



# Modular and Incremental Automated Termination Proofs

XAVIER URBAIN

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

(Received: 16 January 2002; accepted: April 2004)

**Abstract.** We propose a modular approach of term rewriting systems, making the best of their hierarchical structure. We define *rewriting modules* and then provide a new method to prove termination incrementally. We obtain new and powerful termination criteria for standard rewriting, thanks to the combination of dependency pairs and  $\mathcal{C}_\mathcal{E}$ -termination. Taking benefit of the generality of the module approach while restraining the notion of termination itself (thus relaxing constraints over hierarchies components), we can easily express previous results and methods the premises of which either include restrictions over unions or make a particular reduction strategy compulsory. We describe our implementation of the modular approach. Proofs are fully automated and performed incrementally. Since convenient orderings are simpler, we observe a dramatic speedup in the finding of the proof.

**Key words:** termination, term rewriting systems.

## 1. Introduction

Termination is the property of a program any execution of which terminates after a finite amount of time. It deserves its status of fundamental property because it is indissociable from the very existence of any calculus defined by a program. Without termination there is no result, at least in finite time. The termination property arises in various domains. It also acts as a preliminary for proofs of other properties.

We focus in this paper on termination of term rewriting systems (TRSs).

Rewriting is used for specification, in automated proofs, and also for programming. Yet, while programs are (should be) developed in an incremental way, a TRS is still considered in practice as a single set of rules. Termination proofs are made on the whole system without benefiting from its possible hierarchical structure.

Termination is undecidable. In particular, methods for proving termination are incomplete. Thus, most efforts focus on defining techniques devoted to proving termination of as many programs as possible. Lately, new methods have induced breakthroughs in automated termination proof, namely, Borralleras, Ferreira, and Rubio's *MSPO* [3] and Arts and Giesl's dependency pairs approach [2].

Nevertheless, proving termination of a term rewriting system still remains hard, especially when the system consists of many rules. The reason is that a divide-and-conquer strategy cannot be applied directly, thus making automation of proofs for systems with many rules a difficult task (see below). To provide a significant improvement in proving termination automatically, we focus on two critical points: automating termination proofs and computing them incrementally, so as to deal with systems of hundreds of rules (common in practice) efficiently.

Even if it allows a splitting up of a termination proof in several easier-to-solve subproblems, the incremental approach has not yet changed the way one deals with term rewriting systems, nor (even if numerous works on that subject can be found) the way one deals with the process itself of proving termination.

The great difficulty in guaranteeing termination of a union from termination of its components has actually too often been avoided by putting hard restrictions on relations between components (see, for instance, Krishna Rao's restricted proper extensions [21, 22] or Dershowitz's constructor-based extensions [9]), that is, without referencing to the problem within the notion of termination itself. Such strong requirements exclude most of the unions naturally occurring in programming practice.

Indeed, as shown by Toyama [32], termination is not even modular for systems with disjoint signatures.

EXAMPLE 1. Let us consider Toyama's example.

$$R : \{ f(0, 1, x) \rightarrow f(x, x, x) \} \quad \pi : \begin{cases} G(x, y) \rightarrow x \\ G(x, y) \rightarrow y \end{cases}$$

Both systems  $R$  and  $\pi$  terminate. But if we consider their union,

$$\begin{cases} G(x, y) \rightarrow x \\ G(x, y) \rightarrow y \\ f(0, 1, x) \rightarrow f(x, x, x), \end{cases}$$

we may find an infinite reduction sequence, for instance,

$$\begin{aligned} f(G(0, 1), G(0, 1), G(0, 1)) &\rightarrow f(0, G(0, 1), G(0, 1)) \\ &\rightarrow f(0, 1, G(0, 1)) \\ &\rightarrow f(G(0, 1), G(0, 1), G(0, 1)) \rightarrow \dots \end{aligned}$$

Instead of restraining the scope of our study by putting strong constraints on unions, we allow weakly constrained unions (so-called hierarchical) and consider a (slightly) restricted notion of termination that is preserved under nondeterministic collapse,  $\mathcal{C}_\varepsilon$ -termination. From a historical point of view, Rusinowitch [30] remarked that the infinite reduction of Example 1 was due to the projective behavior of system  $\pi$ . In fact, similar counterexamples may arise when one deals with systems with a nondeterministic projective behavior, more precisely when a

term  $t$  may be reduced to two distinct variables. Gramlich [17] then defined the notion of *termination preserved under nondeterministic collapse*, for which he had results. This is the property of terminating TRSs that remain terminating when one adds rules  $\{C[x, y] \rightarrow x; C[x, y] \rightarrow y\}$  for a context  $C$  (which express exactly that some term  $t$  containing variables  $x$  and  $y$  can be reduced to one of those). This is equivalent to adding system  $\pi$  where  $G$  is fresh. That property was later renamed *collapse extended termination* ( $\mathcal{C}_\varepsilon$ -termination, for short) and studied by Ohlebusch [27]: a system  $R$  is said to be  $\mathcal{C}_\varepsilon$ -terminating if  $R \uplus \pi$  (where  $G$  is a fresh symbol) is terminating.

We claim that focusing on  $\mathcal{C}_\varepsilon$ -termination is not too much of a restriction in practice. Actually, virtually all automated techniques for proving termination lead to  $\mathcal{C}_\varepsilon$ -termination, and among those, the powerful dependency pairs approach in particular (when used in conjunction with simplification orderings). In other words,  $\mathcal{C}_\varepsilon$ -termination provides means for controlling the  $\pi$ -projection's possibly improper behavior – illustrated by Toyama's example – with reference to termination of most systems occurring in practice and for which we want an automated proof.

System  $R$  of Example 1 is not  $\mathcal{C}_\varepsilon$ -terminating and so does not enter the scope of our study anymore: we simply will not try to prove its termination automatically. That system may be considered as “pathological,” and such rules barely (if not never) occur in practical applications. The motivation of our work is to prove termination of systems with many rules that may be encountered in practice.

### *Hierarchical Structure*

The common presentation of TRSs is based on the union of sets of rules (over some signatures) that are TRSs by themselves. This is not well suited to the expression of their inner hierarchical structure for at least the following two reasons: it brings redundancy at the common subsystem definition level when symbols or rules are shared, and it does not keep track of the actual sequential construction and thus does not enhance any incremental process (say, a termination proof). Both contribute to separating rewriting from actual programming.

To give a general framework bringing the hierarchical structure of TRSs to the fore, and thus provide automated methods to prove termination incrementally, we first recall some generalities. Then, in Section 3 we introduce *rewriting modules*. We show how the intrinsic hierarchical structure of TRSs naturally emerges from their formalization in terms of modules.

In particular, this approach allows one to build TRSs the same way programs are built: modules after modules. It is also possible to split a whole TRS into several modules. Termination criteria based on these modules and leading to incremental and modular termination proofs would constitute a major breakthrough with reference to the study of large TRSs, as well as in the (incremental) development of specifications or applications.

To that purpose, *dependency pairs of modules* as well as *relative dependency chains* are defined in Section 4. These powerful tools lead us to new criteria for incremental and modular termination (Theorem 1 and Theorem 2).

Indeed, these criteria enjoy the full benefit of a really modular and incremental approach together with the advantages of a dependency pairs approach: proving termination with modules amounts to solving fewer and simpler termination constraints than in a global approach.

Section 5 provides a class of orderings with good properties with respect to projection, hence to  $\mathcal{C}_\varepsilon$ -termination (the so-called  $\pi$ -expandable orderings).

Those results provide an incremental proof method illustrated with the complete example of Section 6.1 and a new notion of termination directly related to the incremental process *hierarchical simple termination* (Section 7).

### *An Incremental and Modular Termination Tool*

All those results were developed with a constant thought for automation. Thanks to their generality as well as the purely syntactical tests provided, they are now part of the CiME 2 rewriting tool [5]. With our incremental and modular criteria, CiME 2 gives its user the possibility of searching termination proofs in a totally automated fashion. Section 6.2 provides benchmarks showing how important the obtained gain is (with our tool) compared to some other techniques. We also tested CiME 2 on TRSs of approximately 400 rules with success: a termination proof is found in a few seconds. Examples came from a  $\mu$ -CRL specification of communicating processes by Thomas Arts,\* which means they occur in practice. Their huge number of rules is not a problem for our termination tool because the incremental approach reduces termination constraints dramatically.

We discuss in Section 8 how this work compares to that of others, especially to results of Arts and Giesl [1] and Dershowitz [9].

## 2. Preliminaries

We recall usual notions about rewriting [10] and give our notation.

A *signature*  $\mathcal{F}$  is a finite set of *symbols* with arities. Let  $X$  be a countable set of *variables*;  $T(\mathcal{F}, X)$  denotes the set of finite *terms* on  $\mathcal{F}$  and  $X$ . Terms can be seen as trees: root position is then denoted by  $\Lambda$ , symbol at root position in a term  $t$  by  $\Lambda(t)$ ,  $t|_p$  denotes the subterm of  $t$  at position  $p$ . A *substitution* is a mapping  $\sigma$  from variables to terms. We use postfix notation for substitution applications. A substitution can easily be extended to endomorphisms of  $T(\mathcal{F}, X)$ :  $f(t_1, \dots, t_n)\sigma = f(t_1\sigma, \dots, t_n\sigma)$ . Then,  $t\sigma$  is called an *instance* of  $t$ .

A *rewrite relation* is a binary relation  $\rightarrow$  on terms that is monotonic and closed under substitution;  $\rightarrow^*$  denotes its reflexive-transitive closure. A *term rewriting system* (TRS for short) over a signature\*\*  $\mathcal{F}$  and a set of variables  $X$  is a (possibly

\* Personal communication.

\*\* We shall omit the signature if there is no ambiguity.

infinite) set  $R(\mathcal{F})$  of *rewrite rules*  $l \rightarrow r$ . A TRS  $R$  defines a rewrite relation  $\rightarrow_R$  in the following way:  $s \rightarrow_R t$  if there is a position  $p$  such that  $s|_p = l\sigma$  and  $t = s[r\sigma]_p$  for a rule  $l \rightarrow r \in R$  and a substitution  $\sigma$ . We say then that  $s|_p$  is a *redex* and that  $s$  *reduces to  $t$  at position  $p$  with  $l \rightarrow r$*  denoted  $s \xrightarrow[p]{l \rightarrow r} t$ . In cases where knowing the exact applied rule of  $R$  does not matter, or if there is no ambiguity, we simply say that the reduction is performed *with  $R$*  and denote it  $s \xrightarrow[R]{p} t$ . Symbols occurring at root position in the left-hand sides of rules in  $R$  are said to be *defined (in  $R$ )*; the others are said to be *constructors (in  $R$ )*.

A term is  *$R$ -strongly normalizable (SN)* if it cannot reduce infinitely many times for the relation defined by system  $R$ . If  $R$  is clear from the context, we omit it. Similarly, a substitution  $\sigma$  is said to be  *$R$ -strongly normalizable* if for any variable  $x$ ,  $x\sigma$  is SN.

A rewrite relation is *terminating* or *terminates* if any term is SN.

Termination is usually proven with the help of *reduction orderings* [8] or quasi-orderings with dependency pairs. We briefly recall what we need. An *ordering pair* is a pair  $(\succeq, >)$  of relations over  $T(\mathcal{F}, X)$  such that (1)  $\succeq$  is a quasi-ordering (i.e., reflexive and transitive), (2)  $>$  is a strict ordering (i.e., irreflexive and transitive), and (3)  $> \cdot \succeq = >$  or  $\succeq \cdot > = >$ . We refer to these pairs as *term orderings*, since it seems natural to keep the usual denomination w.r.t. the use we make of them (especially for all comparisons on terms), even if the actual technical object is different. A term ordering is said to be *well-founded* if there is no infinite strictly decreasing sequence  $t_1 > t_2 > \dots$ ; *stable* if both  $>$  and  $\succeq$  are stable under substitutions; and *weakly (resp. strictly) monotonic* if for all terms  $t_1$  and  $t_2$ , for all  $f \in \mathcal{F}$ , if  $t_1 \succeq$  (resp.  $>$ )  $t_2$ , then  $f(\dots, t_1, \dots) \succeq$  (resp.  $>$ )  $f(\dots, t_2, \dots)$ . A term ordering  $(\succeq, >)$  is called a *weak (resp. strict) reduction ordering* if it is well-founded, stable, and weakly (resp. strictly) monotonic; a *simplification ordering* if it is stable and monotonic and has the subterm property with reference to  $\succeq$ :  $C[t] \succeq t$  for any context  $C$  and term  $t$ .

We point out that our notion of weak reduction ordering is a particular case of the very general notion of *weak reduction pair* of Kusakari et al. [24], which requires (1')  $\succeq$  to be any monotonic and stable relation (but not necessarily reflexive nor transitive), (2')  $>$  to be well-founded and stable, and (3')  $> \cdot \succeq \subseteq >$  or  $\succeq \cdot > \subseteq >$ . However, it is easy to see that if  $\succeq$  is reflexive, (3') implies (3); in other words, any weak reduction pair made of orderings is a weak reduction ordering in our setting.

We distinguish between modular and incremental proofs. A modular property  $\mathcal{P}$  behaves as follows: if components  $C_1$  and  $C_2$  have property  $\mathcal{P}$ , so does the union  $C_1 \cup C_2$ . Proving a property  $\mathcal{P}'$  in an incremental manner consists first of proving  $\mathcal{P}'$  on a component  $C_1$ , then of showing  $\mathcal{P}'$  on  $C_1 \cup C_2$ , possibly using the knowledge that  $C_1$  verifies  $\mathcal{P}'$ .

### $\mathcal{C}_\varepsilon$ -Termination

A TRS  $R(\mathcal{F})$  is said to be *simply terminating* if  $R \cup (\bigcup_{f \in \mathcal{F}} \{f(x_1, \dots, x_j, \dots, x_n) \rightarrow x_j \mid 1 \leq j \leq n\})$  terminates.

A TRS  $R(\mathcal{F})$  is said to be  $\mathcal{C}_\varepsilon$ -*terminating* if  $R \uplus \pi$ , where  $\pi = \{G(x, y) \rightarrow x, G(x, y) \rightarrow y\}$  (with  $G \notin \mathcal{F}$ ) is *terminating*. Throughout this article,  $G$  always is used for the special symbol of TRS  $\pi$ .

Unfortunately,  $\mathcal{C}_\varepsilon$ -termination is not modular for unions of nonfinitely branching TRSs sharing constructors, as shown by Ohlebusch [27]. Hence we restrict our study to the *finitely branching* case, that is, the case of TRS  $R$  such that for any  $l \rightarrow r$  of its rules, there are finitely many different rules in  $R$  with  $l$  as the left-hand side. This implies, in particular, that only finitely many rules may be applied to a given term.

