A SELF-MODIFYING THEOREM PROVER

Cynthia A. Brown

GTE LABORATORIES INCORPORATED
40 Sylvan Road
Waltham, Massachusetts 02254

Abstract:

Theorem provers can be viewed as containing declarative knowledge (in the form of axioms and lemmas) and procedural knowledge (in the form of an algorithm for proving theorems). Sometimes, as in the case of commutative laws in a Knuth-Bendix prover, it is appropriate or necessary to transfer knowledge from one category to the other. We describe a theorem proving system that independently recognizes opportunities for such transfers and performs them dynamically.

Theorem proving algorithms

Theorem provers can be divided into two general classes: those that operate without human intervention to prove straightforward consequences of a set of axioms, and those that serve as a mathematician's assistant in the search for a proof of a mathematically significant theorem. The first type of prover is needed for program verification and artificial intelligence applications, where the necessity of human intervention would severely disrupt the intended application. The second type of theorem prover ususally contains one or more of the first type. It is the first type of prover that we are concerned with.

Our model of a theorem prover is thus an algorithm that establishes the truth of a statement by showing that it is a logical consequence of a given set of axioms. In the process of establishing that truth, the theorem prover may obtain intermediate results that play the role of lemmas. The power and efficiency of the theorem prover depend on the algorithm that is employed (and there is often a trade-off between these two characteristics of the system).

Two major classes of theorem proving algorithms are the resolution-based methods [ROB65,OVE75,BOY71] and Knuth-Bendix type methods [KNU70,HUE80,HUE80b,JEA80,MUS80,PET81, STI81]. There are several ambitious theorem provers of the second type that incorporate one or the other of these methods; for example, the Affirm system [THO79] includes a Knuth-Bendix prover, and the ITP theorem prover [MCC76,OVE75,LUS84] uses hyperresolution, an efficient form of resolution, along with other techniques. Both the Knuth-Bendix algorithm and resolution methods can also be used as the basis of an algorithmic theorem prover. The Knuth-Bendix method is usually more efficient in the cases where it applies, but resolution is much more general.

Declarative versus procedural knowledge in theorem provers

Both of these basic approaches to theorem proving can viewed as, in a very general sense, expert systems: they operate on a data base of explicit knowledge (the axioms) using an inference engine (the theorem-proving algorithm) to solve a problem. This analogy cannot be pushed too far, but it does lead to some interesting questions about the structure of theorem-proving systems. In particular, the debate about the role of declarative versus procedural knowledge has a very real application in the theorem-proving field.

Certain information in a theorem proving system can be represented explicitly (as axioms) or incorporated into the inference mechanism. There are at least two important examples of this phenomenon. One is the use of paramodulation [GLO80,OVE75,WOS70,WOS80] in resolution-type theorem provers. The basic resolution algorithm makes no special provision for the use of the equality predicate in the statement of axioms. To introduce equality it is necessary to provide axioms that specify its properties. Chang and Lee [CHA73] give a set of ten such axioms, which establish that equality is reflexive, symmetric, and transitive and allow equals to be substituted for equals in any expression. To use such axioms in the proof of a theorem would obviously lead to an extremely inefficient proof. The alternative is to build into the theorem-proving algorithm the knowledge required to handle equality appropriately. The inference rule that is used for this purpose is called *paramodulation*. Adding paramodulation to a resolution theorem prover results in an ability to prove theorems about systems that contain equality in a natural, efficient way.

Another example of the same phenomenon can be found in Knuth-Bendix theorem proving systems. These systems operate by deriving all the interesting theorems implied by a set of axioms. The axioms and theorems are expressed as rewrite rules. When the algorithm is successful, the result is an extremely efficient method of proving any additional theorems in that system.

Unfortunately, the algorithm can fail. The rewriting process on which it depends requires a partial order on terms of the system such that each rewrite results in a term that is smaller under the partial order than the one from which it was derived. It is therefore, for example, impossible to include axioms that express the commutativity of an operator, since there is no way to orient a commutative law consistent with a partial order. The solution is to deal with the commutative property of operators on the level of the rewriting algorithm, rather than treating it explicitly as a rewriting rule. The fact that an operator is commutative can be recorded; then, whenever the algorithm checks whether an expression involving that operator can be applied, it tries both ways of ordering the operands.

