Elementary Interpretations in Proofs of Termination

Pierre 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

Abstract

This paper 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 mechanization of the comparison of expressions built with polynomials and exponentials.

1 Proving termination of rewrite systems by interpretation

Automating proofs of termination, especially proofs of termination of rewrite systems containing associative and commutative (AC) operators, is a major challenge in programming and theorem proving. To prove termination, we propose to use elementary functions, defined by addition, multiplication and exponentiation and to mechanize the process of proving that a function bounds another for every value of the variables. The idea of using elementary functions for proving termination is quite natural and was basically used since the beginning of computer science [Itu67]. Filman (cited in the chapter on *Rewrite Systems* of the *Handbook of Theoretical Computer Science* [DJ90]) proposed in 1978 to use elementary interpretations to prove the termination of the following rewrite system which computes disjunctive normal forms, a typical rewrite system for automated deduction:

$$\begin{array}{rcl}
Not(Not(x)) & \to & x \\
Not(x \vee y) & \to & Not(x) \wedge Not(y) \\
Not(x \wedge y) & \to & Not(x) \vee Not(y) \\
x \wedge (y \vee z) & \to & (x \wedge y) \vee (x \wedge z) \\
(y \vee z) \wedge x & \to & (y \wedge x) \vee (z \wedge x)
\end{array}$$

which can be proved using the following interpretation:

$$\begin{array}{rcl}
[\![Not(x)]\!] & = & 2^{[\![x]\!]} \\
[\![x \vee y]\!] & = & [\![x]\!] + [\![y]\!] + 2 \\
[\![x \wedge y]\!] & = & [\![x]\!][\![y]\!] + 1
\end{array}$$

Indeed

$$\begin{array}{rcl}
[\![Not(Not(x))]\!] = 2^{2^{[\![x]\!]}} & > & [\![x]\!] \\
[\![Not(x \vee y)]\!] = 2^{[\![x]\!]+[\![y]\!]+2} & > & 2^{[\![x]\!]}2^{[\![y]\!]} + 1 = [\![Not(x) \wedge Not(y)]\!] \\
[\![Not(x \wedge y)]\!] = 2^{[\![x]\!][\![y]\!]+1} & > & 2^{[\![x]\!]} + 2^{[\![y]\!]} + 2 \\
[\![x \wedge (y \vee z)]\!] = [\![x]\!][\![y]\!] + [\![x]\!][\![z]\!] + 2[\![x]\!] + 1 & > & [\![x]\!][\![y]\!] + [\![x]\!][\![z]\!] + 4 = [\![x \wedge (y \vee z)]\!] \\
[\![(y \vee z) \wedge x]\!] = [\![y]\!][\![x]\!] + [\![z]\!][\![x]\!] + 2[\![x]\!] + 1 & > & [\![y]\!][\![x]\!] + [\![z]\!][\![x]\!] + 4 = [\![(y \vee z) \wedge x]\!]
\end{array}$$

The main goal of this paper is to show how all these inequalities can be proved automatically.

For the point of view of functional programming, consider now the rewrite system $FACT$ (a program that describes a *factorial* function)

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

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 [KL80] cannot handle associative and commutative rewrite systems. Classical extensions of the recursive path orderings [BP85, GL86] do not allow the precedence $* > + > S$ that would be necessary in this case. Delor and Puel [Del91, DP93] relaxed this restriction on the precedence (see also [DM92, Bac92, KSZ90, RN93]), but there are still rewrite systems (including the *fib* example of Section 8) 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 a full specification of the function *fact* including a specification of $+$ and $*$ cannot have a proof of termination done by polynomial interpretations [CL92]. This absence of implemented mechanized 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* [HR87, BDP89].

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

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

and

$$[\![0]\!]_2 \quad = \quad 2$$

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

saying that $s > t$ if ($[\![s]\!]_1 >_{\mathbf{N}} [\![t]\!]_1$) or ($[\![s]\!]_1 \geq_{\mathbf{N}} [\![t]\!]_1$ 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 $>_{\mathbf{N}}$ and $\geq_{\mathbf{N}}$ are the comparisons 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 $*$. In all the examples proposed in this paper, $[\![\_]\!]_2$ is called only when the first interpretations are equal (see [Les93] for case where $\geq_{\mathbf{N}}$ is used.) The proof of termination requires more than the functions allowed by these restrictions and cartesian product of two or many levels of interpretation preserves stability by associativity and commutativity.

The method described in [BCL87] for mechanizing 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 mechanization 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. A preliminary version of this article was presented at the *3rd conference on Algebraic and Logic Programming (Volterra, Italy)* [Les92].