$\mathcal{C}_\varepsilon$ -termination behaves nicely with reference to unions of TRSs. It is actually a modular property for unions of

- disjoint TRSs (Gramlich, Ohlebusch) [16, 27],
- finitely branching and constructor sharing systems (Gramlich) [16],
- finitely branching and composable systems (Kurihara and Ohuchi) [23].

Ohlebusch showed that the finitely branching condition is compulsory [27].

$\mathcal{C}_\varepsilon$ -termination is less restrictive than simple termination. For instance, a system  $R$  is simply terminating if the union of  $R$  and all projections for *all* symbols still terminates; this allows elimination of any constructors “hiding” a redex, while such an elimination is impossible with rules of  $\pi$  only.

**PROPOSITION 1** (Gramlich [16]). *Any simply terminating TRS is  $\mathcal{C}_\varepsilon$ -terminating.*

### The Dependency Pairs Approach

We briefly recall the dependency pairs approach by Arts and Giesl [2].

Termination proofs based on dependency pairs were recently introduced by Arts and Giesl. This approach focuses on a deeper analysis of the structure of terms that can be reduced.

**DEFINITION 1** (Arts and Giesl [2]). Let  $R(\mathcal{F})$  be a TRS, and let  $\mathcal{F}$  be split up into two sets  $\mathcal{F}_D$  and  $\mathcal{F}_C$  containing respectively the symbols defined in  $R$  and the constructors. Let  $\widehat{\mathcal{F}}$  be the extension of signature  $\mathcal{F}$  with fresh symbols  $\widehat{f}$  for all  $f \in D$ . If

$$f(s_1, \dots, s_n) \rightarrow C[g(t_1, \dots, t_m)]$$

is a rewrite rule of  $R$ , for  $C$  being a context and with  $g \in D$ , then the pair  $(\widehat{f}(s_1, \dots, s_n), \widehat{g}(t_1, \dots, t_m))$  is called a *dependency pair* of  $R$ .

The set of all dependency pairs of all rules in  $R$  is denoted  $\text{DP}(R)$ .

Dependency pairs may be used for proving termination using the notion of *dependency chains*: a sequence  $\dots \langle s_i, t_i \rangle \langle s_{i+1}, t_{i+1} \rangle \dots$  of pairs in  $\text{DP}(R)$  is an *R-(dependency) chain* if there exists a substitution  $\sigma$  such that

$$t_j \sigma \xrightarrow[R]{*} s_{j+1} \sigma$$

holds for every two consecutive pairs  $\langle s_i, t_i \rangle$  and  $\langle s_{i+1}, t_{i+1} \rangle$  in the sequence.

Arts and Giesl prove the following criterion.

**THEOREM** (Arts and Giesl [2]). *A TRS  $R(\mathcal{F})$  is terminating if and only if no infinite R-chain exists.*

We note that proving automatically that no infinite *R*-chain exists is easier than proving the strict decrease of each rule. In particular, the use of strictly monotonic orderings is not mandatory.

### The Running Example

We describe here the main example of the paper. Instead of choosing the usual Peano’s arithmetic example, we prefer an example occurring in real programming. Binary arithmetic is really implemented, so that is what we are going to study.

**EXAMPLE 2.** The following system *R* describes addition and multiplication of natural numbers in functional notation: # denotes  $0_{\mathbb{N}}$ ,  $(x)0$  denotes the value of *x* multiplied by two, and  $(x)1$  denotes the value of *x* multiplied by  $2_{\mathbb{N}}$  plus  $1_{\mathbb{N}}$ . In this formalism,  $6_{\mathbb{N}}$  is written #110, that is, the usual binary notation with a # in front.

$$R : \left\{ \begin{array}{ll} \#0 \rightarrow \# & \\ \# + x \rightarrow x & x + \# \rightarrow x \\ x0 + y0 \rightarrow (x + y)0 & x1 + y0 \rightarrow (x + y)1 \\ x0 + y1 \rightarrow (x + y)1 & x1 + y1 \rightarrow ((x + y) + \#)1 \\ \# \times x \rightarrow \# & x \times \# \rightarrow \# \\ x0 \times y \rightarrow (x \times y)0 & x1 \times y \rightarrow (x \times y)0 + y \end{array} \right.$$

The first (simplification) rule ensures that  $2_{\mathbb{N}} \times 0_{\mathbb{N}}$  is  $0_{\mathbb{N}}$ , that is, that zeros in front of numbers in binary notation can be erased.

### 3. Rewriting Modules

In this section, we define *rewriting modules* and show how they bring to the fore the hierarchical structure of term rewriting systems.

From an operational point of view, a module consists of a set of “new” symbols together with the rules that define them.

**DEFINITION 2.** Let  $R_1$  be a term rewriting system over a signature  $\mathcal{F}_1$ . A *module* extending  $R_1(\mathcal{F}_1)$  is a pair  $[\mathcal{F}_2 \mid R_2]$  such that

- (1)  $\mathcal{F}_1 \cap \mathcal{F}_2 = \emptyset$  (signatures are disjoint);
- (2)  $R_2$  is a term rewriting system over  $\mathcal{F}_1 \cup \mathcal{F}_2$ ;
- (3) for all  $l \rightarrow r \in R_2$ ,  $\Lambda(l) \in \mathcal{F}_2$ .

One can easily see that  $R_1 \cup R_2$  is a TRS over  $\mathcal{F}_1 \cup \mathcal{F}_2$ . We say then that system  $R_1 \cup R_2$  over  $\mathcal{F}_1 \cup \mathcal{F}_2$  is a *hierarchical extension of  $R_1(\mathcal{F}_1)$  with module  $[\mathcal{F}_2 \mid R_2]$* . We write such an extension as

$$R_1(\mathcal{F}_1) \longleftarrow [\mathcal{F}_2 \mid R_2].$$

A module can extend more than one system, and the extension by a module naturally associates “arrowwise.” Hence we write

$$R_1(\mathcal{F}_1) \longleftarrow [\mathcal{F}_2 \mid R_2] \longleftarrow [\mathcal{F}_3 \mid R_3]$$

the extension

$$(R_1(\mathcal{F}_1) \longleftarrow [\mathcal{F}_2 \mid R_2]) \longleftarrow [\mathcal{F}_3 \mid R_3].$$

The extension of several distinct systems by a single module is an alternative notation for the (simple) extension of the union of the relevant systems by the module.

*Remark 1.* For the sake of readability, we may denote

$$[\mathcal{F}_2 \mid R_2] \longleftarrow [\mathcal{F}_3 \mid R_3]$$

the hierarchical extension (with  $[\mathcal{F}_3 \mid R_3]$ ) of the whole hierarchy extended with (that is, *headed by*)  $[\mathcal{F}_2 \mid R_2]$ .

For instance, we may abbreviate “the hierarchy headed by  $[\mathcal{F}_2 \mid R_2]$  is extended with  $[\mathcal{F}_3 \mid R_3]$ ,” denoted

$$R_1(\mathcal{F}_1) \longleftarrow [\mathcal{F}_2 \mid R_2] \longleftarrow [\mathcal{F}_3 \mid R_3],$$

in “ $[\mathcal{F}_3 \mid R_3]$  extends  $[\mathcal{F}_2 \mid R_2]$ ,” denoted

$$[\mathcal{F}_2 \mid R_2] \longleftarrow [\mathcal{F}_3 \mid R_3].$$

Two disjoint modules may extend the same base hierarchy.

**DEFINITION 3.** We say that a module  $[\mathcal{F}_2 \mid R_2]$  extends a hierarchy headed by  $[\mathcal{F}_0 \mid R_0]$  *independently of* a module  $[\mathcal{F}_1 \mid R_1]$  if

- $\mathcal{F}_1 \cap \mathcal{F}_2 = \emptyset$ ,
- $[\mathcal{F}_0 \mid R_0] \longleftarrow [\mathcal{F}_1 \mid R_1]$  and
- $[\mathcal{F}_0 \mid R_0] \longleftarrow [\mathcal{F}_2 \mid R_2]$ .



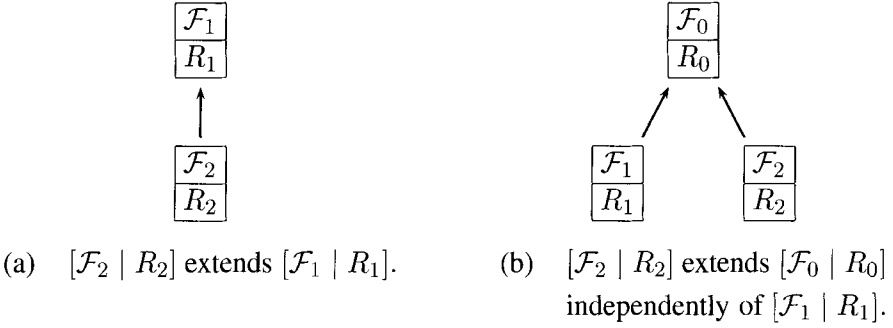


Figure 1. Graphical notations of hierarchical extensions.

Such an extension may be seen as a *union of composable systems* [23, 26, 28].

Figure 1 illustrates the two different kinds of extensions.

One can express different kinds of classical extensions by means of extensions of modules. For instance, we obtain a *disjoint union*  $R_1(\mathcal{F}_1) \uplus R_2(\mathcal{F}_2)$  whenever  $[\mathcal{F}_1 \mid R_1]$  and  $[\mathcal{F}_2 \mid R_2]$  extend  $[\emptyset \mid \emptyset]$  independently of each other.

We may also describe a *constructor shared union* with modules. A union of two TRSs  $R_1(\mathcal{F}_1)$  and  $R_2(\mathcal{F}_2)$  is said to be a constructor shared union if all symbols in  $\mathcal{F}_1 \cap \mathcal{F}_2$  are constructors for both  $R_1$  and  $R_2$ . Let us consider two systems  $R_1(\mathcal{F}_1)$  and  $R_2(\mathcal{F}_2)$  that share only a set of common constructors  $C_0 = \mathcal{F}_1 \cap \mathcal{F}_2$ . Then the constructor-shared union  $R_1 \cup R_2$  is easily denoted with two modules  $[\mathcal{F}_1 \mid R_1]$  and  $[\mathcal{F}_2 \mid R_2]$  independently extending the module of “constructors”  $[C_0 \mid \emptyset]$ .

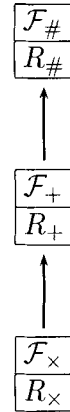
The notion of *hierarchical extensions with common subsystem* [27] is captured because extension with modules associates arrowwise.

We further note that the only condition we put on extensions is that new rules must have a new symbol at the root position of their left-hand side (see Definition 2). Thus, module extensions subsume notions of heavily constrained hierarchical extensions such as *constructor-based extensions* [9] (systems in which no left-hand side has a symbol below the top that appears at the top of any left-hand side) and *proper extensions* [21, 22] (involving constraints on right-hand sides subterms with reference to a dependency relation on symbols).

**EXAMPLE 3.** Let us consider the binary arithmetic example (Example 2). System  $R$  may be seen as an extension involving three modules:

- module  $R_{\#}$  actually defining integers in binary notation;
- module  $R_+$  consisting of rules for addition over integers;
- module  $R_{\times}$  describing multiplication.

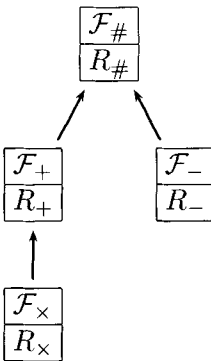
$$\begin{aligned}
 & \mathcal{F}_\# \{ \#, 0, 1 \} \\
 & R_\# \{ \#0 \rightarrow \# \} \\
 & \mathcal{F}_+ \{ + \} \\
 & R_+ \left\{ \begin{array}{ll} \# + x \rightarrow x & x + \# \rightarrow x \\ x0 + y0 \rightarrow (x + y)0 & x1 + y0 \rightarrow (x + y)1 \\ x0 + y1 \rightarrow (x + y)1 & x1 + y1 \rightarrow ((x + y) + \#)0 \end{array} \right. \\
 & \mathcal{F}_\times \{ \times \} \\
 & R_\times \left\{ \begin{array}{ll} \# \times x \rightarrow \# & x \times \# \rightarrow \# \\ \{ \times \} x0 \times y \rightarrow (x \times y)0 & x1 \times y \rightarrow (x \times y)0 + y \end{array} \right.
 \end{aligned}$$



Now adding rules for subtraction consists only in extending  $R_\#$  by the relevant module  $[\mathcal{F}_- \mid R_-]$  independently of each other module.

$$\mathcal{F}_- \{ - \} \\
 R_- \left\{ \begin{array}{ll} x - \# \rightarrow x & \# - x \rightarrow \# \\ (x)0 - (y)0 \rightarrow (x - y)0 & (x)0 - (y)1 \rightarrow ((x - y) - (\#)1)1 \\ (x)1 - (y)0 \rightarrow (x - y)1 & (x)1 - (y)1 \rightarrow (x - y)0 \end{array} \right.$$

And the hierarchy becomes



Now we would like to prove termination of the obtained hierarchy in an incremental way, that is, using the knowledge that a subhierarchy terminates while proving termination of its extension by a module.

MODULAR DECOMPOSITION

Any TRS can be studied as a certain modules hierarchy exploited in its present state (i.e., as it is provided). However, one can consider a unique canonical decomposition. A TRS can actually be seen as a hierarchy of *minimal modules*, modules that cannot be split up themselves in a hierarchy of nonempty modules.

For that purpose, we use the graph of a purely syntactical dependency relation between symbols.

DEFINITION 4. For a TRS  $R(\mathcal{F})$ , we say that a symbol  $f \in \mathcal{F}$  *directly depends* on a  $g \in \mathcal{F}$  if and only if there is a rule  $l \rightarrow r \in R$  with

- $f = \Lambda(l)$  and
- $g$  either occurs in  $l$  or in  $r$ .

We note  $\triangleright_d$  that relation.

The decomposition is done in two steps. For a TRS  $R(\mathcal{F})$ ,

1. we build a graph  $\mathcal{G}$  the vertices of which are symbols of  $\mathcal{F}$  and such that there is an arc from a vertex  $x$  to a vertex  $y$  if and only if  $x \triangleright_d y$ ,
2. we pack together symbols of strongly connected components of  $\mathcal{G}$ , that is, symbols  $f$  and  $g$  such that

$$f \triangleright_d^* g \quad \text{and} \quad g \triangleright_d^* f.$$

In other words, signatures of modules are classes for the equivalence relation generated by  $\triangleright_d$ .

Building modules from these packs is easily done by joining for each of their symbols the rules for which they occur at  $\Lambda$ . The module hierarchy may then be read on graph  $\mathcal{G}$ .

Note that there is no cycle in the obtained hierarchy because symbols of mutually recursive functions appear in same packs. Thus, they belong to same modules. Such a decomposition is clearly unique.

*Remark 2.* For the sake of readability, that is, in order to avoid many modules with no rule and only one constructor, one can gather constructor symbols reachable from the same packs.

In particular, the hierarchy shown in Example 3 is a canonical one with reference to Remark 2.

