theorem 3.1 (resp. Corollary 3.1). □

Corollary 3.3: *Let $[\mathcal{F}_1 \mid R_1] \longleftarrow [\mathcal{F}_2 \mid R_2]$ be a hierarchical extension of $R_1(\mathcal{F}_1)$, let $[\mathcal{F}_3 \mid R_3]$ be a module extending R_1 independently of R_2. If*

1. *$R_1 \cup R_2$ is CE-terminating modulo AC, and*
2. *there is a weakly monotonic π-expandable ordering $(\succsim, \succ)$, AC and marks compatible, such that:*
   - $R_1 \cup R_3 \subseteq \succsim$,
   - $\mathrm{MDP}([\mathcal{F}_3 \mid R_3]) \subseteq \succ$

then $R_1 \cup R_2 \cup R_3$ is CE-terminating modulo AC.

Proof: The ordering ensures that there is no infinite relative AC-chain of $[\mathcal{F}_3 \mid R_2]$ over $R_1 \cup R_3 \cup \Pi$, we may then apply Theorem 3.2. □

These last results can be compared to what can be obtained by using dependency graphs analysis. The latter method would lead to the same set of strict constraints coming from dependency pairs of module R_3 after computation of strongly connected components, or even a smaller one because it does not take only the root symbol into account . But non-strict constraints would be $R_1 \cup R_2 \cup R_3 \subseteq \succsim$, which contains additional constraints coming from rules of R_2. This clearly suggests that the most efficient method would be to apply the hierarchical criteria above, then the dependency graph analysis on each modules.

4. Examples

The modular/incremental approach for AC-termination has been implemented in C*i*ME [8]. The implemented notion of AC-dependency pairs is the one of Kusakari, Marché and Urbain [29], with the possibility of choosing whether symbols should be marked or not. Default settings use no marks on AC-symbols while using marks on the others, as discussed in Section 2.3.3. In the following examples, we ask C*i*ME to find automatically an ordering based on polynomial interpretations, satisfying the constraints induced by Corollary 3.3, but we also use in the implementation the dependency graph criteria for each module, as discussed at end of previous section.

4.1. Example of an incremental proof: ternary arithmetics and bags

In this first example, we introduce rewriting modules each in turn, and develop an incremental termination proof.

Instead of the usual Peano's arithmetic example, we choose a more efficient efficient representation, proposed by Contejean, Marché and Rabehasaina [9]: ternary notation. This is implemented as a module whose signature $\mathcal{F}_{int}$ contains the constant $\#$ and unary symbols 1, 0 and j which will be used as postfix operators. Intuitively, $\#$ represents 0, $(x)0$ represents $3x$, $(x)1$ represents $3x+1$ and $(x)j$ represents $3x-1$. The rule part contains only one rule $R_{int} = \{\#0 \to \#\}$ (which clearly CE-terminates) to encode the relation between those constructors.

We define addition with extending signature $\mathcal{F}_+ = \{+\}$ where $+$ is associative and commutative. The rule part of our second module is:

$$R_+ = \left\{ \begin{array}{lclllcl} x + \# & \to & x & & & & \\ x0 + y0 & \to & (x+y)0 & & x0 + y1 & \to & (x+y)1 \\ x1 + y1 & \to & (x+y+\#1)j & & x0 + yj & \to & (x+y)j \\ xj + yj & \to & (x+y+\#j)1 & & x1 + yj & \to & (x+y)0 \end{array} \right.$$

Termination of $R_{int} \cup R_+$ is proven using Corollary 3.2, actually its enhanced version with dependency graphs, that is using the interpretation:

$$[\![\#]\!] = 0 \qquad [\![0]\!](x) = [\![1]\!](x) = [\![j]\!](x) = x + 1 \qquad [\![+]\!](x,y) = x + y$$

for the whole graph and the interpretation

$$[\![\#]\!] = [\![0]\!](x) = 0 \qquad [\![1]\!](x) = [\![j]\!](x) = 1 \qquad [\![+]\!](x,y) = x + y$$

for the component formed by pairs $\langle xj + yj + z, (x + y + \#j)1 + z\rangle$ and $\langle x1 + y1 + z, (x + y + \#1)j + z\rangle$.