2 Interpretations and termination of rewrite systems

The method for proving termination of rewrite systems, called "interpretations", 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. The interested reader is advised to read [Zan92]. Lankford for instance proposed to associate polynomials [Lan79]. 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 $l \rightarrow r$, the function $[\![l]\!]_1$ is larger than the function $[\![r]\!]_1$, then one can prove the termination of the rewrite system. Here, "larger" means that for each instantiation ι 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 $[\![r]\!]_1(\iota(X_1),\ldots,\iota(X_n))$. c is taken as the minimum of the $[\![t]\!]_1$'s for all the terms t of the given signature. If one takes again $[\![\_]\!]_1$ of Section 1, the inequality

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

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

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 $S(x) + y \rightarrow S(x + y)$ 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 mechanized [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 mechanize 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 [GG89] or, ORME [Les90].

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 [WAL+89]) 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]. ORME has now been upgraded to include the method described in this paper.

3 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 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 X_i^n. In what follows, the 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 $\mathcal{R}$ modulo associativity and commutativity of '+' and '·' with four rules (the rules for rings)

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

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

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

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

$$\begin{array}{rcl} 1 & \rightsquigarrow & 0 \\ \mathsf{X} & \rightsquigarrow & 1 \end{array}$$

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 $(\to \cup \hookrightarrow \cup \rightsquigarrow)^* \circ \rightsquigarrow \circ (\to \cup \hookrightarrow \cup \rightsquigarrow)^*$. It is easy to see that $P >_{\mathbf{N}} Q$ if $P \Longrightarrow \circ \overset{*}{\leftarrow} Q$. In other words P is greater than Q if one reaches an $\mathcal{R}$-reduced form of Q from P by steps of $\to$ or $\hookrightarrow$ or $\rightsquigarrow$ with at least one step of $\rightsquigarrow$. Another relation is interesting, especially when one composes lexicographically polynomial orderings, namely the relation $(\to \cup \hookrightarrow)^*$. One gets $P \geq_{\mathbf{N}} Q$ if $P(\to \cup \hookrightarrow)^* Q$.

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),

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

Let L be the left-hand side and R, the right-hand side, and let 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 $\mathsf{UXYZ} + \mathsf{UXYZ} + \mathsf{X^2YZ} + \mathsf{XZ}$. Reducing Y to 1 by $\rightsquigarrow$ in the second monomial gives $\mathsf{UXYZ} + \mathsf{UXZ} + \mathsf{X^2YZ} + \mathsf{XZ}$. Reducing X in the third monomial gives $\mathsf{UXYZ} + \mathsf{UXZ} + \mathsf{XYZ} + \mathsf{XZ}$ which is equal to R. Therefore $L \Longrightarrow R$ which implies $L >_{\mathbf{N}} R$. Note that we can check easily that $R = \mathsf{UXYZ}+\mathsf{UXZ}+\mathsf{XYZ}+\mathsf{XZ}$ is embedded in $\mathsf{UXYZ}+\mathsf{UXYZ}+\mathsf{X^2YZ}+\mathsf{XZ}$. 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 $\mathcal{R}$-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. This is also implemented directly i.e., without explicit rewriting.

The set $\mathcal{H}$ contains essentially the heuristics for decreasing a polynomial to a smaller one, hence its name. Therefore it can be extended by other rules, provided $P \hookrightarrow Q$ implies $P \geq_{\mathbf{N}} Q$. For instance, rules like

$$\begin{array}{rcl} \mathsf{XX} + \mathsf{YY} & \hookrightarrow & \mathsf{XY} + \mathsf{XY} \\ (x \cdot \mathsf{XX}) + (x \cdot \mathsf{YY}) & \hookrightarrow & (x \cdot \mathsf{XY}) + (x \cdot \mathsf{XY}). \end{array}$$

remind that here x stands for polynomials, thus the two rules are basically different. This allows one to prove $\mathsf{XX} + \mathsf{YY} + 1 - \mathsf{XY} - \mathsf{XY} > 0$. Zantema (private communication) mentioned me an example of integers with addition, subtraction (t), multiplication and squaring (Sqr) where such an argument is necessary, namely the system:

$$\begin{array}{rcl} S(P(x)) & \to & x \\ x + 0 & \to & x \\ x + S(y) & \to & S(x+y) \\ x + P(y) & \to & P(x+y) \\ x - 0 & \to & x \\ x - S(y) & \to & P(x-y) \\ x - P(y) & \to & S(x-y) \\ S(x) - S(y) & \to & x - y \\ P(x) - P(y) & \to & x - y \end{array} \qquad \begin{array}{rcl} P(S(x)) & \to & x \\ x * 0 & \to & 0 \\ x * S(y) & \to & a(x*y, x) \\ x * P(y) & \to & t(x*y, x) \\ Sqr(0) & \to & 0 \\ Sqr(S(x)) & \to & S(Sqr(x) + (x+x)) \\ Sqr(P(x)) & \to & S(Sqr(x) - (x+x)) \\ Sqr(x) - Sqr(y) & \to & (x+y)*(x-y) \end{array}$$

which can be proved to terminate using the following interpretation:

$$\begin{array}{rcl} [\![0]\!]_1 & = & 2 \\ [\![S(x)]\!]_1 & = & [\![x]\!] + 3 \\ [\![P(x)]\!]_1 & = & [\![x]\!] + 3 \\ [\![x+y]\!]_1 & = & [\![x]\!] + 2[\![y]\!] \\ [\![*]\!]_1(X_1, X_2) & = & [\![x]\!] \cdot [\![y]\!] \\ [\![x-y]\!]_1 & = & [\![x]\!] + 2[\![y]\!] \\ [\![Sqr(x)]\!]_1(X) & = & [\![x]\!]^4 \end{array}$$

