

# Axiomatizing Einstein's Mice: Solving Logic Problems with ATP

Jay Halcomb<sup>1</sup>   Randall Schulz<sup>2</sup>

<sup>1</sup>**H&S Information Systems**  
Belmont, CA

<sup>2</sup>**H&S Information Systems**  
Belmont, CA

**Stanford Logic Seminars, Abstract**



# Outline

- 1 **Axiomatizing Einstein's Mice**
  - Tau and the KIF / CLIF Languages
  - The Logical Theory of Tau
- 2 Algorithms
  - Resolution, Model Elimination and Search
  - Brand Transformations
  - Martelli and Montanari, Stillman Subsumption
- 3 Theories and Computational Results
  - Logic Theorems, Identity Problems, and Theories
  - Computational Results

# Outline

- 1 **Axiomatizing Einstein's Mice**
  - Tau and the KIF / CLIF Languages
  - The Logical Theory of Tau
- 2 **Algorithms**
  - Resolution, Model Elimination and Search
  - Brand Transformations
  - Martelli and Montanari, Stillman Subsumption
- 3 **Theories and Computational Results**
  - Logic Theorems, Identity Problems, and Theories
  - Computational Results

# Outline

- 1 **Axiomatizing Einstein's Mice**
  - Tau and the KIF / CLIF Languages
  - The Logical Theory of Tau
- 2 **Algorithms**
  - Resolution, Model Elimination and Search
  - Brand Transformations
  - Martelli and Montanari, Stillman Subsumption
- 3 **Theories and Computational Results**
  - Logic Theorems, Identity Problems, and Theories
  - Computational Results

# Abstract

Despite well-known difficulties, the quest for an *Ars Universalis* does not die. 'Einstein's Mice' (although of very doubtful attribution) is one of a class of traditional logic puzzles. We use it to illustrate techniques in automated theorem proving with Tau, a practical (web-deployed) and extensible hybrid theorem prover for first-order predicate calculus with identity. We use Tau to solve 'Einstein's Mice' and similar logic puzzles, and to represent mathematical theories. We conclude with a plea for a 'common logic' (FOL-based) approach to ATP, and report on a potential framework - an extensible syntax and semantics for such work.

# The Problem

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

**Source:** <http://www.weitz.de/einstein.html>

## Challenge: how do we solve the problem?

- **A procedural solution? Using classical programming?**  
The answers: `(#(MICKEY GOUDA SEINFELD) #(MINNY BRIE SIMPSONS) #(MIGHTY EMMENTAL ER))`
- Here is the clever and quick programming solution by Dr. Weitz which solves the mice problem in Lisp: `riddle.lisp`
- **Sample times ranged 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 much quicker times - by a factor of 10.)
- **Or do we find a logical/inferential solution (using classical logic)?**

## Challenge: how do we solve the problem?

- **A procedural solution? Using classical programming?**  
The answers:  `#(MICKEY GOUDA SEINFELD) #(MINNY BRIE SIMPSONS) #(MIGHTY EMMENTAL ER)`
- Here is the clever and quick programming solution by Dr. Weitz which solves the mice problem in Lisp: **riddle.lisp**
- **Sample times ranged 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 much quicker times - by a factor of 10.)
- **Or do we find a logical/inferential solution (using classical logic)?**

## Challenge: how do we solve the problem?

- **A procedural solution? Using classical programming?**  
The answers: `(#(MICKEY GOUDA SEINFELD) #(MINNY BRIE SIMPSONS) #(MIGHTY EMMENTAL ER))`
- Here is the clever and quick programming solution by Dr. Weitz which solves the mice problem in Lisp: **riddle.lisp**
- **Sample times ranged 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 much quicker times - by a factor of 10.)
- Or do we find a logical/inferential solution (using classical logic)?

