Termination of Associative-Commutative Rewriting using Dependency Pairs Criteria[*]

Kusakari Keiichirou[*] Claude Marché[†]
Xavier Urbain[†]

[*] Logic for Information Science (Toyama laboratory)
Research Institute of Electrical Communication, Tohoku University
2-1-1, Katahira, Aoba-ku, Sendai, 980-8577 Japan
E-mail: `kusakari@nue.riec.tohoku.ac.jp`

[†]Laboratoire de Recherche en Informatique (LRI)
CNRS UMR 8623
Bât. 490, Université Paris-Sud, Centre d'Orsay
91405 Orsay Cedex, France
E-mail: `{marche,urbain}@lri.fr`

Abstract

In 1997, Arts and Giesl proposed new criteria for proving termination of rewriting, based on the so-called *dependency pairs*. We show how these criteria can be generalized to rewriting modulo associativity and commutativity. Proving termination by a dependency pair criterion amounts to finding a suitable *weak reduction ordering*, so we show in this paper how one may define *AC-compatible weak reduction ordering* for AC termination.

1 Introduction

For almost all applications of rewrite systems, one is interested in finiteness 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.

---

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

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 term rewriting system (TRS for short), 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 [1,2,4] 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.

Rewriting modulo AC (associativity and commutativity) is an important issue in practice: we often have symbols f that fulfil the commutativity axiom $f(x,y) = f(y,x)$. This axiom 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 fulfils 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 classes of terms modulo associativity.

The purpose of this paper is to provide a generalization of the dependency pairs methods to rewriting modulo AC. The results given here extend previous works by the same authors: Marché and Urbain in 1998 [25] proposed criteria for termination of AC rewriting on *flat terms* (see Section 2.2.3), but unfortunately their criterion ‘with marks’ appears to be wrong; Kusakari and Toyama in 1998 [21] proposed other criteria, for termination of AC class rewriting (see Section 2.2.1), but they required a strong property called ’marking condition’. In this paper we propose a new criterion with marks, that both corrects the former and extends the latter.

In Section 2, we first define the different kinds of AC rewriting known in the litterature. Theorem 2.5, showing equivalence between flat rewriting and class rewriting, justify our choice to consider flat rewriting in the rest of the paper.

Applying depencency pair criteria amounts to finding *weakly monotonic* reduction orderings. In the case of AC rewriting, one needs *AC-compatible* weakly monotonic reduction orderings and in Section 3, we will show how such orderings can be built.

As said above, giving a criterion with the so-called marked symbols is not a simple task, so as a first step, we show in Section 4 how the dependency pairs criterion *without marks* for standard rewriting is lifted up to flat AC rewriting (Theorem 4.6 and Corollary 4.7), which is a quite straightforward extension of the standard case.

Arts and Giesl also have shown that the dependency pairs criteria 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 5 how this improvement is lifted up to AC rewriting.

However, standard definition of dependency pairs is still too close to the structure of terms to be exploited to its full extends. In order to reach higher level of abstraction and to use criteria with marks, we introduce in Section 6 a new notion of dependency pairs set by defining it intentionally.

We actually propose in Section 6.2 a *marked DP condition* over sets that ensure any set verifying them to be a suitable sets of marked dependency pairs. Theorem 6.11 is the main result, allowing to remove marks in the so-called *depencency chains*. Sections 6.4.1 and 6.4.2 provide two different definitions of sets of pairs satisfying the marked DP conditions, hence providing two different termination criteria.

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 [12] developed in our research team.

2 Preliminaries

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

A signature $\mathcal{F}$ is a finite set of symbols with arities, $\mathcal{F}_n$ denoting the set of n-ary symbols. 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 (or 'top') symbol of a term t is denoted $\Lambda(t)$, $t|_p$ denotes the subterm of t at position p, and $C[t]_p$ denotes the term obtained by putting the term t in 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 $l \to r$. If R is a set of rules, we denote by

$$s \underset{R}{\to}^{\star} t$$

if s rewrites to t by R in 0, 1 or several steps, and

$$s \xrightarrow[R]{>p}{}^{\star} t$$

if s rewrites to t by R in 0, 1 or several steps *below* p.

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

We say that a term t is *strongly normalizable* (SN for short) w.r.t. R if there is no infinite sequence of reductions by $\rightarrow_R$ starting from t. We say that R *terminates* if all terms are SN.

2.1 AC signatures, flat terms, AC equivalence

An *AC signature* is a signature $\mathcal{F}$ where the set of binary symbols $\mathcal{F}_2$ 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. Symbols that are neither in $\mathcal{F}_{\mathrm{AC}}$ nor in $\mathcal{F}_{\mathrm{C}}$ are called *free*.

Example 2.1 *The AC signature of natural numbers in Peano notation is given by*

$$\begin{array}{rcl} \mathcal{F}_0 & = & \{0, true, false\} \\ \mathcal{F}_1 & = & \{s\} \\ \mathcal{F}_2 & = & \{+, \times, =, \geq\} \\ \mathcal{F}_{\mathrm{AC}} & = & \{+, \times\} \\ \mathcal{F}_{\mathrm{C}} & = & \{=\} \end{array}$$

We call *equality modulo AC*, and denote $=_{\mathrm{AC}}$, the congruence on $T(\mathcal{F}, X)$ generated by commutativity axioms

$$f(x, y) \quad = \quad f(y, x)$$

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 $f \in \mathcal{F}_{\mathrm{AC}}$.

2.1.1 Varyadic terms

The set of *varyadic* terms, denoted $T_{var}(\mathcal{F}, X)$ is the set of terms over $\mathcal{F}$ and X but where AC symbols may occur at any arity greater or equal to 2. It is defined inductively as follows:

- If $x \in X$ then $x \in T_{var}(\mathcal{F}, X)$;
- If $f \notin \mathcal{F}_{\mathrm{AC}}$ and $arity(f) = n$, if $t_1, \ldots, t_n \in T_{var}(\mathcal{F}, X)$, then $f(t_1, \ldots, t_n) \in T_{var}(\mathcal{F}, X)$;
- If $f \in \mathcal{F}_{\mathrm{AC}}$ and $n \geq 2$, if $t_1, \ldots, t_n \in T_{var}(\mathcal{F}, X)$, then $f(t_1, \ldots, t_n) \in T_{var}(\mathcal{F}, X)$.

The $=_{\mathrm{AC}}$ congruence on $T_{var}(\mathcal{F}, X)$ is generated by commutativity axioms

$$f(x, y) \quad = \quad f(y, x)$$

for each $f \in \mathcal{F}_{\mathrm{C}} \cup \mathcal{F}_{\mathrm{AC}}$; permutation axioms

$$f(x_1, \ldots, x_n) \quad = \quad f(x_{\pi(1)}, \ldots, x_{\pi(n)})$$

for each $f \in \mathcal{F}_{\mathrm{AC}}$, $n \geq 2$, π permutation of $[1, \ldots, n]$; and associativity axioms

$$\begin{array}{l} f(x_1, \ldots, x_{i-1}, f(y_1, \ldots, y_k), x_{i+1}, \ldots, x_n) = \\ \qquad f(x_1, \ldots, x_{i-1}, y_1, \ldots, y_k, x_{i+1}, \ldots, x_n) \end{array}$$

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 to non-varyadic terms coincides with the former definition of $=_{\mathrm{AC}}$.

2.1.2 Flat terms

The set of *flat* terms is the subset of $T_{var}(\mathcal{F}, X)$ of 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* is the set of rewrite rules

$$\begin{array}{l} f(x_1, \ldots, x_{i-1}, f(y_1, \ldots, y_k), x_{i+1}, \ldots, x_n) \rightarrow \\ \qquad f(x_1, \ldots, x_{i-1}, y_1, \ldots, y_k, x_{i+1}, \ldots, x_n) \end{array}$$

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 respect to flattening rules, which is a flat term, called its *flatten form* and 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, $s =_{\mathrm{AC}} t$ iff $\overline{s} \equiv \overline{t}$.

2.2 Rewriting modulo AC

We now focus on definition of rewriting modulo associativity and commutatity. As said before, several variants exist and we compare them.

2.2.1 Class rewriting

Definition 2.2 *For terms s and t in $T_{var}(\mathcal{F}, X)$, we say that s rewrites to t by a rule $l \rightarrow r$ modulo AC, denoted by*

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

if there are s' and t' such that $s =_{\mathrm{AC}} s'$, $t =_{\mathrm{AC}} t'$ and

$$s' \xrightarrow[l \to r]{} t'.$$

The relation R/AC is called rewriting modulo AC, *or* class rewriting.

Notice that this definition involves arbitrary many AC symbols and C symbols, so what we mean by rewriting *modulo AC* is actually rewriting modulo all associativity and commutativity axioms of all non-free symbols of the signature.

2.2.2 AC-extended rewriting

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 [28] 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 2.3 *The* AC-extended *rewrite relation is defined by*

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

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

Using AC-extended rewriting may lead to pathological examples where for a rewrite system R, $\to_{\mathrm{AC}\backslash R}$ 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: 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]:

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

should imply that there exist v and w such that

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

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 rule $f(f(s, t), x) \to f(r, x)$ where x is a new variable.

2.2.3 Flat AC rewriting

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

Definition 2.4 *The* flat AC *rewrite relation [23] is defined on flat terms by*

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

if there is a substitution σ such that either

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

We call the second case extended matching.

Flat AC rewriting is extended to arbitrary varyadic terms s *and* t *by*

$$s \xrightarrow[l\to r/\overline{AC}]{} t \text{ iff } \overline{s} \xrightarrow[l\to r/\overline{AC}]{} \overline{t}$$

This definition seems not to be very different from using AC-extended rewriting with always coherent systems, but actually 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. That change in the definition was shown quite powerful for example for AC completion: the completeness proof is far much simpler [23]. As shown by this last example, the flat AC rewriting relation is in between AC-extended rewriting and class rewriting: when rewriting a given varyadic term like $a + (b + c)$, it is allowed to have AC-steps at the rewriting position, or just above it as soon as the symbol above is the same AC-symbol as at the rewriting position. This idea is crucial for proving Theorem 2.5 below.