For operators that are both commutative and associative, a more elaborate scheme is required. In this case a commutative and associative unification algorithm, which checks all possible ways of matching two expressions that involve nested applications of a commutative and associative operator, is used. Such algorithms are surprisingly complex; the problem is equivalent to finding certain partitions of integers [STI81] Nevertheless, its use allows the application of Knuth-Bendix methods to a much broader range of systems than would otherwise be possible.

To handle commutativity with a special unification algorithm is a necessity in Knuth-Bendix systems; to deal with equality via paramodulation or a similar approach is a practical necessity in resolution systems. There are also cases of operators or properties that can be handled either declaratively or procedurally. An example is the idempotent property of an associative and commutative operator; there is an associative-commutative-idempotent unification algorithm, and idempotency can also be dealt with by explicit rewriting. Some investigators are attempting to develop special unification algorithms to deal with other common properties of operators in the framework of Knuth-Bendix systems [JOU83,JOU82,HUL80].

Dynamic recognition of commutative and associative laws

A Knuth-Bendix theorem prover starts with a set of axioms and derives rewrite rules from them. If the prover is able to deal with commutative operators, then it will notice whether the main operator in an expression is commutative or not and apply the appropriate unification algorithm. Ordinarily the prover is told which operators are commutative at the start of its run. However, it is possible for the stated properties of an operator to imply that it is commutative without the commutative axiom being given explicitly. For example, here are the group axioms, written in additive notation:

1. 0+x = x
2. -x+x = 0
3. (x+y)+z = x+(y+z).

These axioms can be completed by the Knuth-Bendix procedure to obtain a set of ten rewrite rules sufficient to decide whether any two expressions written in this system are equivalent. If we start with an additional axiom,

4. x+x = 0,

the system will derive the fact that + is a commutative operation. At that point it would be necessary to restart the process, this time with the initial stipulation that + is commutative.

The necessity of restarting the system is annoying and is contrary to the goal of avoiding the necessity of human intervention in the theorem proving process. (This goal is extremely important for practical program verification and artificial intelligence systems.) Fortunately, a commutative rule is easy to recognize. It is straightforward to have the prover examine any unorderable rewrite rule it has obtained (either as an initial axiom or during the proving process) to see if it is a commutative law. If a commutative law is discovered, that fact can be recorded and all future applications of that operator can be done using the commutative unification principle. If the operator is also associative, the more powerful associative-commutative unification can be used. It is also helpful to check the already derived rewrite rules to see if any can be simplified using the fact that the operator is commutative.

One important unanswered question is the extent to which previous work must be redone when an operator is discovered to be commutative. All previously discovered rules must still be valid, but it is possible that some further rules may be discovered by reconsidering the earlier results. Until this can be resolved theoretically, the system plays it safe by regenerating all relevant potential rules using the fact that the operator is commutative, and checks that they reduce to a common form.

Implementation

A Knuth-Bendix theorem prover embodying these ideas has been implemented in Prolog. At present it is able to recognize a commutative law and use the commutative unification algorithm on operators for which such laws have been discovered. It also recognizes explicit associative laws, and will use commutative-associative unification for operators that have both properties. If an operator that is known to be associative is found to also be commutative, the associative law is removed from the list of explicit rewrite rules belonging to the system. (No practical unification algorithm for the associative property alone is known.)

As an example, consider the group theory axioms given above. The system completes the original three axioms by deriving rewrite rules in the following order:

```
 4. -x+(x+y) = y
 5. -0+x = x
 6. --x+0 = x
 7. --0+x = x
 8. -0 = 0
 9. --x+y = x+y
10. x+0 = x
11. x+(-x) = 0
12. --x = x
13. x+(-x+y) = y
14. x+(y-(x+y)) = 0
15. -(x+y)+(x+(y+z)) = z
16. x+(y+(-(x+y)+z)) = z
17. x+(y+(z-(x+(y+z)))) = 0
18. x-(y+x) = -y
19. x+(y-(z+(x+y))) = -z
20. x+(-(y+x)+z) = -y+z
21. -(x+y) = -y+(-x)
```

Rules 5,6,7,9,14,15,16,17,18,19, and 20 are eliminated during the process by being simplified using later rules; the final set has the remaining ten elements.

