ASSOCIATION FOR AUTOMATED REASONING NEWSLETTER

No. 17 March 1991

From the AAR President, Larry Wos...

This our first issue of 1991 features three articles from readers around the world. J. Zhang (China) has a short paper on ternary Boolean algebra. K. Truemper (Texas—deserves a world of its own) presents a new logic programming system called Leibniz. And M. Bruschi (Italy) revives discussion of the halting problem, first presented in the *AAR Newsletter* in 1987. I am delighted to see that the *AAR Newsletter* is truly taking on an international flavor.

A Bigger Deal

In the last issue of the *AAR Newsletter,* I made an offer I assumed no one could refuse: an $11 savings to AAR members in the subscription cost for the *Journal of Automated Reasoning.* My careful calculations were based on the belief that Kluwer was charging $61 for the 1991 volume.

I was wrong — for the first time since 1928.

In fact, AAR members will save $19, because the 1991 subscription price for **non-members** is actually $69.

Now that *is* a big deal.

Poll of AAR Members

The *Journal of Automated Reasoning* is planning to begin accepting LaTex input from authors. Since many of our AAR members are also subscribers to the *Journal,* we are using this issue to conduct an informal poll to determine the easiest form for authors to submit articles.

Specifically, were you to submit an article to the *Journal of Automated Reasoning,* would you be able (1) to produce LaTex input, and (2) to submit the article on an IBM diskette? (To be

perfectly clear, we do not mean other diskettes, such as those from an Apple, Macintosh, Sun, or RISC system).

We would appreciate your sending your answer by e-mail to Gail Pieper (pieper@mcs.anl.gov) or by surface mail to Gail Pieper, Mathematics and Computer Science Division, Argonne National Laboratory, 9700 South Cass Avenue, Argonne, IL 60439-4844.

Conferences

Frontiers of Computer Science: The Bledsoe Symposium

A group of distinguished speakers will address recent advances in artificial intelligence, automated reasoning, computational aspects of biology, and related topics on the occasion of W. W. Bledsoe's 70th birthday.

The talks will occur on November 15–16, 1991 (Friday and Saturday), at the Thompson Conference Center, located on the campus of the University of Texas at Austin. A banquet will be held at the Driskill Hotel on Friday evening, November 15. The Driskill Hotel is offering a special rate for conference attendees (604 Brazos St., Austin, TX 78701, (512) 474-5911). Early hotel and air registration is advisable because Austin hotels and flights are sometimes full on a football weekend.

For registration information, contact

Joanne Click
Office of External Affairs
Computer Sciences Department
University of Texas at Austin
Austin, Texas 78712-1188

Internet email: click@cs.utexas.edu
Phone: (512) 471-9729
FAX: (512) 471-8815

There is no registration fee, but registration will be required because seating is limited. There will be a charge for the banquet.

Sponsored by the Computer Sciences Department of the University of Texas at Austin, Ricoh, and EDS.

11th International Conference on Automated Deduction

The 11th International Conference on Automated Deduction (CADE) will take place June 14-18, 1992. CADE is the major research forum covering all aspects of automated deduction. Original papers in automated deduction (for nonclassical as well as classical logics) are invited; specific topics of interest include (but are not limited to) the following:

Applications	Induction	Program Synthesis
Abductive Reasoning	Inference Systems	Rewrite Rules
Deductive Databases	Logic Programming	Theorem Proving
Decision Procedures	Program Verification	Unification Theory

CADE-11 will be held at the Ramada Renaissance Hotel in Saratoga Springs, New York, and will be hosted by the State University of New York at Albany. Saratoga Springs is located about 25 miles (40 km) north of the Albany airport.

Original research papers, system summaries that describe working reasoning systems, and problem sets that provide innovative, challenging tests for automated reasoning systems are solicited. Research papers should not exceed 5,000 words (about 15 proceedings pages). System summaries and problem sets will be limited to two and five proceedings pages, respectively.