## Challenge: how do we solve the problem?

- **A procedural solution? Using classical programming?**  
The answers: `(#(MICKEY GOUDA SEINFELD) #(MINNY BRIE SIMPSONS) #(MIGHTY EMMENTAL ER))`
- Here is the clever and quick programming solution by Dr. Weitz which solves the mice problem in Lisp: **riddle.lisp**
- **Sample times ranged 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 much quicker times - by a factor of 10.)
- **Or do we find a logical/inferential solution (using classical logic)?**

# An inferential solution

Here is an inferential solution:

**EinsteinsMice26R.html**

Conclusion: Proved creating 4,939 clauses taking 1.993 seconds.

## However...

**All is not entirely well in prover-land.** Inspection of various axiomatizations of the Mice problem and alternative proof attempts shows that some simpler axiomatization (lacking certain ground cases used in the previous proof), fail - because **the prover is unable to produce the ground instances which drive a successful proof quickly enough.**

**This defect is remediable**, though, by two variations of 'lemmaization'; i.e., by instructing the prover to find the significant ground instances more quickly. This can be done either at the level of the internal proof structures (clausal forms), thus **extending the Model Elimination method**, or at the proof strategy level, by **judicious instantiation**. We haven't yet implemented either strategy in Tau, but it is a definite next step for the prover.

# Initial Tau Screen

<http://hsinfosystems.com/taujay/>

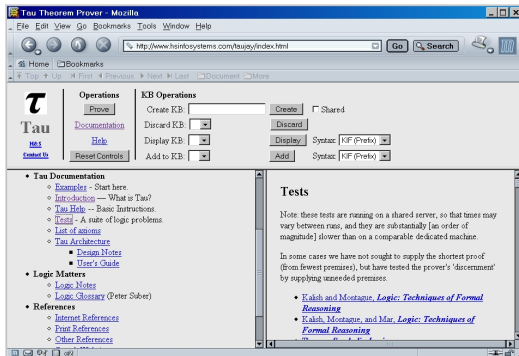


Figure: Initial Tau Screen

# How Tau Works

- Hybrid theorem prover for FOPC with identity
- Brand's Modification Method to implement identity
- Implements mathematical induction
- Rule-based problem rewriting and Model Elimination
- On the web, flexible and extensible
- Accepts the FOL KIF and CLIF languages, or infix FOPC
- Implemented in Java
- Multiple user interfaces
- User-configurable heuristic search
- Web interface or command-line interface

# How Tau Works

- Hybrid theorem prover for FOPC with identity
- Brand's Modification Method to implement identity
- Implements mathematical induction
- Rule-based problem rewriting and Model Elimination
- On the web, flexible and extensible
- Accepts the FOL KIF and CLIF languages, or infix FOPC
- Implemented in Java
- Multiple user interfaces
- User-configurable heuristic search
- Web interface or command-line interface

# How Tau Works

- Hybrid theorem prover for FOPC with identity
- Brand's Modification Method to implement identity
- Implements mathematical induction
- Rule-based problem rewriting and Model Elimination
- On the web, flexible and extensible
- Accepts the FOL KIF and CLIF languages, or infix FOPC
- Implemented in Java
- Multiple user interfaces
- User-configurable heuristic search
- Web interface or command-line interface

# How Tau Works

- Hybrid theorem prover for FOPC with identity
- Brand's Modification Method to implement identity
- Implements mathematical induction
- Rule-based problem rewriting and Model Elimination
- On the web, flexible and extensible
- Accepts the FOL KIF and CLIF languages, or infix FOPC
- Implemented in Java
- Multiple user interfaces
- User-configurable heuristic search
- Web interface or command-line interface

# How Tau Works

- Hybrid theorem prover for FOPC with identity
- Brand's Modification Method to implement identity
- Implements mathematical induction
- Rule-based problem rewriting and Model Elimination
- On the web, flexible and extensible
- Accepts the FOL KIF and CLIF languages, or infix FOPC
- Implemented in Java
- Multiple user interfaces
- User-configurable heuristic search
- Web interface or command-line interface

# How Tau Works

- Hybrid theorem prover for FOPC with identity
- Brand's Modification Method to implement identity
- Implements mathematical induction
- Rule-based problem rewriting and Model Elimination
- On the web, flexible and extensible
- Accepts the FOL KIF and CLIF languages, or infix FOPC
- Implemented in Java
- Multiple user interfaces
- User-configurable heuristic search
- Web interface or command-line interface

