Termination of Associative-Commutative Rewriting by Dependency Pairs*

Claude Marché and Xavier Urbain

LRI, CNRS URA 410
Bât. 490, Université Paris-Sud, Centre d'Orsay
91405 Orsay Cedex, France
Phone:+33 1 69 15 64 85, Fax: +33 1 69 15 65 86
Email: {marche,urbain}@lri.fr

Abstract. A new criterion for termination of rewriting has been described by Arts and Giesl in 1997. We show how this criterion can be generalized to rewriting modulo associativity and commutativity. We also show how one can build *weak* AC-compatible reduction orderings which may be used in this criterion.

1 Introduction

For almost all applications of rewrite systems, one is interested in finite termination of reductions, a property called *termination* or *strong normalization* of a system. When rewriting is used as a model of computation, for example in relation to functional programming, termination insures the existence of the result of computation. When rewriting is used for automated reasoning in equational logic, existence of normal forms is also important, and a terminating rewrite system can also be the basis of a Noetherian induction.

Thus, checking termination of a given rewrite system is a crucial issue. Although this property is known to be computationally unsolvable in general, it is important to have automatic tools, correct but incomplete, that can prove termination of rewrite systems one meets in practice.

Until 1997, among the methods known for proving termination of a TRS, the only implementable one was the technique consisting in building a *reduction ordering* on terms with which we can check automatically that the rewrite rules we consider all strictly decrease with respect to this ordering. Thus, implementing a program for checking termination consisted mainly as implementing ways of building reduction orderings.

In 1997, Arts and Giesl [1,2] proposed a new method for proving termination which does not consist anymore in checking that rules strictly decrease but in a more careful check on structure of the rules, the important point being that this remains a purely syntactic finite test (the so-called *dependency pairs*) thus it is still implementable.

* This research was supported in part by the EWG CCL II, the HCM Network CONSOLE, and the "GDR de programmation du CNRS".

In this paper, we propose a generalization of the dependency pairs method to rewriting modulo AC (associativity and commutativity) which is an important issue in practice.

In fact, there are quite different notions of AC rewriting, and in Section 2 we will first describe clearly which kind of AC rewriting we are going to consider: *flat* AC rewriting, and Theorem 4 shows why using flat AC rewriting is important for us. In Section 3, we will then show how the dependency pairs criterion for standard rewriting is lifted up to flat AC rewriting (Theorem 8 and Corollary 9).

Arts and Giesl also have shown that the dependency pairs criterion can be made a bit stronger by use of a *dependency graph*, again a deeper analysis of the rewrite rules structure, and we show in Section 4 how this improvement is lifted up to AC rewriting.

The dependency pairs criterion still amounts the use of a term ordering for proving termination, but a slightly larger class of orderings is allowed: *weak* reduction orderings, that is orderings that do not need to fulfill the *strict* monotonicity property ($t_1 > t_2$ implies $f(\dots,t_1,\dots) > f(\dots,t_2,\dots)$) but only the *weak* monotonicity one ($t_1 \geq t_2$ implies $f(\dots,t_1,\dots) \geq f(\dots,t_2,\dots)$). In the case of AC rewriting, one needs *AC-compatible* weak reduction orderings and in Section 5, we will show how such orderings can be built.

Finally, in Section 6 we give a small but important remark on the use of this new criterion in completion procedures.

In the course of the paper, we will give some examples, for which the termination has been proved automatically by a new termination toolbox that we implemented inside the C*i*ME system [10] developed in our research team.

2 Basic notions

We recall first the usual notions about rewriting and give our notations. We then focus on the definition of rewriting modulo AC.

A signature $\mathcal{F}$ is a finite set of symbols with arities, the set of terms on $\mathcal{F}$ and variables X is denoted by $T(\mathcal{F},X)$. The root position is denoted by Λ, the root symbol of a term t is denoted $\mathrm{Head}(t)$, $t|_p$ denotes the subterm of t at position p, and $C[t]_p$ denotes the term obtained by putting the term t in the context C at position p.

We denote $s \xrightarrow[l\to r]{p} t$ if s rewrites to t at position p by a rule $l \to r$, and $s \xrightarrow[R]{>p}{}^{\star} t$ if s rewrites to t by a set of rules R in 0, 1 or several steps *below* p. We say that a rewrite system R *terminates* if no infinite sequence of reduction by $\to_R$ exists. We say that a term t is *strongly normalizable* (SN for short) if there is no infinite sequence of reduction by $\to_R$ starting from t.

We say that f is a *defined* symbol of R if it occurs at the top of a left-hand side, and is a *constructor* otherwise.

2.1 Terms with AC symbols, flat terms

In practice, we often have symbols f that fulfill the commutativity axiom $f(x,y) = f(y,x)$. This cannot be turned into a rule without losing termination, and the standard technique to deal with it is to consider classes of terms modulo commutativity. Moreover, if f also fulfills associativity $f(x,f(y,z)) = f(f(x,y),z)$, this equation itself cannot be turned into a rule without losing termination modulo commutativity ($f(x,f(y,z)) \to f(f(x,y),z) =_C f(z,f(x,y)) \to \cdots$), hence in such a case we also consider class of terms modulo associativity. In the following, we call such an f an AC symbol, and we denote by $\mathcal{F}_{\mathrm{AC}}$ the set of AC symbols. We do not consider symbols which are only commutative in the following, but everything can be specialized to that case.

Formally speaking, we consider the set of *varyadic* terms: AC symbols may have any arity greater than 2. The *flat form* of a term t, denoted $\overline{t}$, is the normal form of t for the 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. It is clear that $s =_{\mathrm{AC}} t$ iff $\overline{s} \equiv \overline{t}$ where $\equiv$ is the equivalence up to permutations of AC symbols arguments.

2.2 Rewriting modulo AC

Definition 1. We say that s rewrites to t by a rule $l \to r$ modulo AC, denoted by $s \xrightarrow[l\to r/\mathrm{AC}]{} t$, iff there are s' and t' s.t. $s =_{\mathrm{AC}} s'$, $t =_{\mathrm{AC}} t'$ and $s' \xrightarrow[l\to r]{} t'$.

