COMPUTER EXPERIMENTS WITH THE *REVE* TERM REWRITING SYSTEM GENERATOR

Pierre LESCANNE
Centre de Recherche en Informatique de Nancy
Campus scientifique - B.P. 239
54506 Vandoeuvre les Nancy Cedex, France

ABSTRACT

A term rewriting system generator called REVE is described. REVE builds confluent and uniformly terminating term rewriting systems from sets of equations. Particular emphasis is placed on mechanization of termination proof. Indeed, REVE is one of the few such systems which can actually be called automatic because termination is fully integrated into the algorithms. REVE uses an incremental termination method based on recursive decomposition ordering which constructs the termination proof step by step from the presentation of the set of equations and which requires little knowledge of termination methods from the user. All examples from this paper are taken from abstract data type specifications.

KEY-WORDS

Equational Theories, Term Rewriting Systems, Abstract Data Types, Termination, Word Problem, Induction, Theorem Prover, Program Verifier.

1. THE MAIN CONCEPTS

The specification of an abstract data type can be regarded as a set of equations describing the operations of the type and their relations to one another. Sets of such equations are special cases of elementary and universal equational theories ; i.e., universally quantified equalities between expressions. Theorems about abstract data types are usually proven to check that a specification has an expected property. Proofs of theorems in equational theories can also be of use to program verifiers when verifying programs that incorporate abstract data types. Therefore, efficient and highly automated theorem provers are needed for such theories. That is the aspect of equational theories that concerns us here.

The statements and theorems under consideration are equalities between well-formed expressions. Theorems in equational theories are divided into two categories depending on the way they are proven or, what is strongly related, on the family of models in which they are true. Theorems of the first kind are called *equational* theorems; they are proven by replacing expressions by equal expressions with respect to the equations. According to the Birkhoff Theorem, these statements are provable if and only if they are valid equalities; i.e., they are true in any model of the equations. The theorems in the second category are called *inductive* theorems; their proofs require an induction rule in addition to equational reasoning because they are statements that are valid in the family of models generated by the operations and the constants. These models are variously called standard, prime algebras or finitely generated algebras.

1.1. Proofs of equational theorems

The fundamental decision problem for equational theories is the word problem. It consists of finding a decision procedure for proving and disproving identities from a set E of equations. Although the word problem is generally unsolvable, e.g., when E contains the equations of certain semigroups (Post [18]) or groups (Novikov [16]), recent studies and experiments have shown that many of the word problems for abstract data types in particular and abstract algebras in general are solvable (not only theoretically, but practically) by a uniform methodology based upon term rewriting methods. In the past, some apparent intractability and/or inefficiency has frequently been due to ignorance of how to use the equations or generalize them appropriately.

This work was done while the author was visiting the laboratory for Computer Science of the Massachussetts Institute of Technology. During this period he was supported by the National Science Foundation grant 811 9846, by DARPA (N00014-75-C 0661), by the Centre National de la Recherche Scientifique, and by the Greco Programmation.

A *term rewriting system* is a set of oriented equations or rewrite rules that are always used from left to right. In this framework, the method of proving an equational theorem like $A=B$ is to rewrite A and B using the rewrite rules until one gets irreducible terms A^* and B^*. If A^* and B^* are the same then $A=B$ is valid (see [8] for more details). The following question naturally arises: Is this method a decision procedure? To answer that, two problems must be addressed.

1. Is the irreducible expression A^*, associated with A , unique?

2. Does the process of rewriting an expression A always terminate?

1.2. Proving confluence of term rewriting systems

A positive answer to the second question follows from the *confluence* of the term rewriting system (also called the Church-Rosser property or diamond lemma property). That means that if A rewrites to B and C, there exists an expression D such that B and C rewrite to D (using the relation of the term rewriting system zero or more times). A computer program checking for confluence was developed by Knuth and Bendix [11] based upon the work of Evans [2]. A term rewriting system is *locally confluent* if for any A which rewrites to B and C by using a rewriting relation once in each case, there exists a D such that B and C rewrite to D. We can deduce confluence from local confluence if we have *uniform termination*. A term rewriting system is uniformly terminating if there exists no infinite chain $A_0 \to A_1 \to \ldots \to A_n \to \ldots$ where $\to$ is the rewriting relation. Therefore, uniform termination is a central issue in term rewriting systems and in proof systems for equational theories, not only because of the first question above, but also because of its necessity in guaranteeing that confluence follows from local confluence. We says that a term rewriting system is *convergent* if it is uniformly terminating and confluent. A finite convergent term rewriting system is the basis of a decision procedure for its associated equational theory.