#### 4. Incremental and Modular Termination

The module framework provides a dependency pairs approach applicable to an incremental treatment of the termination proof.

##### 4.1. DEPENDENCY PAIRS OF MODULES

DEFINITION 5. Let  $\widehat{\mathcal{F}}$  be signature  $\mathcal{F}$  extended with “marked copies”  $\widehat{f}$  for all symbols  $f$ . Marking of a term is as follows: for a nonvariable term  $t \in T(\mathcal{F}, X)$ , we denote  $\widehat{t}$  the term  $t$  in which the symbol at  $\Lambda$  has been replaced by its marked copy.

Let  $M = [\mathcal{F} \mid R]$  be a module. A *dependency pair of module  $M$*  is a pair of terms  $(\widehat{l}, \widehat{r}')$  such that there is a rule  $l \rightarrow r \in R$  for which term  $r'$  is a subterm of  $r$  with  $\Lambda(r') \in \mathcal{F}$  defined in  $R$ .

We denote  $\text{MDP}(M)$  the set of all dependency pairs of a module  $M$ .

Since we consider now only subterms whose rhs root symbols are defined *locally*, all dependency pairs of a module  $[\mathcal{F} \mid R]$  extending a system  $R_0(\mathcal{F}_0)$  belong to the set of “classical” dependency pairs of  $R$  considered in  $R \cup R_0$ . In other words,  $\text{MDP}([\mathcal{F} \mid R]) \subseteq \text{DP}(R \cup R_0)$ .

EXAMPLE 4. In the binary arithmetic example, the classical approach amounts to considering five dependency pairs for rules defining multiplication (that is, with symbol  $\times$  at  $\wedge$ ).

$$\begin{array}{lll} \langle x0 \widehat{\times} y, x \widehat{\times} y \rangle & \langle x0 \widehat{\times} y, (x \times y)\widehat{0} \rangle & \\ \langle x1 \widehat{\times} y, x \widehat{\times} y \rangle & \langle x1 \widehat{\times} y, (x \times y)\widehat{0} \rangle & \langle x1 \widehat{\times} y, (x \times y)0 \widehat{+} y \rangle. \end{array}$$

But there are only two dependency pairs of module  $R_\times$ , namely,  $\langle x0 \widehat{\times} y, x \widehat{\times} y \rangle$  and  $\langle x1 \widehat{\times} y, x \widehat{\times} y \rangle$ , since  $\times$  is the only symbol of signature  $\mathcal{F}_\times$ .

*Remark 3.* Let us consider a system  $R(\mathcal{F})$  over constructors  $\mathcal{F}_C$  and defined symbols  $\mathcal{F}_D$ . When such a system is seen as the extension  $[\mathcal{F}_C \mid \emptyset] \leftarrow [\mathcal{F}_D \mid R]$ , the dependency pairs of module  $[\mathcal{F}_D \mid R]$  are exactly the “classical” dependency pairs of  $R(\mathcal{F})$  in Arts and Giesl’s approach. That is,  $\text{MDP}([\mathcal{F}_D \mid R])$  and  $\text{DP}(R(\mathcal{F}))$  coincide.

## 4.2. RELATIVE DEPENDENCY CHAINS

We saw that, considered in a hierarchy, the notion of “defined symbol” loses its absolute meaning to become local to a module of that hierarchy.

The same phenomenon arises with chains of dependency pairs of modules: they indeed rely on what rules *occurring in the hierarchy* may be applied between MDP-steps, and not only on what is provided by the considered module. In other words, dependency chains become *relative* to some relevant set of rules.

DEFINITION 6. Let  $M = [\mathcal{F} \mid R]$  be a module, and let  $S$  be an arbitrary term rewriting system. A *dependency chain of  $M$  over  $S$*  is a sequence of pairs of  $\text{MDP}(M)$  together with a substitution  $\sigma$  such that for any two successive pairs  $\langle s_i, t_i \rangle$  and  $\langle s_{i+1}, t_{i+1} \rangle$ ,

$$t_i \sigma \xrightarrow[S]{\neq \Lambda^*} s_{i+1} \sigma.$$

A dependency chain of a module  $M$  over a system  $S$  with a substitution  $\sigma$  is said to be *minimal* if  $\sigma$  is  $S$ -strongly normalizable.

Note that since  $S$  is an arbitrary TRS, it may be completely different from  $R$ . In particular, we may have  $S \supset R$ .

Further note that requiring nonroot reductions between pairs avoid assumptions on those: marked or unmarked pairs may be used.

The following proposition gives a (new) characterization of  $\mathcal{C}_\varepsilon$ -termination in terms of relative dependency chains.

**PROPOSITION 2.** *A term rewriting system  $R(\mathcal{F})$  is  $\mathcal{C}_\varepsilon$ -terminating if and only if there is no infinite chain of  $[\mathcal{F} \mid R]$  over  $R \cup \pi$ .*

*Proof.* Since  $G$  does not belong to  $\mathcal{F}$  and since right-hand sides of  $\pi$  are variables,  $DP(R \cup \pi) = DP(R)$  and  $\mathcal{C}_\varepsilon$ -termination of  $R$  is shown with the help of Remark 3. □

This characterization is entirely expressed within the module framework and, thus, is quite convenient in proofs, as a built-in property.

### 4.3. TERMINATION WITH MODULES

Dependency pairs of modules and relative dependency chains allow us to define some purely syntactical tests so as to prove termination in an incremental fashion. This section is organized as follows: first we make a remark about chains (Lemma 1); then we state our main result, Theorem 1. Its proof is rather technical and involves a key lemma, Lemma 2. To prove this lemma, we define an interpretation of terms and state several lemmas about it. Eventually we prove Lemma 2.

From Theorem 1 we obtain as a corollary sufficient conditions to ensure composability of  $\mathcal{C}_\varepsilon$ -termination (Corollary 1), that is, that it can be proven incrementally from  $\mathcal{C}_\varepsilon$ -terminating hierarchies. We then focus on extensions of a system with two independent modules and state Theorem 2, a corollary of which is a previous result by Kurihara and Ohuchi [23]. The hierarchy graph of theorems and lemmas may be found Figure 2.

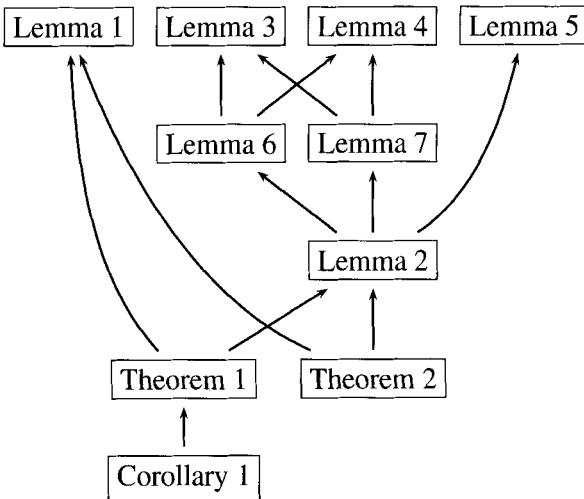


Figure 2. Hierarchy graph of theorems and lemmas.

### Termination with Modules

We point out here an interesting property of chains of modules.

LEMMA 1. *Let  $R_1(\mathcal{F}_1)$  be a TRS and  $[\mathcal{F}_2 \mid R_2]$  be a module such that  $[\mathcal{F}_1 \mid R_1] \leftarrow [\mathcal{F}_2 \mid R_2]$ .*

*Then for any two pairs  $\langle u_1, v_1 \rangle, \langle u_2, v_2 \rangle$  s.t.  $\langle u_1, v_1 \rangle \in \text{MDP}([\mathcal{F}_1 \mid R_1])$  and  $\langle u_2, v_2 \rangle \in \text{MDP}([\mathcal{F}_2 \mid R_2])$ , there is no substitution  $\sigma$  such that*

$$v_1\sigma \xrightarrow{\neq \Lambda^*} u_2\sigma.$$

*Proof.* Since  $\Lambda(u_2\sigma) = \Lambda(u_2) \in \widehat{\mathcal{F}}_2$  and  $\Lambda(v_1\sigma) = \Lambda(v_1) \in \widehat{\mathcal{F}}_1$ , we obtain  $\Lambda(u_2) \neq \Lambda(v_1)$ .  $\square$

This property will be useful in the proof of the two main results, Theorems 1 and 2.

We now state Theorem 1.

THEOREM 1. *Let  $[\mathcal{F}_1 \mid R_1] \leftarrow [\mathcal{F}_2 \mid R_2]$  be a hierarchical extension of  $R_1(\mathcal{F}_1)$ ; if*

- (1)  $R_1$  is  $\mathcal{C}_\varepsilon$ -terminating, and
- (2) there is no infinite dependency chain of  $[\mathcal{F}_2 \mid R_2]$  over  $R_1 \cup R_2$ ,

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

In the proof of this theorem, as well as in the proof of Theorem 2, we use a more general result given here as a technical key lemma: Lemma 2.

LEMMA 2. *Let  $S_1$  and  $S_2$  be two TRSs over signature  $\mathcal{F}_1$ . Let  $S_3(\mathcal{F}_1 \cup \mathcal{F}_2)$  be such that*

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

*Then, from an infinite minimal dependency chain of  $[\mathcal{F}_1 \mid S_2]$  over  $S_1 \cup S_2 \cup S_3$ , one can build an infinite dependency chain of  $[\mathcal{F}_1 \mid S_2]$  over  $S_1 \cup S_2 \cup \pi$  consisting of*

- the same sequence of pairs,
- a new substitution as well as new rewriting steps.

*Proof of Theorem 1.* By contradiction. The proof scheme is as follows. Let us suppose that there is an infinite dependency chain of  $R_1 \cup R_2$ . We are going to show that

- either there is an infinite dependency chain of  $[\mathcal{F}_2 \mid R_2]$  over  $R_1 \cup R_2$ , thus contradicting the second premise,
- or  $R_1$  is not  $\mathcal{C}_\varepsilon$ -terminating, now contradicting the first premise of Theorem 1.

We suppose that  $R_1 \cup R_2$  does not terminate. Thus there is an infinite dependency chain of module  $[\mathcal{F}_1 \cup \mathcal{F}_2 \mid R_1 \cup R_2]$  over  $R_1 \cup R_2$ . Recall that marked symbols occur at root position only.

Pairs of MDP( $[\mathcal{F}_1 \cup \mathcal{F}_2 \mid R_1 \cup R_2]$ ) consist of

- ① pairs of  $[\mathcal{F}_1 \mid R_1]$ ;
- ② pairs of  $[\mathcal{F}_2 \mid R_2]$ ;
- ③ pairs  $(\widehat{l}, \widehat{r'})$  such that  $l \rightarrow r \in R_2$  and  $r'$  is a subterm of  $r$  the root symbol  $\Lambda(r')$  of which belongs to  $\mathcal{F}_1$ .

Premises give us information with reference to the first two cases. To avoid the third one, we use Lemma 1.

From that lemma we know that pairs ② and ③ may follow pairs ② only in a dependency chain. Similarly, pairs ① may follow pairs ① or ③ only.

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

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

- First case: An infinite dependency chain of pairs of  $[\mathcal{F}_2 \mid R_2]$  over  $R_1 \cup R_2$  contradicts the second premise of Theorem 1.
- Cases 2 and 3: In both cases an infinite chain of  $[\mathcal{F}_1 \mid R_1]$  over  $R_1 \cup R_2$  occurs. We will show that such a chain can be translated in an chain of  $[\mathcal{F}_1 \mid R_1]$  over  $R_1 \cup \pi$ . Thus, we can obtain an infinite chain of  $[\mathcal{F}_1 \mid R_1]$  over  $R_1 \cup \pi$ , that is, a infinite chain of  $[\mathcal{F}_1 \cup \{G\} \mid R_1 \cup \pi]$  over  $R_1 \cup \pi$ , the existence of which contradicts first premise of Theorem 1:  $\mathcal{C}_g$ -termination of  $R_1$ .

That proof mainly consists of an application of Lemma 2 with  $R_1 = S_1 = S_2$  and  $R_2 = S_3$ .

Thus, for any infinite dependency chain of  $[\mathcal{F}_1 \mid R_1]$  over  $R_1 \cup R_2$ , we can build a corresponding (infinite) chain of  $[\mathcal{F}_1 \mid R_1]$  over  $R_1 \cup \pi$ , that is, an infinite dependency chain of  $R_1 \cup \pi$ . Since we supposed  $R_1$   $\mathcal{C}_g$ -terminating, such a chain raises a contradiction.

This ends the proof of Theorem 1. □

*Prerequisites of proof of Lemma 2.* The proof sketch is as follows. To get rid of symbols of  $\mathcal{F}_2$ , we provide an interpretation  $I$  of terms (see Definition 7). Then we prove that this interpretation is sufficient for our purpose: first it is well defined (Lemmas 3, 4, and 5), and second we may “simulate” any  $S_1 \cup S_2 \cup S_3$  step with (a finite number of)  $S_1 \cup S_2 \cup \pi$  steps (Lemmas 6 and 7).

Eventually, we may build a suitable infinite dependency chain of  $[\mathcal{F}_1 \mid S_2]$  over  $S_1 \cup S_2 \cup \pi$  from any infinite minimal dependency chain of  $[\mathcal{F}_1 \mid S_2]$  over  $S_1 \cup S_2 \cup S_3$  with substitution  $\sigma$  by keeping the same pairs, and using substitution  $\sigma'$  such that for all  $x$ ,  $x\sigma' = I(x\sigma)$ .  $\square$

We use an interpretation of terms akin to Gramlich's [16]. The main difference between our interpretation and Gramlich's  $\Phi$  is related to the definition of  $\text{Red}(t)$ . We actually take *all one-step reductions* into account while function  $\text{SUCC}^{\mathcal{F}_1}(t)$  occurring in  $\Phi$  selects from terms obtained in *reductions* by  $\rightarrow_s^*$  the ones whose root symbol belongs to  $\mathcal{F}_1$ .

This interacts with the sizes of terms, bigger in our case, and would result in other proofs, since interpretations of two terms  $s$  and  $t$  might be identical if more than one-step reductions are considered. Thus, using Gramlich's  $\Phi$ , we would end with

$$\Phi(s) \xrightarrow[S_1 \cup S_2 \cup \pi]^* \Phi(t)$$

instead of

$$\Phi(s) \xrightarrow[S_1 \cup S_2 \cup \pi]^+ \Phi(t)$$

as conclusions of main lemmas. Further note that even with these changes, Theorems 1 and 2 would be proven in a similar manner. This ends the sketch of the proof.

We expose now the prerequisite of the proof of Lemma 2: we define the interpretation and state some useful lemmas.

We denote  $T_\infty(\mathcal{F}, X)$  the set of infinite terms over signature  $\mathcal{F}$  and variables set  $X$ .