The termination criterion described in next section is based on detecting at which position a term can be rewritten indefinitely, so the notion of position discussed above is really important. Notice that 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 2.5 *Termination of flat AC rewriting is equivalent to termination of class rewriting.*

Proof. Assume that a term s is not SN with respect to $\to_{R/\mathrm{AC}}$. 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 $\rightarrow_{R/\overline{\mathrm{AC}}}$ 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. ☐

A remark about flat AC rewriting concerning efficiency: as for extension rules *à la* Peterson-Stickel, it is not necessary to use extended matching for rules that have a particular form. A rule needs extended matching if and only if:

1. root symbol f of the left-hand side l is AC and
2. there is no variable x in l such that
   (a) x occurs only once in l, and at depth 1;
   (b) the right-hand side is either x or of the form $f(x, r)$ where x does not occur in r.

Proposition 2.6 *Let*

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

be a rewrite step that uses extended matching, such that rule $l \rightarrow r$ does not fulfil the requirement above. Then there is another substitution σ' such that

$$s \xrightarrow[l \rightarrow r/\mathrm{AC}]{p,\sigma'} t$$

without extended matching.

Proof. If

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

uses extended matching, it means that the top symbol of l, say $+$, is AC, and that $s|_p = (l + z)\sigma$ and $t = s[(r + z)\sigma]_p$ where z is a new variable.

But if $l \rightarrow r$ does not fulfil the requirements for using extended-matching, we know that there is a variable x such that $l = l' + x$ and either $r = x$ or $r = r' + x$ where x does not occur in l' or r'. Let us define σ' by $x\sigma' = x\sigma + z\sigma$ and $y\sigma' = y\sigma$ for y variable other than x and z. Then we have

$$s|_p = (l + z)\sigma = (l' + x + z)\sigma = (l' + x)\sigma' = l\sigma'$$

and either

$$t = s[(r + z)\sigma]_p = s[(x + z)\sigma]_p = s[x\sigma']_p = s[r\sigma']_p$$

if $r = x$, or

$$t = s[(r + z)\sigma]_p = s[(r' + x + z)\sigma]_p = s[(r' + x)\sigma']_p = s[r\sigma']_p$$

if $r = r' + x$. $\square$

Examples of rules that need not extended matching, although their left-hand side have AC symbol $+$ at top, are $x + 0 \to x$ or $x + (-x) + y \to y$.

On the contrary, the following rules need extended matching.

- $x + (-x) \to 0$ because x occurs twice in left-hand side.
- $R = \{x \times 0 \to 0\}$ because x does not occur in right-hand side. For example, rewrite step $a \times b \times 0 \to_R a \times 0$ needs extended matching.
- $R = \{x + s(y) \to s(x + y)\}$ because x does not occur at depth 1 in right-hand side. For instance, rewrite step $0 + s(0) + 0 \to_R s(0 + 0) + 0$ definitely needs extended matching.

Notice finally that the conditions on the right-hand side were not required in Peterson-Stickel's AC completion procedure, hence the last two rules above do not require extension rules. It is enough if you are concerned in confluence: on examples above, $a + b + 0$ and $a + 0$ can be reduced both to 0 with extended matching, and also $0 + s(0) + 0$ and $s(0 + 0) + 0$ can be reduced to $s(0 + 0 + 0)$. But for termination issues, it is very important to authorize extended matching on these kinds of rules.

In the rest of the paper, we always consider flat AC rewriting, so for easier reading, we abbreviate $\xrightarrow[R/\overline{\mathbf{AC}}]{}$ into $\underset{R}{\to}$.

2.3 Term orderings

For general notions about term orderings and termination, we refer to [15]. We briefly recall here what we need. With dependency pair criteria, both quasi-orderings and strict orderings are used to compare terms. We follow here the notion of *ordering pair* [10, 20] although we still call it *term ordering* for simplicity.

Definition 2.7 *A* term ordering *is a pair $(\succeq, \succ)$ of relations over $T_{var}(F, X)$ such that:*

- *$\succeq$ is a quasi-ordering, that is reflexive and transitive;*
- *$\succ$ is a strict ordering, that is irreflexive and transitive;*
- *$\succ \subseteq \succeq$ that is for all terms t_1 and t_2, if $t_1 \succ t_2$ then $t_1 \succeq t_2$;*
- *$\succ \cdot \succeq \subseteq \succ$ and $\succeq \cdot \succ \subseteq \succ$ that is for all terms t_1, t_2 and t_3, if $t_1 \succ t_2 \succeq t_3$ or $t_1 \succeq t_2 \succ t_3$ then $t_1 \succ t_3$.*

A term ordering $(\succeq, \succ)$ is said:

- well-founded *if there is no infinite strictly decreasing sequence $t_1 \succ t_2 \succ t_3 \succ \cdots$;*

- stable *if both* $\succ$ *and* $\succeq$ *are stable under substitutions, that is for all terms* t_1 *and* t_2*, for all substitution* σ*, if* $t_1 \succ t_2$ *then* $t_1\sigma \succ t_2\sigma$*, and if* $t_1 \succeq t_2$ *then* $t_1\sigma \succeq t_2\sigma$*;*

- weakly monotonic *if for all terms* t_1 *and* t_2*, for all context* $C[.]$*, if* $t_1 \succeq t_2$ *then* $C[t_1] \succeq C[t_2]$*;*

- strictly monotonic *if for all terms* t_1 *and* t_2*, for all context* $C[.]$*, if* $t_1 \succ t_2$ *then* $C[t_1] \succ C[t_2]$.

A term ordering $(\succeq, \succ)$ *is called:*

- *a* weak reduction ordering *if it is well-founded, stable and weakly monotonic;*

- *a* (strict) reduction ordering *if it is well-founded, stable and both weakly and strictly monotonic.*

A term ordering $(\succeq, \succ)$ *is* AC-compatible *if for all* s*,* s'*,* t *and* t' *such that* $s =_{\mathrm{AC}} s'$ *and* $t =_{\mathrm{AC}} t'$*,* $s \succeq t$ *implies* $s' \succeq t'$ *and* $s \succ t$ *implies* $s' \succ t'$*. Notice that this means in particular that the ordering is compatible with flattening: if* $s \succeq t$ *then* $\overline{s} \succeq \overline{t}$ *and the same for* $\succ$*.*

2.4 Standard termination criterion

The following well-known proposition (adapted to flat rewriting) gives a termination criterion for rewriting by a system R modulo AC by showing that all rules of R decrease with respect to an AC-compatible reduction ordering.

Proposition 2.8 $\to_R$ *terminates iff there is an AC-compatible strict reduction ordering such that* $l \succ r$ *for all rules* $l \to r \in R$*.*

Proof. Let us assume

$$s \xrightarrow[l \to r]{p,\sigma} t$$

with a given rule $l \to r$, at some position p and with some substitution σ.

If the rewrite step uses standard matching, then $s|_p =_{\mathrm{AC}} l\sigma$ and $t = s[r\sigma]_p$, but we know that $l \succ r$ so, by stability $l\sigma \succ r\sigma$, hence by AC compatibility $s|_p \succ t|_p$, and then $s \succ t$ by monotonicity.

If the rewrite step uses extended matching, then $s|_p =_{\mathrm{AC}} (l+z)\sigma$ for some new variable z, and $t = s[(r+z)\sigma]_p$. But $l \succ r$ implies $l + z \succ r + z$ by monotonicity so we conclude again that $s \succ t$.

In both case, $s \succ t$ and no infinite reduction may exist since $\succ$ is well-founded.

The converse implication is true since if $\to_R$ is terminating then $(\to_R^\star, \to_R^+)$ is trivially an AC-compatible strict reduction ordering. □

3 Construction of weak AC reduction orderings

As will be shown in the next sections, proving termination *via* dependency pair criteria amounts to building weak reduction orderings.

Several constructions exist and here we are going to describe first how one can build weak orderings from polynomial interpretations; then we focus on AC-compatible path orderings, which are always strictly monotonic, but also allow to build weak reduction orderings by first applying *recursive program schemes*; and we eventually mention how one can compose lexicographically weakly monotonic orderings.

3.1 Weak polynomial interpretations

For ordering defined by polynomial interpretations, we first consider standard (i.e. non varyadic) terms.

Definition 3.1 *A polynomial interpretation is given by, for each symbol f of arity n, a polynomial $[\![f]\!]$ with coefficients in $\mathbb{Z}$ and n variables. Such an interpretation defines an interpretation of any term with variables by homomorphism*

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

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

Given an integer μ, we define $\geq_\mu$ the quasi-ordering on polynomial functions given by $P \geq_\mu Q$ iff for every $\mathbf{x} \geq \mu$, $P(\mathbf{x}) \geq Q(\mathbf{x})$ ($\mathbf{x}$ abbreviates $x_1, \dots, x_n$ and $\mathbf{x} \geq \mu$ means $x_i \geq \mu$ for all i); and $>_\mu$ the strict ordering on polynomial functions given by $P >_\mu Q$ iff for every $\mathbf{x} \geq \mu$, $P(\mathbf{x}) > Q(\mathbf{x})$.

A polynomial interpretation and a value μ define a term ordering by $s \succeq t$ iff $[\![s]\!] \geq_\mu [\![t]\!]$ and $s \succ t$ iff $[\![s]\!] >_\mu [\![t]\!]$.

Notice that $\succ$ is *not* the strict part of $\succeq$, but $(\succeq, \succ)$ is a term ordering (that is an ordering pair) as defined in Section 2.3.

