Automated Reasoning: Past Story and New Trends*

Andrei Voronkov
The University of Manchester

Abstract

We overview the development of first-order automated reasoning systems starting from their early years. Based on the analysis of current and potential applications of such systems, we also try to predict new trends in first-order automated reasoning. Our presentation will be centered around two main motives: efficiency and usefulness for existing and future potential applications.

This paper expresses the views of the author on past, present, and future of theorem proving in first-order logic gained during ten years of working on the development, implementation, and applications of the theorem prover Vampire, see [Riazanov and Voronkov, 2002a]. It reflects our recent experience with applications of Vampire in verification, proof assistants, theorem proving, and semantic Web, as well as the analysis of future potential applications.

1 Theorem Proving in First-Order Logic

The idea of automatic theorem proving has a long history both in mathematics and computer science. For a long time, it was believed by many that hard theorems in mathematics can be proved in a completely automatic way, using the ability of computers to perform fast combinatorial calculations. The very first experiments in automated theorem proving have shown that the purely combinatorial methods of proving first-order theorems are too week even for proving theorems regarded as relatively easy by mathematicians.

Provability in first-order logic is a very hard combinatorial problem. First-order logic is undecidable, which means that there is no terminating procedure checking provability of formulas. There are decidable classes of first-order formulas but formulas of these classes do not often arise in applications. Due to undecidability, very short formulas may turn out to be extremely complex, while very long ones rather easy. Sometimes first-order provers find proofs consisting of several thousand steps in a few seconds, but sometimes it takes hours to find a ten-step proof. The theory of first-order reasoning is centered around the completeness theorems while in practice completeness is often not an issue due to the intrinsic complexity of first-order reasoning and is often compromised in favor of a faster inference mechanism.

However, despite the complexity of theorem proving in first-order logic, an impressive progress in the area has been achieved in recent years. This progress is due to several factors: development of theory, implementation techniques, and experience with applications. Modern theorem provers are very powerful programs able to find in a few seconds very complex combinatorial proofs which would require several man-month to be solved by mathematicians. Many ideas developed in theoretical papers in this area are on their way to implementation. There is extensive research on aspects of theorem proving vital for applications, such as proof-checking and interfacing with proof assistants. The next generation of theorem provers will incorporate some of these ideas and become even more powerful tools increasingly used for applications which are currently beyond their reach.

The aim of this paper is to give a brief overview of first-order provers and point out new trends in theorem proving which are likely to be implemented in theorem provers of the next generation.

Currently, theorem provers are used in the following way. The user specifies a problem by giving a set of *axioms* (a set of first order formulas or clauses) and a *conjecture* (again, a first-order formula or a set of clauses). If the input is given by first-order formulas, the prover should check whether the conjecture logically follows from the axioms. If the input is given by a set of clauses, the prover should check whether the set of clauses is inconsistent. In either case, for many applications it is desirable that the prover output a proof, if the logical consequence or inconsistency has been established. The proofs should either be human-readable (for example, when the provers are used for proving theorems in mathematics), or machine-checkable (for example, when provers are used as subsystems of proof assistants or verification systems).

2 Applications

The main application area of theorem provers has been, and continues to be, *verification of software and hardware*. Full applications of this kind usually cannot be directly represented in the first-order form, so provers are normally used to prove subgoals generated by other systems, for example VHDL-to-first-order transformation systems or proof assistants based on higher-order logic or type theories. There are

*Partially supported by a grant from EPSRC.

too many papers on this subject to be mentioned here. Finite-state model checkers and interactive proof assistants are currently prevailing in verification, but with the growing complexity of hardware first-order logic and its extensions are likely to play an increasingly important role.

Theorem proving in mathematics has been the first application area for theorem provers. Provers are not very good at working in structured mathematical theories but they are very efficient in fields of mathematics where combinatorial reasoning is required, for example, in algebra. Applications of provers in algebra are numerous. The monograph [McCune and Padmanabhan, 1996] gives an example of how algebra can be developed with the help of a theorem prover.

