Modular & 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

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. Our policy of restraining termination itself (thus relaxing constraints over hierarchies components) together with the generality of the module approach are sufficient to express previous results and methods the premisses 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 speed up in the finding of the proof.

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 since 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 (TRS).

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 of its possible modular structure.

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

Nevertheless, proving termination of a term rewriting system still remains very hard, especially when the system is large. This is due to the fact 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). In order to provide a significant improvement in proving termination automatically, we focus on two critical points: 1) Automating termination proofs and 2) 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 did not yet change 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 to guaranty termination of a union from termination of its components has actually to often been avoided by putting hard restrictions upon relations between components (see for instance Krishna Rao's *restricted proper extensions* [17, 16] or Dershowitz's *constructor based extensions* [6]). 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 [19], termination is not even modular for systems with disjoint signatures.

Example 1.

Let us consider Toyama's example.

$$R : \left\{ \; f(0,1,x) \;\; \rightarrow \;\; f(x,x,x) \right. \qquad\qquad \pi : \left\{ \begin{array}{lcl} G(x,y) & \rightarrow & x \\ G(x,y) & \rightarrow & y \end{array} \right.$$

Both systems R and π terminates. But if we consider their union,

$$\left\{ \begin{array}{lcl} G(x,y) & \rightarrow & x \\ G(x,y) & \rightarrow & y \\ f(0,1,x) & \rightarrow & f(x,x,x), \end{array} \right.$$

we may find an infinite reduction sequence, for instance:

$$\begin{array}{lcl} f(G(0,1),G(0,1),G(0,1)) & \rightarrow & f(0,G(0,1),G(0,1)) \\ & \rightarrow & f(0,1,G(0,1)) \\ & \rightarrow & f(G(0,1),G(0,1),G(0,1)) \rightarrow \ldots \end{array}$$

Instead of restraining the scope of our study by putting strong constraints upon unions, we rather allow weakly constrained unions (so-called *hierarchical*) and consider a (slightly) restricted notion of termination which is preserved under non-deterministic collapse, $\mathcal{C}_{\mathcal{E}}$-*termination*.

From an historical point of view, Rusinowitch [18] remarked that the infinite reduction of Example 1 was due to the projective behavior of system π. Gramlich [9] then defined the notion of *termination preserved under non-deterministic* *collapse* for which he had results. It was later renamed *Collapse Extended termination* ($\mathcal{C}_\mathcal{E}$-termination, for short) and studied by Ohlebusch [14].

A system R is $\mathcal{C}_\mathcal{E}$-terminating if and only if it is terminating and $R \uplus \pi$ (where G is a fresh symbol) is terminating.

We claim that focusing on $\mathcal{C}_\mathcal{E}$-termination is not too much of a restriction in practice. Actually, virtually all automated techniques for proving termination lead to $\mathcal{C}_\mathcal{E}$-termination, and amongst those, the powerful dependency pairs approach in particular. In other words, $\mathcal{C}_\mathcal{E}$-termination provides means for controlling the π-projection's possibly 'unproper' behavior with reference to termination of most systems occurring in practice and for which we want an automated proof.

Example 1 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 while actually programming an application.

Hierarchical structure

The common presentation of TRS is based on the union of sets of rules (over some signatures) which are TRS by themselves. This is not well suited to the expression of their inner hierarchical structure for at least the following two reasons: 1) It brings redundancy at the common subsystem definition level when symbols or rules are shared, and 2) It does not keep track of the actual sequential construction and thus does not enhance any incremental process (say for instance a termination proof). Both contribute to separating rewriting from actual programming.

In order to give a general framework bringing modular structure of TRS to the fore, and so as to provide automated methods to prove termination incrementally, we will firstly recall some generalities. Then in Sec. 3 we shall introduce *rewriting modules*. We will show how the intrinsic hierarchical structure of TRS naturally emerges from their formalization in terms of modules.

The framework of modules settled, *dependency pairs of modules* as well as *relative dependency chains* will be defined in Sec. 4 and these powerful tools will lead us to new criteria for incremental and modular termination (Theorem 1 and Theorem 2).

Section 5 will provide a class of orderings with good properties with respect to projection, hence to $\mathcal{C}_\mathcal{E}$-termination (the so-called *π-expandable orderings*).

Those results provide an incremental proof method illustrated with the complete example of Sec. 6.1 and a new notion of termination directly related to the incremental process: *Hierarchical Simple termination* (Sec. 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 C*i*ME 2 rewriting tool. With our incremental and modular criteria, C*i*ME 2 gives its user the possibility of searching termination proofs in a totally automated fashion. We tested it on TRS of approximately 400 rules with success: a termination proof is found in a few seconds. Examples came from a μ-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 since the incremental approach reduces termination constraints dramatically.

We will eventually discuss in Sec. 8 how this work compares to others, especially to results of Arts & Giesl [1] and Dershowitz [6].

2 Preliminaries

We recall usual notions about rewriting [7], and give our notations.

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 $\wedge$, 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 σ from variables to terms such that $\{x \in X \mid \sigma(x) \neq x\}$ is finite. We use postfix notation for substitution applications. A substitution can easily be extended to endomorphisms of $T(\mathcal{F}, X)$: $f(t_1, \ldots, t_n)\sigma = f(t_1\sigma, \ldots, t_n\sigma)$. Then, $t\sigma$ is called an *instance* of t.

A *rewrite relation* is a binary relation $\rightarrow$ on terms which is monotonic and closed under substitution; $\rightarrow^\star$ will denote its reflexive-transitive closure. A *term rewriting system* (TRS for short) over a signature[1] $\mathcal{F}$ and a set of variables X is a set $R(\mathcal{F})$ of *rewrite rules* $l \rightarrow r$. A TRS R defines a rewrite relation $\rightarrow_R$ 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 σ. We then say that $s|_p$ is a *redex* and that s *reduces to* t *at position* p *with* $l \rightarrow r$ or, if the rule of R is not relevant, *with* R, respectively denoted $s \xrightarrow[l\rightarrow r]{p} t$ and $s \xrightarrow[R]{p} t$.

