Formal Aspects of Computing

# Termination of Rewrite Systems by Elementary Interpretations

Pierre Lescanne

Centre de Recherche en Informatique de Nancy (CNRS) and INRIA-Lorraine, Vandœuvre-lès-Nancy, France

**Keywords:** Rewrite systems; Termination; Well-foundedness; Total connection

**Abstract.** We focus on termination proofs of rewrite systems, especially of rewrite systems containing associative and commutative operators. We prove their termination by elementary interpretations, more specifically, by functions defined by addition, multiplication and exponentiation. We discuss a method based on polynomial interpretations and propose an implementation of a mechanisation of the comparison of expressions built with polynomials and exponentials.

## 1. Proving Termination of Rewrite Systems by Number Theoretic Functions

Automating proofs of termination, especially proofs of termination of rewrite systems containing associative and commutative (AC) operators, is a major challenge in programming. To prove termination, we use elementary functions, defined by addition, multiplication and exponentiation. Consider, for instance, the rewrite system *FACT* (a functional programmes that describes a *factorial* function)

$$\begin{aligned} 0+x &\rightarrow x \\ S(x)+y &\rightarrow S(x+y) \\ 0*x &\rightarrow 0 \\ S(x)*y &\rightarrow (x*y)+y \\ x*(y+z) &\rightarrow (x*y)+(x*z) \end{aligned}$$

*Correspondence and offprint requests to*: P. Lescanne, Centre de Recherche en Informatique de Nancy (CNRS) and INRIA-Lorraine, Campus Scientifique, BP 239, 54506 Vandœuvre-lès-Nancy, France, email: Pierre.Lescanne@loria.fr.

$$\begin{aligned} fact(0) &\rightarrow S(0) \\ fact(S(x)) &\rightarrow S(x) * fact(x) \end{aligned}$$

where $+$ and $*$ are associative and commutative. Its proof of termination requires a well-founded ordering compatible with associativity and commutativity (AC-compatible, for short), having the replacement property (i.e., compatible with the term structure) and fully invariant (i.e., stable by substitutions). Such AC-compatible well-founded orderings are rare in the literature. Indeed, the lexicographic path ordering [KaL80] cannot handle associative and commutative rewrite systems. Classical extensions of the recursive path orderings [BaP85, GnL86] do not allow the precedence $* > + > S$ that would be necessary in this case. Delor [Del91] relaxed this restriction on the precedence (see also [DeM92, KSZ90]), but there are still rewrite systems (including the *fib* example of section 7.2) that cannot be handled by a precedence-based ordering. Polynomial interpretations [Lan79] with the restrictions for associative and commutative operators proposed in [BCL87] (namely, that the interpretations of AC operators be of the form $a\mathsf{XY}+b(\mathsf{X}+\mathsf{Y})+c$ with $ac + b - b^2 = 0$) do not work, since the function *fact* has exponential growth and therefore cannot have a proof of termination done by polynomial interpretations [CiL91]. This absence of implemented mechanised methods is a great drawback if one wants to fully automate a completion procedure for associative-commutative rewrite systems (for example [Les90]) or to use *oriented paramodulation* or *ordered completion* [HsR87, BDP89]

On the other hand, the rewrite system *FACT* can be proved to terminate if one uses the following interpretations:

$$\begin{aligned} [\![0]\!]_1 &= 2 \\ [\![S]\!]_1(X) &= X + 2 \\ [\![+]\!]_1(X_1, X_2) &= X_1 + X_2 + 1 \\ [\![*]\!]_1(X_1, X_2) &= X_1 \cdot X_2 \\ [\![fact]\!]_1(X) &= X^X + 1 \end{aligned}$$

and

$$\begin{aligned} [\![0]\!]_2 &= 2 \\ [\![S]\!]_2(X) &= X + 2 \\ [\![+]\!]_2(X_1, X_2) &= X_1 \cdot X_2 \\ [\![*]\!]_2(X_1, X_2) &= X_1 \cdot X_2 \\ [\![fact]\!]_2(X) &= X^X + 1 \end{aligned}$$

saying that $s > t$ if $([\![s]\!]_1 >_{\mathbf{N}} [\![t]\!]_1)$ or $([\![s]\!]_1 = [\![t]\!]_1$ and $[\![s]\!]_2 >_{\mathbf{N}} [\![t]\!]_2)$, where $[\![s]\!]_1$ and $[\![s]\!]_2$ are two elementary functions over the non-negative integers. These functions are the interpretations of the term $s$ deduced from the interpretations $[\![\_]\!]_1$ and $[\![\_]\!]_2$ of the operators and $>_{\mathbf{N}}$ is the comparison of the functions over the set $\mathbf{N}$ of non-negative integers. Note that *fact* is interpreted by an exponential. Two levels of interpretation are necessary because the restriction on interpretations of AC operators imposes strict limitations on the interpretations of $+$ and $*$. The proof of termination requires more than the functions allowed by these restrictions and cartesian product of two or many level of interpretations preserves stability by associativity and commutativity.