# How Tau Works

- Hybrid theorem prover for FOPC with identity
- Brand's Modification Method to implement identity
- Implements mathematical induction
- Rule-based problem rewriting and Model Elimination
- On the web, flexible and extensible
- Accepts the FOL KIF and CLIF languages, or infix FOPC
- Implemented in Java
- Multiple user interfaces
- User-configurable heuristic search
- Web interface or command-line interface

# How Tau Works

- Hybrid theorem prover for FOPC with identity
- Brand's Modification Method to implement identity
- Implements mathematical induction
- Rule-based problem rewriting and Model Elimination
- On the web, flexible and extensible
- Accepts the FOL KIF and CLIF languages, or infix FOPC
- Implemented in Java
- Multiple user interfaces
- User-configurable heuristic search
- Web interface or command-line interface

# How Tau Works

- Hybrid theorem prover for FOPC with identity
- Brand's Modification Method to implement identity
- Implements mathematical induction
- Rule-based problem rewriting and Model Elimination
- On the web, flexible and extensible
- Accepts the FOL KIF and CLIF languages, or infix FOPC
- Implemented in Java
- Multiple user interfaces
- User-configurable heuristic search
- Web interface or command-line interface

# How Tau Works

- Hybrid theorem prover for FOPC with identity
- Brand's Modification Method to implement identity
- Implements mathematical induction
- Rule-based problem rewriting and Model Elimination
- On the web, flexible and extensible
- Accepts the FOL KIF and CLIF languages, or infix FOPC
- Implemented in Java
- Multiple user interfaces
- User-configurable heuristic search
- Web interface or command-line interface

## Semantic Tableaux

Consider the remarks of [Bachmair and Ganzinger, 1998] in “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.

# What you can presently do with Tau

- Prove theorems of logic
- Test the FOL validity of formal arguments (derive a conclusion from premises)
- Normalize formulas
- Construct a formal FOPC theory. Examples constructed include: theorems in **Presburger** and **Peano arithmetic**, with and without **mathematical induction**; **commutative ordered fields**; **graph theory**.
- We emphasize the **logical soundness of the proof method** and the extensibility of the software. Via the command line interface we have flexibility in choosing proof strategies, and control over the presentations and annotations.

# What you can presently do with Tau

- Prove theorems of logic
- Test the FOL validity of formal arguments (derive a conclusion from premises)
- Normalize formulas
- Construct a formal FOPC theory. Examples constructed include: theorems in **Presburger** and **Peano arithmetic**, with and without **mathematical induction**; **commutative ordered fields**; **graph theory**.
- We emphasize the **logical soundness of the proof method** and the extensibility of the software. Via the command line interface we have flexibility in choosing proof strategies, and control over the presentations and annotations.

# What you can presently do with Tau

- Prove theorems of logic
- Test the FOL validity of formal arguments (derive a conclusion from premises)
- Normalize formulas
- Construct a formal FOPC theory. Examples constructed include: theorems in **Presburger** and **Peano arithmetic**, with and without **mathematical induction**; **commutative ordered fields**; **graph theory**.
- We emphasize the **logical soundness of the proof method** and the extensibility of the software. Via the command line interface we have flexibility in choosing proof strategies, and control over the presentations and annotations.

## What you can presently do with Tau

- Prove theorems of logic
- Test the FOL validity of formal arguments (derive a conclusion from premises)
- Normalize formulas
- Construct a formal FOPC theory. Examples constructed include: theorems in **Presburger** and **Peano arithmetic**, with and without **mathematical induction**; **commutative ordered fields**; **graph theory**.
- We emphasize the **logical soundness of the proof method** and the extensibility of the software. Via the command line interface we have flexibility in choosing proof strategies, and control over the presentations and annotations.