Proposition 3.2 *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 moreover $x_i \geq y_i$ for all i and $x_i > y_i$ for at least one i implies $f(\mathbf{x}) > f(\mathbf{y})$ then $\succ$ is monotonic.*

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

The first part of this proposition is almost straightforward and the second part is already known [9]. Note that several methods exist for checking whether a polynomial is greater than another with respect to $>_\mu$ [9]. Moreover, there are methods for automatically finding a polynomial interpretation satisfying required comparison constraints [16, 30]. These methods are of course partial, since the problem is undecidable (it requires to solve arbitrary Diophantine inequations), but they are efficient enough to find solutions.

Notice that as soon as a given interpretation generates an AC-compatible ordering, we have an ordering in varyadic terms, by interpreting varyadic terms into the interpretation of any non-varyadic term it is equivalent to.

3.2 Path orderings

We assume the reader familiar with the *Recursive Path Ordering with status* [14, 22]. 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 3.3 *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 with respect to some well-chosen rewrite systems [5, 6, 8, 13]. We will use in this paper the *modified associative path ordering* (MAPO for short) defined by Delor and Puel [13]. More elaborated AC-compatible path orderings are known [13, 18, 26, 29] but we don't need those powerful orderings in this paper, in particular the totality property is not required.

Assume given a precedence such that for each AC symbol f, the precedence of symbols less than f has 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:

$$\begin{array}{rcl@{\qquad}rcl} f(x,g(y,z)) & \to & g(f(x,y),f(x,z)) & u_i(g(x,y)) & \to & g(u_i(x),u_i(y)) \text{ for all } i \\ f(x,u_i(y)) & \to & u_i(f(x,y)) \text{ for all } i & u_i(u_j(x)) & \to & u_j(u_i(x)) \text{ for all } i > j \end{array}$$

Proposition 3.4 (Delor and Puel [13]) *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.

Due to this system, distributivity rules cannot be oriented with MAPO. This problem is easily solved using a lexicographic combination (see Section 3.3) with the standard polynomial interpretation dealing with distributivity (here with + and × as involved AC symbols):

$$\left\{ \begin{array}{lcl} [\![\times]\!](x,y) & = & xy, \\ [\![+]\!](x,y) & = & x+y+2, \\ [\![u_i]\!](x) & = & (i+1)x+1. \end{array} \right.$$

Remark that the orderings defined here are always strictly monotonic. In Section 3.4, we will see a construction aimed at making weak reduction orderings from them.

3.3 Lexicographic combination of weak reduction orderings

Lexicographic combinations of orderings is a useful way of building complex orderings from simpler ones. However one has to be a bit careful with weak monotonic orderings, has shown by the following proposition.

Proposition 3.5 *If $\succeq_1$ and $\succeq_2$ are two quasi-orderings, the lexicographic composition $\succeq = (\succeq_1, \succeq_2)$ defined by $s \succeq t$ iff $s \succ_1 t$ or $s \simeq_1 t$ and $s \succeq_2$ satisfies the following properties:*

- *if $\succeq_1$ and $\succeq_2$ are reduction orderings then $\succeq$ is a reduction ordering;*
- *if $\succeq_1$ is a strict reduction ordering and $\succeq_2$ is a weak reduction orderings then $\succeq$ is a weak reduction ordering;*
- *if $\succeq_1$ and $\succeq_2$ are AC-compatible then $\succeq$ is AC-compatible.*

Proof. Straightforward. □

But beware that it is not true in general that if $\succeq_1$ and $\succeq_2$ are weak reduction orderings then $\succeq$ is a weak reduction ordering.

3.4 Recursive program schemes

Definition 3.6 *A rewrite system R is called a* recursive program scheme *(RPS for short) if:*

- *each rule is of 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 that 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 two variables on left-hand side, see counter-example below.

Proposition 3.7 *Any terminating AC RPS is AC-confluent.*

Proof. As the system is terminating, it suffices to check confluence of 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 [7]): $\sigma_1 = \{x \mapsto x', y \mapsto y'\}$ 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 hypothesis. ☐

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

Proposition 3.8 *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 3.9 *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 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. ☐

Now, let's go back to the proof of Proposition 3.8. Let's prove stability first: 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 3.9, $(s\sigma){\downarrow_P} \succeq (t\sigma){\downarrow_P}$ thus $s\sigma \succeq_P t\sigma$. Same process for strict part.

To prove monotonicity, let's assume $s \succeq_P t$, that is $s{\downarrow_P} \succeq t{\downarrow_P}$. Term $f(...,x,...){\downarrow_P}$ is an arbitrary term containing arbitrarily many occurrences of x, and by several application of monotonicity property of $\succeq$, we get

$$f(...,x,...){\downarrow_P}\{x \mapsto s{\downarrow_P}\} \succeq f(...,x,...){\downarrow_P}\{x \mapsto t{\downarrow_P}\}$$

and then by Lemma 3.9 we have

$$(f(...,x,...)\{x \mapsto s\}){\downarrow_P} \succeq (f(...,x,...)\{x \mapsto t\}){\downarrow_P}$$

i.e. $f(...,s,...) \succeq_P f(...,t,...)$.

Finally, let us prove AC-compatibility: suppose that $s \succeq_P t$ with $s =_{AC} s'$ and $t =_{AC} t'$. Since P is AC-confluent, we get

$$s{\downarrow_P} =_{AC} s'{\downarrow_P} \text{ and } t{\downarrow_P} =_{AC} t'{\downarrow_P},$$

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 lhs, Proposition 3.8 is wrong: with RPS $\{x+y+z \rightarrow a\}$ we would have $a + a \succeq_P a$ but when putting another a in the context we would get $a + a + a \rightarrow_P a \preceq a + a$, a contradiction to monotonicity.

4 Basic dependency pairs termination criterion

We focus now on finding termination criteria for flat AC rewriting, based on dependency pairs.

4.1 The basic criterion

The key idea is to show that if a rewrite system does not terminate modulo AC, then there are infinite reductions of a particular form, involving the so-called dependency pairs.

Definition 4.1 *For a given rewrite rule $f(t_1, \dots, t_n) \rightarrow r$, its set of* dependency pairs *is the set of pairs of terms of form $\langle f(t_1, \dots, t_n), g(u_1, \dots, u_m)\rangle$ where $g(u_1, \dots, u_m)$ 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 $f(t_1, \dots, t_n, x) \rightarrow \overline{f(r, x)}$ where x is a new variable that does not occur in the rule. Note that flattening is necessary if $\Lambda(r) = f$. Further note that if the rule does not need extended matching as discussed in the previous section (Proposition 2.6), then it is not necessary to compute its extended pairs.*

Example 4.2 (Peàno arithmetic) *Addition and multiplication on natural numbers, presented by the constant* 0 *and successor fonction s, can be defined by*

$$\begin{array}{rcll} x + 0 & \rightarrow & x & (1) \\ x + s(y) & \rightarrow & s(x + y) & (2) \end{array} \qquad \begin{array}{rcll} x \times 0 & \rightarrow & 0 & (3) \\ x \times s(y) & \rightarrow & (x \times y) + x & (4) \end{array}$$

where $+$ and $\times$ are AC. The constructors are 0 *and s so the dependency pairs of this TRS are*

$$\langle x + s(y), x + y\rangle$$
$$\langle x + s(y) + z, x + y\rangle$$
$$\langle x + s(y) + z, s(x + y) + z\rangle$$

from the second rule,

$$\langle x \times 0 \times z, 0 \times z\rangle$$

from the third, and

$$\begin{gathered}\langle x \times s(y), x \times y\rangle \\ \langle x \times s(y), (x \times y) + x\rangle \\ \langle x \times s(y) \times z, x \times y\rangle \\ \langle x \times s(y) \times z, ((x \times y) + x)\rangle \\ \langle x \times s(y) \times z, ((x \times y) + x) \times z\rangle\end{gathered}$$

from the fourth.

To obtain our termination criterion, we first need a lemma (akin to Arts and Giesl's [4]) which is proved by a straightforward structural induction.

Lemma 4.3 *Let R be a non-AC-terminating rewrite system. Let t be non-SN term. Then t contains a subterm $u = f(u_1, \dots, u_n)$ such that*

1. *u is non-SN;*
2. *each u_i, $1 \leq i \leq n$, is SN;*
3. *f is a defined symbol;*
4. *if $f \in AC$, then there is a k, $2 \leq k \leq n$, such that $f(u_1, \dots, u_k)$ is non-SN whereas $f(u_1, \dots, u_{k-1})$ is SN.*

Proof. By a minimality argument, one can find a subterm u satisfying (1) and (2). But then, any infinite reduction sequence from u must have a reduction step at top, thus f is a defined symbol. In the case where f is AC, k can be find also by a minimality argument. □

In the following, we shall write $f(\mathbf{u})$ instead of $f(u_1, \dots, u_n)$ and $f(\mathbf{u}', \mathbf{u}'')$ instead of $f(u_1, \dots, u_k, u_{k+1}, \dots, u_n)$. Notice that vector $\mathbf{u}''$ is actually empty when $k = n$. We say that a vector $\mathbf{u}$ is SN whenever each component is SN.

Lemma 4.4 *Let R be a non-terminating rewrite system, and t be a non-SN term. Then there exists positions p and p', contexts C and C', a dependency pair $\langle s_1, s_2\rangle$ and a substitution σ such that*

$$t \xrightarrow[R]{>p}{}^{\star} C[s_1\sigma]_p \xrightarrow[R]{p} C[C'[s_2\sigma]_{p'}]_p$$

where $s_2\sigma$ is non-SN.

Proof. Let us suppose that t is a non-SN term. By Lemma 4.3, t can be written $C[f(\mathbf{u})]_p$ where $f(\mathbf{u})$ is non-SN and $\mathbf{u}$ is SN.