The method described in [BCL87] for mechanising the comparison of polynomials does not work on account of the exponential; we are going to show

here how that method can be extended. We propose an implementation of a mechanisation of the comparison of expressions built with polynomials and exponentials that are a subclass of "elementary functions" (see [Pét67], chapter 8). Such expressions are called EP-terms in the rest of this paper. We show how they can be used for proving termination of rewrite systems. Note that EP-terms contain any tower of exponentials.

First, we describe our method for the comparison of polynomials in a formalism of rewrite rules, in order to prepare for the presentation of the comparison of elementary functions described in the section 3.

## 2. Interpretations and Termination of Rewrite Systems

The method for proving termination of rewrite systems, called "polynomial interpretations" [Lan79], is based on the idea that a natural way for proving termination is to associate with each term a natural number. Actually, since one works with terms with variables, it is wiser to associate a function. Therefore, a natural idea is to assign to each function symbol a function on the naturals and to compute, by induction on the structure of the terms, the function associated with compound terms. More precisely, with each function symbol $f \in \mathcal{F}$ of arity $n$, one associates a function $[\![f]\!](X_1,\ldots,X_n)$ over natural numbers. By induction, one can associate with each term $t$ with variables $\{x_1,\ldots,x_n\}$ a function $[\![t]\!](X_1,\ldots,X_n)$ over natural numbers with $n$ parameters. For instance, with the interpretation $[\![\_]\!]_1$ of section 1, $[\![fact(S(x))]\!]_1(X) = (X+2)^{X+2}+1 = X^2\cdot(X+2)^X + 4X\cdot(X+2)^X + 4(X+2)^X + 1$. If one can prove that for each rule $l \to r$, the function $[\![l]\!]_1$ is larger than the function $[\![r]\!]_1$ (here, "larger" means that for each instantiation $\iota$ of the parameters by natural numbers in an interval $[c,\infty)$, the number $[\![l]\!]_1(\iota(X_1),\ldots,\iota(X_n))$ is larger than the number $[\![r]\!]_1(\iota(X_1),\ldots,\iota(X_n))$), then one can prove the termination of the rewrite system. If one takes again $[\![\_]\!]_1$ of section 1, the inequality

$$[\![fact(S(x))]\!]_1(X) = X^2\cdot(X+2)^X + 4X\cdot(X+2)^X + 4(X+2)^X + 1$$
$$>$$
$$[\![S(x) * fact(x)]\!]_1(X) = X\cdot X^X + 2X$$

is satisfied for all $X \geq 2$. If one could prove the same for the other rules, *FACT* would be proved to terminate. Actually, for rule $S(x)+y \to S(x+y)$ one has $[\![S(x)+y]\!]_1(X_1,X_2) = [\![S(x+y)]\!]_1(X_1,X_2) = X_1 + X_2 + 3$, but one has

$$[\![S(x)+y]\!]_2(X_1,X_2) = X_1\cdot X_2 + 2X_2 > [\![S(x+y)]\!]_2(X_1,X_2) = X_1\cdot X_2 + 2$$

which means *FACT* terminates. Therefore, the problem of termination boils down, in this case, to provide adequate interpretations for basic operators and then to prove that a function is larger than another over an interval of the naturals. The first step is usually done by the user and the second is mechanised [BCL87], which is very helpful since such comparisons are needed often for orienting equalities during the process of completion.

Until now, to make comparisons of functions easy and, especially, to mechanise the ordering, one restricted interpretations to be polynomials only. Additionally, polynomial interpretations work well for proving termination of rewrite systems modulo the associativity and the commutativity of some operations. Indeed, if an operation is associative and commutative, the interpretation has to satisfy a

condition that is easy to check, namely the polynomial has to be of the form $aXY + b(X + Y) + c$ with $ac + b - b^2 = 0$. In section 1, $[\![+]\!]_1$, $[\![*]\!]_1$, $[\![+]\!]_2$ and $[\![*]\!]_2$ fulfill these requirements. This makes the number theoretic interpretation method the only practical one in this case. Methods based on polynomial interpretations have been implemented and included in software that handles rewrite systems, including REVE [For84, Les83], COMTES [AMS89], LP [GaG89] or, ORME [Les90].

## 3. ORME

Since the procedures we are going to describe have been implemented in ORME, we would like to mention a few things about this software. ORME is a set of tools for dealing with equational theories, rewrite techniques and completion procedures. It is written in ML (more precisely in CAML [WAL89]) and incorporates an associative and commutative completion procedure. The completion procedure is described by transformation rules [Bac91]. Proofs of termination of associative and commutative rewrite systems is, therefore, an important part of ORME. ORME has been described in [Les90, Les89] and is available by anonymous ftp[1] or upon request to the author. ORME has now been upgraded to include the method described in this paper.

## 4. Comparing Polynomials Using Rewrite Systems

In this section, we describe a method for comparing polynomials which is the key to our method for proving termination based on polynomial interpretations. For simplicity, we restrict comparisons to the interval of numbers greater than or equal to 2; hence, the aim is to prove that a polynomial $P(\mathsf{X}_1, \ldots, \mathsf{X}_m)$ is greater than a polynomial $Q(\mathsf{X}_1, \ldots, \mathsf{X}_m)$ over the set $[2, +\infty)$ of integers larger than 1.

