ASSOCIATION FOR AUTOMATED REASONING NEWSLETTER

No. 25 February 1994

From the AAR President, Larry Wos... I am impressed, delighted, and indeed honored by the response to my letter in the previous *AAR Newsletter* asking readers to renew their memberships. The number of renewals has been overwhelming, and I am assured by our new secretary Bob Veroff that the association is once again flourishing.

Whether my note also triggered readers to attack new problems suitable for publication in the newsletter, I am not sure; but I am pleased that, in addition to membership renewals, I also received several interesting articles. We publish some of these articles here, with a promise of others in subsequent issues.

Call for Papers

1st International Conference on Constraints in Computational Logics

The first international conference on "Constraints in Computational Logics" (CCL) will take place September 7–9, 1994, at the Technical University Munich, in Munich, Germany. CCL is a new conference sponsored by the ESPRIT WORKING GROUP "Construction of Computational Logics."

The past two decades have seen a proliferation of different programming styles: functional, logical, constraints based, object-oriented, among others. More recently, it has been recognized that these styles complement rather than exclude each other by being suitable for particular problem domains. As a consequence, combining programming paradigms has emerged as a significant research direction of its own, and constraints have often been used as a glue for these combinations.

CCL aims to attract high-quality original papers covering theoretical and practical issues in this direction of combining and extending programming paradigms preferably (but not exclusively) by using constraints. Topics of interest include symbolic constraints, set constraints, numerical constraints, constraints for knowledge representation and processing, use of constraints for type checking and program analysis, multiparadigm programming, abstract properties of combined calculi, combinations of computational logics, constraints in rewriting, deduction, and symbolic computations.

CCL'94 invited five lecturers: Max Dauchet (symbolic constraints and tree automata), Dexter Kozen (set constraints and logic programming), Helmut Simonis (applications of constraint logic programming), Gert Smolka (concurrent constraint programming), and Wayne Snyder (constraints in automated deduction).

Paper submission: 5 hard copies of a full paper and 16 additional copies of the cover page and second page must be received by March 25, 1994, by the program chair: Jean-Pierre Jouannaud, LRI, Bât. 490, CNRS et Université de Paris-Sud, 91405 Orsay Cedex, FRANCE (e-mail ccl@lri.fr; phone (33) 1-69416905; fax: (33) 1-69416586). The cover page of the submission should include the title, the abstract, and for all authors, the name, address, phone number, fax number, and e-mail when available. Each submission must include as the second page a clear statement of the issues, a summary of the main results, and an explanation of their significance and relevance to the conference, all phrased for the nonspecialist. Technical developments, directed to the specialist, should follow starting on the third page. The whole paper should not exceed 16 pages in LNCS LaTeX format. Papers must be in English and must provide sufficient explanations to allow the program committee to assess the merits of the paper. References and comparisons with related work should be included. Proofs, if omitted in the paper, must appear in an additional appendix for which there is no space limit.

People planning to attend CCL are requested to send a note as early as possible to the conference chair: Tobias Nipkow, Institut für Informatik, Technische Universität München, D-80538 München, GERMANY (e-mail: ccl@informatik.tu-muenchen.de; phone: (49) 89 2105 2690; fax: (49) 89 2105 8183).

KI-94

The 18th German Annual Conference on Artificial Intelligence will take place at Saarbrücken, September, 18–23, 1994. General Chairs are Jörg Siekmann and Hans-Jürgen Bürckert. Contributions to the Scientific Conference and the Industrial Congress will be published in a set of Springer-Verlag conference publications.

The KI-94 Scientific Conference will take place Sept. 18–22. The program consists of invited talks, reviewed paper presentations, tutorials, workshops, poster sessions, and system demonstrations. Conference languages are German and English. Contributions from all aspects of AI are welcome, especially those that address the following subjects:

knowledge representation	knowledge acquisition
deduction, inference systems, logic programming	cognition
machine learning	architectures, methods, tools
natural language processing	robotics, intelligent interfaces
image processing and understanding	diagnosis, planning
social impacts	configuration
qualitative reasoning	neural networks