Given a set of equations, the corresponding set of rules is not always convergent. For instance, the set of equations

$x*e = x$,
$x*i(x) = e$,
$(x*y)*z = x*(y*z)$, and
$x/y = x*i(y)$,

which defines the class of all groups with right division, may be converted into the term rewriting system

$x*e \to x$
$x*i(x) \to e$
$(x*y)*z \to x*(y*z)$
$x/y \to x*i(y)$

which is not convergent, as are none of the systems obtained by orienting these four equations in various ways. For example e and $x*(y*i(x*y))$ are two irreducible terms obtained by rewriting the term $(x*y)*i(x*y)$. The Knuth-Bendix algorithm attemps to transform a set of equations into an equivalent convergent term rewriting system. *Equivalent* means that the reduction method based on the convergent term rewriting system proves the same equational theorems as the method based on the original set of equations. In other words, the resulting term rewriting system is a decision procedure for the original equational theory. The Knuth-Bendix algorithm generates new equations which are oriented to make new rules. The orientation of an equation is done by a specific algorithm which checks that the rule preserves the uniform termination of the entire system. By using these new rules some old rules may be modified or even collapse and disappear. For example, with the above group equations, the algorithm generates the rule $x*(y*i(x*y)) \to e$ which will disappear when the rule $i(x*y) \to i(y)*i(x)$ is generated. It turns out that we can generate two distinct but equivalent convergent term rewriting systems from the above example depending on which operation, $*$ or $/$, plays the main role (see Fig.1, Fig.2 and Appendix II for a proof of termination). The second one is new and was generated by REVE.

$x*e \to x$
$e*x \to x$
$x*i(x) \to e$
$i(x)*x \to e$
$(x*y)*z \to x*(y*z)$
$x*(i(x)*y) \to y$
$i(x)*(x*y) \to y$
$i(e) \to e$
$i(i(x)) \to x$
$i(x*y) \to i(y)*i(x)$
$x/y \to x*i(y)$

Fig. 1. The Knuth-Bendix Convergent System for Group Theory with Right Division.

$x/e \to x$
$e/x \to i(x)$
$x/x \to e$
$x/(y/z) \to (x/i(y))/z$
$(x/y)/i(y) \to x$
$(x/i(y))/y \to x$
$i(e) \to e$
$i(i(x)) \to x$
$i(x/y) \to y/x$
$x*y \to x/i(y)$

Fig. 2. A New Convergent System for Group Theory Based Upon Right Division.

1.3. Proving uniform termination of term rewriting systems

The uniform termination problem for term rewriting systems is an undecidable problem (Huet and Lankford [6] , Lipton and Snyder [14]), but we want REVE to provide powerful and mechanized tools that can be used in most practical situations, and which do not require that the user know much about how termination algorithms work. In most cases, REVE does the proof automatically without intervention from the user. The method is based on *simplification orderings*. A partial ordering on terms is a simplification ordering if it satisfies the following two properties, for all A, A_1, A_2:

Subterm Property: $A < f(\ldots,A,\ldots)$.

Compatibility Property:
$A_1 < A_2 \Rightarrow f(\ldots,A_1,\ldots) < f(\ldots,A_2,\ldots)$.

Dershowitz (1) proved the following theorem:

A term rewriting system $\Sigma = \{l_i \Rightarrow r_i \mid i \in I\}$ with a finite number of symbols is uniformly terminating if there exists a simplification ordering $<$ such that for all i in I and for all substitutions σ of ground terms for variables, $\sigma(l_i) > \sigma(r_i)$.

In REVE there are actually two uniform termination algorithms. One is based on the *recursive path ordering* (written $\overset{*}{<}$ here) derived from the work of Plaisted (17), Dershowitz (1), and Kamin & Lévy (10), and the other on a *recursive decomposition ordering* (written $\overset{d}{<}$ here) developed by Jouannaud, Lescanne and Reinig (9). Let us now give an informal description of them. Both are extensions of a precedence that is an ordering on the basic symbols. In addition, the recursive path ordering requires knowledge of the status of the operator symbols. The status can be "multiset", "left-to-right" or "right-to-left". Typically with a "left-to-right" symbol we first look at the leftmost direct subterms, so we consider a "right-leaning" term to be less than another with the same symbols and variables. Thus, if $+$ has a left-to-right status, then for the associative equation we have $(x+y)+z \overset{*}{>} x+(y+z)$. The right-to-left status is symmetric: if $+$ has a right-to-left status we have $x+(y+z) \overset{*}{>} (x+y)+z$. With a multiset status we look at all the direct subterms in any order to find those to be compared. For example, with a multiset status for $+$, $(x+y)+z$ and $x+(y+z)$ are not ordered with respect to $\overset{*}{<}$, but we do have $x+(y+z) \overset{*}{>} z+x$. The recursive path ordering compares the terms by first examining their root symbols and then recursively comparing terms and their direct subterms according to a strategy determined by the value of that comparison. The decomposition ordering works in a different manner. It first processes the terms in order to build their *decompositions*. A decomposition records the results of a careful analysis of a term; it determines which symbols, called *leaders*, play significant roles with respect to their positions in the term and to a given precedence. Then it compares the decompositions following a specific strategy.

