

Tau: A Web-Deployed Hybrid Prover for First-Order Logic with Identity and Optional Inductive Proof

Jay Halcomb and Randall R. Schulz
H&S Information Systems
P.O. Box 1058, Belmont, CA 94002, USA
<http://hsinfosystems.com>
<mailto:hsis@hsinfosystems.com>

Abstract. We outline Tau, a practical and extensible hybrid theorem prover for first-order predicate calculus with identity. Tau is flexible and user-configurable, accepts the KIF Language, is implemented in Java, and has multiple user interfaces. Tau combines rule-based problem rewriting with Model Elimination, uses Brand's Modification Method to implement identity, and accepts user-configurable heuristic search to speed the search for proofs. Tau optionally implements mathematical induction. Formulas are input and output in KIF or infix FOPC, and other external forms can be added. Tau can be operated from a Web interface or from a command-line interface. Tau is implemented entirely in Java and can run on any system for which a current Java Virtual Machine is available.

Keywords: automated theorem proving (ATP), first-order logic (FOL), hybrid prover, Knowledge Interchange Format (KIF), web-based, interactive, model elimination (ME), resolution, rewriting, Java

Table of Contents

| | | |
|----|--|----|
| 1 | Introduction: How Tau Works | 2 |
| 2 | Tau and the KIF Language | 5 |
| 3 | A typical constraint deduction: axiomatizing Einstein's Mice | 6 |
| 4 | The Logical Theory of Tau | 10 |
| 5 | Resolution | 11 |
| 6 | Model Elimination and Proof Search | 11 |
| 7 | Heuristic Search | 12 |
| 8 | Brand Transformations | 15 |
| 9 | Martelli and Montanari | 15 |
| 10 | Stillman's Subsumption Algorithm | 16 |
| 11 | Computational Results | 16 |
| 12 | Logic Theorems | 19 |
| 13 | Identity Problems | 20 |
| 14 | Theory of a Successor, Presburger and Peano Arithmetic | 21 |
| 15 | Mathematical Induction | 22 |
| 16 | Graph Theory | 24 |
| 17 | Sample Statistics | 25 |
| 18 | Disproofs | 27 |
| 19 | Next Extensions of Tau | 27 |

1. Introduction: How Tau Works

Consider the remarks of (Bachmair and Ganzinger, 1998) in the *Handbook of Automated Reasoning*, “Resolution Theorem Proving”:

“It has been pointed out that **a weakness of resolution is its lack of goal orientation**. Simplification and clause elimination based on redundancy helps ameliorate the problem, but one might also consider possible combinations of resolution with such goal-oriented methods as the sequent calculus or semantic tableaux. Semantic tableaux and variants thereof, including the Davis-Putnam method, model elimination and SL-resolution can be viewed as tree-like theorem proving process in which the limits of the individual branches are saturated under (ordered) resolution with selection. **This view may serve as a basis for further investigations of the combination problem.**”, P. 94, Vol. 1, emphasis added.

That was the spirit with which we approached the Tau project. It is perhaps as difficult to induce a computer to ‘reason logically’ as it is to induce a human to do so, but with the Tau theorem prover and knowledge base (eventually we hope Tau to be a formal theory repository), we are aiming to produce the first interactive, easy-to-use, and comprehensive prover of its kind on the Internet. Tau is sound and theoretically complete for the First Order Predicate Calculus with Identity – a phrase which can cover a multitude of sins, due to the general undecidability of FOPC. Tau’s syntax is full FOPC with sentential constants, relation symbols, function symbols, complex terms, and identity.

What you can presently do with Tau: test the FOL validity of symbolic formulas; test the FOL validity of formal arguments (derive a conclusion from premises); normalize formulas – by command line interface only at this time; construct a formal FOPC theory and make deductions from it. Examples already constructed include: theorems in Presburger and Peano arithmetic, both with and without mathematical induction; theorems in the theory of commutative ordered fields; theorems in graph theory.

Tau’s initial screen is shown in Figure 1. Use of Tau is quite simple: well-formed formulas of KIF can be typed or pasted into the browser window. Then (perhaps after selecting an option check box) press ‘Prove’.

Tau is written in the Java programming language (Sun Microsystems, Ongoing). Its Web interface uses the Tomcat servlet container ((, Apache Jakarta Project)Ongoing). The user interface of the browser version of Tau is implemented in HTML and CSS using a forms-based

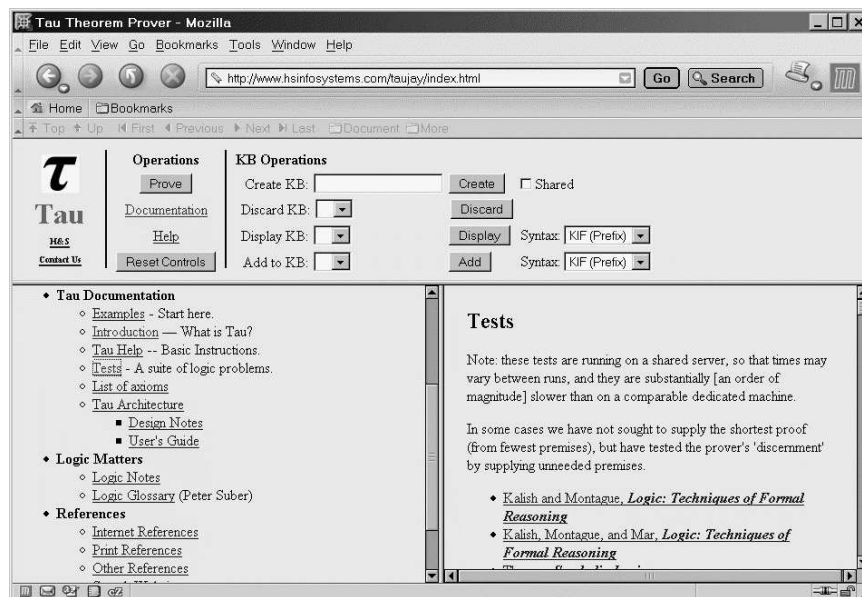


Figure 1. Initial Tau Screen

submission. The primary proof procedure employed by Tau is Loveland's well-known Model Elimination algorithm (Loveland, 1968), (Loveland, 1969), and (Loveland, 1978) augmented with a selectable variety of search algorithms, including heuristic search guided by a user-supplied heuristic ranking function.

Prior to submission to the Model Elimination algorithm, problems are optionally subjected to a process of rewriting, in which the original conclusion is rewritten into logically equivalent formulas that are more tractable. The rewriting process is recursive in the sense that the result of a rewriting may itself be rewritten further. When the problem submitted includes use of the identity predicate, the Model Elimination prover stage applies the necessary transformations, using a variant of Brand's Modification Method (Brand, 1975).

Tau is intended for experimentation and educational use. Tau can be used as a proof assistant and as a teaching aid.