Especially,

$$\begin{aligned} [\![Sqr(x) - Sqr(y)]\!] = [\![x]\!]^4 + 2[\![y]\!]^4 = ([\![x]\!]^2)^2 + ([\![y]\!]^2)^2 + [\![y]\!]^4 &\geq 2[\![x]\!]^2[\![y]\!]^2 + [\![y]\!]^4 \\ \geq 4[\![x]\!]^2[\![y]\!] + 4[\![y]\!]^2 > [\![x]\!]^2[\![y]\!] &+ 2[\![x]\!]^2[\![y]\!] + 4[\![y]\!]^2 \\ > [\![x]\!]^2 + 4[\![x]\!][\![y]\!] + 4[\![y]\!]^2 &= [\![(x+y)*(x-y)]\!] \end{aligned}$$

Zantema can prove that any proof of termination based on polynomial interpretations requires to use the above heuristic. Another example due to Zimmerman (see system ϕ in [Les93]) seems also to show that if one wants to be really general an elaborate computer algebra system based on Tarsky decision procedure and derived from Collin and MacCallum algorithm may be necessary [Rou91]. Anyway our simple and cheap system is worth being implemented in rewriting laboratories.

4 The theoretical limits of polynomial interpretations

Derivation length is the function that associates the size of a term with the length of the longest derivation starting from this term. It can be shown that proofs by polynomial interpretations make strong constraints on the derivation length. Indeed it was proved by Hofbauer and Lautemann [HL89] and Geupel [Geu88] that derivation length is $O(2^{2^n})$ and $\Omega(2^{2^n})$. This is illustrated by the following rewrite system:

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

Its termination can be proved by the polynomial interpretation:

$$\begin{array}{rcl} [\![0]\!] & = & 2 \\ [\![s]\!](X) & = & X+1 \\ [\![c]\!](X,Y) & = & 2X^2+3Y. \end{array}$$

The normal form of $c(s^m(0), s^n(0))$ is $s^{m^2+2n}(0)$ with computation length $\frac{m(m+1)}{2} + n + 1$. Therefore the term

$$c(c(\ldots c(c(s(s(0)),0),0)\ldots,0),0)$$

with $k+1$ occurrences of c is of size $2k+5$ and reduces to $c(s^{2^{2^k}}(0),0)$ which itself normalizes to $s^{2^{2^{k+1}}}(0)$ in $2^{2^{k+1}-1} + 2^{2^k-1} + 1$ steps that is larger than 2^{2^k} steps.

Often rewrite systems are used for computing values of natural functions defined by a method called the *constructor discipline* [O'D85]. The constructor discipline partitions the function symbols into two classes: the *constructor symbols* 0 and s used to describe natural numbers and the *defined symbols* used to perform computations and associated with each computed functions. In addition, each left-hand side of a rule contains only constructors and variables except at the top where there is a defined symbol. For such rewrite systems the *derivation length* is not the adequate measure of the complexity of the function. For instance, suppose that a computation starts with

$$f(s^{n_1}(0), s^{n_2}(0), \ldots, s^{n_p}(0))$$

and ends with

$$s^{\{f\}(n_1,\ldots,n_p)}(0),$$

which is the value associated with the function f for the given input. It seems that the complexity of the rewrite system should be better seen as the complexity of the function

$$(n_1, \ldots, n_p) \mapsto \{f\}(n_1, \ldots, n_p)$$

since $\{f\}(n_1, \ldots, n_p)$ is the intended value of the function described by the rewrite system. For instance, with the above function c, one gets $\{c\}(m,n) = m^2+2n$ which is polynomial. Adam Cichon and the author have proven [CL92] that if the function is described by a rewrite system with a polynomial interpretation based termination proof, then $\{f\}$ is polynomial in the size of the starting terms or equivalently is polynomial in $n_1, n_2, \ldots, n_p$ unlike its derivation length which is doubly exponential. This result holds even for functions with exponential computation length, like

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

which is proved to terminate by the polynomial interpretation

$$\begin{array}{rcl} [\![0]\!] & = & 2 \\ [\![s]\!](X) & = & 3X \\ [\![g]\!](X,Y) & = & X+Y \end{array}$$

and computes $\{g\}(m,n) = n$ in 2^m steps.

