ASSOCIATION FOR AUTOMATED REASONING NEWSLETTER

No. 15 May 1990

From the AAR President, Larry Wos. . .

In this issue of the *AAR Newsletter,* we discuss a new implementation of a theorem-proving strategy, present a new result for a familiar challenge problem, announce a new version of OTTER, and request your assistance in collecting open questions.

Xumin Nie has implemented a positive refinement of model elimination. On numerous tests on non-Horn problems, including the non-obvious problem that has been the focus of several articles in earlier AAR newsletters, Nie has obtained good results.

Art Quaife presents a new solution to a challenge problem posed more than a decade ago. His solution performs more efficiently than other known proofs—but Quaife does offer our readers the challenge that there is still room for improvement.

Bill McCune has developed a new version of OTTER, and provides details for how to obtain this powerful theorem prover—free of charge.

And I have issued a request to AAR members to help compile a set of open questions that researchers can attack with automated reasoning programs. A sample question and details are given at the end of the *Newsletter.*

Conferences

Logic in Computer Science

The fifth annual IEEE Symposium on Logic in Computer Science will take place on June 4-7, 1990, in Philadelphia, Penn. The LICS Symposium aims for a wide coverage of theoretical and practical issues in computer science relating to logic—including algebraic, categorical, and topological approaches.

Of particular interest to our AAR members is a special session on automated deduction, on June 5 from 10:40 to 11:10, chaired by Mark Stickel. Three presentations will be given: *Searching for fixed point combinators,* by W. McCune, *Theorem proving with ordered equations,* by N. Dershowitz, and *Automated reasoning in geometry using algebraic methods,* by S.-Ch. Chou.

The symposium is sponsored by the IEEE Technical Committee on Mathematical Foundations of Computing in cooperation with the Association for Symbolic Logic and the European Association of Theoretical Computer Science. For further information contact Prof. A. Meyer, MIT Lab. for Computer Science, NE43-315, 545 Technology Square, Cambridge, MA 02139.

CADE-10

The 10th International Conference on Automated Deduction will be held in Kaiserslautern, West Germany, on July 23-27, 1990. CADE is the major forum at which research on all aspects of automated deduction can be presented. More than 40 papers will be presented on the following topics: theorem proving, unification, term rewriting, decision procedures, program verification and synthesis, deductive databases, logic programming, and inference systems. Invited talks will be presented by R. Boyer and J Strother Moore (Computational Logic Inc.), Woody Bledsoe (University of Texas), Wolfgang Bibel (Technische Hochschule Darmstadt), and Alan Bundy (University of Edinburgh). Inquiries about CADE can be sent to Mark E. Stickel, AI Center, SRI International, 333 Ravenswood Avenue, Menlo Park, CA 94025 (email stickel@ai.sri.com).

AAAI-90

The American Association for Artificial Intelligence will hold this year's conference in Boston, Massachusetts, on July 29-August 3, 1990. Dr. Craig Fields, Director of DARPA, will present the plenary address, focusing on AI in relation to America's technological future. The schedule also includes tutorials on machine learning, reasoning, expert systems, natural language processing, and systems. For information or registration, write to AAAI, Menlo Park, CA 94025 (phone 415-328-3123).

ICALP 91

The 18th International Colloquium on Automata, Languages, and Programming will be held on July 8-12, 1991, at the Universidad Complutense, Madrid, Spain. The topics to be covered include theory of knowledge bases, foundations of logic and functional programming, term rewriting systems, theory of robotics, and transformation and verification. Authors are invited to submit papers by November 15, 1990, to Prof. Mario Rodriguez Artalego, Departamento de Informatica y Automatica, Facultad de Matematicas, Universidad Complutense, Av. Complutense s/n, 28040 Madrid, Spain.

Model Elimination and Its Positive Refinement

Xumin Nie (SUNY at Albany)

Model Elimination (ME) [1] is a well-known theorem-proving strategy and has been efficiently implemented [3]. Model elimination is essentially Prolog, with contrapositives of clauses used when there are non-Horn clauses. In addition, if a subgoal is complementary to one of its ancestors, the subgoal succeeds. This is commonly called *reduction operation*. Recently, a *positive refinement* of model elimination (MEP) was proposed [2]. The basic idea is to perform reduction operation only on negative subgoals.