Note: The URLs for test cases mentioned in this paper refer to a shared, commercial Internet hosting server with limited computational resources. As a result, problems run slowly there. Full performance of Tau can be witnessed on a dedicated host. Interested parties should contact the authors at their email address for access to this host. Also, note that in some test cases we have not sought to supply the shortest proof (from fewest premises), but we have tested the prover's 'discernment' by supplying unneeded premises.

| | |
|--|---|
| <p>2: To prove:</p> <pre>(<=> (forall ?X-3 (=> (F ?X-3) (H ?X-3))) (forall ?X-4 (=> (G ?X-4) (J ?X-4))))</pre> | <p>Split the equivalence, yielding:</p> <p><u>3:</u></p> <pre>(=> (forall ?X-3 (=> (F ?X-3) (H ?X-3))) (forall ?X-4 (=> (G ?X-4) (J ?X-4))))</pre> <p><u>4:</u></p> <pre>(=> (forall ?X-4 (=> (G ?X-4) (J ?X-4))) (forall ?X-3 (=> (F ?X-3) (H ?X-3))))</pre> |
|--|---|

Figure 2. An Example of Rewriting

Tau proofs are based upon proof-by-contradiction using a linear restriction of resolution invented by Loveland called Model Elimination. Tau's implementation of Model Elimination also incorporates (by default) the so-called Set-of-Support restriction, in which the contradiction sought in the indirect proof must exist between the negated conclusion and the premises or within the negated conclusion itself. Tau proof displays, being based upon proof-by-contradiction using a resolution strategy (specifically, Model Elimination), are not informative in the way a natural deduction style of presentation is. We intend later to expose in a more natural way some N.D. proof structure, and to provide further aid to using Tau as a semi-automated proof assistant. We do presently display some of the initial rewriting techniques used; see Figure 2.

Tau can prove either valid theorems or arguments. Other examples (e.g., the theorems labeled T260, T265, T327, HeadOfAHorse, and AssocAdd) with which to try Tau are at the URL :

<http://www.hsinfosystems.com/taujay/doc/samples/tests/>

We've been using theorems from (Kalish and Montague, 1964), (through the chapter on identity) for many of our basic tests. The entire test directory (containing over 250 tests) is at:

<http://www.hsinfosystems.com/taujay/doc/samples/testsJSP.html>.

2. Tau and the KIF Language

Typical KIF looks like:

```
(=<=>
 (exists ?Y
  (and
   (forall ?X (<=> (f ?X) (= ?X ?Y)))
   (g ?Y)))
 (and
  (exists ?Y
   (forall ?X (<=> (f ?X) (= ?X ?Y))))
  (forall ?X (=> (f ?X) (g ?X)))))
```

You can run a Tau proof of this theorem by clicking 'Prove' at:

<http://www.hsinfosystems.com/taujay/doc/samples/tests/T324.jsp>

After running that test you will see the trace of Tau's proof of the theorem. That page will show a verbose trace of Tau's actions in proving this theorem, displaying how the theorem was broken down into sub-proofs and how the formulas were rewritten and normalized to facilitate the proof. More concise proof display options are also available from running Tau in a command line mode.

KIF (Knowledge Interchange Format) is essentially a parenthesized prefix version of common first-order logical notation, which largely emanates from environs of Stanford University; a KIF-like notation is also subsumed as a part of the (Common Logic Standard, Ongoing) effort, under ISO (the International Organization for Standardization) - see end note ¹.

Being a prefix form, it is efficient for many computer applications and for that reason we adopted KIF as Tau's first internal language. Tau also has an infix syntax which is consistent with typical conventions used in ASCII computer settings. Both of these concrete syntaxes are intended for situations where no special logic symbols are available. Our architecture admits unlimited additional concrete syntaxes, including

those which include proper mathematical and logical symbology such as TeX or MathML; we intend to incorporate graphical notations and I/O into Tau.

Tau accepts both a prefix version of FOL, called KIF (Knowledge Interchange Format), and a related infix form of FOL. Tau KIF is a Lisp-like, S-Expression prefix syntax based on the KIF 3 standard. Details of KIF 3 are available at the Knowledge Interchange Format home page, <http://ksl-web.stanford.edu/knowledge-sharing/kif/>. An HTML conversion of the TeX original from the preceding page is here:

<http://logic.stanford.edu/kif/Hypertext/kif-manual.html>.

Variables in KIF are preceded by a ‘?’; individual constants, predicates, relations, and functions may be a single alphabetic character or a string of such. Computer generated individual constants appearing in our proofs are preceded by a ‘\$’.

For further details, please see:

Knowledge Interchange Format (KIF):

<http://www-ksl.stanford.edu/knowledge-sharing/kif/>

Knowledge Interchange Format, dpAns:

<http://logic.stanford.edu/kif/dpans.html>

Knowledge Interchange Format:

<http://logic.stanford.edu/kif/specification.html>

3. A typical constraint deduction: axiomatizing Einstein’s Mice

‘Einstein’s Mice’ (although of very doubtful attribution) is one of a class of traditional logic puzzles. We discuss it to illustrate some of the techniques in automated theorem proving used in Tau. The problem is given by Dr. Edmund Weitz, who provides a functional programming solution in the Lisp programming language. His formulation, discussion, and Lisp code reside on the Internet at: <http://www.weitz.de/einstein.html>.

”Our problem consists of three mice living next to each other in three holes in the wall. Each mouse has a favorite cheese flavor and a favorite TV show. Here are the hints:

1. Mickey Mouse loves Gouda.
2. Mighty Mouse’s favorite TV show is Emergency Room.

3. The mouse that lives in the left hole
never misses an episode of Seinfeld.
4. Mickey Mouse and Mighty Mouse have
one mouse hole between them.
5. The Simpsons fan does not live
on the left of the Brie lover."

Dr. Weitz' clever and quick procedural solution using classical programming correctly gives the answers: (MICKEY GOUDA SEINFELD) (MINNY BRIE SIMPSONS) and (MIGHTY EMMENTALER ER)). The sample times he reports range from .31 seconds to 3.6 seconds under various Lisp implementations and using various operating systems on a laptop (Pentium III 850 MHz, 256 MB RAM). An optimized version of the program, using heuristics, produced quicker times - quicker by a factor of 10. As you will see, our own solution times are competitive with Dr. Weitz's (a bit quicker, actually), although done inferentially rather than procedurally.

The Lisp code by Dr. Weitz is at:

<http://www.weitz.de/files/riddle.lisp>.

We formalize the Einstein's Mice problem in the following way. We use the following relational predicates: (livesInHole ?X ?Y), (favCheese ?X ?Y), (favTVShow ?X ?Y), and we denominate the holes as Hole1, Hole2, and Hole3 ¹.

