AUTOMATED REASONING

MARIA PAOLA BONACINA

Dipartimento di Informatica – Università degli Studi di Verona

ALBERTO MARTELLI

Dipartimento di Informatica – Università degli Studi di Torino

1 Introduction

A central problem in *automated reasoning* is to determine whether a conjecture φ, that represents a property to be veriÞed, is a logical consequence of a set S of assumptions, which express properties of the object of study (e.g., a system, a circuit, a program, a data type, a communication protocol, a mathematical structure).

A conjoint problem is that of *knowledge representation*, or Þnding suitable formalisms for S and φ to represent aspects of the real world, such as action, space, time, mental events and commonsense reasoning. While *classical logic* has been the principal formalism in automated reasoning, and many proof techniques have been studied and implemented, *non-classical logics*, such as modal, temporal, description or nonmonotonic logics, have been widely investigated to represent knowledge.

2 Automated reasoning in classical logic

Given the above central problem, one can try to answer afÞrmatively, by Þnding a *proof* of φ from S. This problem and the methods to approach it are called *theorem proving*. Theorem proving comprises both *deductive theorem proving*, which is concerned precisely with the entailment problem as stated above (in symbols: $S \models \varphi$), and *inductive theorem proving*, where the problem is to determine whether S entails all ground instances of φ (in symbols: $S \models \varphi\sigma$, for all ground substitutions σ).

In *(fully) automated theorem proving*, the machine alone is expected to Þnd a proof. In *interactive theorem proving*, a proof is born out of the interaction between human and machine. Since it is too difÞcult to Þnd a proof ignoring the conjecture, the vast majority of theorem-proving methods work *refutationally*, that is, they prove that φ follows logically from S, by showing that $S \cup \{\neg\varphi\}$ generates a contradiction, or is *inconsistent*.

Otherwise, given assumptions S and conjecture φ, one can try to answer negatively, disproving φ, by Þnding a *counter-example*, or *counter-model*, that is, a model of $S \cup \{\neg\varphi\}$. This branch of automated reasoning is called *automated model building*.

In classical Þrst-order logic, deductive theorem proving is *semi-decidable*, while inductive theorem proving and model building are *not even semi-decidable*. It is significant that while books in theorem proving date from the early seventies [22, 48, 16, 27, 77, 44, 70], the Þrst book on model building appeared only recently [21]. Most approaches to automated model building belong to one of the following three classes, or combine their principles:

1. *Enumeration methods* generate interpretations and test whether they are models of the given set of formul¾;

2. *Saturation methods* extract models from the Þnite set of formul¾ generated by a failed refutation attempt; and

3. *Simultaneous methods* search simultaneously for a refutation or a model of the given set of formul¾.

In higher-order logics, that allow universal and existential statements, not only on individuals, but also on functions and predicates, even deductive theorem proving is *no longer semi-decidable*. Clearly, fully automated theorem proving focuses on deductive theorem proving, while induction, model generation and reasoning in higher-order logics resort to a larger extent to interactive theorem proving. Since the most important feature of higher-order logic for computer science are *higher-order functions*, that are a staple of functional programming languages, an intermediate solution is to develop a Þrst-order system, with a functional programming language, used simultaneously as programming language and as logical language [20, 43].

2.1 Fully automated theorem proving

Semi-decidability means that no algorithm is guaranteed to halt, and return a proof, whenever $S \cup \{\neg\varphi\}$ is inconsistent, or a model, whenever $S \cup \{\neg\varphi\}$ is consistent. The best one can have is a *semi-decision procedure*, that is guaranteed to halt and return a proof, if $S \cup \{\neg\varphi\}$ is inconsistent. If it halts without a proof, we can conclude that $S \cup \{\neg\varphi\}$ is consistent, and try to extract a model from its output. However, if $S \cup \{\neg\varphi\}$ is consistent, the procedure is not guaranteed to halt.