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, thus it is not efficient at all. Peterson and Stickel [24] proposed the use of a restricted definition where when one wants to match a rule on a given term at a given position p, AC-equivalence is allowed only at or below position p.

Definition 2. The *AC-extended* rewrite relation is defined by $s \xrightarrow[\mathrm{AC}\setminus l\to r]{p} t$ iff there is a substitution σ s.t. $s|_p =_{\mathrm{AC}} l\sigma$ and $t = s[r\sigma]_p$.

Using AC-extended rewriting may lead to pathological examples where for a rewrite system R, $\to_{\mathrm{AC}\setminus R}$ terminates whereas $\to_{R/\mathrm{AC}}$ does not. This is the case for example with $R = \{a+b \to a+(b+c)\}$, where $+$ is an AC symbol: the term $a+(b+c)$ is irreducible by AC-extended rewriting, but it is reducible infinitely by class rewriting because $a+(b+c) =_{\mathrm{AC}} (a+b)+c$ which contains a redex.

This phenomenon arises because a system like R above is not *coherent* with AC in the sense defined by Jouannaud and Kirchner [17]: $s =_{\mathrm{AC}} t$ and $t \xrightarrow[\mathrm{AC}\setminus R]{} u$ should imply that there exist v and w s.t. $s \xrightarrow[\mathrm{AC}\setminus R]{}{}^{\star} v =_{\mathrm{AC}} w \xleftarrow[\mathrm{AC}\setminus R]{}{}^{\star} u$.

In fact, one may make any rewrite system R AC-coherent by adding to R the so-called *extended rules*, that is for each rule $f(s,t) \to r$ where f is AC, add the rule $f(f(s,t),x) \to f(r,x)$ where x is a new variable.

There is a variant of AC-extended rewriting that, in some sense, builds in the rewriting relation the extended rules, by considering flat terms.

Definition 3. The *flat-AC* rewrite relation [21] is defined on flat terms by $s \xrightarrow[l \to r/\overline{\mathrm{AC}}]{p} t$ iff there is a substitution σ s.t. either

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

The second case is called *extended matching*. Note that as for extension rules *à la* Peterson-Stickel, this case is not necessary if l has already a linear variable as an immediate subterm, like $x + 0 \to x$.

This definition seems not to be very different from using AC-extended rewriting with always coherent systems, but in fact it is important because the notion of position in a term has changed a bit: with flat AC rewriting, it does not mean anything to rewrite a term like $a + (b + c)$ at position 2, it is a rewrite of $a+b+c$ at position Λ with extended matching. This change in the definition was shown quite powerful for example for AC-completion: the completeness proof is far much simpler [21].

The termination criterion described in the next section is based on detecting on which position a term can be rewritten indefinitely, so the notion of position discussed above is really important.

Flat-AC rewriting is always coherent. More generally, it is Church-Rosser if it is locally confluent and terminating. As shown by Jouannaud and Kirchner [17], this Church-Rosser property is valid only if class rewriting is proved terminating. Fortunately, the following proposition allows to check only termination of flat AC-rewriting.

Theorem 4. *Termination of flat AC rewriting is equivalent to termination of class rewriting.*

Proof. Assume that a term s is not SN w.r.t. $\to_{R/\mathrm{AC}}$. An infinite reduction of s starts with $s \xrightarrow[l \to r/\mathrm{AC}]{p} t$, that is $s =_{\mathrm{AC}} s'$ where $s'|_p = l\sigma$. But then it is easy to see that $\overline{s}$ also contains an (extended) instance of l, hence rewrites with $\to_{R/\overline{\mathrm{AC}}}$ to a flat term AC-equivalent to t. By repeating this construction we build an infinite reduction for $R/\overline{\mathrm{AC}}$. The reciprocal is trivial.

2.3 Term orderings and termination

For general notions about term orderings and termination, we refer to [14]. We briefly recall what we need.

Definition 5. A *quasi-ordering* on terms is any reflexive and transitive relation $\succeq$ on $T(\mathcal{F}, X)$. The *strict part* of $\succeq$ is defined by $s \succ t$ iff $s \succeq t$ and $t \not\succeq s$, and the *equivalence part* is defined by $s \simeq t$ iff $s \succeq t$ and $t \succeq s$. $\succeq$ is said to be *well-founded* if $\succ$ is Noetherian; that is no infinite decreasing sequence $t_1 \succ t_2 \succ t_3 \succ \cdots$ exists. $\succeq$ is said *weakly monotonic* if $s \succeq t$ implies $C[s] \succeq C[t]$

for any context C, and (strictly) *monotonic* if $s \succ t$ implies $C[s] \succ C[t]$ also. It is said *stable* if $s \succeq t$ implies $s\sigma \succeq t\sigma$ and $s \succ t$ implies $s\sigma \succ t\sigma$ for any substitution σ.

A (strict) *reduction ordering* is a well-founded, monotonic and stable quasi-ordering, it is a *weak* reduction ordering if it is only weakly monotonic.

A quasi-ordering is *AC-compatible* if $s \succ t$, $s =_{\mathrm{AC}} s'$ and $t =_{\mathrm{AC}} t'$ imply $s' \succ t'$.

The following well-known proposition gives a termination criterion for rewriting by a system R modulo AC by showing that all rules of R decrease w.r.t. an AC-compatible reduction ordering.

Proposition 6. *$\to_{R/\overline{AC}}$ terminates iff there is an AC-compatible reduction ordering $\succeq$ such that $l \succ r$ for all rules $l \to r \in R$.*

3 The dependency pairs termination criterion

3.1 The basic criterion

The key idea is to show that there are infinite reductions of a particular form, involving the so-called dependency pairs.