Now we prove termination of the extension with the (third) module below, defining multiplication, as well as opposite and subtraction as auxiliary operations:

$$R_\times = \left\{ \begin{array}{lclllcl} x \times \# & \to & \# & & opp(\#) & \to & \# \\ x0 \times y & \to & (x \times y)0 & & opp((x)0) & \to & (opp(x))0 \\ x1 \times y & \to & (x \times y)0 + y & & opp((x)1) & \to & (opp(x))j \\ xj \times y & \to & (x \times y)0 - y & & opp((x)j) & \to & (opp(x))1 \\ x - y & \to & x + opp(y) & & & & \end{array} \right.$$

Knowing that $R_{int} \cup R_+$ CE-terminates, using Corollary 3.2 (enhanced using dependency graph) with the following polynomial interpretation is sufficient to fulfil the remaining constraints required for termination of $R_{int} \cup R_+ \cup R_\times$:

$$\begin{array}{ll} [\![\#]\!] = 0 & [\![0]\!](x) = [\![1]\!](x) = [\![j]\!](x) = x + 1 \\ [\![opp]\!](x) = [\![\widehat{opp}]\!] = x & [\![+]\!](x,y) = [\![-]\!](x,y) = x + y \\ [\![\times]\!](x,y) = xy + x + y & \end{array}$$

Note that the termination proof given in [9] was a hand-made proof, involving a composition of complicated polynomial interpretations as well as *ad-hoc* reasoning. The first automatic termination proof of $R_{int} \cup R_+ \cup R_\times$ was given by Borralleras & Rubio [6], and the above is another quite simple and automated proof.

Now, we may define bags, then sum and product of elements in a bag of integers. Our fourth module is independent of the others, and defines bags, using signature $\mathcal{F}_{\mathsf{Bag}}$ which contains the constant $\emptyset$ to denote the empty bag, the unary symbol $\{\_\}$ to build singletons, and the union $\cup$ which is AC. Its rule part consists of the single rule TRS $R_{\mathsf{Bag}} = \{\emptyset \cup x \to x\}$, which clearly CE-terminates.

Our fifth module defines the sum of elements of a bag of integers:

$$R_\Sigma = \left\{ \begin{array}{lllllll} \Sigma(\emptyset) & \to & \# & \quad & \Sigma(x \cup y) & \to & \Sigma(x) + \Sigma(y) \\ \Sigma(\{x\}) & \to & x & & & & \end{array} \right.$$

Applying Corollary 3.3, the polynomial interpretation

$$\begin{array}{ll} [\![\#]\!] = [\![\emptyset]\!] = [\![0]\!](x) = [\![1]\!](x) = [\![j]\!](x) = 0 & \quad [\![+]\!](x,y) = x + y \\ [\![\{\_\}]\!](x) = [\![\Sigma]\!](x) = [\![\widehat{\Sigma}]\!](x) = x & \quad [\![\cup]\!](x,y) = x + y + 1 \end{array}$$

is enough to prove the termination of the set of all rules introduced so far. In particular, no additional constraint on multiplication is required, since R_Σ extends R_+ independently of $R_\times$.

We may now introduce our sixth module defining product of elements of a bag of integers:

$$R_\bigotimes = \left\{ \begin{array}{lllllll} \bigotimes(\emptyset) & \to & \#1 & \quad & \bigotimes(x \cup y) & \to & \bigotimes(x) \times \bigotimes(y) \\ \bigotimes(\{x\}) & \to & x & & & & \end{array} \right.$$

Applying again Corollary 3.3, the polynomial interpretation

$$\begin{array}{ll} [\![\#]\!] = [\![\emptyset]\!] = 0 & \quad [\![0]\!](x) = [\![1]\!](x) = [\![j]\!](x) = [\![opp]\!](x) = 0 \\ [\![\cup]\!](x,y) = x + y + 1 & \quad [\![+]\!](x,y) = [\![\times]\!](x,y) = x + y \\ [\![-]\!](x,y) = x & \quad [\![\{\_\}]\!](x) = [\![\bigotimes]\!](x) = [\![\widehat{\bigotimes}]\!](x) = x \end{array}$$

is enough to prove the termination of the set of all rules introduced, in particular no additional constraint on Σ is required.