The title page of the submission should include author's name, address, phone number, and email address. Papers must be unpublished and not submitted for publication elsewhere. Submissions that are late, too long, or that require major revision risk rejection.

Authors should send 6 copies of their submission to the Program Chair Deepak Kapur (518-442-4281; kapur@cs.albany.edu), Institute for Programming and Logics, Department of Computer Science, CSI LI67A, State University of New York at Albany, Albany, NY 12222.

Preliminary inquiries may be sent to the Program or to the Local Arrangements Chair Neil V. Murray (518 442-3393; nvm@cs.albany.edu).

The Program Committee, submission deadline, notification date, and camera-ready copy due date will be announced in a future mailing.

A 3-place Commutative Operator from Ternary Boolean Algebra

Jian Zhang, Institute of Software, Academia Sinica, Beijing, P.R. China

In his book *Automated Reasoning: 33 Basic Research Problems* [3], Wos posed the following open question: Is there a complete set of reductions for ternary Boolean algebra (TBA) ? TBA is defined by the following five axioms:

```
(1) f(f(v,w,x),y,f(v,w,z)) = f(v,w,f(x,y,z))
(2) f(y,x,x) = x
(3) f(x,y,g(y)) = x
(4) f(x,x,y) = x
(5) f(g(y),y,x) = x
```

From the above axioms, we can derive some interesting equalities. The results were obtained using my own implementation of the Knuth-Bendix algorithm [1]. Lemma 6 is borrowed from [2]. The notation adopted here is as follows. Each derivation step takes the form of $< (m)$:

$s; (n1), (n2), .. >$, where $(m), (n1), (n2)$ are equalities, and s is a substitution. It means we first make the substitution s in (m) and then apply $(n1)$, $(n2)$, etc. Thus, each step is a paramodulation plus some demodulations. The following is the derivation sequence:

```
(6) g(g(x)) = x
                < (1) : v = g(g(x)), w = x, y = z = g(x) ; (2) (3) >
(7) f(f(v,w,x),y,w) = f(v,w,f(x,y,w))   < (1) : z = w ; (2) >
(8) f(w,y,f(v,w,z)) = f(v,w,f(w,y,z))   < (1) : x = w ; (2) >
(9) f(f(w,y,v),w,y) = f(v,w,y)          < (8) : z = y ; (2) (7) >
(10) f(g(y),w,y) = w                    < (9) : v = g(y) ; (3) (4) >
(11) f(v,y,w) = f(v,w,y)                < (7) : x = g(w) ; (10) >
(12) f(x,w,g(x)) = w                    < (10): y = g(x) ; (6) >
(13) f(v,w,y) = f(w,v,y)                < (8) : z = g(w) ; (3)(12)(11) >
```

The equalities (11) and (13) imply that the operator f has the interesting property of 3-place commutativity, that is, $f(x1, y1, z1) = f(x, y, z)$ if $(x1, y1, z1)$ is a permutation of (x, y, z). Thus, if we adopt the classical definition of "completeness" and not rewrite modulo a set of equations, then there is no complete set of reductions for TBA.

References

[1] Knuth, D. E., and Bendix, P. B., "Simple word problems in universal algebras," in *Computational Problems in Abstract Algebra,* J. Leech, ed., Pergamon Press (1970), pp. 263–297.

[2] Winker, S., and Wos, L., "Automated generation of models and counterexamples and its application to open questions in ternary Boolean algebra," Proc. 8th Inter. Symp. on Multiple Valued Logic, IEEE & ACM (1978), pp. 251–256.

[3] Wos, L., *Automated Reasoning: 33 Basic Research Problems,* Prentice-Hall (1988), p. 156.

The Leibniz Logic Programming System

K. Truemper, University of Texas at Dallas

We have created a new logic programming system called Leibniz that is intended to be a direct competitor with Prolog languages, expert system shells, and other logic programming software. Leibniz is used as follows.