In general, the recursive decomposition ordering and the recursive path ordering yield similar results when comparing two terms. However, there exist some pairs of terms that can be compared with $\overset{d}{<}$ and not $\overset{*}{<}$ when all symbols have multiset status (as in the original Dershowitz definition); the reverse is true when right-to-left and left-to-right status is permitted. However, the main advantage of the recursive decomposition ordering over the recursive path ordering lies in a property that the authors call *incrementality*. Indeed, when the recursive decomposition method fails to orient terms, it suggests enlargements of the precedence. These enlargements are a set of ordered pairs of symbols, which are extracted from the leaders of both terms. In many cases, REVE decides to add all the suggestions to the precedence; in others, it asks the user to indicate which one it must keep. Thus, the precedence is built up step by step by REVE and in general requires no intervention from the user. The orderings $\overset{d}{<}$ and $\overset{*}{<}$ are *monotonic* with respect to the precedence; i.e., when a new pair is added to the precedence, new pairs of terms may be added to $\overset{d}{<}$ and $\overset{*}{<}$, but none are removed or changed. Because of this monotonicity, the enlarged ordering is consistent with the previous ordering, and so the final decomposition ordering can be built incrementally. In our experiments with abstract data types, user help is only needed before starting the Knuth-Bendix completion algorithm (except for changing the status of an operator). He or she is asked to give:

1. a presentation of the equations in the direction that should yield a uniformy terminating term rewriting system of rule, (the user just follows his or her intuition in doing this), and

2. a declaration of the constructors of the data types REVE can then initialize a precedence by assuming that each constructor is less than each non-constructor.

Intuitively an operation f is less than another operation g with respect to the precedence if f is computationally less complex than g. In this hierarchy the constructors are at the bottom because they are not defined in terms of other operations.

In its current formulation, the recursive decomposition ordering cannot be used to prove the uniform termination when some equations describe the associativity of an operation or a related property. On the other hand because of its incremental property it can construct the precedence which will prove the uniform termination of the rules. In REVE it is possible to combine the recursive decomposition ordering and the recursive path ordering in such a way that the recursive path ordering is used to orient the equations and the recursive decomposition ordering is called for help; i.e., for enlarging the precedence when the recursive path ordering fails. The recursive decomposition ordering then provides the suggestions that the recursive path ordering needs.

1.4. Proof of inductive theorems

Huet and Hullot (7) developed a method to prove inductive theorems without explicitly invoking induction that simplifies the work of Musser (15) and Goguen (3). They used a modified version of the Knuth-Bendix completion algorithm. Their inductionless induction works as follows: To prove an inductive theorem, you add the statement to a given convergent set and try to generate a new convergent set while checking that a few simple form conditions are satisfied. If the algorithm succeeds, your statement is a theorem. If it fails by generating a forbidden equation (like a relation between the constructors) your statement is not a theorem. If it runs forever you can say nothing; perhaps by presenting your theorem in a different way or adding lemmas you could succeed. A justification of this method is based on the assumption that the non-constructors are "well defined". Intuitively, that means that the equations completely define these operations without ambiguities. The easiest way to check this "well definition" is based on the uniform termination associated with confluence and some syntactical properties of the left-hand side of the rules. These ideas are generalized by Lankford [13] to congruence class term rewriting systems whose congruence classes are all finite. He also says, "In our opinion, proving the finite termination property for a set of reductions claimed to be complete may turn out to be the most difficult part of the inductionless induction approach. (It is obviously the part that many term rewriting researchers continue to neglect in their experiments.)". In REVE we have given special attention to the uniform termination problem.

2. OTHER ASPECTS OF REVE AND CONCLUSION

1. REVE was used to prove results in algebra that were never before done by computer; e.g., that the equation

$$x/(((x/x)/y)/z)/(((x/x)/x)/z)) = y$$

determines groups (Higman and Neuman [4]). The proof provided by REVE is conceptually simpler than that of Higman and Neuman, but could not be done by hand.

REVE was also able to solve a problem posed by Knuth and Bendix [11], about Taussky's axioms for groups [19]. Indeed it generated the confluent rewriting system of Figure 1 from these axioms.

2. Other unification algorithms will be implemented, for example, to handle equational theories with commutative or associative and commutative operators, and more generally unification that can be described by rewriting systems and sets of equations as proposed by J.P. Jouannaud, C.Kirchner and H. Kirchner [20].

3. REVE is written in CLU, so it constitutes an interesting application for object-oriented languages based on abstract data types and modularity. Advantages for reliability, readability and maintenance will be described in a forthcoming report.

As we have emphasized the uniform termination is an important and often neglected aspect of decision and proof methods based on rewriting. Among softwares that manipulate rewriting systems, the originality of REVE lies in its ability to easily prove uniform termination. Especially, the incremental increase of the operator precedence derived from properties of the recursive decomposition ordering needs little intervention from the user, and requires little knowledge about how the termination algorithm works. The examples that follow in Appendix I partially illustrate REVE's ease and flexibility in proving and disproving properties of abstract data types.