Intuitively, proofs of inconsistency of a given problem $S \cup \{\neg\varphi\}$ are Þnite, if they exist, but there is an *inÞnite search space* of logical consequences where to look for a contradiction. A machine can explore only a Þnite part of this inÞnite space, and the challenge is to Þnd a proof using as little resources as possible. A fundamental insight was the recognition that the ability to detect and discard *redundant* formul¾ is as crucial as the ability to generate consequences of given formul¾ In addition to standard *expansion* inference rules of the form

$$\frac{A_1 \quad \dots \quad A_n}{B_1 \quad \dots \quad B_m} \tag{1}$$

which add inferred formul¾ $B_1, \dots, B_m$ to the set of known theorems, that already includes the premises $A_1, \dots, A_n$, contemporary *inference systems* feature *contraction* rules, that delete or simplify already-inferred theorems. The Òdouble-ruled inference ruleÓ form

$$\frac{A_1 \quad \dots \quad A_n}{\overline{B_1 \quad \dots \quad B_m}} \tag{2}$$

means that the formul¾ (A_i) above the rule are *replaced* by those below (B_j). It is a *deletion* rule if the consequences are a proper subset of the premises; otherwise, it is a *simpliÞcation* rule.

An expansion rule is *sound* if what is generated is logical consequence of the premises $(\{A_1 \dots\ A_n\} \models \{B_1 \dots B_m\})$. Classical examples are *resolution* and *paramodulation*. A contraction rule is *sound* if what is removed is logical consequence of what is left or added $(\{B_1 \dots B_m\} \models \{A_1 \dots\ A_n\})$. Classical examples are *subsumption* and *equational simpliÞcation* from *Knuth-Bendix completion*. An inference system is *sound* if all its rules are, and it is *refutationally complete*, if it allows us to derive a contradiction, whenever the initial set of formul¾ is inconsistent. The challenge is dealing with contraction without endangering completeness [36, 7, 8, 18]: a key ingredient is to order the data (terms, literals, clauses, formul¾ proofs) according to *well-founded orderings*. Inference systems of this nature were applied successfully also to inductive theorem proving as in *inductionless induction* or *proof by the lack of inconsistency* [37, 40, 18].

2.2 Decision procedures and SAT solvers

Decidable instances of reasoning problems do exist. For these problems, the search space is Þnite and *decision procedures* are known. Decidability may stem from imposing restrictions on

1. the logic,
2. the form of admissible formulae for S or φ, or
3. the theory presented by the assumptions in S.

An example of Case 1 is the *guarded fragment* of Þrst-order logic, which propositional modal logic can be reduced to. The most prominent instance is *propositional logic*, whose decidable satisÞability problem is known as SAT. Many problems in computer science can be encoded in propositional logic, reduced to SAT and submitted to *SAT solvers*. As automated reasoning is concerned primarily with *complete* SAT solvers, the dominating paradigm is the DPLL procedure [25, 24, 79], implemented, among others, in [78, 52].

As an example of Case 2, the Bernays-SchonÞnkel class admits only sentences in the form

$$\exists x_1, \dots x_n. \forall y_1, \dots y_m. P[x_1, \dots x_n, y_1, \dots y_m]$$

where P is quantiÞer-free. Decidable classes based on syntactic restrictions are surveyed in [21].

Case 3 includes Presburger arithmetic or theories of data structures, such as lists or arrays. For the latter, the typical approach is to build a Ò*little engine of proof*Ó for each theory [66], by building the theoryÕs axioms into a *congruence closure* algorithm to handle ground equalities [67, 54, 9]. Little engines are combined to handle combinations of theories [53, 68, 31]. However, also generic theorem-proving methods proved competitive on these problems [5].

Decidable does not mean pratical, and the decidable reasoning problems are typically *NP-complete*. Since automated reasoning problems range from decidable, but *NP-complete*, to semi-decidable, or not even semi-decidable, automated reasoning relies pretty much universally on the artiÞcial intelligence paradigm of *search*.

2.3 Automated reasoning as a search problem

Automated reasoning methods are *strategies*, composed of an *inference system* and a *search plan*. The inference system is a *non-deterministic* set of inference rules, that deÞnes a *search space* containing all possible inferences. Describing formally the search space of a reasoning problem is not obvious, and can be approached through different formalisms that capture different levels of abstraction [62, 19]. The search plan guides the search and determines the unique *derivation*

$$S_0 \vdash S_1 \vdash \dots S_i \vdash S_{i+1} \vdash \dots$$

that the strategy computes from a given input $S_0 = S \cup \{\neg\varphi\}$. It is the addition of the search plan that turns a non-deterministic inference system into a *deterministic proof procedure*.