Unfortunately, $\mathcal{C}_\mathcal{E}$-termination is not modular for constructor sharing unions with non-finitely branching TRS, as shown by Ohlebusch [14], hence we restrict our study to the *finitely branching* case, i.e. such that the set of rules that can be applied to a term is always finite.

A term is *strongly normalizable* (SN) if it cannot reduce indefinitely. A rewrite relation is *strongly normalizing* or *terminates* if any term is SN. In this paper, we will use indifferently 'strongly normalizing' and 'terminating'.

Termination is usually proven with the help of *reduction orderings* [5] or quasi-orderings with dependency pairs. We briefly recall what we need. A *term ordering*, also known as *ordering pair* [11] is a pair $(\succeq, \succ)$ of relations over $T(\mathcal{F}, X)$ such that: $\succeq$ is a quasi-ordering, i.e. reflexive and transitive; $\succ \subseteq \succeq - \preceq$ is a strict ordering, i.e. irreflexive and transitive and such that $\succ \subseteq \succeq$, $\succ \cdot \succeq \subseteq \succ$ and $\succeq \cdot \succ \subseteq \succ$. A term ordering is said to be *well-founded* if there is no infinite strictly

[1]We shall omit the signature if there is no ambiguity.

decreasing sequence $t_1 \succ t_2 \succ \cdots$; *stable* if both $\succ$ and $\succeq$ are stable under substitutions; *weakly* (resp. *strictly*) *monotonic* if for all terms t_1 and t_2, for all $f \in \mathcal{F}$, if $t_1 \succeq$ (resp. $\succ$) t_2 then $f(\ldots, t_1, \ldots) \succeq$ (resp. $\succ$) $f(\ldots, t_2, \ldots)$; A term ordering $(\succeq, \succ)$ is called a *weak (resp. strict) reduction ordering* if it is well-founded, stable and weakly (resp. strictly) monotonic; *a simplification ordering* if it is stable, monotonic and has the subterm property.

We distinguish between modular and incremental proofs. A modular property $\mathcal{P}$ behaves as follow: 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 in proving $\mathcal{P}'$ on a component C_1 firstly, then of showing $\mathcal{P}'$ on $C_1 \cup C_2$, possibly using the knowledge that C_1 verifies $\mathcal{P}'$.

$\mathcal{C}_\mathcal{E}$-termination

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

- disjoint TRS (Gramlich, Ohlebusch) [8, 14],
- finitely branching and constructor sharing systems (Gramlich) [8],
- finitely branching and composable systems (Kurihara & Ohuchi) [10].

Ohlebusch shown that the finitely branching condition is compulsory [14].

$\mathcal{C}_\mathcal{E}$-termination is less restrictive than simple termination: whenever adding projections for *all* symbols allows elimination of any constructors ‘hiding’ a redex, it is impossible with rules of π only.

In particular:

Proposition 1. (Gramlich 94) [8]
Any simply terminating TRS is $\mathcal{C}_\mathcal{E}$-terminating.

The running example

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

Example 2.
The following system R describes addition and multiplication of nonnegative rational integers 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.*

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

The first (simplification) rule ensures that $2_{\mathbb{N}} \times 0_{\mathbb{N}}$ *is* $0_{\mathbb{N}}$.

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 which define them.

Definition 1. — 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 $l \to r \in R_2$, $\Lambda(l) \in \mathcal{F}_2$.

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:

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

A module can extend more than one system and the extension by a module naturally associates 'arrow-wise'. Hence we will 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 2. — 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
- $[\mathcal{F}_0 \mid R_0] \longleftarrow [\mathcal{F}_2 \mid R_2]$.

Such an extension may be seen as a *union of composable systems*[12, 13, 10, 15]. Figure 1 illustrates the two different kinds of extensions.

It is actually possible to 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. Let us consider two systems $R_1(\mathcal{F}_1)$ and $R_2(\mathcal{F}_2)$ which only share 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* [14] is captured since extension with modules associates arrow-wise.

Further note that the only condition we put on extensions is that new rules must have a new symbol at root position of their left-hand side (see Def. 1), thus, modules extensions subsume notions of heavily constrained hierarchical extensions such as *constructor based extensions* [6] and *proper extensions* [17, 16].

Example 3.

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

(a) $[\mathcal{F}_2 \mid R_2]$ extends $[\mathcal{F}_1 \mid R_1]$. (b) $[\mathcal{F}_2 \mid R_2]$ extends $[\mathcal{F}_0 \mid R_0]$ independently of $[\mathcal{F}_1 \mid R_1]$.

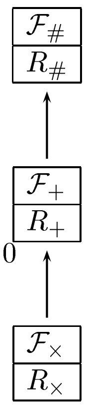
Figure 1: Graphical notations of hierarchical extensions.

- *module R_0 actually defining integers in binary notation;*
- *module R_+ consisting of rules for addition over integers;*
- *module $R_\times$ describing multiplication.*

$$
\begin{array}{ll}
\mathcal{F}_\# & \{\#, 0, 1\} \\
R_\# & \{\#0 \ \rightarrow \ \#
\end{array}
$$

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

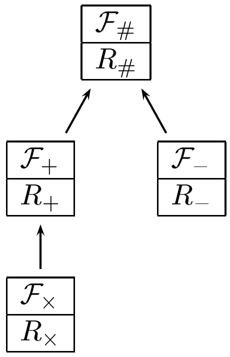
$$
\begin{array}{ll}
\mathcal{F}_\times & \{\times\} \\
R_\times & \left\{ \begin{array}{lllllll}
\# \times x & \rightarrow & \# & & x \times \# & \rightarrow & \# \\
x0 \times y & \rightarrow & (x \times y)0 & & x1 \times y & \rightarrow & (x \times y)0 + y
\end{array} \right.
\end{array}
$$



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

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

And the hierarchy becomes:



3.1 Modular decomposition

Any TRS can be studied as a certain modules hierarchy exploited in its present state i.e., as it is provided. However, it is possible to consider a unique canonical decomposition. A TRS can actually be seen as a hierarchy of *minimal modules*, modules which cannot be split up themselves in a hierarchy of non-empty modules.