In what follows, since we consider polynomials with non-negative integer coefficients, we represent monomials $c\mathsf{X}_1^{n_1} \ldots \mathsf{X}_m^{n_m}$ as the sum of $c$ monomials $\mathsf{X}_1^{n_1} \ldots \mathsf{X}_m^{n_m}$. Similarly, we do not use exponents, but repeat $\mathsf{X}_i$ as often as it occurs in the monomials; in other words, we write

$$\underbrace{\mathsf{X}_i \ldots \mathsf{X}_i}_{n \text{ times}}$$

instead of $\mathsf{X}_i^n$. In what follows, the $\mathsf{X}_i$'s are called "letters". For instance, $3\mathsf{X}_1\mathsf{X}_2^2$ will be represented internally as $\mathsf{X}_1\mathsf{X}_2\mathsf{X}_2 + \mathsf{X}_1\mathsf{X}_2\mathsf{X}_2 + \mathsf{X}_1\mathsf{X}_2\mathsf{X}_2$. With these conventions, the method of [BCL87] can be presented by three rewrite systems: The first is a rewrite system $\mathscr{R}$ modulo associativity and commutativity of '+' and '·' with four rules

$$\begin{aligned} 0 + x &\to x \\ 0 \cdot x &\to 0 \\ 1 \cdot x &\to x \\ x \cdot (y + z) &\to (x \cdot y) + (x \cdot z) \end{aligned}$$

[1] ftp on machine *ftp.loria.fr*.

for reducing polynomials with positive coefficients to their canonical form. There is a rule for reducing a polynomial to a smaller one, namely $\mathscr{H}$:

$$\mathsf{X} \hookrightarrow 1+1$$

for every letter X and there are two rules for reducing a polynomial to a strictly smaller one, namely $\mathscr{P}$:

$$1 \rightsquigarrow 0$$
$$\mathsf{X} \rightsquigarrow 1$$

for every letter X. Since 0 is the identity of '+' and 1, the identity of '·', the relation $P \overset{+}{\rightsquigarrow} Q$ (where $\overset{+}{\rightsquigarrow}$ is the transitive closure of $\rightsquigarrow$) can be implemented by a function that checks whether $Q$ is embedded into $P$ where a monomial $m$ is embedded in another monomial $m'$ if the set of the letters of $m$ is a subset of the set of the letters of $m'$, and a polynomial $P$ is embedded in a polynomial $Q$ if each monomial in $P$ is embedded in a monomial in $Q$. The implementation of the embedding is made easier if one has a canonical representation for polynomials, obtained, for instance, by sorting the monomial according to a given ordering. Let us write $\Longrightarrow$ the relation $(\rightarrow \cup \hookrightarrow \cup \rightsquigarrow)^* \circ \rightsquigarrow \circ (\rightarrow \cup \hookrightarrow \cup \rightsquigarrow)^*$. It is easy to see that $P >_{\mathrm{N}} Q$ if $P \Longrightarrow \circ \overset{*}{\leftarrow} Q$. In other words $P$ is greater than $Q$ if one reaches an $\mathscr{R}$-reduced form of $Q$ from $P$ by many steps of $\rightarrow$ or $\hookrightarrow$ or $\rightsquigarrow$ with at least one step of $\rightsquigarrow$.

Let us look at an example (the same one as presented on page 143 of [BCL87] for the proof of termination of a distributivity rule),

$$\mathsf{UX^2YZ + X^2YZ + XZ} \quad > \quad \mathsf{UXYZ + UXZ + XYZ + XZ}$$

Let $L$ be the left-hand side and $R$, the right-hand side, and let $\mathsf{X^2}$ be an abbreviation for XX. Rewriting X to $1+1$ by $\hookrightarrow$ in the first monomial of $L$ and computing the normal form of the polynomial gives $\mathsf{UXYZ + UXYZ + X^2YZ + XZ}$. Reducing Y to 1 by $\rightsquigarrow$ in the second monomial gives $\mathsf{UXYZ + UXZ + X^2YZ + XZ}$. Reducing X in the third monomial gives $\mathsf{UXYZ + UYZ + XYZ + XZ}$ which is equal to $R$. Therefore $L \Longrightarrow R$ which implies $L >_{\mathrm{N}} R$. Note that we can check easily that $R = \mathsf{UXYZ + UXZ + XYZ + XZ}$ is embedded in $\mathsf{UXYZ + UXYZ + X^2YZ + XZ}$. A direct implementation of the embedding will be really useful and efficient with expressions containing exponentials.

In [BCL87], the key point was to provide heuristics for reaching $Q$ from $P'$ (where $P'$ is the $\mathscr{R}$-normal form of $P$) by $\Longrightarrow$-reductions. Reachability in terminating rewrite systems is decidable, but a decision procedure based on an exhaustive search could be inefficient. The implementation of [BCL87] is efficient, since it is based on finding monomials $m$ in $P$ and $m'$ in $Q$ such that the difference of the degrees is minimum; then $m$ is $\hookrightarrow$-rewritten at the letters where the degree is different and as many times as the difference in degrees.