It is interesting to contrast this derivation with the one obtained by starting with the three group theory axioms plus the fourth axiom given above, stating that the group operation is idempotent. The following rules are discovered in order:

```
 5. -x+(x+y) = y
 6. x+(y+(x+y)) = 0
 7. x+(x+y) = y
 8. -0+x = x
 9. -0 = 0
10. --x+0 = x
11. -x+0 = x
12. -x = x
13. x+0 = x
14. x+(y+(x+(y+z))) = z
15. x+(y+(z+(x+(y+z)))) = 0
16. x+(y+x) = y
17. x+(y+(z+(x+y))) = z
18. x+(y+(x+z)) = y+z
19. x+y = y+x
```

At this point, the system recognizes that + is commutative and associative. Rule 3 is removed from the rule database, and the shift to associative-commutative unification for applications of the + operator is made. The discovery of rule 19, like the discovery of rule 21 in the previous example, causes many of the earlier rules to disappear. For example, it is no longer necessary to have rules showing that 0 is both a left and a right identity.

This example demonstrates the ability of the theorem prover to modify itself in response to its discoveries. The extra checking involved consumes a negligible amount of time and greatly expands the capacities of the prover to handle sets of axioms without human intervention.

Future work

Many further improvements are possible on this system. As unification algorithms for other properties are discovered, the ability to recognize those properties can be built into the system, provided they can be easily recognized syntactically. Unfortunately, it is necessary to devise a new unification algorithm for each desired *combination* of properties, and this is no easy matter. Associativity and commutativity are probably the most generally useful properties, and a system that simply recognizes those (and thereby provides a good method for handling commutativity) should be most useful. A review of the currently available unification algorithms and their properties is given in [SIE84].

Also, while the commutative property is easy to recognize syntactically, there may be axioms present that imply the associative property and that lead to unorderable rules when the commutative law is applied to them. It would be helpful if the system could attempt to prove the associative law whenever it got into trouble with an unorderable rule for an operator known to be commutative and not known to be associative. In this way the order of discovery of properties would not be such a crucial factor in the success of the algorithm. (In general, it is a good strategy to postpone processing of unorderable rules in the hope that future rewrite rules can be used to eliminate them.)

Another area for research is the efficiency of the associative and commutative unification algorithm. The usual algorithm often generates many redundant possibilities; it should be possible to develop methods to recognize and avoid them. Finally, the Knuth-Bendix algorithm itself should be studied to identify potential speed-ups. Progress in this direction has already been made by Huet [HUE81].

References

[BOY71] BOYER, R. S., "Locking: a restriction of resolution", Ph.D. Thesis, University of Texas at Austin, 1971.

[BOY79] Boyer, R. S., and J. S. Moore, *A Computational Logic*, Academic Press, New York, 1979.

[CHA73] Chang, C. L. and R. C. T. Lee, *Symbolic Logic and Mechanical Theorem Proving*, Academic Press, New York, 1973.

[FAG84] Fages, F., "Associative-Commutative Unification", INRIA, Domaine de Volceau Rocquencourt, 78153 Le Chesnay, France, 1984.

[GLO80] Gloess, P. Y., and J. P. H. Laurent, "Adding Dynamic Paramodulation to Rewrite Algorithms", in *5th Conference on Automated Deduction*, W. Bibel and R. Kowalski, eds., Springer Verlag LNCS 87, Berlin 1980, pp.195-207.

[GOG78] Goguen, J. A., J. W. Thatcher, and E. G. Wagner, "An Initial Algebra Approach to the Specification, Correctness, and Implementation of Abstract Data Types", in *Current Trends in Programming Methodology*, v. 4, R. Yeh, ed., Prentice-Hall (1978), pp. 80-149.

[GOG80] Goguen, J. A., "How to prove algebraic inductive hypotheses without induction, with applications to the correctness of data type implementation", in *5th Conference on Automated Deduction*, W. Bibel and R. Kowalski, eds., Springer Verlag LNCS 87, Berlin 1980, pp.356-373.

[GUT78] Guttag, J., E. Horowitz, and D. Musser, "Abstract Data Types and Software Validation", *CACM* v. 21, no. 12, 1978, pp.1048-1063.