**DEFINITION 7.** Let us denote  $S = S_1 \cup S_2 \cup S_3$ , and let  $>$  be an arbitrary *total* ordering over  $T_\infty(\widehat{\mathcal{F}}_1 \cup \{G : 2\} \cup \{\perp : 0\}, X)$ .

Interpretation  $I(x) : T(\widehat{\mathcal{F}}_1 \cup \mathcal{F}_2, X) \rightarrow T_\infty(\widehat{\mathcal{F}}_1 \cup \{G : 2\} \cup \{\perp : 0\}, X)$  is defined as follows:

$$I(x) = x \quad \text{if } x \in X,$$

$$I(f(t_1 \dots t_n)) = \begin{cases} f(I(t_1), \dots, I(t_n)) & \text{if } f \in \widehat{\mathcal{F}}_1, \\ \text{Comb}(\text{Red}(f(t_1, \dots, t_n))) & \text{if } f \in \mathcal{F}_2, \end{cases}$$

where

$$\text{Red}(t) = \{I(t') \mid t \xrightarrow[S_1 \cup S_2 \cup S_3]{} t'\},$$

$$\text{Comb}(\emptyset) = \perp,$$

$$\text{Comb}(\{a\} \cup E) = G(a, \text{Comb}(E)), \quad \text{where for all } e \in E, a < e.$$



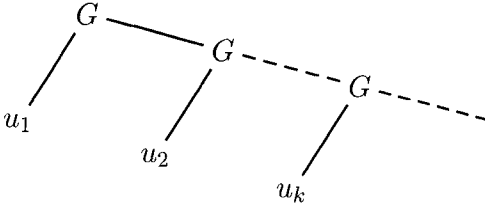


Figure 3. Structure of interpreted terms.

$\text{Red}(t)$  denotes the set  $E$  of interpreted one-step-reducts of  $t$ . To avoid any ambiguity on the actual construction of tree  $\text{Comb}(E)$  from  $E$  (nonordered set), we need a total ordering  $>$  providing a building strategy.

*Remark 4.* The interpretation of a term  $t = f(t_1, \dots, t_n)$  where  $f \in \mathcal{F}_2$  is, as Figure 3 illustrates, a sequence of its one-step-reducts interpretations, each  $u_i$  being an element of  $\text{Red}(t)$ . Since those are interpretations themselves, it is possible to reach any of them by using a suitable  $\xrightarrow[\pi_2]{*} \xrightarrow[\pi_1]$  reduction.

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

LEMMA 3. For each  $t \in T(\widehat{\mathcal{F}}_1, X)$  and each substitution  $\sigma$ ,

$$I(t\sigma) = tI(\sigma).$$

*Proof.* Structural induction on  $t$ . □

LEMMA 4. For all  $t_1, \dots, t_n$  of  $T(\widehat{\mathcal{F}}_1 \cup \mathcal{F}_2, X)$  and for any context  $C$  over  $\mathcal{F}_1$  with  $n$  holes,

$$I(C[t_1, \dots, t_n]) = C[I(t_1), \dots, I(t_n)].$$

*Proof.* Structural induction on  $C$ . □

LEMMA 5. For each term  $t$  strongly normalizable for  $S_1 \cup S_2 \cup S_3$ ,  $I(t)$  is finite.

*Proof.* Immediate, since we are interested in finitely branching systems only. □

We state now the two fundamental lemmas for the proof of Lemma 2.

LEMMA 6. For all  $s$  and  $t$  in  $T(\widehat{\mathcal{F}}_1 \cup \mathcal{F}_2, X)$  and each rule  $l \rightarrow r \in S_1 \cup S_2$ ,

$$\text{if } s \xrightarrow[l \rightarrow r]{p} t, \quad \text{then } I(s) \xrightarrow[S_1 \cup S_2 \cup \pi]{+} I(t).$$

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

*Proof.* Two cases depending on symbols occurring on path from  $\Lambda$  to  $p$ .

1. If there are only symbols of  $\mathcal{F}_1$ , then  $s = C[s_1, \dots, l\sigma, \dots, s_n]$ ,  $s|_p = l\sigma$  and  $C$  is a context with  $n$  holes over  $\mathcal{F}_1$ . We have

$$\begin{aligned}
 I(s) &= I(C[s_1, \dots, l\sigma, \dots, s_n]) \\
 &= C[I(s_1), \dots, I(l\sigma), \dots, I(s_n)] \quad (\text{Lemma 4}) \\
 &= C[I(s_1), \dots, lI(\sigma), \dots, I(s_n)] \quad (\text{Lemma 3}) \\
 &\xrightarrow[p]{s_1 \cup s_2} C[I(s_1), \dots, rI(\sigma), \dots, I(s_n)] \quad (\text{Premises}) \\
 &= C[I(s_1), \dots, I(r\sigma), \dots, I(s_n)] \quad (\text{Lemma 3}) \\
 &= I(C[s_1, \dots, r\sigma, \dots, s_n]) \quad (\text{Lemma 4}) \\
 &= I(t).
 \end{aligned}$$

2. If symbols of  $\mathcal{F}_2$  occur, then there is a smallest  $p' < p$  (with reference to the prefix ordering) such that  $\Lambda(s|_{p'}) \in \mathcal{F}_2$ . We may again assume (without any loss of generality) that  $s = C[s_1, \dots, s', \dots, s_n]$ , where  $C$  is a context with  $n$  holes (possibly empty) over  $\mathcal{F}_1$ ,  $p = p'q$  and  $s|_{p'} = s'$  with  $s' = C'[l\sigma] \xrightarrow[p]{s_1 \cup s_2} C'[r\sigma] = t'$ . Hence,

$$\begin{aligned}
 I(s) &= I(C[s_1, \dots, s', \dots, s_n]) \\
 &= C[I(s_1), \dots, I(s'), \dots, I(s_n)] \quad (\text{Lemma 4}).
 \end{aligned}$$

From Definition 7,  $I(s') = \text{Comb}(\text{Red}(s'))$ . But

$$s'|_q = l\sigma \xrightarrow[p]{s_1 \cup s_2} r\sigma.$$

We can then deduce from definition of  $\text{Red}$ :  $I(r\sigma) \in \text{Red}(l\sigma)$ . Thus,  $I(t')$  is a subterm of  $I(s')$  and

$$I(s') \xrightarrow[\pi]^+ I(t').$$

Eventually,

$$\begin{aligned}
 C[I(s_1), \dots, I(s'), \dots, I(s_n)] &\xrightarrow[\pi]^+ C[I(s_1), \dots, I(t'), \dots, I(s_n)] \\
 &= I(C[s_1, \dots, t', \dots, s_n]) \quad (\text{Lemma 4}) \\
 &= I(t).
 \end{aligned}$$

□

**LEMMA 7.** For all  $s$  and  $t$  in  $T(\widehat{\mathcal{F}}_1 \cup \mathcal{F}_2, X)$ , if  $s \xrightarrow[p]{S_3} t$ , then  $I(s) \xrightarrow[\pi]^+ I(t)$ .

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

*Proof.* Similar to case 2 in proof of Lemma 6. □

We may now tackle the actual proof of Lemma 2.

*Proof of Lemma 2.* Let  $\langle u_1, v_1 \rangle, \langle u_2, v_2 \rangle, \dots$  be an infinite minimal dependency chain of  $[\mathcal{F}_1 \mid \mathcal{S}_2]$  over  $\mathcal{S}_1 \cup \mathcal{S}_2 \cup \mathcal{S}_3$  with a substitution  $\sigma$ . Let  $\sigma'$  be the substitution such that for all  $x$ ,  $x\sigma' = I(x\sigma)$ .

Substitution  $\sigma$  is strongly normalizable since the considered chain is minimal. Then from Lemma 5 we know that  $\sigma'$  substitutes finite terms only.

We are going to show that  $\langle u_1, v_1 \rangle, \langle u_2, v_2 \rangle, \dots$  together with  $\sigma'$  is actually a dependency chain of  $[\mathcal{F}_1 \mid \mathcal{S}_2]$  over  $\mathcal{S}_1 \cup \mathcal{S}_2 \cup \pi$ .

To do that, we have to prove that for all  $i$ ,

$$v_i\sigma' \xrightarrow[\mathcal{S}_1 \cup \mathcal{S}_2 \cup \pi]{\neq \Lambda}^* u_{i+1}\sigma'.$$

We know that

$$v_i\sigma \xrightarrow[\mathcal{S}_1 \cup \mathcal{S}_2 \cup \mathcal{S}_3]{\neq \Lambda}^* u_{i+1}\sigma.$$

Let us consider a step  $s \xrightarrow[\mathcal{S}_1 \cup \mathcal{S}_2 \cup \mathcal{S}_3]{p} t$ . Since

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

then from Lemma 7 or from Lemma 6 we have

$$I(s) \xrightarrow[\mathcal{S}_1 \cup \mathcal{S}_2 \cup \pi]{\neq \Lambda}^* I(t).$$

We may build the expected sequence step by step in order to obtain

$$I(v_i\sigma) \xrightarrow[\mathcal{S}_1 \cup \mathcal{S}_2 \cup \pi]{\neq \Lambda}^* I(u_{i+1}\sigma).$$

Since  $I(v_i\sigma) = v_i\sigma'$  and  $I(u_{i+1}\sigma) = u_{i+1}\sigma'$ , we conclude by Lemma 3.

This ends the proof of Lemma 2. □

### *Corollary and Independent Extensions*

By considering the extension of  $R_1$  with  $[\mathcal{F}_2 \cup \{G : 2\} \mid R_2 \cup \pi]$  and with the help of Theorem 1, we obtain as a corollary a sufficient condition for proving  $\mathcal{C}_\varepsilon$ -termination of the extension itself.

**COROLLARY 1.** *Let  $[\mathcal{F}_1 \mid R_1] \longleftarrow [\mathcal{F}_2 \mid R_2]$  be a hierarchical extension of  $R_1(\mathcal{F}_1)$ ; if*

- (1)  $R_1$  is  $\mathcal{C}_\varepsilon$ -terminating, and
- (2) there is no infinite dependency chain of  $[\mathcal{F}_2 \mid R_2]$  over  $R_1 \cup R_2 \cup \pi$ ,

*then  $R_1 \cup R_2$  is  $\mathcal{C}_\varepsilon$ -terminating.*

We may compose applications of this corollary in order to perform a termination proof in an incremental fashion, by proceeding from the base to the top of the modules hierarchy.

**THEOREM 2.** *Let  $[\mathcal{F}_1 \mid R_1] \leftarrow [\mathcal{F}_2 \mid R_2]$  be a hierarchical extension of  $R_1(\mathcal{F}_1)$ , and let  $[\mathcal{F}_3 \mid R_3]$  be a module extending  $R_1$  independently of  $R_2$ . If*

- (1)  $R_1 \cup R_2$  is  $\mathcal{C}_\varepsilon$ -terminating, and
- (2) there is no infinite dependency chain of  $[\mathcal{F}_3 \mid R_3]$  over  $R_1 \cup R_3 \cup \pi$ ,

then  $R_1 \cup R_2 \cup R_3$  is  $\mathcal{C}_\varepsilon$ -terminating.

*Proof.* Let us suppose that there is an infinite dependency chain of  $R_1 \cup R_2 \cup R_3 \cup \pi$ . We are going to show that in such a case, we may conclude either on non- $\mathcal{C}_\varepsilon$ -termination of  $R_1 \cup R_2$ , thus contradicting first premise, or on the existence of an infinite relative dependency chain of module  $[\mathcal{F}_3 \mid R_3]$  over  $R_1 \cup R_3 \cup \pi$ , a contradiction to the second premise.

Following from the definition of hierarchical extensions and by Lemma 1, we know that chains of  $R_1 \cup R_2 \cup R_3 \cup \pi$  are

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

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

- There is no infinite chain of  $[\mathcal{F}_3 \mid R_3]$  over  $R$ . Otherwise, by Lemma 2 with
  - $S_1 = R_1$ ,
  - $S_2 = R_3 \cup \pi$ ,
  - $S_3 = R_2$ ,
  - $\mathcal{F}_1 = \mathcal{F}_1 \cup \mathcal{F}_3$  and  $\mathcal{F}_2 = \mathcal{F}_2$ ,

we would end with an infinite chain of  $[\mathcal{F}_3 \mid R_3]$  over  $R_1 \cup R_3 \cup \pi$ . But all those are finite from the premises.

- There is no infinite chain of  $[\mathcal{F}_1 \cup \mathcal{F}_2 \mid R_1 \cup R_2]$  over  $R$ . Otherwise, applying Lemma 2 with
  - $S_1 = \emptyset$ ,
  - $S_2 = R_1 \cup R_2$ , and
  - $S_3 = R_3 \cup \pi$ ,

we would end with an infinite chain of  $[\mathcal{F}_1 \cup \mathcal{F}_2 \mid R_1 \cup R_2]$  over  $R_1 \cup R_2 \cup \pi$ . But these chains are all finite because  $R_1 \cup R_2$   $\mathcal{C}_\varepsilon$ -terminates from premises.

Hence,  $R_1 \cup R_2 \cup R_3$  is  $\mathcal{C}_\varepsilon$ -terminating. □

*Remark 5.* The crucial point in Theorem 2 is that *no premise bounds  $R_2$  and  $R_3$  together*. Thus, a proof can actually be performed in an incremental and modular fashion.

Further note that  $\mathcal{C}_\varepsilon$ -termination of  $R_1 \cup R_2$  is necessary: we may otherwise encounter Toyama’s counterexample by choosing

- $R_1 = \emptyset$ ,
- $R_2 = \{f(0, 1, x) \rightarrow f(x, x, x)\}$  and
- $R_3 = \pi$ .

Eventually, by Proposition 2 we have as a corollary of Theorem 2 a previous result by Kurihara and Ohuchi [23]:  $\mathcal{C}_\varepsilon$ -termination is a modular property for unions of composable TRSs.

Theorem 1 is clearly an incremental result, while Theorem 2 is a modular one: irrelevant rules (those in  $R_2$ ) do not interfere.

## 5. Proving Termination

Termination is usually proven by using appropriate well-founded orderings. We propose in this section a class of orderings ( $\pi$ -expandable orderings) that are well suited for  $\mathcal{C}_\varepsilon$ -termination, that is, take care of the possible projective behavior of an additional set of rules. Using these orderings, we obtain corollaries of Theorems 1 and 2 that provide effective methods for incremental/modular termination proof.

### 5.1. $\pi$ -EXPANDABLE ORDERINGS

**DEFINITION 8.** A term ordering  $(\succeq, >)$  over  $T(\mathcal{F}, X)$  is said to be  $\pi$ -expandable if there is a reduction ordering  $(\succeq', >')$  over  $T(\mathcal{F} \cup \{G : 2\}, X)$  such that

- $(\succeq', >')$  restricted to  $T(\mathcal{F}, X)$  is exactly  $(\succeq, >)$ ;
- $G(s, t) \succeq' s$  and  $G(s, t) \succeq' t$  for all  $s$  and  $t$  in  $T(\mathcal{F}, X)$ .