Symbolic computation and computer algebra systems need help from theorem provers, for example, when side conditions for applying a simplification rule have to be checked. However, these systems are already very complex and either do not use provers at all or use very primitive ones. Interfacing theorem provers and symbolic computation systems is becoming an important application of theorem provers.

Knowledge bases play a significant role in the semantic web project. Some of these knowledge bases use first-order logic or its simple extensions, see, for example, [IEEE SUO-KIF project, 2002].

Teaching logic and mathematics is an area where first-order provers can serve as valuable tools. The first experiments in teaching mathematics using first-order provers are reported in [McMath *et al.*, 2001].

Other application of first-order provers include retrieval of software components [Schumann and Fischer, 1997], reasoning in non-classical logics (e.g., [de Nivelle *et al.*, 2000]), and program synthesis [Stickel *et al.*, 1994]. First-order theorem provers are in practice used much more than it is reported in the literature, since many companies and organizations do not disclose proprietary use of provers.

3 Modern Theorem Provers

The main currently supported systems are given in Figure 1. Among these systems, Setheo is based on model elimination, while other provers are based on resolution. In the rest of this paper we will overview resolution theorem provers only.

In the early papers, e.g., [Robinson, 1965; Robinson and Wos, 1969] resolution is defined as a logical system consisting of several inference rules operating on clauses, for example, *resolution* and *paramodulation*:

$$\frac{A \vee C \quad \neg A' \vee D}{(C \vee D)\mathrm{mgu}(A, A')} \; ; \quad \frac{s = t \vee C \quad L[s'] \vee D}{(C \vee L[t] \vee D)\mathrm{mgu}(s, s')} \; .$$

where mgu denotes the most general unifier. Since these rules are *local*, i.e., their applicability is identified only by a small number of clauses and the result of a rule application does not change the previously generated clauses, resolution is implemented using *saturation algorithms*.

These algorithms operate on a set of clauses S, initially the set of input clauses. Roughly speaking, they are based on the following loop.

1. Apply inferences to clauses in S, adding to S the conclusions of these inferences.
2. If the empty clause $\square$ is proved, terminate with success. If no inference rule is applicable, terminate with failure.

The set S is the current *search space*. However, a simple implementation of this loop would hardly solve even some of the problems considered trivial by modern theorem provers due to the fast growth of the search space. Already in the first paper on resolution [Robinson, 1965] it was noted that some clauses can be removed from the search space without losing completeness.

More precisely, it has been observed that clauses *subsumed* by other clauses can be removed from the search space without losing completeness and thus regarded as *redundant*. Later, several other notions of redundancy were discovered. For example, [Brand, 1975] proved that the function reflexivity inferences and paramodulation into a variable are redundant. Modern theorem provers use many *redundancy criteria* to prune the search space. These criteria can be divided in two categories: *redundant inferences* and *redundant clauses*. Many useful notions of redundancy are based on *simplification orders* introduced in [Knuth and Bendix, 1970]. These orders are orders $\succ$ on terms which can be extended to literals and clauses. An example of an inference which is redundant due to the orderings restrictions is a paramodulation inference

$$\frac{s = t \vee C \quad L[s'] \vee D}{(C \vee L[t] \vee D)\theta} \; ,$$

for which $t\theta \succ s\theta$. In the 1970s–1980s many notions of redundant inferences and clauses were investigated. In the 1990s a general theory of redundancies was described in [Bachmair and Ganzinger, 1994a]. Nearly all state-of-the-art resolution theorem provers are based on this theory. Resolution with redundancy is based on the following saturation algorithm.

1. Apply all *non-redundant* inferences to clauses in S, adding to S those conclusions of these inferences *that are non-redundant*.
2. If the empty clause $\square$ is proved, terminate with success. If no inference rule is applicable, terminate with failure.
3. Remove all clauses that *become redundant* due to the addition of these conclusions of inferences.

In addition to standard inference rules, these algorithms also operate with *simplifications*. An inference is called a *simplification* if it makes at least one clause in S redundant. Many modern provers implement an eager search for simplifying inferences, while ordinary generating inferences are applied lazily.