1. If f is not an AC symbol.

It follows from $\mathbf{u}$ strongly normalizable that there has to be a rewrite step at Λ in any infinite reduction of $f(\mathbf{u})$. Thus, we know that there exists an infinite reduction of t starting like:

$$C[f(\mathbf{u})]_p \xrightarrow[R]{>p}{}^{\star} C[f(\mathbf{v})]_p \xrightarrow[l \to r]{p} C[t']_p$$

where $l \to r$ is a rule of R, and t' is non-SN.

That means that there is a substitution σ such that

$$f(\mathbf{v}) \equiv \overline{l\sigma} \text{ and } t' \equiv \overline{r\sigma}.$$

Since t' is non-SN, by Lemma 4.3 again we get $t' = C'[f'(\mathbf{u}')]_{p'}$ where $f'(\mathbf{u}')$ is non-SN.

Moreover, for each variable y of l, $y\sigma$ is a subterm of $\mathbf{v}$ so is SN. That means that the symbol f' at position p' of t' cannot be inside the substitution σ. Thus, we actually get $r = C'[f'(\mathbf{w})]_{p'}$ where $\mathbf{w}\sigma = \mathbf{u}'$, hence the pair of terms $\langle l, f'(\mathbf{w})\rangle$ is a dependency pair of the rule $l \to r$.

2. If f is an AC symbol.

   By Lemma 4.3, we know there exists k such that $f(\mathbf{u}) = f(u_1, \dots, u_k, u_{k+1}, \dots, u_n) = f(\mathbf{u}', \mathbf{u}'')$ where $f(\mathbf{u}')$ is non-SN whereas $f(u_1, \dots, u_{k-1})$ is SN. As in first case, $\mathbf{u}'$ strongly normalizable implies that, in any infinite reduction of $f(\mathbf{u}')$, there must be a reduction step at Λ.

   So, there exists an infinite reduction of t starting like

$$C[f(\mathbf{u}', \mathbf{u}'')]_p \xrightarrow[R]{>p}{}^{\star} C[f(\mathbf{v}, \mathbf{u}'')]_p \xrightarrow[l \to r]{p} C[t']_p$$

   where $l \to r$ is a rule of R, and t' is non-SN.

   That means that there is a substitution σ such that either

$$f(\mathbf{v}, \mathbf{u}'') \equiv \overline{l\sigma}$$

   or, by extended matching

$$f(\mathbf{v}, \mathbf{u}'') \equiv \overline{f(l, x)\sigma}$$

   with a new variable x.

   We are going to distinguish two cases depending on whether this reduction step uses extended matching or not. But notice that since we are in fact considering an infinite reduction of $f(\mathbf{u}')$, if $\mathbf{u}''$ is not empty then this rewrite step necessarily uses extended matching.

(a) If it does not use extended matching.
As said before, $\mathbf{v}''$ is an empty vector, hence we have

$$f(\mathbf{v}) \equiv \overline{l\sigma} \xrightarrow{\Lambda} t'.$$

Since t' is non-SN, by Lemma 4.3, we may write

$$t' = C'[f'(\mathbf{w}')]_{p'}$$

where $f'(\mathbf{w}')$ is non-SN. As in first case, position p' cannot occur inside σ, so $\mathbf{w}' = \mathbf{w}\sigma$, $\langle l, f'(\mathbf{w})\rangle$ is again a dependency pair of R.

(b) If it uses extended matching.
We have

$$f(\mathbf{v}, \mathbf{u}'') \equiv \overline{f(l, x)\sigma}$$

where the rewriting is in fact a rewriting of $f(\mathbf{v})$ so $\mathbf{u}''$ is inside $x\sigma$. We may write $f(\mathbf{v}) = f(\mathbf{v}', \mathbf{v}'')$ ($\mathbf{v}''$ possibly empty) where $f(\mathbf{v}') = l\sigma$ and $f(\mathbf{v}'', \mathbf{u}'') = x\sigma$ (either $\mathbf{u}''$ or $\mathbf{v}''$ is a non-empty vector).
So, our infinite reduction of t starts like

$$C[f(\mathbf{u}', \mathbf{u}'')]_p \xrightarrow[R]{>p}{}^{\star} C[f(\mathbf{v}', \mathbf{v}'', \mathbf{u}'')]_p \xrightarrow[l \to r]{p} C[f(\mathbf{r}\sigma, \mathbf{v}'', \mathbf{u}'')]_p$$

where $\mathbf{r} = r$ if the root symbol of r is not f, or $r = f(\mathbf{r})$ otherwise.
Since $f(\mathbf{r}\sigma, \mathbf{v}'')$ is non-SN, by Lemma 4.3:

$$f(\mathbf{r}\sigma, \mathbf{v}'') = C'[f'(\mathbf{w}')]_{p'}$$

where $f'(\mathbf{w}')$ is non-SN hence p' cannot be inside σ, so $\mathbf{w}' = \mathbf{w}\sigma$.

- If $f' \neq f$ or C'' is a non-empty context, symbol f' cannot come from instantiation σ since $\mathbf{v}''$ is SN. Hence f' appears in r and $\langle f(l, x), f'(\mathbf{w})\rangle$ is a dependency pair of $l \to r$.
- If $f' = f$ and C'' is the empty context, $\langle f(l, x), f(r, x)\rangle$ is a dependency pair of $l \to r$ and we are done.

□

Definition 4.5 *An R-chain of dependency pairs is a sequence $\langle s_1, t_1\rangle, \langle s_2, t_2\rangle, ...$ of pairs of R such that for a substitution σ and for each i,*

$$t_i\sigma \xrightarrow[R]{>\Lambda}{}^{\star} s_{i+1}\sigma.$$

(We consider only one substitution for all i because we can assume by renaming that sets of variables of two different pairs are disjoint.)

Equivalently, a R-chain is a sequence of rewrite steps with R at non-root positions or with $DP(R)$ at root positions.

Theorem 4.6 *For a given rewrite system R, the relation* $\xrightarrow[R]{}$ *terminates if and only if there is no infinite* R*-chain.*

Proof.

Let us assume that R is a non-terminating AC-TRS, and let t be a non-SN term. Lemma 4.4 can be applied repeatedly to t in order to obtain an infinite reduction:

$$\cdots \xrightarrow[l_1 \to r_1]{p_1} C_2[f_2(\mathbf{u_2})]_{p_2} \xrightarrow[R]{>p_2}{}^{\star} C_2[f_2(\mathbf{v_2})]_{p_2} \xrightarrow[l_2 \to r_2]{p_2} C_3[f_3(\mathbf{u_3})]_{p_3} \cdots$$

In this reduction, remember that at each step between dependency pair steps

$$f_i(\mathbf{u_i}) \xrightarrow[R]{>\Lambda}{}^{\star} f_i(\mathbf{v_i}).$$

Reciprocally, if we have an infinite R-chain $...\langle s_i, t_i\rangle\langle s_{i+1}, t_{i+1}\rangle...$, we obtain directly from the definition of an R-chain an infinite R-reduction with

$$\ldots t_i\sigma \xrightarrow[R]{>\Lambda}{}^{\star} s_{i+1}\sigma \ldots$$

and next reduction steps inside $t_{i+1}\sigma$, subterm of $s_{i+1}\sigma$. □

4.2 Application to proving termination

An easy corollary is the following.

Corollary 4.7 *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 is terminating.*

Proof. Let $\succeq$ be such an ordering and let us suppose that R does not terminate. Then we dispose of an infinite R-chain $...\langle s_i, t_i\rangle\langle s_{i+1}, t_{i+1}\rangle...$ with a substitution σ. From $s_i \succ t_i$ and $\succeq$ stable we get $s_i\sigma \succ t_i\sigma$ and from $t_i\sigma \xrightarrow[R]{>\Lambda}{}^{\star} s_{i+1}\sigma$, $l \succeq r$ for each rule $l \to r$ and weak monotonicity of $\succeq$ we get $t_i\sigma \succeq s_{i+1}\sigma$. Then we have an infinite decreasing sequence $...s_i\sigma \succ t_i\sigma \succeq s_{i+1}\sigma \succ t_{i+1}\sigma...$ that contradicts well-foundedness of $\succeq$. □

Example 4.8 (Multiset intersection) *Let us consider the following system for computing intersection of multisets of numbers. Multisets are presented by constant* $\emptyset$*, unary symbol* $\{\_\}$ *that builds a singleton, and the union.*

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

where $\cup$ *and* $\cap$ *are AC and* eq *is commutative.*

Proving its termination amounts to checking that the rules weakly decrease and that the following dependency pairs strictly decrease:

$$\begin{array}{l}
\langle \mathit{eq}(s(x), s(y)), \mathit{eq}(x, y)\rangle,\\
\langle \{x\} \cap \{y\}, \mathit{if}(\mathit{eq}(x, y), \{x\}, \emptyset)\rangle,\\
\langle \{x\} \cap \{y\}, \mathit{eq}(x, y)\rangle,\\
\langle x \cap (y \cup z), (x \cap y) \cup (x \cap z)\rangle,\\
\langle x \cap (y \cup z), x \cap y\rangle,\\
\langle x \cap (y \cup z), x \cap z\rangle,\\
\langle y \cap x \cap \emptyset, y \cap \emptyset\rangle,\\
\langle z \cap \{x\} \cap \{y\}, \mathit{if}(\mathit{eq}(x, y), \{x\}, \emptyset)\rangle,\\
\langle z \cap \{x\} \cap \{y\}, \mathit{eq}(x, y)\rangle,\\
\langle z \cap \{x\} \cap \{y\}, z \cap \mathit{if}(\mathit{eq}(x, y), \{x\}, \emptyset)\rangle,\\
\langle t \cap x \cap (y \cup z), (x \cap y) \cup (x \cap z)\rangle,\\
\langle t \cap x \cap (y \cup z), x \cap y\rangle,\\
\langle t \cap x \cap (y \cup z), x \cap z\rangle,\\
\langle t \cap x \cap (y \cup z), t \cap ((x \cap y) \cup (x \cap z))\rangle.
\end{array}$$

The following weak polynomial interpretation suffices to prove termination:

$$\begin{array}{l}
[\![\mathit{true}]\!] = 0;\\
[\![\mathit{false}]\!] = 0;\\
[\![\mathit{if}]\!](x_0, x_1, x_2) = x_2 + x_1;\\
[\![0]\!] = 0;\\
[\![s]\!](x) = x + 1;\\
[\![\mathit{eq}]\!](x_0, x_1) = x_0 x_1;\\
[\![\emptyset]\!] = 0;\\
[\![\{\_\}]\!](x) = x;\\
[\![\cup]\!](x_0, x_1) = x_1 + x_0 + 2;\\
[\![\cap]\!](x_0, x_1) = 2x_0 x_1 + 2x_1 + 2x_0 + 1.
\end{array}$$

5 Dependency graph criterion

For standard rewriting, dependency graphs [4] 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.