- The user formulates the logic aspects of a problem using the Leibniz syntax. The statements are collected in a file, say PROBLEM.LOG. In mathematical terms, PROBLEM.LOG contains the axioms of a logic problem. Leibniz accepts any mixture of CNF and DNF statements of propositional logic, IF..THEN statements, certain quantification and predicate cases of first order logic, and certain specified predicates (e.g., equal, or less than). We should emphasize, though, that at this time we do not treat Skolem functions except for Skolem constants. Thus Leibniz is much more restricted in its use than many of the theorem provers discussed in the

AAR Newsletter. Leibniz has been purposely designed for practical applications involving CAD design problems, supervision of operations, analysis of malfunctions, etc., which do not involve Skolem functions. At present, Leibniz processes up to 1,000 variables and 1,000 clauses. But, in principle, much larger problems can be handled.

- The **Translator** of Leibniz converts the axioms of PROBLEM.LOG into internal format, say creating a new file PROBLEM.CNF.

- The **Program Generator** of Leibniz examines the data of PROBLEM.CNF and constructs a solution algorithm, called the Program Generator. The Program Generator is a huge program containing numerous algorithms. The analysis by the Program Generator is complicated and can be time consuming, but has polynomial time complexity when the Simplex Method (which is one of the algorithms used) is replaced by any polynomial time algorithm for linear programming. The theoretical foundation of the Program Generator includes diverse ideas of combinatorial theory, and linear and Boolean algebra. No use is made of resolution or of resolution-related methods. The solution algorithms produced by the Program Generator can be proved to have polynomial time complexity for large classes of logic problems, including many classes of logic problems arising from applications.

- The user writes a program in C or Fortran that deals with the problem of interest. Whenever a logic question comes up, the user program invokes the solution program generated in step 3 via a conveniently organized interface to get an answer, within the guaranteed time bound. Thus, time-critical problems such as the monitoring of operations or the analysis of critical equipment failures can be processed in real time, with the answer guaranteed to be available within a typically small amount of time.

In timing comparisons against a number of Prolog languages and theorem provers, the solution algorithms produced by the Leibniz Program Generator have consistently proved to be faster than other methods, often by a factor of 10, and on difficult problems often by a factor of 50 and more. More important, the Leibniz solution times are stable and reliable, regardless of whether a problem is satisfiable or not. Also, small problem changes cause small variations in solution times.

Let us look at a simple example problem that has occupied almost every issue of th *AAR Newsletter,* the "Nonobvious Problem" of Pelletier and Rudnicki. To make the problem a bit more interesting, we have recast it. We assume a Herbrand set $\{a, b, c, d\}$ and get the following self-explanatory formulation.

SATISFY

nonobvious problem by Pelletier and Rudnicki

SETS

DEFINE hset

a

b

c

d

```
  PREDICATES
    DEFINE p ON hset X hset
    DEFINE q ON hset X hset
FACTS
    FOR ALL x IN hset
    FOR ALL y IN hset
      p(x,y) OR q(x,y).
    FOR ALL x IN hset
    FOR ALL y IN hset
      q(x,y) OR NOT q(y,x).
    FOR ALL x IN hset
    FOR ALL y IN hset
    FOR ALL z IN hset
      q(x,z) OR NOT q(x,y) OR NOT q(y,z).
    FOR ALL x IN hset
    FOR ALL y IN hset
    FOR ALL z IN hset
      p(x,z) OR NOT p(x,y) OR NOT p(y,z).
ENDATA
```

The Translator converts the input file into internal format in 2.3 sec (all times are on a Toshiba 5200, about 2.8 MIPS). The Program Generator produces the solution program in 40.5 sec and lists an upper time bound for theorem proving of 3.7 sec. A second (optional) time-bound calculation is done in 1.6 sec and yields a tighter upper time bound of 1.7 sec. We have applied the solution program to settle for a number of statements whether they are theorems for the given system with the given Herbrand set. Below we list three examples of the negated statements:

p(a,b) AND q(c,d) (not satisfiable; this is the original "nonobvious problem")

p(a,b) (satisfiable)

NOT q(a,b) (satisfiable)

In each case we have tried, the answer is given in less than 0.5 sec, well below the upper bound time of 1.7 sec.