Papers and proposals for posters should be sent in six copies to one of the two program chairs by April 8, 1994: Prof. Dr. Leoni Dreschler-Fischer, Universität Hamburg, FB Informatik, Bodenstedtstrasse 16, D-22765 Hamburg, Phone: + 49 40 4123-6132 (-6128), e-mail: dreschler@rz.informatik.uni-hamburg.d400.de; or Prof. Dr. Bernhard Nebel, Universität Ulm, Fakultät für Informatik, James-Franck-Ring, D-89081 Ulm, Phone: +49 731 502-4122 (-4121), e-mail: nebel@informatik.uni-ulm.de

The KI-94 Industrial Congress will take place on September 22–23. The aim is to stimulate the exchange of information between AI-researchers and AI-users and to support the transfer of research results to practical and economical applications. Papers (approximately 6000 words) are invited from areas that include industrial production, sales and marketing, disposition and planning, banking and finance, insurance, transport, medicine, office, communication, and AI software environments. For further information, contact Prof. Dr. Bernd Neumann, Labor für Künstliche Intelligenz, Bodenstedtstr. 16, D-22765 Hamburg.

AAAI Events

AAAI Spring Symposium Series 1994, March 21–23, 1993, at Stanford University, California. Contact: AAAI, 445 Burgess Drive, Menlo Park, Calif.; 415-328-3123; fax: 415-321-4457; e-mail: sss@aaai.org

AAAI-94, Twelfth National Conference on Artificial Intelligence, July 31–August 4, 1994, at Seattle, Washington. Contact: AAAI, 445 Burgess Drive, Menlo Park, Calif.; 415-328-3123; fax: 415-321-4457; e-mail: ncai@aaai.org

IAAI-94. Sixth Annual Conference on Innovative Applications of Artifical Intelligence, July 31–August 4, 1994, at Seattle, Washington. Contact: AAAI, 445 Burgess Drive, Menlo park, Calif.; 415-328-3123; fax: 415-321-4457; e-mail: iaai@aaai.org

New Software Releases

OTTER v3.0

W. W. McCune, Mathematics and Computer Science Division,
Argonne National Laboratory, Argonne, Illinois
mccune@mcs.anl.gov

I am pleased to announce the release of version 3.0 OTTER.

Here is the abstract from the manual:

OTTER (Organized Techniques for Theorem-proving and Effective Research) is a resolution-style theorem-proving program for first-order logic with equality. OTTER includes the inference rules binary resolution, hyperresolution, UR-resolution, and binary paramodulation. Some of its other abilities and features are conversion from first-order formulas to clauses, forward and back subsumption, factoring, weighting, answer literals, term ordering, forward and back demodulation, evaluable functions and predicates, and Knuth-Bendix completion. OTTER is coded in C, is free, and is portable to many different kinds of computer.

This release is for UNIX-type operating systems. I have had reports that a beta version OTTER 3 compiles in DOS (Turbo-C and GCC) and for Macintosh (Think-C) with minor changes to the source.

Some features new in OTTER 3:

- In the autonomous mode, the user simply inputs clauses, and OTTER decides on inference rules and strategies.

- The hot list can be used to give emphasis to some of the input clauses.
- The user can declare function symbols to be infix with associativity and precedence so that expressions can be written in a natural way.
- The inference rule gL builds in a generalization principle for cubic curves.
- Given clauses can be selected interactively.
- Some optimizations have been implemented for propositional problems.

You can FTP a copy from info.mcs.anl.gov, file pub/Otter/otter-3.0.0.tar.Z . Unpack it in the usual way:

```
uncompress otter-3.0.0.tar.Z
tar xvf otter-3.0.0.tar
```

This will create a directory otter-3.0.0 containing several subdirectories. See otter-3.0.0/README for further information. Send a note to otter@mcs.anl.gov for current information.

When bugs surface and are fixed, patch levels will be released. The n-th round of bug fixes will be called otter-3.0.n. The patch levels are full distributions, so it is nearly always best to fetch the highest patch level.

The TPTP Problem Library, Release v1.0.0

Geoff Sutcliffe, Christian Suttner, and Theodor Yemenis

The TPTP (Thousands of Problems for Theorem Provers) Problem Library is a collection of test problems for automated theorem provers (ATPs), using the clausal normal form of first-order predicate logic. The TPTP aims to supply the ATP community with the following:

1. A comprehensive list of the ATP test problems available today, in order to provide a simple and unambiguous reference mechanism.
2. New generalized variants of those problems whose original presentation is hand-tailored towards a particular automated proof.
3. The availability of these problems via FTP in a general-purpose format, together with a utility to convert the problems to existing ATP formats. (Currently the OTTER, MGTP, PTTP, SETHEO, and SPRFN formats are supported, and the utility can easily be extended to produce any format required.)
4. A comprehensive list of references and other interesting information on each problem.
5. General guidelines outlining the requirements for ATP system evaluation.