In modern provers inference selection is performed via *clause selection*. For this reason saturation algorithms implemented in such provers are called the *given clause algorithms*. There are two main concretisations of the saturation algorithm based on the clause selection: the Otter algorithm [Lusk, 1992; McCune, 1994] and the Discount algorithm [Denzinger *et al.*, 1997]. These algorithms are described and analyzed in more detail in [Riazanov and Voronkov, 2002b]. The simple description of these algorithms given above may create an illusion of their simplicity, but in fact, some of these algorithms are extremely complex, and problems related to

prover	affiliation	references
General-purpose provers		
Otter	Argonne National Laboratory	[McCune, 1994]
Setheo	Munich University of Technology	[Moser *et al.*, 1997]
Spass	Max-Planck Institute	[Weidenbach *et al.*, 1996]
Vampire	University of Manchester	[Riazanov and Voronkov, 2002a]
Gandalf	Tallinn University of Technology	[Tammet, 1997]
SCOTT	Australian National University	[Slaney *et al.*, 1994]
E	Munich University of Technology	[Schulz, 2001]
Snark	SRI	[Stickel *et al.*, 2001]
Bliksem	Max-Planck Institute	[de Nivelle, 2000]

Figure 1: First-order theorem provers

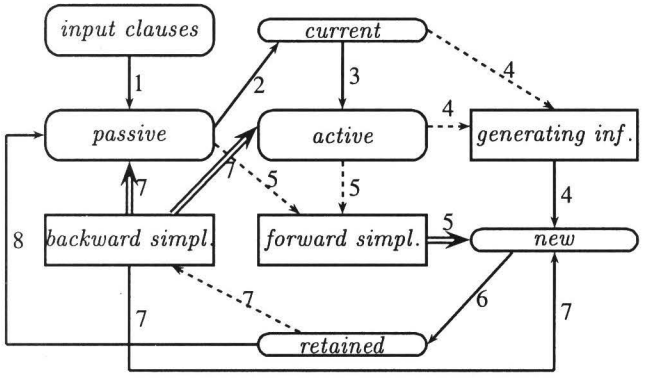

Figure 2: A Given Clause Algorithm of Vampire

the memory management for these algorithms have not been properly solved. For example, one of the given clause algorithms of Vampire is schematically shown in Figure 2 (taken from [Riazanov and Voronkov, 2002a]).

Even proof-search in inference systems with redundancy creates enormously large such spaces. For example, storing 10^6 clauses is not unusual. Of these clauses 10^5 can participate in inferences. Some operations on clauses are very expensive if implemented naively. For example, subsumption on multi-literal clauses is NP-complete and must ideally be performed between each pair of clauses in the search space. It is difficult to imagine an implementation able to perform 10^{12} operations in reasonable time.

In the state-of-the-art theorem provers all expensive operations are implemented using *term indexing* [Graf, 1996; Sekar *et al.*, 2001]. The problem of term indexing can be formulated abstractly as follows. Given a set L of *indexed terms or clauses*, a binary relation R over terms or clauses (called the *retrieval condition*) and a term or clause t (called the *query term or clause*), identify the subset M of L that consists of the terms or clauses l such that $R(l, t)$ holds. A typical retrieval condition used in theorem proving is *subsumption*: retrieve all clauses in L subsumed by t. In order to support rapid retrieval of candidate clauses, we need to process the indexed set into a data structure called the *index*. Modern theorem provers maintain several indexes to support expensive operations. For example, Vampire [Riazanov and Voronkov, 2002a] uses *flatterms* in constant memory for storing temporary clauses, *code trees* [Voronkov, 1995] for forward subsumption, *code trees with precompiled ordering constraints* for forward simplification by unit equalities, *perfectly shared terms* for storing clauses, *shared terms with renaming lists* for backward simplification by unit equalities, *path index with compiled database joins* for backward subsumption and some other indexes.