Encouraged by David Plaisted, I modified a Prolog version of PTTP [4], which implements ME, to implement MEP. I have performed tests on non-Horn problems, mainly from [3]. In comparison with ME, MEP performs better (about 20% or more) or equally well on most of the problems (35 out of 38 problems), in spite of the fact that MEP usually needs to search one or two levels deeper than ME to find the proofs. We show some of the results in the following table (''example'' is the non-obvious problem that has generated considerable discussion in the *AAR Newsletter*). The data were obtained on a Sun 3/60 using the ALS Prolog compiler (Version 0.60) and do not indicate the performance of PTTP in general.

Theorem	ME (CPU sec.)	MEP (CPU sec.)
chang&lee8	15.10	6.90
example	>24 hrs	19088.23
fex4t2	4158.40	10596.90
fex6t1	58968.62	10577.00
fex6t2	184.40	138.02
hasparts2	278.62	14.18
ls65	42.00	42.00
ls75	19.13	9.30
ls87	279.83	155.68
ls103	8.48	5.42
ls116	37.35	44.93
prim	15.10	6.33
qw	10.00	26.00
schubert	2483.08	920.23
wos19	23.72	18.55
wos29	22.68	20.70

References

[1] Loveland, D.W., "A Simplified Format for the Model Elimination Theorem-Proving Procedure," *J. ACM*, Vol. 16, no. 3, pp. 349-363, July 1969.

[2] Plaisted, D.A, "A Sequent Style Model Elimination Strategy and a Positive Refinement," *Journal of Automated Reasoning*, in press. (Also TR89-014, Department of Computer Science, University of North Carolina, Chapel Hill, 1989.)

[3] Stickel, M.E., "A PROLOG Technology Theorem Prover," *Journal of Automated Reasoning*, Vol. 4, no. 4, pp. 353-380, 1988.

[4] Stickel, M.E., "A Prolog Technology Theorem Prover: A New Exposition and Implementation in Prolog," Technical Note 464, SRI International, 1989.

Andrews' Challenge Problem Revisited

Art Quaife (quaife@garnet.berkeley.edu)

Peter Andrews posed the following challenge problem at the Fourth Workshop on Automated Deduction in 1979:

Theorem A

$$((\exists x \forall y(p(x) \equiv p(y)) \equiv (\exists u q(u) \equiv \forall u u q(uu))) \equiv$$

$$(\exists w \forall z(q(w) \equiv q(z)) \equiv (\exists v p(v) \equiv \forall v v p(vv)))).$$

Andrews reportedly stated that he would supply the first 500 clauses for free. Each equivalence sign that is eliminated produces a doubling of the subformulas that it governs.

Guha and Zhang addressed this challenge problem in [2]. They compared solutions obtained with their program RRL to solutions obtained by the resolution theorem prover OTTER. The clausal conversion routines contained in OTTER produce 128 clauses for this problem. By introducing new predicates as abbreviations for subformulas, Guha and Zhang obtained several conversions with fewer clauses. Their best conversion contains only 36 clauses.

My clausal conversion program handles this problem even more efficiently *without* introducing any abbreviations, producing only 32 clauses in the conversion. This conversion is shown in the Appendix. My program performs more efficiently than the OTTER conversion routine, in that it (1) pulls out and unites existential quantifiers, replacing $(\exists x\, A(x) \vee \exists y\, B(y) \vee C)$ by $\exists x\, (A(x) \vee B(x) \vee C)$; (2) checks whether any clause is subsumed by a simpler factor; and (3) eliminates any clause that is subsumed by another clause. There is still room for improvement in my conversion, since only 16 of the 32 clauses are actually used in the subsequent proof.

For the same reason one can see that Guha and Zhang's 36-clause conversion is not optimal. By introducing defined abbreviations, I have obtained a conversion that produces only 24 clauses. Every clause is used in the subsequent proof.

In the conversion to clausal form, I eliminate equivalence signs by using one of the two formulas

$$\neg (X \equiv Y) \text{ iff } ((X \vee Y) \,\&\, (\neg X \vee \neg Y)),$$

$$(X \equiv Y) \text{ iff } ((\neg X \vee Y) \,\&\, (X \vee \neg Y)).$$

When such a doubling of the subformulas is about to occur, my conversion program makes estimates of the number of literals in the conjunctive normal form that will result, with and without introducing an abbreviation for a subformula. It introduces an abbreviation if the estimated number of literals will be reduced thereby.