[HER30] Herbrand, J., "Investigations in proof theory: the properties of propositions", in *From Frege to Goedel: A Source Book in Mathematical Logic*, J. van Heijenoort, ed., Harvard University Press, Cambridge, Mass.

[HUE80] Huet, G., and J. M. Hullot, "Proofs by Induction in Equational Theories with Constructors", Twenty-first Annual Symposium on Foundations of Computer Science, IEEE Computer Society, 1980, pp. 96-107.

[HUE80b] Huet, G., and D. Oppen, "Equations and Rewrite Rules - A Survey", in *Formal Language Theory*, R. Book, ed., Academic Press, N.Y., 1980, pp.349 - 405.

[HUE81] Huet, G., "A Complete Proof of the Correctness of the Knuth and Bendix Completion Algorithm," JCSS 23, 1981, pp. 11-21.

[HUL80] Hullot, J. M., "Canonical Forms and Unification", *5th Conference on Automated Deduction*, W. Bibel and R. Kowalski, eds., Springer Verlag LNCS 87, Berlin 1980, pp.396-405.

[JEA80] Jeanrond, H. J., "Deciding Unique Termination of Permutative Rewriting Systems: Choose Your Term Algebra Carefully", in *5th Conference on Automated Deduction*, W. Bibel and R. Kowalski, eds., Springer Verlag LNCS 87, Berlin 1980, pp.335-355.

[JOU83] Jouannaud, J. P., "Confluent and Coherent Equational Term Rewriting Systems: Application to Proofs in Abstract Data Types," Technical Report 83-R-005, Centre de Recherche en Informatique de Nancy, 1983.

[JOU82] Jouannaud, J. P., C. Kirchner, and H. Kirchner, "Incremental Unification in Equational Theories", Proceedings of the Twentieth Annual Allerton Conference, 1982, pp. 396-405.

[KNU70] Knuth, D., and P. B. Bendix, "Simple Word Problems in Universal Algebras", in COMPUTATIONAL PROBLEMS IN ABSTRACT ALGEBRA, J. LEECH, PERGAMON PRESS 1970, PP. 263-279.

[LUS84] Lusk, E.L., and Overbeek, R. A., "A Portable Environment for Research in Automated Reasoning", in *7th Conference on Automated Deduction*, R. Shostak, ed., Springer Verlag LNCS 170, Berlin 1984, pp.1-42.

[MCC76] McCharen, J. D., R. A. Overbeek, and L. Wos, "Problems and experiments for and with automated theorem-proving programs", *IEEE Trans. Comput.*, v. 25, No. 8, 1976, pp. 773-782.

[MUS80] Musser, D. L., "On proving inductive properties of abstract data types", *Proc. Seventh Annual ACM Symp. on POPL*, 1980, pp. 154-162.

[OVE75] Overbeek, R. A., "An implementation of hyperresolution", *Comput. Math. Appl.* 1, 1975, pp.201-214.

[PET81] Peterson, G. E., and M. E. Stickel, "Complete sets of reductions for some equational theories", *JACM* v. 28, no. 2, 1981, pp. 233-264.

[ROB65] Robinson, J. A., "A machine-oriented logic based on the resolution principle", JACM 12(1), 1965, pp. 23-41.

[ROB65b] Robinson, J. A., "Automatic deduction with hyper-resolution", *Internat. J. Comput. Math.*, 1, pp. 227-234.

[SIE84] Siekmann, J. H., "Universal Unification", in *7th Conference on Automated Deduction*, R. Shostak, ed., Springer Verlag LNCS 170, Berlin 1984, pp.1-42.

[STI81] Stickel, M. E., "A unification algorithm for associative-commutative functions", *JACM*, v. 28, no.3, 1981, pp.423-434.

[THO79] Thompson, D. H., ed., *AFFIRM Reference Manual*, USC Information Sciences Institute, 1979.

[WOS70] Wos, L., and G. A. Robinson, "Paramodulation and Set of Support", *Proc. Symp. Automatic Demonstration, Versailles, France, 1968*, Springer-Verlag, New York (1970), pp.276-310.

[WOS80] Wos, L., R. Overbeek, and L. Henschen, "Hyperparamodulation: A Refinement of Paramodulation", in *5th Conference on Automated Deduction*, W. Bibel and R. Kowalski, eds., Springer Verlag LNCS 87, Berlin 1980, pp.208-219.