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

Definition 3. — 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 $l \to r \in R$ 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 $\mathcal{G}$ 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 $\mathcal{G}$, i.e., symbols f and g such that:

$$f \xrightarrow[\mathcal{G}]{}^{\star} g \text{ and } g \xrightarrow[\mathcal{G}]{}^{\star} f.$$

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 Λ. 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, one can gather constructor symbols reachable from the same packs.

The hierarchy shown in Example 3 is in particular a canonical one modulo 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 4. — Let $\widehat{\mathcal{F}}$ be signature $\mathcal{F}$ extended with 'marked copies' $\widehat{f}$ for all symbols f. Marking of a term is as follow: for a term $t \in T(\mathcal{F}, X)$, we denote $\widehat{t}$ term t for which the symbol at Λ 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 $\langle \widehat{l}, \widehat{r'} \rangle$ such that there is a rule $l \to r \in R$ for which term r' is a subterm of r with $\Lambda(r') \in \mathcal{F}$.

We denote DP(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 $R \cup R_0$.

Example 4.

In the binary arithmetic example, the classical approach amounts to considering five dependency pairs for multiplication rules:

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

But there is only two dependency pairs of module $R_\times$, namely $\langle x0 \mathbin{\widehat{\times}} y, x \mathbin{\widehat{\times}} y \rangle$ and $\langle x1 \mathbin{\widehat{\times}} y, x \mathbin{\widehat{\times}} y \rangle$ 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 $[\mathcal{F}_C \mid \emptyset] \longleftarrow [\mathcal{F}_D \mid R]$, the dependency pairs of module $[\mathcal{F}_D \mid R]$ are exactly the dependency pairs of $R(\mathcal{F})$ in the Arts & Giesl's approach.

4.2 Relative dependency chains

We saw that, considered in a hierarchy, the notion of 'defined symbol' lose 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

DP-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 5. — 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 DP(M) together with a substitution σ 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 is said to be *minimal* if all proper subterms of instantiated pairs rhs are S strongly normalizable.

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

$\mathcal{C}_\mathcal{E}$-termination is easily expressed in terms of relative dependency chains.

Proposition 2.

A term rewriting system $R(\mathcal{F})$ is $\mathcal{C}_\mathcal{E}$-strongly normalizing 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 π are variables, DP($R \cup \pi$) $=$ DP(R) and $\mathcal{C}_\mathcal{E}$ termination of R is shown with the help of Remark 3. □

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 follow: firstly we will state and prove our main result, Theorem 1. The proof is rather technical and involves a technical key lemma, Lemma 2. From Theorem 1 we will obtain as a corollary sufficient conditions to ensure composability of $\mathcal{C}_\mathcal{E}$ termination (Cor. 1–1). We will 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 & Ohuchi [10].

Theorem 1.

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

1. if R_1 is $\mathcal{C}_\mathcal{E}$-strongly normalizing,
2. if 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 strongly normalizing.

PROOF. By contradiction. Proof scheme is as follow: 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;

- either R_1 is not $\mathcal{C}_{\mathcal{E}}$-terminating, now contradicting the first premise of Theorem 1.

Let us 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 $R_1 \cup R_2$. Please recall that marked symbols occur at root position only.

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

① pairs of $[\mathcal{F}_1 \mid R_1]$;

② pairs of $[\mathcal{F}_2 \mid R_2]$;

③ finally pairs $\langle \widehat{l}, \widehat{r'} \rangle$ such that $l \to r \in R_2$ 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. In order to avoid the third one we need a lemma.

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

Then for all pairs $\langle u_1, v_1 \rangle \in \mathrm{DP}(R_1)$ and $\langle u_2, v_2 \rangle$ of $[\mathcal{F}_2 \mid R_2]$ there is no substitution σ such that:

$$v_1\sigma \overset{\neq\Lambda}{\longrightarrow}{}^{\star} u_2\sigma.$$

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 $\Lambda(u_2) \neq \Lambda(v_1)$. □

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

- First case: An infinite dependency chain of pairs of $[\mathcal{F}_2 \mid R_2]$ over $R_1 \cup R_2$ contradicts second premise of Theorem 1.

- Cases 2 & 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: $\mathcal{C}_{\mathcal{E}}$-termination of R_1.

  In that proof, as well as in Theorem 2 proof, we will use a more general result given here as a technical key lemma: Lemma 2. Its proof actually provides the expected translation process.

  Lemma 2.
  Let S_1 and S_2 be two TRS 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 $l \to r \in S_3$, $\Lambda(l) \in \mathcal{F}_2$.

  Then, from an infinite minimal dependency chain of $[\mathcal{F}_1 \mid S_2]$ over $S_1 \cup S_2 \cup S_3$, it is possible to 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.

What remains of Theorem 1 proof mainly consists in 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 $\mathcal{C}_{\mathcal{E}}$-strongly normalizing, such a chain raises a contradiction. This ends the proof of Theorem 1. □

Proof of Lemma 2

We use an interpretation of terms akin to Gramlich's [8]. The main difference between our interpretation and his Φ is related to the definition of $\text{Red}(t)$. We actually take *all one-step reductions* into account while function $SUCC^{\mathcal{F}_1}(t)$ occurring in Φ selects from terms obtained in *reductions by* $\to_S^\star$ 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 alternate proofs since interpretations of two terms s and t might be identical if more than one-step reductions are considered. Thus, using Gramlich's Φ we would end with

$$\Phi(s) \xrightarrow[S_1 \cup S_2 \cup \pi]{}^{\star} \Phi(t)$$

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 very similar manner.

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

Definition 6. — 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 follow:

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

Where

$$\begin{array}{rcl} \text{Red}(t) & = & \{I(t') / \, t \xrightarrow[S_1 \cup S_2 \cup S_3]{} t'\}, \\ \text{Comb}(\emptyset) & = & \perp, \\ \text{Comb}(\{a\} \cup E) & = & G(a, \text{Comb}(E)) \text{ where for all } e \in E, a < e). \end{array}$$

Red(t) denotes set E of interpreted one-step-reducts of t. In order to avoid any ambiguity on the actual construction of tree Comb(E) from E (non-ordered set), we need a total ordering $>$ providing a building strategy.