5 Comparing elementary functions

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

$$\begin{array}{rcl}
\mathsf{EP_term} & ::= & \mathsf{0} \ \mid \ \mathsf{EP_monomial} + \mathsf{EP_term} \\[2ex]
\mathsf{EP_monomial} & ::= & \mathsf{1} \ \mid \ \mathsf{EP_component} \quad \mathsf{EP_monomial} \\[2ex]
\mathsf{EP_component} & ::= & \mathsf{String} \ \mid \ (\mathsf{EP_term})^{(\mathsf{EP_term})}
\end{array}$$

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

$$\mathcal{R}: \left\{ \begin{array}{lcl}
0 + x & \to & x \\
0 \cdot x & \to & 0 \\
1 \cdot x & \to & x \\
x \cdot (y+z) & \to & (x \cdot y) + (x \cdot z) \\[1ex]
x^{y+z} & \to & x^y \cdot x^z \\
(x \cdot y)^z & \to & x^z \cdot y^z \\
(x^y)^z & \to & x^{y \cdot z} \\
x^1 & \to & x \\
x^0 & \to & 1
\end{array} \right.$$

for reducing exponentials. System $\mathcal{H}$ is modified to include new facts about exponentials and binomials

$$\mathcal{H}: \left\{ \begin{array}{lcl}
X & \hookrightarrow & 1+1 \\
(x+y)^z & \hookrightarrow & x^z + y^z \\
x.(x+y)^z & \hookrightarrow & x.x^z + y.x^z + x.y^z \\
x^y & \hookrightarrow & x.y
\end{array} \right.$$

$\mathcal{P}$ is the same as in Section 2, that is:

$$\mathcal{P}: \left\{ \begin{array}{lcl}
1 & \leadsto & 0 \\
X & \leadsto & 1
\end{array} \right.$$

and $P >_{\mathbf{N}} Q$ if $P \Longrightarrow \circ \overset{*}{\leftarrow} Q$ where $\Longrightarrow$ is $(\to \cup \hookrightarrow \cup \leadsto)^* \circ \leadsto \circ (\to \cup \hookrightarrow \cup \leadsto)^*$ as previously.

This three names have a specific meaning. Indeed $\mathcal{R}$ stands for *reduction* in the sense that its reduces each EP-term to its normal forms with respect to the underlying theory (rings for polynomials, rings plus exponentials for elementary functions). $\mathcal{P}$ stands for positiveness in the sense it is the real place such an argument is actually used. $\mathcal{H}$ stands for heuristics, since it is the part where most of the heuristics stays. In all the examples mentioned in this paper the first heuristic is enough, but for some applications the three others where useful.

6 Implementation

Data structure and implementation of $\mathcal{R}$

The above described method has been implemented in CAML as a part of ORME[1]. EP-terms are represented by the following data structure: EP-terms are lists of

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

$$
\begin{array}{rcl}
0 + x & \to & x \\
S(x) + y & \to & S(x + y) \\
0 * x & \to & 0 \\
S(x) * y & \to & (x * y) + y \\
x * (y + z) & \to & (x * y) + (x * z) \\
\mathit{fact}(0) & \to & S(0) \\
\mathit{fact}(S(x)) & \to & S(x) * \mathit{fact}(x) \\
\mathit{fib}(0) & \to & 0 \\
\mathit{fib}(S(0)) & \to & S(0) \\
\mathit{fib}(S(S(x))) & \to & \mathit{fib}(x) + \mathit{fib}(S(x)) \\
2\mathit{exp}(0) & \to & S(0) \\
2\mathit{exp}(S(x)) & \to & 2\mathit{exp}(x) + 2\mathit{exp}(x)
\end{array}
$$

with interpretations

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

Figure 1: The system NAT + DIST + FACT + FIB + 2EXP

EP-monomials and 0 is the empty list of EP-monomials. EP-monomials are lists 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. $\mathcal{R}$ is implemented directly by three functions called *EP_term_norm*, *EP_monomial_norm* and *EP_component_norm*, acting on EP-terms. Actually only $P \Longrightarrow Q{\downarrow_{\mathcal{R}}}$ is implemented, which means we only work on normal forms modulo $\mathcal{R}$. 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 $\mathcal{P}$.

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.

Implementation of $\mathcal{P}$

$\mathcal{P}$ 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

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

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

with interpretations

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

Figure 2: The system NAT + DIST + EXP

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.

Implementation of $\mathcal{H}$

$\mathcal{H}$ is implemented by choosing a letter X and replacing it by $1 + 1$, by choosing a power of the form $(x+y)^z$ and by replacing it by $x^z + z \cdot x^{z-1} + y^z$ (if this quantity is multiplied by x x^{-1} will disappear, else the whole expression is canceled) and by $x^z + y^z$, by choosing a power of the form m^x and replacing by $m \cdot x$. This is, in effect, a step of rewriting and contains most of the heuristic of the method. $\mathcal{H}$-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.