We say that such a suitable  $(\succeq', >')$  is an *associate ordering* of  $(\succeq, >)$ .

$\pi$ -expandable orderings may be used for proving  $\mathcal{C}_\varepsilon$ -termination.

**PROPOSITION 3.** *Let  $(\succeq, >)$  be a strictly monotonic  $\pi$ -expandable ordering. If for each rule  $l \rightarrow r$  of a term rewriting system  $R$ ,  $l > r$ , then  $R$  is  $\mathcal{C}_\varepsilon$ -terminating.*

Clearly, any simplification ordering is  $\pi$ -expandable. In particular, RPO and the orderings induced by polynomial interpretations are  $\pi$ -expandable. We may indeed combine them so as to obtain some new ones.

#### *Lexicographical Compositions*

In particular, lexicographical compositions may be useful for building  $\pi$ -expandable orderings.

**DEFINITION 9.** Let  $(\succeq_1, >_1)$  and  $(\succeq_2, >_2)$  be term orderings. The *lexicographical composition*  $((\succeq_1, >_1), (\succeq_2, >_2))_{lex}$  of  $(\succeq_1, >_1)$  and  $(\succeq_2, >_2)$  is a pair  $(\succeq, >)$  such that

- $s > t$  iff  $s >_1 t$  or  $s \succeq_1 t$  and  $s >_2 t$ ;
- $s \succeq t$  iff  $s >_1 t$  or  $s \succeq_1 t$  and  $s \succeq_2 t$ .

It is easily shown that the lexicographical composition of two term orderings is itself a term ordering.

**PROPOSITION 4.** *If  $(\succeq_1, >_1)$ ,  $(\succeq_2, >_2)$  are  $\pi$ -expandable orderings with  $(\succeq_1, >_1)$  strictly monotonic, then the lexicographical composition  $((\succeq_1, >_1), (\succeq_2, >_2))_{lex}$  is a  $\pi$ -expandable ordering.*

*Moreover, if  $(\succeq_2, >_2)$  is strictly monotonic itself, then so is the composition  $((\succeq_1, >_1), (\succeq_2, >_2))_{lex}$ .*

*Proof.* We have to find a suitable  $(\succeq', >')$  satisfying the two conditions of Definition 8. Let us suppose that  $(\succeq'_1, >'_1)$ , strictly monotonic, and  $(\succeq'_2, >'_2)$  are associate orderings to, respectively,  $(\succeq_1, >_1)$  and  $(\succeq_2, >_2)$ . We may verify that  $(\succeq', >') = ((\succeq'_1, >'_1), (\succeq'_2, >'_2))_{lex}$  is suitable, strict monotonicity of  $(\succeq'_1, >'_1)$  implying the (weak) monotonicity of the composition.

We need to show

1. correct comparison of  $G(s, t)$  and  $s$  for all  $s$  and  $t$ ;
2. correct comparison of  $G(s, t)$  and  $t$  for all  $s$  and  $t$ ;
3. equality of  $(\succeq', >')$  restricted to  $T(\mathcal{F}, X)$  and  $((\succeq_1, >_1), (\succeq_2, >_2))_{lex}$ .
  - $G(s, t) \succeq' s$ . Since  $(\succeq_1, >_1)$  is  $\pi$ -expandable, we know  $G(s, t) \succeq'_1 s$ ; similarly since  $(\succeq_2, >_2)$  is  $\pi$ -expandable, we know  $G(s, t) \succeq'_2 s$ , hence  $G(s, t) \succeq' s$ .
  - $G(s, t) \succeq' t$ . Similar to previous case.
  - $(\succeq', >')|_{T(\mathcal{F}, X)} \subseteq ((\succeq_1, >_1), (\succeq_2, >_2))_{lex}$ . Let us get into details for the  $s \succeq' t$  case for  $s$  and  $t \in T(\mathcal{F}, X)$ : Either  $s >'_1 t$  and from premises  $s >_1 t$ , or  $s \succeq'_1 t$  and we face two possibilities (1)  $s >'_2 t$  and then from premises  $s >_2 t$  or (2)  $s \succeq'_2 t$  and then from premises  $s \succeq_2 t$ .
  - $(\succeq', >')|_{T(\mathcal{F}, X)} \supseteq ((\succeq_1, >_1), (\succeq_2, >_2))_{lex}$ . Immediate.

Proof regarding  $>$  is similar. □

### *Recursive Program Schemes*

As well as lexicographical compositions, *recursive program schemes* [7, 20] may be used to construct  $\pi$ -expandable orderings.

**DEFINITION 10.** A *recursive program scheme* (RPS) is a term rewriting system such that

- each defined symbol appears at root position in only one rule, and
- each rule is of the form  $f(x_1, \dots, x_i, \dots, x_n) \rightarrow r$  where  $x_i$  are pairwise distinct variables and  $r$  is any term.

All RPSs are confluent, and their termination is decidable. All RPSs considered hereafter are supposed to be complete; in particular, we denote by  $t \downarrow_P$  the unique  $P$ -normal form of any term  $t$ .

**DEFINITION 11.** Given a term ordering  $(\succeq_1, >_1)$  and a RPS  $P$ , we define  $(\succeq, >) = (\succeq_1, >_1) \downarrow_P$  the following way:

- $s \succeq t$  iff  $s \downarrow_P \succeq_1 t \downarrow_P$  and
- $s > t$  iff  $s \downarrow_P >_1 t \downarrow_P$ .

Arts and Giesl have shown that if term ordering  $(\succeq_1, >_1)$  is weakly monotonic, then  $(\succeq_1, >_1) \downarrow_P$  is a weakly monotonic term ordering [2]. It is not strictly monotonic in general, even if  $(\succeq_1, >_1)$  is strictly monotonic.

**PROPOSITION 5.** *Let  $(\succeq, >)$  be a  $\pi$ -expandable ordering. If  $P$  is a recursive program scheme over  $\mathcal{F}$  such that  $G \notin \mathcal{F}$ , then  $(\succeq, >) \downarrow_P$  is a  $\pi$ -expandable ordering.*

*Proof.* Let  $(\succeq_1, >_1)$  be a  $\pi$ -expandable ordering, and let  $(\succeq, >)$  be defined by  $(\succeq, >) = (\succeq_1, >_1) \downarrow_P$ . We have to find an ordering  $(\succeq', >')$  satisfying the two conditions of Definition 8.

Let us suppose that  $(\succeq', >')$  is an associate ordering of  $(\succeq_1, >_1)$ . We shall prove that  $(\succeq', >')$  defined by  $s \succeq' t$  (resp.  $>'$ ) iff  $s \downarrow_P \succeq'_1 t \downarrow_P$  (resp.  $>'$ ) suits.

First, let us check that rules of  $\pi$  are oriented in a correct way. Since  $G(s \downarrow_P, t \downarrow_P) \succeq'_1 s \downarrow_P$  because  $\succeq_1$  is  $\pi$ -expandable, we have  $G(s, t) \succeq' s$  by definition. Comparison to  $t$  is checked similarly.

Second, let us show that  $(\succeq', >')|_{T(\mathcal{F}, X)} = (\succeq, >)$ .

- $(\succeq', >')|_{T(\mathcal{F}, X)} \subseteq (\succeq, >)$ . If  $s \succeq' t$ , we get  $s \downarrow_P \succeq'_1 t \downarrow_P$ . But  $G \notin \mathcal{F}$ ; hence  $s$  and  $t$  belong to  $T(\mathcal{F}, X)$ . Since  $\succeq_1$  is  $\pi$ -expandable, we know that  $s \downarrow_P \succeq_1 t \downarrow_P$ , that is,  $s \succeq t$ .
- $(\succeq', >')|_{T(\mathcal{F}, X)} \supseteq (\succeq, >)$ . If  $s \succeq t$ , then since  $G \notin \mathcal{F}$  we know that  $s$  and  $t$  belong to  $T(\mathcal{F}, X)$ . Hence  $s \downarrow_P \succeq_1 t \downarrow_P$  and  $s \downarrow_P \succeq'_1 t \downarrow_P$ , which means by definition  $s \succeq' t$ .

Proof regarding  $>$  is similar. □

## 5.2. METHODS FOR PROVING TERMINATION

From Proposition 2 we have a first test for termination.

**COROLLARY 2.** *Let  $(\succeq, >)$  be a (weakly monotonic)  $\pi$ -expandable ordering such that*

- (1)  $l \succeq r$  for each  $l \rightarrow r \in R$ , and
- (2)  $s > t$  for each  $\langle s, t \rangle \in \text{DP}(R)$ .

Then  $R$  is  $\mathcal{C}_g$ -terminating.

We obtain similarly effective corollaries to our theorems. A corollary of Theorem 1 is given first.

**COROLLARY 3.** *Let  $[\mathcal{F}_1 \mid R_1] \leftarrow [\mathcal{F}_2 \mid R_2]$  be a hierarchical extension of  $R_1(\mathcal{F}_1)$ ; if*

- (1)  $R_1$  is  $\mathcal{C}_g$ -terminating, and
- (2) there is a weakly monotonic reduction ordering (resp. weakly monotonic  $\pi$ -expandable)  $(\succeq, >)$  such that
  - $R_1 \cup R_2 \subseteq \succeq$  and
  - $\text{MDP}([\mathcal{F}_2 \mid R_2]) \subseteq >$ ,

then  $R_1 \cup R_2$  terminates (resp.  $\mathcal{C}_g$ -terminates).

*Proof.* By contradiction. Let us suppose the existence of an infinite chain. Since the ordering (weakly) decreases for each rewriting step and strictly decreases for each MDP step, there is a infinite sequence strictly decreasing for  $(\succeq, >)$ . This contradicts the well-foundedness of  $(\succeq, >)$ , and  $R_1 \cup R_2$  terminates by Theorem 1.

If  $(\succeq, >)$  is  $\pi$ -expandable, then  $R_1 \cup R_2 \cup \pi \subseteq \succeq$  and  $R_1 \cup R_2$   $\mathcal{C}_g$ -terminates by Corollary 1.  $\square$

Similarly,  $\pi$ -expandable orderings give us a simple way of using Theorem 2.

**COROLLARY 4.** *Let  $[\mathcal{F}_1 \mid R_1] \leftarrow [\mathcal{F}_2 \mid R_2]$  be a hierarchical extension of  $R_1(\mathcal{F}_1)$ , and let  $[\mathcal{F}_3 \mid R_3]$  be a module that extends  $R_1$  independently of  $[\mathcal{F}_2 \mid R_2]$ . If*

- (1)  $R_1 \cup R_2$  is  $\mathcal{C}_g$ -terminating, and
- (2) there is a weakly  $\pi$ -expandable ordering  $(\succeq, >)$  such that
  - $R_1 \cup R_3 \subseteq \succeq$  and
  - $\text{MDP}([\mathcal{F}_3 \mid R_3]) \subseteq >$ ,

then  $R_1 \cup R_2 \cup R_3$   $\mathcal{C}_g$ -terminates.

*Remark 6.* Optimizations with dependency graphs [2] may be applied to all results presented here [33].

In particular Arts and Giesl showed how to obtain an *estimation\** of the dependency graph, that is, a graph that contains the actual dependency graph. This approximation [2] is based on a replacement in dependency pairs of the proper

\* Middeldorp proposed another approximation [25], which is closer to the actual dependency graph (hence with more termination power) but slightly less simple to compute.



subterms whose root symbol are defined by fresh variables (the CAP operation). All those variables are made pairwise distinct (the REN operation); then the edges of the estimated graph are determined following unifiability of members of modified dependency pairs: there is an edge from  $\langle s_1, t_1 \rangle$  to  $\langle s_2, t_2 \rangle$  if  $s_2$  and  $\text{REN}(\text{CAP}(t_1))$  are unifiable.

We point out that system  $\pi$  adds arcs to the dependency graph but, *for Arts and Giesl's estimation*, the estimated dependency graph remains unmodified.

This last remark amounts to comparing to what can be obtained by using dependency graphs analysis only [1]. The latter method would lead to the same set of strict constraints coming from dependency pairs of module  $R_3$  after computation of strongly connected components, or even a smaller one because it does not take only the root symbol into account. Regarding nonstrict constraints, their set would be  $R_1 \cup R_2 \cup R_3 \subseteq \succeq$ , which contains additional constraints coming from rules of  $R_2$ , contrary to what happens with Corollary 4 whose (weak) requirements involve  $R_1 \cup R_3$  only.

Ending with fewer constraints over suitable orderings makes the discovery of one of them easier. Hence, applying the hierarchical criteria above, then using the dependency graph analysis on each module will be a significant improvement over previous approaches, as we shall illustrate in Section 6.2.

## 6. Modules and Automation

Thanks to their generality and to the purely syntactical tests, we implemented our methods in the termination toolbox of the CiME 2 system [5].

When it comes to automation of proofs, ordering constraints are the main problem. For large TRSs, even if rules are “simple,” their huge number is almost insurmountable for most solvers in reasonable time. If they are very strict, an automated search for a suitable ordering often fails.

Our results induce a significant decrease in the number (modularity with Theorem 2 and the corollaries) and in the strictness (incrementality with Theorem 1 and the corollaries) of these constraints, and such improvements show on termination proofs in practice.

For instance, CiME 2 took less than a second for finding all interpretations in Section 6.1, in a completely automated way, using the incremental and modular methods.

A catalogue of examples treated with CiME can be found in the CiME distribution [5].

### 6.1. EXAMPLE

We present in this section a complete example of an incremental proof using our results. We will actually extend Example 2. The complete hierarchy is illustrated by Figure 6 (page 340).

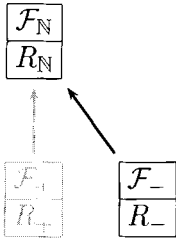


Figure 4. Independent extensions with  $[\mathcal{F}_+ | R_+]$  and  $[\mathcal{F}_- | R_-]$ . The proof of  $\mathcal{C}_\mathcal{E}$ -termination of the union uses  $\mathcal{C}_\mathcal{E}$ -termination of  $[\mathcal{F}_+ | R_+]$ , but that module does not interfere in ordering constraints: no constraint (weak or strong) arises from it.

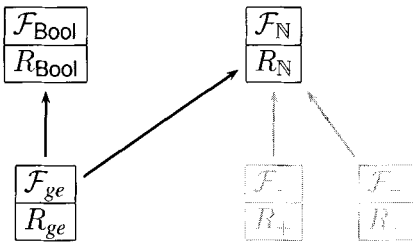


Figure 5. Arithmetical operators and comparisons are independent.