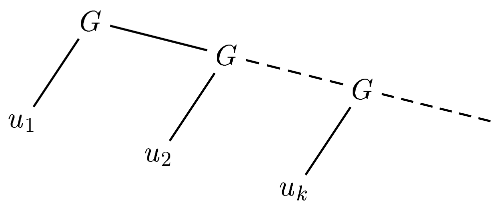


Figure 2: Structure of interpreted terms.

Remark 4. — The resulting tree is 'comb-like'. As Figure 2 illustrates, the interpretation of term $t = f(t_1, \dots, t_n)$ where $f \in \mathcal{F}_2$ actually is a sequence of its one-step-reducts interpretations. Those are 'teeth' of the comb; each u_i being an element of Red(t).

Since each tooth itself is an interpretation, it is possible to reach any of them using a suitable $\xrightarrow[\pi_2]{}^{\star} \xrightarrow[\pi_1]{}$ reduction.

We need a few more technical lemmas to ensure that our interpretation behaves in an adequate manner.

Given a substitution σ, by $I(\sigma)$ we denote substitution σ' 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 σ,

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

PROOF. Structural induction on t. □

Lemma 4.

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

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

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. □

Lemma 6.

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

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

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 Λ to p.

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

$$\begin{array}{rcl}
I(s) & = & I(C[s_1, \ldots, l\sigma, \ldots, s_n]) \\
& = & C[I(s_1), \ldots, I(l\sigma), \ldots, I(s_n)] \text{ (Lemma 4)} \\
& = & C[I(s_1), \ldots, lI(\sigma), \ldots, I(s_n)] \text{ (Lemma 3)} \\
& \xrightarrow[S_1\cup S_2]{p} & C[I(s_1), \ldots, rI(\sigma), \ldots, I(s_n)] \text{ (Premises)} \\
& = & C[I(s_1), \ldots, I(r\sigma), \ldots, I(s_n)] \text{ (Lemma 3)} \\
& = & I(C[s_1, \ldots, r\sigma, \ldots, s_n]) \\
& = & I(t).
\end{array}$$

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 $s = C[s_1, \ldots, s', \ldots, s_n]$ where C is a context with n holes (possibly empty) over $\mathcal{F}_1$, $p = p'q$ and $s|_{p'} = s'$ with $s' = C'[l\sigma] \xrightarrow[S_1 \cup S_2]{} C'[r\sigma] = t'$. Hence:

$$\begin{array}{rcl} I(s) & = & I(C[s_1, \ldots, s', \ldots, s_n]) \\ & = & C[I(s_1), \ldots, I(s'), \ldots, I(s_n)] \text{ (Lemma 4).} \end{array}$$

From Definition 6, $I(s') = \text{Comb}(\text{Red}(s'))$. But

$$s'|_q = l\sigma \xrightarrow[S_1 \cup S_2]{} r\sigma.$$

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

$$I(s') \quad \underset{\pi}{\rightarrow}^{+} \quad I(t').$$

Eventually,

$$\begin{array}{rcl} C[I(s_1), \ldots, I(s'), \ldots, I(s_n)] & \underset{\pi}{\rightarrow}^{+} & C[I(s_1), \ldots I(t'), \ldots, I(s_n)] \\ & = & I(C[s_1, \ldots, t', \ldots, s_n]) \text{ (Lemma 4)} \\ & = & I(t). \end{array}$$

□

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

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

PROOF. Let $\langle u_1, v_1 \rangle, \langle u_2, v_2 \rangle, \ldots$ be a dependency chain of $[\mathcal{F}_1 \mid S_2]$ over $S_1 \cup S_2 \cup S_3$ with a substitution σ. Let σ' be the substitution such that for all x, $x\sigma' = I(x\sigma)$.

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

We are going to show that $\langle u_1, v_1 \rangle, \langle u_2, v_2 \rangle, \ldots$ together with σ' is actually a dependency chain of $[\mathcal{F}_1 \mid S_2]$ over $S_1 \cup S_2 \cup \pi$.

So as to do that we have to prove that for all i,

$$v_i\sigma' \xrightarrow[S_1 \cup S_2 \cup \pi]{\neq\Lambda}{}^{\star} u_{i+1}\sigma'.$$

We know that

$$v_i\sigma \xrightarrow[S_1\cup S_2\cup S_3]{\neq\Lambda}{}^{\star} u_{i+1}\sigma.$$

Let us consider a step $s \xrightarrow[S_1\cup S_2\cup S_3]{p} t$. 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:

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

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

$$I(v_i\sigma) \xrightarrow[S_1\cup S_2\cup \pi]{\neq\Lambda}{}^{\star} I(u_{i+1}\sigma).$$

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

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 composability of $\mathcal{C}_\mathcal{E}$-termination.

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

1. if R_1 is $\mathcal{C}_\mathcal{E}$-strongly normalizing,

2. if 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 $\mathcal{C}_\mathcal{E}$-strongly normalizing.

Theorem 2.
Let $[\mathcal{F}_1 \mid R_1] \longleftarrow [\mathcal{F}_2 \mid R_2]$ be a hierarchical extension of $R_1(\mathcal{F}_1)$ and let $[\mathcal{F}_3 \mid R_3]$ be a module extending R_1 independently of R_2;

1. if $R_1 \cup R_2$ is $\mathcal{C}_\mathcal{E}$-strongly normalizing,

2. if 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 $\mathcal{C}_\mathcal{E}$-strongly normalizing.

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-$\mathcal{C}_\mathcal{E}$-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 either:

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

Hence, $R_1 \cup R_2 \cup R_3$ is $\mathcal{C}_{\mathcal{E}}$-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 fashion.

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

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

Eventually, by Proposition 2 we have as a corollary of Theorem 2 a previous result by Kurihara & Ohuchi [10]: $\mathcal{C}_{\mathcal{E}}$-termination is a modular property for unions of composable TRS.

Theorem 1 is clearly an incremental result while Theorem 2 is a modular one.