Three 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. Similarly, checking that the set of the letters of a term s is a subset of the letters of a term t is also cheap, therefore it is wise to check this property before proving that s is less than t. These tricks improved the efficiency of the implementation dramatically when used for orienting equalities, as in completion.

EP_less and EP_order

The main procedures 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*.

7 Examples

As an example, we tried on our computers using ORME the rewrite system for computing disjunctive normal forms. We used an interpretation which is not this proposed by Filman in order to work with our heuristics. This is a feature of the implementation: sometime the interpretations have to be adapted to ability of the system to make comparisons, We also tried $FACT$ and the rewrite systems given in Figures 1 and 2. Notice that the interpretations of $\vee$, $\wedge$, $+$ and $*$ satisfy the restrictions on AC-operators.

Figure 3 is a specification of binomial coefficients i.e., $C(x, y)$ stands for $\binom{x}{y}$ which is obviously a function with exponential growth. Two different interpretations give different computing time. The first (one level) interpretation is slower than the second (two level) one. In [CHL92] gives a confluent rewrite system for λ-calculus with explicit substitution (Figure 4) and they propose an elementary interpretation (Figure 5), for which they perform the computation by hand. ORME hopefully succeeded to prove the termination. In [Les93], we give other uses of elementary interpretations for proving termination of rewrite systems related to λ-calculus.

8 Use of elementary interpretations in associative and commutative 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. This method offers at least three features:

- it works in presence of associativity and commutativity,
- it can handle functions with exponential growth,
- it orients equations in a direction no precedence ordering would do and this allows often the user to get behavior he wishes, as for *fib*.

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

$$
\begin{array}{rcl}
x + 0 & \to & x \\
x * 0 & \to & 0 \\
x * (y + z) & \to & (x * y) + (x * z) \\
\\
fact(0) & \to & 1 \\
fact(1) & \to & 1 \\
fact(x + 1) & \to & (x + 1) * fact(x) \\
\\
C(x, 0) & \to & 1 \\
C(x, x) & \to & 1 \\
C(x + 1, y + 1) & \to & C(x, y + 1) + C(x, y) \\
C(x + y, y) * fact(x) * fact(y) & \to & fact(x + y) \\
C(x, x + y + 1) & \to & 0
\end{array}
$$

with interpretation

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

or interpretations

$$
\begin{array}{rclcrcl}
[\![0]\!]_2 & = & 2 & & & & \\
[\![1]\!]_2 & = & 2 & & & & \\
[\![+]\!]_2(X_1, X_2) & = & X_1 + X_2 & & [\![+]\!]_3(X_1, X_2) & = & X_1 + X_2 + 1 \\
[\![*]\!]_2(X_1, X_2) & = & X_1 \cdot X_2 & & [\![*]\!]_3(X_1, X_2) & = & X_1 \cdot X_2 \\
[\![fact]\!]_2(X_1, X_2) & = & X_1^{X_1} & & & & \\
[\![C]\!]_2(X_1, X_2) & = & (X_1 + X_2)^{X_1} & & & &
\end{array}
$$

Figure 3: Binomial coefficients

$$
\begin{array}{lrcl}
\text{(Beta)} & (\lambda a)b & \to & a[b \cdot id] \\
\text{(App)} & (ab)[s] & \to & a[s]b[s] \\
\text{(Lambda)} & (\lambda a)[s] & \to & \lambda(a[\Uparrow(s)]) \\
\text{(Clos)} & a[s][t] & \to & a[s \circ t] \\
\text{(VarShift1)} & V(\mathtt{n})[\uparrow] & \to & V(\mathrm{S}(\mathtt{n})) \\
\text{(VarShift2)} & V(\mathtt{n})[\uparrow \circ s] & \to & V(\mathrm{S}(\mathtt{n}))[s] \\
\text{(FVarCons)} & V(\mathtt{1})[a \cdot s] & \to & a \\
\text{(FVarLift1)} & V(\mathtt{1})[\Uparrow(s)] & \to & V(\mathtt{1}) \\
\text{(FVarLift2)} & V(\mathtt{1})[\Uparrow(s) \circ t] & \to & V(\mathtt{1})[t] \\
\text{(RVarCons)} & V(\mathrm{S}(\mathtt{n}))[a \cdot s] & \to & V(\mathtt{n})[s] \\
\text{(RVarLift1)} & V(\mathrm{S}(\mathtt{n}))[\Uparrow(s)] & \to & V(\mathtt{n})[s \circ \uparrow] \\
\text{(RVarLift2)} & V(\mathrm{S}(\mathtt{n}))[\Uparrow(s) \circ t] & \to & V(\mathtt{n})[s \circ (\uparrow \circ t)] \\
\text{(AssEnv)} & (s \circ t) \circ u & \to & s \circ (t \circ u) \\
\text{(MapEnv)} & (a \cdot s) \circ t & \to & a[t] \cdot (s \circ t) \\
\text{(ShiftCons)} & \uparrow \circ (a \cdot s) & \to & s \\
\text{(ShiftLift1)} & \uparrow \circ \Uparrow(s) & \to & s \circ \uparrow \\
\text{(ShiftLift2)} & \uparrow \circ (\Uparrow(s) \circ t) & \to & s \circ (\uparrow \circ t) \\
\text{(Lift1)} & \Uparrow(s) \circ \Uparrow(t) & \to & \Uparrow(s \circ t) \\
\text{(Lift2)} & \Uparrow(s) \circ (\Uparrow(t) \circ u) & \to & \Uparrow(s \circ t) \circ u \\
\text{(LiftEnv)} & \Uparrow(s) \circ (a \cdot t) & \to & a \cdot (s \circ t) \\
\text{(IdL)} & id \circ s & \to & s \\
\text{(IdR)} & s \circ id & \to & s \\
\text{(LiftId)} & \Uparrow(id) & \to & id \\
\text{(Id)} & a[id] & \to & a
\end{array}
$$