We formalize premises 1-4 as:

- 1'. (favCheese Mickey Gouda)
- 2'. (favTVShow Mighty ER)
- 3'. (forall ?X (<=> (livesInHole ?X Hole1)
(favTVShow ?X Seinfeld)))
- 4'. (and (livesInHole Minnie Hole2)
(or
(and (livesInHole Mighty Hole1)
(livesInHole Mickey Hole3))
(and (livesInHole Mighty Hole3)
(livesInHole Mickey Hole1))))

Premise 5 is logically more complex. One way to express it would be to use additional predicates which would order the holes explicitly, and to use identity and Trichotomy (for instance). The way we've

¹ This style of symbolization is somewhat crude in the sense that our mouse's preferences are probably for a type of cheese, not simply some instances of the preferred types. But we ignore the philosophical difficulties.

chosen (to avoid using additional predicates) is to assume implicitly that the holes are ordered and to express condition 5 in a complex way as follows.

```
5'. (and
  ((forall (?X ?Y) (=>
    (and (livesInHole ?X Hole1)
         (livesInHole ?Y Hole2))
    (not (and (favTVShow ?X Simpsons)
              (favCheese ?Y Brie))))))

  (forall (?X ?Y) (=>
    (and (livesInHole ?X Hole1)
         (livesInHole ?Y Hole3))
    (not (and (favTVShow ?X Simpsons)
              (favCheese ?Y Brie))))))

  (forall (?X ?Y) (=>
    (and (livesInHole ?X Hole2)
         (livesInHole ?Y Hole3))
    (not (and (favTVShow ?X Simpsons)
              (favCheese ?Y Brie))))))
```

From the problem statement we also have to express the additional premises that each mouse lives in a unique hole, has a unique favorite cheese, and has a unique favorite TV show. That each mouse lives in some hole we express by (forall ?X (or (livesInHole ?X Hole1) (livesInHole ?X Hole2) (livesInHole ?X Hole3))). That each mouse lives in a unique hole (has a unique favorite cheese and TV show) we express with several simple Horn clauses of the form

```
(forall ?X (=> (livesInHole ?X Hole1)
              (not (livesInHole ?X Hole2))))), etc.
```

In this manner, we achieve the benefit of using Horn clauses and avoid the use of identity, which is computationally expensive.

With this axiomatization we ask Tau to verify the following:

```
(and
  (favCheese Mickey Gouda)
  (favCheese Minnie Brie)
  (favCheese Mighty Emmentaler)
  (livesInHole Minnie Hole2)
  (livesInHole Mickey Hole1)
```

```
(livesInHole Mighty Hole3)
(favTVShow Mighty ER)
(favTVShow Mickey Seinfeld)
(favTVShow Minnie Simpsons))
```

The result is: **Conclusion Proved creating 575 clauses taking 0.074 seconds**. This problem may be run, and the results seen in more detail, at:

<http://hsinfosystems.com/taujay/doc/samples/EinsteinsMice28R.html>

As an example of detail, the subconjunct, (favCheese Minnie Brie), is proved in 19 ME inference steps creating 151 clauses taking 0.025 seconds. This is the most difficult inference chain in the proof, taking the longest number of ME inference steps.

Of course, as Tau is an inference engine, once the problem has been axiomatized, Tau can establish the results of other logical inquiries about the problem statement. **The ability to answer such general questions is a distinct advantage which an inference engine has over a procedurally coded solution. Another advantage is that the terms and parameters of the problem can be easily changed, and one doesn't have to be a Lisp programmer to produce such results** ².

E.g., Tau can make such inferences as these.

Minnie lives in some hole:

```
(exists ?X (livesInHole Minnie ?X));
```

Minnie either lives in Hole2 or Hole3:

```
(or (livesInHole Minnie Hole2)
    (livesInHole Minnie Hole3));
```

etc.

In response to a question from Professor Feferman: we do not have to merely ask Tau to verify answers already known to be correct. We can instead instruct Tau to search for the answers to questions (e.g.): what is Mickey's favorite cheese, which hole does Minnie live in, what are the favorite TV shows, etc.? We do this by adding an additional premise which constrains the appropriate answer, and then by returning the result of the successful variable binding. For example, to ask what Mickey's favorite cheese is, we add the premise:

² Of course, one does have to be able to frame problems in logic notation. But our axiomatized statement of the problem is much simpler than the corresponding Lisp code

```
(forall ?X (=> (and (livesInHole Mighty ?X)
                    (not (livesInHole Minnie ?X))
                    (not (livesInHole Mickey ?X)))
              (answer ?X)))
```

We then ask Tau to prove, (exists ?X (answer ?X)). The resulting proof returns the value 'Hole1' in ?X. ³ The result is: **the Model Elimination proof succeeded. Proved in 9 ME inference steps creating 307 clauses taking 0.031 seconds..**

This test (and similar ones) may be run at:

<http://hsinfosystems.com/taujay/doc/samples/EinsteinsMice29R.html>

4. The Logical Theory of Tau

The Tau prover is essentially an indirect prover that proves formulas by establishing the mutual unsatisfiability of the set of clauses that result from the Skolemized form of the original input problem's formulas with the conclusion to prove first negated.

Before Skolemization, clausalization and the application of Model Elimination, the conclusion is subject to a process of rule-driven rewriting that replaces the original conclusion with other more tractable but (collectively) equivalent conclusions, each of which is proved independently. The result of any given rewriting is itself subject to rewriting.

This recursive decomposition process produces a tree of sub-proofs. Both conjunctive (all sub-proofs produced by a given rewriting must succeed) and disjunctive (only one of a rewriting's sub-proof need succeed) sub-proof combination rules are allowed. The system can optionally compute estimates of the proof complexity of each resulting sub-proof and then order the attempts to prove them so as to conclude the overall proof successfully (or fail) in the shortest time.

For some types of problems we have also implemented a direct instantiation method and an optional incremental satisfiability checker (based upon Davis-Putnam-Loveland); see (Hooker, 1993) and (Hooker, 1993a).

Tau is based on: reductio ad absurdum, or contradiction testing; normalization (see, e.g., (Baaz et al, 2002), and also see (Nonnengart

³ We've not yet made the result available in the displayed HTML on the webserver, but we will do so.

and Weidenbach, 2002)); our version of Brand transformations (see, e.g., (Brand, 1975), and (Degtyarev and Voronkov, 1999), “Equality reasoning in sequent-based calculi”), to implement identity rewriting strategies.

Tau’s use of a Model Elimination technique in conjunction with selection heuristics and proof strategizing helps overcome some of the difficulties resulting from a lack of goal-directedness.

Our primary emphasis is on the logical soundness of the proof method and the integrity of the software design. Principally via the command line interface we have a good deal of control and flexibility in choosing proof strategies, and over the presentations and annotations, and we are adding these options judiciously to the browser interface.

5. Resolution

Resolution proof was introduced in (Robinson, 1965) and (Robinson, 1971); the well-known (Chang and Lee, 1973) gave resolution further impetus. However, the resolution method requires considerable augmentation by efficient search techniques to be of practical use.

6. Model Elimination and Proof Search

The Model Elimination technique was introduced in (Loveland, 1968), (Loveland, 1969), and (Loveland, 1978), and is theoretically sound and complete. Interest in it was more lately revived with Stickel’s work on the theorem prover PTTP, e.g. (Stickel, 1984). Tau uses a version of Model Elimination with refinements to handle certain completeness issues which may arise from an uncaredful application of search techniques; for example, the Inoue problem (see below). In this regard, Tau also offers multiple search strategies, with selection heuristics (clausal weighting).

As with all automated theorem proving, search plays a central role. Tau’s implementation of the Model Elimination procedure implements these kinds of search: breadth-first search, depth-first search, heuristic search, and modified search. In all cases, a user-specified depth-cutoff is applied.

Breadth-first search is guaranteed to find the shortest proof possible for the problem, but will typically examine far too many clauses in the process of finding that shortest proof. Breadth-first search also tends to consume excessive amounts of primary storage holding clauses at the frontier of the proof search tree.