Finally, we have proven the termination of the whole system (containing 23 rules, 3 AC-symbols and 6 modules) by using polynomial interpretations only. As far as we know, any former method would need a more complicated ordering: other approaches may need for instance lexicographical composition which is difficult to search for automatically, and ACRPO [41] is of no use here because of rule $x1 + y1 \to (x + y + \#1)j$. Moreover those interpretations were found in a few seconds by C*i*ME in a fully automated way.

$$R_1 \left\{ \begin{array}{rclrcl}
\mathsf{ef}(x)[y]_{\mathtt{t}} & \rightarrow & \mathsf{ef}(x[y]_{\mathtt{t}}) & \mathsf{Pe}(x)[y]_{\mathtt{f}} & \rightarrow & \mathsf{Pe}(x[y]_{\mathtt{t}}) \\
(f \vee g)[s]_{\mathtt{f}} & \rightarrow & (f[s]_{\mathtt{f}}) \vee (g[s]_{\mathtt{f}}) & \neg(f)[s]_{\mathtt{f}} & \rightarrow & \neg(f[s]_{\mathtt{f}}) \\
(f \wedge g)[s]_{\mathtt{f}} & \rightarrow & (f[s]_{\mathtt{f}}) \wedge (g[s]_{\mathtt{f}}) & x[i]_{\mathtt{t}} d & \rightarrow & x \\
(f \Rightarrow g)[s]_{\mathtt{f}} & \rightarrow & (f[s]_{\mathtt{f}}) \Rightarrow (g[s]_{\mathtt{f}}) & 1[x \cdot s]_{\mathtt{t}} & \rightarrow & x \\
\exists(f)[s]_{\mathtt{f}} & \rightarrow & \exists(f[1 \cdot (s \circ \uparrow)]_{\mathtt{f}}) & f[i]_{\mathtt{f}} d & \rightarrow & f \\
\forall(f)[s]_{\mathtt{f}} & \rightarrow & \forall(f[1 \cdot (s \circ \uparrow)]_{\mathtt{f}}) & (f[s]_{\mathtt{f}})[t]_{\mathtt{f}} & \rightarrow & f[s \circ t]_{\mathtt{f}} \\
(x[s]_{\mathtt{t}})[t]_{\mathtt{t}} & \rightarrow & x[s \circ t]_{\mathtt{t}} & id \circ s & \rightarrow & s \\
\uparrow \circ (x \cdot s) & \rightarrow & s & (s \circ t) \circ u & \rightarrow & s \circ (t \circ u) \\
(x \cdot s) \circ t & \rightarrow & (x[t]_{\mathtt{t}}) \cdot (s \circ t) & s \circ id & \rightarrow & s \\
1 \cdot \uparrow & \rightarrow & id & (1[s]_{\mathtt{t}}) \cdot (\uparrow \circ s) & \rightarrow & s
\end{array} \right.$$

Figure 1: Sequent calculus modulo, part I

4.2. Using automatic hierarchical decomposition: Sequent calculus modulo

This example by Deplagne comes from his study of sequent calculus modulo [11] to which we refer for further details. No proof of its termination was known when it was submitted to us.

The considered TRS is in two parts. The first one, R_1, given in Figure 1, provides explicit substitutions for quantifiers while the second one, R_2, given Figure 2, describes a congruence over a sequent calculus. Operators $\{_{\mathtt{s}}\_\}$ and $\{_{\mathtt{f}}\_\}$ denote respectively singletons of sequents and singletons of formulae. The union operators of these sets are AC and are denoted respectively '$\bullet$' and ','.