Leibniz is being used by students of our university in courses in logic programming and expert systems. Students with no prior knowledge of expert systems have built interesting systems with Leibniz quickly and reliably. Their logic formulations sometimes result in apparently hard logic problems that go way beyond the simplicity of production rules and that nevertheless are readily processed by Leibniz.

Leibniz has a number of additional features. Here we mention just two of them.

1. Leibniz can solve min-cost satisfiability problems, where each variable or predicate instance has two costs, one for "True" and one for "False." One must find a satisfying solution of minimum total cost or prove that no satisfying solution exists. This feature is very useful for the design of expert systems where certain solutions are preferred to others.

2. Leibniz allows expressions of fuzzy logic in the problem formulation. For example, one may say: "IF gauge_low AND temperature_high THEN FREQUENTLY oil_loss OR coolant_loss." Instead of "FREQUENTLY", one could also have specified "SOMETIMES", etc. Leibniz then produces several answers for a given satisfiability question. The answers are ordered by the likelihood of being correct, with the most likely one listed first. Repeated theorem proving is needed to locate the answers. But Leibniz has been designed to be good at that task and thus locates the desired answers rapidly.

The Leibniz Translator and Program Generator run on any IBM-compatible PC with at least 5 MB memory and Microsoft OS2 operating system. Installation on other computer platforms can be readily accomplished. The solution programs produced by the Program Generator run on virtually every computer. They require between 200 kB and 2 MB of memory, depending on the size of the solution algorithm.

Leibniz is available in three versions:

- FiXpert (system is as described above, with fuzzy logic, but without min-cost satisfiability option).
- QuiXpert (system exactly as described above).
- OptXpert (= QuiXpert plus capability of solving (together with logic computation) linear equations, linear inequality systems, linear and integer programming problems).

Special Leibniz use licenses are available for academic and nonprofit institutions. A drastically downscaled MS-DOS version of QuiXpert exists and is used by students at our university. It runs on any IBM-compatible PC with 640 kB memory and Microsoft DOS 3.0 or later, and handles up to 200 variables and 300 clauses; the user program must be written in C.

License fees for commercial applications are not low. The systems are available from Leibniz, 2304 Cliffside Drive, Plano, TX 75023.

The Leibniz company was established for the support of research in computational logic and logic programming. Related research efforts have been supported in part by the National Science Foundation (currently under Grant DMS-9000376) and by the German Alexander von Humboldt Foundation under the Senior Scientist Award Program.

The Halting Problem

Massimo Bruschi (Study University of Milan - mbruschi@imiucca.unimi.it)

Recently, I found in a very old issue of the *AAR Newsletter* [2] a formulation of a theorem I had worked on more than a year ago. The theorem involves a first-order axiomatization that L.

Burkholder had given of the halting problem. In [1] he had given also a (manual) natural deduction proof. No mention of (automated solutions of) this problem has been made in subsequent issues of the *Newsletter*. I present here an automated proof obtained after a manual conversion of the original formulation.

The original symbolization key is as follows:

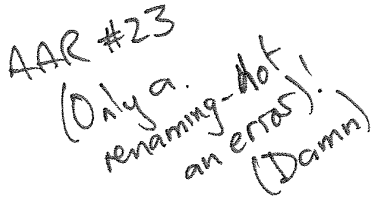
```
Ax     : x is an algorithm
Cx     : x is computer program in some programming language
Dxyz   : x is able to decide whether y halts given input z
H2xy   : x halts on given input y
H3xyz  : x halts on given input the pair <y,z>
Oxy    : x outputs y
```