Definition 7. For a given rewrite rule $f(t_1, \ldots, t_n) \to r$, its set of *dependency pairs* is the set of pairs of terms of the form $\langle f(t_1, \ldots, t_n), g(u_1, \ldots, u_m)\rangle$ where $g(u_1, \ldots, u_m)$ is any subterm of r with a defined root symbol g. If f is AC, the set of *AC-extended* dependency pairs is the union of the set of standard dependency pairs and the set of dependency pairs of $f(t_1, \ldots, t_n, x) \to \overline{f(r, x)}$ where x is a new variable that does not occur in the rule. Note that flattening is necessary if $\mathrm{Head}(r) = f$ and that extended pair is not necessary if one of t_i is already a linear variable of the initial rule, that is if the rule does not need extended matching as remark in Definition 3.

An R-chain of dependency pairs is a sequence $\langle s_1, t_1\rangle, \langle s_2, t_2\rangle, \ldots$ of pairs of R such that for a substitution σ and for each i, $t_i\sigma \xrightarrow[R/\overline{\mathrm{AC}}]{>\Lambda}{}^{\star} s_{i+1}\sigma$. (We consider only one substitution for all i because we can assume by renaming that sets of variables of two different pairs are disjoint.)

Example: Peàno arithmetic. Addition and multiplication on natural numbers presented by 0 and successor s can be defined by

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

where $+$ and $\times$ are AC. The constructors are 0 and s so the dependency pairs of this TRS are $\langle x + s(y), x + y\rangle$, $\langle x + s(y) + z, x + y\rangle$ and $\langle x + s(y) + z, s(x + y) + z\rangle$ from the second rule; $\langle x \times 0 \times z, 0 \times z\rangle$ from the third; and $\langle x \times s(y), x \times y\rangle$, $\langle x \times s(y), (x \times y) + x\rangle$, $\langle x \times s(y) \times z, x \times y\rangle$, $\langle x \times s(y) \times z, ((x \times y) + x)\rangle$ and $\langle x \times s(y) \times z, ((x \times y) + x) \times z\rangle$ from the fourth.

Theorem 8. *For a given rewrite system R, the relation $\xrightarrow[R/\overline{AC}]{}$ terminates if and only if there is no infinite R-chain.*

We do not give the proof here due to lack of space, see [26]. It does not differ essentially from the one of Arts and Giesl, one simply has to take care where flattening may occur.

Corollary 9. *If there is an AC-compatible weak reduction ordering $\succeq$ such that $l \succeq r$ for each rule of R and $s \succ t$ for each dependency pair $\langle s, t\rangle$ of R, then $R/\overline{AC}$ is terminating.*

Note that weak monotonicity is enough because each dependency pair corresponds to a rewrite step without context in R-chains.

3.2 Dependency pairs criterion with marked symbols

Arts and Giesl have shown an enhancement of the criterion by marking, in R-chains, the top symbols of the non-SN subterms. Because of flattening, the definition of R-chains have to be modified a bit, and the reason for this modification will be shown by the counter-example below.

Definition 10. Let $\check{\mathcal{F}} = \mathcal{F} \cup \{\check{f}, f \text{ defined symbol}\}$ an extended signature where each defined symbol is copied with a mark. For a rule $f(t_1, \ldots, t_n) \to r$, its set of *dependency pairs with marks* is the set of pairs of terms in $T(\check{\mathcal{F}}, X)$ of the form $\langle \check{f}(t_1, \ldots, t_n), \check{g}(u_1, \ldots, u_m)\rangle$ where $g(u_1, \ldots, u_m)$ is any subterm of r with a defined root symbol g. The marked symbols only occur at root position. The set of marked *AC-extended* dependency pairs is defined accordingly.

Let $\check{R}$ be R augmented with rules $l \to \check{f}(\mathbf{t})$ for each R-rule of the form $l \to f(\mathbf{t})$ where f is a defined AC symbol and is not at root position in l. A *marked R-chain* of dependency pairs is a sequence $\langle s_1, t_1\rangle, \langle s_2, t_2\rangle, \ldots$ of pairs of R s.t. for a substitution σ and for each i, $t_i\sigma \xrightarrow[\check{R}/\overline{\mathrm{AC}}]{>\Lambda}{}^{\star} s_{i+1}\sigma$.

Proposition 11. *For a given rewrite system R, there are infinite marked R-chains iff there are infinite non-marked R-chains.*

Proof. Given a chain with marks, you can remove the marks and reciprocally, given a chain without marks, you can put marks, possibly introducing rewrite steps with rules in $\check{R}$.

Marked symbols will allow a larger class of orderings to be used in the following corollary.

Corollary 12. *If there is an AC-compatible weak reduction ordering $\succeq$ such that $l \succeq r$ for each rule of $\check{R}$ and $s \succ t$ for each marked dependency pair $\langle s, t\rangle$ of R, then $R/\overline{AC}$ is terminating.*

Counter-example (if we do not consider rules of $\check{R}$). The rewrite system $R = \{a \to b + c, d + b \to d + a\}$ is not terminating, because of the infinite sequence $d{+}a \to d{+}b{+}c \to d{+}a{+}c$ etc. which appears to be an infinite non-marked chain. However the correponding marked chain is $d\check{+}a \to d\check{+}b\check{+}c \to d\check{+}a\check{+}c$ involving a rewrite step $a \to b\check{+}c$.

One can see that in some sense, this modification is made because we need to "flatten" $d\check{+}(b + c)$ into $d\check{+}b\check{+}c$.