The principal motivation for this project is to move the testing and evaluation of ATP systems from the present ad hoc situation onto a firm footing. This is necessary since results currently being published seldom provide an accurate reflection of the intrinsic power of the ATP system being considered. A common library of problems is necessary for meaningful system evaluations, meaningful system comparisons, repeatability of testing, and the production of statistically significant results. The TPTP is such a library.

Release v1.0.0 of the TPTP is now available.

The TPTP can be obtained by anonymous ftp from either the Department of Computer Science, James Cook University, Australia, or the Institut fuer Informatik, TU Muenchen, Germany. There are three files, ReadMe, TPTP-v1.0.0.tar.Z, and TR-v1.0.0.ps.Z. General information about the library is in the ReadMe file, the library is packaged in TPTP-v1.0.0.tar.Z, and a technical report describing the TPTP (in postscript form) is in TR-v1.0.0.ps.Z. Please read the ReadMe file, as it contains up-to-date information about the TPTP. The technical report serves as a manual explaining the structure and use of the TPTP. It also explains much of the reasoning behind the development of the TPTP.

The TPTP is regularly updated with new problems, additional information, and enhanced utilities. If you would like to be kept informed of such developments, please e-mail Geoff Sutcliffe (geoff@cs.jcu.edu.au) or Christian Suttner (suttner@informatik.tu-muenchen.de).

Automatic Proofs of Some Boolean Algebra Problems

Hantao Zhang
The University of Iowa, Iowa City, IA 52242
hzhang@cs.uiowa.edu

In the November 1993 issue of the AAR Newsletter (No. 24), Bob Veroff gives us a set of interesting problems on Boolean algebra. For RRL, a theorem prover based on rewriting and completion, the most difficult problem in this set is to prove the associativity of the operator $*$. Using the RRL's default ordering—the recursive path ordering – to orient equations into rewrite rules, RRL could not prove this theorem. However, if we use the Knuth-Bendix ordering (assuming the weight of each function is 1), RRL is able to prove this theorem in a minute (running in Lucid Common Lisp) on a Sparc 2 workstation. Actually, it is Deepak Kapur who first used RRL to obtain this proof during a workshop held at Argonne National Laboratory in the summer of 1990.

Interestingly, before the associativity of $*$ is generated, RRL generates all the other theorems in Bob's list, with the only exception of theorem **TE** (which is not equational; see below). We think that RRL's proofs may help people to solve these problems using their own theorem provers.

The axioms of the Boolean algebra as given by Bob are as follows:

A1. Commutativity: $x + y = y + x$; $x * y = y * x$.

A2. Distributivity: $x + (y * z) = (x + y) * (x + z)$; $x * (y + z) = (x * y) + (x * z)$.

A3. Identity: $x + 0 = x$; $x * 1 = x$.

A4. Complement: $x + i(x) = 1$; $x * i(x) = 0$.

The theorems are (the number in the bracket preceding each equation is the rewrite rule number assigned to that equation in our proof):

TA. Idempotence: [28] $x + x = x$; [27] $x * x = x$.

TB. Boundedness: [24] $x + 1 = 1$; [25] $x * 0 = 0$.

TC. Absorption: [33] $x + (x * y) = x$; [26] $x * (x + y) = x$.

TD. Associativity: [78] $x + (y + z) = (x + y) + z$; [165] $x * (y * z) = (x * y) * z$.

TE. Uniqueness of Complement: if $x + y = 1$ and $x * y = 0$, then $y = i(x)$.

TF. Involution: [49] $i(i(x)) = x$.

TG. Complement of 0 and 1: [10] $i(0) = 1$; [9] $i(1) = 0$.

TH. DeMorgan' Laws: [159] $i(x + y) = i(x) * i(y)$; [156] $i(x * y) = i(x) + i(y)$.

In the following, we provide the detailed proofs produced by Herky, a theorem prover developed in the RRL environment for efficient completion. We present the proofs as a sequence of rewrite rules generated by Herky during the completion of the axioms.