The search plan decides, at each step, which inference rule to apply to which data. If it selects an expansion rule, the set of formul¾is expanded:

$$\frac{S}{S'} \quad S \subset S'$$

If it selects a contraction rule, the set of formul¾ is contracted:

$$\frac{S}{S'} \quad S \not\subseteq S' \quad S' \prec_{mul} S$$

where $\prec_{mul}$ is the multiset extension of a well-founded ordering on clauses. Strategies that employ well-founded orderings to restrict expansion and deÞne contraction are called *ordering-based*. Ordering-based strategies with a *contraction-Þrst* search plan, that gives higher priority to contraction inferences, are termed *contraction-based*.

These strategies work primarily by *forward reasoning*, because they do not distinguish between clauses coming from S and clauses coming from $\neg\varphi$. *Semantic strategies*, strategies with *set of support* and *target-oriented* strategies were devised to limit this effect.

At the other extreme of the spectrum, *subgoal-reduction* strategies work by reducing goals to subgoals. This class includes methods based on *model elimination*, *linear resolution*, *matings* and *connections*, all eventually understood in the context of *clausal normalform tableaux*.

The picture is completed by *instance-based* strategies, that date back to *GilmoreÕs multiplication method*. These strategies generate ground instances of the clauses in the set to be refuted, and detect inconsistencies at the propositional level by using a SAT solver. A survey of strategies, according to this classiÞcation, with the relevant references, was given in [17].

Interactive reasoning systems with higher-order features also employ search, but only indirectly, or at the *meta-level*, because the search is made of both automated and human-driven steps [23, 33, 60, 4, 13, 15]. An interactive session generates a *proof plan*, that is, a sequence of *actions* to reach a proof. Actions may be chosen by the user or the search plan of the interactive prover. In turn, an action can be the application of an inference rule of the interactive prover, the introduction of a lemma by the user, the invocation of an automated Þrst-order prover [12] or a decision procedure [59], to dispatch a Þrst-order conjecture or a decidable subproblem, respectively.

2.4 Applications

Its intrinsic difÞculty notwithstanding, automated reasoning is important in several ways. Its direct applications, such as *hardware/software veriÞcation* and *program generation*, are of the highest relevance to computing and society. Theorem provers [73, 50, 42, 46, 74, 55, 63, 64, 71] were applied successfully to the veriÞcation of cryptographic protocols, message-passing systems and software speciÞcations [72, 65]. Furthermore, automated reasoning contributes techniques to other Þelds of artiÞcial intelligence, such as *planning*, *learning* and *natural language understanding*, symbolic computation, such as *constraint problem solving* and *computer algebra*, computational logic, such as *declarative programming* and *deductive databases*, and mathematics, as witnessed by the existence of *databases of computer-checked mathematics* [51]. Theorem provers are capable of proving non-trivial mathematical theorems in theories such as Boolean algebras, rings, groups, quasigroups and many-valued logic [3, 2, 41, 49, 75]. Last, the study of mechanical forms of logical reasoning is part of the fundamental quest about what computing machines can do.

3 Automated reasoning in non-classical logic

Many aspects of AI problems can be modeled with logical formalisms, and in particular, with so called nonclassical logics, such as modal or temporal logics. Automated deduction techniques have been developed for those logics, for instance by proposing tableau proof methods [34]. Another approach is to translate formulas of nonclassical logic into formulas of classical logic, so as to give users of nonclassical logics access to the sophisticated state-of-the-art tools that are available in the area of Þrst-order theorem proving [57].

An important research problem in AI is the logical formalization of commonsense reasoning. The observation that traditional logics, even nonclassical ones, are not suitable to express revisable inferences, led to the deÞnition of nonmonotonic logics. Various approaches have been used to do nonmonotonic reasoning, based on *Þxpoint* techniques or *semantic preference*. [58] contains a survey of tableau based proof methods for nonmonotonic logics.

As we cannot give here the details of all techniques for automated reasoning in those logics, we will describe only some speciÞc approaches that have been used with success.

3.1 Extensions of Logic Programming

Logic programming was proposed with the goal of combining the use of logic as a representation language with efÞcient deduction techniques, based on a backward inference process (goal-directed) which allows to consider a set of formulas as a program. Prolog is the most widely used logic programming language. While originally logic programming was conceived as a subset of classical logic, it was soon extended with some nonclassical features, in particular *negation as failure*. To prove a negated goal $not\ p$, Prolog tries to prove p; if p cannot be proved, then the goal $not\ p$ succeeds, and vice versa. This simple feature of Prolog has been widely used to achieve nonmonotonic behavior. In fact, by adding new formulas, a goal p which previously was not derivable might become true, and, as a consequence, $not\ p$ might become false.