Example: Binary arithmetic. For computing with natural numbers, a binary notation is much more efficient than Peàno notation. Such a binary notation can be defined by a constant # and two unary postfixed functions $(_)0$ and $(_)1$. For example, 4 is written $(((\#)1)0)0$ that is a # followed by the usual binary notation. (Semantically, # denotes 0, $(x)0$ denotes $2x$ and $(x)1$ denotes $2x + 1$.) Addition and multiplication can be defined by the following TRS:

$$\begin{array}{lll} (\#)0 \to \# & (x)0 + (y)0 \to (x + y)0 & x \times (y)0 \to (x \times y)0 \\ x + \# \to x & (x)0 + (y)1 \to (x + y)1 & x \times (y)1 \to x + (x \times y)0 \\ x \times \# \to \# & (x)1 + (y)1 \to (x + y + (\#)1)0 & \end{array}$$

where $+$ and $\times$ are AC. This TRS has 26 marked AC-extended dependency pairs, we only give here the dependency pairs of the third rule in the middle.

$$\begin{array}{cc} \langle (x)1\check{+}(y)1, x\check{+}y\check{+}(\#)1\rangle & \langle z\check{+}(x)1\check{+}(y)1, x\check{+}y\check{+}(\#)1\rangle \\ \langle (x)1\check{+}(y)1, (x + y + (\#)1)\check{0}\rangle & \langle z\check{+}(x)1\check{+}(y)1, (x + y + (\#)1)\check{0}\rangle \\ & \langle z\check{+}(x)1\check{+}(y)1, z\check{+}(x + y + (\#)1)0\rangle \end{array}$$

When computing $2+3$, that is $((\#)1)0+((\#)1)1$, with this system we obtain an R-chain: $\langle (x)0\check{+}(y)1, x\check{+}y\rangle\ \langle (x')1\check{+}(y')1, x'\check{+}y'\check{+}(\#)1\rangle$ with substitution $\{x \mapsto (\#)1, y \mapsto (\#)1, x' \mapsto \#, y' \mapsto \#\}$.

4 Dependency graphs

Dependency graphs allow one to not check that *all* dependency pairs strictly decrease. We now lift this refinement up to AC-TRS by defining AC-dependency graphs.

Definition 13. The *AC-dependency graph* of a rewrite system R is a graph having AC-extended dependency pairs as vertices and s.t. there is an edge from $\langle s_1, t_1\rangle$ to $\langle s_2, t_2\rangle$ iff there is a substitution σ that allows $t_1\sigma \xrightarrow[\check{R}/\overline{\mathrm{AC}}]{>\Lambda}{}^{\star} s_2\sigma$.

We know that R terminates modulo AC iff there is no R-chain, but an R-chain corresponds to an infinite path in the graph. Since the graph is always finite, infinite paths depend on cycles. Hence, we just have to check that the pairs decrease in cycles.

Corollary 14. *Let R be an AC-TRS and $\mathcal{G}$ be its AC-dependency graph. If there is an AC-compatible weak reduction quasi-ordering $\succeq$ such that:*

– *for each rule* $l \to r \in \check{R}$, $l \succeq r$,
– *for each cycle* $\mathcal{C} \in \mathcal{G}$ *and each* $\langle s, t\rangle \in \mathcal{C}$, $s \succeq t$,
– *each cycle* $\mathcal{C} \in \mathcal{G}$ *contains a strictly decreasing pair,*

then $R/\overline{AC}$ *is terminating.*

Unfortunately, existence of a σ s.t. $t_1\sigma \xrightarrow[\check{R}/\overline{\mathrm{AC}}]{>\Lambda}{}^{\star} s_2\sigma$ for some pairs $\langle s_1, t_1\rangle$ and $\langle s_2, t_2\rangle$ is undecidable in general. An approximation of the dependency graph must be computed in such a way that the criterion soundness is preserved. Again we adapt the approximation proposed by Arts and Giesl for standard rewriting.

Definition 15. For any term t, let $\mathrm{CAP}(t)$ be t where all subterms headed by defined symbols (or variables) have been replaced by new distinct variables. A term t is said *connectable* to a term t' if $\mathrm{CAP}(t)$ and t' unify modulo AC. The *approximated AC-dependency graph* of an AC-TRS R is a graph whose vertices are the AC-extended dependency pairs and s.t. there is an edge from $\langle s_1, t_1\rangle$ to $\langle s_2, t_2\rangle$ iff t_1 and s_2 are connectable.

Our aim now is to show that this approximated graph indeed contains the AC-dependency graph.

Lemma 16. *If* $s \xrightarrow[\check{R}/\overline{AC}]{}{}^{\star} t$ *then* $\mathrm{CAP}(s)$ *and* t *are AC-unifyable.*

Proof. By induction on the structure of s. If s is a variable or $\mathrm{Head}(s)$ is a defined symbol then $\mathrm{CAP}(s)$ is a variable, hence it unifies with t.

If $\mathrm{Head}(s)$ is a constructor, say c, then $s = c(s_1, \ldots, s_n)$, hence $\mathrm{CAP}(s) = c(\mathrm{CAP}(s_1), \ldots, \mathrm{CAP}(s_n))$. Since c is a constructor no rewriting at root can occur, thus $t = \underline{c(t_1, \ldots, t_n)}$ where for each i $s_i \xrightarrow[\check{R}/\overline{\mathrm{AC}}]{}{}^{\star} t_i$, the flattening being necessary if some t_i has c as root symbol. By induction hypothesis $\mathrm{CAP}(s_i)$ AC-unifies with t_i for each i, but since all $\mathrm{CAP}(s_i)$ have pairwise distinct variables all these unifiers for each i can be combined into a single AC-unifier of s and t.

Corollary 17. *If there is a substitution* σ *s.t.* $s\sigma \xrightarrow[\check{R}/\overline{AC}]{>\Lambda}{}^{\star} t\sigma$ *for two terms* s *and* t*, then* s *is connectable to* t*.*

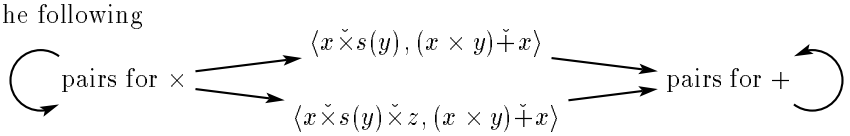
Example. Let $R = \{f(x, x) \to f(0, 1)\}$. If f is free then R terminates because the only dependency pair is $\langle \check{f}(x, x), \check{f}(0, 1)\rangle$ and $\mathrm{CAP}(\check{f}(0, 1)) = \check{f}(0, 1)$ does not unify with $\check{f}(x, x)$, hence there are no cycle at all in the dependency graph.