Following each rewrite rule, we append a message indicating how this rewrite rule is obtained. For instance, if a rewrite rule is made by superposing $l_1 \rightarrow r_1$ into $l_2 \rightarrow r_2$, we provide the subterm t of l_2 to indicate that the rewrite rule is obtained by unifying l_1 and t with the mgu (most general unifier) σ, and the critical pair is $\sigma r_2 = \sigma l_2[t \leftarrow r_1]$.

Herky can detect the communtative and/or associative laws of any binary operator, and then automatically turn on special completion procedures for these properties. The use of these special properties in generating a rewrite rule is not acknowledged in the message following that rule. Even though in the proof, these special properties are listed as rewrite rules, they are not used for rewriting.

In a superposition, two rewrite rules are assumed to be variable-disjoint. In Herky, this is done by explicitly renaming x to x_1, y to y_1, etc., in one rule (for instance, if the original rule is numbered as [8], the variable-renamed rule is denoted by [8*]).

```
[1] (x + y) ---> (y + x)     { from Input #1. }
[2] (x * y) ---> (y * x)     { from Input #2. }
[3] (0 + x) ---> x     { from Input #3. }
[4] (1 * x) ---> x     { from Input #4. }
[5] (i(x) + x) ---> 1     { from Input #5. }
[6] (i(x) * x) ---> 0     { from Input #6. }
[7] ((x * y) + (x * z)) ---> ((y + z) * x)     { from Input #7. }
[8] ((x + y) * (x + z)) ---> ((y * z) + x)     { from Input #8. }
[9] i(1) ---> 0     { by superposing Rule [4] into (x1 * i(x1)) of Rule [6*] with
     the mgu [x1|1, x|i(1)]. }
[10] i(0) ---> 1     { by superposing Rule [3] into (x1 + i(x1)) of Rule [5*] with the
     mgu [x1|0, x|i(0)]. }
[11] ((x + y) * x) ---> ((0 * y) + x)     { by superposing Rule [3] into (x1 + z1) of
     Rule [8*] with the mgu [z1|0, x|x1]. }
[12] (i(x) + (x * y)) ---> (i(x) + y)     { by superposing Rule [5] into (x1 + y1) of
     Rule [8*] with the mgu [x1|i(y1), x|y1], and then reduced by Rule [4]. }
[13] ((i(x) * y) + x) ---> (x + y)     { by superposing Rule [5] into (x1 + z1) of [8*]
     with the mgu [z1|i(x1), x|x1], and then reduced by Rule [4]. }
```

```
[14] ((0 * x) + 1) ---> (1 + x)     { by superposing Rule [4] into (x1 * (x1 + y1)) of
       Rule [11*] with the mgu [x1|1, x|(1 + y1)]. }
[15] ((0 * 0) + x) ---> (x * x)     { by superposing Rule [3] into (y1 + x1) of [11*]
       with the mgu [y1|0, x|x1]. }
[21] ((0 * x) * 1) ---> (1 * 0)     { by superposing Rule [14*] into (x + y) of [11]
       with the mgu [x|(0 * x1), y|1], and then reduced by Rule [7]. }
[24] (1 + x) ---> 1     { by superposing Rule [4] into (y1 * i(x1)) of Rule [13*] with
       the mgu [y1|1, x|i(x1)], and then reduced by Rule [5]. }
[25] (0 * x) ---> 0     { by deleting [21] by [24], and then reduced by [24] and [4]. }
[26] ((x + y) * x) ---> x    { by reducing the right-hand side of [11] by [25], [3]. }
[27] (x * x) ---> x     { by deleting [15] by [25], and then reduced by [3]. }
[28] (x + x) ---> x     { by superposing Rule [6] into (i(x1) * y1) of Rule [13*], and
       then reduced by Rule [3]. }
[29] ((i(y) + x) + y) ---> 1     { by superposing Rule [26] into (y1 * i(x1)) of [13*]
       with the mgu [y1|(i(x1) + y), x|i(x1)], and then reduced by [5]. }
[33] ((x * y) + x) ---> x     { by superposing Rule [28] into (x + y) of [8] with
       the mgu [x1|y, x|y], and then reduced by Rules [26]. }
[34] ((x + y) + y) ---> (x + y)     { by superposing Rule [26] into (x1 * y1) of [33*]
       with the mgu [x1|(y1 + y), x|y1]. }
[35] ((x * y) * x) ---> (x * y)     { by superposing Rule [33*] into (x + y) of [26]