Acknowledgements: I would like to thank Dallas Lankford and Randy Forgaard who read carefully a preliminary version of this paper and provided useful comments. John Guttag welcomed me at MIT Laboratory for Computer Science and without him this work would never be done. I am grateful to Paul Johnson, Jean-Pierre Jouannaud, Bob Sheifler and Jeannette Wing for their friendly help and also to Nachum Dershowitz, Samuel Kamin, Jean-Jacques Lévy and Fernand Reinig for discussions about their algorithms.

3. REFERENCES

1. Dershowitz N., "Orderings for Term Rewriting Systems," *Proc. 20th Symposium on Foundations of Computer Science*, (1979), 123-131. Also, *Theoretical Computer Science* 17 (1982), 279-301.

2. Evans T., "On Multiplicative Systems Defined by Generators and Relations, I. Normal Form Theorems," *Proc. Cambridge Philos. Soc.* 47 (1951), 637-649.

3. Goguen J.A., "How to Prove Algebraic Inductive Hypotheses Without Induction," 5th Conf. On Automated Deduction, *Lecture Notes in Computer Science*, 87 (1980), 356-373.

4. Higman G., Neumann B.H., "Groups as Groupoids with One Law," *Publ. Math. Debrecen.* 2 (1952), 215-221.

5. Huet G., "A Complete Proof of Correctness of the Knuth-Bendix Completion Algorithm," *J.Comp. Sys. Sc.*, 23 (1981), 11-21.

6. Huet G., Lankford D., "On the Uniform Halting Problem for Term Rewriting Systems," *Rapport Laboria* 283 (mars 1978).

7. Huet G., Hullot J., "Proofs by Induction in Equational Theories with Constructors," *Proc. 21st Symposium on Foundations of Computer Science* (1980).

8. Huet G., Oppen D.C., "Equations and Rewrite Rules: A Survey," in *Formal Languages: Perspectives and Open Problems*, Ed. Book R., Academic Press (1980). Also, Technical Report CSL-11, SRI International (Jan. 1980).

9. Jouannaud J.P., Lescanne P., Reinig F., "Recursive Decomposition Ordering," *Conf. on Formal Description of Programming Concepts*, Garmisch, (1982).

10. Kamin S., Lévy J.J., "Attempts for Generalizing the Recursive Path Ordering," (Feb. 1980).

11. Knuth D.E., Bendix P.B., "Simple Word Problems in Universal Algebras," in *Computational Problems in Abstract Algebra*, Ed. Leech J., Pergamon Press (1969), 263-297.

12. Lankford D., "Research in Applied Equational Logic," Louisiana Tech. Univ., Math. Dept., report MTP-15, (Dec. 1980).

13. Lankford D., "A Simple Explanation of Inductionless Induction," Louisiana Tech. Univ., Math. Dept., report MTP-14, (August 1981).

14. Lipton R., Snyder L., "On the Halting Problem for Term Replacement Systems," *Proc. Conf. on Theoretical Comp. Sci.*, Univ. of Waterloo,

(july 1977), 43-46.

15. Musser D.L., "On Proving Inductive Properties of Abstract Data Types," *Proc. 7th ACM Symposium on Principles of Programming Languages* (1980), 154-162.

16. Novikov P., "The Algorithm Unsolvability of the World Problem for Group Theory," *Tr. Mat. Inst. Steklov.* 44 (*AMS Translations Series 2* 9 (1955) 1-124).

17. Plaisted D., "A Recursively Defined Ordering for Proving Termination of Term Rewriting Systems," Dept. of Computer Science, Report 78-943, Univ. of Illinois at Urbana-Champaign, (sept. 1978).

18. Post E., "Recursive Unsolvability of a Problem of Thue," *J. Symb. Logic* 12 (1947), 1-11.

19. Taussky O., "Zur Axiomatik der Gruppen", *Ergebnisse eines Math. Kolloquiums Wien* 4 (1963), 2-3.

20. Jouannaud J.P., Kirchner C., Kirchner H., "Incremental Unification in Equational Theories," *Proc. 20th Allerton Conf. on Communication, Control and Computing*, (oct. 1982).

APPENDIX I - TWO EXAMPLES

Italic comments are added by the author. Underlined parts are commands entered by the user.

EXAMPLE 1

reve
Hello! My name is REVE (dream in french). I am a rewriting system laboratory.

What do you want to do now? (Type ? for help)
read
Which one?
set
A specification of sets of natural numbers with "has" and < (subset).

Your system is now:

1 if(tt,x,y) == x
2 if(ff,x,y) == y
3 if(x,y,y) == y *3 and 4 are inductive theorems*
4 if(x,tt,ff) == x *of booleans.*
5 (o=o) == tt *"o" is the operator correspon-*
6 (o=s(x)) == ff *ding to zero.*
7 (s(x)=o) == ff *"s" is the operator successor.*
8 (s(x)=s(y))==(x=y) *Equations 5,6,7 and 8 define*
9 has(empty_set,x)==ff *an operator = used in the Equation 10.*
10 has((u+x),y)==if((x=y),tt,has(u,y)) *+ is insert. has(u,x) means "u has x" or "x belongs to u".*
11 (empty_set<v)==tt *< is for subset.*
12 ((u+x)<v)==if(has(v,x),(u<v),ff)

REVE initializes the precedence and declares the constructors. REVE will use this in proofs of induction theorems.
The constructors are ff, tt, o, s empty_set, +.