Depth-first search requires the least amount of storage and depends strongly on the depth cutoff to prevent its becoming trapped in unbounded sub-trees of the proof tree.

Heuristic search is the default and almost always produces the best overall results. Each clause in the set of clauses produced by the conclusion and each clause (or *chain*, in Loveland's terminology) that arises by successful applications of the Model Elimination inference operations is evaluated by a user-specified *heuristic function* whose purpose is to estimate the distance from the specified clause to a successful proof (i.e., an empty clause). Pending clauses, those that occupy the current frontier of the Model Elimination proof search tree, are held in a priority queue that is ordered by the aforementioned heuristic function. At each cycle of the Model Elimination proof search, the clause with the lowest heuristic value (i.e., the one deemed closest to yielding a successful proof) is chosen for processing.

Modified search, as described in (Chang and Lee, 1973), is an option for any of the three basic proof search procedures mentioned above. Model Elimination includes three kinds of inference operation: Factorization, Reduction and Extension. Factorization and Reduction operate on single clauses, while Extension operates on pairs of clauses. Modified search differs from basic search only with respect to the pairs of clauses that participate in the Extension operation. Instead of computing all of the Extension operations possible for a given clause as a single operation, all potential Extension side (or *auxiliary*) clauses are determined and each of the resulting center-clause / side-clause Extension pairs are scheduled independently. This allows for a more refined heuristic to be computed than is possible if only the center clause is examined, because the heuristic function has access to both the center and the side clauses. Factorization is itself optional at the user's discretion. In most cases, modified search produces better performance than basic search.

7. Heuristic Search

Under heuristic proof search, the choice of which clause or clause pair to expand next is governed by the value produced by the *heuristic function*. At any time during the ongoing search for a Model Elimination proof, the frontier of the proof tree is held in a priority queue which is ordered by the value produced by the heuristic function. Each cycle of the proof search consists of removing from the priority queue and then expanding the lowest-valued clause or clause pair (when modified search is in effect).

The primitive heuristic functions, one for single-clause tree nodes and one for extension clause pair nodes, are defined by the user and take the form of a simple linear function combining any of a variety of built-in parameters describing the clause or clause pair. The function is specified in a Lisp-like S-Expression that includes decimal numeric constants and the names of the clause / clause-pair parameter functions following one of the target keywords *node*, *pair*, *center* or *side* combined using any of the four arithmetic operators.

Single Clause Parameter Functions

nLiterals The number of literals, framed or unframed, in the clause

nFLiterals The number of framed literals in the clause

nUFLiterals The number of unframed literals in the clause

nIdentity The number of identity literals in the clause

nIdentityIF The number of unframed identity literals

termDepth The maximum depth of nesting of complex terms

termDepthIF The maximum depth of nesting of complex terms in unframed literals

termVolume The total number of symbols in the clause

nVars The number of distinct variables in the clause

nVarsIF The number of distinct variables in unframed literals

depth The depth of the clause in the proof search tree

Extension Pair Parameter Functions

nLiterals The sum of the number of literals, framed or unframed, in each clause

nFLiterals The sum of the number of framed literals in each clause

nUFLiterals The sum of the number of unframed literals in each clause

nIdentity The sum of the number of identity literals in the each clause

nIdentityIF The sum of the number of unframed identity literals in each clause

termDepth The maximum depth of nesting of complex terms in either clause

termDepthIF The maximum depth of nesting of complex terms in the unframed literals of each clause

termVolume The sum of the total number of symbols in each clause

termVolumeIF The sum of the total number of symbols in unframed literals in each clause

nVars The sum of the number of distinct variables in each clause

nVarsIF The sum of the number of distinct variables in unframed literals of each clause

depth The depth of the center clause in the proof search tree

Here is a histogram of a sample proof, showing the number of clauses examined at each depth of the proof:

Sub-proof 1 Clause-Depth Histogram:

| | | |
|-----|------------|------|
| 0 | [+] | 1 |
| 1 | [+] | 4 |
| 2 | [+] | 57 |
| 3 | [--+] | 678 |
| 4 | [-----+] | 1083 |
| 5 | [-----+] | 1226 |
| 6 | [-----+] | 2597 |
| 7 | [-----+] | 3077 |
| 8 | [-----+] | 4015 |
| 9 | [-----+] | 2841 |
| 10 | [-----+] | 3667 |
| 11 | [-----+] | 2634 |
| 12 | [-----+] | 2473 |
| 13* | [-----+]* | 1267 |
| 14 | [-----+] | 1181 |
| 15 | [--+] | 330 |
| 16 | [+] | 52 |
| 17 | [+] | 18 |

As described above, the clause pair functions are applied to both clauses in the extension clause pair. When desired, these functions can instead be applied to the center or side clause alone. Examples: fewest literals first, (node nLiterals); literal count + maximum term nesting

depth without regard for the kind of proof tree node (single clause or extension clause pair), (+ (node nLiterals) (node termDepth)). Note that framed literals are also known as *A literals* and unframed literals as *B literals*.

8. Brand Transformations

Brand transformations are rewrites of standard clausal forms which contain identities. There is a transformation corresponding to the transitivity of identity, and one to the symmetry of identity. These transformations were introduced in (Brand, 1975); they are further discussed in (Degtyarev and Voronkov, 1999). Apart from Brand’s *flattening* transform, which supplies the substitutivity of identity and is applied unconditionally to problems that include application of the identity predicate, the transitivity and symmetry properties of identity may be supplied either by introducing the pertinent axioms as additional premises or by the application of the corresponding Brand transformation.

9. Martelli and Montanari

Resolution theorem proving and all its derivatives and variants rely heavily on the use of unification between first-order expressions. The efficiency of the unifier bears heavily on the overall speed of the prover. In addition to the classic recursive “mesh” unification algorithm presented in many texts, papers and books, Tau implements the efficient unification algorithm discussed in (Martelli and Montanari, 1977) and in (Martelli and Montanari, 1982). This unification algorithm treats the expressions to be unified, any number of them, as a system of simultaneous equations and solves that system. It is folklore that the Martelli and Montanari algorithm, although providing the best theoretical complexity result, is not always the best algorithm in practice due to the overloading of handling complex data structure. With Tau, we have found that with some problems M&M has substantially improved typical and worst-case complexity by comparison with the classical mesh unification algorithm; i.e., there are problems that generate terms whose structure tips the balance of net run-time cost in favor of M&M. In fact, we use the mesh unifier by default (because measurement confirmed this folklore), but the advantage is small and while we have not confirmed the claim generally, we believe there are problems that produce term structures for which it is true. There are also optimization techniques (low-level programming, not algorithmic) that could yet

close the gap for the majority of problems and make M&M the overall winner. The biggest problem is the large number of very short-lived set data structures produced when executing the M&M algorithm. If we can cut the overhead of their generation and reclamation, we hope to see M&M to surpass the mesh unifier.

10. Stillman's Subsumption Algorithm

Another time-consuming operation for resolution-based theorem provers is computing clause subsumption. In addition to the classic subsumption algorithm described in (Chang and Lee, 1973), Tau implements the better-performing subsumption algorithm invented by Stillman and described in (Gottlob and Leitsch, 1985).