Figure 4: The rewrite system $\lambda\sigma_{\Uparrow}$ (Hardin and Lévy)

$$
\begin{array}{rcl@{\qquad}rcl}
[\![V(\mathtt{n})]\!]_1 & = & 2^{[\![\mathtt{n}]\!]_1} & [\![V(\mathtt{n})]\!]_2 & = & [\![\mathtt{n}]\!]_2 \\
[\![\mathrm{S}(\mathtt{n})]\!]_1 & = & [\![\mathtt{n}]\!]_1 + 1 & [\![\mathrm{S}(\mathtt{n})]\!]_2 & = & [\![\mathtt{n}]\!]_2 \\
[\![\mathtt{1}]\!]_1 & = & 2 & & & \\
[\![ab]\!]_1 & = & [\![a]\!]_1 + [\![b]\!]_1 & [\![ab]\!]_2 & = & [\![a]\!]_2 + [\![b]\!]_2 + 1 \\
[\![\lambda a]\!]_1 & = & [\![a]\!]_1 + 2 & & & \\
[\![a[s]\,]\!]_1 & = & [\![a]\!]_1 [\![s]\!]_1 & [\![a[s]\,]\!]_2 & = & ([\![a]\!]_2 [\![s]\!]_2) + [\![a]\!]_2 \\
[\![a \cdot s]\!]_1 & = & [\![a]\!]_1 + [\![s]\!]_1 & [\![a \cdot s]\!]_2 & = & [\![a]\!]_2 + [\![s]\!]_2 + 1 \\
[\![\uparrow]\!]_1 & = & 2 & [\![\uparrow]\!]_2 & = & 2 \\
[\![id]\!]_1 & = & 2 & [\![id]\!]_2 & = & 2 \\
[\![s \circ t]\!]_1 & = & [\![s]\!]_1 [\![t]\!]_1 & [\![s \circ t]\!]_2 & = & ([\![s]\!]_2 [\![t]\!]_2) + [\![s]\!]_2 \\
[\![\Uparrow(s)]\!]_1 & = & [\![s]\!]_1 & [\![\Uparrow(s)]\!]_2 & = & 4[\![s]\!]_2
\end{array}
$$

Figure 5: Interpretations for proving the termination of $\sigma_{\Uparrow}$

$*$ and $+$, which are given to be associative and commutative:

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

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 interpretations):

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

The completion ends and yields the system:

$$\begin{array}{rcl} (0+x) & \to & x \\ S(x)+y & \to & S(x+y) \\ (0*x) & \to & 0 \\ S(x)*y & \to & (x*y)+y \\ x*(y+z) & \to & (x*y)+(x*z) \\[1ex] \mathit{fact}(x) & \to & \mathit{ifact}(x,S(0)) \\[1ex] \mathit{ifact}(x,0) & \to & 0 \\ \mathit{ifact}(x,y)+\mathit{ifact}(x,z) & \to & \mathit{ifact}(x,y+z) \\ \mathit{ifact}(x,y)+\mathit{ifact}(x,z)+u & \to & \mathit{ifact}(x,y+z)+u \\ \mathit{ifact}(x,y)*z & \to & \mathit{ifact}(x,y*z) \\[1ex] \mathit{ifact}(0,x) & \to & x \\ \mathit{ifact}(S(x),y) & \to & \mathit{ifact}(x,y+(x*y)) \end{array}$$

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

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 $*$:

$$\begin{array}{rcl} 0+x & = & x \\ S(x)+y & = & S(x+y) \\ 0*x & = & 0 \\ S(x)*y & = & (x*y)+y \end{array}$$

