Association for Automated Reasoning Newsletter

No. 14 December 1989

From the AAR President, Larry Wos...

This last newsletter for 1989 features two articles contributed by Hantao Zhang and his colleagues. We are delighted to report on his research, and we encourage others to follow his lead. We are especially interested in collecting new challenge problems that our readers may attack with various automated reasoning programs. Such problems are essential to evaluating progress and generating new ideas.

International Symposium on Artificial Intelligence and Mathematics

The International Symposium on Artificial Intelligence and Mathematics will take place January 3-5, 1990, at Fort Lauderdale, Florida. It is the first of a biennial series featuring applications of mathematics in artificial intelligence as well as artificial intelligence techniques and results in mathematics. Among the invited presentations are the following:

Martin Davis, Courant Institute, "Remarks on the Foundations of Artificial Intelligence"
Zohar Manna, Stanford, "Automated Deduction—Techniques and Applications"
Drew McDermott, Yale, "Numerical Methods in Artificial Intelligence"
Alan Robinson, Syracuse, "Artificial and Natural Proofs in Mathematics"
Leslie Valian, Harvard, "Computational Learning Theory"

For more information, contact the Symposium Organizing Chair, Dr. Frederick Hoffman, Florida Atlantic University, Dept. of Mathematics, P.O. Box 3091, Boca Raton, Florida 33431-0991, (407) 367-3345, email hoffman@servax.bitnet.

Non–Obviousness — Last Time ?

Deepak Kapur (SUNY at Albany) and Hantao Zhang (University of Iowa)

The so-called non-obvious problem introduced by Pelletier and Rudnicki in *AAR Newsletter* 6 has been extensively discussed (see also *Newsletters* 7, 9, and 10), with the most recent solution given in *Newsletter* 11. We had, in fact, solved the problem quite easily and automatically (i.e., with no guidance to the theorem prover such as providing a set of support or using any special technique such as symmetry) with the aid of the theorem prover *RRL* (*Rewrite Rule Laboratory*) [3] soon after the problem was first posed. At that time, we did not see any need to publish the proofs in the AAR newsletter because of the *obviousness* of the proofs and a lack of much challenge in the problem.

We now feel that it might be instructive for the readers to study the proofs obtained on *RRL* using the *Gröbner–basis* method developed by Kapur and Narendran [2] and the *clausal superposition* method developed by the authors [5]. These two proofs are different (at least in their appearance) from those presented previously in the above cited newsletters.

The input of the problem consists of the following six clauses:

$$\begin{array}{lll} (1)\ \neg q(c,d) & (3)\ p(x,y) \vee q(x,y) & (5)\ q(x,z) \vee \neg q(x,y) \vee \neg q(y,z) \\ (2)\ \neg p(a,b) & (4)\ q(x,y) \vee \neg q(y,x) & (6)\ p(x,z) \vee \neg p(x,y) \vee \neg p(y,z) \end{array}$$

Like Hsiang's method [1], the Gröbner–basis method needs to transform formulas into polynomials over a Boolean ring (using the two rules $\neg x \rightarrow x \oplus 1$ and $x \vee y \rightarrow (x \wedge y) \oplus x \oplus y$, where $\oplus$ denotes the *exclusive-or*) and then transform each formula into a rewrite rule. While the right side of a rule must be either 0 or 1 in Hsiang's method, it can be a polynomial less than the left side in the Gröbner–basis method. For instance, the rule made from $q(x,y) \vee \neg q(y,x)$ is $(q(x,y) \wedge q(y,x)) \oplus q(y,x) \rightarrow 0$ in Hsiang's method, but is $q(x,y) \wedge q(y,x) \rightarrow q(y,x)$ in the Gröbner–basis method. The reduction and superposition inference rules are similar to those in Buchberger's Gröbner–basis algorithm for polynomial ideals over the rationals, and are discussed in [2]. When the above clauses are input to *RRL*, after generating 280 critical pairs (from which 87 rewrite rules are made), the inconsistency $1 = 0$ is found (a critical pair is similar to a resolvent). The whole process takes about 1 minute on a Vax 780 (run in Franz Lisp). The proof consists of 44 rewrite rules (including the 6 made from the input) and is included at the end of this note.