If f is AC, the situation is different since there is the extended pair $\langle \check{f}(x, x, y), \check{f}(0, 1, y)\rangle$ and $\mathrm{CAP}(\check{f}(0, 1, y)) = \check{f}(0, 1, y')$ AC-unifies with both left-hand sides of the two pairs. The dependency graph is then

$$\langle \check{f}(x, x), \check{f}(0, 1)\rangle \longrightarrow \langle \check{f}(x, x, y), \check{f}(0, 1, y)\rangle \circlearrowright$$

In fact, one unifier of $\check{f}(0, 1, y')$ and $\check{f}(x, x, y)$ is $\{x \mapsto 0, y \mapsto 1, y' \mapsto 0\}$ and this substitution gives us a counter-example for termination: $f(0, 0, 1) \to f(0, 1, 1) \to f(0, 0, 1) \to \cdots$. We see on this example how the graph criterion may also give some diagnostic when it cannot conclude to termination.

Example: Peàno arithmetic continued. The shape of the AC-dependency graph is the following



Thus, there is no need for an orientation for $\langle x \check{\times} s(y), (x \times y) \check{+} x \rangle$ and $\langle x \check{\times} s(y) \check{\times} z, (x \times y) \check{+} x \rangle$.

5 Building AC-compatible weak reduction orderings

As shown in the previous sections, proving termination via the dependency pair criterion amounts to building weak reduction orderings. Several constructions exist and here we are going to describe first some AC-compatible path orderings (which actually are always strictly monotonic); then we focus on how one can build weak orderings from polynomial interpretations; and finally we will show orderings built by *recursive program schemes* (which allow in particular building weak reduction orderings based on RPO).

5.1 Path orderings

We assume the reader familiar with the *Recursive Path Orderings with status* [13, 20]. We use this ordering on flat terms by requiring that AC symbols always have multiset status. But such an ordering is not always monotonic: we need high restrictions on the precedence, as shown by the following proposition.

Proposition 18. *If all AC-symbols are minimal in the precedence, then $\succeq_{rpo}$ is an AC-compatible simplification ordering.*

Associative path orderings are extensions of RPO that allow a more general precedence, they are built on RPO by comparing normal forms of terms w.r.t. some well-chosen rewrite systems [3,4,7,12]. We will use in this paper the *modified associative path ordering* defined by Delor and Puel [12]. More elaborated AC-compatible path orderings are known [12,18,23] but we don't need these powerful orderings in this paper.

Assume given a precedence s.t. for each AC symbol f, the precedence of symbols less than f has the form $f > u_1 > \cdots > u_k > g > a$ where the u_i are unary, g is AC and a is a constant (g and/or a may of course be absent). Let D be the following set of rules:

$$f(x, g(y, z)) \to g(f(x, y), f(x, z)) \qquad u_i(g(x, y)) \to g(u_i(x), u_i(y)) \text{ for all } i$$
$$f(x, u_i(y)) \to u_i(f(x, y)) \text{ for all } i \qquad u_i(u_j(x)) \to u_j(u_i(x)) \text{ for all } i > j$$

Proposition 19 (Delor and Puel [12]). *D is AC-convergent. Let $s \succeq_{mapo} t$ iff $s \downarrow_D \succeq_{rpo} t \downarrow_D$. $\succeq_{mapo}$ is an AC-compatible simplification ordering, called* modified associative path ordering.

Example: Addition in Peàno arithmetic. The precedence $+ > s$ suffices to prove that $x{+}0 \succeq x$, $x{+}s(y) \succeq s(x{+}y)$ and $x \check{+} s(y) \succ x \check{+} y$. Note that in the second rule the two sides are in fact equivalent, and if one wants to prove termination of this system without the dependency pairs criterion, one would need to compose this MAPO with a polynomial interpretation. Note also that MAPO does not permit proving termination of the system with multiplication, because the precedence $\times > + > s$ is not allowed.

Remark that the orderings defined here are always strictly monotonic. In Section 5.3, we will see a construction aimed at making weak reduction orderings from them.

5.2 Weak polynomial interpretations

Definition 20. A polynomial interpretation is given by, for each symbol f of arity n, a polynomial $[\![f]\!]$ with integral coefficients and n variables. Such an interpretation defines an interpretation of any term with variables by

- $[\![x]\!] = x$ for any variable x;
- $[\![f(t_1, \ldots, t_n)]\!] = [\![f]\!]([\![t_1]\!], \ldots, [\![t_n]\!])$;

thus for any term t, $[\![t]\!]$ is a polynomial function of the variables of t.

A polynomial interpretation defines a strict ordering on terms by $s \succ t$ iff $[\![s]\!] >_\mu [\![t]\!]$ where μ is any integer and $>_\mu$ is the ordering on polynomial functions given by $P >_\mu Q$ iff for every $\mathbf{x} \geq \mu$, $P(\mathbf{x}) > Q(\mathbf{x})$ ($\mathbf{x}$ abbreviates $x_1, \ldots, x_n$ and $\mathbf{x} \geq \mu$ means $x_i \geq \mu$ for all i). We say $s \succeq t$ iff $[\![s]\!] \geq_\mu [\![t]\!]$ where $P \geq_\mu Q$ iff for every $\mathbf{x} \geq \mu$, $P(\mathbf{x}) \geq Q(\mathbf{x})$. (Be careful that $\succ$ is *not* the strict part of $\succeq$.)

Proposition 21. *If for each $f \in \mathcal{F}$ of arity n, $\mathbf{x} \geq \mu$ implies $[\![f]\!](\mathbf{x}) \geq \mu$ and $x_i \geq y_i$ for all i implies $[\![f]\!](\mathbf{x}) \geq [\![f]\!](\mathbf{y})$ then $\succeq$ is a weak reduction ordering.*

If for any AC symbol f, $[\![f]\!](x, y)$ has the form $axy + b(x + y) + c$ with $b^2 = b + ac$, then it is AC-compatible.