Serious work with theorem provers requires extensive experiments. Every simple modification to a prover should be tested both for bugs and for efficiency. A typical experiment with Vampire consists of running it for several hours on over 5,000 TPTP problems in a number of modes on a network of around 50 computers. Such experiments require good infrastructure, both software and hardware, to facilitate debugging and evaluation, so Vampire is augmented by a number of programs intended for performing large-scale experiments. The necessity of having such programs and interfacing them with Vampire adds to the complexity of the system.

4 History of Development

Since the early work on automated theorem proving, the area witnessed an impressive progress. This progress is due to several factors described below:

1. Development of *theory*, both of the saturation-based theorem proving with redundancy, see [Bachmair and Ganzinger, 2001; Nieuwenhuis and Rubio, 2001], and of the tableau and model elimination proof-search, see [Hähnle, 2001; Letz and Stenz, 2001; Degtyarev and Voronkov, 2001].
2. Development of *implementation techniques* (term indexing, new algorithms) [Graf, 1996; Sekar *et al.*, 2001; Nieuwenhuis *et al.*, 2001].
3. Growing *experience*, including experiments with applications, the development of TPTP [Sutcliffe and Suttner, 1998], and the annual competitions of theorem provers CASC [Sutcliffe *et al.*, 2002].

Even a naively implemented theorem prover may be powerful enough to solve many problems collected in the TPTP library or created automatically by proof assistants. However, there have been a considerable progress in solving *difficult* problems. Every ten years enhancements in the theory and im-

plementation techniques resulted in a large number problems which could be solved *several orders of magnitude faster* than by the previous generations of theorem provers. Provers implemented around 1970 were very inefficient, especially for equality reasoning. In 1980 the *paramodulation* rule has been generally adopted and the *general architecture* of saturation-based provers understood. By 1990 provers started to use routinely *simplification orderings* and *term indexing*. The provers used in 2000 employ the *general theory of resolution with redundancy*, and in particular *literal selection functions*. The provers of 2010 are likely to benefit from a better *clause selection*, *built-in theories*, and maybe from parallel or concurrent architectures.

5 Future Generation Theorem Provers

Theorem proving is a very hard problem. The next generation of theorem provers will incorporate new theory, data structures, algorithms, and implementation techniques. Their development will be driven by the quest for *flexibility* and *efficiency*. Flexibility is required to adapt provers to new applications. Efficiency can be reformulated as *controlling redundancy in large search spaces* [Lusk, 1992].

It is unreasonable to expect future theorem provers to be much faster on all possible problems. However, if we can increase performance of provers by *several orders of magnitude* for a *large number of problems coming from applications*, many of these problems will be routinely solved, thus saving time for application developers. The development of next generation provers will require:

1. development of *new theory*;
2. addition of *new features*;
3. development of new *algorithms and data structures*;
4. understanding how the theory developed so far can be efficiently implemented *on top of the existing architectures* of theorem provers;

This development is impossible without considerable implementation efforts and extensive experiments.

There is a large number of research areas to be pursued in automated reasoning in the near future, let me mention some of them.

Built-in theories. Already in the case of equality the naive addition of equality axioms is too inefficient for solving even very simple problems. Development of superposition-based equality reasoning resulted in an impressive improvement of theorem provers. There are important theories other than equality which arise in many applications. Development of specialized reasoning methods for these theories will be a central problem in theorem proving. This problem is different from the problem of designing decision procedures for theories such as Presburger arithmetic because relations and functions used in these theories could be used together with other relations and functions and arbitrary quantifiers. Procedures for the combination of decision and unification algorithms can be of some help [Nelson and Oppen, 1979; Shostak, 1984; Baader and Schulz, 1996; Rueß and Shankar, 2001].

The following built-in theories arise in many application and are likely to attract attention in automated reasoning.

- AC (the theory of associative and commutative functions). These axioms occur in axiomatizations very often. There are many results related to building-in AC in theorem provers but implementation techniques, including term indexing modulo AC, are still in their infancy. Associativity and commutativity were built in the EQP theorem prover [McCune, 1997] with a considerable success but essentially without term indexing. Term indexing modulo AC was considered in [Bachmair *et al.*, 1993] but only for a very special case.
- Theories of transitive relations and orders [Bachmair and Ganzinger, 1994b].
- Various first-order theories of arithmetic.
- Term algebras and other constructor-based structures.