The clausal superposition method [5] transforms each clause into a conditional rewrite rule. For instance, the clause $q(x,y) \vee \neg q(y,x)$ gives the rule $q(x,y) \rightarrow 1$ **if** $q(y,x)$. Conditional rewriting is performed by these rules to simplify other clauses and clausal superposition is performed to generate new clauses. After generating only 73 critical pairs (from which 68 conditional rewrite rules are made), *RRL* found a proof consisting of 30 rules in half a minute on Vax 780. We include the proof obtained by this method also at the end of this note.

Both the methods are based on rewriting techniques and employ completion procedure based approach to automated deduction. They are implemented in *RRL* and have been successfully tried on a number of challenge problems [3,4].

In the proofs below, `+` stands for $\oplus$ and `~` stands for $\neg$; $\wedge$ is omitted; `[3, 4]` denotes that Rule `[3]` is superposed into Rule `[4]` to produce another rule.

Proof found by the Gröbner–Basis Method

```
Input [1] q(c, d) ---> 0
Input [2] p(a, b) ---> 0
Input [3] (p(x, y)q(x, y)) ---> (1 + p(x, y) + q(x, y))
Input [4] (q(x, y)q(y, x)) ---> q(y, x)
Input [5] (q(x, y)q(x, z)q(y, z)) ---> (q(x, y)q(y, z))
Input [6] (p(x, y)p(x, z)p(y, z)) ---> (p(x, y)p(y, z))
[3, 2] (reduced by [2])   [7] q(a, b) ---> 1
[6, 2]   [8] (p(y, b)p(a, y)) ---> 0
[4, 7] (reduced by [7])   [9] q(b, a) ---> 1
[5, 7] (reduced by [7])   [10] (q(a, z)q(b, z)) ---> q(b, z)
[5, 7] (reduced by [7])   [11] (q(x, a)q(x, b)) ---> q(x, a)
[5, 9] (reduced by [10], [9])   [12] q(a, z) ---> q(b, z)
[5, 9] (reduced by [11], [9])   [14] q(x, a) ---> q(x, b)
[3, 14] (reduced by [14])   [15] (p(x, a)q(x, b)) ---> (1 + p(x, a) + q(x, b))
[4, 14] (reduced by [14], [12], [4])   [16] (q(x, b) + q(b, x)) ---> 0
[3, 12] (reduced by [12])   [17] (p(a, z)q(b, z)) ---> (1 + p(a, z) + q(b, z))
[3, 1] (reduced by [1])   [18] p(c, d) ---> 1
[4, 1]   [19] q(d, c) ---> 0
[5, 1]   [20] (q(y, d)q(c, y)) ---> 0
[3, 19] (reduced by [19])   [21] p(d, c) ---> 1
[5, 19]   [22] (q(y, c)q(d, y)) ---> 0
[6, 21] (reduced by [21])   [23] (p(c, z)p(d, z)) ---> p(c, z)
[6, 21] (reduced by [21])   [24] (p(x, c)p(x, d)) ---> p(x, d)
[6, 18] (reduced by [23], [18])   [25] p(c, z) ---> p(d, z)
[18, 25]   [26] p(d, d) ---> 1
[6, 18] (reduced by [24], [25], [26])   [27] p(x, c) ---> p(x, d)
[3, 27] (reduced by [27])   [28] (p(x, d)q(x, c)) ---> (1 + p(x, d) + q(x, c))
[17, 27] (reduced by [27])   [29] (p(a, d)q(b, c)) ---> (1 + p(a, d) + q(b, c))
[3, 25] (reduced by [25])   [30] (p(d, z)q(c, z)) ---> (1 + p(d, z) + q(c, z))
[15, 25] (reduced by [25])   [31] (p(d, a)q(c, b)) ---> (1 + p(d, a) + q(c, b))
[20, 31] (reduced by [20])   [37] (p(d, a)q(b, d)) ---> q(b, d)
[4, 37] (reduced by [15], [4])   [39] p(d, a) ---> 1
[6, 39] (reduced by [39])   [41] (p(x, a)p(x, d)) ---> p(x, d)
[3, 41] (reduced by [41], [3])   [42] (p(x, a)q(x, d)) ---> (1 + p(x, a) + q(x, d))
[28, 41] (reduced by [41], [28])   [46] (p(x, a)q(x, c)) ---> (1 + p(x, a) + q(x, c))
[22, 46] (reduced by [22])   [59] (p(x, a)q(d, x)) ---> q(d, x)
[3, 59] (reduced by [59], [3])   [60] (p(x, a)p(d, x)) ---> (1 + p(x, a) + p(d, x))
[4, 59] (reduced by [42], [4])   [61] p(x, a) ---> 1 + q(x, d) + q(d, x)
[60, 61] (reduced by [61], [3])   [70] (p(d, x)q(x, d)) ---> (1 + p(d, x) + q(x, d))
[8, 70] (reduced by [8], [17])   [79] q(b, d) ---> 1
[16, 70] (reduced by [79], [3])   [80] q(d, b) ---> 1
[20, 70] (reduced by [30], [20])   [81] p(d, x) ---> 1
[8, 81]   [83] p(a, d) ---> 0
[29, 83] (reduced by [83])   [84] q(b, c) ---> 1
[22, 84] (reduced by [80])   1 == 0

The proof length is 43.
```