The set $\mathscr{H}$ can be extended by other rules, provided $P \hookrightarrow Q$ implies $P \geq_{\mathrm{N}} Q$. For instance, rules like

$$\mathsf{XX + YY} \quad \hookrightarrow \quad \mathsf{XY + XY}$$
$$(x \cdot \mathsf{XX}) + (x \cdot \mathsf{YY}) \quad \hookrightarrow \quad (x \cdot \mathsf{XY}) + (x \cdot \mathsf{XY}).$$

This allows one to prove $\mathsf{XX + YY + 1 - XY - XY} > 0$. Hans Zantema (private communication) mentioned me an example of integers with addition, subtraction, multiplication and squaring where such an argument is necessary.

## 5. Comparing EP-Terms

First, let us say a few words on the kind of expressions we consider. They are defined by the following grammars for EP-terms:

$$\text{EP\_term} ::= 0 \mid \text{EP\_monomial} + \text{EP\_term}$$

$$\text{EP\_monomial} ::= 1 \mid \text{EP\_component}\ \ \text{EP\_monomial}$$

$$\text{EP\_component} ::= \text{String} \mid (\text{EP\_term})^{(\text{EP\_term})}$$

String represents the letters of the EP expressions. In addition to those of section 2 for polynomials, the system, still called $\mathscr{R}$, for reducing to normal forms has rules

$$\mathscr{R}: \begin{cases} 0+x & \to & x \\ 0\cdot x & \to & 0 \\ 1\cdot x & \to & x \\ x\cdot(y\cdot z) & \to & (x\cdot y)+(x\cdot z) \\ \\ x^{(y+z)} & \to & x^y\cdot x^z \\ (x\cdot y)^z & \to & x^z\cdot y^z \\ (x^y)^z & \to & x^{(y\cdot z)} \\ x^1 & \to & x \\ x^0 & \to & 1 \end{cases}$$

System $\mathscr{H}$ is modified to include new facts about binomials it is

$$\mathscr{H}: \begin{cases} X & \hookrightarrow & 1+1 \\ (x+y)^z & \hookrightarrow & x^z+y^z \end{cases}$$

In a recent version we introduced a rule

$$(x+y)^z \hookrightarrow x^z + z\cdot x^{z-1} + y^z$$

which requires to change the data structure described in the next section. $\mathscr{P}$ is the same as in Section 2, that is:

$$\mathscr{P}: \begin{cases} 1 & \rightsquigarrow & 0 \\ X & \rightsquigarrow & 1 \end{cases}$$

and $P >_N Q$ if $P \Longrightarrow \circ \overset{*}{\leftarrow} Q$.

## 6. Implementation and Examples

### 6.1. Data Structure and Implementation of $\mathscr{R}$

The above described method has been implemented in CAML as a part of ORME[2]. EP-terms are represented by the following data structure: EP-terms are lists of EP-monomials and 0 is the empty list of EP-monomials. EP-monomials are lists

[2] The CAML documented code of the implementation of EP is about 500 lines and 40 definitions of functions.

of EP-components and 1 is the empty list of EP-components. EP-components are either a letter or a power, i.e., a pair of EP-terms, namely the base and the exponent. $\mathscr{R}$ is implemented directly by three functions called *EP_term_norm*, *EP_monomial_norm* and *EP_component_norm*, acting on EP-terms. EP-terms are sorted with respect to a canonical ordering using a function *EP_term_sort* in order to build canonical representations for EP-terms as a part of the implementation of $\mathscr{P}$.

### 6.2. Canonical Comparison of EP Canonical Forms

The total ordering between components on which the sorting is based compares letters alphabetically and makes them smaller than raising to a power; powers are compared lexicographically, i.e., the bases first, then exponents. Comparisons of bases and exponents recursively invoke the ordering between EP-terms. Note that this total ordering is not at all the ordering we are building for proving termination.

### 6.3. Implementation of $\mathscr{P}$

$\mathscr{P}$ is implemented by an embedding similar to the one described above for polynomials. An EP-term $P$ is embedded in an EP-term $Q$ if with each EP-monomial $m$ of $P$ one can associate one-to-one an EP-monomial of $Q$ in which $m$ is embedded. Similarly, an EP-monomial $m$ is embedded in an EP-monomial $m'$ if with each EP-component of $m$ one can associate one-to-one an EP-component of $m'$ in which it is embedded. At the EP-component level, a letter is embedded in a letter of same name or in a power if it occurs in the base or in the exponent. A power $b^e$ is embedded in another power $p^q$ if the base $b$ is embedded in the base $p$ and the exponent $e$ is embedded in the exponent $q$.

### 6.4. Implementation of $\mathscr{H}$

$\mathscr{H}$ is implemented by choosing a letter X and replacing it by $1+1$ or by choosing a power of the form $(x+y)^z$ and by replacing it by $x^z + z \cdot x^{z-1} + y^z$. This is, in effect, a step of rewriting. $\mathscr{H}$-rewriting is done breadth first and is the most expensive step of the process. In the implementation, the depth of search is limited. In our experiments, it was limited to 4.