11. Computational Results

The notion of an empirically successful theorem prover is difficult to define, and has a problematic history. As with human provers, it is not clear or uncontroversial exactly what to count as virtue in a prover. Is it: speed, some idea of completeness or comprehensiveness, ease of use, subtlety and originality, or some other factor, or some combination of these? In a practical sense, the idea is one of instrumental virtue, and thus relative to the various conceptions of good use of logic. The TPTP (Thousands of Problems for Theorem Provers) Problem Library, however, is now providing a more uniform basis for assessments; <http://www.cs.miami.edu/tptp/>.

We have begun testing Tau on the TPTP library, which provides a large and challenging repository of benchmarks for provers. TPTP provides tools for translation of TPTP problems into KIF, but due to Tau's preference for FOF over CNF forms, its treatment of identities, and its rewrite strategies, in some cases further aligning of the TPTP tests is necessary before a reasonably full and fair comparison can be made. To date, using the automatic translation tools, we have translated the Geo (geometry) set of problems, with a solution rate of about one-third. For illustration, a proof trace, showing the ME proof steps, is given below of TPTP (Number Theory) NUM016-1, the intended interpretation of which is that there exist infinitely many prime numbers. This is followed by sample statistics of a run of this problem. [Note: some of the run time includes initial invocation of the prover when run in the shell mode.]

Root :

```

{ (not (prime ?X));
  (not (less a ?X));
  (less (factorial_plus_one a) ?X) }
Extend:
{ (not (less ?X-2 ?Y-3));
  (not (less ?Y-3 ?X-2)) }
[ ?Y-3 -> (factorial_plus_one a), ?X -> ?X-2 ]

Clause:
{ (not (prime ?X-2));
  (not (less a ?X-2));
  [(less (factorial_plus_one a) ?X-2)];
  (not (less ?X-2 (factorial_plus_one a))) }
Reduce: [ ?X-2 -> (factorial_plus_one a) ]

Clause:
{ (not (prime (factorial_plus_one a)));
  (not (less a (factorial_plus_one a))) }
Extend:
{ (not (divides ?X-4y (factorial_plus_one ?Y-4z)));
  (less ?Y-4z ?X-4y) }
[ ?Y-4z -> a, ?X-4y -> (factorial_plus_one a) ]

Clause:
{ (not (prime (factorial_plus_one a)));
  [(not (less a (factorial_plus_one a)))];
  (not (divides (factorial_plus_one a)
                (factorial_plus_one a))) }
Extend:
{ (divides ?X-6f ?X-6f) }
[ ?X-6f -> (factorial_plus_one a) ]

Clause:
{ (not (prime (factorial_plus_one a))) }
Extend:
{ (prime ?X-71);
  (divides (prime_divisor ?X-71) ?X-71) }
[ ?X-71 -> (factorial_plus_one a) ]

Clause:
{ [(not (prime (factorial_plus_one a)))];
  (divides (prime_divisor (factorial_plus_one a)
                        (factorial_plus_one a)) ) }

```

Extend:

```
{ (not (divides ?X-77 ?Y-78));
  (not (less ?Y-78 ?X-77)) }
[ ?X-77 -> (prime_divisor (factorial_plus_one a)),
  ?Y-78 -> (factorial_plus_one a) ]
```

Clause:

```
{ [(not (prime (factorial_plus_one a)))]];
  [(divides (prime_divisor (factorial_plus_one a)
                           (factorial_plus_one a)))]];
  (not (less (factorial_plus_one a)
             (prime_divisor (factorial_plus_one a)))) }
```

Extend:

```
{ (not (prime ?X-8c));
  (not (less a ?X-8c));
  (less (factorial_plus_one a) ?X-8c) }
[ ?X-8c -> (prime_divisor (factorial_plus_one a)) ]
```

Clause:

```
{ [(not (prime (factorial_plus_one a)))]];
  [(divides (prime_divisor (factorial_plus_one a)
                           (factorial_plus_one a)))]];
  [(not (less (factorial_plus_one a)
             (prime_divisor (factorial_plus_one a))))];
  (not (prime (prime_divisor (factorial_plus_one a))));
  (not (less a (prime_divisor (factorial_plus_one a)))) }
```

Extend:

```
{ (not (divides ?X-r5 (factorial_plus_one ?Y-r6)));
  (less ?Y-r6 ?X-r5) }
[ ?Y-r6 -> a, ?X-r5 -> (prime_divisor (factorial_plus_one a)) ]
```

Clause:

```
{ [(not (prime (factorial_plus_one a)))]];
  [(divides (prime_divisor (factorial_plus_one a)
                           (factorial_plus_one a)))]];
  [(not (less (factorial_plus_one a)
             (prime_divisor (factorial_plus_one a))))];
  (not (prime (prime_divisor (factorial_plus_one a)))) }
```

Extend:

```
{ (prime ?X-1cx);
  (prime (prime_divisor ?X-1cx)) }
[ ?X-1cx -> (factorial_plus_one a) ]
```

Clause:
 { }

Step Stats: elapsedTime=0.364; cpuTime=0.0; steps=9; roots=1; inputs=12; hornInputs=9; definiteInputs=6; generalInputs=3; factors=0; premiseLiterals=19; rootLiterals=3; generated=349; predicates=3; functions=2; constants=1; skFunctions=0; skConstants=0; expanded=348; derivations=348; factorizations=0; reductions=4; extensions=345; symIDUnif=0; outOrder=1.0; maxQueue=472; nResidual=468; nTooDeep=0; nUnacceptable=9; nXUnacceptable=0; nSubsumed=0; nVacuous=0; proofLength=9

Proof Stats: proved=1; elapsedTime=0.364; cpuTime=0.0; subproofs=1; successes=1; steps=9; roots=1; inputs=12; hornInputs=9; definiteInputs=6; generalInputs=3; factors=0; premiseLiterals=19; rootLiterals=3; generated=349; predicates=3; functions=2; constants=1; skFunctions=0; skConstants=0; expanded=348; derivations=348; factorizations=0; reductions=4; extensions=345; symIDUnif=0; outOrder=1.0; maxQueue=472; nResidual=468; nTooDeep=0; nUnacceptable=9; nXUnacceptable=0; nSubsumed=0; nVacuous=0; proofLength=9

Elapsed: 0m2s; User: 0m1.7s; System: 0m0.1s

Our work with TPTP has just begun. We will not be so rash as to claim that Tau surpasses any of the well-known provers, such as Otter, Setheo, Meteor, Protein, or Snark, but initial tests indicate that Tau behaves respectably on a variety of TPTP problems. However, that may be, we shall give below more sample Tau statistics, after a discussion of some various types of problems.

12. Logic Theorems

There are at present 78 logical theorems available for testing in the Tau browser, derived from (Kalish and Montague, 1964) and (Montague, Kalish, and Mar, 1980).

A simple theorem which caused an incompleteness problem for some older resolution style provers was posed in (Inoue, 1992).

```
(=> (and
      (forall ?X (or (not (Q ?X)) (P ?X) (P a))
                    (not (P B))
                    (Q B) )
      (P a))
```

Tau handles such problems easily; a test run can be made at

<http://hsinfosystems.com/taujay/doc/samples/tests/In001.jsp>.