Proof found by the Clausal Superposition Method

```
Input    [1] q(c, d) ---> 0
Input    [2] p(a, b) ---> 0
Input    [3] p(x, y) ---> 1 if  { ~q(x, y) }
Input    [4] q(x, y) ---> 1 if  { q(y, x) }
Input    [5] q(x, z) ---> 1 if  { q(x, y), q(y, z) }
Input    [6] p(x, z) ---> 1 if  { p(x, y), p(y, z) }
[1, 4]     [7] q(d, c) ---> 0
[2, 3] (reduced by [1])    [8] q(a, b) ---> 1
[2, 6]      [9] p(a, y) ---> 0 if  { p(y, b) }
[3, 9]     [10] p(y, b) ---> 0 if  { ~q(a, y) }
[3, 10]    [12] q(a, y) ---> 1 if  { ~q(y, b) }
[6, 10]    [13] p(y, y1) ---> 0 if  { ~q(a, y), p(y1, b) }
[3, 13]    [14] q(y, y1) ---> 1 if  { p(y1, b), ~q(a, y) }
[1, 14]    [16] p(d, b) ---> 0 if  { ~q(a, c) }
[7, 14]    [17] p(c, b) ---> 0 if  { ~q(a, d) }
[6, 17]    [19] p(y, b) ---> 0 if  { ~q(a, d), p(c, y) }
[3, 16]    [20] q(a, c) ---> 1 if  { ~q(d, b) }
[3, 19]    [26] p(c, y) ---> 0 if  { ~q(a, d), ~q(y, b) }
[3, 26]    [28] q(y, b) ---> 1 if  { ~q(a, d), ~q(c, y) }
[7, 5]     [45] q(y, c) ---> 0 if  { q(d, y) }
[4, 45]    [46] q(c, y) ---> 0 if  { q(d, y) }
[12, 45]   [47] q(d, a) ---> 0 if  { ~q(c, b) }
[20, 45]   [48] q(d, a) ---> 0 if  { ~q(d, b) }
[4, 48] (reduced by [12])      [55] q(d, b) ---> 1
[4,47] (reduced by [46], [55], [55])   [56] q(a, d) ---> 0
[28, 56]     [58] q(y, b) ---> 1 if  { ~q(c, y) }
[19, 56]     [61] p(y, b) ---> 0 if  { p(c, y) }
[4, 56] (reduced by [61])     [64] q(d, a) ---> 0
[5, 64] (reduced by [3], [56], [2])     [66] q(y, a) ---> 0 if  { q(d, y) }
[4, 66]     [67] q(a, y) ---> 0 if  { q(d, y) }
[8, 67] (reduced by [58], [1])     1 == 0

The proof length is 30.
```

References

[1] Hsiang, J. (1985). Refutational theorem proving using term-rewriting systems. *Artificial Intelligence* Journal, 25, 255-300.

[2] Kapur, D., and Narendran, P. (1985). An equational approach to theorem proving in first-order predicate calculus. Proc. of *9th IJCAI*, Los Angeles, Calif., pp. 1146–1153. An expanded version appeared as GE Technical Report 84CRD232.

[3] Kapur, D., and Zhang, H. (1988). *RRL: A Rewrite Rule Laboratory.* Proc. of *Ninth International Conference on Automated Deduction* (CADE-9), Argonne, Ill., May 1988, pp. 768–769.