5 Proving termination

Termination is usually proven using appropriate well-founded orderings. We propose in this section a class of orderings (π-*expandable orderings*) which are well-suited for $\mathcal{C}_{\mathcal{E}}$-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 which provide effective methods for incremental/modular termination proof.

5.1 π-expandable orderings

Definition 7. — A term ordering $(\succeq, \succ)$ over $T(\mathcal{F}, X)$ is said to be π-*expandable* if there is a reduction ordering $(\succeq', \succ')$ over $T(\mathcal{F} \cup \{\pi : 2\}, X)$ such that:

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

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

A π-expandable ordering is *strongly* π-*expandable* (respectively *weakly* π-*expandable*) if both it and its associate ordering are strongly monotonic (resp. weakly monotonic).

π-expandable orderings may be used for proving termination.

Proposition 3.
Let $(\succeq, \succ)$ be a strictly monotonic π-expandable ordering. If for each rule $l \to r$ of a term rewriting system R, $l \succ r$, then R is $\mathcal{C}_{\mathcal{E}}$-strongly normalizing.

It is clear that any simplification ordering is π-expandable. In particular RPO and the orderings induced by polynomial intepretations are π-expandable. We may indeed combine them so as to obtain some new ones.

Proposition 4.
If $(\succeq_1, \succ_1)$ is a strongly π-expandable ordering and if $(\succeq_2, \succ_2)$ is π-expandable, then the lexicographical composition $((\succeq_1, \succ_1), (\succeq_2, \succ_2))_{lex}$ is a π-expandable ordering.

Moreover, if $(\succeq_2, \succ_2)$ is strongly π-expandable itself, then so is composition $((\succeq_1, \succ_1), (\succeq_2, \succ_2))_{lex}$.

Proof. We have to find a suitable $(\succeq', \succ')$ satisfying the two conditions of Def. 7. Let us suppose that $(\succeq'_1, \succ'_1)$, strictly monotonic, and $(\succeq'_2, \succ'_2)$ are associate ordering to, respectively, $(\succeq_1, \succ_1)$ and $(\succeq_2, \succ_2)$. We may verify that $(\succeq', \succ') = ((\succeq'_1, \succ'_1), (\succeq'_2, \succ'_2))_{lex}$ is suitable, strict monotonicity of $(\succeq'_1, \succ'_1)$ impliying the composition (weak) monotonicity.

We have 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 $(\succeq', \succ')$ restricted to $T(\mathcal{F}, X)$ and $((\succeq_1, \succ_1), (\succeq_2, \succ_2))_{lex}$.

- $G(s,t) \succeq' s$. Either $G(s,t) \succ'_1 s$, or $G(s,t) \succeq'_1 s$ and $G(s,t) \succ'_2 s$, or $G(s,t) \succeq'_1 s$ and $G(s,t) \succeq'_2 s$. In all three cases, from the definition of a lexicographical composition, we have $G(s,t) \succeq' s$.

- $G(s,t) \succeq' t$. Similar to previous case.

- $(\succeq', \succ')|_{T(\mathcal{F},X)} \subseteq ((\succeq_1, \succ_1), (\succeq_2, \succ_2))_{lex}$. Let us get into details for the $s \succeq' t$ case for s and $t \in T(\mathcal{F}, X)$: Either $s \succ'_1 t$ and from premises $s \succ_1 t$, or $s \succeq'_1 t$ and we face to possibilities 1) $s \succ'_2 t$ and then from premises $s \succ_2 t$ or 2) $s \succeq'_2 t$ and then from premises $s \succeq_2 t$, Q.E.D.

- $(\succeq', \succ')|_{T(\mathcal{F},X)} \supseteq ((\succeq_1, \succ_1), (\succeq_2, \succ_2))_{lex}$. Immediate.

Proof regarding $\succ$ is similar. □

As well as lexicographical compositions, *Recursive Program Schemes* [2] may be use for building π-expandables orderings.

Proposition 5.
Let $(\succeq_1, \succ_1)$ be a π-expandable ordering. If P is a Recursive Program Scheme over $\mathcal{F}$ such that $G \notin \mathcal{F}$, then $(\succeq, \succ)$ defined by $s \succeq t$ (resp. $\succ$) if and only if $s{\downarrow_P} \succeq_1 t{\downarrow_P}$ (resp. $\succ_1$) is a π-expandable ordering.

We do not give the proof here but it can be found in the French version of this paper [20].

5.2 Methods for proving termination

From Proposition 2 we have a first test for termination.

Corollary.
Let $(\succeq, \succ)$ be a (weakly) π-expandable ordering such that:

1. $l \succeq r$ for each $l \to r \in R$,

2. $s \succ t$ for each $\langle s, t \rangle \in \mathrm{DP}(R)$,

then R is $\mathcal{C}_{\mathcal{E}}$-strongly normalizable.

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

Corollary 1–2.
Let $[\mathcal{F}_1 \mid R_1] \longleftarrow [\mathcal{F}_2 \mid R_2]$ be a hierarchical extension of $R_1(\mathcal{F}_1)$;

1. if R_1 $\mathcal{C}_{\mathcal{E}}$-terminates and

2. if there is a weak reduction ordering (resp. weakly π-expandable) $(\succeq, \succ)$ such that:

- $R_1 \cup R_2 \subseteq \succeq$ and
- DP($[\mathcal{F}_2 \mid R_2]$) $\subseteq \succ$,

then $R_1 \cup R_2$ terminates (resp. $\mathcal{C}_\mathcal{E}$-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 DP step, there is a infinite sequence strictly decreasing for $(\succeq, \succ)$. This contradicts the well-foundedness of $(\succeq, \succ)$. $\square$

Similarly, π-expandable orderings give us a simple way of using Theorem 2.