### 6.5. Two Optimizations

Since the comparison of polynomials by the procedure described in [BCL87] is anyway better that the comparison of *EP_term*'s, this procedure is invoked in the case of pure polynomials. On the other hand, comparing the evaluations of two elementary terms on a specific value is cheap. For empirical reasons, we have chosen the value 3; thus, before trying to prove that an EP-term $s$ is smaller than an EP-term $t$, we compare their evaluations on 3, if $s(3) \geq t(3)$ we do not pursue the comparison. This trick improved the efficiency of the implementation dramatically when used for orienting equalities, as in completion.

$$
\begin{aligned}
0 + x &\rightarrow x \\
S(x) + y &\rightarrow S(x + y) \\
0 * x &\rightarrow 0 \\
S(x) * y &\rightarrow (x * y) + y \\
x * (y + z) &\rightarrow (x * y) + (x * z) \\
fact(0) &\rightarrow S(0) \\
fact(S(x)) &\rightarrow S(x) * fact(x) \\
fib(0) &\rightarrow 0 \\
fib(S(0)) &\rightarrow S(0) \\
fib(S(S(x))) &\rightarrow fib(x) + fib(S(x)) \\
sqr(0) &\rightarrow S(0) \\
sqr(S(x)) &\rightarrow sqr(x) + sqr(x)
\end{aligned}
$$

with the interpretations

$$
\begin{aligned}
[\![0]\!]_1 &= 2 & [\![0]\!]_2 &= 2 \\
[\![S]\!]_1(X) &= X + 2 & [\![S]\!]_2(X) &= X + 2 \\
[\![+]\!]_1(X_1, X_2) &= X_1 + X_2 + 1 & [\![+]\!]_2(X_1, X_2) &= X_1 \cdot X_2 \\
[\![*]\!]_1(X_1, X_2) &= X_1 \cdot X_2 & [\![*]\!]_2(X_1, X_2) &= X_1 \cdot X_2 \\
[\![fact]\!]_1(X) &= X^X + 1 & [\![fact]\!]_2(X) &= X^X + 1 \\
[\![fib]\!]_1(X) &= 2^X & [\![fib]\!]_2(X) &= 2^X \\
[\![sqr]\!]_1(X) &= 2^X + 1 & [\![sqr]\!]_2(X) &= 2^X + 1
\end{aligned}
$$

**Fig. 1.** The system NAT + DIST + FACT + FIB + POWER2.

### 6.6. EP_less and EP_order

The main procedure of this process are called *EP_less* and *EP_order*. *EP_less* takes two EP-terms $s$ and $t$ and returns true if $s$ can be proved less than $t$, and false, otherwise. It calls the procedures and the methods described above. *EP_order* takes an interpretation and two terms and returns *Greater*, *Less*, *Equiv* or, *Undef*.

### 6.7. Benchmarks

As examples, we tried *FACT* and the rewrite systems given in Figs 1 and 2. On a SUN 4/75 (SPARC2) workstation and using non-optimized code, we have oriented the rules of *FACT* in 0.35 s, the rules of *NAT + DIST + FACT + FIB + POWER2* in 0.58 s and the rules of *NAT + DIST + EXP* in 0.68 s . Notice that the interpretations of + and * satisfy the restrictions on AC-operators.

$$
\begin{aligned}
0 + x &\rightarrow x \\
S(x) + y &\rightarrow S(x + y) \\
0 * x &\rightarrow 0 \\
S(x) * y &\rightarrow (x * y) + y \\
x * (y + z) &\rightarrow (x * y) + (x * z)
\end{aligned}
$$

$$
\begin{aligned}
x \uparrow 0 &\rightarrow S(0) \\
x \uparrow S(y) &\rightarrow x * (x \uparrow y) \\
x \uparrow (y + z) &\rightarrow (x \uparrow y) * (x \uparrow z) \\
(x * y) \uparrow z &\rightarrow (x \uparrow z) * (y \uparrow z) \\
(x \uparrow y) \uparrow z &\rightarrow x \uparrow (y * z)
\end{aligned}
$$

with the interpretations

$$
\begin{aligned}
[\![0]\!]_1 &= 2 & [\![0]\!]_2 &= 2 \\
[\![S]\!]_1(X) &= X + 3 & [\![S]\!]_2(X) &= X + 2 \\
[\![+]\!]_1(X_1, X_2) &= X_1 + X_2 + 3 & [\![+]\!]_2(X_1, X_2) &= X_1 \cdot X_2 \\
[\![*]\!]_1(X_1, X_2) &= X_1 \cdot X_2 & [\![*]\!]_2(X_1, X_2) &= X_1 \cdot X_2 + 1 \\
[\![\uparrow]\!]_1(X_1, X_2) &= (X_1)^{(X_2+1)} & [\![\uparrow]\!]_2(X_1, X_2) &= (X_1)^{(X_2+1)}
\end{aligned}
$$

**Fig. 2.** The system NAT + DIST + EXP.

## 7. Use of the Ordering in Completions

This ordering is especially interesting in a completion, since it allows a completion without any interaction with the user. The ordering is indeed used for orienting the equalities generated as critical pairs into rewrite rules and therefore it preserves the termination of the rewrite system. More precisely, the user provides the interpretations and starts the completion process. No interaction, like orientation of an equality into a rule, is required later on in the course of completion, if the interpretations are adequately chosen.