As a first step toward building-in important theories one can consider creation of libraries of axioms/theorems about commonly used data types. An example of such a project is the Standard Upper Ontology [IEEE SUO-KIF project, 2002]. One can also enhance theorem provers by constructs for specifying built-in theories, for example, by constraints or by additional inference rules.

Inductively defined types. In many applications of interactive proof assistants one has to deal with inductively defined types and functions on these types. The proof assistants such as Isabelle [Paulson, 2002], HOL [Gordon and Melham, 1993], COQ [The Coq Development Team, 2001], Twelf [Pfenning and Schuermann, 1998] have facilities to define data types and functions inductively. First-order theorem provers have no such facilities. The work on building-in inductively defined types can be developed in the following directions:

- First-order reasoning on data types given by inductive definitions.
- First-order reasoning on functions defined over such data types.
- Limited forms of inductive reasoning.

Working with large axiomatizations. Very often theorem provers must work with large axiomatizations containing many irrelevant axioms. This is typical for applications such as reasoning with ontologies, assisting proof assistants, and verification using hierarchically defined theories. Recognition of irrelevant axioms is one of the most important problems in theorem proving. Many of these axioms are definitions of relations and functions in various forms. Recognition of the most typical definitions and efficient work with them play a major role in the future provers. The first steps in this direction are reported in [Ganzinger *et al.*, 2001; Afshordel *et al.*, 2001; Degtyarev *et al.*, 2002].

Ontology-based knowledge representation. First-order formulations of many applied problems, if not created manually, often refer to large axiomatizations. This situation is common for knowledge-base reasoning, proof assistants, and

theorem proving in mathematics. Using these large axiomatizations and trying to extract only a small number of relevant axioms can be a hopeless task. The problem arises of structuring these large axiomatizations in a such a way that a relevant subset of them could be easily extracted automatically and that particular properties of the axiomatization can be exploited by provers. For example, provers could use a fact that a subset of axioms axiomatize properties of a relation in terms of other relations. Development of knowledge representation formalisms suitable for representing large applied theories and adaptation of provers to such formalisms could become an important problem in automated reasoning.

Proof checking. Strangely enough, modern theorem provers are not always good at producing proofs. Some of them print detailed resolution and paramodulation proofs but none gives a proof of the preprocessing steps, such as skolemisation. Without proof-checking one cannot use provers in verification and in assisting proof assistants. We expect that in the near future proof-checking components will be added to all major first-order provers. We conjecture that the main approaches to proof-checking will be similar to those used in proof-carrying code [Necula and Lee, 2000], so that a proof is built using a number of inference rules, and foundational proof-carrying code [Appel, 2001], so that a proof will given in a system whose language is rich enough to prove even the preprocessing steps, such as HOL.

Reasoning with **nonstandard quantifiers,** such as those specifying restrictions on the number of elements satisfying a relation, for example, $\exists^{\leq n}$ (there exists at most n). Such quantifiers are familiar to the description logic community. One can translate formulas with such quantifiers into first-order logic with equality but the translation will create formulas of a prohibitive size. It is interesting to develop special ways of reasoning with such quantifiers, although this is not an obvious task.

Distributed and other non-standard architectures. It has been observed that cooperating heterogeneous theorem provers can perform much better than the mere sum of their components [M.P. Bonacina, 1994; Denzinger and Fuchs, 1999]. However, modern theorem prover architectures are not suited for cooperative or distributed theorem proving.

Non-resolution inference system. The recent rapid progress of the DPLL-based satisfiability checkers [Zhang and Malik, 2002] suggests that non-resolution systems could also play a significant role in first-order theorem proving. The first implementation of first-order DPLL [Baumgartner, 2000] was not very encouraging, but so were the first implementations of resolution. To be more generally applicable, non-resolution system musts find an efficient solution to the problem of built-in theories [Degtyarev and Voronkov, 2001].