The ‘Los theorem’ was considered a surprise in the early days of theorem proving, as no one seems to have thought it intuitive, and it was discovered first by a theorem prover. The theorem is:

```
(=> (and
(forall (?X ?Y ?Z) (=> and (P ?X ?Y) (P ?Y ?Z)) (P ?X ?Z))
(forall (?X ?Y ?Z)(=> (and (Q ?X ?Y) (Q ?Y ?Z)) (Q ?X ?Z))))
(forall (?X ?Y)(=> (Q ?X ?Y) (Q ?Y ?X)))
(forall (?X ?Y) (or (P ?X ?Y)(Q ?X ?Y))) )
(or (forall (?X ?Y) (P ?X ?Y)) (forall (?X ?Y)(Q ?X ?Y))))
```

Tau also handles this problem easily; a test run can be seen at

<http://hsinfosystems.com/taujay/doc/samples/tests/Los001.jsp>.

13. Identity Problems

As you have seen, Tau solves a variety of identity tests. However, there are two theorems involving identity from (Montague, Kalish, and Mar, 1980) which Tau has not yet been able to prove (except in simplified form) are T328 and T329:

T328

```
(forall (?A ?B ?C)
(=>
  (and (=> (exists ?Z (forall ?X (<=> (f ?X) (= ?X ?Z))))
        (f ?A))
    (=> (not (exists ?Z (forall ?X (<=> (f ?X) (= ?X ?Z))))
        (= ?A ?C))
    (=> (exists ?Z (forall ?Y (<=> (f ?Y) (= ?Y ?Z))))
        (f ?B))
    (=> (not (exists ?Z (forall ?Y (<=> (f ?Y) (= ?Y ?Z))))
        (= ?B ?C)) )
  (= ?A ?B)))
```

T329

```
(forall (?A ?B ?C)
(=>
  (and (=> (exists ?Y (forall ?X (<=> (f ?X) (= ?X ?Y))))
        (f ?A))
    (=> (not (exists ?Y (forall ?X (<=> (f ?X) (= ?X ?Y))))))
```

```

(= ?A ?C))
(=> (exists ?Y (forall ?X (<=> (g ?X) (= ?X ?Y))))
      (g ?B))
(=> (not (exists ?Y (forall ?X (<=> (g ?X) (= ?X ?Y))))))
      (= ?B ?C))
(forall ?X (<=> (f ?X) (g ?X))) )
(= ?A ?B))

```

There are particularly interesting problems for intelligent automation, because a human proof would naturally employ (after instantiation) the lemmas which the premises intuitively reveal, as in .e.g. T328,

```

(and (=> phi (f a))
      (=> (not phi) a=c)
      (=> phi (f b)))
(=> (not phi) (= b c))),

```

together with the knowledge that \neg (or phi (not phi)), in order to reach the conclusion expeditiously. Tau's ME strategy, however, becomes swamped with these problems when run under its normal limits. This is a "lack of goal orientation" – a form of tunnel-vision – which a trained human intelligence easily overcomes. The adoption of a named subformula / rewriting strategy, together with an intelligent lemmaization scheme is clearly called for in such cases, and we are now working on adding lemmaization to Tau⁴.

See (Bachmair and Ganzinger, 1998) for more discussion of identity handling in theorem provers.

14. Theory of a Successor, Presburger and Peano Arithmetic

We denote the theory of a successor, Succ. It is a subtheory of Peano Arithmetic, expressed in KIF by:

```

(forall ?X (not (= 0 (succ ?X))))
(forall ?X (forall ?Y (=> (= (succ ?X) (succ ?Y)) (= ?X ?Y))))

```

Tau tests in the theory Succ are at:

<http://www.hsinfosystems.com/taujay/doc/samples/testsJSP.html#Succ>

⁴ There are problems in which the full complement of identity inferences is not needed; i.e., transitivity or symmetry, e.g. When they are recognized by the user, prover performance can be enhanced by not attempting the irrelevant inferences.

Note that to prove even the simple $(\text{forall } ?X (\text{not } (= ?X (\text{succ } ?X))))$ requires the use of mathematical induction, discussed in the next section.

Presburger Arithmetic axioms (the theory PrA), is also a sub-theory of Peano arithmetic, lacking multiplication; it is decidable, but already has difficult computational complexity. It is expressed in KIF by the axioms:

```
(forall ?X (not (= 0 (succ ?X))))
(forall ?X (forall ?Y (=> (= (succ ?X) (succ ?Y)) (= ?X ?Y))))
(forall ?X (= (+ ?X 0) ?X))
(forall ?X (forall ?Y (= (+ ?X (succ ?Y)) (succ (+ ?X ?Y)))))
```

Tau tests in the theory PrA are at:

<http://www.hsinfosystems.com/taujay/doc/samples/testsJSP.html#Presburger>

Peano Arithmetic adds the multiplication axioms:

```
(forall ?X (forall ?Y (= (* 0 ?X) 0)))
(forall ?X (forall ?Y (= (* ?X (succ ?Y)) (+ (* ?X ?Y) ?X))))
```

Tau tests in simple Peano Arithmetic and more tests using induction are at:

<http://www.hsinfosystems.com/taujay/doc/samples/testsJSP.html>

Combinations and reductions of these sets of axioms (PA, PrA, and Succ), together with the introduction of definitions give us other theories, which we will denote below, while presenting some sample axioms of each theory. Each of these theories may also be extended by the use of induction. Some of the theories involve only ‘succ’, some involve addition and multiplication also. There are various courses possible with extension by definition and with axiomatization by primitives. For example, ‘<’ may be defined axiomatically in an extension of the theory Succ; alternatively, it may be defined in PrA. Tau also has sample problems which are extensions of PrA and PA: these involve various definitions of the predicates ‘even’, ‘odd’, ‘=<’, and others.

See (Enderton, 2001) for further discussion.

15. Mathematical Induction

Mathematical induction may be used (see the checkbox on the Tau website) in Tau: (all instances of)

```
(=> (and (F 0) (forall ?X (=> (F ?X) (F (succ ?X))))
      (forall ?X (F ?X))),
```

where F represents any formula with one free variable.

Note that some proofs in Peano arithmetic and Presburger arithmetic require that Tau apply mathematical induction or use results which haven been previously proved by induction, while others do not. It is a good exercise for the user or student to determine what the dependencies are.

A sample proof using induction may be run at:

<http://hsinfosystems.com/taujay/doc/samples/PALT.web/PALT02Ind.html>.

You will note that in this proof a logical axiom for an identity substitution is also supplied as a premise. We have found that, in some instances, the Brand transformations are speeded up thereby, when heuristic search is also used.

Below is the ME proof as it appears in its proof trace file.