We are interested in proving termination of TRS $R_1 \cup R_2$, made of 53 rules and having 2 AC-symbols. We gathered the results in the following table (times are obtained on a Pentium 3, 933MHz CPU). The first column distinguishes

proof method	bound	no marks	marks on non-AC	all marks
global	2	22"6	-	-
global	3	18"3	390"9	-
incremental	2	8"6	33"3	253"4
incremental	3	6"6	15"6	137"9

global approach of the termination proof (but still using estimated dependency graphs, without them finding a proof is hopeless), and automatic decomposition into minimal modules (leading to 19 modules, 9 of which are trivial, i.e. with an empty rule part) allowing an incremental proof. The second column gives the bound we chose for the polynomial coefficients to look for (2 is the default in C*i*ME, 3 leads to better results on that particular example). Others columns gives results for several marking choices with respectively, no marks at all, marks on non-AC symbols and on all symbols. A '-' means we interrupted computation after at least 15 minutes.

$$R_2 \left\{ \begin{array}{rcl}
a, \triangledown & \rightarrow & a \\
a, a & \rightarrow & a \\
a \bullet \diamond & \rightarrow & a \\
a \bullet a & \rightarrow & a \\
\neg(\neg(f)) & \rightarrow & f \\
f \wedge f & \rightarrow & f \\
f \vee f & \rightarrow & f \\
f \Rightarrow g & \rightarrow & \neg(f) \vee g \\
\exists(f) & \rightarrow & \neg(\forall(\neg(f))) \\
(a, \{_{\mathtt{f}} \neg(f)\}) \vdash b & \rightarrow & a \vdash (\{_{\mathtt{f}} f\}, b) \\
\{_{\mathtt{f}} \neg(f)\} \vdash b & \rightarrow & \triangledown \vdash (b, \{_{\mathtt{f}} f\}) \\
a \vdash (\{_{\mathtt{f}} \neg(f)\}, b) & \rightarrow & (a, \{_{\mathtt{f}} f\}) \vdash b \\
a \vdash \{_{\mathtt{f}} \neg(f)\} & \rightarrow & (a, \{_{\mathtt{f}} f\}) \vdash \triangledown \\
(a, \{_{\mathtt{f}} f \wedge g\}) \vdash b & \rightarrow & (a, \{_{\mathtt{f}} f\}, \{_{\mathtt{f}} g\}) \vdash b \\
\{_{\mathtt{f}} f \wedge g\} \vdash b & \rightarrow & (\{_{\mathtt{f}} f\}, \{_{\mathtt{f}} g\}) \vdash b \\
a \vdash (\{_{\mathtt{f}} f \vee g\}, b) & \rightarrow & a \vdash (\{_{\mathtt{f}} f\}, \{_{\mathtt{f}} g\}, b) \\
a \vdash \{_{\mathtt{f}} f \vee g\} & \rightarrow & a \vdash (\{_{\mathtt{f}} f\}, \{_{\mathtt{f}} g\}) \\
\{_{\mathtt{s}} a \vdash (\{_{\mathtt{f}} f \wedge g\}, b)\} & \rightarrow & \{_{\mathtt{s}} a \vdash (\{_{\mathtt{f}} f\}, b)\} \bullet \{_{\mathtt{s}} a \vdash (\{_{\mathtt{f}} g\}, b)\} \\
\{_{\mathtt{s}} a \vdash \{_{\mathtt{f}} f \wedge g\}\} & \rightarrow & \{_{\mathtt{s}} a \vdash \{_{\mathtt{f}} f\}\} \bullet \{_{\mathtt{s}} a \vdash \{_{\mathtt{f}} g\}\} \\
\{_{\mathtt{s}} (a, \{_{\mathtt{f}} f \vee g\}) \vdash b\} & \rightarrow & \{_{\mathtt{s}} (a, \{_{\mathtt{f}} f\}) \vdash b\} \bullet \{_{\mathtt{s}} (a, \{_{\mathtt{f}} g\}) \vdash b\} \\
\{_{\mathtt{s}} \{_{\mathtt{f}} f \vee g\} \vdash b\} & \rightarrow & \{_{\mathtt{s}} \{_{\mathtt{f}} f\} \vdash b\} \bullet \{_{\mathtt{s}} \{_{\mathtt{f}} g\} \vdash b\} \\
\{_{\mathtt{s}} (a, \{_{\mathtt{f}} f\}) \vdash (\{_{\mathtt{f}} f\}, b)\} & \rightarrow & \diamond \\
\{_{\mathtt{s}} (a, \{_{\mathtt{f}} f\}) \vdash \{_{\mathtt{f}} f\}\} & \rightarrow & \diamond \\
\{_{\mathtt{s}} \{_{\mathtt{f}} f\} \vdash (b, \{_{\mathtt{f}} f\})\} & \rightarrow & \diamond \\
\{_{\mathtt{s}} \{_{\mathtt{f}} f\} \vdash \{_{\mathtt{f}} f\}\} & \rightarrow & \diamond \\
\{_{\mathtt{s}} a \vdash b\} \bullet \{_{\mathtt{s}} (a, f) \vdash (g, b)\} & \rightarrow & \{_{\mathtt{s}} a \vdash b\} \\
\{_{\mathtt{s}} a \vdash b\} \bullet \{_{\mathtt{s}} (a, f) \vdash b\} & \rightarrow & \{_{\mathtt{s}} a \vdash b\} \\
\{_{\mathtt{s}} a \vdash b\} \bullet \{_{\mathtt{s}} a \vdash (b, f)\} & \rightarrow & \{_{\mathtt{s}} a \vdash b\} \\
\{_{\mathtt{s}} a \vdash \triangledown\} \bullet \{_{\mathtt{s}} (a, f) \vdash b\} & \rightarrow & \{_{\mathtt{s}} a \vdash \triangledown\} \\
\{_{\mathtt{s}} a \vdash (b, f)\} \bullet \{_{\mathtt{s}} \triangledown \vdash b\} & \rightarrow & \{_{\mathtt{s}} \triangledown \vdash b\} \\
\{_{\mathtt{s}} a \vdash b\} \bullet \{_{\mathtt{s}} \triangledown \vdash b\} & \rightarrow & \{_{\mathtt{s}} \triangledown \vdash b\} \\
\{_{\mathtt{s}} a \vdash b\} \bullet \{_{\mathtt{s}} a \vdash \triangledown\} & \rightarrow & \{_{\mathtt{s}} a \vdash \triangledown\} \\
\{_{\mathtt{s}} a \vdash b\} \bullet \{_{\mathtt{s}} \triangledown \vdash \triangledown\} & \rightarrow & \{_{\mathtt{s}} \triangledown \vdash \triangledown\}
\end{array} \right.$$