### 7.1. Automatic Synthesis of an Iterative Factorial

An interesting example is the automatic generation of the definition of an iterative factorial *IFACT* from its definition in term of *FACT* and properties relating it to * and +, which are given to be associative and commutative:

$$
\begin{aligned}
ifact(x, S(0)) &= fact(x) \\
ifact(x, y + z) &= ifact(x, y) + ifact(x, z) \\
ifact(x, y * z) &= ifact(x, y) * z
\end{aligned}
$$

One completes the system *FACT* together with these three equations, using the following interpretations, which satisfy conditions on the associativity and commutativity of + and * (we omit unnecessary components of the interpretation):

$$
\begin{aligned}
[\![0]\!]_1 &= 2 \\
[\![S]\!]_1(X) &= X+2 \\
[\![+]\!]_1(X_1,X_2) &= X_1+X_2+1 \\
[\![*]\!]_1(X_1,X_2) &= X_1 \cdot X_2 \\
[\![fact]\!]_1(X) &= 6X^X \\
[\![ifact]\!]_1(X_1,Y_2) &= X_1^{X_1}.(X_2+1)
\end{aligned}
\qquad
\begin{aligned}
[\![S]\!]_2(X) &= X+2 \\
[\![+]\!]_2(X_1,X_2) &= X_1 \cdot X_2
\end{aligned}
$$

The completion ends and yields the system:

$$
\begin{aligned}
(0+x) &\to x \\
S(x)+y &\to S(x+y) \\
(0*x) &\to 0 \\
S(x)*y &\to (x*y)+y \\
x*(y+z) &\to (x*y)+(x*z) \\
\\
fact(x) &\to ifact(x,S(0)) \\
\\
ifact(x,0) &\to 0 \\
ifact(x,y)+ifact(x,z) &\to ifact(x,y+z) \\
ifact(x,y)+ifact(x,z)+u &\to ifact(x,y+z)+u \\
ifact(x,y)*z &\to ifact(x,y*z) \\
\\
ifact(0,x) &\to x \\
ifact(S(x),y) &\to ifact(x,y+(x*y))
\end{aligned}
$$

The completion required 10 calls to the procedure *EP_order*. Cases where only comparisons between polynomials can be used are not counted.

## 7.2. Automatic Synthesis of an Iterative Fibonacci Function

Like in the case of factorial, one starts with a naive definition of Fibonacci numbers and, by completion, one generates a set of rules that gives an iterative or tail recursive function that can be used to compute Fibonacci numbers. Unlike *FACT*, the completion fails on an equation that cannot be oriented, but by that time all interesting rules have been generated.

One starts with the following set of equations, together with the associativity and commutativity of + and *:

$$
\begin{aligned}
0+x &= x \\
S(x)+y &= S(x+y) \\
0*x &= 0 \\
S(x)*y &= (x*y)+y \\
x*(y+z) &= (x*y)+(x*z) \\
\\
fib(0) &= 0 \\
fib(S(0)) &= S(0) \\
fib(S(S(x))) &= fib(x)+fib(S(x))
\end{aligned}
$$

$$ifib(x,y,z) = (fib(S(x)) * y) + (fib(x) * z)$$

$$\begin{aligned} ifib(x, y_1 + y_2, z_1 + z_2) &= ifib(x, y_1, z_1) + ifib(x, y_2, z_2) \\ (fib(S(x)) * y_1) + ifib(x, y_2, z) &= ifib(x, y_1 + y_2, z) \\ (fib(x) * z_1) + ifib(x, y, z_2) &= ifib(x, y, z_1 + z_2) \end{aligned}$$

The following three-level interpretations, which satisfy restrictions on the associativity and commutativity of + and *, are used:

$$\begin{aligned} [0]_1 &= 2 \\ [S]_1(X) &= X + 2 \\ [+]_1(X_1, X_2) &= X_1 + X_2 \\ [*]_1(X_1, X_2) &= X_1 \cdot X_2 \\ [fib]_1(X) &= 2^X \\ [ifib]_1(X_1, X_2) &= 2^{X_1+1}.X_2 + 2^{X_1}.X_3 \end{aligned}$$

$$\begin{aligned} [0]_2 &= 2 \\ [S]_2(X) &= X + 2 \\ [+]_2(X_1, X_2) &= X_1 + X_2 + 1 \\ [*]_2(X_1, X_2) &= X_1 \cdot X_2 \\ [fib]_2(X) &= 2^X \\ [ifib]_2(X_1, X_2) &= 2^{X_1+2}.X_2 + 2^{X_1}.X_3 \end{aligned}$$

$$\begin{aligned} [S]_3(X) &= X + 2 \\ [+]_3(X_1, X_2) &= X_1 \cdot X_2 \end{aligned}$$