5.1 The dependency graph criterion

Definition 5.1 *The* AC dependency graph *of a rewrite system R is a graph having AC-extended dependency pairs as vertices and such that 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*

$$t_1\sigma \xrightarrow[R/\overline{AC}]{>\Lambda}{}^{\star} s_2\sigma.$$

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 pairs decrease inside cycles.

Corollary 5.2 *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 $l \to r \in R$, $l \succeq r$,*
- *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.

Proof. An infinite path in finite graph $\mathcal{G}$ must cross one pair $\langle u, v \rangle$ infinitely many times. So it must follow one cycle C crossing $\langle u, v \rangle$ infinitely many times. There is a strict decreaseness each time this cycle C is followed and this contradicts well-foundedness of $\succ$. ☐

5.2 Approximated dependency graphs

Unfortunately, existence of a σ such that

$$t_1\sigma \xrightarrow[R/\overline{\mathrm{AC}}]{>\Lambda}{}^{\star} s_2\sigma$$

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 5.3 *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 such that 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 5.4 *If*

$$s \xrightarrow[R/\overline{AC}]{}{}^{\star} t$$

then $\mathrm{CAP}(s)$ and t are AC-unifyable.

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

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

Corollary 5.5 *If there is a substitution σ such that*

$$s\sigma \xrightarrow[R/\overline{AC}]{>\Lambda}{}^{\star} t\sigma$$

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 $\langle f(x,x), f(0,1)\rangle$ and $\textsc{cap}(f(0,1)) = f(0,1)$ does not unify with $f(x,x)$, 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 $\langle f(x,x,y), f(0,1,y)\rangle$ and $\textsc{cap}(f(0,1,y)) = f(0,1,y')$ AC-unifies with left-hand sides of both pairs. The dependency graph is then

$$\begin{tikzcd}
\langle f(x,x), f(0,1)\rangle \arrow[r] & \langle f(x,x,y), f(0,1,y)\rangle \arrow[loop right]
\end{tikzcd}$$

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

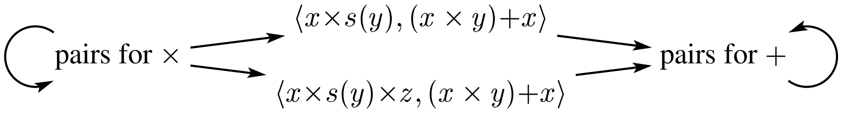
Example: Peàno arithmetic continued Let us consider again the system of Example 4.2:

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

As the dependency pairs are

$$\begin{array}{l}
\left.\begin{array}{l}
\langle x + s(y), x + y\rangle, \\
\langle z + x + s(y), x + y\rangle, \\
\langle z + x + s(y), z + s(x + y)\rangle,
\end{array}\right\} \text{pairs for } + \\
\left.\begin{array}{l}
\langle x \times s(y), x \times y\rangle, \\
\langle y \times x \times 0, y \times 0\rangle, \\
\langle z \times x \times s(y), x \times y\rangle, \\
\langle z \times x \times s(y), z \times ((x \times y) + x)\rangle,
\end{array}\right\} \text{pairs for } \times \\
\left.\begin{array}{l}
\langle x \times s(y), (x \times y) + x\rangle, \\
\langle z \times x \times s(y), (x \times y) + x\rangle.
\end{array}\right\} \text{‘mixed’ pairs}
\end{array}$$

the shape of the dependency graph is like:



Thus, there is no need for any orientation of $\langle x \times s(y), (x \times y) + x\rangle$ and $\langle z \times x \times s(y), (x \times y) + x\rangle$.

6 Marked AC dependency pairs criteria

In this section, we propose more powerful criteria based on a marking of some symbols, extending the use of capital symbols in the standard dependency pair criterion. The extension to AC of the use of such special symbols is not straightforward at all; in particular, in the case of AC rewriting, to a non-marked dependency pair may correspond several marked pairs.

We are going to proceed in two steps: firstly we will to give an abstract *dependency pair condition*, that is a set of conditions that marked dependency pairs must satisfy in order to have a sound termination criterion; and secondly, we will define a way of computing such marked pairs, we will actually give two different variants.

6.1 Marked symbols and marked terms

As before, let us assume given a rewrite system R for which we want to prove its termination. We first define the set of marked symbols, and the operation of marking a term at top.

Definition 6.1 *Let $\widehat{\mathcal{F}} = \mathcal{F} \cup \{\widehat{f}, f$ defined symbol$\}$ an extended signature where each defined symbol is copied with a mark. For a flat term s over signature $\mathcal{F}$, whose root symbol is a defined one, we denote $\widehat{s}$ the term over $\widehat{\mathcal{F}}$ obtained by replacing the root symbol with its marked copy.*

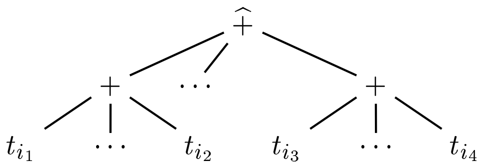
The standard dependency pairs criterion with marks considers terms where the top symbol is the only one marked. In the following, in order to deal with terms headed by AC symbols, we will have to consider also *partially marked* terms. Those are terms whose top symbol $\widehat{f}$ is marked, and possibly with non-marked symbols f below $\widehat{f}$ (even if f is AC).

Definition 6.2 *Let t be a term with top symbol f. A term t′ is said* partially marked *from t if*

- $\Lambda(t') = \widehat{f}$*;*
- *the only marked symbol in* t' *is* $\widehat{f}$*;*
- *if* f *is not AC, then the only occurrence of* $\widehat{f}$ *is at top;*
- *an occurrence of* $\widehat{f}$ *in* t' *never occurs below a non-marked symbol;*
- *when replacing each occurrence of* $\widehat{f}$ *by* f *in* t'*, we get a term AC-equivalent to* t*.*

$Par(t)$ *denotes the set of partial markings of t.*

To explain this more graphically, for a given AC symbol $+$, $Par(t_1 + \cdots + t_n)$ is the set of terms of the form



for any permutation of the t_i. In particular $\widehat{t} \in Par(t)$.

Example 6.3 (Peàno arithmetic continued) *The constructors are* 0 *and* s *so* $\widehat{\mathcal{F}} = \{0, s, +, \times, \widehat{+}, \widehat{\times}\}$. *We have for example*

$$\begin{aligned}
\widehat{0+s(0)} &= 0\widehat{+}s(0)\\
Par(0+s(0)) &= \{0\widehat{+}s(0)\}\\
0+s(\widehat{0)+s}(s(0)) &= 0\widehat{+}s(0)\widehat{+}s(s(0))\\
Par(0+s(0)+s(s(0))) &= \{0\widehat{+}s(0)\widehat{+}s(s(0)),\\
&\quad (0+s(0))\widehat{+}s(s(0)),\\
&\quad 0\widehat{+}(s(0)+s(s(0))),\\
&\quad s(0)\widehat{+}(0+s(s(0)))\}
\end{aligned}$$

Definition 6.4 *Let R be a TRS, we denote $\widehat{R}$ the following 'mark erasing' system:*

$$\{\widehat{f}(\mathbf{u}) \to f(\mathbf{u}) \ \mid f \text{ defined symbol in } R\}.$$

and we denote by $\xrightarrow{m.e.}$ the rewrite relation by $\widehat{R}$ modulo AC, restricted to pair of terms that are both partially marked from a given term.

We denote $\mathsf{Par}_{\downarrow}(t)$ the set of $u \in \mathsf{Par}(t)$ that are in $\xrightarrow{m.e.}$-normal form.

Notice that in particular, $\xrightarrow{m.e.}$ cannot erase the mark at top.

Example 6.5 *If $+$ is a defined AC symbol, the term $0\widehat{+}s(0)\widehat{+}s(s(0))$ can be rewritten by $\xrightarrow{m.e.}$ into any of the terms $(0+s(0))\widehat{+}s(s(0))$, $0\widehat{+}(s(0)+s(s(0)))$, or $s(0)\widehat{+}(0+s(s(0)))$, those three being in normal form.*

Proposition 6.6 *For any term t, for all $t' \in \mathsf{Par}(t)$,*

$$\widehat{t} \xrightarrow{m.e.}{}^{\star} t'.$$

Proof. Trivial. □

In particular, $\mathit{Par}_{\downarrow}(t)$ is the set of normal forms of $\widehat{t}$ for $\xrightarrow{m.e.}$.