What do you want to do now? (Type ? for help)
cons
Give me a list of symbols
ff tt o s empty_set +

What do you want to do now? (Type ? for help)
kbi *Knuth-Bendix with incremental decomposition ordering is used.*

In a first part REVE orients the equations into rules.

New rule added
if(tt,x,y) → x

New rule added
if(ff,x,y) → y

New rule added
if(x,y,y) → y

New rule added
if(x,tt,ff) → x

New rule added
(o=o) → tt

New rule added
(o=s(x)) → ff

New rule added
(s(x)=o) → ff

New rule added
(s(x)=s(y)) → (x=y)

New rule added
has(empty_set,x) → ff

New rule added
(empty_set < v) → tt

REVE adds two new pairs.

I suggest that the following ordered pairs be added:
if < has *REVE's suggestions.*
= < has
to validate the rule
has((u+x),y) → if((x=y),tt,has(u,y))

New rule added
has((u+x),y) → if((x=y),tt,has(u,y))

REVE adds again two new pairs.

I suggest that the following ordered pairs be added:
if < <
has < <
to validate the rule
((u+x) < v) → if(has(v,x),(u < v),ff)

New rule added
((u+x) < v) → if(has(v,x),(u < v),ff)

In a second part REVE looks for critical pairs and it does not find any.

I am computing the critical pairs with
(o=o) → tt

I am computing the critical pairs with
has(empty_set,x) → ff

I am computing the critical pairs with
(empty_set < v) → tt

I am computing the critical pairs with
if(tt,x,y) → x

I am computing the critical pairs with
if(ff,x,y) → y

I am computing the critical pairs with
if(x,y,y) → y

I am computing the critical pairs with
if(x,tt,ff) → x

I am computing the critical pairs with
(o=s(x)) → ff

I am computing the critical pairs with
(s(x)=o) → ff

I am computing the critical pairs with
(s(x)=s(y)) → (x=y)

I am computing the critical pairs with
has((u+x),y) → if((x=y),tt,has(u,y))

I am computing the critical pairs with
((u+x) < v) → if(has(v,x),(u < v),ff)

The system is "consistent" and the Knuth-Bendix algorithm has transformed the previous set of equations into a confluent and noetherian (convergent) set of rules.

Your convergent system is now :

1 if(tt,x,y) → x
2 if(ff,x,y) → y
3 if(x,y,y) → y
4 if(x,tt,ff) → x
5 (o=o) → tt
6 (o=s(x)) → ff
7 (s(x)=o) → ff
8 (s(x)=s(y)) → (x=y)
9 has(empty_set,x) → ff
10 (empty_set < v) → tt
11 has((u+x),y) → if((x=y),tt,has(u,y))
12 ((u+x) < v) → if(has(v,x),(u < v),ff)

PROOF OF A THEOREM

What do you want to do now? (Type ? for help)
<u>a</u> *The user adds the identity to be proved. cf. § 1.3*
Type your new rules or equations. It or they must be terminated by a "."
<u>has((u+x),x) == tt.</u>

Your system is now:

1 if(tt,x,y) → x
2 if(ff,x,y) → y
3 if(x,y,y) → y
4 if(x,tt,ff) → x
5 (o=o) → tt
6 (o=s(x)) → ff
7 (s(x)=o) → ff
8 (s(x)=s(y)) → (x=y)
9 has(empty_set,x) → ff
10 (empty_set < v) → tt
11 has((u+x),y) → if((x=y),tt,has(u,y))
12 ((u+x) < v) → if(has(v,x),(u<v),ff)
13 has((u+x),x) == tt

What do you want to do now? (Type ? for help)
<u>kbi</u> *The left-hand side of the given equation is reduced and the new created equation is transformed into a rule.*

New rule added
if((x=x),tt,has(u,x)) → tt

I am computing the critical pairs with
if((x=x),tt,has(u,x)) → tt

New rule added
if((x=x),tt,has(y,s(x))) → tt

New rule added
(x=x) → tt *is an inductive theorem of the natural numbers.*

The left hand-side of the following rule is now reducible
(o=o) → tt
The left hand-side of the following rule is now reducible
if((x=x),tt,has(u,x)) → tt
The left hand-side of the following rule is now reducible
if((x=x),tt,has(y,s(x))) → tt

I am computing the critical pairs with
(x=x) → tt

Your convergent system is now:

1 if(tt,x,y) → x
2 if(ff,x,y) → y
3 if(x,y,y) → y
4 if(x,tt,ff) → x
5 (o=s(x)) → ff
6 (s(x)=o) → ff
7 (s(x)=s(y)) → (x=y)
8 has(empty_set,x) → ff
9 (empty_set < v) → tt
10 has((u+x),y) → if((x=y),tt,has(u,y))
11 ((u+x) < v) → if(has(v,x),(u < v),ff)
12 (x=x) → tt

PROVING THAT A STATEMENT IS NOT A THEOREM

What do you want to do now? (Type ? for help)
<u>a</u> *The user adds the identity to be proved (or disproved!).*
Type your new rules or equations. It or they must be terminated by a "."
<u>((u+x) < (v+x)) == (u < v).</u>

Your system is now:

1 if(tt,x,y) → x
2 if(ff,x,y) → y
3 if(x,y,y) → y
4 if(x,tt,ff) → x
5 (o=s(x)) → ff
6 (s(x)=o) → ff
7 (s(x)=s(y)) → (x=y)
8 has(empty_set,x) → ff
9 (empty_set < v) → tt
10 has((u+x),y) → if((x=y),tt,has(u,y))
11 ((u+x) < v) → if(has(v,x),(u < v),ff)
12 (x=x) → tt
13 ((u+x) < (v+x)) == (u < v)

What do you want to do now? (Type ? for help)
<u>kbi</u>

New rule added
(u < (v+x)) → (u < v) *The left-hand side of equation 13 is reduced by 11, 10, 12, 1 and is transformed into a rule.*

I am computing the critical pairs with
(u < (v+x)) → (u < v)

```
New rule added
if(if((y=z),tt,has(x,z)),(u < x),ff) -> if(has(x,z),
      (u < x),ff)

      I am computing the critical pairs with
      if(if((y=z),tt,has(x,z)),(u < x),ff) ->
             if(has(x,z),(u < x),ff)

New rule added
if(if((x=y),tt,has(z,s(y))),(u < z),ff) ->
      if(has(z,s(y)),(u < z),ff)

New rule added
if((x=y),(z < empty_set),ff) -> ff

New rule added
if((x=y),tt,has(z,y)) -> has(z,y)
      The left hand-side of the following rule is
      now reducible
      if(if((y=z),tt,has(x,z)),(u < x),ff) ->
             if(has(x,z),(u < x),ff)

New rule added
if(has(x,y),(z < x),ff) -> (z < x)

      I am computing the critical pairs with
      if((x=y),(z < empty_set),ff) -> ff

New rule added
(x=y) -> ff
      The left hand-side of the following rule is
      now reducible
      (o=s(x)) -> ff
      The left hand-side of the following rule is
      now reducible
      (s(x)=o) -> ff
      The left hand-side of the following rule is
      now reducible
      (s(x)=s(y)) -> (x=y)
      The left hand-side of the following rule is
      now reducible
      (x=x) -> tt
      The left hand-side of the following rule is
      now reducible
      if(if((x=y),tt,has(z,s(y))),(u < z),ff) ->
             (u < z)
      The left hand-side of the following rule is
      now reducible
      if((x=y),(z < empty_set),ff) -> ff
      The left hand-side of the following rule is
      now reducible
      if((x=y),tt,has(z,y)) -> has(z,y)
```

The rule (x = y) -> ff is incompatible with the rule (x = x) -> tt.

```
**Your theorem is false, or your specification is
  not consistent**

I deduced the following equation
ff == tt
```

A PROOF RUNNING FOREVER

```
What do you want to do now? (Type ? for help)
a
Type your new rules or equations. It or they must
be terminated by a "."
(u < (u+x)) == tt.

Your system is now:

1  if(tt,x,y) -> x
2  if(ff,x,y) -> y
3  if(x,y,y) -> y
4  if(x,tt,ff) -> x
5  (o=s(x)) -> ff
6  (s(x)=o) -> ff
7  (s(x)=s(y)) -> (x=y)
8  has(empty_set,x) -> ff
9  (empty_set < v) -> tt
10 has((u+x),y) -> if((x=y),tt,has(u,y))
11 ((u+x) < v) -> if(has(v,x),(u < v),ff)
12 (x=x) -> tt
13 (u < (u+x)) == tt
What do you want to do now? (Type ? for help)
kbi
New rule added
(u < (u+x)) -> tt

      I am computing the critical pairs with
      (u < (u+x)) -> tt
New rule added
(x < ((x+y)+z)) -> tt

      I am computing the critical pairs with
      (x < ((x+y)+z)) -> tt
New rule added
(x < (((x+y)+z)+u)) -> tt

      I am computing the critical pairs with
      (x < (((x+y)+z)+u)) -> tt
New rule added
(x < (((x+y)+z)+u)+v)) -> tt

      I am computing the critical pairs with
      (x < ((((x+y)+z)+u)+v)) -> tt

                 ⋮
```

EXAMPLE 2

```
reve
Hello! My name is REVE (dream in french). I am a
rewriting system laboratory.

What do you want to do now? (Type ? for help)
read
Which one?
fib
```