The semantics of negation as failure has been deeply studied, and the relations with nonmonotonic logics have been pointed out. The most widely accepted semantics is the *answer set semantics* [30]. According to this semantics, a logic program may have several alternative models, called *answer sets*, each corresponding to a possible view of the world.

Logic programming has been made more expressive by extending it with the so called classical negation, that is monotonic negation of classical logic, and disjunction in the head of the rules. Recently, a new approach to logic programming, called *answer set programming* (ASP), has emerged. Syntactically ASP programs look like Prolog programs, but the computational mechanisms used in ASP are different: they are based on the ideas that have led to the creation of fast satisfiability solvers for propositional logic. ASP has emerged from interaction between two lines of research, the semantics of negation in logic programming and application of satisfiability solvers to search problems. Several efficient answer set solvers have been developed, among which we can mention Smodels [69] and DLV [45], the latter providing an extension for dealing with preferences.

Often, automated reasoning paradigms in AI mimic human reasoning, providing a formalisation of the human basic inferences. *Abductive reasoning* is one such paradigm, and it can be seen as a formalisation of abductive reasoning and hypotheses making. Hypotheses make up for lack of information, and they can be put forward to support the explanation of some observation.

Abductive logic programming is an extension of logic programming in which the knowledge base may contain special atoms that can be *assumed* to be true, even if they are not defined, or cannot be proven. These atoms are called *abducibles*. Starting from a goal G, an abductive derivation tries to verify G, by using deductive inference steps as in logic programming, but also by possibly assuming that some abducibles are true. In order to have this process converging to a meaningful explanation, an abductive theory normally comes together with a set of *integrity constraints* IC, and, in this case, hypotheses are required to be consistent with IC [39, 28, 38].

It is worth mentioning that the goal directed approach of logic programming has been used also to formulate the proof theory of many non-classical logics. For instance [29] presents a uniform Prolog-like formulation for many intuitionistic and modal logics.

3.2 Model checking

Model checking is an automatic technique for formally verifying finite state concurrent systems, which has been successfully applied in computer science to verify properties of distributed software systems. The process of model checking consists of the following steps. First the software system to be verified must be translated into a suitable formalism, where the actions of the systems are represented in terms of *states* and *transitions*, thus obtaining the *model*. Then the properties to be verified will be specified as a formula φ in some logical formalism. Usually properties have to do with the evolution of the behavior of the system over time, and are expressed by means of *temporal logic*. The last step consists in the verification that φ holds in the model. The verification techniques depend on the kind of temporal logic which is used, i.e. *branching-time* or *linear-time*.

Many model checking tools have been developed, among which we can mention NuSMV [56] and SPIN [35].

Although model checking has been mainly used for verification of distributed systems, there have been proposals to use this technique also for the verification of AI systems, such as multi-agent systems. These proposals deal with the problem of expressing properties regarding not only temporal evolution, as usual in model checking, but also mental attitudes of agents, such as knowledge, beliefs, desires, intentions (BDI). This requires to combine temporal logic with modal (epistemic) logics which have been used to model mental attitudes.

The goal of [11] is to extend model checking to make it applicable to multi-agent systems, where agents have BDI attitudes. This is achieved by using a new logic which is the composition of two logics, one formalizing temporal evolution and the other formalizing BDI attitudes. The model checking algorithm keeps the two aspects separated: when considering the temporal evolution of an agent, BDI atoms are considered as atomic proposition.

A different framework for verifying temporal and epistemic properties of multi-agent systems by means of model checking techniques is presented by Penczek and Lomuscio [61]. Here multi-agent systems are formulated in the logic language CTLK, which adds to the temporal logic CTL an epistemic operator to model knowledge, using *interpreted systems* as underlying semantics.

3.3 Applications

3.3.1 Reasoning about actions

The most famous approach to reasoning about actions is situation calculus, proposed by John McCarthy. Situations are logical terms which describe the state of the world whenever an action is executed. A situation defines the truth value of a set of *fluents*, predicates that vary from one situation to the next. Actions are described by specifying their preconditions and effects by means of first-order logic formulas. For instance, the formula $p(s) \rightarrow q(result(a, s))$ means that, if p holds in situation s, then q will hold after executing action a.