Definition 6.7 *We define the* head *of a term t as*

$$\mathit{Head}(t) = \left\{ p \in \mathcal{P}os(t) \text{ such that } \ \forall q \prec p, \Lambda(t_q) = \Lambda(t) \text{ or } \widehat{\Lambda(t)} \ \right\}$$

where $\prec$ is the prefix ordering. Conversely we define the body *of t as $\mathcal{P}os(t) \setminus \mathit{Head}(t)$.*

Reduction steps are said inner *or* outer *steps depending on whether they occur in the body or not. We shall write $s \xrightarrow{i} t$ (resp. $s \xrightarrow{o} t$) if that reduction step is an inner (resp. outer) step.*

Example 6.8 *R: $\{g(x) \to x\}$, $s = a + g(a+a)$ then $\mathit{Head}(s) = \{\Lambda\}$, $s \to t = a + (a+a)$ and $\mathit{Head}(t) = \{\Lambda, 2\}$.*

6.2 Marked dependency pair condition

We define now the so-called *DP conditions.*

Definition 6.9 *Let be given an unmarked pair $\langle u, v\rangle$ of R. A set of pairs $\{\dots, \langle u_i, v_i\rangle, \dots\}$ is a* set of marked dependency pairs *for $\langle u, v\rangle$ if each of the following* DP conditions *holds:*

1. *for all i, there is a substitution σ_i such that $\widehat{v\sigma_i} \xrightarrow{m.e.}{}^{\star} v_i$*

$$\begin{tikzcd}
{} \arrow[r, dashed] & s \arrow[r, leftrightarrow, "AC"'] & u\sigma \arrow[rr, "\langle u,v\rangle \in DP(R)"] & & v\sigma \equiv t \arrow[r, dashed] & {} \\
& & \widehat{u\sigma} \arrow[dl, "\star"', "m.e." near end] & \widehat{t} \arrow[dr, "\star", "m.e."'] & & \\
{} \arrow[r, dashed] & s' \arrow[dr, dotted, "\star", "m.e."'] & & & v_i\sigma' \arrow[r, leftrightarrow, "AC"'] & t' \arrow[r, dashed] & {} \\
& & u_i\sigma' \arrow[urr, bend right, "\langle u_i,v_i\rangle \in \widehat{DP}"'] & & &
\end{tikzcd}$$

Figure 1: Unmarked chain and corresponding marked chain.

2. *for all substitution σ, for all $t \in \mathrm{Par}(u\sigma)$, there exists an index i and a substitution σ' such that*

$$\sigma =_{AC} \sigma_i\sigma'$$
$$t \xrightarrow{m.e.}{}^{\star} u_i\sigma'$$

A set of marked dependency pairs *for R is a union of sets of marked dependency pairs for all unmarked pairs of R.*

We assume now given a set of marked dependency pairs of R. We can define a *marked dependency chain* as follows.

Definition 6.10 *A* marked R-chain *of dependency pairs is a sequence $\langle s_1, t_1\rangle$, $\langle s_2, t_2\rangle$,... of marked pairs of R such that for a substitution σ and for each i,*

$$t_i\sigma(\xrightarrow[R]{i})^\star \cdot \xrightarrow{m.e.}{}^{\star} s_{i+1}\sigma$$

Theorem 6.11 *Let R be a TRS, $\widehat{DP}$ a set of marked pairs, assume that $\widehat{DP}$ satisfies the DP conditions. If there is an (non-marked) infinite chain then an infinite marked chain exists.*

Proof. We build the marked dependency chain from the unmarked one. In order to do this, we need two intermediate lemmas.

Lemma 6.12 *If $t(\xrightarrow{i})^\star s$ then for all $t' \in \mathrm{Par}(t)$ there is a $s' \in \mathrm{Par}(s)$ such that $t'(\xrightarrow{i})^\star s'$.*

Proof. Hypothesis: $t \xrightarrow{i\ n} s \implies \forall t' \exists s'$ s.t. $t' \xrightarrow{i\ \star} s'$.
Induction on n:

- Base: $t \sim_{AC} s$ we can choose for s' any member of the t' AC equivalence class.
- General case: $t \xrightarrow{i\ n+1} s$ can be split up in $t \xrightarrow{i\ n} u \xrightarrow{i} s$. Since steps are inner steps we may consider without loss of generality that $\mathit{Head}(s) = \Lambda$. Hence let u be $f(u_1, \ldots, u_k)$ and s be $f(s_1, \ldots, s_k)$, $\exists i \in \{1, \ldots, k\}$ such that $u_i \rightarrow s_i$ thus $\forall u'\ \exists s' \in \mathit{Par}(s)$ such that $u' \xrightarrow{i} s'$.

  By induction hypothesis $\forall t' \in \mathit{Par}(t)$, $\exists u' \in \mathit{Par}(u)$ such that $t' \xrightarrow{i\ \star} u'$ hence we get $t' \xrightarrow{i\ \star} u' \xrightarrow{i} s'$ that is $t' \xrightarrow{i\ \star} s'$.

$\square$

Lemma 6.13 *if* $s \xrightarrow[\langle u,v\rangle]{DP} t$ *then for all* $s' \in \mathit{Par}(s)$*, there is a pair* $\langle u_i, v_i\rangle \in \widehat{DP}$*,* $s'' \in \mathit{Par}(s)$ *and* $t' \in \mathit{Par}(t)$ *such that*

$$s' \xrightarrow{m.e.}{}^{\star} s'' \xrightarrow[\langle u_i,v_i\rangle]{\widehat{DP}} t'.$$

Moreover, if $s = u\sigma$ *then* $t' = \widehat{v\sigma}$*.*

Proof. Given s and t, we are going to build suitable s' and t', the construction being shown Figure 1.

If $s \xrightarrow[\langle u,v\rangle]{DP} t$ then there exists σ such that $s \sim_{AC} u\sigma$ and $t \equiv v\sigma$. Let us then consider $\widehat{(u\sigma)}$:

- $\widehat{(u\sigma)} \xrightarrow{m.e.}{}^{\star} \widehat{u}\sigma$, since $\mathit{Head}(u) \subseteq \mathit{Head}(u\sigma)$;
- $\widehat{(u\sigma)} \xrightarrow{m.e.}{}^{\star} s'$, since $s \sim_{AC} u\sigma \implies \widehat{(u\sigma)} \sim_{AC} \widehat{s} \xrightarrow[\widehat{R}]{\star} s'$;

hence $s' \in \mathit{Par}(u\sigma)$ and by condition 2 of Definition 6.9 we get an index i and a substitution σ' such that

$$\sigma =_{AC} \sigma_i\sigma' \text{ and } s \xrightarrow{m.e.}{}^{\star} u_i\sigma'.$$

Thus, if we choose $s'' = u_i\sigma'$, we get

$$s' \xrightarrow{m.e.}{}^{\star} s'' \xrightarrow[\langle u_i,v_i\rangle]{\widehat{DP}} v_i\sigma'$$

If we choose $t' = v_i\sigma'$, we are left to prove that $t' \in \mathit{Par}(t)$. Condition 1 of Definition 6.9 tells us that $\widehat{v\sigma_i} \xrightarrow{m.e.}{}^{\star} v_i$, that is, since $t = v\sigma$ and $\sigma =_{AC} \sigma_i\sigma'$, $\widehat{t} \xrightarrow{m.e.}{}^{\star} v_i\sigma' = t'$.

□

We are now ready to prove Theorem 6.11.

Let $\langle s_1, t_1\rangle, \langle s_2, t_2\rangle$,... be an infinite dependency chain without marks, together with a substitution σ such that for each i,

$$t_i\sigma \xrightarrow{\neq\Lambda} s_{i+1}\sigma.$$

We build an infinite marked chain inductively, by translating each step of the former chain.

Starting from any $s'_1 \in \mathit{Par}(s_1\sigma)$, we apply Lemma 6.13 to get a marked pair $\langle s_{1_i}, t_{1_i}\rangle$ such that

$$s'_1 \xrightarrow{m.e.}{}^{\star} s''_1 \xrightarrow[\langle s_{1_i}, t_{1_i}\rangle]{\widehat{DP}} t'_1$$

for a $t'_1 \in \mathit{Par}(t_1\sigma)$. Then, we apply Lemma 6.12 to t'_1 to get a s'_2 such that

$$t'_1 \xrightarrow{i}{}^{\star} s'_2$$

and $s'_2 \in \mathit{Par}(s_2\sigma)$. We deal similarly with each step, by straightforward induction. □

6.3 Marked dependency pairs criteria

The following corollary gives a termination criterion using marked pairs. Because of marks erasing steps, we require that mark erasing rules weakly decrease with respect to the ordering.

Corollary 6.14 *If there is an AC-compatible weak reduction ordering $\succeq$ such that $l \succeq r$ for each rule of R, $s \succ t$ for each marked dependency pair $\langle s, t\rangle$ of R, and for each defined AC symbol f, $\widehat{f}(x, y) \succeq f(x, y)$ then R is AC-terminating.*

Proof. Similar to Corollary 4.7. The third condition is needed to insure weak decreasingness of mark erasing steps. □

Is easy to adapt the dependency graph technique to pairs with marks. We get the following criterion.

Corollary 6.15 *Assume $\widehat{DP}(R)$ is a set of marked dependency pairs for R, and G is the (approximated) dependency graph of those pairs. If there is an AC-compatible weak reduction ordering $\succeq$ such that*

- *$l \succeq r$ for each rule of R ;*
- *for each defined AC symbol f, $\widehat{f}(x, y) \succeq f(x, y)$;*
- *$u \succeq v$ for each pair $\langle u, v\rangle$ of $\widehat{DP}(R)$ that belongs to a cycle of G ;*
- *for each cycle C of G there is at least one pair $\langle u, v\rangle$ of C such that $u \succ v$;*

then R is AC-terminating.

6.4 Computing marked dependency pairs

All we need after Theorem 6.11 is some way to compute an acceptable set $\widehat{DP}$ for any system R. We give now two possible variants, we will compare them afterward.