[4] Kapur, D., and Zhang, H. (1987). *RRL: A Rewrite Rule Laboratory* – User's Manual. June 1987. Revised, May 1989.

[5] Zhang, H., and Kapur, D. (1987). First-order theorem proving using conditional rewriting. Proc. of *Ninth International Conference on Automated Deduction* (CADE-9), Argonne, Ill., May 1988, pp. 1–20.

Andrews' Challenge Problem: Clause Conversion and Solutions

Angshuman Guha and Hantao Zhang (University of Iowa)

In 1979 at the fourth Workshop on Automated Deduction, P. Andrews posed the following challenge problem:

$$((\exists x \forall y p(x) \equiv p(y)) \equiv ((\exists u q(u)) \equiv (\forall v p(v)))) \equiv \\ ((\exists w \forall z q(z) \equiv q(w)) \equiv ((\exists x_1 p(x_1)) \equiv (\forall x_2 q(x_2)))) \qquad (1)$$

It was reported in [1], but we learned it from [6], i.e., no. 34 of Pelletier's 75 problems for testing automatic theorem provers. It is said in [6] that "the problem is logically simple, but its size makes it difficult" because about 1600 clauses will be obtained by converting the negation of (1) into clausal form. Using our clause conversion program based on the technique implemented in *RRL* [3] and that of [5], we were able to obtain a set of as few as only 36 clauses from the negation of (1). The dramatically lesser number of clauses is promising for use as input to a resolution–style theorem prover like OTTER [4]. In this note, we describe briefly our conversion program through this example and also present some experimental results.

Clause Conversion

Traditional methods transform a first-order logic formula into clauses by (i) representing the formula by only $\wedge, \vee$ and $\neg$; (ii) removing each outmost quantifier by skolemization; and (iii) converting the formula into conjunctive normal form. It is obvious that such a transformation is not good for (1), because too many clauses will be produced.

Our clause–conversion program does not try to remove quantifiers in sequential. Instead, we replace subformulas of which the outmost symbol is a quantifier by a new predicate, and then introduce an axiom to make this new predicate equivalent to that subformula. By this way, the negation of (1) becomes

$$\neg((n_1 \equiv (n_2 \equiv n_3)) \equiv (n_3 \equiv (n_4 \equiv n_5))) \qquad (2)$$
$$n_1 = (\exists x \forall y p(x) \equiv p(y)) \qquad (3)$$
$$n_2 = (\exists u q(u)) \qquad (4)$$
$$n_3 = (\forall v p(v)) \qquad (5)$$
$$n_3 = (\exists w \forall z q(z) \equiv q(w)) \qquad (6)$$
$$n_4 = (\exists x_1 p(x_1)) \qquad (7)$$
$$n_5 = (\forall x_2 q(x_2)) \qquad (8)$$

Now, each equality is replaced by two implications of which quantifiers are removed by skolemization (in the actual implementation, these two steps are merged into one). For instance, the equality (3) is replaced by $n_2 \rightarrow q(s)$ and $q(u) \rightarrow n_2$. After this step, the formulas become quantifier–free. If we transform these formulas by the traditional method, only 52 clauses will be obtained. We do not know the origin of this technique of removing quantifiers; we learned it from Deepak Kapur and it was implemented in *RRL* to support the Gröbner–Base method [2].

The next step of our program consists in replacing complex subformulas by new predicates. The basic idea was described in [5] and is called "structure–preserving transformation," but our implementation is more flexible. Suppose a formula is considered as a tree in the usual way. Let us define that the *depth* of a subtree is 0 if it is an atom; otherwise, it is one plus the maximal depth of its sons. In a bottom–up search, our program replaces any subtree of depth α by a new predicate, where α is a non-negative integer specified by the user. For instance, suppose the parameter α is 2, then (2) is transformed into

$$\begin{aligned}&\neg(n_6 \equiv n_7)\\ &n_6 = (n_1 \equiv (n_2 \equiv n_3))\\ &n_7 = (n_3 \equiv (n_4 \equiv n_5))\end{aligned}$$

After this step, each formula has a depth of at most $\alpha + 1$ and is transformed into clauses in a traditional way ($=$ and $\equiv$ are treated identically).