The first part of this proposition is almost straightforward and the second part is already known [8]. Note that several methods exist for checking whether a polynomial is greater than another w.r.t. $>_\mu$ [8, 15, 25].

Example: Peàno arithmetic continued. Since there is no need for an orientation for $\langle x \check{\times} s(y), (x \times y) \check{+} x \rangle$, the following simple interpretation does the trick ($\mu = 1$).

$$[\![\times]\!](x, y) = xy + x + y \qquad [\![\check{\times}]\!](x, y) = x + y$$
$$[\![+]\!](x, y) = x + y \qquad [\![\check{+}]\!](x, y) = x + y$$
$$[\![s]\!](x) = x + 1 \qquad [\![0]\!] = 1$$

Example: Binary arithmetic continued. There is no infinite chain because we can apply Corollary 9 with the polynomial interpretation ($\mu = 2$):

$$\begin{array}{ll} [\![\times]\!](x,y) = xy - x - y + 2 & [\![\check{\times}]\!](x,y) = xy + 2(x+y) + 2 \\ [\![+]\!](x,y) = x + y - 2 & [\![\check{+}]\!](x,y) = x + y - 1 \\ [\![(_)0]\!](x) = x + 1 & [\![(_)\check{0}]\!](x) = x + 1 \\ [\![(_)1]\!](x) = x + 2 & [\![\#]\!] = 2 \end{array}$$

Example: Ternary integral arithmetic. Contejean, Marché and Rabehasaina proposed the following system for addition and multiplication of integers in *balanced ternary notation* [11, 19] (semantically, $(x)0$, $(x)1$ and $(x)j$ denote respectively $3x$, $3x+1$ and $3x-1$):

$$\begin{array}{rl} (\#)0 \to \# & x - y \to x + \mathrm{opp}(y) \\ x + \# \to x & \mathrm{opp}(\#) \to \# \\ (x)0 + (y)0 \to (x+y)0 & \mathrm{opp}((x)0) \to (\mathrm{opp}(x))0 \\ (x)0 + (y)1 \to (x+y)1 & \mathrm{opp}((x)1) \to (\mathrm{opp}(x))j \\ (x)0 + (y)j \to (x+y)j & \mathrm{opp}((x)j) \to (\mathrm{opp}(x))1 \\ (x)1 + (y)j \to (x+y)0 & x \times \# \to \# \\ (x)1 + (y)1 \to (x+y+(\#)1)j & x \times (y)0 \to (x \times y)0 \\ (x)j + (y)j \to (x+y+(\#)j)1 & x \times (y)1 \to x + (x \times y)0 \\ & x \times (y)j \to (x \times y)0 + \mathrm{opp}(x) \end{array}$$

To prove this system terminates modulo AC they need a quite complicated ad-hoc demonstration, and no automatic proof of termination was known. With the dependency pairs criterion, termination can be checked automatically by the interpretation:

$$\begin{array}{lll} [\![\#]\!] = 2 & [\![\check{\times}]\!](x,y) = xy + 2(x+y) + 2 & [\![\check{-}]\!](x,y) = x + y + 2 \\ [\![(_)0]\!](x) = x + 1 & [\![\check{+}]\!](x,y) = x + y - 1 & [\![\mathrm{o\check{p}p}]\!](x) = x + 2 \\ [\![(_)1]\!](x) = x + 2 & [\![\times]\!](x,y) = xy - (x+y) + 2 & [\![-]\!](x,y) = x + y \\ [\![(_)j]\!](x) = x + 2 & [\![+]\!](x,y) = x + y - 2 & [\![\mathrm{opp}]\!](x) = x \\ [\![(_)\check{0}]\!](x) = x + 1 & & \end{array}$$

5.3 Recursive program schemes

Definition 22. A rewrite system R is called a *recursive program scheme* (RPS for short) if:

- each rule is of the form $f(x_1, ..., x_n) \to r$ where x_i are pairwise distinct variables and r is any term;
- each defined symbol occurs at most once on left-hand sides;

It is an *AC-RPS* if for each defined AC symbol f, the rule for f has the form $f(x,y) \to r$ where $r =_{AC} r\{x \mapsto y, y \mapsto x\}$.

Note that it is easy to check whether a rewrite system of this form terminates: there should be neither recursive nor mutually recursive rules. Further note that we do not allow rules for AC symbols having more than 2 variables on the left-hand side, see the counter-example below.

Proposition 23. *Any terminating AC-RPS is AC-confluent.*

Proof. It suffices to check confluence of the AC-critical pairs. There are no superpositions with rules for free symbols. Let us consider an AC symbol f, a rule $f(x,y) \to r$ and a renaming $f(x',y') \to r\tau$ where $\tau = \{x \mapsto x', y \mapsto y'\}$. AC-unification of $f(x,y)$ and $f(x',y')$ yields 7 solutions, only two of them have to be considered since the others are reducible (unblocked criterion [5]): $\sigma_1 = \{x \mapsto x', y \mapsto y'\}$ and $\sigma_2 = \{x \mapsto y', y \mapsto x'\}$. Then $r\sigma_1 =_{AC} r\tau\sigma_1$ trivially and $r\sigma_2 =_{AC} r\tau\sigma_2$ by the hypothesis.

The following proposition is a reformulation of Theorem 5.3.3 of [1] in terms of weak reduction orderings, and generalized to the AC case.