Figure 2: Sequent calculus modulo, part II

These proofs are the only termination proofs of this system known to date. They were discovered automatically, thanks to the C*i*ME tool. The figures lead to the following comments:

- Using hierarchical criteria, and dependency graph criteria on each modules, improves significantly over using dependency graph approach on the whole system. This can be explained simply because the non-strict constraints to solve are weaker: $l \gtrsim r$ is not required for all rules.

- Proofs without marks are found significantly quicker. This is clearly because the number of unknowns (i.e. values of coefficients of polynomials) is smaller. Of course, examples exist where marks are mandatory.
- The fact that it is quicker to find a solution with a bound 3 for coefficients instead of 2 can be surprising since the search space is larger. In fact, for a larger search space, there are many more possible solutions, so finding one of those appears to be easier.

Second and third points stress that choosing good heuristics is quite a difficult issue when one is concerned with achieving full automation.

5. Conclusions and further work

We combined the respective powers of methods based on dependency pairs for AC-rewriting [17, 29, 31, 34], which are covered by our notion of abstract AC-dependency pairs; exploitation of the hierarchical structure; and CE-termination.

In so doing, we obtained new incremental/modular termination criteria for AC-rewriting (Theorem 3.1, Theorem 3.2), an extension to the AC-case of the modularity results for CE-termination (Theorem 3.3), and corollaries providing an automatic and incremental/modular way for proving AC-CE-termination. These criteria were implemented in the termination toolbox of the C*i*ME rewriting tool and we gave significant examples of its power.