Note that simply marking heads of unmarked 'flat' dependency pairs would *not* result in a set of dependency pairs and Example 6.16 clearly shows how that would lead to erroneous conclusions.

Example 6.16 *let R be the following TRS:*

$$\begin{cases} f(x) & \to & x + a, \\ c + a & \to & f(b + c). \end{cases}$$

This system is not terminating because of the infinite reduction

$$f(b + c) \to b + c + a \to b + f(b + c) \to \cdots$$

and its AC dependency pairs are

$$\begin{array}{ll} \langle f(x), x + a\rangle & \langle c + a + x, f(b + c) + x\rangle \\ \langle c + a, f(b + c)\rangle & \langle c + a + x, f(b + c)\rangle \\ \langle c + a, b + c\rangle & \langle c + a + x, b + c\rangle. \end{array}$$

We indeed have the infinite chain:

$$\langle f(b + c), b + c + a\rangle, \langle c + a + b, f(b + c)\rangle, \langle f(b + c), b + c + a\rangle, \ldots$$

But marking now the whole head of pairs lhs, we obtain a set of pairs:

$$\begin{array}{ll} \langle \widehat{f}(x), x \widehat{+} a\rangle & \langle c \widehat{+} a \widehat{+} x, f(b + c) \widehat{+} x\rangle \\ \langle c \widehat{+} a, \widehat{f}(b + c)\rangle & \langle c \widehat{+} a \widehat{+} x, \widehat{f}(b + c)\rangle \\ \langle c \widehat{+} a, b \widehat{+} c\rangle & \langle c \widehat{+} a \widehat{+} x, b \widehat{+} c\rangle \end{array}$$

for which the corresponding chain is

$$\langle \widehat{f}(b + c), (b + c) \widehat{+} a\rangle$$

thus finite since no pair lhs can match $(b + c) \widehat{+} a$.

Definition 6.17 *Let $t = u_1 + \ldots + u_n$ be a non-marked, flattened term where $+$ is AC. We define:*

$$\begin{array}{rcl} I_t & = & \{i \in 1, \ldots, n \mid u_i \text{ is a variable}\}, \\ X_t & = & \{u_i \mid i \in I\} \text{ and} \\ m_t(x) & = & \#\{i \in I \mid u_i = x\}. \end{array}$$

We will refer to $m_t(x)$ as the multiplicity *of x in t.*

Example 6.18 *For term* $t = x + x + f(y) + z + f(z)$*, we get* $I_t = \{1, 2, 4\}$*,* $X_t = \{x, z\}$*,* $m_t(x) = 2$ *and* $m_t(z) = 1$*.*

The key point in an unmarked dependency R-chains step is a term (say t) being an instantiation *modulo AC* of a dependency pair (say $\langle u, v\rangle$) left-hand side, that is $t =_{AC} u\sigma$; AC steps may occur anywhere.

Similarly, in order to mimic unmarked chain steps with marked ones, we have to rewrite a *partially marked term* (say $t' \in \mathit{Par}(t)$), by $\xrightarrow{m.e.}$, to an instantiation modulo AC of a marked DP lhs. But since we introduced marks, it became impossible for any marked AC symbol, say for example $\widehat{+}$, to distribute over its unmarked version $+$, thus considerably constraining AC steps.

Hence we may encounter problems if what should be an instance of a variable occurring in a marked pair is actually shared out by both sides of the marked symbol (sharing that would be easily solved in an unmarked term thanks to associativity and commutativity). See for instance Example 6.19.

Example 6.19 *Let* $\{x + x + g(y) \to h(x, y)\} \in R$ *with* g *defined and* $+ \in AC$*. Now let us have a look at* $t = (a_1 + a_1 + a_2 + a_4)\widehat{+}(a_2 + a_3 + a_3 + a_4 + g(y))$*. If* t *is clearly not an instantiation of any marked version of* $x + x + g(y)$ *it nevertheless starts a reduction and thus a chain. Hence we have to deal with it.*

In order to overcome that technical difficulty, we introduce a function *Split*, providing all suitable splitting of variables according to their multiplicity. Correctness of this function will be obtained while proving Proposition 6.23.

Definition 6.20 *Function Split is defined from variables to the substitutions power set such that:*

$$\mathit{Split}_t(x) = \bigcup_{i=1}^{m_t(x)} \{\sigma \mid \sigma(x) = x_0 + \ldots + x_i\}$$

where $+$ *is the symbol at* Λ *in t, and the* x_j *are fresh variables.*

We denote Σ_t *the set of all possible compositions of all* σ *in* $\mathit{Split}_t(x)$ *for all variables* $x \in X_t$*, that is:*

$$\Sigma_t = \bigcup_{\substack{\sigma_1 \in \mathit{Split}_t(x_1) \\ \vdots \\ \sigma_k \in \mathit{Split}_t(x_k) \\ \text{where } \{x_1, \ldots, x_k\} = X_t}} \sigma_1 \circ \cdots \circ \sigma_k$$

Example 6.21 *For term* $t = x + x + f(y) + z + f(z)$*, we get*

$$\mathit{Split}_t(x) = \{x \mapsto x_0, x \mapsto x_0 + x_1, x \mapsto x_0 + x_1 + x_2\}$$

and

$$\begin{array}{rcl}\Sigma_t & = & \left\{\begin{array}{l} x \mapsto x_0 \\ z \mapsto z_0 \end{array}\right., \left\{\begin{array}{l} x \mapsto x_0 + x_1 \\ z \mapsto z_0 \end{array}\right., \left\{\begin{array}{l} x \mapsto x_0 + x_1 + x_2 \\ z \mapsto z_0 \end{array}\right., \\ & & \left\{\begin{array}{l} x \mapsto x_0 \\ z \mapsto z_0 + z_1 \end{array}\right., \left\{\begin{array}{l} x \mapsto x_0 + x_1 \\ z \mapsto z_0 + z_1 \end{array}\right., \left\{\begin{array}{l} x \mapsto x_0 + x_1 + x_2 \\ z \mapsto z_0 + z_1 \end{array}\right. \end{array}$$

Building a set of marked dependency pairs now consists mainly in:

- Firstly, computing Σ which is quite easy since the substitutions domains are disjoint.
- Then marking on top all members of the AC equivalence class of instantiations of unmarked pairs left-hand sides by each $\sigma \in \Sigma$; by picking up one member of each AC class, we have got the left-hand sides of marked pairs.
- Eventually, building the corresponding right-hand sides and here is where we propose two variants; in the first one we put marks on the whole head of instantiations of unmarked pairs rhs, while in the second one only one occurrence of the top symbol is marked and we have to distinguish all subterm sharings by both sides of the marked top position.

6.4.1 First variant

Definition 6.22 *Let R be a term rewriting system, we define the function* $\mathtt{Pairs}$ *as:*

$$\mathtt{Pairs}(R) = \bigcup_{\substack{\langle u, v\rangle \in DP(R), \\ \sigma \in \Sigma_u}} \{\langle u', \widehat{(v\sigma)}\rangle \text{ such that } u' \in \mathit{Par}_{\downarrow}(u\sigma)\}.$$

Proposition 6.23 $\mathtt{Pairs}(R)$ *is a set of marked dependency pairs of R.*

Proof. We have to check that for each non-marked pair $\langle u, v\rangle$, the set

$$\bigcup_{\sigma \in \Sigma_u} \{\langle u', \widehat{(v\sigma)}\rangle \text{ such that } u' \in \mathit{Par}_{\downarrow}(u\sigma)\}$$

satisfies the two DP conditions. For a given marked pair $\langle u_i, v_i\rangle$ of this set obtained from a given $\sigma \in \Sigma_u$, Condition 1

$$\widehat{v\sigma_i} \xrightarrow{m.e.}{}^{\star} v_i$$

trivially holds with $\sigma_i = \sigma$ since we have in fact $v_i = \widehat{(v\sigma)}$.

For Condition 2 now, assume given a substitution σ and a term $t \in \mathit{Par}(u\sigma)$, we must find substitutions σ_i and σ' verifying

$$\begin{gathered} \sigma =_{AC} \sigma_i\sigma' \\ t \xrightarrow{m.e.}{}^{\star} u_i\sigma' \end{gathered}$$

Since, by Proposition 6.6, it is always possible to reduce t with $\xrightarrow{m.e.}$ to a term in $\mathit{Par}_{\downarrow}(u\sigma)$ we may consider without loss of generality that $t \in \mathit{Par}_{\downarrow}(u\sigma)$, and we have to prove that

$$\begin{gathered} \sigma =_{AC} \sigma_i\sigma' \\ t =_{AC} u_i\sigma' \end{gathered}$$

For the sake of better readability we consider that the AC symbol on top of t is $+$, that is we have $t = t_1 \widehat{+} t_2$ where t_1 and t_2 do not have any more mark.