A specification of the natural numbers and two definitions of the Fibonacci function.

```
Your system is now:

1  (o+x) == x
2  (s(x)+y) == s((x+y))
3  ((x+y)+z) == (x+(y+z))
4  fib(o) == o
5  fib(s(o)) == s(o)
6  fib(s(s(x))) == (fib(x)+fib(s(x)))
7  dfib(o,y) == y
8  dfib(s(o),y) == s(y)
9  dfib(s(s(x)),y) == dfib(s(x),dfib(x,y))
```

"o" is the operator corresponding to zero. (line 1)

associativity is an inductive theorem. (line 3)

the classical definition of the Fibonacci function. (line 6)

another definition of the Fibonacci function without reference to addition. (line 9)

The constructors of the natural numbers are o and s.

What do you want to do now? (Type ? for help)
cons
Give me a list of symbols
o s

What do you want to do now? (Type ? for help)
kbri *Knuth-Bendix with recursive path ordering helped by the recursive decomposition ordering for adding pairs of symbols to the precedence.*

New rule added
(o+x) → x

New rule added
(s(x)+y) → s((x+y))

New rule added
fib(o) → o

New rule added
fib(s(o)) → s(o)

New rule added
dfib(o,y) → y

New rule added
dfib(s(o),y) → s(y)

dfib is declared as having a "left-to-right" status. Notice that with a "multiset" status, the equation would be oriented in the opposite direction. REVE takes the presentation into account.

Would you like to set the status of an operator to validate the equation? (Y/N)
dfib(s(s(x)),y) == dfib(s(x),dfib(x,y))
y
Give me a symbol or ask for help(?)
dfib
Give the status RL or LR
lr
OK you don't change the ordering for the other rules.

New rule added
dfib(s(s(x)),y) → dfib(s(x),dfib(x,y))

I cannot suggest any ordered pair to validate the equation
((x+y)+z) == (x+(y+z))

+ is declared as having a left-to-right status.

Would you like to set the status of an operator? (Y/N)
y
Give me a symbol or ask for help(?)
+
Give the status RL or LR
lr
OK you don't change the ordering for the other rules.

New rule added
((x+y)+z) → (x+(y+z))

REVE adds the pair + < fib.

I suggest that the following ordered pair be added:
+ < fib
to validate the rule
fib(s(s(x))) → (fib(x)+fib(s(x)))

New rule added
fib(s(s(x))) → (fib(x)+fib(s(x)))

I am computing the critical pairs with
fib(o) → o

I am computing the critical pairs with
(o+x) → x

I am computing the critical pairs with
fib(s(o)) → s(o)

I am computing the critical pairs with
dfib(o,y) → y

I am computing the critical pairs with
(s(x)+y) → s((x+y))

I am computing the critical pairs with
dfib(s(o),y) → s(y)

I am computing the critical pairs with
fib(s(s(x))) → (fib(x)+fib(s(x)))

I am computing the critical pairs with
dfib(s(s(x)),y) → dfib(s(x),dfib(x,y))

I am computing the critical pairs with
((x+y)+z) → (x+(y+z))

Your convergent system is now:

1 (o+x) → x
2 (s(x)+y) → s((x+y))
3 fib(o) → o
4 fib(s(o)) → s(o)
5 dfib(o,y) → y
6 dfib(s(o),y) → s(y)
7 dfib(s(s(x)),y) → dfib(s(x),dfib(x,y))
8 ((x+y)+z) → (x+(y+z))
9 fib(s(s(x))) → (fib(x)+fib(s(x)))

PROOF OF DFIB(X,Y) == (FIB(X)+Y)

What do you want to do now? (Type ? for help)
a
Type your new rules or equations. It or they must be terminated by a "."
dfib(x,y) == (fib(x)+y).

the user adds the equation to be proved.

Your system is now:

1 (o+x) → x
2 (s(x)+y) → s((x+y))
3 fib(o) → o
4 fib(s(o)) → s(o)
5 dfib(o,y) → y
6 dfib(s(o),y) → s(y)
7 dfib(s(s(x)),y) → dfib(s(x),dfib(x,y))
8 ((x+y)+z) → (x+(y+z))
9 fib(s(s(x))) → (fib(x)+fib(s(x)))
10 dfib(x,y) == (fib(x)+y)

What do you want to do now? (Type ? for help)
kbri *REVE proposes the pair fib < dfib.*

I suggest that the following ordered pair be added:
fib < dfib
to validate the rule
dfib(x,y) → (fib(x)+y)

New rule added
dfib(x,y) → (fib(x)+y)
The left hand-side of the following rule is now reducible
dfib(o,y) → y

The left hand-side of the following rule is now reducible
dfib(s(o),y) → s(y)
The left hand-side of the following rule is now reducible
dfib(s(s(x)),y) → dfib(s(x),dfib(x,y))

New rule added
(fib(s(x))+(fib(x)+y)) → (fib(x)+(fib(s(x))+y))

I am computing the critical pairs with
dfib(x,y) → (fib(x)+y)

I am computing the critical pairs with
(fib(s(x))+(fib(x)+y)) → (fib(x)+fib(s(x))+y))

The theorem is proved.