An alternative logical representation of actions is by means of modal logic, where each modality represents an action [26]. For instance, the formula $\Box(p \rightarrow [a]q)$ has the same meaning as the previous one ($\Box\varphi$ means that φ is true in each state). Since the semantics of modal logic is based on the so called *possible worlds*, it is rather natural to adopt it for reasoning about actions, by associating possible worlds with states, and transitions between worlds with actions.

An important problem which arises in reasoning about actions is the so called *frame problem*, i.e. the problem of specifying in an efÞcient way what are the ßuents that do not change from one situation to the next one when an action is executed. Usually this problem is formulated in a nonmonotonic way, by saying that we assume that each ßuent persists if it is consistent to assume it. The frame problem has been formally represented by means of non-monotonic formalisms, or in classical logic by means of a ÒcompletionÓ construction due to Reiter.

Among other formalisms we can mention the *event calculus*, an extension of logic programming with explicit time points, and *ßuent calculus*.

Formal techniques for reasoning about actions have been mainly applied in the area of planning, where the term *cognitive robotics* was coined. In this context, the robot programming language GOLOG [47] has been deÞned, based on the situation calculus. GOLOG allows to write programs by means of statements of imperative programming languages (similar to those provided by dynamic logic). GOLOG programs are nondeterministic, and plans can be obtained by searching for suitable program executions satisfying a given goal. The language has been extended to deal with concurrency and sensing.

A different approach, based on modal logic, is presented in [10] where programs consist of sets of Prolog-like rules and can be executed by means of a goal-directed proof procedure.

3.3.2 Multi-agent systems

Many of the techniques described in this article have been applied to reasoning in multi- agent systems. We have already mentioned extensions to model checking to deal with agentsÕ mental attitudes.

The issue of developing semantics for agent communication languages has been examined by many authors, in particular by considering the problem of giving a *veriÞable* semantics, i.e. a semantics *grounded* on the computational models. Given a formal semantics, it is possible to deÞne what it means for an agent to be respecting the semantics of the communicative action when sending a message. VeriÞcation techniques, such as model checking can be used to check it. For instance, in [76] agents are written in MABLE, an imperative programming language, and have a mental state. MABLE systems may be augmented by the addition of formal claims about the system, expressed using a quantiÞed, linear time temporal BDI logic. Properties of MABLE programs can be veriÞed by means of the SPIN model checker, by translating BDI formulas into the form used by SPIN.

The problem of verifying agentsÕ compliance with a protocol at runtime is addressed in [1]. Protocols are speciÞed in a logic-based formalism based on Social Integrity Constraints, which constrain the agentsÕ observable behavior. The paper present a system that, during the evolution of a society of agents, veriÞes the compliance of the agentsÕ behavior to the protocol, by checking fulÞllment or violation of expectations.

Another approach for the speciÞcation and veriÞcation of interaction protocols is proposed in [32] using a combination of dynamic and temporal logic. Protocols are expressed as regular expressions, (communicative) actions are speciÞed by means of action and precondition laws, and temporal properties can be expressed by means of the *until* operator. Several kinds of veriÞcation problems can be addressed in that framework, including the veriÞcation of protocol properties and the veriÞcation that and agent is compliant with a protocol.

3.3.3 Automated reasoning on the web

Automated reasoning is becoming an essential issue in many web systems and applications, especially in emerging *Semantic Web* applications. The aim of the Semantic Web initiative is to advance the state of the web through the use of semantics. Various formalisms have already emerged, like RDF or OWL, an ontology language stemming from description logics. So far, reasoning on the Semantic Web is mostly reasoning about knowledge expressed in a particular ontology.

The next step will be the logic of proof layers, and logic programming based rule systems appear to lie in the mainstream of such activities. Combinations of logic programming and description logics have been studied, and nonmonotonic extensions have been proposed, in particular regarding the use of *Answer Set Programming*. These research issues are investigated in REWERSE, ÒReasoning on the Web with Rules and SemanticsÓ, a research Network of Excellence of the 6th Framework Programme (http://rewerse.net/).

Web services are rapidly emerging as the key paradigm for the interaction and coordination of distributed business processes. The ability to automatic reason about web services, for instance to verify some properties or to compose them, is an essential step toward the real usage of web services. Web services have many analogies with agents, and thus many of the techniques previously mentioned are also being used to reason about web services. In particular, regarding web service composition, we can mention [14] and the ASTRO project [6] which has developed techniques and tools for web service composition, in particular by making use of sophisticated planning techniques, which can deal with nondeterminism, partial observability and extended goals.