We are going to build the right σ_i by composition of well-chosen substitutions $\sigma_i^x \in \mathit{Split}_u(x)$ for each variable x of u, so that σ_i will be in Σ_u:

1. If x does not occur at depth 1 in u we simply put $\sigma_i(x) = x$.
2. If x occurs at depth 1 in u with multiplicity $m_u(x) = m$ then
   (a) If the root symbol of $x\sigma$ is not $+$, we simply choose $\sigma_i^x : x \mapsto x_0$ which is always in $\mathit{Split}_u(x)$.
   (b) If the root symbol of $x\sigma$ is $+$, say $x\sigma = s_1 + ... + s_n$: since multiplicity of x in u is m, we can write

   $$u = \underbrace{x + \cdots + x}_{m \text{ times}} + u'$$

   hence

   $$u\sigma = \underbrace{s_1 + \cdots + s_1}_{m \text{ times}} + \cdots + \underbrace{s_n + \cdots + s_n}_{m \text{ times}} + u'\sigma$$

   hence

   $$\widehat{u\sigma} = \underbrace{s_1 \widehat{+} \cdots \widehat{+} s_1}_{m \text{ times}} \widehat{+} \cdots \widehat{+} \underbrace{s_n \widehat{+} \cdots \widehat{+} s_n}_{m \text{ times}} \widehat{+} u''$$

   where $u'' = \widehat{u'\sigma}$ if its root is $+$ and $u'' = u'\sigma$ otherwise. We have

   $$\widehat{u\sigma} \xrightarrow{m.e.}{}^{\star} t_1 \widehat{+} t_2$$

   and all the point is to look at how many occurrences of the s_j stand, after marks erasing, in the left part t_1 and the right part t_2. Hence we define for each k, $0 \leq k \leq m$, S_k^x as the set of u_j with exactly k

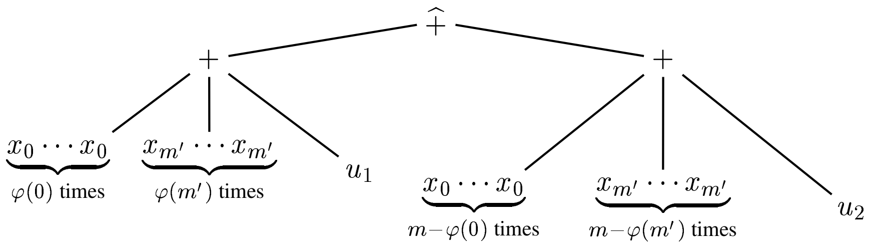
occurrences in t_1 (and consequently $m-k$ occurrences in t_2). Since some of these sets might be empty, we eventually get $m'+1$ non-empty sets with $0 \le m' \le m$. Let $x_0, \ldots, x_{m'}$ be $m'+1$ fresh variables, and let $\varphi : [0, m'] \to [0, m]$ be a map that associates to each x_j one of the non-empty set $S^x_{\varphi(j)}$. We choose then $\sigma^x_i : x \mapsto x_0 + \cdots + x_{m'}$.

We now take for σ' the substitution that, for each variable x, maps a fresh variable x_j to the sum of elements of $S^x_{\varphi(j)}$: $x_j\sigma' = w_1 + \cdots + w_k$ if $S^x_{\varphi(j)} = \{w_1, \cdots, w_k\}$ (or simply $x_j\sigma' = w_1$ if this set has only one element).

The last thing we have to do is to choose the right $u_i \in \mathit{Par}_\downarrow(u\sigma_i)$ such that $t_1 \widehat{+} t_2 =_{AC} u_i\sigma'$: we select the u_i which has, for each variable x, exactly $\varphi(j)$ occurrences of the fresh variable x_j on the left of its root symbol $\widehat{+}$, and $m' - \varphi(j)$ on the right. That is, for each variable x we can write

$$\begin{aligned} u_i \;=\; & (\underbrace{x_0 + \cdots + x_0}_{\varphi(0)\text{ times}} + \cdots + \underbrace{x_{m'} + \cdots + x_{m'}}_{\varphi(m')\text{ times}} + u_1) \\ & \widehat{+} (\underbrace{x_0 + \cdots + x_0}_{m-\varphi(0)\text{ times}} + \cdots + \underbrace{x_{m'} + \cdots + x_{m'}}_{m-\varphi(m')\text{ times}} + u_2) \end{aligned}$$

or, more graphically



and now in $u_i\sigma'$, a given term w that is in a $S^x_{\varphi(j)}$ appears exactly $\varphi(j)$ times, as many as it occurs in t_1, so we get exactly $u_i\sigma' = t_1 \widehat{+} t_2 = t$. $\square$

Example 6.24 (Example 6.19 revisited) *The* S^x_i *are*

$$\begin{aligned} S^x_0 &= \{a_3\}, \\ S^x_1 &= \{a_2, a_4\}, \\ S^x_2 &= \{a_1\} \end{aligned}$$

and since $m_t(x) = 2$ *we split* x *in* x_0*,* x_1 *and* x_2*. We have* $u_i = (x_1 + x_2 + x_2) \widehat{+} (x_0 + x_0 + x_1 + g(y))$ *and* $t = u_i\sigma'$ *with*

$$\sigma' : \begin{cases} x_0 & \mapsto \; a_3, \\ x_1 & \mapsto \; a_2 + a_4, \\ x_2 & \mapsto \; a_1. \end{cases}$$

6.4.2 Second variant

We now define a modified version of function `Pairs`, leading to pairs which are easier to orient (an important issue when building suitable orderings) but at a sensible combinatorial cost. The main difference is that right-hand sides are now in $\xrightarrow{m.e.}$ normal form.

Note that asking for marked symbols to be greater than their unmarked copies (Corollary 6.14) together with uniqueness of marked symbol in a term make modified rhs 'even smaller' and thus pairs easier to orient.

Counterpart to easy orientation is that modified pairs are numerous. Computing a set of modified pairs indeed amounts to going through all sharings by both sides of the marked symbol in rhs.

Definition 6.25 *Let R be a term rewriting system, we define the function `ModifiedPairs` as:*

$$\mathtt{ModifiedPairs}(R) = \bigcup_{\substack{\langle u, v\rangle \in DP(R),\\ \sigma \in \Sigma_u}} \left\{ \langle u', v'\rangle \textit{ such that } \begin{array}{l} u' \in \mathsf{Par}_{\downarrow}(\widehat{u\sigma}), \\ v' \in \mathsf{Par}_{\downarrow}(\widehat{v\sigma}) \end{array} \right\}.$$

Proposition 6.26 *$\mathtt{ModifiedPairs}(R)$ is a set of marked dependency pairs of R.*

Proof. Straightforward since rhs of modified pairs are obtained by erasing marks of right-hand sides of the AC dependency pairs defined in the first variant. □

Example 6.27 (Example 4.8 continued) *To prove termination of this system, the idea is to use MAPO with precedence $\cap > \cup$, $\mathsf{eq} > \mathsf{false}$ and $\mathsf{eq} > \mathsf{true}$ which will do the trick for all rules except the last one: we would like $\cap > \mathit{if}$ 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*

$$\langle z \mathbin{\widehat{\cap}} \{x\} \mathbin{\widehat{\cap}} \{y\}, z \mathbin{\widehat{\cap}} \mathit{if}(\mathsf{eq}(x, y), \{x\}, \emptyset)\rangle$$

decrease. This can be done by the RPS

$$\begin{aligned} \mathit{if}(x, y, z) &\to y \cap z \\ x \mathbin{\widehat{\cap}} y &\to x \cap y \\ x \mathbin{\widehat{\cup}} y &\to x \cup y \end{aligned}$$

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

$$\begin{aligned} \widehat{\mathit{if}}(x, y, z) &\to y \cap z \\ \widehat{\mathsf{eq}}(x, y) &\to x \cap y \end{aligned}$$

Finally, note that our implementation is not able to find this termination proof automatically, since no methods for generating RPS is implemented. Such a method as been proposed by Arts [4]: a finite number of 'interesting' RPS can be tested, and is certainly generalisable to rewriting modulo AC.

7 Conclusion and future work

We proposed extensions of dependency pairs criteria to AC case. While lifting up criterion without marks happens to be quite straigthforward, marked dependency pairs cannot be definied so easily and require higher abstraction. We propose then abstract conditions over candidate sets of pairs, *marked DP conditions*, and two ways of building suitable sets.

Those criteria are implemented in our system C*i*ME2 which is now able to prove termination in a fully automated way.

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

Firstly, as remarked in a preliminary version of this paper [25], dependency pairs criteria can be used in completion procedures. But 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 dependency pairs criteria 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, 4] 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 [3], 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 [24] which is a special case of hierarchical rewriting.

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] T. Arts and J. Giesl. Modularity of termination using dependency pairs. In Nipkow [27], pages 226–240.

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

[5] L. Bachmair. Associative-commutative reduction orderings. *Info Proc. Letters*, 1992.

[6] 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.

[7] L. Bachmair and N. Dershowitz. Critical pair criteria for completion. *Journal of Symbolic Computation*, 6(1):1–18, 1988.

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

[9] 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.

[10] C. Borralleras, M. Ferreira, and A. Rubio. Complete monotonic semantic path orderings. In D. McAllester, editor, *17th International Conference on Automated Deduction*, volume 1831 of *Lecture Notes in Computer Science*, Pittsburgh, PA, USA, June 2000. Springer-Verlag.

[11] 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.

[12] E. Contejean, C. Marché, B. Monate, and X. Urbain. Cime version 2, 2000. Prerelease available at `http://cime.lri.fr/`.

[13] 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.

[14] N. Dershowitz. Orderings for term rewriting systems. *Theoretical Computer Science*, 17(3):279–301, Mar. 1982.

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

[16] 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.

[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 [11].

[19] K. Kusakari. *Termination, AC-Termination and Dependency Pairs of Term Rewriting Systems*. PhD thesis, JAIST, 2000.

[20] 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.

[21] K. Kusakari and Y. Toyama. On proving AC-termination by AC-dependency pairs. Research Report IS-RR-98-0026F, School of Information Science, JAIST, 1998.

[22] P. Lescanne. Uniform termination of term rewriting systems with status. In *Proc. 9th CAAP*, Cambridge, 1984. Cambridge University Press.

[23] 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.

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

[25] C. Marché and X. Urbain. Termination of associative-commutative rewriting by dependency pairs. In Nipkow [27], pages 241–255.

[26] 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.

[27] T. Nipkow, editor. *9th International Conference on Rewriting Techniques and Applications*, volume 1379 of *Lecture Notes in Computer Science*, Tsukuba, Japan, Apr. 1998. Springer-Verlag.

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

[29] 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.

[30] 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.