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

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

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

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

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

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

$$\begin{aligned}
[\![0]\!]_2 &= 2 \\
[\![S]\!]_2(X) &= X+2 \\
[\![+]\!]_2(X_1,X_2) &= X_1+X_2+1 \\
[\![*]\!]_2(X_1,X_2) &= X_1\cdot X_2 \\
[\![\mathit{fib}]\!]_2(X) &= 2^{X+1} \\
[\![\mathit{ifib}]\!]_2(X_1,X_2,X_3) &= 2^{X_1}.(X_2+X_3+4)
\end{aligned}
\qquad
\begin{aligned}
[\![S]\!]_3(X) &= X+2 \\
[\![+]\!]_3(X_1,X_2) &= X_1\cdot X_2
\end{aligned}$$

The completion fails on the equation

$$(x_1 * \mathit{fib}(S(x_2))) + \mathit{ifib}(x_3,\mathit{fib}(x_2),x_4) = (x_4 * \mathit{fib}(x_3)) + \mathit{ifib}(x_2,x_1,\mathit{fib}(S(x_3)))$$

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

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

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

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

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

$$\begin{array}{rcl}
u + (\mathit{fib}(S(x)) * y_1) + \mathit{ifib}(x, y_2, z) & \rightarrow & u + \mathit{ifib}(x, y_1 + y_2, z) \\
(\mathit{fib}(x) * z_1) + \mathit{ifib}(x, y, z_2) & \rightarrow & \mathit{ifib}(x, y, z_1 + z_2) \\
u + (\mathit{fib}(x) * z_1) + \mathit{ifib}(x, y, z_2) & \rightarrow & u + \mathit{ifib}(x, y, z_1 + z_2) \\
\\
\mathit{ifib}(0, y, z) & \rightarrow & y \\
\mathit{ifib}(S(x), y, z) & \rightarrow & \mathit{ifib}(x, (y + z), y) \\
\\
\mathit{ifib}(x, 0, z) & \rightarrow & z * \mathit{fib}(x) \\
\mathit{ifib}(x, y, 0) & \rightarrow & y * \mathit{fib}(S(x)) \\
u * \mathit{ifib}(x, y, z) & \rightarrow & \mathit{ifib}(x, (y * u), (u * z)) \\
\mathit{fib}(x) + \mathit{ifib}(x, y, z) & \rightarrow & \mathit{ifib}(x, y, S(z)) \\
u + \mathit{fib}(x) + \mathit{ifib}(x, y, z) & \rightarrow & u + \mathit{ifib}(x, y, S(z)) \\
\mathit{fib}(S(x)) + \mathit{ifib}(x, y, z) & \rightarrow & \mathit{ifib}(x, S(y), z) \\
u + \mathit{fib}(S(x)) + \mathit{ifib}(x, y, z) & \rightarrow & u + \mathit{ifib}(x, S(y), z)
\end{array}$$

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

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

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

$$\mathit{ifib}(x, 0, z) \quad \rightarrow \quad z * \mathit{fib}(x)$$

and rule

$$(\mathit{fib}(S(x)) * y) + (\mathit{fib}(x) * z) \quad \rightarrow \quad \mathit{ifib}(x, y, z)$$

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 would make the completion fail too early to generate the expected tail definition of *ifib*. *EP_order* was called 27 times in this completion.

9 Conclusion

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 program is very useful in this phase. The method has also theoretical limits as we showed in section 4. Intuitively, we feel that the interpretation has to be larger in growth than 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 (an 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 [CL92], which we proved with Adam Cichon, was the starting point for this research. Alfons Geser mentioned to me the analogy between *transformation orderings* [BL90, 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 $\mathcal{R} \oplus \mathcal{H} \oplus \mathcal{P}$. 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.

References

[AMS89] J. Avenhaus, K. Madlener, and J. Steinbach. COMTES — an experimental environment for the completion of term rewriting systems. In N. Dershowitz, editor, *Proceedings 3rd Conference on Rewriting Techniques and Applications, Chapel Hill (N.C., USA)*, volume 355 of *Lecture Notes in Computer Science*, pages 542–546. Springer-Verlag, April 1989.

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

[Bac92] L. Bachmair. Associative-commutative reduction orderings. *Information Processing Letters*, 43(1):21–28, August 1992.

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

[BDP89] L. Bachmair, N. Dershowitz, and D. Plaisted. 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.

[BL90] F. Bellegarde and P. Lescanne. Termination by completion. *Applicable Algebra in Engineering, Communication and Computation*, 1(2):79–96, 1990.

[BP85] L. Bachmair and D. A. Plaisted. Termination orderings for associative-commutative rewriting systems. *Journal of Symbolic Computation*, 1:329–349, 1985.

[CHL92] P.-L. Curien, Th. Hardin, and J.-J. Lévy. Confluence properties of weak and strong calculi of explicit substitutions. RR 1617, INRIA, Rocquencourt, February 1992.