The completion fails on the equation

$$(x_1 * fib(S(x_2))) + ifib(x_3, fib(x_2), x_4) = (x_4 * fib(x_3)) + ifib(x_2, x_1, fib(S(x_3)))$$

that obviously cannot be oriented. At the time of failure, the completion has generated the rewrite system

$$\begin{aligned} 0 + x &\to x \\ S(x) + y &\to S(x + y) \\ 0 * x &\to 0 \\ S(x) * y &\to (x * y) + y \\ x * (y + z) &\to (x * y) + (x * z) \end{aligned}$$

$$\begin{aligned} fib(0) &\to 0 \\ fib(S(0)) &\to S(0) \\ fib(S(S(x))) &\to fib(x) + fib(S(x)) \end{aligned}$$

$$\begin{aligned} (fib(S(x)) * y) + (fib(x) * z) &\to ifib(x, y, z) \\ (fib(S(x)) * y) + (fib(x) * z) + u &\to ifib(x, y, z) + u \end{aligned}$$

$$\begin{aligned} ifib(x, y_1, z_1) + ifib(x, y_2, z_2) &\to ifib(x, y_1 + y_2, z_1 + z_2) \\ u + ifib(x, y_1, z_1) + ifib(x, y_2, z_2) &\to u + ifib(x, y_1 + y_2, z_1 + z_2) \\ (fib(S(x)) * y_1) + ifib(x, y_2, z) &\to ifib(x, y_1 + y_2, z) \\ u + (fib(S(x)) * y_1) + ifib(x, y_2, z) &\to u + ifib(x, y_1 + y_2, z) \\ (fib(x) * z_1) + ifib(x, y, z_2) &\to ifib(x, y, z_1 + z_2) \\ u + (fib(x) * z_1) + ifib(x, y, z_2) &\to u + ifib(x, y, z_1 + z_2) \end{aligned}$$

$$ifib(0, y, z) \to y$$

$$
\begin{aligned}
ifib(S(x), y, z) &\rightarrow ifib(x, (y + z), y) \\
ifib(x, 0, z) &\rightarrow z * fib(x) \\
ifib(x, y, 0) &\rightarrow y * fib(S(x)) \\
u * ifib(x, y, z) &\rightarrow ifib(x, (y * u), (u * z)) \\
fib(x) + ifib(x, y, z) &\rightarrow ifib(x, y, S(z)) \\
u + fib(x) + ifib(x, y, z) &\rightarrow u + ifib(x, y, S(z)) \\
fib(S(x)) + ifib(x, y, z) &\rightarrow ifib(x, S(y), z) \\
u + fib(S(x)) + ifib(x, y, z) &\rightarrow u + ifib(x, S(y), z)
\end{aligned}
$$

All these rules are interesting properties of *fib* and *ifib*; the most interesting are perhaps

$$
\begin{aligned}
ifib(0, y, z) &\rightarrow y \\
ifib(S(x), y, z) &\rightarrow ifib(x, (y + z), y)
\end{aligned}
$$

which are tail-recursive definitions of *ifib*. On the other hand, the rule

$$ifib(x, 0, z) \rightarrow z * fib(x)$$

and rule

$$(fib(S(x)) * y) + (fib(x) * z) \rightarrow ifib(x, y, z)$$

show that elementary interpretations do not work like a precedence ordering which could not orient the rules this way. Indeed, with such orderings, the terms containing *ifib* have to be both in the right-hand side or both in the left-hand side. This will make the completion fail too early to generate the expected tail definition of *ifib*. *EP_order* was called 27 times in this completion.

## 8. Conclusion

EP is available as a part of ORME [Les90] and currently we are investigating how this method can be applied to automate proofs of termination of other rewrite systems. The method has some limits; the most obvious one is that interpretations are not easy to find and require expertise. The problem here is harder than for polynomial interpretations, since, in addition to the difficulty of finding an interpretation that actually can be used to prove the termination, the interpretation must also to be tractable for our inefficient procedure. For instance, all our examples required some tuning before working; nevertheless, the programme is very useful in this phase. The method has also theoretical limits. Indeed, for number theoretic functions (with notation of [CiL91]) defined using a 0-$S$ discipline, function $\{f\}$ (the actual function computed by the rewrite system) is related to function $[\![f]\!]$ with conditions that are made explicit in [CiL91] in the case of polynomial interpretations. Intuitively, we feel that the interpretation has to be larger in growth that the computed functions, but we do not know by how much. Actually, there are examples where the computed functions are elementary (exponential, for instance), but require a non-elementary function for a proof of termination. A similar result should hold for functions on lists with a *nil-cons* discipline, and we have an example of a definition of a permutation function (a $n^n$ function), where the only simple available interpretation contains a superexponential, more precisely, the function $e(2) = 2$ and $e(n + 1) = e(n)^{e(n)}$.

## Acknowledgements

The result of [CiL91], which we proved with Adam Cichon, was the starting point for this research. Alfons Geser mentioned to me the analogy between *transformation orderings* [BeL90, Ges90] and our use of proofs of positiveness in *polynomial interpretations* as described in [BCL87]. This comment was a source of inspiration for the implementation and helped me to see the proof of positiveness of a reachability problem for a specific rewrite system, namely $\mathscr{R} \oplus \mathscr{H} \oplus \mathscr{P}$. This analogy should be studied further and should lead to other implementations. I would like also to thank Nachum Dershowitz for his help in the preparation of this paper and the members of the research group EURECA (among them Adam Cichon, Isabelle Gnaedig, Miki Hermann, Dieter Hofbauer, and Gregory Kucherov) for many interactions.