## What you can presently do with Tau

- Prove theorems of logic
- Test the FOL validity of formal arguments (derive a conclusion from premises)
- Normalize formulas
- Construct a formal FOPC theory. Examples constructed include: theorems in **Presburger** and **Peano arithmetic**, with and without **mathematical induction**; **commutative ordered fields**; **graph theory**.
- We emphasize the **logical soundness of the proof method** and the extensibility of the software. Via the command line interface we have flexibility in choosing proof strategies, and control over the presentations and annotations.

# The KIF / CLIF Languages

## CLIF Notes

KIF (Knowledge Interchange Format)[Genesereth et al, 1995] is essentially a parenthesized prefix version of common first-order logical notation, which largely emanates from environs of Stanford University; KIF is also a part of the ISO (International Organization for Standardization) [Common Logic Standard] effort, in the form of Common Logic Interchange Format (CLIF), which at its core is the FOL part of KIF. Being a prefix form, the language is efficient for many computer applications and for that reason we adopted KIF/ core CLIF 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 proper mathematical and logical symbology such as LaTeX or MathML.

## A Typical KIF/CLIF Sentence

Syntactically core CLIF \*is\* FOL KIF, modulo variable indentifiers. Typical KIF / CLIF 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)))))
```

# The Logical Theory of Tau

- **reductio ad absurdum, or contradiction testing**
- Skolemization
- normalization (see, e.g., [Baaz et al, 2002], and also see [Nonnengart 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.

# The Logical Theory of Tau

- **reductio ad absurdum, or contradiction testing**
- **Skolemization**
- normalization (see, e.g., [Baaz et al, 2002], and also see [Nonnengart 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.

# The Logical Theory of Tau

- reductio ad absurdum, or contradiction testing
- Skolemization
- normalization (see, e.g., [Baaz et al, 2002], and also see [Nonnengart 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.

# The Logical Theory of Tau

- reductio ad absurdum, or contradiction testing
- Skolemization
- normalization (see, e.g., [Baaz et al, 2002], and also see [Nonnengart 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.

# The Logical Theory of Tau

- reductio ad absurdum, or contradiction testing
- Skolemization
- normalization (see, e.g., [Baaz et al, 2002], and also see [Nonnengart 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.

## Rewriting and Indirect Proof

- Tau is essentially an **indirect prover** that proves formulas by establishing the mutual unsatisfiability of the clauses resulting from the Skolemized form of the original input formulas, with the conclusion to prove negated.
- Before Skolemization, clausalization and the application of Model Elimination, the conclusion is subject to a **process of rewriting** that replaces the original conclusion with 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 (or fail) the proof in the shortest time.

## Rewriting and Indirect Proof

- Tau is essentially an **indirect prover** that proves formulas by establishing the mutual unsatisfiability of the clauses resulting from the Skolemized form of the original input formulas, with the conclusion to prove negated.
- Before Skolemization, clausalization and the application of Model Elimination, the conclusion is subject to a **process of rewriting** that replaces the original conclusion with 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 (or fail) the proof in the shortest time.

## Rewriting and Indirect Proof

- Tau is essentially an **indirect prover** that proves formulas by establishing the mutual unsatisfiability of the clauses resulting from the Skolemized form of the original input formulas, with the conclusion to prove negated.
- Before Skolemization, clausalization and the application of Model Elimination, the conclusion is subject to a **process of rewriting** that replaces the original conclusion with 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 (or fail) the proof in the shortest time.

# 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.

# What Tau does internally: a typical ME Extension

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))) ] }
```

# Model Elimination

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 recently revived with Stickel's work on the theorem prover PTPP, e.g. [Stickel, 1984]. Tau uses a version of Model Elimination with refinements to handle certain completeness issues which may arise from an heedless 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).

# Proof Search

- 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
  - Modified search

# Proof Search

- 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
  - Modified search

# Proof Search

- 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
  - Modified search

# Proof Search

- 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
  - Modified search

# Proof Search

- 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
  - Modified search

# Proof Search

- **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.
- **Modified search** allows exploring all the potentially acceptable pairs of an ME extension and scheduling them for independent exploration.