       with the mgu [x|(y * y1), x1|y], and then reduced by Rule [7]. }
[43] (((x + z) * y) + x) ---> ((y * z) + x)     { by superposing Rule [34*] into (z + x)
       of Rule [8] with the mgu [z|(x1 + x), y1|x], and then reduced by [8]. }
[48] (((i(x) + z) * y) + x) ---> (x + y)     { by superposing Rule [29*] into (z + x) of
       [8] with the mgu [z|(i(x) + x1), y1|x], and then reduced by [4]. }
[49] i(i(x)) ---> x     { by superposing Rule [6] into (x1 * y1) of Rule [12*] with
       the mgu [x1|i(y1), x|y1], and then reduced by Rules [28], and [3]. }
[52] (i((x * y)) + y) ---> 1     { by superposing Rule [35] into (x1 * y1) of [12*] with
       the mgu [x1|(y1 * y), x|y1], and then reduced by Rule [5]. }
[53] (i((x + y)) + ((y * z) + x)) ---> (i((x + y)) + (x + z))  { by superposing [8] into
       (x1 * y1) of Rule [12*] with the mgu [x1|(x + y), y1|(x + z)]. }
[57] (i((x * z)) + (x * y)) ---> (i((x * z)) + y)     { by superposing Rule [52*] into
       (x + y) of Rule [8] with the mgu [x|i((x1 * y)), y1|y], and then reduced by [4]. }
[73] (((x * y) + z) + y) ---> (y + z)     { by superposing [8] into ((z1 + x1) * y1) of
       [43*] with the mgu [x|z1, y|x1, y1|(z1 + z)], and then reduced by [26]. }
[77] ((x + y) + (y + z)) ---> ((x + y) + z)     { by superposing Rule [26] into
       (y1 * x1) of Rule [73*] with the mgu [y1|(x1 + y), x|x1]. }
[78] ((x + y) + z) ---> ((y + z) + x)     { by superposing Rule [26] into (y1 * x1) of
       [73*] with the mgu [y1|(x1 + y), x|x1], and then reduced by [77]. }
[128] (i((i(x) + y)) + x) ---> x     { by superposing Rule [6] into (y1 * (i(x1) + z1))
       of [48*] with the mgu [x|(i(x1) + z1), y1|i((i(x1) + z1))], and reduced by [3]. }
[129] (i((x + y)) + i(x)) ---> i(x)     { by superposing [49] into i(x1) of [128*]. }
[132] (i((x * y)) + i(x)) ---> i((x * y))     { by superposing Rule [6] into (y1 * x1)
       of Rule [57*] with the mgu [x|x1, y1|i(x1)], and then reduced by Rule [3]. }
[154] (i((x + y)) + x) ---> (i(y) + x)     { by superposing Rule [6] into (z1 * y1) of
       Rule [53*] with the mgu [x|y1, z1|i(y1)], and then reduced by [129] and [3]. }
[156] (i(x) + i(y)) ---> i((x * y))     { by superposing [12] into (x1 + y1) of [154*]
       with the mgu [y1|(x * y), x1|i(x)], and then reduced by Rules [132] and [154]. }
[157] i((i(x) * y)) ---> (i(y) + x)     { by superposing Rule [49] into i(x1) of Rule
```

```
      [156*] with the mgu [x1|i(x)]. }
[158] i((i(y) + x)) ---> (i(x) * y)      { by superposing Rule [157*] into i(x) of [49]
      with [x|(i(x1) * y1)]. }
[159] (i(x) * i(y)) ---> i((x + y))      { by superposing Rule [49] into i(y1) of [158*]
      with the mgu [y1|i(x)]. }
[163] ((i(z) * y) * x) ---> ((i(z) * x) * y)      { by superposing Rule [158] into
      i((i(y1) + x1)) of Rule [158*] with the mgu [x1|(i(y) + v23), x|(i(y1) + v23)],
      and then reduced by Rule [158]. }
[165] ((x * y) * z) ---> ((y * z) * x)      { by superposing Rule [49] into i(z1) of
      Rule [163*] with the mgu [z1|i(x)], and then reduced by Rule [49]. }

Number of rules generated              = 165
Number of rules retained               = 73
Number of critical pairs               = 3785 (of which 903 are unblocked.)
Time used (incl. garbage collection)   = 59.20 sec
```

Theorem **TE** can be easily proved if we add $a + b = 1$ and $a * b = 0$ as axioms and ask RRL to prove $b = i(a)$, where a and b are Skolem constants. Note also that if we replace $x * (y + z) = (x * y) + (x * z)$ by $x * 0 = 0$ in the axiom set, the associativity of $*$ can still be proved.