Satisfiability-checking and model building. For many applications it is desirable to be able to establish satisfiability of sets of clauses and build models for satisfiable sets. Satisfiability of first-order is conceptually a much harder problem than unsatisfiability. The set of satisfiable formulas is not recursively enumerable, which means that there is no semi-decision procedure for satisfiability-checking.

Other features. There are other features required of the next generation first-order theorem provers. For example, for the naive users who do not know (and usually do not want to know) much about theorem proving, theorem provers must have a strong auto-mode which will try to select automatically a strategy or strategies best suited for solving the given problem. For more advanced users, there should be options to specify term orderings, literal selection, and clause selection.

References

[Afshordel *et al.*, 2001] B. Afshordel, T. Hillenbrand, and C. Weidenbach. First-order atom definitions extended. In R. Nieuwenhuis and A. Voronkov, editors, *LPAR 2001*, volume 2050 of *LNAI*, pages 309–319. Springer Verlag, 2001.

[Appel, 2001] A.W. Appel. Foundational proof-carrying code. In *LICS 2001*, pages 247–258, 2001.

[Baader and Schulz, 1996] F. Baader and K.U. Schulz. Unification in the union of disjoint equational theories: Combining decision procedures. *Journal of Symbolic Computations*, 21:211–243, 1996.

[Bachmair and Ganzinger, 1994a] L. Bachmair and H. Ganzinger. Rewrite-based equational theorem proving with selection and simplification. *Journal of Logic and Computation*, 4(3):217–247, 1994.

[Bachmair and Ganzinger, 1994b] L. Bachmair and H. Ganzinger. Rewrite techniques for transitive relations. In *LICS'94*. IEEE Computer Society Press, 1994.

[Bachmair and Ganzinger, 2001] L. Bachmair and H. Ganzinger. Resolution theorem proving. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume I, chapter 2, pages 19–99. Elsevier Science, 2001.

[Bachmair *et al.*, 1993] L. Bachmair, T. Chen, and I.V. Ramakrishnan. Associative-commutative discrimination nets. In M.-C. Gaudel and J.-P. Jouannaud, editors, *TAPSOFT'93*, volume 668 of *LNCS*, pages 61–74. Springer Verlag, 1993.

[Baumgartner, 2000] P. Baumgartner. FDPLL — a first-order Davis-Putnam-Logemann-Loveland procedure. In D. McAllester, editor, *CADE-17*, volume 1831 of *LNAI*, pages 200–219. Springer Verlag, 2000.

[Brand, 1975] D. Brand. Proving theorems with the modification method. *SIAM Journal of Computing*, 4:412–430, 1975.

[de Nivelle *et al.*, 2000] H. de Nivelle, R.A. Schmidt, and U. Hustadt. Resolution-based methods for modal logics. *Logic Journal of the IGPL*, 8(3):265–292, 2000.

[de Nivelle, 2000] H. de Nivelle. *Bliksem 1.10 User's Manual*. MPI für Informatik, Saarbrücken, 2000.

[Degtyarev and Voronkov, 2001] A. Degtyarev and A. Voronkov. Equality reasoning in sequent-based calculi. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume I, chapter 10, pages 611–706. Elsevier Science, 2001.

[Degtyarev *et al.*, 2002] A. Degtyarev, R. Nieuwenhuis, and A. Voronkov. Stratified resolution. *Journal of Symbolic Computations*, pages 1–24, 2002. to appear.

[Denzinger and Fuchs, 1999] J. Denzinger and D. Fuchs. Cooperation of heterogeneous provers. In T. Dean, editor, *Proc. (IJCAI-99)*, volume 1, pages 10–15, 1999.

[Denzinger *et al.*, 1997] J. Denzinger, M. Kronenburg, and S. Schulz. DISCOUNT — a distributed and learning equational prover. *Journal of Automated Reasoning*, 18(2):189–198, 1997.

[Ganzinger *et al.*, 2001] H. Ganzinger, R. Nieuwenhuis, and P. Nivela. *The Saturate system*. Max-Planck Institute für Informatik, 2001.

[Gordon and Melham, 1993] M. Gordon and T. Melham. *Introduction to HOL*. Cambridge University Press, 1993.