# Proof Search

- **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.
- **Modified search** allows exploring all the potentially acceptable pairs of an ME extension and scheduling them for independent exploration.

# Proof Search

- **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.
- **Modified search** allows exploring all the potentially acceptable pairs of an ME extension and scheduling them for independent exploration.

# Heuristic Search

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.

# 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. Another transformation (flattening), handles substitution. 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.

# 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. Another transformation (flattening), handles substitution. 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.

# 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.

# Martelli and Montanari Folklore

- Folklore: 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. We believe there are problems that produce term structures for which it is false. 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.

# Martelli and Montanari Folklore

- Folklore: 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. We believe there are problems that produce term structures for which it is false. 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.

# Stillman 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].

# Theories

- As well as proving theorems of logic, Tau allows the construction of a formal FOPC theory and making deductions from it.
- Theories already axiomatized
  - Presburger and Peano arithmetic, with and without mathematical induction
  - Commutative ordered fields
  - Graph theory.
  - **Axioms.html**

# Theories

- As well as proving theorems of logic, Tau allows the construction of a formal FOPC theory and making deductions from it.
- Theories already axiomatized
  - Presburger and Peano arithmetic, with and without mathematical induction
  - Commutative ordered fields
  - Graph theory.
  - **Axioms.html**

# Theories

- As well as proving theorems of logic, Tau allows the construction of a formal FOPC theory and making deductions from it.
- Theories already axiomatized
  - Presburger and Peano arithmetic, with and without mathematical induction
  - Commutative ordered fields
  - Graph theory.
  - **Axioms.html**

# Theories

- As well as proving theorems of logic, Tau allows the construction of a formal FOPC theory and making deductions from it.
- Theories already axiomatized
  - Presburger and Peano arithmetic, with and without mathematical induction
  - Commutative ordered fields
  - Graph theory.
  - [Axioms.html](#)

# Theories

- As well as proving theorems of logic, Tau allows the construction of a formal FOPC theory and making deductions from it.
- Theories already axiomatized
  - Presburger and Peano arithmetic, with and without mathematical induction
  - Commutative ordered fields
  - Graph theory.
  - **Axioms.html**

# Theorems of Logic

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]. Sample problems: **T260**, **T265**, **T324**, **T327**

# Inoue Problem

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 **Inoue problem**.

# Los Theorem

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 handles such problems easily; a test run can be made at **Los problem**.

# Theorems of Identity

Tau solves a variety of identity tests. Usually, Brand transformations are applied, but in some proofs a logical axiom for an identity substitution is supplied as a premise as well. We have found that, in some instances, the Brand transformations are speeded up thereby, when heuristic search is also used. However, there are two interesting theorems involving identity from [Montague, Kalish, and Mar, 1980] which Tau has not yet been able to prove, except in a simplified form. These are T328 and T329.

# T328

```
(=> (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))
```

# Overcoming tunnel-vision

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

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

together with the knowledge that :- (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 Tau.

# Theory of a Successor, Presburger and Peano Arithmetic

For axioms and theorems, see: **Axioms.html**, **Presburger**,  
**Peano**, **Arithmetic.th**.

# 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.

- Sample proofs using induction may be run at  
**PALT02Ind.html, PrA55Ind.html, AssocAdd.html.**

# Graph Theory

We have axiomatized in KIF / CLIF several problems in finite graph theory. 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. A sample can be seen at **[GM08d.html](#)** .

# Disproofs

Tau can in certain cases disprove purported 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 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 typical disproof (from [Thomas, 1977]) may be run at **Th267-gA**. 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.

# 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/>.

# 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 / CLIF, 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. Another sample is "Every integer greater than one is divisible by some prime", **NUM015-1** We follow with sample statistics of a run of TPTP-NUM016-1. [Note: some of the run time shown next includes initial invocation of the prover when run in the shell mode.]