Corollary 2–1.
Let $[\mathcal{F}_1 \mid R_1] \longleftarrow [\mathcal{F}_2 \mid R_2]$ be a hierarchical extension of $R_1(\mathcal{F}_1)$, let $[\mathcal{F}_3 \mid R_3]$ be a module which extends R_1 independently of $[\mathcal{F}_2 \mid R_2]$;

1. if $R_1 \cup R_2$ $\mathcal{C}_\mathcal{E}$-terminates and
2. if there is a weakly π-expandable ordering $(\succeq, \succ)$ such that:
   - $R_1 \cup R_3 \subseteq \succeq$ and
   - DP($[\mathcal{F}_3 \mid R_3]$) $\subseteq \succ$,

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

Remark 6. — Optimizations with dependency graphs [2] may be applied to all results presented here [20]. Further note that system π adds arcs to the dependency graph but, *for Arts & Giesl estimation*, the estimated dependency graph remains unmodified.

6 Modules and Automation

Thanks to their generality and to the purely syntactical tests, we implemented our methods in the termination tool-box of the C*i*ME 2 system [4].

When it comes to automation of proofs, ordering constraints are the main problem. For large TRS, 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 Thm 2 and cor.) and in the strictness (incrementality with Thm 1 and cor.) of these constraints and such improvements show on proofs.

For instance, C*i*ME 2 took less than a second for finding all interpretations in Section 6.1, in a completly automated way using the incremental and modular methods.

We also tried our results on large systems provided by Thomas Arts. Those systems come from a real application μ-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 linear interpretation only; it was looking for simple interpretations[2] for more than ten days when we stopped the computer. But with our incremental/modular methods it took *less than three seconds* for C*i*ME 2 to find a termination proof with linear interpretations only [20]. Automated termination proof for real programs is no longer out of reach.

C*i*ME 2 is available at `http://cime.lri.fr`

6.1 Example

We present in this section as a complete example of an incremental proof using our results. We will actually extend Example 2.

So let us consider the system describing nonnegative rational integers (which clearly $\mathcal{C}_{\mathcal{E}}$-terminates):

$$\begin{array}{ll} \mathcal{F}_{\mathbb{N}} & \{\# : \textit{constant}\, ; 1, 0 : \textit{unary}\} \\ R_{\mathbb{N}} & \{\#0 \to \#. \end{array}$$

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:

$$\begin{array}{ll} \mathcal{F}_+ & \{+ : \textit{infix binary}\} \\ R_+ & \left\{ \begin{array}{lll} x + \# & \to & x \\ \# + x & \to & x \\ x0 + y0 & \to & (x+y)0 \\ x0 + y1 & \to & (x+y)1 \\ x1 + y0 & \to & (x+y)1 \\ x1 + y1 & \to & ((x+y)+\#1)0 \\ x + (y+z) & \to & (x+y)+z. \end{array} \right. \end{array}$$

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

$$\mathrm{DP}([\mathcal{F}_+ \mid R_+]) : \left\{ \begin{array}{l} \langle x0 \mathbin{\widehat{+}} y0, x \mathbin{\widehat{+}} y \rangle; \\ \langle x0 \mathbin{\widehat{+}} y1, x \mathbin{\widehat{+}} y \rangle; \\ \langle x1 \mathbin{\widehat{+}} y0, x \mathbin{\widehat{+}} y \rangle; \\ \langle x1 \mathbin{\widehat{+}} y1, x \mathbin{\widehat{+}} y \rangle; \\ \langle x1 \mathbin{\widehat{+}} y1, (x+y) \mathbin{\widehat{+}} \#1) \rangle; \\ \langle x \mathbin{\widehat{+}} (y+z), x \mathbin{\widehat{+}} y \rangle; \\ \langle x \mathbin{\widehat{+}} (y+z), (x+y) \mathbin{\widehat{+}} z \rangle \end{array} \right\} \qquad \begin{array}{l} [\![\#]\!] = 0, \\ [\![0]\!](x) = x+1, \\ [\![1]\!](x) = x+2, \\ [\![+]\!](x,y) = x+y+1, \\ [\![\widehat{+}]\!](x,y) = x+2y. \end{array}$$

[2] *Simple* is here the Steinbach's notion.

With reference to the ordering defined using the interpretation above, pairs of DP($[\mathcal{F}_+ \mid R_+]$) strictly decrease while rules of $R_{\mathbb{N}} \cup R_+ \cup \pi$ weakly decrease. Corollary 1–2 allows us to conclude on the $\mathcal{C}_{\mathcal{E}}$-termination of $R_{\mathbb{N}} \cup R_+$.

We may want to perform subtraction.

$$
\begin{array}{ll}
\mathcal{F}_- & \{- : \textit{infix binary}\} \\
R_- & \left\{ \begin{array}{lcl}
x - \# & \to & x \\
\# - x & \to & \# \\
x0 - y0 & \to & (x-y)0 \\
x1 - y1 & \to & (x-y)0 \\
x1 - y0 & \to & (x-y)1 \\
x0 - y1 & \to & ((x-y) - \#1)1.
\end{array} \right.
\end{array}
$$

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

$$
\begin{array}{l}
[\![\#]\!] = 0, \\
[\![0]\!](x) = x + 1, \\
[\![1]\!](x) = x + 1, \\
[\![-]\!](x, y) = x, \\
[\![\widehat{-}]\!](x, y) = x,
\end{array}
$$

pairs of $[\mathcal{F}_- \mid R_-]$ strictly decrease while rules in $R_{\mathbb{N}} \cup R_-$ weakly decrease. Applying Corollary 2–1 we obtain that $R_{\mathbb{N}} \cup R_- \cup R_+$ is $\mathcal{C}_{\mathcal{E}}$-strongly normalizing.

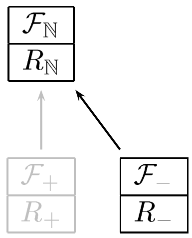


Figure 3: Independent extensions with $[\mathcal{F}_+ \mid R_+]$ and $[\mathcal{F}_- \mid R_-]$.
The proof of $\mathcal{C}_{\mathcal{E}}$-termination of the union uses $\mathcal{C}_{\mathcal{E}}$-termination of $[\mathcal{F}_+ \mid R_+]$ but that module does not interfere in ordering constraints.

In order to compare integers, we need boolean operators. We add a new module, namely $[\mathcal{F}_{\mathsf{Bool}} \mid R_{\mathsf{Bool}}]$.