Four premises are given ("Ex" means "exists x"; "Vx" means "all x"). The first premise states that if an algorithm that solves the halting problem exists, then it can be written as a computer program in some language:

```
1. Ex[Ax & Vy(Cy -> VzDxyz)] -> Ew[Cw & Vy(Cy -> VzDwyz)]
```

The second premise states what is meant when a program w is able to decide whether a program y halts or does not halt on input z. What program w does is the following: If program y halts given input z, then program w halts given as input the pair $< y, z >$ and prints out 'G' (for "good"). On the other hand, if program y does not halt given input z but goes on forever, then program w halts on $< y, z >$ and prints out 'B' (for "bad"):

```
2. Vw([Cw & Vy(Cy -> VzDwyz)] -> Q
      VyVz([(Cy &  H2yz) -> (H3wyz & OwG)] &
           [(Cy & -H2yz) -> (H3wyz & OwB)]))
```



Now, program y may be given iteslf as input. The third premise says that if program w decides the termination of a program y on y as an input, then there exist a slightly different program v that can determine only whether a program halts on itself:

```
3. Ew[Cw & Vy([(Cy &  H2yy) -> (H3wyy & OwG)] &
              [(Cy & -H2yy) -> (H3wyy & OwB)])] ->
   Ev[Cv & Vy([(Cy &  H2yy) -> (H2vy & OvG)] &
              [(Cy & -H2yy) -> (H2vy & OvB)])]
```

The final premise says that if a program such as v exists, then so does a yet slightly different program wthat goes into a loop just in those cases when program v would halt and print out "G":

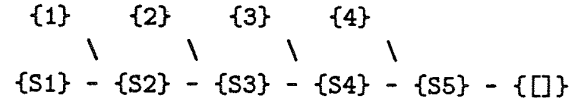
```
4. Ev[Cv & Vy([(Cy &  H2yy) -> (H2vy & OvG)] &
              [(Cy & -H2yy) -> (H2vy & OvB)])] ->
   Eu[Cu & Vy([(Cy &  H2yy) -> -H2uy] &
              [(Cy & -H2yy) -> (H2uy & OuB)])]
```

In [1], the outlines of the described programs can be found. The natural deduction proof given in [1] is by contradiction and starts with the assumption that an algorithm to solve the halting problem exists:

```
S1. Ex[Ax & Vy(Cy -> VzDxyz)]
```

I was unable to obtain a mechanical proof of the theorem simply by applying ENprover, the DSI implementation of the Hsiang's EN-Strategy method, to this axiomatization. Applications of OTTER [4], Argonne's theorem prover, also were unsuccessful. The problem seems to lie in the number and the length of the clauses derived from the transformation of the given input: 84 from the axioms and 2 from the negation of the thesis, some of them with 7 literals.

Surprisingly, an attempt to split the proof in subparts, following guidelines coming from the original one, failed. This focused attention on the structure of the proof and of the input formulas. A high-level graph of the manual proof is



The steps to get S2, S4, and S5 are given by modus ponens. For clausal-input-based techniques like EN-Strategy, a natural step that derives B from $\{A, A \rightarrow B\}$ can be hard, depending on the complexity of the structure of the sentences A and B: the number of resolution steps that simulate the modus ponens behavior depends on the number of the clauses given from A and B. A way to reduce the number of clauses correspondent to a sentence is to simplify its "structure" by introducing new predicate symbols—and the corresponding definitions—for repeated subsentences, for instance, in the spirit of the structure preserving transformation described in [5,3]. The indentation I have given to the formulas helps to "see" the idea. As an example, look at premises 1 and 2. The subsentence

```
Vy(Cy -> VzDxyz)
```

occurs twice in 1 and once in 2. It talks about an x (the only free variable) that for all the programs y and input z decides the halting of y on z. We can add the definition

```
Vx[CDx <-> Vy(Cy -> VzDxyz)]
```

and replace the original subsentence in 1 and 2 with the right instance of CDx. The equivalence maintains the validity of the original models for the reformulated theory.