[CL92] E. A. Cichon and P. Lescanne. Polynomial interpretations and the complexity of algorithms. In D. Kapur, editor, *Proceedings 11th International Conference on Automated Deduction, Saratoga Springs (N.Y., USA)*, volume 607 of *Lecture Notes in Computer Science*, pages 139–147. Springer-Verlag, June 1992.

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

[DJ90] N. Dershowitz and J.-P. Jouannaud. *Handbook of Theoretical Computer Science*, volume B, chapter 6: Rewrite Systems, pages 244–320. Elsevier Science Publishers B. V. (North-Holland), 1990. Also as: Research report 478, LRI.

[DM92] N. Dershowitz and S. Mitra. RPO for AC-termination. Unpublished manuscript, U. of Illinois, 1992.

[DP93] C. Delor and L. Puel. Extension of the associative path ordering to a chain of associative symbols. In C. Kirchner, editor, *Proceedings 5th Conference on Rewriting Techniques and Applications, Montreal (Canada)*, volume 690 of *Lecture Notes in Computer Science*, pages 389–404. Springer-Verlag, 1993.

[For84] R. Forgaard. 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] A. Geser. *Relative Termination*. Dissertation, Universität Passau (Germany), 1990.

[Geu88] O. Geupel. Terminationbeweise bei termersetzungssytem, 1988. Diplomarbeit.

[GG89] S. J. Garland and John V. Guttag. An overview of LP, the Larch Prover. In N. Dershowitz, editor, *Proceedings 3rd Conference on Rewriting Techniques and Applications, Chapel Hill (N.C., USA)*, volume 355 of *Lecture Notes in Computer Science*, pages 137–151. Springer-Verlag, April 1989.

[GL86] I. Gnaedig and P. Lescanne. Proving termination of associative rewriting systems by rewriting. In J. Siekmann, editor, *Proceedings 8th International Conference on Automated Deduction, Oxford (UK)*, volume 230 of *Lecture Notes in Computer Science*, pages 52–61. Springer-Verlag, 1986.

[HL89] D. Hofbauer and C. Lautemann. Termination proofs and the length of derivations. In N. Dershowitz, editor, *Proceedings 3rd Conference on Rewriting Techniques and Applications, Chapel Hill (N.C., USA)*, volume 355 of *Lecture Notes in Computer Science*, pages 167–177. Springer-Verlag, April 1989.

[HR87] J. Hsiang and M. Rusinowitch. 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 *Lecture Notes in Computer Science*, pages 54–71. Springer-Verlag, 1987.

[Itu67] R. Iturriaga. *Contributions to mechanical mathematics*. PhD thesis, Carnegie-Mellon University, Pittsburgh, PA, USA, 1967.

[KL80] S. Kamin and J.-J. Lévy. Attempts for generalizing the recursive path ordering. Unpublished manuscript, 1980.

[KSZ90] D. Kapur, G. Sivakumar, and H. Zhang. 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 *Lecture Notes in Computer Science*, pages 133–148. Springer-Verlag, 1990.

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

[Les83] P. Lescanne. 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] P. Lescanne. Completion procedures as transition rules + control. In M. Diaz and F. Orejas, editors, *TAPSOFT'89*, volume 351 of *Lecture Notes in Computer Science*, pages 28–41. Springer-Verlag, 1989.

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

[Les92] P. Lescanne. Termination of rewrite systems by elementary interpretations. In Hélène Kirchner and G. Levi, editors, *Proceedings 3rd International Conference on Algebraic and Logic Programming, Volterra (Italy)*, volume 632 of *Lecture Notes in Computer Science*, pages 21–36. Springer-Verlag, September 1992.

[Les93] P. Lescanne. From $\lambda\sigma$ to $\lambda\upsilon$, a journey through calculi of explicit substitutions, July 1993.

[O'D85] M. J. O'Donnell. *Equational Logic as a Programming Language*. Foundation of Computing. MIT Press, 1985.

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

[RN93] A. Rubio and R. Nieuwenhuis. A precedence-based total ac-compatible ordering. In C. Kirchner, editor, *Proceedings 5th Conference on Rewriting Techniques and Applications, Montreal (Canada)*, volume 690 of *Lecture Notes in Computer Science*, pages 374–388. Springer-Verlag, 1993.

[Rou91] Jocelyne Rouyer. *Calcul formel en géométrie algébrique réelle appliqué à la terminaison des systèmes de réécriture*. Thèse de Doctorat d'Université, Université de Nancy I, 1991.

[WAL+89] P. Weis, M. V. Aponte, A. Laville, M. Mauny, and A. Suárez. The CAML reference manual. Technical report, Projet Formel, INRIA-ENS, 1989. Version 2.6.

[Zan92] H. Zantema. Termination of term rewriting by interpretation. Technical Report RUU-CS-92-14, Utrecht University, April 1992.