Since dependency pairs criteria exist for termination of rewriting modulo E [17], a natural question is whether it is possible to extend our modularity results to this arbitrary case. In the cases where E consists of so-called *permutative axioms* (that is of the form $f(x_1, .., x_n) = f(x_{\rho(1)}, .., x_{\rho(n)})$ for some permutation ρ), it is quite obvious that our technique will still work: in particular, since those axioms are preserving the root symbol, the abstract property (ii) will hold. On the other hand, if E is more complex, the axioms of E will probably destroy the hierarchical structure of dependency chains, thus a similar proof will not work. But notice that in practice, rewriting modulo a theory E that is not made only of permutative axioms hardly ever terminates [33]: for instance, modulo the idempotency axiom $x + x = x$, any rule $l \to r$ generates infinite reductions $l = l + l \to l + r \to l + l + r \to \cdots$

A question of more concern in practice is how to generalise these incrementality results to variants of rewriting such as conditional rewriting, constrained rewriting and higher-order rewriting. For all these cases, a first step would be to design a variant of dependency pairs criteria. For higher-order rewriting, only weak versions are known [42] (requiring non-duplication in particular). For conditional rewriting, some transformations to unconditional TRS are known [38], which preserves hierarchical structure, hence allowing incremental proofs, but this does not allow to establish any modularity result on termination of unions of terminating CTRSs.

Finally, another interesting future work is to generalise the incrementality results to rewriting with strategies. Among these, innermost strategy has been extensively studied, and indeed known dependency pair criteria [16] for innermost termination are enough to allow modular/incremental proofs. Another kind of strategy is context-sensitive rewriting [32], an important issue for OBJ [18] or Maude [7] systems, for which some partial modularity results are known [20] but no dependency pair criteria so far.

Acknowledgments

We gratefully thank the anonymous referees for their fruitful help in improving this paper.

References

1. T. Arts and J. Giesl. Automatically proving termination where simplification orderings fail. In M. Bidoit and M. Dauchet, editors, *Theory and Practice of Software Development*, volume 1214 of *Lecture Notes in Computer Science*, Lille, France, Apr. 1997. Springer-Verlag.

2. T. Arts and J. Giesl. Termination of term rewriting using dependency pairs. *Theoretical Computer Science*, 236:133–178, 2000.

3. F. Baader and T. Nipkow. *Term Rewriting and All That*. Cambridge University Press, 1998.

4. L. Bachmair and D. A. Plaisted. Termination orderings for associative-commutative rewriting systems. *Journal of Symbolic Computation*, 1(4):329–349, Dec. 1985.

5. A. Ben Cherifa and P. Lescanne. Termination of rewriting systems by polynomial interpretations and its implementation. *Science of Computer Programming*, 9:137–159, 1987.

6. C. Borralleras and A. Rubio. Monotonic AC-compatible semantic path orderings. In R. Nieuwenhuis, editor, *14th International Conference on Rewriting Techniques and Applications*, volume 2706 of *Lecture Notes in Computer Science*, Valencia, Spain, June 2003. Springer-Verlag.

7. M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and J. F. Quesada. Maude: Specification and programming in rewriting logic. *Theoretical Computer Science*, 285(2):187–243, Aug. 2002.

8. E. Contejean, C. Marché, B. Monate, and X. Urbain. Proving termination of rewriting with CiME. In A. Rubio, editor, *Extended Abstracts of the 6th International Workshop on Termination, WST'03*, pages 71–73, June 2003. `http://cime.lri.fr`.

9. E. Contejean, C. Marché, and L. Rabehasaina. Rewrite systems for natural, integral, and rational arithmetic. In H. Comon, editor, *8th International Conference on Rewriting Techniques and Applications*, volume 1232 of *Lecture Notes in Computer Science*, Barcelona, Spain, June 1997. Springer-Verlag.