Applying this method repeatedly to the problem, I obtained the following formulation:

```
1'.  ExACDx -> EwCCDw
2'.  Vw[CCDw -> VyVz(CH2H3OwyzG & CnH2H3OwyzB)]
3'.  Ew[Cw & Vy(CH2H3OwyyG & CnH2H3OwyyB)] ->
     Ev[Cv & Vy(CH2H2OvyG & CnH2H2OvyB)]
4'.  Ev[Cv & Vy(CH2H2OvyG & CnH2H2OvyB)] ->
     Eu[Cu & Vy((CH2yy -> -H2uy) & CnH2H2OuyB)]

 5.  Vx[CDx <-> Vy(Cy -> VzDxyz)]
 6.  Vx[CCDx <-> (Cx & CDx)]
 7.  Vx[ACDx <-> (Ax & CDx)]
 8.  VxVy[CH2xy <-> (Cx & H2xy)]
 9.  VxVyVzVw[H3Oxyzw <-> (H3xyz & Oxw)]
10.  VxVy[CnH2xy <-> (Cx & -H2xy)]
11.  VxVyVw[H2Oxyw <-> (H2xy & Oxw)]
12.  VxVyVzVw[CH2H3Oxyzw <->(CH2yz -> H3Oxyzw)]
13.  VxVyVzVw[CnH2H3Oxyzw <->(CnH2yz -> H3Oxyzw)]
14.  VxVyVw[CH2H2Oxyw <-> (CH2yy -> H2Oxyw)]
15.  VxVyVw[CnH2H2Oxyw <-> (CnH2yy -> H2Oxyw)]
```

Bugs pointed out by Gamzingos

This formulation gives a set of 42 clauses for the axioms and 1 clause for the (negation of the) thesis halving the original numbers. Also the length of the heaviest clause has been reduced from 7 to 5.