Consider a subformula of the form

$$C \equiv D,$$

where C and D are both complex sentences. Guha and Zhang's procedure abbreviates this as

$$n1 \equiv n2,$$

and introduce new axioms

$$n1 \equiv C, \quad n2 \equiv D,$$

where $n1$ and $n2$ are new propositional constants. But clearly, one more new constant has been introduced here than is needed. My program abbreviates this as

$$m1 \equiv C,$$

with one new axiom

$$m1 \equiv D.$$

For this reason, and because my program only introduces an abbreviation when an estimate suggests that it will be helpful, my program introduces only 3 new constants in the conversion of Theorem A. By comparison the Guha and Zhang procedure introduces 10 new constants in their best conversion.

Two other solutions to Andrews' problem have been reported that I know of. Henschen [3] reported solving the problem with a conversion program that produced 86 clauses, followed by a resolution proof that terminated at line 1024. De Champeaux [1] reported solving the problem by reducing the negated theorem to FALSE with his INSURER/INSTANCE conversion routines that have more quantifier logic built into them.

Guha and Zhang also present the following variant of Andrews' challenge:

Theorem P

$$((\exists x \forall y(p(x) \equiv p(y)) \equiv (\exists u q(u) \equiv \forall u u - p(uu))) \equiv$$
$$(\exists w \forall z(q(w) \equiv q(z)) \equiv (\exists v p(v) \equiv \forall v v q(vv)))).$$

Here again their best conversion produced 36 clauses, with 10 new constants.

If I set my conversion program not to introduce abbreviations, it also produces 36 clauses on this problem. But when I let it introduce abbreviations, it does even better on this problem than the last, producing only 22 clauses with 3 new constants. Again, every clause is used in the subsequent proof. I show this conversion in the Appendix.

The constants my program introduces are obtained from Guha and Zhang's constants as follows:

$$m1 = n9, \qquad m2 = n10, \qquad m3 = (n6 \equiv n10).$$

My conversion program is written in Arity Prolog and uses many routines provided by William McCune. The two conversions that introduce definitions each take about 2 seconds on my desktop Everex 386/25. The two conversions that do not introduce definitions take about 15 seconds each.

The following are the statistics for the subsequent OTTER version 2.0 proofs, run on a VAX 8800. The clauses retained and proof lengths include counting the axioms. All proofs were obtained using hyperresolution and factoring as the inference rules. For the proofs of A-24 and P-22 (using abbreviations), I put only the last clause in the set of support. For the proofs of A-32 and P-36 (no abbreviations), I put all clauses in the set of support, and also set the OTTER flag "process_input" to first factor all the input clauses.

Theorem	Clauses retained	Proof length	Time (sec.)
A-24 clauses	194	81	6.74
A-32 clauses	97	53	2.71
P-22 clauses	89	49	1.42
P-36 clauses	107	53	3.03

For comparison, the proof times required by Guha and Zhang on their best 36-clause conversions were 39.9 and 25.5 seconds, respectively, using RRL on a VAX 11/780.

My estimating formulas are not sufficiently sophisticated to be worth reproducing. There is a problem with relying on an estimate of the number of clauses that will be produced. Consider a formula of propositional logic of the form $P \equiv Q$. If this formula is a theorem, then any propositional constant within P that can influence P's value must also appear in Q, and vice versa. This means that when we multiply the negation of this formula out, many simplifications should be possible, such as elimination of tautologous clauses and subsumption of other clauses. As a result, an *a priori* estimate of number of resulting clauses based on a symbol count may turn out erroneously high. Of course, rather than relying on an estimate, one could carry out the *full* conversions both with and without the definition, and use the better one.

It is worth commenting on an anomaly in OTTER. A natural way to order clauses is to place simplest ones first, and most complicated ones last. For example if one is developing a theory, one will normally first have a list of axioms, followed by a list of previously proven theorems which are usually more complicated. However, in applying an inference rule, OTTER searches the axiom list from *bottom to top*. If the clauses are naturally ordered, this means that OTTER tries to make more complicated inferences before it tries simple inferences, and it often happens that OTTER will find a complicated proof when a simpler one was at hand. For this reason, in the clause list for Theorem P in the Appendix, I have put the shortest clauses at the bottom of my axiom list.