Using the method described above with the default parameter value of 2, we produced 38 clauses from the negation of (1). With 3 as the parameter value, 36 clauses are produced. Note that if the parameter α in the structure–preserving transformation is big enough ($\alpha > 3$ in this example), then no new predicates will be introduced and formulas will be intact in that step. As we mentioned before, the number of clauses obtained at this case is 52.

Solutions

Many solutions of (1) are already known [1,6,4]. Andrews' problem presented in [6] (after fixing a typo) is

$$\begin{aligned}&((\exists x \forall y p(x) \equiv p(y)) \equiv ((\exists u q(u)) \equiv (\forall v q(v)))) \equiv\\ &\quad ((\exists w \forall z q(z) \equiv q(w)) \equiv ((\exists x_1 p(x_1)) \equiv (\forall x_2 p(x_2)))) \qquad (9)\end{aligned}$$

which is different from (1) in that $p(v)$ was replaced by $q(v)$ and $q(x_2)$ was replaced by $p(x_2)$. This is still a valid theorem because we can derive a contradiction from its negation. The number of clauses obtained by our program from the negation of (9) is 38 with the default parameter value of 2, and 36 with a parameter value of 3.

We experimented these two versions of Andrews's problem in OTTER [4] and *RRL* [3] with different formats of the input. The experimental results of the following problems are summarized in Table 1:

A1: the original formula of the negation of (1);

A36: the 36 clauses obtained by our program ($\alpha = 3$) from the negation of (1);

A38: the 38 clauses obtained by our program ($\alpha = 2$) from the negation of (1);

A52: the 52 clauses obtained by our program ($\alpha > 3$) from the negation of (1);

P1: the original formula of the negation of (9);

P36: the 36 clauses obtained by our program ($\alpha = 3$) from the negation of (9);

P38: the 38 clauses obtained by our program ($\alpha = 2$) from the negation of (9);

P52: the 52 clauses obtained by our program ($\alpha > 3$) from the negation of (9).

In the table, "infer. rule" gives the inference rule used in OTTER: "binary" means binary resolution and "hyper" means hyperresolution. In each case, factoring is used as an additional inference rule. "clause gene." and "crit. pairs" give the number of generated clauses or critical pairs (similar to resolvents). "clause retain" and "rules gene." give the number of retained clauses or rewrite rules made from retained critical pairs. "proof length" gives the number of clauses or rewrite rules which define a proof. Empty entries in the table correspond to cases where we were unable to obtain proofs. The timing is taken on a Vax 11/780.

OTTER has a procedure for converting formulas into clauses. The procedure produces only 128 instead of 1600 clauses from the negation of (1) without introducing any new predicates.

Prob.	OTTER					RRL			
	infer. rule	clause gene.	clause retain	time (sec.)	proof length	crit. pairs	rules gene.	time (sec.)	proof length
A1	binary	1226	550	255.0	163	43	39	14.2	31
A36	—	—	—	—	—	54	84	39.9	59
A38	hyper	32180	841	535.2	148	33	80	41.9	59
A52	binary	11316	1122	194.7	110	24	51	14.9	35
P1	binary	1074	468	199.0	155	28	38	13.3	31
P36	hyper	8087	247	80.4	86	22	66	25.5	52
P38	hyper	4129	307	46.6	94	23	69	25.2	52
P52	binary	11276	1120	194.4	109	15	43	11.6	34

Table 1: Summary of Experimental Results

We learned from Bill McCune that in OTTER, the equality $A \equiv B$ is replaced by $AB \vee \bar{A}\bar{B}$ instead of $(\bar{A} \vee B) \wedge (A \vee \bar{B})$. The former turns out to be much better than the latter when an equivalence symbol ($\equiv$) appears under the negation symbol.

The Gröbner-basis method of *RRL* can accept any format of the input. It transforms formulas into polynomials over a Boolean ring and then transform each formula into a rewrite rule [2]. The technique discussed in the previous section is used to remove quantifiers before a formula is transformed into polynomials.

From the experimental results, we may conclude that (i) Andrews' challenge problem (either (1) or (9)) is not suitable for a clause–based prover (transforming the input into clauses results in inefficiency); (ii) the clause transformation techniques described in this note can reduce the number of clauses and may speed up a resolution–style prover like OTTER, but fewer clauses do not imply that faster solutions can be found (A36, for example). Finally, we point out that structurally almost identical formulas like A36 and P36 (see Appendix) can have completely different degrees of difficulty in theorem proving.