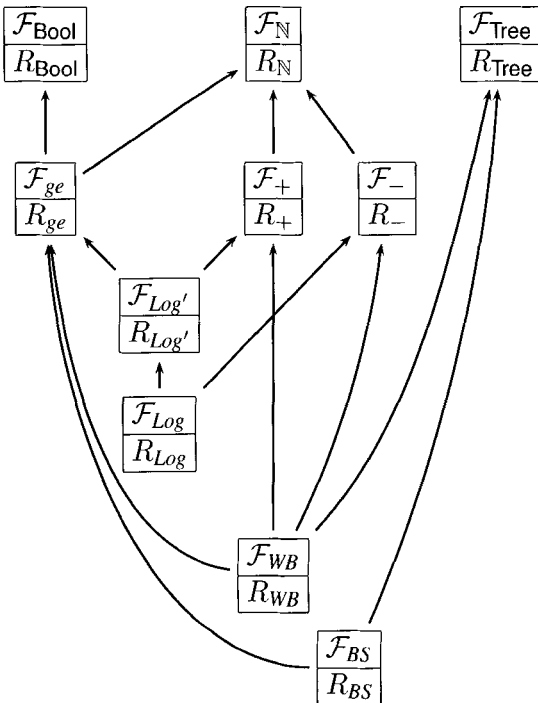


Figure 6. Modules hierarchy.

So let us consider the system describing natural numbers (which clearly  $\mathcal{C}_g$ -terminates):

$$\begin{aligned} \mathcal{F}_{\mathbb{N}}\{\# : \text{constant}; 1, 0 : \text{unary}, \\ R_{\mathbb{N}}\{\#0 \rightarrow \#\}. \end{aligned}$$

We already defined some arithmetic on these integers, in particular addition with  $[\mathcal{F}_+ \mid R_+]$  to which we add an associativity rule in order to make it overlapping:

$$\begin{aligned} \mathcal{F}_+\{+ : \text{infix binary}\} \\ R_+ \left\{ \begin{array}{l} x + \# \quad \rightarrow x \\ \# + x \quad \rightarrow x \\ x0 + y0 \quad \rightarrow (x + y)0 \\ x0 + y1 \quad \rightarrow (x + y)1 \\ x1 + y0 \quad \rightarrow (x + y)1 \\ x1 + y1 \quad \rightarrow ((x + y) + \#)0 \\ x + (y + z) \rightarrow (x + y) + z \end{array} \right. \end{aligned}$$

Termination of  $R_{\mathbb{N}} \cup R_+$  is proven using dependency pairs and a polynomial interpretation.

$$\text{MDP}([\mathcal{F}_+ \mid R_+]) : \left\{ \begin{array}{l} \langle x0 \hat{+} y0, x \hat{+} y \rangle \\ \langle x0 \hat{+} y1, x \hat{+} y \rangle \\ \langle x1 \hat{+} y0, x \hat{+} y \rangle \\ \langle x1 \hat{+} y1, x \hat{+} y \rangle \\ \langle x1 \hat{+} y1, (x + y) \hat{+} \# \rangle \\ \langle x \hat{+} (y + z), x \hat{+} y \rangle \\ \langle x \hat{+} (y + z), (x + y) \hat{+} z \rangle \end{array} \right\} \begin{array}{l} \llbracket \# \rrbracket = 0 \\ \llbracket 0 \rrbracket(x) = x + 1 \\ \llbracket 1 \rrbracket(x) = x + 2 \\ \llbracket + \rrbracket(x, y) = x + y + 1 \\ \llbracket \hat{+} \rrbracket(x, y) = x + 2y \end{array}$$

With reference to the ordering defined using the  $\pi$ -expandable interpretation above, pairs of  $\text{MDP}([\mathcal{F}_+ \mid R_+])$  strictly decrease while rules of  $R_{\mathbb{N}} \cup R_+ \cup \pi$  weakly decrease. Corollary 3 allows us to conclude on the  $\mathcal{C}_g$ -termination of  $R_{\mathbb{N}} \cup R_+$ .

We may want to perform subtraction.

$$\begin{aligned} \mathcal{F}_-\{- : \text{infix binary}\} \\ R_- \left\{ \begin{array}{l} x - \# \quad \rightarrow x \\ \# - x \quad \rightarrow \# \\ x0 - y0 \rightarrow (x - y)0 \\ x1 - y1 \rightarrow (x - y)0 \\ x1 - y0 \rightarrow (x - y)1 \\ x0 - y1 \rightarrow ((x - y) - \#)1 \end{array} \right. \end{aligned}$$

Again, dependency pairs of modules together with a polynomial interpretation are sufficient for showing that  $R_{\mathbb{N}} \cup R_-$   $\mathcal{C}_g$ -terminates. Indeed, for

$$\begin{aligned}
\llbracket \# \rrbracket &= 0 \\
\llbracket 0 \rrbracket(x) &= x + 1 \\
\llbracket 1 \rrbracket(x) &= x + 1 \\
\llbracket - \rrbracket(x, y) &= x \\
\llbracket \widehat{-} \rrbracket(x, y) &= x
\end{aligned}$$

pairs of  $[\mathcal{F}_- \mid R_-]$  strictly decrease while rules in  $R_{\mathbb{N}} \cup R_-$  weakly decrease. Applying Corollary 4, we obtain that  $R_{\mathbb{N}} \cup R_- \cup R_+$  is  $\mathcal{C}_g$ -terminating (cf. Figure 4).

To compare integers, we need Boolean operators. We add a new module, namely,  $[\mathcal{F}_{\text{Bool}} \mid R_{\text{Bool}}]$ .

$$\begin{aligned}
&\mathcal{F}_{\text{Bool}}\{\text{true}, \text{false} : \text{constant}; \neg : \text{unary}; \wedge : \text{infix binary}; \text{if} : \text{ternary}\} \\
R_{\text{Bool}} &\left\{ \begin{array}{ll} \neg(\text{true}) & \rightarrow \text{false} \\ \neg(\text{false}) & \rightarrow \text{true} \\ x \wedge \text{true} & \rightarrow x \\ x \wedge \text{false} & \rightarrow \text{false} \\ \text{if}(\text{true}, x, y) & \rightarrow x \\ \text{if}(\text{false}, x, y) & \rightarrow y \end{array} \right.
\end{aligned}$$

This system is dependency pairs free; hence it trivially  $\mathcal{C}_g$ -terminates.

We can now define a comparison in module  $[\mathcal{F}_{ge} \mid R_{ge}]$ , extending both  $R_{\mathbb{N}}$  and  $R_{\text{Bool}}$ .

$$\begin{aligned}
&\mathcal{F}_{ge}\{ge : \text{binary}\} \\
R_{ge} &\left\{ \begin{array}{ll} ge(x0, y0) & \rightarrow ge(x, y) \\ ge(x0, y1) & \rightarrow \neg ge(y, x) \\ ge(x1, y0) & \rightarrow ge(x, y) \\ ge(x1, y1) & \rightarrow ge(x, y) \\ ge(x, \#) & \rightarrow \text{true} \\ ge(\#, x0) & \rightarrow ge(\#, x) \\ ge(\#, x1) & \rightarrow \text{false} \end{array} \right.
\end{aligned}$$

Termination of  $R_{\mathbb{N}} \cup R_{\text{Bool}} \cup R_{ge}$  is shown by RPO with  $\{ge > \neg > (\text{true}, \text{false})\}$  directly. As a simplification ordering, RPO is  $\pi$ -expandable. Hence, the relevant union  $\mathcal{C}_g$ -terminates thanks to Proposition 3. We may then apply Theorem 2 and thus obtain  $\mathcal{C}_g$ -termination of  $R_{\mathbb{N}} \cup R_{\text{Bool}} \cup R_{ge} \cup R_+ \cup R_-$  (cf. Figure 5).

We add a new function over integers: base 2 logarithm rounded down. For technical reasons, it is easier to define first a  $Log'$  such that  $Log'(x) = Log(x) + 1$  with convention  $Log'(0) = 0$ .

$$\begin{aligned}
&\mathcal{F}_{Log'}\{Log' : \text{unary}\} \\
R_{Log'} &\left\{ \begin{array}{ll} Log'(\#) & \rightarrow \# \\ Log'(x1) & \rightarrow Log'(x) + \#1 \\ Log'(x0) & \rightarrow \text{if}(ge(x, \#1), Log'(x) + \#1, \#) \end{array} \right.
\end{aligned}$$

We use dependency pairs of modules and polynomial interpretations.

$$\begin{aligned} \text{MDP}([\mathcal{F}_{Log'} \mid R_{Log'}]) : & \left\{ \begin{array}{l} \langle \widehat{Log'}(x1), \widehat{Log'}(x) \rangle \\ \langle \widehat{Log'}(x0), \widehat{Log'}(x) \rangle \end{array} \right\} \\ \llbracket \# \rrbracket = & 0 \\ \llbracket 0 \rrbracket(x) = & x + 1 & \llbracket 1 \rrbracket(x) = & x + 1 & \llbracket + \rrbracket(x, y) = & x + y \\ \llbracket false \rrbracket = & 0 & \llbracket true \rrbracket = & 0 & \llbracket \neg \rrbracket(x) = & 0 \\ \llbracket ge \rrbracket(x) = & 0 & \llbracket if \rrbracket(x, y, z) = & y + z & \llbracket \wedge \rrbracket(x, y) = & x \\ \llbracket Log' \rrbracket(x) = & x & \llbracket \widehat{Log'} \rrbracket(x) = & x \end{aligned}$$

Dependency pairs strictly decrease while rules in  $R_{\mathbb{N}} \cup R_+ \cup R_{\text{Bool}} \cup R_{ge} \cup R_{Log'}$  weakly decrease. We may then apply Corollary 4 in order to show  $\mathcal{C}_{\mathcal{E}}$ -termination of  $R_{\mathbb{N}} \cup R_+ \cup R_{\text{Bool}} \cup R_{ge} \cup R_{Log'} \cup R_-$ .

The “correct” logarithm is computed by using module  $[\mathcal{F}_{Log} \mid R_{Log}]$ :

$$\begin{aligned} & \mathcal{F}_{Log}\{Log : unary\} \\ & R_{Log}\{Log(x) \rightarrow Log'(x) - \#1\} \end{aligned}$$

Since  $[\mathcal{F}_{Log} \mid R_{Log}]$  has no dependency pairs, we apply Theorem 2 and obtain  $\mathcal{C}_{\mathcal{E}}$ -termination of  $R_{\mathbb{N}} \cup R_+ \cup R_- \cup R_{\text{Bool}} \cup R_{ge} \cup R_{Log'} \cup R_{Log}$ .

Beside arithmetics we may want to work with binary trees over our integers. It suffices to define a module  $[\mathcal{F}_{Tree} \mid R_{Tree}]$  extending  $R_{\mathbb{N}}$ :

$$\begin{aligned} & \mathcal{F}_{Tree}\{\mathcal{L}, Val : unary; \mathcal{N} : ternary\} \\ & R_{Tree} \left\{ \begin{array}{l} Val(\mathcal{L}(x)) \rightarrow x \\ Val(\mathcal{N}(x, l, r)) \rightarrow x \end{array} \right. \end{aligned}$$

This module has no dependency pairs.

To test whether a tree is a binary search tree (BS), we introduce module  $[\mathcal{F}_{BS} \mid R_{BS}]$ , extending both  $R_{ge}$  and  $R_{Tree}$ :

$$\begin{aligned} & \mathcal{F}_{BS}\{BS, Min, Max : unary\} \\ & R_{BS} \left\{ \begin{array}{l} Min(\mathcal{L}(x)) \rightarrow x \\ Min(\mathcal{N}(x, l, r)) \rightarrow Min(l) \\ Max(\mathcal{L}(x)) \rightarrow x \\ Max(\mathcal{N}(x, l, r)) \rightarrow Max(r) \\ BS(\mathcal{L}(x)) \rightarrow true \\ BS(\mathcal{N}(x, l, r)) \rightarrow (ge(x, Max(l)) \wedge ge(Min(r), x)) \wedge \\ (BS(l) \wedge BS(r)) \end{array} \right. \end{aligned}$$

We have six dependency pairs for this module:

$$\text{MDP}([\mathcal{F}_{BS} \mid R_{BS}]) : \left\{ \begin{array}{l} \langle \widehat{Min}(\mathcal{N}(x, l, r)), \widehat{Min}(l) \rangle \\ \langle \widehat{Max}(\mathcal{N}(x, l, r)), \widehat{Max}(r) \rangle \\ \langle \widehat{BS}(\mathcal{N}(x, l, r)), \widehat{Max}(l) \rangle \\ \langle \widehat{BS}(\mathcal{N}(x, l, r)), \widehat{Min}(r) \rangle \\ \langle \widehat{BS}(\mathcal{N}(x, l, r)), \widehat{BS}(l) \rangle \\ \langle \widehat{BS}(\mathcal{N}(x, l, r)), \widehat{BS}(r) \rangle \end{array} \right\}$$

These strictly decrease while rules in  $R_{\mathbb{N}} \cup R_{\text{Bool}} \cup R_{ge} \cup R_{\text{Tree}} \cup R_{BS}$  weakly decrease w.r.t. polynomial interpretation

$$\begin{array}{lll}
\llbracket \# \rrbracket = 0 & \llbracket 0 \rrbracket(x) = 0 & \llbracket 1 \rrbracket(x) = 0 \\
\llbracket \text{false} \rrbracket = 0 & \llbracket \text{true} \rrbracket = 0 & \llbracket \neg \rrbracket(x) = 0 \\
\llbracket ge \rrbracket(x) = 0 & \llbracket \text{if} \rrbracket(x, y, z) = y + z & \llbracket \wedge \rrbracket(x, y) = x \\
\llbracket \mathcal{L} \rrbracket(x) = x & \llbracket \mathcal{N} \rrbracket(x, l, r) = x + l + r + 1 & \llbracket \text{Min} \rrbracket(x) = x \\
\llbracket \text{Max} \rrbracket(x) = x & \llbracket BS \rrbracket(x) = 0 & \llbracket \widehat{\text{Min}} \rrbracket(x) = x \\
\llbracket \widehat{\text{Max}} \rrbracket(x) = x & \llbracket \widehat{BS} \rrbracket(x) = x & \llbracket \text{Val} \rrbracket(x) = x
\end{array}$$

Thus,  $R_{\mathbb{N}} \cup R_{\text{Bool}} \cup R_{ge} \cup R_{\text{Tree}} \cup R_{BS}$   $\mathcal{C}_{\mathcal{E}}$ -terminates from Corollary 1.