References

[1] De Champeaux, D., ''Subproblem Finder and Instance Checker,'' *J. ACM*, Vol. 33, no. 4, pp. 633-657, 1986.

[2] Guha, A., and Zhang, H. ''Andrews' Challenge Problem: Clause Conversion and Solutions,'' *AAR Newsletter* No. 14, pp. 5-8, December 1989.

[3] Henschen, L., et al. ''Challenge Problem 1,'' *SIGART Newsletter*, 72, pp. 30-31, July 1980.

Appendix

The following is the list of clauses I obtained for the two problems. Symbols beginning with ''f'' or ''c'' are Skolem functors or individual constants, respectively. I use sequent notation, in which commas before the conditional sign stand for &, while those that follow the implication sign stand for v.

Clauses for negation of Theorem A, not using abbreviations:

1. $p(cx),q(cw) \rightarrow p(y3),q(z3)$.
2. $p(cx),q(z4) \rightarrow p(y3),q(cw)$.
3. $p(y4),q(cw) \rightarrow p(cx),q(z3)$.
4. $p(y4),q(z4) \rightarrow p(cx),q(cw)$.
5. $p(cx),p(x7),p(fy5(x7)),q(cw) \rightarrow q(z3)$.
6. $p(cx),p(x7),p(fy5(x7)),q(z4) \rightarrow q(cw)$.
7. $p(cx),q(cw),q(w9),q(fz5(w9)) \rightarrow p(y3)$.
8. $p(cx),q(w2) \rightarrow p(y1),q(w4),q(fz(w4))$.
9. $p(cx),q(w3),q(fz(w3)) \rightarrow p(y1),q(w4)$.
10. $p(cx) \rightarrow p(y1),q(cw),q(w3),q(fz(w3))$.

11. $p(x5),p(fy(x5)),q(cw) \rightarrow p(x2),q(z1)$.
12. $p(x5),q(cw) \rightarrow p(x2),p(fy(x2)),q(z1)$.
13. $p(x9),p(fy5(x9)),q(z4) \rightarrow p(x8),q(cw)$.
14. $p(x9),q(z4) \rightarrow p(x8),p(fy5(x8)),q(cw)$.
15. $p(y2) \rightarrow p(cx),q(cw),q(w3),q(fz(w3))$.
16. $p(y4),q(cw),q(w9),q(fz5(w9)) \rightarrow p(cx)$.
17. $p(y4),q(w6) \rightarrow p(cx),q(w9),q(fz5(w9))$.
18. $p(y4),q(w9),q(fz5(w9)) \rightarrow p(cx),q(w10)$.
19. $q(cw) \rightarrow p(cx),p(x1),p(fy(x1)),q(z1)$.
20. $q(z2) \rightarrow p(cx),p(x1),p(fy(x1)),q(cw)$.
21. $p(cx),p(x1),p(fy(x1)),q(w3),q(fz(w3)) \rightarrow q(cw)$.
22. $p(cx),p(x1),p(fy(x1)) \rightarrow q(cw),q(w3),q(fz(w3))$.
23. $p(cx),p(x7),p(fy5(x7)),q(cw),q(w9),q(fz5(w9)) \rightarrow$.
24. $p(x1),p(fy(x1)),q(cw),q(w3),q(fz(w3)) \rightarrow p(cx)$.
25. $p(x4),p(fy(x4)),q(w1) \rightarrow p(x2),q(w4),q(fz(w4))$.
26. $p(x4),p(fy(x4)),q(w3),q(fz(w3)) \rightarrow p(x2),q(w4)$.
27. $p(x4),q(w1) \rightarrow p(x2),p(fy(x2)),q(w4),q(fz(w4))$.
28. $p(x4),q(w3),q(fz(w3)) \rightarrow$ '$p(x2),p(fy(x2)),q(w4)$.
29. $p(x7),p(fy5(x7)) \rightarrow p(cx),q(cw),q(w9),q(fz5(w9))$.
30. $q(cw),q(w3),q(fz(w3)) \rightarrow p(cx),p(x1),p(fy(x1))$.
31. $q(w9),q(fz5(w9)) \rightarrow p(cx),p(x7),p(fy5(x7)),q(cw)$.
32. $\rightarrow p(cx),p(x7),p(fy5(x7)),q(cw),q(w9),q(fz5(w9))$.