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

This system is dependency pairs free, hence it trivially $\mathcal{C}_{\mathcal{E}}$-terminates.

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

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

Termination of $R_{\mathbb{N}} \cup R_{\mathsf{Bool}} \cup R_{\mathit{ge}}$ is shown by RPO with $\{\mathit{ge} > \neg > (\mathit{true}, \mathit{false})\}$ directly. Since RPO is a simplification ordering, it is π-expandable. Hence, the relevant union $\mathcal{C}_{\mathcal{E}}$-terminates thanks to Proposition 3. We may then apply Theorem 2 and thus obtain $\mathcal{C}_{\mathcal{E}}$-termination of $R_{\mathbb{N}} \cup R_{\mathsf{Bool}} \cup R_{\mathit{ge}} \cup R_{+} \cup R_{-}$.

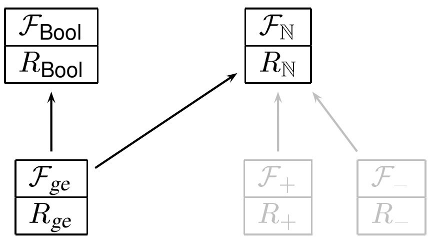


Figure 4: Arithmetical operators and comparisons are independent.

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

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

We use dependency pairs of modules and polynomial interpretations.

$$\mathrm{DP}([\mathcal{F}_{Log'} \mid R_{Log'}]) : \left\{ \begin{array}{l} \langle \widehat{Log'}(x1), \widehat{Log'}(x) \rangle; \\ \langle \widehat{Log'}(x0), \widehat{Log'}(x) \rangle \end{array} \right\}$$

$$\begin{array}{lll}
[\![\#]\!] = 0, & & \\
[\![0]\!](x) = x+1, & [\![1]\!](x) = x+1, & [\![+]\!](x,y) = x+y, \\
[\![\mathit{false}]\!] = 0, & [\![\mathit{true}]\!] = 0, & [\![\neg]\!](x) = 0, \\
[\![\mathit{ge}]\!](x) = 0, & [\![\mathit{if}]\!](x,y,z) = y+z, & [\![\wedge]\!](x,y) = x, \\
[\![\mathit{Log'}]\!](x) = x, & [\![\widehat{Log'}]\!](x) = x. &
\end{array}$$

Dependency pairs strictly decrease while rules in $R_{\mathbb{N}} \cup R_{+} \cup R_{\mathsf{Bool}} \cup R_{ge} \cup R_{Log'}$ weakly decrease. We may then apply Corollary 2–1 in order to show $\mathcal{C}_{\mathcal{E}}$-termination of $R_{\mathbb{N}} \cup R_{+} \cup R_{\mathsf{Bool}} \cup R_{ge} \cup R_{Log'} \cup R_{-}$.

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

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

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

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

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

This module has no dependency pairs.

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

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

We have six dependency pairs for this module:

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

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

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

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

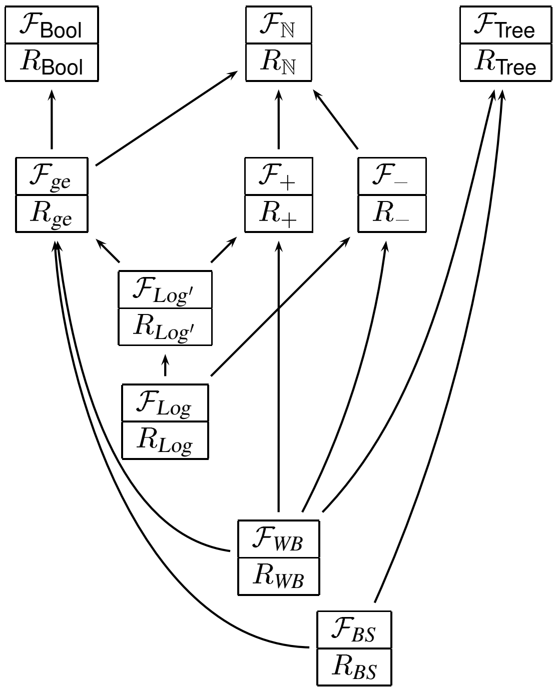


Figure 5: Modules hierarchy.

Eventually, in order to decide if 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.

$$
\begin{array}{ll}
\mathcal{F}_{WB} & \{\mathit{WB}, \mathit{Size} : \mathit{unary}\} \\
R_{WB} & \left\{
\begin{array}{lcl}
\mathit{Size}(\mathcal{L}(x)) & \to & \#1 \\
\mathit{Size}(\mathcal{N}(x,l,r)) & \to & (\mathit{Size}(l) + \mathit{Size}(r)) + \#1 \\
\mathit{WB}(\mathcal{L}(x)) & \to & \mathit{true} \\
\mathit{WB}(\mathcal{N}(x,l,r)) & \to & \quad \mathit{if} \quad (\mathit{ge}(\mathit{Size}(l), \mathit{Size}(r)), \\
 & & \qquad\quad \mathit{ge}(\#1, \mathit{Size}(l) - \mathit{Size}(r)), \\
 & & \qquad\quad \mathit{ge}(\#1, \mathit{Size}(r) - \mathit{Size}(l))) \\
 & & \quad \wedge \quad (\mathit{WB}(l) \wedge \mathit{WB}(r)).
\end{array}
\right.
\end{array}
$$

The set of dependency pairs is the following:

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

With help of the polynomial interpretation

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

we may prove easily, simply using Corollary 2–1, that the union of *all* rules $\mathcal{C}_{\mathcal{E}}$-terminates. Please note how simple interpretations are.

7 Hierarchical Simple termination

We provide in this section a new notion of termination which is modular for unions of composable term rewriting systems. In order to obtain such modularity results we distinguish relevant sets of rules by means of dependency relation over symbols $\triangleright_d$ (see Def. 3).

Definition 8. — Let $(\mathcal{F}, \mathcal{R})$ be a term rewriting system.