References

[1] Champeaux, D. (1979) "Sub-problem finder and instance checker: Two cooperating pre-processors for theorem provers," IJCAI 6, 191–196.

[2] Kapur, D., and Narendran, P. (1985). An equational approach to theorem proving in first-order predicate calculus. Proc. of *8th IJCAI*, Los Angeles, Calif.

[3] Kapur, D., and Zhang, H. (1988). *RRL: A Rewrite Rule Laboratory.* Proc. of *Ninth International Conference on Automated Deduction* (CADE-9), Argonne, Ill., May 1988, pp. 768–769.

[4] McCune, W., (1988) "Otter 1.0 Users' Guide," Argonne National Laboratory, Argonne, Ill., ANL-88-44.

[5] Plaisted, D., Greenbaum, S., (1986) "A structure–preserving clause form translation," *J. of Symbolic Computation*, 2, 293–304.

[6] Pelletier, F. J., (1986) "Seventy-five problems for testing automatic theorem provers," *J. of Automated Reasoning*, 2, 191–216.

Appendix

The 36 Clauses from the Negation of (1)

```
 1. -n2 | -n9 | -n6 | -n10
 2. -n2 | -n9 | n6 | n10
 3. n2 | n9 | -n6 | -n10
 4. n2 | n9 | n6 | n10
 5. -n2 | n9 | -n6 | n10
 6. -n2 | n9 | n6 | -n10
 7. n2 | -n9 | -n6 | n10
 8. n2 | -n9 | n6 | -n10
 9. -q(s8) | n8
10. -n8 | q(x10)
11. -p(x9) | n7
12. -n7 | p(s7)
13. -n5(x8) | n6
14. -n6 | n5(s6)
15. -q(s5(w)) | -q(w) | n5(w)
16. q(s5(w)) | q(w) | n5(w)
17. -n5(w) | -q(x7) | q(w)
18. -n5(w) | q(x7) | -q(w)
19. -p(s4) | n4
20. -n4 | p(x6)
21. -q(x5) | n3
22. -n3 | q(s3)
23. -n1(x4) | n2
24. -n2 | n1(s2)
25. -p(x) | -p(s1(x)) | n1(x)
26. p(x) | p(s1(x)) | n1(x)
27. -n1(x) | -p(x) | p(x3)
28. -n1(x) | p(x) | -p(x3)
29. -n3 | -n4 | n9
30. n3 | n4 | n9
31. -n3 | n4 | -n9
32. n3 | -n4 | -n9
33. -n7 | -n8 | n10
34. n7 | n8 | n10
35. -n7 | n8 | -n10
36. n7 | -n8 | -n10
```

The 36 Clauses from the Negation of (9)

```
 1. -n2 | -n9 | -n6 | -n10
 2. -n2 | -n9 | n6 | n10
 3. n2 | n9 | -n6 | -n10
 4. n2 | n9 | n6 | n10
 5. -n2 | n9 | -n6 | n10
 6. -n2 | n9 | n6 | -n10
 7. n2 | -n9 | -n6 | n10
 8. n2 | -n9 | n6 | -n10
 9. -p(s8) | n8
10. -n8 | p(x8)
11. -p(x7) | n7
12. -n7 | p(s7)
13. -n5(x6) | n6
14. -n6 | n5(s6)
15. -q(x) | -q(s5(x)) | n5(x)
16. q(x) | q(s5(x)) | n5(x)
17. -n5(x) | -q(x) | q(x5)
18. -n5(x) | q(x) | -q(x5)
19. -q(s4) | n4
20. -n4 | q(x4)
21. -q(x3) | n3
22. -n3 | q(s3)
23. -n1(x2) | n2
24. -n2 | n1(s2)
25. -p(x) | -p(s1(x)) | n1(x)
26. p(x) | p(s1(x)) | n1(x)
27. -n1(x) | -p(x) | p(x1)
28. -n1(x) | p(x) | -p(x1)
29. -n3 | -n4 | n9
30. n3 | n4 | n9
31. -n3 | n4 | -n9
32. n3 | -n4 | -n9
33. -n7 | -n8 | n10
34. n7 | n8 | n10
35. -n7 | n8 | -n10
36. n7 | -n8 | -n10
```