Solution to Another Open Question in Combinatory Logic

Jian Zhang
Institute of Software, Academia Sinica
P.O. Box 8718, Beijing 100080, P.R. China

Larry Wos [1] proposed several (open) problems on the existence of fixed point combinators in certain fragments of combinatory logic. One of them is: Does the fragment {B, N1} satisfy the strong fixed point property? The combinators B and N1 can be defined by the following two equations:

```
((Bx)y)z = x(yx)
((N1x)y)z = ((xy)y)z
```

The strong fixed point property holds if there exists a combinator y such that for all combinators `x, yx = x(yx)`.

The answer to the above problem is, *No.* In fact, the following matrix is a model of {B, N1}, but the strong fixed point property does not hold.

*	0	1	2	3	4
0	0	0	2	3	4
1	0	0	2	3	4
2	2	2	2	2	2
3	2	2	2	2	2
4	4	4	4	4	4

My program FALCON (a tool for constructing finite algebras, and a descendant of Mod/E [2]) can find such a model in about 20 seconds on a SPARCstation 2. The specification of the problem is as follows:

```
## Size of the model
(5)

## Functions
 { a(2)  }

## Equations ( 0; B; 1; N1 )
[ a(a(a(0,x),y),z) = a(x,a(y,z)) ]
[ a(a(a(1,x),y),z = a(a(a(x,y),y),z) ]

## Property
<Ay.Ex.!(a(y,x)=a(x,a(y,x)))>
```

John Slaney's program FINDER [3] can also find the model in about 80 seconds when given the following specification:

```
setting { solutions: 1 }
sort {
        int
        cardinality = 5
}
function {
        *: int, int -> int
           change-order -T
        f: int -> int.
}
clause {
        (((0 * x) * y) * z) = (x * (y * z)).
        (((1 * x) * y) * z) = (((x * y) * y) * z).
        (y * f(y)) = (f(y) * (y * f(y))) -> false.
}
```

With a more elaborate specification, FINDER could find a model in 10 seconds [4]. When B and N1 are allowed to be equal (N1 = B = 0), there is a four-element model.

Perhaps some other programs can also find the models. I am not going to make any comparison between programs or discuss the "art" of generating large models. The important fact is that the fragment {B, N1} does not possess the strong fixed point property.

References

1. Wos, L., The kernel strategy and its use for the study of combinatory logic, *J. Automated Reasoning* 10 (1993) 287–343.

2. Zhang, J., Search for models of equational theories, *Proc. 3rd Int'l. Conf. for Young Computer Scientists* (ICYCS'93), Beijing, July 1993.

3. Slaney, J., *FINDER: Finite domain emulator. Version 3.0, notes and guide,* Australian National Universtiy, 1993.

4. Slaney, J., personal communication, January 1994.

An Application of Resolution to Expert Systems: Overcoming Schoenmakers' Paradigm

Joseph S. Fulda

Introduction

In his brief but influential paper [1], Schoenmakers presents a logical paradigm that neatly captures a difficulty that arises in cases where the knowledge base of an expert system is acquired from more than one domain specialist. In this note, we review Schoenmakers' paradigm, generalize it, and show how to overcome it by applying resolution to the knowledge base, thereby exposing hidden inconsistencies between the domain specialists that would otherwise remain undetected, but would nonetheless compromise the integrity of the knowledge base.

Schoenmakers' Paradigm

Suppose one has an expert judging system that is applied to the following criminal case. There are two witnesses (the experts), one of whom testifies **P** and the other of whom testifies **P→Q**. The judge (the expert system) infers **Q**. Unfortunately, that conclusion results in the execution of an innocent man, since *both* witnesses believe **-Q**, even though neither offered testimony on the truth value of **Q**. Furthermore, neither witness' beliefs nor the testimony taken together (the knowledge base) is inconsistent, since **P & -Q** is consistent, **P→Q & -Q** is consistent, and **P→Q & P** is consistent. Nor did the judge reason incorrectly: He merely applied what is perhaps *the* most elementary principle of logic, modus ponens. Rather, what has happened is that an inconsistency between the witnesses, namely, a disagreement about the truth value of **P,** is hidden from the judge, and the verdict is consequently wrong.