REFERENCES

[1] M. Alberti, D. Daolio, P. Torroni, M. Gavanelli, E. Lamma, and P. Mello. SpeciÞcation and veriÞcation of agent interaction protocols in a logic-based system. In *SAC*, pages 72 78, 2004.

[2] S. Anantharaman and M. P. Bonacina. An application of automated equational reasoning to many-valued logic. In *CTRS-90*, volume 516 of *LNCS*, pages 156 161. Springer, 1990.

[3] S. Anantharaman and J. Hsiang. Automated proofs of the Moufang identities in alternative rings. *J. Automat. Reason.*, 6(1):76 109, 1990.

[4] P. B. Andrews, M. Bishop, S. Issar, D. Nesmith, F. Pfenning, and H. Xi. TPS: a theorem proving system for classical type theory. *J. Automat. Reason.*, 16(3):321 353, 1996.

[5] A. Armando, M.P. Bonacina, S. Ranise, and S. Schulz. On a rewriting approach to satisÞability procedures: extension, combination of theories and an experimental appraisal. In *FroCoS-5*, volume 3717 of *LNAI*, pages 65 80. Springer, 2005.

[6] ASTRO. http://sra.itc.it/projects/astro/.

[7] L. Bachmair and N. Dershowitz. Equational inference, canonical proofs, and proof orderings. *J. ACM*, 41(2):236 276, 1994.

[8] L. Bachmair and H. Ganzinger. Rewrite-based equational theorem proving with selection and simpliÞcation. *J. Logic and Comput.*, 4(3):217 247, 1994.

[9] L. Bachmair, A. Tiwari, and L. Vigneron. Abstract congruence closure. *J. Automat. Reason.*, 31(2):129 168, 2003.

[10] M. Baldoni, L. Giordano, A. Martelli, and V. Patti. Programming rational agents in a modal action logic. *Annals of Mathematics and ArtiÞcial Intelligence*, 41(2 4):207 257, 2004.

[11] M. Benerecetti, F. Giunchiglia, and L. SeraÞni. Model checking multiagent systems. *J. Log. Comput.*, 8(3):401 423, 1998.

[12] C. Benzmuller, L. Cheikhrouhou, D. Fehrer, A. Fiedler, Huang, M. Kerber, M. Kohlhase, K. Konrad, and E. Melis. ΩMEGA: towards a mathematical assistant. In *CADE-14*, volume 1249 of *LNAI*, pages 252 255. Springer, 1997.

[13] C. Benzmuller and M. Kohlhase. LEO - A higher-order theorem prover. In *CADE-15*, volume 1421 of *LNAI*, pages 139 143. Springer, 1998.

[14] D. Berardi, G. De Giacomo, M. Lenzerini, M. Mecella, and D. Calvanese. Synthesis of underspeciÞed composite -services based on automated reasoning. In *ICSOC*, pages 105 114, 2004.

[15] Y. Bertot and P. Casteran. *Interactive Theorem Proving and Program Development CoqÕArt: The Calculus of Inductive Constructions*. Springer, 2004.

[16] W. Bibel. *Automated Theorem Proving*. Friedr. Vieweg & Sohn, 2nd edition, 1987.

[17] M.P. Bonacina. A taxonomy of theorem-proving strategies. In *ArtiÞcial Intelligence Today Recent Trends and Developments*, volume 1600, pages 43 84. Springer, 1999.

[18] M.P. Bonacina and J. Hsiang. Towards a foundation of completion procedures as semidecision procedures. *Theor. Comput. Sci.*, 146:199 242, 1995.

[19] M.P. Bonacina and J. Hsiang. On the modelling of search in theorem proving towards a theory of strategy analysis. *Inf. Comput.*, 147:171 208, 1998.

[20] R. S. Boyer and J S. Moore. *A Computational Logic Handbook*. Academic Press, 1988.

[21] R. Caferra, A. Leitsch, and N. Peltier. *Automated Model Building*. Kluwer, 2004.

[22] C. L. Chang and R. C. T. Lee. *Symbolic Logic and Mechanical Theorem Proving*. Academic Press, 1973.

[23] R. L. Constable. *Implementing Mathematics with the Nuprl Proof Development System*. Prentice Hall, 1986.

[24] M. Davis, G. Logemann, and D. W. Loveland. A machine program for theorem proving. *C. ACM*, 5:394 397, 1962.