Proposition 24. *Let P be a terminating AC-RPS and $\succeq$ be any AC-compatible weak reduction ordering. The relation $\succeq_P$ given by $s \succeq_P t$ iff $s{\downarrow_P} \succeq t{\downarrow_P}$ is an AC-compatible weak reduction ordering.*

Proof. It is obviously a well-founded quasi-ordering. In order to prove stability and monotonicity we need a lemma:

Lemma 25. *Let P be an AC-RPS. For each term t and each substitution σ: $(t\sigma){\downarrow_P} = (t{\downarrow_P})(\sigma{\downarrow_P})$.*

Proof. We have $(t\sigma){\downarrow_P} = (t{\downarrow_P})(\sigma{\downarrow_P}){\downarrow_P}$ thus we just have to prove $(t{\downarrow_P})(\sigma{\downarrow_P})$ is a normal form. Since the AC-symbol arity on left-hand sides is always 2, $(t{\downarrow_P})$ cannot contain any defined symbol, similarly with $x(\sigma{\downarrow_P})$ for all x. Hence $(t{\downarrow_P})(\sigma{\downarrow_P})$ contains no defined symbol so cannot be reduced, thus it is a normal form.

For stability now, if $s \succeq_P t$ then $s{\downarrow_P} \succeq t{\downarrow_P}$ hence $(s{\downarrow_P})(\sigma{\downarrow_P}) \succeq (t{\downarrow_P})(\sigma{\downarrow_P})$ since $\succeq$ is a reduction ordering. That is, by Lemma 25, $(s\sigma){\downarrow_P} \succeq (t\sigma){\downarrow_P}$ thus $s\sigma \succeq_P t\sigma$. Same process for strict part.

For monotonicity, if $s \succeq_P t$ then $s{\downarrow_P} \succeq t{\downarrow_P}$ hence $f(...,x,...){\downarrow_P}\{x \mapsto s{\downarrow_P}\} \succeq f(...,x,...){\downarrow_P}\{x \mapsto t{\downarrow_P}\}$ since $\succeq$ is a reduction ordering. By Lemma 25 we get $(f(...,x,...)\{x \mapsto s\}){\downarrow_P} \succeq (f(...,x,...)\{x \mapsto t\}){\downarrow_P}$ i.e. $f(...,s,...) \succeq_P f(...,t,...)$.

We now prove AC-compatibility. Let us suppose that $s \succeq_P t$ with $s =_{AC} s'$ and $t =_{AC} t'$. Since P is AC-confluent, we get $s{\downarrow_P} =_{AC} s'{\downarrow_P}$ and $t{\downarrow_P} =_{AC} t'{\downarrow_P}$, but $\succeq$ is AC-compatible then $s'{\downarrow_P} \succeq t'{\downarrow_P}$ that is $s' \succeq_P t'$.

Counter-example. If we allow RPS rules for AC symbols having more than 2 variables in the lhs, Proposition 24 is wrong: with the RPS $\{x+y+z \to a\}$ we would have $a+a \succeq_P a$ but when putting another a in the context we would get $a+a+a \to_P a \preceq a+a$, a contradiction to monotonicity.

Example. Let us consider the following system for computing intersection of multisets of numbers. Multisets are presented by the constant $\emptyset$, the unary symbol $\{_\}$ that builds a singleton, and the union.

$$\begin{array}{ccc} \emptyset \cup x \to x & eq(0,0) \to true & x \cap \emptyset \to \emptyset \\ if(true,x,y) \to x & eq(0,s(x)) \to false & x \cap (y \cup z) \to (x \cap y) \cup (x \cap z) \\ if(false,x,y) \to y & eq(s(x),s(y)) \to eq(x,y) & \{x\} \cap \{y\} \to if(eq(x,y),\{x\},\emptyset) \end{array}$$

where $\cup$ and $\cap$ are AC and eq is commutative. To prove termination of this system, the idea is to use MAPO with precedence $\cap > \cup$, $eq > false$ and $eq > true$ which will do the trick for all rules except the last one: we would like $\cap > if$ but it is not allowed in MAPO. The idea is then to say that in some sense, the first argument of if is not important, but only that $\{x\}$ and $\emptyset$ are smaller than the left-hand side. This is made evident when computing the dependency graph of this system, in which the only problem is to make the pair $\langle z \check{\cap} \{x\} \check{\cap} \{y\}, z \check{\cap} if(eq(x,y),\{x\},\emptyset)\rangle$ decrease. This can be done by the RPS $\{if(x,y,z) \to y \cap z, x \check{\cap} y \to x \cap y, x \check{\cup} y \to x \cup y\}$ which satisfies the AC-RPS conditions. Note that without the graph improvement, the basic criterion can also be used if we add in the RPS $\check{if}(x,y,z) \to y \cap z$, $\check{eq}(x,y) \to x \cap y$.

6 Dependency pairs criterion and theorem proving

An important question is how the dependency pairs criterion can be used in theorem proving. We explain in this section how it can be used in Knuth-Bendix completion or AC-completion, in which in the inference rule

$$\textbf{(Orient)}\ E \cup \{u = v\}, R \vdash E, R \cup \{u \to v\} \text{ if } u > v$$

we would like to replace $u > v$ by $u \geq v$ and $s > t$ for each $\langle s,t\rangle$ dependency pair of rule $u \to v$. For this to be correct, we have to make sure that the set of constructors or defined symbols does not change during completion, so we will assume that the set of constructors is fixed at once by the user, and any attempt to make a new rule with a constructor at root of left-hand side will raise a failure.

The main question is whether the *completeness* of the procedure is preserved that is: is the following theorem [6, 16] still true?

Theorem 26. *Let E_0 be the initial set of equations and $\succeq$ a weak reduction ordering. Let E_i (resp. R_i) be the set of equations (resp. rules) at completion step i, $E_\infty = \bigcap_n \bigcup_{i=n}^\infty E_i$, $R^\star = \bigcup_i R_i$, $R_\infty = \bigcap_n \bigcup_{i=n}^\infty R_i$.*

If completion succeeds ($E_\infty = \emptyset$) and is fair ($CP(R_\infty) \subseteq \bigcup_i E_i$), then $s =_{E_0} t$ iff $s\downarrow_{R_\infty} =_{\mathrm{AC}} t\downarrow_{R_\infty}$.