## Step Statistics

**Proved Step Stats:** elapsedTime=1.133; cpuTime=0.0;  
steps=5; roots=1; inputs=8; hornInputs=8; definiteInputs=6;  
generalInputs=0; factors=0; premiseLiterals=337;  
rootLiterals=10; generated=189; predicates=6; functions=10;  
constants=6; skFunctions=4; skConstants=2; expanded=156;  
derivations=188; factorizations=0; reductions=0;  
extensions=189; symIDUnif=0; outOrder=1.2051282;  
maxQueue=8127; nResidual=8126; nTooDeep=0;  
nUnacceptable=35; nXUnacceptable=1; nSubsumed=0;  
nVacuous=0; proofLength=5

# Proof Statistics

**Proof Stats:** proved=1; elapsedTime=1.264; cpuTime=0.0;  
subproofs=2; successes=2; steps=7; roots=4; inputs=129;  
hornInputs=121; definiteInputs=110; generalInputs=8;  
factors=0; premiseLiterals=648; rootLiterals=28;  
generated=201; predicates=6; functions=10; constants=6;  
skFunctions=4; skConstants=2; expanded=166;  
derivations=197; factorizations=0; reductions=0;  
extensions=199; symIDUnif=0; outOrder=1.186747;  
maxQueue=8127; nResidual=8594; nTooDeep=0;  
nUnacceptable=35; nXUnacceptable=1; nSubsumed=0;  
nVacuous=1; proofLength=5  
Elapsed: 0m4s; User: 0m2.4s; System: 0m0.1s

# Sample Batch Statistics

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;

## Problem: the Tower of Babel

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.

- Contains notation for a first order logic language for knowledge interchange
- Provides a core semantic framework for logic
- Provides the basis for a set of syntactic forms (dialects) all sharing a common semantics

## Problem: the Tower of Babel

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.

- Contains notation for a first order logic language for knowledge interchange
- Provides a core semantic framework for logic
- Provides the basis for a set of syntactic forms (dialects) all sharing a common semantics

## Problem: the Tower of Babel

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.

- Contains notation for a first order logic language for knowledge interchange
- Provides a core semantic framework for logic
- Provides the basis for a set of syntactic forms (dialects) all sharing a common semantics

# Common Logic

- **Common Logic is a logic framework intended for information exchange and transmission on computer networks.**
- 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 - 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.
- Has a semantics which allows theories to describe intensional entities such as Common Logic properties.
- Fixes the meanings of conventions in widespread use, such as numerals to denote integers and quotation marks to denote character strings. Has provision for the use of datatypes and for naming, importing and transmitting content on the World Wide Web using XML.

# Common Logic

- Common Logic is a logic framework intended for information exchange and transmission on computer networks.
- 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 - 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.
- Has a semantics which allows theories to describe intensional entities such as Common Logic properties.
- Fixes the meanings of conventions in widespread use, such as numerals to denote integers and quotation marks to denote character strings. Has provision for the use of datatypes and for naming, importing and transmitting content on the World Wide Web using XML.

# Common Logic

- Common Logic is a logic framework intended for information exchange and transmission on computer networks.
- 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 - 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.
- Has a semantics which allows theories to describe intensional entities such as Common Logic properties.
- Fixes the meanings of conventions in widespread use, such as numerals to denote integers and quotation marks to denote character strings. Has provision for the use of datatypes and for naming, importing and transmitting content on the World Wide Web using XML.

# Common Logic

- Common Logic is a logic framework intended for information exchange and transmission on computer networks.
- 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 - 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.
- Has a semantics which allows theories to describe intensional entities such as Common Logic properties.
- Fixes the meanings of conventions in widespread use, such as numerals to denote integers and quotation marks to denote character strings. Has provision for the use of datatypes and for naming, importing and transmitting content on the World Wide Web using XML.

# Common Logic

- Common Logic is a logic framework intended for information exchange and transmission on computer networks.
- 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 - 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.
- Has a semantics which allows theories to describe intensional entities such as Common Logic properties.
- Fixes the meanings of conventions in widespread use, such as numerals to denote integers and quotation marks to denote character strings. Has provision for the use of datatypes and for naming, importing and transmitting content on the World Wide Web using XML.