Eventually, to decide whether a tree is well balanced (*WB*), that is, if the difference between sizes of left and right subtrees is at most 1, we have to compute sizes of trees.

$$\mathcal{F}_{WB}\{WB, \text{Size} : \text{unary}\}$$

$$R_{WB} \left\{ \begin{array}{l}
\text{Size}(\mathcal{L}(x)) \rightarrow \#1 \\
\text{Size}(\mathcal{N}(x, l, r)) \rightarrow (\text{Size}(l) + \text{Size}(r)) + \#1 \\
WB(\mathcal{L}(x)) \rightarrow \text{true} \\
WB(\mathcal{N}(x, l, r)) \rightarrow \text{if}(ge(\text{Size}(l), \text{Size}(r)), \\
\quad ge(\#1, \text{Size}(l) - \text{Size}(r)), \\
\quad ge(\#1, \text{Size}(r) - \text{Size}(l))) \\
\quad \wedge (WB(l) \wedge WB(r))
\end{array} \right.$$

The set of dependency pairs is the following:

$$\text{MDP}(\{\mathcal{F}_{BS} \mid R_{BS}\}) : \left\{ \begin{array}{l}
\langle \widehat{\text{Size}}(\mathcal{N}(x, l, r)), \widehat{\text{Size}}(l) \rangle \\
\langle \widehat{\text{Size}}(\mathcal{N}(x, l, r)), \widehat{\text{Size}}(r) \rangle \\
\langle \widehat{WB}(\mathcal{N}(x, l, r)), \widehat{\text{Size}}(l) \rangle \\
\langle \widehat{WB}(\mathcal{N}(x, l, r)), \widehat{\text{Size}}(r) \rangle \\
\langle \widehat{WB}(\mathcal{N}(x, l, r)), \widehat{WB}(l) \rangle \\
\langle \widehat{WB}(\mathcal{N}(x, l, r)), \widehat{WB}(r) \rangle
\end{array} \right\}$$

With help of the polynomial interpretation

$$\begin{array}{lll}
\llbracket \# \rrbracket = 0 & \llbracket 0 \rrbracket(x) = 0 & \llbracket 1 \rrbracket(x) = 0 \\
\llbracket + \rrbracket(x, y) = x + y & \llbracket - \rrbracket(x, y) = x & \llbracket \text{false} \rrbracket = 0 \\
\llbracket \text{true} \rrbracket = 0 & \llbracket \neg \rrbracket(x) = 0 & \llbracket ge \rrbracket(x) = 0 \\
\llbracket \text{if} \rrbracket(x, y, z) = y + z & \llbracket \wedge \rrbracket(x, y) = x & \llbracket \mathcal{L} \rrbracket(x) = x \\
\llbracket \text{Val} \rrbracket(x) = x & \llbracket \mathcal{N} \rrbracket(x, l, r) = x + l + r + 1 & \\
\llbracket \text{Size} \rrbracket(x) = 0 & \llbracket WB \rrbracket(x) = 0 & \\
\llbracket \widehat{\text{Size}} \rrbracket(x) = x & \llbracket \widehat{WB} \rrbracket(x) = x &
\end{array}$$

we may prove easily, simply using Corollary 4, that the union of *all* rules  $\mathcal{C}_{\mathcal{E}}$ -terminates. Please note how simple interpretations are.

## 6.2. COMPARISON WITH OTHER TECHNIQUES IN CiME

We emphasize in this section how our results empower termination proofs. Using the CiME 2 termination tool, we compare the time needed for termination proofs of some practical examples.

Tests are as follows. The system is provided as a single set of rules; then termination proofs are searched for with

- first, dependency pairs and graphs with modularity results from Arts and Giesl [1], denoted *DPQ*, since they are related to the notion of DP-(quasi) simple termination [14], and
- second, dependency pairs of modules (and related dependency graphs) over a hierarchy of minimal modules automatically obtained from the TRS.

The time of automated decomposition of TRSs into a relevant hierarchy is included in the proof search time.

Recall from Remark 6 that it is possible to use, for example, the DPQ criterion for each module.

Tests are performed on a computer equipped with a P-III 933 MHz processor and 1 GB RAM, running DEBIAN LINUX. We search successively for *Linear*, then *Simple*, polynomials w.r.t. Steinbach's notions [31]. Bounds refer to the maximum that coefficients in polynomials can reach. A search for a proof can be characterized by the conjunction of these restrictions; we denote it hereafter by pairing the kind of polynomials with the chosen bound.

*Fail* means that CiME 2 found no solution. *Abort* characterizes a computation interrupted after a time that seemed to us reasonably large: 48 hours.

Times given in brackets correspond to unnecessary expensive choices of bounds or polynomials.

### 6.2.1. The Log Example (31 rules)

Let us consider the TRS obtained from Section 6.1 by union

$$R_{\mathbb{N}} \cup R_{+} \cup R_{-} \cup R_{\text{Bool}} \cup R_{ge} \cup R_{Log'} \cup R_{Log}.$$

We point out that rule  $x + (y + z) \rightarrow (x + y) + z$ , describing associativity of  $+$ , makes the system overlapping: criteria based on innermost termination do not apply because innermost termination of an overlapping system does not imply its termination.

The decomposition leads to the hierarchy of 13 minimal modules that is described Figure 7, where

$$\begin{aligned} R_{-} &= \{\neg(\text{true}) \rightarrow \text{false}; \neg(\text{false}) \rightarrow \text{true}\} \\ R_{\wedge} &= \{x \wedge \text{true} \rightarrow x; x \wedge \text{false} \rightarrow \text{false}\} \quad \text{and} \\ R_{if} &= \{\text{if}(\text{true}, x, y) \rightarrow x; \text{if}(\text{false}, x, y) \rightarrow y\} \end{aligned}$$

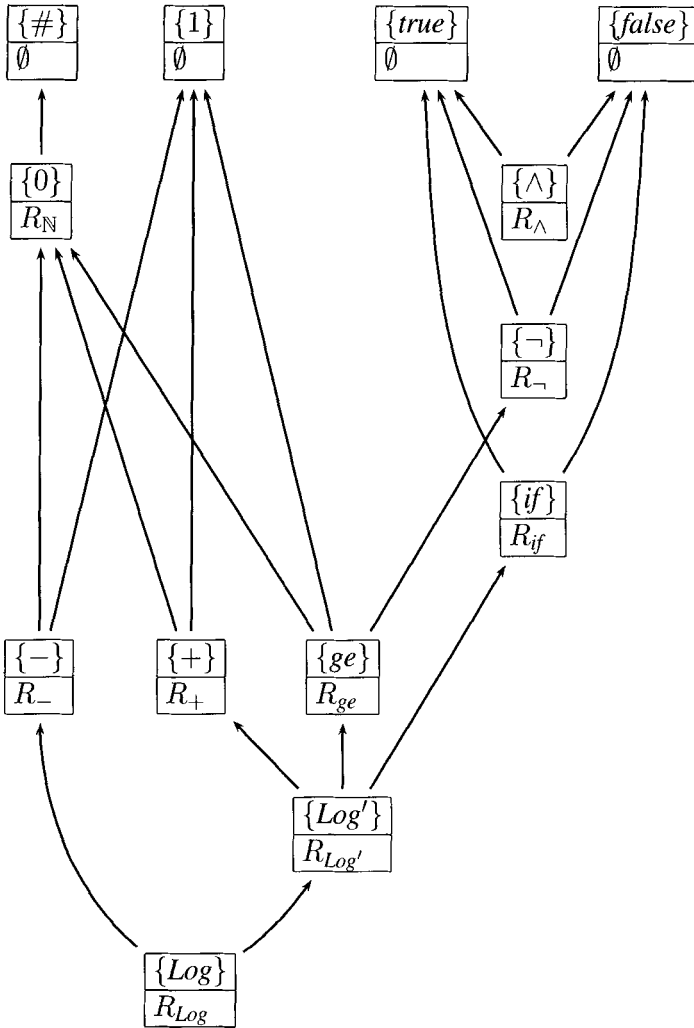


Figure 7. Hierarchy of minimal modules for the *Log* example.

Regarding constraints, we focus on the maximal number of termination constraints (TC, depending on criterion) and, since we are looking for orderings based on polynomial interpretations, on the maximal number of polynomial ordering constraints (POC, depending on parameters set for the search). Solving polynomial constraints can be complicated and tricky: for CiME it amounts to solving nonlinear Diophantine constraints the number of which may be worthy of consideration [6].

The modular approach leads to a maximum of 24 TC and 51 POC (for linear polynomials, same module) and succeeds, while DPQ criteria lead to a maximum of 38 TC and 88 POC (for linear polynomials, same subgraph) and fails.



Number of minimal modules in the hierarchy: 13.

Number of modules with no rule in the hierarchy: 4.

Polynomials and bounds	DPQ Time	Modules Time
Linear, 1	Fail	0.51 s
Linear, 2	Fail	0.35 s
Linear, 3	Fail	(0.35 s)
Linear, 6	Abort	(0.35 s)
Simple, 2	Fail	(15.08 s)
Simple, 3	Abort	(393 s)

The best choice here is clearly the modular approach. In particular, the fact that more easily solvable constraints are obtained with modules is worth mentioning here.

If we omit the associativity rule, previous approaches may end with a termination proof; linear polynomials are even suitable. Nevertheless, they require more than twice as much search time, at best.

Number of minimal modules in the hierarchy: 13.

Number of modules with no rule in the hierarchy: 4.

Polynomials and bounds	DPQ Time	Modules Time
Linear, 1	0.78 s	0.32 s
Linear, 2	0.82 s	0.35 s
Linear, 3	(82 s)	(0.35 s)
Linear, 6	(0.82 s)	(0.35 s)
Simple, 2	(30.3 s)	(3.96 s)
Simple, 3	(86.64 s)	(11.5 s)

### 6.2.2. The Tree Example (39 rules)

Results are similar when trying to prove termination of the TRS, which describes binary search trees and well-balanced trees. That is,

$$R_{\mathbb{N}} \cup R_{+} \cup R_{-} \cup R_{\text{Bool}} \cup R_{ge} \cup R_{\text{Tree}} \cup R_{BS} \cup R_{WB}.$$

Number of minimal modules in the hierarchy: 19.

Number of modules with no rule in the hierarchy: 6.

Polynomials and bounds	DPQ Time	Modules Time
Linear, 1	Fail	0.71 s
Linear, 2	5.7 s	0.6 s
Linear, 3	Abort	(0.6 s)
Linear, 6	Abort	(0.6 s)
Simple, 2	Abort	(Abort)
Simple, 3	Abort	(Abort)

Here again we notice that our modular approach requires lower bounds than does the DPQ approach.

### 6.2.3. A Sum and Product Example (35 rules)

One of the advantages of an incremental/modular approach is that different kinds of orderings may be used to prove termination, thus restraining expensive searches to modules that need them and using fast but less powerful criteria for the others.

We saw on previous examples that the search for a termination proof of the *Log* example ends successfully very quickly when looking for *linear* interpretations but lasts several minutes (depending on bounds) when looking for *simple* interpretations. Suppose that we add independent rules requiring simple interpretations. Then we obtain a TRS the termination of which may be very long to prove if the search is limited to simple polynomials.

Here is where incrementality and modularity play an important role: we may look for linear interpretations (with some bounds), and if a search ever fails on a module, we may switch *for that particular module only* to simple interpretations (with some other bounds when necessary).

Let us consider the following system combining the *Log* example and multiplication:

$$R_{\mathbb{N}} \cup R_{+} \cup R_{-} \cup R_{\text{Bool}} \cup R_{ge} \cup R_{Log'} \cup R_{Log} \cup R_{\times} \quad \text{with}$$

$$R_{\times} \left\{ \begin{array}{l} x \times \# \rightarrow \# \\ \# \times x \rightarrow \# \\ x0 \times y \rightarrow (x \times y)0 \\ x1 \times y \rightarrow (x \times y)0 + y \end{array} \right.$$

A search for a proof with our incremental/modular criteria ends with the following times.

[*param*<sub>1</sub>; *param*<sub>2</sub>; . . .] denotes the successive tries of parameters as follows: for each module, if no proof is found with parameters *param*<sub>1</sub>, then CiME looks for a proof with next parameters in the list (here *param*<sub>2</sub>), and so on.

Number of minimal modules in the hierarchy: 14.

Number of modules with no rule in the hierarchy: 4.

Parameters	Modules
Polynomials and bounds	Time
Simple, 2	8.08 s
[(Linear, 2); (Simple, 1)]	0.38 s

Comparison of times with other approaches in this particular case is useless because we know from Section 6.2.1 that the search would fail or would be interrupted. Nevertheless, regarding constraints, we point out that the modular approach leads to a maximum of 24 TC and 51 POC (for linear, then simple polynomials, same module) and succeeds, while DPQ criteria lead to a maximum of 42 TC and 136 POC (for linear, then simple polynomials, same subgraph) and fails.

Now if we omit the rule describing associativity, a previous approach may find a proof in a reasonable time. However, the modular approach is again clearly the best choice.

Number of minimal modules in the hierarchy: 14.

Number of modules with no rule in the hierarchy: 4.

Parameters	DPQ	Modules
Polynomials and bounds	Time	Time
Simple, 2	32.45 s	3.91 s
[(Linear, 2); (Simple, 1)]	10.65 s	0.4 s

#### 6.2.4. A Note on Communicating Processes

We also tried our results on large systems provided by Thomas Arts. Those systems come from a real application:  $\mu$ -CRL specifications of communicating processes. Roughly speaking, communicating processes can send a message or perform an action. Termination of such systems implies that an action is always performed after a finite lapse of time.

An attempt to prove termination of the 377 rules directly by means of polynomial interpretations failed. The system found no solution involving only linear interpretation; it had been looking for simple interpretations\* for more than ten days when we stopped the computer. But with our incremental/modular methods it took *less than two seconds* for CiME 2 to find a termination proof with only linear interpretations [33], by splitting up the TRS in a hierarchy of 74 modules (33 of which containing no rule). Regarding constraints, the modular approach led to a maximum of 309 TC and 558 POC (for linear polynomials, same module) and

\* *Simple* is here Steinbach's notion [31].

succeeded quickly, while DPQ criteria led to a maximum of 378 TC and 763 POC (for linear polynomials, same subgraph) and failed.

Note that our modular approach requires CiME solving Diophantine constraints problems only three of which consist of more than 225 constraints (maximum 1486 over 107 variables) while a DPQ approach requires solving a problem of 2334 constraints over 306 variables for which there is no convenient linear interpretation.

Automated termination proof for real programs is no longer out of reach. CiME 2 is available at <http://cime.lri.fr>

## 7. Hierarchical Simple Termination

We provide in this section a new notion of termination that is modular for unions of composable term rewriting systems. To obtain such modularity results, we distinguish relevant sets of rules by means of dependency relation over symbols  $\triangleright_d$  (see Definition 4).

**DEFINITION 12.** Let  $(\mathcal{F}, \mathcal{R})$  be a term rewriting system.

A rule  $l \rightarrow r$  is said to be *defining a symbol*  $f$  if that symbol occurs at the root position in  $l$ .

A rule  $l \rightarrow r$  is said to be *relevant w.r.t. a rule*  $l' \rightarrow r'$  if there is a symbol  $f$  occurring in  $l'$  or  $r'$  such that  $f \triangleright_d^* g$  where  $g$  is defined by  $l \rightarrow r$ .

As in Definition 4, the dependency relation can be represented as a graph whose vertices are symbols and such that there is an arc from  $f$  to  $g$  if and only if  $f \triangleright_d g$ . We can now define hierarchical simple termination.