10. C. Delor and L. Puel. Extension of the associative path ordering to a chain of associative-commutative symbols. In C. Kirchner, editor, *5th International Conference on Rewriting Techniques and Applications*, volume 690 of *Lecture Notes in Computer Science*, pages 389–404, Montreal, Canada, June 1993. Springer-Verlag.

11. É. Deplagne. Sequent Calculus Viewed Modulo. In Catherine Pilière, editor, *Proceedings of the Fifth ESSLLI Student Session*, pages 66–76, University of Birmingham, 2000.

12. N. Dershowitz. Termination of rewriting. *Journal of Symbolic Computation*, 3(1):69–115, Feb. 1987.

13. N. Dershowitz. Hierarchical termination. In N. Dershowitz and N. Lindenstrauss, editors, *Proceedings of the Fourth International Workshop on Conditional and Typed Rewriting Systems (Jerusalem, Israel, July 1994)*, volume 968, pages 89–105, Berlin, 1995. Springer-Verlag.

14. N. Dershowitz and J.-P. Jouannaud. Rewrite systems. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, pages 243–320. North-Holland, 1990.

15. M. Fernández and J.-P. Jouannaud. Modular termination of term rewriting systems revisited. In E. Astesiano, G. Reggio, and A. Tarlecki, editors, *Recent Trends in Data Type Specification*, Lecture notes in Computer Science, vol.906, pages 255–272. Springer-Verlag, 1995. Refereed selection of papers presented at ADT'94.

16. J. Giesl, T. Arts, and E. Ohlebusch. Modular termination proofs for rewriting using dependency pairs. *Journal of Symbolic Computation*, 34(2):21–58, 2002.

17. J. Giesl and D. Kapur. Dependency pairs for equational rewriting. In A. Middeldorp, editor, *12th International Conference on Rewriting Techniques and Applications*, volume 2051 of *Lecture Notes in Computer Science*, pages 93–108, Utrecht, The Netherlands, May 2001. Springer-Verlag.

18. J. A. Goguen, T. Winkler, J. Meseguer, K. Futatsugi, and J.-P. Jouannaud. *Software Engineering with OBJ: Algebraic Specification in Action*, chapter Introducing OBJ*. Kluwers Academic Publishers, 2000.

19. B. Gramlich. Generalized sufficient conditions for modular termination of rewriting. *Applicable Algebra in Engineering, Communication and Computing*, 5:131–158, 1994.

20. B. Gramlich and S. Lucas. Modular termination of context-sensitive rewriting. In C. Kirchner, editor, *Proc. of 4th ACM Sigplan Conference on Principles and Practice of Declarative Programming, PPDP'02*, pages 50–61, Pittsburgh, USA, Oct. 2002. ACM Press, New York.

21. J.-P. Jouannaud and H. Kirchner. Completion of a set of rules modulo a set of equations. *SIAM Journal on Computing*, 15(4):1155–1194, 1986.

22. D. Kapur and G. Sivakumar. A total, ground path ordering for proving termination of AC-rewrite systems. In H. Comon, editor, *8th International Conference on Rewriting Techniques and Applications*, volume 1232 of *Lecture Notes in Computer Science*, Barcelona, Spain, June 1997. Springer-Verlag.

23. D. Kapur and G. Sivakumar. Proving associative-commutative termination using RPO-compatible orderings. In R. Caferra and G. Salzer, editors, *Automated Deduction in Classical and Non-Classical Logics*, number 1761 in Lecture Notes in Artificial Intelligence, pages 40–62. Springer-Verlag, Jan. 2000.

24. M. R. K. Krishna Rao. Simple termination of hierarchical combinations of term rewriting systems. In *Theoretical Aspects of Computer Software*, volume 789 of *Lecture Notes in Computer Science*, pages 203–223. Springer-Verlag, 1994.

25. M. R. K. Krishna Rao. Modular proofs for completeness of hierarchical term rewriting systems. *Theoretical Computer Science*, 151:487–512, 1995.

26. M. Kurihara and A. Ohuchi. Modularity of simple termination of term rewriting systems. *Journal of Information Processing Society, Japan*, 34:632–642, 1990.