Proof Chain -- 5 steps:

Root:

```
{ (not (< (+ (succ $MVI-1) $UI-1) (* (succ $MVI-1) $UI-1))) }
```

Extend:

```
{ (not (< ?F11-17p ?Y-17q));
    (< ?X-17r ?Y-17q);
    (not (= ?F11-17p (succ ?X-17r))) }
[ ?X-17r -> (+ (succ $MVI-1) $UI-1), ?Y-17q ->
    (* (succ $MVI-1) $UI-1) ]
```

Clause:

```
{ [(not (< (+ (succ $MVI-1) $UI-1) (* (succ $MVI-1) $UI-1)))]];
    (not (< ?F11-17p (* (succ $MVI-1) $UI-1)));
    (not (= ?F11-17p (succ (+ (succ $MVI-1) $UI-1)))) }
```

Extend:

```
{ (= ?Eq-2r0 ?Eq-2r0) }
[ ?F11-17p -> (succ (+ (succ $MVI-1) $UI-1)), ?Eq-2r0 ->
    (succ (+ (succ $MVI-1) $UI-1)) ]
```

Clause:

```
{ [(not (< (+ (succ $MVI-1) $UI-1)
    (* (succ $MVI-1) $UI-1)))]];
```

```

      (not (< (succ (+ (succ $MVI-1) $UI-1))
             (* (succ $MVI-1) $UI-1))) }
Extend:
{ (not (< (succ 0) ?X-9tt));
  (not (< (succ 0) ?Y-9tu));
  (< (succ (+ ?X-9tt ?Y-9tu)) (* ?X-9tt ?Y-9tu)) }
[ ?X-9tt -> (succ $MVI-1), ?Y-9tu -> $UI-1 ]

Clause:
{ [(not (< (+ (succ $MVI-1) $UI-1) (* (succ $MVI-1) $UI-1)))]];
  [(not (< (succ (+ (succ $MVI-1) $UI-1))
             (* (succ $MVI-1) $UI-1)))]];
  (not (< (succ 0) (succ $MVI-1)))]];
  (not (< (succ 0) $UI-1)) }
Extend:
{ (< (succ 0) $UI-1) }

[ <empty> ]

Base step:

Clause:
{ [(not (< (+ (succ $MVI-1) $UI-1) (* (succ $MVI-1) $UI-1)))]];
  [(not (< (succ (+ (succ $MVI-1) $UI-1))
             (* (succ $MVI-1) $UI-1)))]];
  (not (< (succ 0) (succ $MVI-1)))] }

Extend:
{ (< (succ 0) (succ $MVI-1)) }
1
[ <empty> ]

Clause:
{ }

```

For the logical basis of this application, see, e.g., (Bundy, 2001).

16. Graph Theory

We have axiomatized in KIF several problems in finite graph theory. A sample can be seen at

<http://hsinfosystems.com/taujay/doc/samples/GMGT.web/GM08d.html>.

These problems are over very small domains, so the universal quantifiers are equivalent to finite conjunctions of atomic sentences, and the existential quantifiers are equivalent to finite disjunction of atomic sentences. Accordingly, we have taken advantage of these equivalences to introduce corresponding proof rewrites into Tau, for such problems.

17. Sample Statistics

Through the command line interface, Proof statistics can be displayed for each submitted problem, as previously shown, or for each sub-proof in a submitted problem, or collectively for batch runs of multiple problems. The sample statistics shown next are for a batch test suite of 233 problems, many taken from . Note that the inferences counted are complex Model Elimination inferences, including factorizations, subsumptions, reductions and extensions, not simply resolutions. Tau's speed of inference upon these problems runs from a few hundred per second up to tens of thousands per second, depending upon the logical complexity of the clausal forms, and upon the specific search mechanism invoked. The overall average for this test suite was about a thousand inferences per second.

Proved: The number of problems successfully proved out of the total number of problems submitted followed by the total number of inference steps in the resulting proofs and lastly the average length of the proofs obtained over the entire test run.

Roots: The number of root clauses used to prime the Model Elimination search tree. This number is greater than or equal to the number of problems attempted because each FOF in the conclusion in general produces multiple clauses, each of which must be used as a root for a Model Elimination proof search.

Inputs: The total number of clauses submitted, whether they derive from conclusions to prove or from the premises.

Horn / Definite / Other: Histogram of input clauses according to type. (Horn clauses have at most one positive literal, Definite clauses have exactly one literal.)

Root Literals: The number of literals in the clauses used as Model Elimination proof search tree roots.

Side Literals: The number of literals in clauses used as Model Elimination side or auxiliary clauses.

Generated: The number of clauses produced before the proof attempt concluded, whether successfully or not.

Expanded: The number of clauses processed by application of the Model Elimination inference operations.

Derivations: The number of successful (new clause-producing) applications of Model Elimination inference operations.

Factorizations: The number of successful applications of the Model Elimination factorization operation.

Reductions: The number of successful applications of the Model Elimination reduction operation.

Extensions: The number of successful applications of the Model Elimination extension operation.

Totals:

| | |
|--------------------------|---|
| Proved: | 216 of 233; 1755 inferences; 8.12 Inferences/Proof |
| Elapsed Time: | 708.54 sec; |
| Roots: | 977; |
| Inputs: | 8858; |
| Horn / Definite / Other: | 8101 / 6329 / 483; |
| Root Literals: | 4001; |
| Side Literals: | 30841; |
| Generated: | 320434; |
| Expanded: | 276289; |
| Derivations: | 319457; |
| Factorizations: | 1971; |
| Reductions: | 22414; |
| Extensions: | 295511; |
| Out-order: | 1.16; |
| Residual: | 1651309; |
| Too deep: | 0; |
| Unacceptable: | 30051; |
| XUnacceptable: | 1815; |
| Subsumed: | 106; |
| Vacuous: | 7542; |

18. Disproofs

Tau can in certain cases disprove theorems. A few of our tests are deliberate disproofs, as a check on soundness. These tests are positively proved invalid and are noted as such when they are run. The soundness of such disproof depends upon the prover's noticing in simple cases that it has exhausted all the possibilities for obtaining a proof by contradiction. When the Model Elimination proof tree is finite and we can build it fully without deriving the empty clause / chain, then we have disproved the conjecture. Simple and not always possible, but when it is, that's all there is to it when these conditions are recognized. A refinement which Tau uses in certain cases is to notice that a theorem has only a very small number of finite models, up to isomorphism, via recognition of identity constraints. A typical disproof (from (Thomas, 1977)) may be run at <http://hsinfosystems.com/taujay/doc/samples/tests/Th267-gA.jsp>.

19. Next Extensions of Tau

“De l’audace, encore de l’audace, et toujours de l’audace!” - Danton

Tau is an ongoing project and its authors plan to follow several paths, including: implementation of MathML notation; implementation of notation for the treatment of variable-binding operators in (Montague, Kalish, and Mar, 1980) (see Chapters X and XI), and of theorem schemata; persistent KB storage across browser sessions; formal language translation facilities; simplified English translation facilities; translation between systems of logic (i.e., intuitionistic and FOL); formal definitions (work partially implemented); implementing the use of metalogical expressions in deduction, as axiom schemata and, particularly, in forming inductive axioms (work now underway); implementation of HOL and the use of theorem schemata; implementation of sequent style proofs (work now underway) [For an introduction to, and discussion of sequent calculi, see, e.g., (Robinson and Voronkov (Eds.), 2002) or (Buss, 1998)]; further TPTP testing (work now underway); graphical notations.

We hope that users will find Tau stimulating.