[25] M. Davis and H. Putnam. A computing procedure for quantiÞcation theory. *J. ACM*, 7:201 215, 1960.

[26] G. De Giacomo and M. Lenzerini. PDL-based framework for reasoning about actions. In M. Gori and G. Soda, editors, *AI*IA*, volume 992 of *Lecture Notes in Computer Science*, pages 103 114. Springer, 1995.

[27] M. Fitting. *First-order Logic and Automated Theorem Proving*. Springer, 1990.

[28] T.H. Fung and R.A. Kowalski. The IFF proof procedure for abductive logic programming. *J. Log. Program.*, 33(2):151 165, 1997.

[29] D.M. Gabbay and N. Olivetti. *Goal-Directed Proof Theory*. Kluwer Academic Publishers, 2000.

[30] M. Gelfond and Vladimir Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Comput.*, 9(3/4):365 386, 1991.

[31] S. Ghilardi, E. Nicolini, and D. Zucchelli. A comprehensive framework for combined decision procedures. In *FroCoS-5*, volume 3717 of *LNAI*, pages 1 30. Springer, 2005.

[32] L. Giordano, A. Martelli, and C. Schwind. Specifying and verifying interaction protocols in a temporal action logic. *Journal of Applied Logic*, 2006. to appear.

[33] M. Gordon and T. F. Melham. *Introduction to HOL - A Theorem Proving Environment for Higher Order Logic*. Cambridge Univ. Press, 1993.

[34] R. Gore. Tableau methods for modal and temporal logics. In M DÕAgostino, D Gabbay, R Haehnle, and J Posegga, editors, *Handbook of Tableau Methods*, pages 297 396. Kluwer Academic Publishers, 1999.

[35] G.J. Holzmann. *The SPIN Model Checker*. Addison-Wesley, 2003.

[36] J. Hsiang and M. Rusinowitch. Proving refutational completeness of theorem proving strategies: the transÞnite semantic tree method. *J. ACM*, 38(3):559 587, 1991.

[37] G. Huet and J. M. Hullot. Proofs by induction in equational theories with constructors. *J. Comput. Syst. Sci.*, 25:239 266, 1982.

[38] A. C. Kakas and P. Mancarella. On the relation between truth maintenance and abduction. In *Proceedings of the 2nd PaciÞc Rim International Conference on ArtiÞcial Intelligence*, 1990.

[39] A. C. Kakas, A. Michael, and C. Mourlas. ACLP: Abductive Constraint Logic Programming. *Journal of Logic Programming*, 44(1-3):129 177, July 2000.

[40] D. Kapur and D. R. Musser. Proof by consistency. *Artif. Intell.*, 31:125 157, 1987.

[41] D. Kapur and H. Zhang. A case study of the completion procedure: proving ring commutativity problems. In *Computational Logic Essays in Honor of Alan Robinson*, pages 360 394. The MIT Press, 1991.

[42] D. Kapur and H. Zhang. An overview of Rewrite Rule Laboratory (RRL). *Computers and Mathematics with Applications*, 29(2):91 114, 1995.

[43] M. Kaufmann, P. Manolios, and J S. Moore. *Computer Aided Reasoning : ACL2 Case Studies*. Kluwer, 2000.

[44] A. Leitsch. *The Resolution Calculus*. Springer, 1997.

[45] N. Leone, G. Pfeifer, W. Faber, T. Eiter, G. Gottlob, S. Perri, and F. Scarcello. The DLV system for knowledge representation and reasoning. *ACM Transactions on Computational Logic*, to appear, 2002.

[46] R. Letz, J.M. Schumann, S. Bayerl, and W. Bibel. SETHEO: a high performance theorem prover. *J. Automat. Reason.*, 8(2):183 212, 1992.

[47] H.J. Levesque, R. Reiter, Y. Lesperance, F. Lin, and R.B. Scherl. GOLOG: A logic programming language for dynamic domains. *Journal of Logic Programming*, 19(20):1 679, 1994.

[48] D. W. Loveland. *Automated Theorem Proving: A Logical Basis*. North-Holland, 1978.

[49] W. W. McCune. Solution of the Robbins problem. *J. Automat. Reason.*, 19(3):263 276, 1997.

[50] W.W. McCune. Otter 3.0 reference manual and guide. Technical Report 94/6, MCS Division, Argonne National Laboratory, 1994.

[51] Mizar. http://mizar.uwb.edu.pl/, 2006.