27. M. Kurihara and A. Ohuchi. Modularity of simple termination of term rewriting systems with shared constructors. *Theoretical Computer Science*, 103:273–282, 1992.

28. M. Kurihara and A. Ohuchi. Decomposable termination of composable term rewriting systems. *IEICE*, E78–D(4):314–320, Apr. 1995.

29. K. Kusakari, C. Marché, and X. Urbain. Termination of associative-commutative rewriting using dependency pairs criteria. Research Report 1304, LRI, 2002. `http://www.lri.fr/~urbain/textes/rr1304.ps.gz`.

30. K. Kusakari, M. Nakamura, and Y. Toyama. Argument filtering transformation. In G. Nadathur, editor, *Principles and Practice of Declarative Programming, International Conference PPDP'99*, volume 1702 of *Lecture Notes in Computer Science*, pages 47–61, Paris, 1999. Springer-Verlag.

31. K. Kusakari and Y. Toyama. On proving AC-termination by AC-dependency pairs. *IEICE Transactions on Information and Systems*, E84-D(5):604–612, 2001.

32. S. Lucas. Context-sensitive rewriting strategies. *Information and Computation*, 178(1):294–343, 2002.

33. C. Marché. Normalized rewriting: an alternative to rewriting modulo a set of equations. *Journal of Symbolic Computation*, 21(3):253–288, 1996.

34. C. Marché and X. Urbain. Termination of associative-commutative rewriting by dependency pairs. In T. Nipkow, editor, *9th International Conference on Rewriting Techniques and Applications*, volume 1379 of *Lecture Notes in Computer Science*, pages 241–255, Tsukuba, Japan, Apr. 1998. Springer-Verlag.

35. A. Middeldorp and Y. Toyama. Completeness of combinations of constructor systems. *Journal of Symbolic Computation*, 15:331–348, 1993.

36. E. Ohlebusch. On the modularity of termination of term rewriting systems. *Theoretical Computer Science*, 136:333–360, 1994.

37. E. Ohlebusch. Modular properties of composable term rewriting systems. *Journal of Symbolic Computation*, 20:1–41, 1995.

38. E. Ohlebusch. *Advanced Topics in Term Rewriting.* Springer-Verlag, 2002.

39. E. Ohlebusch. Hierarchical termination revisited. *Information Processing Letters*, 84(4):207–214, 2002.

40. G. E. Peterson and M. E. Stickel. Complete sets of reductions for some equational theories. *J. ACM*, 28(2):233–264, Apr. 1981.

41. A. Rubio. A fully syntactic AC-RPO. In P. Narendran and M. Rusinowitch, editors, *10th International Conference on Rewriting Techniques and Applications*, volume 1631 of *Lecture Notes in Computer Science*, Trento, Italy, July 1999. Springer-Verlag.

42. M. Sakai and K. Kusakari. On new dependency pair method for proving termination of higher-order rewrite systems. In *International Workshop on Rewriting in Proof and Computation (RPC'01)*, pages 176–187, Sendai, Japan, Oct. 2001. `http://www.sakabe.nuie.nagoya-u.ac.jp/~sakai/papers/rpc01a.pdf`.

43. Y. Toyama. Counterexamples to termination for the direct sum of term rewriting systems. *Information Processing Letters*, 25:141–143, Apr. 1987.

44. X. Urbain. *Approche incrémentale des preuves automatiques de terminaison.* Thèse de doctorat, Université Paris-Sud, Orsay, France, Oct. 2001. `http://www.lri.fr/~urbain/textes/these.ps.gz`.

45. X. Urbain. Automated incremental termination proofs for hierarchically defined term rewriting systems. In R. Goré, A. Leitsch, and T. Nipkow, editors, *International Joint Conference on Automated Reasoning*, volume 2083 of *Lecture Notes in Artificial Intelligence*, pages 485–498, Siena, Italy, June 2001. Springer-Verlag.

46. X. Urbain. Modular and incremental automated termination proofs. *Journal of Automated Reasoning*, 2004.