# Common Logic

- SCL ad-hoc working group (formed Dec 2002):

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

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 - **the working document**

## Next steps for Tau

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

- implementation of the treatment of variable-binding operators in [Montague, Kalish, and Mar, 1980] (Chapters X and XI), and of theorem schemata
- formal language translation facilities and simplified English translation facilities
- implementing the use of metalogical expressions in deduction, as axiom schemata and, particularly, in extending the mechanisms for inductions
- implementation of HOL, a la Common Logic (seq vars); further TPTP testing
- graphical notations, including implementation of MathML notation
- persistent KB storage across browser sessions

## Next steps for Tau

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

- implementation of the treatment of variable-binding operators in [Montague, Kalish, and Mar, 1980] (Chapters X and XI), and of theorem schemata
- formal language translation facilities and simplified English translation facilities
- implementing the use of metalogical expressions in deduction, as axiom schemata and, particularly, in extending the mechanisms for inductions
- implementation of HOL, a la Common Logic (seq vars); further TPTP testing
- graphical notations, including implementation of MathML notation
- persistent KB storage across browser sessions

## Next steps for Tau

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

- implementation of the treatment of variable-binding operators in [Montague, Kalish, and Mar, 1980] (Chapters X and XI), and of theorem schemata
- formal language translation facilities and simplified English translation facilities
- implementing the use of metalogical expressions in deduction, as axiom schemata and, particularly, in extending the mechanisms for inductions
- implementation of HOL, a la Common Logic (seq vars); further TPTP testing
- graphical notations, including implementation of MathML notation
- persistent KB storage across browser sessions

## Next steps for Tau

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

- implementation of the treatment of variable-binding operators in [Montague, Kalish, and Mar, 1980] (Chapters X and XI), and of theorem schemata
- formal language translation facilities and simplified English translation facilities
- implementing the use of metalogical expressions in deduction, as axiom schemata and, particularly, in extending the mechanisms for inductions
- implementation of HOL, a la Common Logic (seq vars); further TPTP testing
- graphical notations, including implementation of MathML notation
- persistent KB storage across browser sessions

## Next steps for Tau

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

- implementation of the treatment of variable-binding operators in [Montague, Kalish, and Mar, 1980] (Chapters X and XI), and of theorem schemata
- formal language translation facilities and simplified English translation facilities
- implementing the use of metalogical expressions in deduction, as axiom schemata and, particularly, in extending the mechanisms for inductions
- implementation of HOL, a la Common Logic (seq vars); further TPTP testing
- graphical notations, including implementation of MathML notation
- persistent KB storage across browser sessions

## Next steps for Tau

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

- implementation of the treatment of variable-binding operators in [Montague, Kalish, and Mar, 1980] (Chapters X and XI), and of theorem schemata
- formal language translation facilities and simplified English translation facilities
- implementing the use of metalogical expressions in deduction, as axiom schemata and, particularly, in extending the mechanisms for inductions
- implementation of HOL, a la Common Logic (seq vars); further TPTP testing
- graphical notations, including implementation of MathML notation
- persistent KB storage across browser sessions

# Summary

- Tau is a **hybrid theorem prover for FOPC with identity**
- Tau is **on the web** and easy to use
- **<http://hsinfosystems.com/taujay/>**
- Implementation of HOL, a la Common Logic (seq vars)
- Next steps
  - More intelligent proof search: named subformulas and lemmaization, and templates for more general inductive techniques
  - More TPTP problems

# Summary

- Tau is a **hybrid theorem prover for FOPC with identity**
- Tau is **on the web** and easy to use
- <http://hsinfosystems.com/taujay/>
- Implementation of HOL, a la Common Logic (seq vars)
- Next steps
  - More intelligent proof search: named subformulas and lemmaization, and templates for more general inductive techniques
  - More TPTP problems

# Summary

- Tau is a **hybrid theorem prover for FOPC with identity**
- Tau is **on the web** and easy to use
- **<http://hsinfosystems.com/taujay/>**
- Implementation of HOL, a la Common Logic (seq vars)
- Next steps
  - More intelligent proof search: named subformulas and lemmaization, and templates for more general inductive techniques
  - More TPTP problems