Your convergent system is now:

1 (o+x) → x
2 (s(x)+y) → s((x+y))
3 fib(o) → o
4 fib(s(o)) → s(o)
5 ((x+y)+z) → (x+(y+z))
6 fib(s(s(x))) → (fib(x)+fib(s(x)))
7 dfib(x,y) → (fib(x)+y)
8 (fib(s(x))+(fib(x)+y)) → (fib(x)+(fib(s(x))+y))

8 is an inductive "lemma" added by REVE which has a flavor of commutativity.

PROOF OF THE SAME THEOREM WITH A DIFFERENT PRESENTATION

What do you want to do now? (Type ? for help)
a
Type your new rules or equations. It or they must be terminated by a "."
(fib(x)+y) == dfib(x,y).

the user adds the same equation but he permutes left hand-side and right-hand side.

Your system is now:

1 (o+x) → x
2 (s(x)+y) → s((x+y))
3 fib(o) → o
4 fib(s(o)) → s(o)
5 dfib(o,y) → y
6 dfib(s(o),y) → s(y)
7 dfib(s(s(x)),y) → dfib(s(x),dfib(x,y))
8 ((x+y)+z) → (x+(y+z))
9 fib(s(s(x))) → (fib(x)+fib(s(x)))
10 (fib(x)+y) == dfib(x,y)

What do you want to do now? (Type ? for help)
kbri

REVE proposes the pair dfib < fib.

I suggest that the following ordered pair be added:
dfib < fib
to validate the rule
(fib(x)+y) → dfib(x,y)

It is not enough; REVE proposes dfib < +.

I suggest that the following ordered pair be added:
dfib < +
to validate the rule
(fib(x)+y) → dfib(x,y)

New rule added
(fib(x)+y) → dfib(x,y)

I am computing the critical pairs with
(fib(x)+y) → dfib(x,y)

New rule added
(dfib(x,y)+z) → dfib(x,(y+z))

New rule added
dfib(s(x),dfib(x,y)) → dfib(x,dfib(s(x),y))

I am computing the critical pairs with
(dfib(x,y)+z) → dfib(x,(y+z))

I am computing the critical pairs with
dfib(s(x),dfib(x,y)) → dfib(x,dfib(s(x),y))

The theorem is proved. Compare with the previous proof.

Your convergent system is now:

1 (o+x) → x
2 (s(x)+y) → s((x+y))
3 fib(o) → o
4 fib(s(o)) → s(o)
5 dfib(o,y) → y
6 dfib(s(o),y) → s(y)
7 dfib(s(s(x)),y) → dfib(x,dfib(s(x),y))
8 ((x+y)+z) → (x+(y+z))
9 fib(s(s(x))) → dfib(x,fib(s(x)))
10 (fib(x)+y) → dfib(x,y)
11 (dfib(x,y)+z) → dfib(x,(y+z))
12 dfib(s(x),dfib(x,y)) → dfib(x,dfib(s(x),y))

What do you want to do now? (Type ? for help)
q

APPENDIX II

Proof of the termination of the examples Fig.1 and Fig.2.

The example of Fig.1 is proved using the recursive path ordering with "left-to-right" status for *, and with the precedence e < * < i < /.

The example of Fig.2 is proved using the recursive path ordering with "right-to-left" status for /, and with the precedence e < i ∿ / < *.

APPENDIX III

Sketch of the definition of the recursive decomposition ordering [9].

A decomposition of a term is a multiset (for all paths in the term) of sets (for all occurrences on the paths) of *elementary decompositions*. An elementary decomposition is a quadruplet

<*leader*: symbol, *main-follower*: set (elementary-decomposition), *other-followers*: multiset(term), *context*: set(elementary-decomposition)>

The elementary decomposition of t (Fig.3) along the path 21 at the occurrence 2 is <*,{<y,{},{},{}>},{z},d> where d is the set of elementary-decompositions of t' along the path 2.

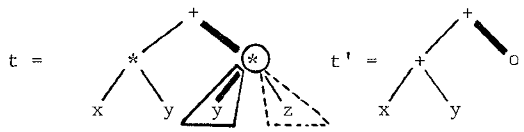

Fig. 3. A term and a context

The ordering on two terms is the ordering on their decompositions, and the ordering on decompositions is obtained by set and multiset extension of the ordering on elementary decompositions. The ordering on elementary decomposition is given lexicographically, recursively using orderings on set (elementary-decomposition) and multiset (terms).

NB1: The decomposition of o is the empty multiset.

NB2: Because set(set(elementary-decomposition)) contains enough information about the terms, we do not need multiset(set(elementary-decomposition)) in the definition. So only set extensions of the ordering are necessary.