[52] M.W. Moskewicz, C.F. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an efÞcient SAT solver. In David Blaauw and Luciano Lavagno, editors, *DAC-39*, 2001.

[53] G. Nelson and D.C. Oppen. SimpliÞcation by cooperating decision procedures. *ACM TOPLAS*, 1(2):245 257, 1979.

[54] G.Nelson and D.C. Oppen. Fast decision procedures based on congruence closure. *J. ACM*, 27(2):356 364, 1980.

[55] R. Niewenhuis, J.M. Rivero, and M.A. Vallejo. The Barcelona prover. *J. Automat. Reason.*, 18(2), 1997.

[56] NuSMV. http://nusmv.irst.itc.it/.

[57] H.J. Ohlbach, A. Nonnengart, M. de Rijke, and D.M. Gabbay. Encoding two-valued nonclassical logics in classical logic. In John Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning*, pages 1403 1486. Elsevier and MIT Press, 2001.

[58] N. Olivetti. Tableaux for nonmonotonic logics. In M DÕAgostino, D Gabbay, R Haehnle, and J Posegga, editors, *Handbook of Tableau Methods*, pages 469 528. Kluwer Academic Publishers, 1999.

[59] S. Owre, J. Rushby, N. Shankar, and D. Stringer-Calvert. PVS: an experience report. In *Applied Formal Methods FM-Trends 98*, volume 1641 of *LNCS*, pages 338 345. Springer, 1998.

[60] L. C. Paulson. *Isabelle: A Generic Theorem Prover*, volume 828 of *LNCS*. Springer, 1994.

[61] W. Penczek and A. Lomuscio. Verifying epistemic properties of multi-agent systems via bounded model checking. *Fundam. Inform.*, 55(2):167 185, 2003.

[62] D. A. Plaisted and Y. Zhu. *The EfÞciency of Theorem Proving Strategies*. Vieweg & Sohns, 1997.

[63] A. Riazanov and A. Voronkov. The design and implementation of VAMPIRE. *J. AI Commun.*, 15(2/3):91 110, 2002.

[64] S. Schulz. E a brainiac theorem prover. *J. AI Commun.*, 15(2 3):111 126, 2002.

[65] J.M. Schumann. *Automated Theorem Proving in Software Engineering*. Springer, 2001.

[66] N. Shankar. Little engines of proof, 2002. Invited talk, 3rd FLoC, and course notes, Fall 2003, http://www.csl.sri.com/users/shankar/LEP.html.

[67] R.E. Shostak. An algorithm for reasoning about equality. *C. ACM*, 21(7):583 585, 1978.

[68] R.E. Shostak. Deciding combinations of theories. *J. ACM*, 31(1):1 12, 1984.

[69] P. Simons, I. Niemela, and T. Soininen. Extending and implementing the stable model semantics. *Artif. Intell.*, 138(1-2):181 234, 2002.

[70] R. Socher-Ambrosius and P. Johann. *Deduction systems*. Springer, 1997.

[71] SPASS. http://spass.mpi-sb.mpg.de/, 2006.

[72] M. E. Stickel, R. Waldinger, M. Lowry, T. Pressburger, and I. Underwood. Deductive composition of astronomical software from subroutine libraries. In *CADE-12*, volume 814 of *LNAI*, pages 341 355. Springer, 1994.

[73] Mark E. Stickel. A Prolog technology theorem prover: new exposition and implementation in Prolog. *Theor. Comput. Sci.*, 104:109 128, 1992.

[74] T. Tammet. Gandalf. *J. Automat. Reason.*, 18(2):199 204, 1997.

[75] L. Vigneron. Automated deduction techniques for studying rough algebras. *Fundamen. Inform.*, 33:85 103, 1998.

[76] M. Wooldridge, M. Fisher, M.-P. Huget, and S. Parsons. Model checking multi-agent systems with mable. In *AAMAS*, pages 952 959. ACM, 2002.

[77] L. Wos, R. Overbeek, E. Lusk, and J. Boyle. *Automated Reasoning: Introduction and Applications*. McGraw-Hill, 2nd edition, 1992.

[78] H. Zhang. SATO: an efÞcient propositional prover. In *CADE-14*, volume 1249 of *LNAI*, pages 272 275. Springer, 1997.

[79] L. Zhang and S. Malik. The quest for efÞcient boolean satisÞability solvers. In *CADE-18*, volume 2392 of *LNAI*, pages 295 313. Springer, 2002.