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

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

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 9. — Let $(\mathcal{F}, R)$ 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 $\mathcal{G}$.

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

- $s \succ_i t$ for all dependency pairs $\langle s, t \rangle$ where both root symbols of s and t occurs in C_i;

- $l \succsim_i r$ for all rules in 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 TRS.

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 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 $R_1 \cup R_2$. □

Hierarchical Simple termination is clearly more general than simple termination since it uses dependency pairs. It also includes DP-simple termination strictly, thanks to the modular decomposition.

8 Related work and Conclusion

With the notion of *rewriting modules* (Def. 1), we defined a new framework very well suited to the study of the intrinsic hierarchical structure of term rewriting systems.

The module framework has many applications regarding termination proofs. In particular, with the help of *dependency pairs of modules* (Def. 4), we obtain powerful methods (Theorems 1 and 2 and corollaries) which allow proofs to be build incrementally and modularly.

Our work has to compare with Dershowitz's results [6]. 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. Moreover our criteria (fully syntactical and applicable to most TRS met in practice) are more suited for automation—because they were designed for this purpose—than the finely tuned conditions of Dershowitz. Similarly, Krishna Rao's proper extensions and restricted proper extensions are constrained modules extensions.

Arts & Giesl exploit the modular structure of dependency graphs [1]. Their approach is fundamentally different from ours because some constraints from irrelevant rules are filtered by an optimization, namely dependency graphs, while in our case such constraints cannot be generated in the framework. Moreover, their criterion still puts (weak) conditions over 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 Rem. 5, criteria based on Theorem 2 do not require anything from irrelevant sets of rules. Our framework furthermore provides for the general case the powerful results they got for the special case of *innermost* rewriting.

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 since only relevant rules are considered. They are also less restrictive, for the same reasons and also 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 far much simpler than in a proof without modules, even if powerful methods such as dependency pairs with dependency graphs are used.

All these results are implemented in the C*i*ME 2 rewriting tool. They considerably enhance a completely automated search of termination proofs and a significant speed up may be observed on large examples which occur in practice and whose verification is not any longer out of reach.

Acknowledgements

The author would like to thank Thomas Arts, for the very interesting examples he gave, and is grateful to Bernhard Gramlich and Claude Marché for judicious comments and discussions about this work.

References

[1] Arts, T. and J. Giesl: 1998, 'Modularity of Termination Using Dependency Pairs'. In: T. Nipkow (ed.): *9th International Conference on Rewriting Techniques and Applications*, Vol. 1379 of *Lecture Notes in Computer Science*. Tsukuba, Japan, pp. 226–240.

[2] Arts, T. and J. Giesl: 2000, 'Termination of term rewriting using dependency pairs'. *Theoretical Computer Science* **236**, 133–178.

[3] Borralleras, C., M. Ferreira, and A. Rubio: 2000, 'Complete Monotonic Semantic Path Orderings'. In: D. McAllester (ed.): *Proceedings of the 17th Conference on Automated Deduction*, Vol. 1831 of *Lecture Notes in Artificial Intelligence*. Pittsburgh, USA, pp. 346–364.

[4] Contejean, E., C. Marché, B. Monate, and X. Urbain: 2000, 'CiME version 2'. Prerelease available at `http://www.lri.fr/~demons/cime.html`.

[5] Dershowitz, N.: 1987, 'Termination of Rewriting'. *Journal of Symbolic Computation* **3**(1), 69–115.

[6] Dershowitz, N.: 1995, '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, pp. 89–105.

[7] Dershowitz, N. and J.-P. Jouannaud: 1990, ‘Notations for Rewriting’. *EATCS Bulletin* **43**, 162–172.

[8] Gramlich, B.: 1994, ‘Generalized Sufficient Conditions for Modular Termination of Rewriting’. *Applicable Algebra in Engineering, Communication and Computing* **5**, 131–158.

[9] Gramlich, B.: 1995, ‘Abstract relations between restricted termination and confluence properties of rewrite systems’. *Fundamenta Informaticæ* **24**, 3–23.

[10] Kurihara, M. and A. Ohuchi: 1995, ‘Decomposable Termination of Composable Term Rewriting Systems’. *IEICE* **E78–D**(4), 314–320.

[11] Kusakari, K., M. Nakamura, and Y. Toyama: 1999, ‘Argument Filtering Transformation’. In: G. Nadathur (ed.): *Principles and Practice of Declarative Programming, International Conference PPDP’99*, Vol. 1702 of *Lecture Notes in Computer Science*. Paris, pp. 47–61.

[12] Middeldorp, A. and Y. Toyama: 1990, ‘Completeness of combinations of constructor systems’. Technical Report R9058, CWI.

[13] Middeldorp, A. and Y. Toyama: 1991, ‘Completeness of combinations of constructor systems’. In: *Proc. 4th Rewriting Techniques and Applications, Como, LNCS 488*. Como, Italy.

[14] Ohlebusch, E.: 1994, ‘On the Modularity of Termination of Term Rewriting Systems’. *Theoretical Computer Science* **136**, 333–360.

[15] Ohlebusch, E.: 1995, ‘Modular Properties of Composable Term Rewriting Systems’. *Journal of Symbolic Computation* **20**, 1–41.

[16] Rao, M. K.: 1995, ‘Modular proofs for completeness of hierarchical term rewriting systems’. *Theoretical Computer Science* **151**, 487–512.

[17] Rao, M. R. K. K.: 1994, ‘Simple Termination of Hierarchical Combinations of Term Rewriting Systems’. In: *Theoretical Aspects of Computer Software*, Vol. 789 of *Lecture Notes in Computer Science*. pp. 203–223.

[18] Rusinowitch, M.: 1987, ‘On termination of the direct sum of term rewriting systems’. *Information Processing Letters* **26**, 65–70.

[19] Toyama, Y.: 1987, ‘Counterexamples to Termination for the Direct Sum of Term Rewriting Systems’. *Information Processing Letters* **25**, 141–143.

[20] Urbain, X.: 2001, ‘Approche incrémentale des preuves automatiques de terminaison’. Thèse de doctorat, Université Paris-Sud, Orsay, France.