Notes

¹ KIF may be viewed as a 'segregated dialect' (sublanguage) of CLIF, the Common Logic Interchange Format. The Common Logic effort is about addressing the Tower of Babel problem. Computers employ a staggering variety of programming languages, methods, and logics to calculate and to exchange information. Can the resulting confusions and incompatibilities be reduced? Common Logic is a logic framework intended for information exchange and transmission. The framework allows for a variety of different syntactic forms, called dialects, all expressible within a common XML-based syntax and all sharing a single semantics.

CLIF (1) contains notation for a first order logic language for knowledge interchange, (2) provides a core semantic framework for logic, and (3) provides the basis for a set of syntactic forms (dialects) all sharing a common semantics. In this, Common Logic is a logic framework intended for information exchange and transmission on computer networks. The CLIF language allows a variety of different syntactic forms, called dialects, all expressible within a common XML-based syntax and all sharing a single denotational semantics. Novel features of CLIF include a "wild and woolly" signature free syntax which permits 'higher-order' constructions such as optional quantification over classes or relations (sequence variable quantification) while permitting a first-order model theory. Clif also has a semantics which allows theories to describe intensional entities such as Common Logic properties. Clif fixes the meanings of conventions in widespread use, such as numerals to denote integers and quotation marks to denote character strings, and has provision for the use of datatypes and for naming, importing and transmitting content on the World Wide Web using XML.

The SCL ad-hoc working group (formed Dec 2002), includes: Pat Hayes IHMC, USA, Chris Menzel Texas A&M U., USA, John Sowa VivoMind, USA, Tanel Tammet U. Goteborg, Sweden, Bill Andersen OntologyWorks, USA, Murray Altheim Open University, UK, Harry Delugach U. Alabama Huntsville, USA, and Michael Gruninger NIST, USA (and Canada)

The ISO effort towards an international standard for Common Logic began in June 2003 with the approval of a New Work Item (NP) for Common Logic. This project was assigned to WG2 (Metadata) under SC32 (Data Interchange) of ISO/IEC JTC1. In October 2003, Harry Delugach was designated the editor for the standard, which may be found at: <http://philebus.tamu.edu/cl/>.

References

- Apache Tomcat servlet container, a product of the Apache Jakarta Project;
<http://jakarta.apache.org/tomcat/index.html>.
- Baaz, Matthias, Uwe and Leitsch, "Normal Form Transformations", in Robinson and Voronkov (Eds), Handbook of Automated Reasoning (2 vols), MIT Press, Cambridge, 2002.
- Bachmair, Leo and Harald Ganzinger. "Equational reasoning in saturation based theorem proving", in Wolfgang Bibel and Peter H. Schmidt, editors, Automated Deduction: A Basis for Applications. Volume I, Foundations: Calculi and Methods, pages 353-398. Kluwer Academic Publishers, Dordrecht, 1998.
- Brand, Daniel. "Proving theorems with the modification method", SIAM Journal on Computing, 4(4):412430, 1975.

- Bundy, Alan. "The Automation of Proof by Mathematical Induction", Handbook of Automated Reasoning 2001: 845-911
- Buss, Samuel R. (Ed.), Handbook of Proof Theory, Elsevier, New York, NY, 1998
- Chang, C.L., and R. C. T. Lee, Symbolic Logic and Mechanical Theorem Proving, Academic Press, New York, 1973.
- Common Logic Standard, an ISO effort towards an international standard for Common Logic <http://philebus.tamu.edu/cl/>
- Degtyarev, Anatoli and Andrei Voronkov, "Equality reasoning in sequent-based calculi". In Alan Robinson and Andrei Voronkov, editors. Handbook of Automated Reasoning, Elsevier Science Publishers, 1999.
- Enderton, Herbert B., A Mathematical Introduction to Logic, 2nd Ed., Harcourt Academic Press, New York, 2001
- Gottlob, G. and L. Leitsch. "On the efficiency of subsumption algorithms", Journal of the ACM, Volume 32, Issue 2, April 1985, pp. 280 - 295; <http://doi.acm.org/10.1145/3149.214118>
- Hooker, J.N. "Solving the incremental satisfiability problem", Journal of Logic Programming 15 (1993) 177-186.
- Hooker, J.N. "New methods for computing inferences in first order logic", Annals of Operations Research (1993) 479-492.
- Inoue, K. "Linear resolution for consequence finding", Artificial Intelligence, 56:301-353, 1992.
- Loveland, D.W. "Mechanical theorem-proving by model elimination," Journal of the ACM, Volume 15, Issue 2, April 1968, pp. 236-251; <http://doi.acm.org/10.1145/321450.321456>, ACM DOI bookmark.
- Loveland, D.W. "A simplified format for the model elimination procedure", Journal of the ACM, Vol. 15, Issue 2 1969, pp. 349-363; <http://doi.acm.org/10.1145/321526.321527>, ACM DOI bookmark.
- Loveland, D.W. Automated Theorem Proving: A Logical Basis, North-Holland, Amsterdam, 1978.
- Martelli, Alberto and Ugo Montanari, "An Efficient Unification Algorithm", ACM Trans. Program. Lang. Syst. 4(2): 258-282 (1982).
- Martelli, A., and Montanari, U. "Theorem proving with structure sharing and efficient unification", Internal Rep. S-77-7, Ist. di Scienze della Informazione, University of Pisa, Pisa, Italy; also in Proceedings of the 5th International Joint Conference on Artificial Intelligence, Boston, 1977, p. 543.
- Montague, Richard and Donald Kalish, Logic, Techniques of Formal Reasoning, New York, Harcourt, Brace and World, Inc., 1964.
- Montague, Richard, Kalish, Donald, and Mar, Gary (Ed. Robert Fogelin), Logic: Techniques of Formal Reasoning, Harcourt Brace, New York, 1980.
- Nonnengart, Andreas and Christoph Weidenbach, "Computing Small Clause Normal Forms", In Alan Robinson and Andrei Voronkov, editors, Handbook of Automated Reasoning. Elsevier Science Publishers, 1999.
- Robinson, J.A. "A machine-oriented logic based on the resolution principle", Jour. Assoc. for Comput. Mach., 1965, 23-41.
- Robinson, J.A., "Computational logic: The unification computation", In Machine Intelligence, vol. 6, B. Meltzer and D. Michie (Eds.). Edinburgh Univ. Press, Edinburgh, Scotland, 1971, pp. 63-72.
- Robinson and Voronkov (Eds), Handbook of Automated Reasoning (2 vols), MIT Press, Cambridge, 2002.
- Sun Java <http://java.sun.com/>.

- Stickel, M.E., "A Prolog technology theorem prover", *New Generation Computing*, 1984, 371-383.
- Sutcliffe, Geoff and Suttner, Christian, "The TPTP (Thousands of Problems for Theorem Provers) Problem Library", <http://www.cs.miami.edu/ tptp/>.
- Thomas, James A., "Symbolic Logic", Merrill, Columbus, Ohio, 1977.

Address for Offprints:

KLUWER ACADEMIC PUBLISHERS PrePress Department,
P.O. Box 17, 3300 AA Dordrecht, The Netherlands
e-mail: TEXHELP@WKAP.NL
Fax: +31 78 6392500