# Summary

- Tau is a **hybrid theorem prover for FOPC with identity**
- Tau is **on the web** and easy to use
- **<http://hsinfosystems.com/taujay/>**
- Implementation of HOL, a la Common Logic (seq vars)
  
- Next steps
  - More intelligent proof search: named subformulas and lemmaization, and templates for more general inductive techniques
  - More TPTP problems





# Summary

- Tau is a **hybrid theorem prover for FOPC with identity**
- Tau is **on the web** and easy to use
- **<http://hsinfosystems.com/taujay/>**
- Implementation of HOL, a la Common Logic (seq vars)
  
- Next steps
  - More intelligent proof search: named subformulas and lemmaization, and templates for more general inductive techniques
  - More TPTP problems




# Summary

- Tau is a **hybrid theorem prover for FOPC with identity**
- Tau is **on the web** and easy to use
- **<http://hsinfosystems.com/taujay/>**
- Implementation of HOL, a la Common Logic (seq vars)
- Next steps
  - More intelligent proof search: named subformulas and lemmaization, and templates for more general inductive techniques
  - More TPTP problems




## For Further Reading I

-  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.
-  Enderton, Herbert B.,  
A Mathematical Introduction to Logic, 2nd Ed.,  
Harcourt Academic Press, New York, 2001
-  Loveland, D.W.  
Automated Theorem Proving: A Logical Basis,  
North-Holland, Amsterdam, 1978.

## For Further Reading II

-  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.
-  Robinson and Voronkov (Eds),  
Handbook of Automated Reasoning (2 vols),  
MIT Press, Cambridge, 2002.

## For Further Reading III

-  Thomas, James A.,  
“Symbolic Logic”,  
Merrill, Columbus, Ohio, 1977.
-  The Tomcat servlet container,  
a product of the Apache Jakarta Project.  
<http://jakarta.apache.org/tomcat/index.html>.
-  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,

## For Further Reading IV

Foundations: Calculi and Methods, pages 353 to 398.  
Kluwer Academic Publishers, Dordrecht, 1998.



Baaz, Matthias, Uwe and Leitsch,  
“Normal Form Transformations”,  
in Robinson and Voronkov (Eds), Handbook of Automated  
Reasoning (2 vols), MIT Press, Cambridge, 2002.



Brand, Daniel.  
“Proving theorems with the modification method”,  
SIAM Journal on Computing, 4(4):412 to 430, 1975.



Bundy, Alan.  
“The Automation of Proof by Mathematical Induction”,  
Handbook of Automated Reasoning 2001: 845-911

## For Further Reading V



**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.

## For Further Reading VI

-  Genesereth et al  
“Knowledge Interchange Format Specification”,  
[http:  
//logic.stanford.edu/kif/specification.html](http://logic.stanford.edu/kif/specification.html)
-  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](http://doi.acm.org/10.1145/3149.214118)
-  Hooker, J.N.  
“Solving the incremental satisfiability problem”,  
[Journal of Logic Programming 15 \(1993\) 177-186.](http://doi.acm.org/10.1145/3149.214118)

## For Further Reading VII

-  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.

## For Further Reading VIII



Loveland, D.W.

“A simplified format for the model elimination procedure”,  
Journal of the ACM, Vol. 15, Issue 2 1969, pp. 349-363;

[http:](http://doi.acm.org/10.1145/321526.321527)



[//doi.acm.org/10.1145/321526.321527](http://doi.acm.org/10.1145/321526.321527), ACM DOI  
bookmark.






Martelli, Alberto and Ugo Montanari,

“An Efficient Unification Algorithm”,  
ACM Trans. Program. Lang. Syst. 4(2): 258-282 (1982).



## For Further Reading IX

-  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.
-  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.

## For Further Reading X

-  Robinson, J.A.  
“A machine-oriented logic based on the resolution principle”,  
Journ. 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.
-  Sun Microsystems,  
Java language,  
<http://java.sun.com/>.

## For Further Reading XI

-  **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/>.