Generalizing the Paradigm

There are n domain specialists, who enter into the knowledge base propositions $\&_{i,j}P_{ij}$, where P_{ij} is the i'th proposition of the j'th expert. In addition, there are other propositions, $\&_{i,j}Q_{ij}$, where Q_{ij} is the i'th proposition of the j'th expert, believed by the experts but not in the knowledge base. Finally, $\&_{i,j}P_{ij}$ is consistent, $(\forall j)(\&_i P_{ij}\ \&\ \&_i Q_{ij}$ is consistent), but $\&_{i,j}P_{ij}\ \&\ \&_{i,j}Q_{ij}$ is inconsistent.

Overcoming the Paradigm

One might assume that overcoming this difficulty would be close to impossible, since the belief spaces of the various domain specialists are extremely large and there are any number of ways in which the small number of propositions entered into the knowledge base might conflict with the much larger belief spaces. Fortunately, this is not the case when resolution-based refutation is used. So long as the form of resolution used is refutation-complete for the propositional calculus, all contradictions will result in a unit conflict. As a result, every such form of resolution will yield every unit clause that follows from the initial set of clauses. The proof of this is straightforward enough. Suppose otherwise. Take any consistent set of clauses that does not produce some unit clause, say, I, which follows from the initial set of clauses. Add to the set of clauses -I. Then the variant of resolution is not refutation-complete.

We therefore offer the following simple yet effective algorithm for detecting inconsistencies implicit in knowledge bases as a result of Schoenmakers-like enthymemes:

1. Transform all propositions in the knowledge base into clause (conjunctive normal) form.
2. Apply a refutation-complete variant of resolution to the knowledge base, storing all unit resolvents.
3. For each unit resolvent, query the domain specialists as to whether any of them is in disagreement with it.
4. If none is in disagreement, the integrity of the knowledge base can be certified. Otherwise, the knowledge acquisition process must be reopened.

In the case of Schoenmakers' original example, (1) yields **P** and **-P v Q**, (2) yields **Q**, and (3) yields a masked disagreement (**Q & -Q** *not* **P & - P**), and per (4) the knowledge acquisition process—here a criminal trial—must be reopened, and a wrongful execution thereby averted.

Future Research

Two extensions of this work merit consideration. First, it would be well to extend the algorithm to propositions expressed in the predicate calculus, a formalism far more suited for an actual knowledge base of an expert system. This is not as easy as it might seem. The standard method for expanding a formalism from the propositional calculus to the predicate calculus is to treat universally general propositions as conjunctions over the domain of discourse and existentially general propositions as disjunctions over the domain of discourse. Hence, if the domain of discourse is {a,b,c,d,e}, (∀x)Px is Pa & Pb & Pc & Pd & Pe, while (∃x)Px is Pa v Pb v Pc v Pd v Pe. The problem with applying this method here is that if two experts agree on (∃x)Px but one believes Pa & Pb & Pc & Pd & -Pe, while the other believes -Pa & Pb & Pc & Pd & Pe, are they in disagreement? There is no algorithm that could ever answer that question; the best we can say is that if the proposition is general and purposively expressed as a general proposition, then disagreement over the specific singular propositions is probably irrelevant. This view captures our normal intuition. Suppose two physicians state, in agreement, that not all depressed patients respond to antidepressants—(∃x)(Dx & -Rx)—but disagree on whether a particular depressed patient, say a, will respond to antidepressants, that is, whether Ra, surely the way the proposition was expressed—in general terms—indicates that the experts are in agreement after all, not in (masked) disagreement. Predicate logic provides for both singular and general propositions and, in this case, reducing the latter to the former does not seem to work.

Second, some restriction strategy to lessen the inefficient demands of (2) above—without losing any unit resolvents, of course—would be most helpful.

Acknowledgments

I have profited from the astute, technical comments of Professors Alan Bundy and Michael Levin on an earlier draft of this paper.

Reference

1. Schoenmakers, W. J. "A Problem in Knowledge Acquisition." *SIGART Newsletter* 95 (January 1986) 56–57.