[Graf, 1996] P. Graf. *Term Indexing*, volume 1053 of *LNCS*. Springer Verlag, 1996.

[Hähnle, 2001] R. Hähnle. Tableaux and related methods. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume I, chapter 3, pages 100–178. Elsevier Science, 2001.

[IEEE SUO-KIF project, 2002] IEEE SUO-KIF project. Standard upper ontology knowledge interchange format, 2002.

[Knuth and Bendix, 1970] D. Knuth and P. Bendix. Simple word problems in universal algebras. In J. Leech, editor, *Computational Problems in Abstract Algebra*, pages 263–297. Pergamon Press, Oxford, 1970.

[Letz and Stenz, 2001] R. Letz and G. Stenz. Model elimination and connection tableau procedures. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume II, chapter 28, pages 2015–2114. Elsevier Science, 2001.

[Lusk, 1992] E.L. Lusk. Controlling redundancy in large search spaces: Argonne-style theorem proving through the years. In A. Voronkov, editor, *LPAR'92.*, volume 624 of *LNAI*, pages 96–106, 1992.

[McCune and Padmanabhan, 1996] W. McCune and R. Padmanabhan. *Automated Deduction in Equational Logic and Cubic Curves*, volume 1095 of *LNAI*. Springer Verlag, 1996.

[McCune, 1994] W.W. McCune. OTTER 3.0 reference manual and guide. Technical Report ANL-94/6, Argonne National Laboratory, 1994.

[McCune, 1997] W. McCune. Solution of the robbins problem. *Journal of Automated Reasoning*, 19(3):263–276, 1997.

[McMath *et al.*, 2001] D. McMath, M. Rozenfeld, and R. Sommer. A computer environment for writing ordinary mathematical proofs. In R. Nieuwenhuis and A. Voronkov, editors, *LPAR 2001*, volume 2050 of *LNAI*, pages 502–511. Springer Verlag, 2001.

[Moser *et al.*, 1997] M. Moser, O. Ibens, R. Letz, J. Steinbach, C. Goller, J. Schumann, and K. Mayr. SETHEO and E-SETHEO—the CADE-13 systems. *Journal of Automated Reasoning*, 18:237–246, 1997.

[M.P. Bonacina, 1994] W.W. McCune M.P. Bonacina. Distributed theorem proving by peers. In A. Bundy, editor, *CADE-12*, volume 814 of *LNAI*, pages 841–845, 1994.

[Necula and Lee, 2000] G.C. Necula and P. Lee. Proof generation in the Touchstone theorem prover. In D. McAllester, editor, *CADE-17*, volume 1831 of *LNAI*, pages 25–44. Springer Verlag, 2000.

[Nelson and Oppen, 1979] G. Nelson and D.C. Oppen. Simplification by cooperating decision procedures. *ACM Transactions on Programming Languages and Systems*, 1(2):245–257, 1979.

[Nieuwenhuis and Rubio, 2001] R. Nieuwenhuis and A. Rubio. Paramodulation-based theorem proving. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume I, chapter 7, pages 371–443. Elsevier Science, 2001.

[Nieuwenhuis *et al.*, 2001] R. Nieuwenhuis, T. Hillenbrand, A. Riazanov, and A. Voronkov. On the evaluation of indexing techniques for theorem proving. In R. Goré, A. Leitsch, and T. Nipkow, editors, *IJCAR 2001*, volume 2083 of *LNAI*, pages 257–271. Springer Verlag, 2001.

[Paulson, 2002] L.C. Paulson. *The Isabelle Reference Manual*. Cambridge University, 2002.

[Pfenning and Schuermann, 1998] F. Pfenning and C. Schuermann. *Twelf User's Guide*, 1998.

[Riazanov and Voronkov, 2002a] A. Riazanov and A. Voronkov. The design and implementation of Vampire. *AI Communications*, 15(2-3):91–110, 2002.

[Riazanov and Voronkov, 2002b] A. Riazanov and A. Voronkov. Limited resource strategy in resolution theorem proving. *Journal of Symbolic Computations*, pages 1–14, 2002. to appear.