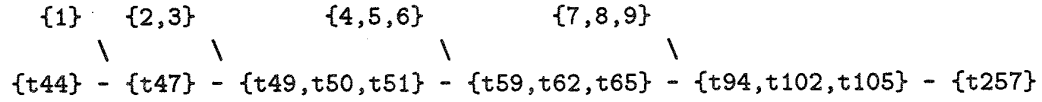
With this input, ENprover gets the following proof using a breadth-first strategy with set of support:

```
1.          axiom: -ACD(x) | CCD(SK1)
2.          axiom: -CCD(x) | CH2H3O(x,y,z,G)
3.          axiom: -CCD(x) | CnH2H3O(x,y,z,B)
4.          axiom: -C(x) | C(SK2) | -CH2H3O(x,SF1(x),SF1(x),G) |
                   -CnH2H3O(x,SF1(x),SF1(x),B)
5.          axiom: -C(x) | CH2H2O(SK2,y,G) |
                   -CH2H3O(x,SF1(x),SF1(x),G) |
```

```
                        -CnH2H30(x,SF1(x),SF1(x),B)
   6.              axiom: -C(x) | -CH2H30(x,SF1(x),SF1(x),G) |
                          CnH2H20(SK2,y,B) | -CnH2H30(x,SF1(x),SF1(x),B)
   7.              axiom: -C(x) | C(SK3) | -CH2H20(x,SF2(x),G) |
                          -CnH2H20(x,SF2(x),B)
   8.              axiom: -C(x) | -CH2(y,y) | -CH2H20(x,SF2(x),G) |
                          -CnH2H20(x,SF2(x),B) | -H2(SK3,y)
   9.              axiom: -C(x) | -CH2H20(x,SF2(x),G) |
                          -CnH2H20(x,SF2(x),B) | CnH2H20(SK3,y,B)
  13.              axiom: C(x) | -CCD(x)
  21.              axiom: -C(x) | CH2(x,y) | -H2(x,y)
  26.              axiom: -CnH2(x,y) | -H2(x,y)
  27.              axiom: -C(x) | CnH2(x,y) | H2(x,y)
  28.         26,27-simp: C(x)H2(x,y) + C(x)CnH2(x,y) + C(x) --> 0
  29.              axiom: H2(x,y) | -H20(x,y,z)
  41.              axiom: -CnH2(x,x) | -CnH2H20(y,x,z) | H20(y,x,z)
 t44.            theorem: ACD(SK4)
 t47.             t44,1-p: CCD(SK1)
 t49.            t47,13-p: C(SK1)
 t50.             t47,3-p: CnH2H30(SK1,x,y,B)
 t51.             t47,2-p: CH2H30(SK1,x,y,G)
 t57.             t51,6-p: -C(SK1) | CnH2H20(SK2,x,B) |
                          -CnH2H30(SK1,SF1(SK1),SF1(SK1),B)
 t59.[t49,t50]
                t57-simp: CnH2H20(SK2,x,B)
 t60.             t51,5-p: -C(SK1) | CH2H20(SK2,x,G) |
                          -CnH2H30(SK1,SF1(SK1),SF1(SK1),B)
 t62.[t49,t50]
                t60-simp: CH2H20(SK2,x,G)
 t63.             t51,4-p: -C(SK1) | C(SK2) |
                          -CnH2H30(SK1,SF1(SK1),SF1(SK1),B)
 t65.[t49,t50]
                t63-simp: C(SK2)
 t92.             t65,8-p: -CH2(x,x) |
                          -CH2H20(SK2,SF2(SK2),G) |
                          -CnH2H20(SK2,SF2(SK2),B) | -H2(SK3,x)
 t94.[t59,t62]
                t92-simp: -CH2(x,x) | -H2(SK3,x)
t100.             t65,9-p: -CH2H20(SK2,SF2(SK2),G) |
                          -CnH2H20(SK2,SF2(SK2),B) | CnH2H20(SK3,x,B)
t102.[t59,t62]
               t100-simp: CnH2H20(SK3,x,B)
t103.             t65,7-p: C(SK3) | -CH2H20(SK2,SF2(SK2),G) |
                          -CnH2H20(SK2,SF2(SK2),B)
t105.[t59,t62]
               t103-simp: C(SK3)
```

```
t123.          t105,28-p: H2(SK3,x) + CnH2(SK3,x) --> 1
t127.          t102,41-p: -CnH2(x,x) | H20(SK3,x,B)
t134.          t94,21-en: -C(SK3) | -H2(SK3,SK3)
t135.     t105,t134-simp: -H2(SK3,SK3)
t136.          t94,21-en: -C(x) | -H2(x,x) | -H2(SK3,x)
t155.       t136,t123-en: -C(SK3) | CnH2(SK3,SK3)
t156.     t105,t155-simp: CnH2(SK3,SK3)
t178.         t135,29-en: -H20(SK3,SK3,x)
t256.       t178,t127-en: -CnH2(SK3,SK3)
t257.     t156,t256-simp: []
```

The graph of this mechanical proof reflects that of the manual one if we substitute the original indexes with those of the corresponding clauses



As I said before, the reformulation has been made by hand. Nevertheless, the possibility of obtaining similar results by using an automatic transformation seems good.

References

[1] Burkholder, L., "The halting problem," SIGCSE Bulletin, 1987

[2] Burkholder, L., "A 76th automated theorem proving problem," *AAR Newsletter* 8, April 1987

[3] Guha, A., and Zhang, H., "Andrews's challenge problem: Clause conversion and solutions," *AAR Newsletter* 14, December 1989

[4] McCune, W., *OTTER 2.0 Users Guide,* Argonne National Laboratory Report ANL-90/9, Argonne, Illinois, March 1990

[5] Plaisted, D., and Greenbaum, S., "A structure preserving clause form translation," J. Symbolic Computation 2, 1986