**DEFINITION 13.** Let  $(\mathcal{F}, \mathcal{R})$  be a term rewriting system, and let  $\mathcal{G}$  be the graph of  $\triangleright_d$  over  $\mathcal{F}$ . We define sets  $C_i$  as strongly connected components of  $\mathcal{G}$  (see Definition 4).

A finitely branching term rewriting system  $(\mathcal{F}, \mathcal{R})$  is said to be *hierarchically simply terminating* (HST) if for each  $C_i$  there is an ordering pair  $(\succsim_i, >_i)$  that constitutes a well-founded simplification ordering and such that

- $s >_i t$  for all dependency pairs  $\langle s, t \rangle$ , where both root symbols of  $s$  and  $t$  occur in  $C_i$ ;
- $l \succsim_i r$  for all rules in  $\mathcal{R}$  relevant w.r.t. rules defining symbols in  $C_i$ .

**THEOREM 3.** *Hierarchical simple termination is a modular property of unions of composable finitely branching TRSs.*

*Proof.* Let  $R_1 = R'_1 \cup R_0$  and  $R_2 = R'_2 \cup R_0$  be two composable systems where  $R_0$  consists of rules defining symbols in common and such that both are hierarchically simply terminating. From the definition of composable systems we know that rules in  $R'_2$  are not relevant w.r.t. rules in  $R'_1 \cup R_0$  and that rules in  $R'_1$  are not relevant w.r.t. rules in  $R'_2 \cup R_0$ . Hence the suitable ordering pairs for hierarchical

simple termination of  $R_1$  and  $R_2$  may still be used so as to show hierarchical simple termination of  $R_1 \cup R_2$ .  $\square$

Hierarchical simple termination is clearly more general than simple termination because it uses dependency pairs. It also includes DP-simple termination strictly [14], thanks to the modular decomposition: DP-simple termination requires orderings that have to orient (weakly at least) all rules in a TRS, while the orderings involved in HST put constraints on relevant rules only. Hence any DP-simply terminating TRS is HS-terminating. The reciprocal does not hold.

EXAMPLE 5. The following system  $R$ , from Giesl and Ohlebusch [14], is not DP-simply terminating.

$$R \left\{ \begin{array}{lll} f(f(x)) \rightarrow f(c(f(x))) & g(c(x)) \rightarrow x & g(c(0)) \rightarrow g(d(1)) \\ f(f(x)) \rightarrow f(d(f(x))) & g(d(x)) \rightarrow x & g(c(1)) \rightarrow g(d(0)) \end{array} \right.$$

System  $R$  is, however, hierarchically simply terminating. Since the only (ordering) constraints are

$$\begin{array}{l} f(f(x)) \geq f(c(f(x))) \\ f(f(x)) \geq f(d(f(x))) \\ \text{and} \\ \widehat{f}(f(x)) > \widehat{f}(x) \end{array}$$

the simplification ordering induced by the following interpretation is enough.

$$\begin{array}{l} \llbracket c \rrbracket(x) = x \\ \llbracket d \rrbracket(x) = x \\ \llbracket f \rrbracket(x) = x + 1 \\ \llbracket \widehat{f} \rrbracket(x) = x \end{array}$$

Further note that CiME 2 is able to find an automated proof for that system, however, by using an ordering that is not a simplification ordering.

Similarly, each DP-quasi simply terminating TRS [14] is hierarchically simply terminating. We conjecture that the reciprocal does not hold.

## 8. Related Work and Conclusion

With the notion of *rewriting modules* (Definition 2), we defined a new framework very well suited to the study of the intrinsic hierarchical structure of term rewriting systems.

The framework of modules has many applications regarding termination proofs. In particular, with the help of *dependency pairs of modules* (Definition 5), we obtain powerful methods (Theorems 1 and 2 and corollaries) that allow proofs to be found incrementally and modularly.

Actually, the use of a concept of dependency pairs based on modules allows incremental and modular proving with state-of-the-art techniques and thus provides the most powerful approach for termination proof of systems with many rules known so far.

As explained below, our results apply to systems met in practice and are suitable for full automation, which was not really the case before either because of strong requirements on systems or unions or because criteria were too much *ad hoc*. They do not generate constraints for irrelevant sets of rules in incremental proving, which is definitely an improvement in the termination proof domain. Eventually a new class of terminating systems naturally emerges from them.

Our work has to compare with Dershowitz's results [9]. Since we do not restrict ourselves to constructor-based systems, and use a slightly more general definition of *hierarchical extensions*, we obtain more general conditions applicable to the systems one has to deal with in practical applications. Moreover, our criteria (fully syntactical and applicable to most TRSs met in practice) are more suited for automation – because they were designed for this purpose – than are the finely tuned conditions of Dershowitz. Similarly, Krishna Rao's proper extensions and restricted proper extensions [21, 22] are constrained modules extensions, and, thus, all our results apply to these.

Arts and Giesl exploit the modular structure of dependency graphs [1]. Their approach is fundamentally different from ours. They use an optimization, namely, dependency graphs, in order to *filter* strict constraints over dependency pairs, while in our case those are simply *not generated*. Moreover, their criterion keeps weak constraints over all rules for the whole system (whatever the extension might be). That is a drawback we wanted to get rid of, because it fundamentally acts as a break upon real incremental/modular proving. As noticed in Remark 5, criteria based on Theorem 2 do not require anything from irrelevant sets of rules, and hence no constraint (neither strict nor weak) comes from those. Our framework furthermore provides for the general case the powerful results they got for the special case of *innermost* rewriting: the only way they can avoid constraints from irrelevant rules is to require innermost rewriting and to use the notion of *usable rules*, while we can prove termination of systems for which innermost termination does not imply termination and without any constraint from irrelevant rules.

The modular approach combines several advantages. The constraints over suitable orderings we obtain from a modular analysis are less numerous at each step than in a “global” approach because only relevant rules are considered. They are also less restrictive, for the same reasons and because optimizations such as dependency graphs may be applied afterward at each step.

Being less numerous and weaker, ordering constraints are easier to solve in our approach. This is especially noticeable at the ordering level where, for instance, polynomial interpretations are much simpler than in a proof without modules, even if powerful methods such as dependency pairs with dependency graphs are used.

The notion of hierarchical simple termination (Section 7), which naturally appears in that framework, defines a class that contains the class of systems that are DP-quasi simply terminating and is stable by (composable) unions. We claim that this class represents most terminating systems from practical applications.

All these results are implemented in the CiME 2 rewriting tool. As illustrated by our benchmarks, they considerably enhance a completely automated search of termination proofs. A significant speedup may be observed on large examples, occurring in practice, and the verification of which is not any longer out of reach.

We point out that during the Termination Workshop WST'03 [29], none of the other tools in competition, such as TTT [19], APROVE [15], or TERMPTATION [4], was able to find termination proofs for the large systems that were easily shown terminating by CiME.\*

This work should extend in several directions and in particular toward extensions of rewriting. First, a quite important extension in practice regards rewriting modulo an equational theory. Some work has already been done on associativity and commutativity [34] and will be pursued.

Second, since this work aims at proving termination for TRSs of large size and practical applications, it is important to study how it can be applied to particular paradigms of programming such as conditional and/or constraint (i.e., with primitive types) rewriting, but also to rewriting with strategies [11, 12] like, for instance, context sensitive rewriting, which has received new interest recently [13, 18].

Finally, an extension of these results and methods to higher-order rewriting would lead to termination tools for functional programs, a crucial issue indeed in automated theorem proving.

## Acknowledgments

The author thanks Thomas Arts for the very interesting examples he gave. The author is grateful to Bernhard Gramlich and Claude Marché for judicious comments and discussions about this work and to Aart Middendorp and the anonymous referees for their fruitful remarks.

## References

1. Arts, T. and Giesl, J.: Modularity of termination using dependency pairs, in T. Nipkow (ed.), *9th International Conference on Rewriting Techniques and Applications*, Lecture Notes in Computer Science 1379, Tsukuba, Japan, 1998, pp. 226–240.
2. Arts, T. and Giesl, J.: Termination of term rewriting using dependency pairs, *Theoretical Computer Science* **236** (2000), 133–178.
3. Bralleras, C., Ferreira, M. and Rubio, A.: Complete monotonic semantic path orderings, in D. McAllester (ed.), *17th International Conference on Automated Deduction*, Lecture Notes in Computer Science 1831, Pittsburgh, PA, USA, 2000.

---

\* Note added in print.

4. Borralleras, C. and Rubio, A.: Termination, in A. Rubio (ed.), *Extended Abstracts of the 6th International Workshop on Termination, WST'03*, Technical Report DSIC II/15/03, Universidad Politécnic de Valencia, Spain, 2003, pp. 61–63.
5. Contejean, E., Marché, C., Monate, B. and Urbain, X.: CiME version 2, 2000. Available at <http://cime.lri.fr/>.
6. Contejean, E., Marché, C., Tomás, A.-P. and Urbain, X.: Mechanically proving termination using polynomial interpretations, Research Report 1382, LRI, 2004.
7. Courcelle, B.: Recursive applicative program schemes, in J. van Leeuwen (ed.), *Handbook of Theoretical Computer Science*, Vol. B, North-Holland, 1990, Chapt. 9, pp. 459–492.
8. Dershowitz, N.: Termination of rewriting, *J. Symbolic Comput.* **3**(1) (1987), 69–115.
9. Dershowitz, N.: Hierarchical termination, in N. Dershowitz and N. Lindenstrauss (eds.), *Proceedings of the Fourth International Workshop on Conditional and Typed Rewriting Systems (Jerusalem, Israel, July 1994)*, Vol. 968, Berlin, 1995, pp. 89–105.
10. Dershowitz, N. and Jouannaud, J.-P.: Notations for rewriting, *EATCS Bull.* **43** (1990), 162–172.
11. Fissore, O., Gnaedig, I. and Kirchner, H.: Termination of rewriting with local strategies, in M. Bonacina and B. Gramlich (eds.), *Selected Papers of the 4th International Workshop on Strategies in Automated Deduction*, Electronic Notes in Theoretical Computer Science 58, Siena, Italy, 2001.
12. Fissore, O., Gnaedig, I. and Kirchner, H.: Simplification and termination of strategies in rule-based languages, in *Proceedings of the Fifth International Conference on Principles and Practice of Declarative Programming*, Uppsala, Sweden, 2003, pp. 124–135.
13. Giesl, J. and Middeldorp, A.: Transforming context-sensitive rewrite systems, in P. Narendran and M. Rusinowitch (eds.), *Proceedings of the 10th International Conference on Rewriting Techniques and Applications*, Lecture Notes in Computer Science 1631, Trento, 1999, pp. 271–285.
14. Giesl, J. and Ohlebusch, E.: Pushing the frontiers of combining rewrite systems farther outwards, in *Proceedings of the Second International Workshop on Frontiers of Combining Systems (FroCoS '98)*, Studies in Logic and Computation 7, Amsterdam, The Netherlands, 2000, pp. 141–160.
15. Giesl, J., Thiemann, R., Schneider-Kamp, P. and Falke, S.: AProVE: A system for proving termination, in A. Rubio (ed.), *Extended Abstracts of the 6th International Workshop on Termination, WST'03*, Technical Report DSIC II/15/03, Universidad Politécnic de Valencia, Spain, 2003. <http://www-i2.informatik.rwth-aachen.de/AProVE>.
16. Gramlich, B.: Generalized sufficient conditions for modular termination of rewriting, *Applicable Algebra in Engineering, Communication and Computing* **5** (1994), 131–158.
17. Gramlich, B.: Abstract relations between restricted termination and confluence properties of rewrite systems, *Fund. Inform.* **24** (1995), 3–23.
18. Gramlich, B. and Lucas, S.: Simple termination of context-sensitive rewriting, in B. Fischer and E. Visser (eds.), *Proceedings of the 3rd ACM Sigplan Workshop on Rule-Based Programming, RULE'02*, Pittsburgh, PA, USA, 2002, pp. 29–41.
19. Hirokawa, N. and Middeldorp, A.: Tsukuba termination tool, in R. Nieuwenhuis (ed.), *14th International Conference on Rewriting Techniques and Applications*, Lecture Notes in Computer Science 2706, Valencia, Spain, 2003, pp. 311–320.
20. Klop, J. W.: Term rewriting systems, in S. Abramsky, D. Gabbay and T. Maibaum (eds.), *Handbook of Logic in Computer Science*, Vol. 2, Clarendon Press, 1992, pp. 1–116.
21. Krishna Rao, M. R. K.: Simple termination of hierarchical combinations of term rewriting systems, in *Theoretical Aspects of Computer Software*, Lecture Notes in Computer Science 789, 1994, pp. 203–223.
22. Krishna Rao, M. R. K.: Modular proofs for completeness of hierarchical term rewriting systems, *Theoretical Computer Science* **151** (1995), 487–512.



23. Kurihara, M. and Ohuchi, A.: Decomposable termination of composable term rewriting systems, *IEICE E78-D*(4) (1994), 314–320.
24. Kusakari, K., Nakamura, M. and Toyama, Y.: Argument filtering transformation, in G. Nadathur (ed.), *Principles and Practice of Declarative Programming, International Conference PDP'99*, Lecture Notes in Computer Science 1702, Paris, 1999, pp. 47–61.
25. Middeldorp, A.: Approximating dependency graphs using tree automata techniques, in R. Goré, A. Leitsch and T. Nipkow (eds.), *International Joint Conference on Automated Reasoning*, Lecture Notes in Artificial Intelligence 2083, Siena, Italy, 2001, pp. 593–610.
26. Middeldorp, A. and Toyama, Y.: Completeness of combinations of constructor systems, *J. Symbolic Comput.* **15** (1993), 331–348.
27. Ohlebusch, E.: On the modularity of termination of term rewriting systems, *Theoretical Computer Science* **136** (1994), 333–360.
28. Ohlebusch, E.: Modular properties of composable term rewriting systems, *J. Symbolic Comput.* **20** (1995), 1–41.
29. Rubio, A. (ed.): Extended Abstracts of the 6th International Workshop on Termination, WST'03, Technical Report DSIC II/15/03, Universidad Politécnic de Valencia, Spain, 2003.
30. Rusinowitch, M.: On termination of the direct sum of term rewriting systems, *Information Processing Letters* **26** (1987), 65–70.
31. Steinbach, J.: Generating polynomial orderings, *Information Processing Letters* **49** (1994), 85–93.
32. Toyama, Y.: Counterexamples to termination for the direct sum of term rewriting systems, *Information Processing Letters* **25** (1987), 141–143.
33. Urbain, X.: Approche incrémentale des preuves automatiques de terminaison, Thèse de doctorat, Université Paris-Sud, Orsay, France, 2001. <http://www.lri.fr/urbain/textes/these.ps.gz>.
34. Urbain, X.: Modular and incremental proofs of AC-termination, Research Report 1317, LRI, 2002.