[Robinson and Wos, 1969] G. Robinson and L.T. Wos. Paramodulation and theorem-proving in first order theories with equality. In B. Meltzer and D. Michie, editors, *Machine Intelligence*, volume 4, pages 135–150. Edinburgh University Press, 1969.

[Robinson, 1965] J.A. Robinson. A machine-oriented logic based on the resolution principle. *Journal of the Association for Computing Machinery*, 12(1):23–41, 1965. Reprinted as [Robinson, 1983].

[Robinson, 1983] J.A. Robinson. A machine oriented logic based on the resolution principle. In J. Siekmann and G. Wrightson, editors, *Automation of Reasoning. Classical Papers on Computational Logic*, volume 1, pages 397–415. Springer Verlag, 1983. Reprinted from [Robinson, 1965].

[Rueß and Shankar, 2001] H. Rueß and N. Shankar. Deconstructing Shostak. In *LICS 2001*, pages 19–28, 2001.

[Schulz, 2001] S. Schulz. System abstract: E 0.61. In R. Goré, A. Leitsch, and T. Nipkow, editors, *IJCAR 2001*, volume 2083 of *LNAI*, pages 370–375. Springer Verlag, 2001.

[Schumann and Fischer, 1997] J. Schumann and B. Fischer. NORA/HAMMR: Making deduction-based software component retrieval practical. In *Proc. Automated Software Engineering (ASE-97)*, pages 246–254. IEEE Computer Society Press, 1997.

[Sekar *et al.*, 2001] R. Sekar, I.V. Ramakrishnan, and A. Voronkov. Term indexing. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume II, chapter 26, pages 1853–1964. Elsevier Science, 2001.

[Shostak, 1984] R.E. Shostak. Deciding combinations of theories. *Journal of the Association for Computing Machinery*, 31(1):1–12, 1984.

[Slaney *et al.*, 1994] J.K. Slaney, E.L. Lusk, and W. McCune. SCOTT: Semantically Constrained Otter (system description). In A. Bundy, editor, *CADE-12*, volume 814 of *LNAI*, pages 764–768, 1994.

[Stickel *et al.*, 1994] M. Stickel, R. Waldinger, R. Lowry, T. Pressburger, and I. Underwood. Deductive composition of astronomical software from subroutine libraries. In A. Bundy, editor, *CADE-12*, volume 814 of *LNAI*, pages 341–355, 1994.

[Stickel *et al.*, 2001] M.E. Stickel, R.J. Waldinger, and V.K. Chaudhri. *A Guide to Snark*. SRI, 2001.

[Sutcliffe and Suttner, 1998] G. Sutcliffe and C. Suttner. The TPTP problem library — CNF release v. 1.2.1. *Journal of Automated Reasoning*, 21(2), 1998.

[Sutcliffe *et al.*, 2002] G. Sutcliffe, C.B. Suttner, and F. J. Pelletier. The IJCAR ATP system competition. *Journal of Automated Reasoning*, 28(3):307–320, 2002.

[Tammet, 1997] T. Tammet. Gandalf. *Journal of Automated Reasoning*, 18(2):199–204, 1997.

[The Coq Development Team, 2001] The Coq Development Team. *The Coq Proof Assistant Reference Manual. Version 7.2*. INRIA, 2001.

[Voronkov, 1995] A. Voronkov. The anatomy of Vampire: Implementing bottom-up procedures with code trees. *Journal of Automated Reasoning*, 15(2):237–265, 1995.

[Weidenbach *et al.*, 1996] C. Weidenbach, B. Gaede, and G. Rock. SPASS & FLOTTER. Version 0.42. In M.A. McRobbie and J.K. Slaney, editors, *CADE-13*, volume 1104 of *LNCS*, pages 141–145, 1996.

[Zhang and Malik, 2002] L. Zhang and S. Malik. The quest for efficient boolean satisfiability solvers. In A. Voronkov, editor, *CADE-18*, volume 2392 of *LNAI*, pages 295–313, 2002.