The proof of this theorem uses in a fundamental way the strict orientation $l > r$ for each rule $l \to r \in R^\star$. Nevertheless there is a nice way of proving that completeness is preserved when using dependency pairs by reducing to the usual case: actually, if completion is successful we know that $\to_{R^\star}$ terminates by the dependency pairs criterion, hence is a reduction ordering by Proposition 6. But then we may suppose that completion was run with this ordering, and the same inferences would be computed, thus we obtain the same R_∞.

7 Conclusion and future work

The dependency pair criterion has been shown powerful for proving termination of rewrite systems, but several problems still have to be studied.

Firstly, if this criterion can be used in completion procedures, it is not clear at all how it can be used for other first-order theorem proving methods, where usually a *simplification* ordering is necessary.

Secondly, we have seen that the dependency pair criterion allows the use of a wider class of term orderings, but it means that it is getting more difficult to build systems that will try to *find* a suitable ordering, via RPS in particular. This question has been already studied by Arts [1] who proposed to try RPS of a certain form.

Thirdly, there is the question of termination of *modular* or *hierarchical* rewrite systems, which have been studied by Arts in the case of innermost strategy, and seems to be very promising since the hierarchy of rules corresponds to some hierarchy between parts of the dependency graphs. It could be interesting also to study *Normalized* rewriting [22] which is a special case of hierarchical rewriting.

Acknowledgments

We would like to thank Keiichirou Kusakari for his useful comments about a preliminary version of this paper.

References

1. T. Arts. *Automatically proving termination and innermost normalisation of term rewriting systems.* PhD thesis, Universiteit Utrecht, 1997.
2. 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.
3. L. Bachmair. Associative-commutative reduction orderings. *Info Proc. Letters*, 1992.
4. L. Bachmair and N. Dershowitz. Commutation, transformation, and termination. In J. H. Siekmann, editor, *Proc. 8th Int. Conf. on Automated Deduction, Oxford, England, LNCS 230*, pages 5–20, July 1986.
5. L. Bachmair and N. Dershowitz. Critical pair criteria for completion. *Journal of Symbolic Computation*, 6(1):1–18, 1988.
6. L. Bachmair, N. Dershowitz, and J. Hsiang. Orderings for equational proofs. In *Proc. 1st IEEE Symp. Logic in Computer Science, Cambridge, Mass.*, pages 346–357, June 1986.
7. L. Bachmair and D. A. Plaisted. Termination orderings for associative-commutative rewriting systems. *Journal of Symbolic Computation*, 1(4):329–349, Dec. 1985.

8. A. Ben Cherifa and P. Lescanne. An actual implementation of a procedure that mechanically proves termination of rewriting systems based on inequalities between polynomial interpretations. In *Proc. 8th Int. Conf. on Automated Deduction, Oxford, England, LNCS 230*, pages 42–51. Springer-Verlag, July 1986.
9. 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. E. Contejean and C. Marché. CiME: Completion Modulo E. In H. Ganzinger, editor, *7th International Conference on Rewriting Techniques and Applications*, volume 1103 of *Lecture Notes in Computer Science*, pages 416–419, New Brunswick, NJ, USA, July 1996. Springer-Verlag. System Description available at `http://www.lri.fr/~demons/cime.html`.
11. E. Contejean, C. Marché, and L. Rabehasaina. Rewrite systems for natural, integral, and rational arithmetic. In Comon [9].
12. C. Delor and L. Puel. Extension of the associative path ordering to a chain of associative-commutative symbols. In *Proc. 5th Rewriting Techniques and Applications, Montréal, LNCS 690*, pages 389–404, 1993.
13. N. Dershowitz. Orderings for term rewriting systems. *Theoretical Computer Science*, 17(3):279–301, Mar. 1982.
14. N. Dershowitz. Termination of rewriting. *Journal of Symbolic Computation*, 3(1):69–115, Feb. 1987.
15. J. Giesl. Generating polynomial orderings for termination proofs. In J. Hsiang, editor, *6th International Conference on Rewriting Techniques and Applications*, volume 914 of *Lecture Notes in Computer Science*, Kaiserslautern, Germany, Apr. 1995. Springer-Verlag.
16. G. Huet. A complete proof of correctness of the Knuth-Bendix completion algorithm. *J. Comput. Syst. Sci.*, 23:11–21, 1981.
17. 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.
18. D. Kapur and G. Sivakumar. A total, ground path ordering for proving termination of AC-rewrite systems. In Comon [9].
19. D. E. Knuth. *The art of computer programming*, volume 2. Addison-Wesley, 2nd edition, 1981.
20. P. Lescanne. Uniform termination of term rewriting systems with status. In *Proc. 9th CAAP*, Cambridge, 1984. Cambridge University Press.
21. C. Marché. *Réécriture modulo une théorie présentée par un système convergent et décidabilité des problèmes du mot dans certaines classes de théories équationnelles.* Thèse de doctorat, Université Paris-Sud, Orsay, France, Oct. 1993.
22. C. Marché. Normalized rewriting: an alternative to rewriting modulo a set of equations. *Journal of Symbolic Computation*, 21(3):253–288, 1996.
23. R. Nieuwenhuis and A. Rubio. A precedence-based total AC-compatible ordering. In C. Kirchner, editor, *Proc. 5th Rewriting Techniques and Applications, Montréal, LNCS 690*. Springer-Verlag, June 1993.
24. G. E. Peterson and M. E. Stickel. Complete sets of reductions for some equational theories. *J. ACM*, 28(2):233–264, Apr. 1981.
25. J. Steinbach. Proving polynomials positive. In R. Shyamasundar, editor, *Foundations of Software Technology and Theoretical Computer Science*, volume 652 of *Lecture Notes in Computer Science*, pages 191–202, New Delhi, India, Dec. 1992. Springer-Verlag.
26. X. Urbain. Preuves de terminaison à l'aide de paires de dépendances. Rapport de DEA, Université Paris-Sud, Orsay, France, 1997.