This research was partly supported by PRC "programmation avancée et outils de l'intelligence artificielle" and by ESPRIT under Basic Research Working Group 3264, COMPASS.

## References

[AMS89] Avenhaus, J., Madlener, K., and Steinbach, J.: COMTES -- an experimental environment for the completion of term rewriting systems. In N. Dershowitz, editor, *Proceedings of the Third International Conference on Rewriting Techniques and Applications*, pages 542–546, Chapel Hill, NC, April 1989. Vol. 355 of *LNCS*, Springer, Berlin.

[Bac91] Bachmair, L.: *Canonical equational proofs.* Computer Science Logic, Progress in Theoretical Computer Science. Birkhäuser Verlag AG, 1991.

[BCL87] Ben Cherifa, A. and Lescanne, P.: Termination of rewriting systems by polynomial interpretations and its implementation. *Science of Computer Programming*, 9(2):137–160, October 1987.

[BDP89] Bachmair, L., Dershowitz, N. and Plaisted, D.: Completion without failure. In H. Aït-Kaci and M. Nivat, editors, *Resolution of Equations in Algebraic Structures, Volume 2: Rewriting Techniques*, pages 1–30. Academic Press, 1989.

[BeL90] Bellegarde, F. and Lescanne, P.: Termination by completion. *Applicable Algebra in Engineering, Communication and Computation*, 1(2):79–96, 1990.

[BaP85] Bachmair, L. and Plaisted, D. A.: Termination orderings for associative-commutative rewriting systems. *Journal of Symbolic Computation*, 1:329–349, 1985.

[CiL91] Cichon, E. A. and Lescanne, P.: Polynomial Interpretations and the Complexity of Algorithms. Rapport interne 91-R-151, Centre de Recherche en Informatique de Nancy, Vandœuvre-lès-Nancy, 1991. to be presented at CADE'92.

[Del91] Delor, C.: Terminaison des systèmes de réécriture, application à la transformation des formules équationnelles. Thèse de l'Université de Paris VII, June 1991.

[DeM92] Dershowitz, N. and Mitra, S.: RPO for AC-termination. Unpublished manuscript, U. of Illinois, 1992.

[For84] Forgaard, R.: A program for generating and analyzing term rewriting systems. Technical Report 343, Laboratory for Computer Science, Massachusetts Institute of Technology, 1984. Master's Thesis.

[Ges90] Geser, A.: Relative termination. Dissertation thesis, Universität Passau (Germany), 1990.

[GaG89] Garland, S. J. and Guttag, J. V.: An overview of LP, the Larch Prover. In N. Dershowitz, editor, *Proceedings 3rd Conference on Rewriting Techniques and Applications, Chapel Hill (North Carolina, USA)*, volume 355 of *LNCS*, pages 137–151. Springer-Verlag, April 1989.

[GnL86] Gnaedig, I. and Lescanne, P.: Proving termination of associative rewriting systems by rewriting. In J. Siekmann, editor, *Proceedings 8th International Conference on Automated Deduction, Oxford (UK)*, volume 230 of *LNCS*, pages 52–61. Springer-Verlag, 1986.

[HsR87] Hsiang, J. and Rusinowitch, M.: On word problem in equational theories. In Th. Ottmann, editor, *Proceedings of 14th International Colloquium on Automata, Languages and Programming, Karlsruhe (Germany)*, volume 267 of *LNCS*, pages 54–71. Springer-Verlag, 1987.

[KaL80] Kamin, S. and Lévy, J.-J.: Attempts for generalizing the recursive path ordering. Unpublished manuscript, 1980.

[KSZ90] Kapur, D., Sivakumar, G. and Zhang, H.: A new method for proving termination of AC-rewrite systems. In *Proceedings 10th Conf. on Foundations of Software Technology and Theoretical Computer Science*, volume 472 of *LNCS*, pages 133–148. Springer-Verlag, 1990.

[Lan79] Lankford, D. S.: On proving term rewriting systems are noetherian. Technical report, Louisiana Tech. University, Mathematics Dept., Ruston LA, 1979.

[Les83] Lescanne, P.: Computer experiments with the REVE term rewriting systems generator. In *Proceedings of 10th ACM Symposium on Principles of Programming Languages*, pages 99–108. Association for Computing Machinery, 1983.

[Les89] Lescanne, P.: Completion procedures as transition rules + control. In M. Diaz and F. Orejas, editors, *TAPSOFT'89*, volume 351 of *LNCS*, pages 28–41. Springer-Verlag, 1989.

[Les90] Lescanne, P.: Implementation of completion by transition rules + control: ORME. In H. Kirchner and W. Wechler, editors, *Proceedings 2nd International Workshop on Algebraic and Logic Programming, Nancy (France)*, volume 463 of *LNCS*, pages 262–269. Springer-Verlag, 1990.

[Pét67] Péter, R.: *Recursive Functions.* Academic Press, 1967.

[WAL89] Weis, P., Aponte, M. V., Laville, A., Mauny, M. and Suárez, A.: The CAML reference manual. Technical report, Projet Formel, INRIA-ENS, 1989. Version 2.6.

*Received June 1993*

*Accepted in revised form January 1994 by U.H.M. Martin*