Clauses for negation of Theorem P, using abbreviations:

1. $m1,m3,p(x1),p(fy(x1)) \rightarrow$.
2. $m1,m3 \rightarrow p(x1),p(fy(x1))$.
3. $m1,p(cx) \rightarrow m3,p(y4)$.
4. $m1,p(y5) \rightarrow m3,p(cx)$.
5. $m2,m3,q(cw) \rightarrow q(z1)$.
6. $m2,m3,q(z) \rightarrow q(cw)$.
7. $m2,q(w2),q(fz5(w2)) \rightarrow m3$.
8. $m2 \rightarrow m3,q(w2),q(fz5(w2))$.
9. $m3,p(cx) \rightarrow m1,p(y1)$.
10. $m3,p(y2) \rightarrow m1,p(cx)$.
11. $m3,q(w1),q(fz2(w1)) \rightarrow m2$.
12. $m3 \rightarrow m2,q(w1),q(fz2(w1))$.
13. $p(x2),p(fy3(x2)) \rightarrow m1,m3$.
14. $q(cw) \rightarrow m2,m3,q(z4)$.
15. $q(z3) \rightarrow m2,m3,q(cw)$.
16. $\rightarrow m1,m3,p(x2),p(fy3(x2))$.
17. $m1,q(u1) \rightarrow q(uu1)$.
18. $m2,p(v1) \rightarrow p(vv1)$.
19. $p(cvv) \rightarrow m2$.
20. $q(cuu) \rightarrow m1$.
21. $\rightarrow m2,p(cv)$.
22. $\rightarrow m1,q(cu)$.

OTTER 2.0 Now Available

Bill McCune (mccune@mcs.anl.gov)

The theorem prover OTTER 2.0 has been released. Improvements over earlier versions include a termination ordering for demodulation, the option of using Prolog-style (upper case) variables, rewriting of atoms as well as terms, and new options for controlling paramodulation and for choosing the given clause.

To get a copy by FTP, connect to hermes.mcs.anl.gov, username anonymous; any password will do. Go to pub/Otter, and follow the directions in README.FTP. (In case you need the net address of hermes.mcs.anl.gov, it is 192.5.200.25.)

If you are not able to use FTP or if you want just the PC or the Macintosh version, let me know—I will send information. (If you can FTP and then get the files to your PC or your Macintosh, that is fine, because the UNIX version contains the PC and the Macintosh executable files.)

Bill McCune
MCS-221
Argonne National Laboratory
Argonne, IL 60439-4844
mccune@mcs.anl.gov

Searching for Open Questions

Larry Wos (wos@mcs.anl.gov)

I am enlisting your assistance in accruing a set of open questions to attack with an automated reasoning program. The attempt to answer various open questions has clearly resulted in significant advances for the fields of automated theorem proving and automated reasoning. When success is obtained, the appreciation for the potential value of our field increases markedly.

At Argonne National Laboratory, we have begun organizing a set of open questions. Currently, we have questions focusing on Robbins algebra, nonassociative rings, Tarskian geometry, and formulas from equivalential calculus. As an example, consider the following:

Question: Is every Robbins algebra Boolean?

The following 3 axioms define a Robbins algebra:

$$x + y = y + x$$
$$(x + y) + z = x + (y + z)$$
$$-(-(x + y) + -(x + -y)) = x$$

A Robbins algebra is Boolean if Huntington's axiom

$$-(-x + y) + -(-x + -y) = x$$

can be derived.

Similar questions would be of interest, but we also welcome questions totally unlike any previously attacked. Questions need not be taken from mathematics and logic; physics, circuit design, or the area of puzzles offers challenging questions to attempt to answer with an automated reasoning program. Ideally, each question should be accompanied by one or more appropriate references, the needed definitions, and some comments about its importance.

I volunteer to maintain at Argonne National Laboratory a file of such questions, available by electronic mail through FTP and other means, and also available by request in the form of surface mail. Questions can be mailed to L. Wos electronically (wos@mcs.anl.gov) or by surface mail (L. Wos, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL 60439-4844).

Perhaps you can interest other mathematicians, logicians, physicists, and other scientists in contributing to this set of open questions. Let us set a goal of having 50 well-defined open questions in hand by the end of 1991. My thanks to each and all of you for the effort that will be required.