1 Introduction

A central problem in automated reasoning is to determine whether a conjecture \( \varphi \), that represents a property to be verified, 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 finding suitable formalisms for \( S \) and \( \varphi \) 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 affirmatively, by finding a proof of \( \varphi \) 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 \( \varphi \) (in symbols: \( S \models \varphi_\sigma \), for all ground substitutions \( \sigma \)).

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

Otherwise, given assumptions \( S \) and conjecture \( \varphi \), one can try to answer negatively, disproving \( \varphi \), by finding 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 rst-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 first 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 formulæ;
2. **Saturation methods** extract models from the finite set of formulæ generated by a failed refutation attempt; and
3. **Simultaneous methods** search simultaneously for a refutation or a model of the given set of formulæ.

In higher-order logics, that allow universal and existen- tial 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 rst-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, if they exist, but there is an in finite search space of logical consequences where to look for a contradiction. A machine can explore only a finite part of this in finite space, and the challenge is to find a proof using as little resources as possible. A fundamental insight was the recognition that the ability to detect and discard redundant formulæ is as crucial as the ability to generate consequences of given formulæ. In addition to standard expansion inference rules of the form

$$\frac{A_1 \ldots A_n}{B_1 \ldots B_m}$$

which add inferred formulæ $B_1, \ldots, B_m$ to the set of known theorems, that already includes the premises $A_1, \ldots, A_n$, contemporary inference systems feature contraction rules, that delete or simplify already-inferred theorems. The double-ruled inference rule form

$$\frac{A_1 \ldots A_n}{B_1 \ldots B_m}$$

means that the formulæ $(A_i)$ 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 simplification rule.

An expansion rule is sound if what is generated is a logical consequence of the premises $\{A_1, \ldots, A_n\} \models (B_1, \ldots, B_m)$. Classical examples are resolution and paramodulation. A contraction rule is sound if what is removed is a logical consequence of what is left or added $\{B_1, \ldots, B_m\} \models \{A_1, \ldots, A_n\}$. Classical examples are subsumption and equational simplification 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 formulæ is inconsistent. The challenge is dealing with combination without endangering completeness [36, 7, 8, 18]: a key ingredient is to order the data (terms, literals, clauses, formulæ, 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 finite and decision procedures are known. Decidability may stem from imposing restrictions on

1. the logic,
2. the form of admissible formulæ for $S$ or $\varphi$, or
3. the theory presented by the assumptions in $S$.

An example of Case 1 is the guarded fragment of rst-order logic, which propositional modal logic can be reduced to. The most prominent instance is propositional logic, whose decidable satisfiability 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-Schönfinkel class admits only sentences in the form

$$\exists x_1, \ldots, x_n \forall y_1, \ldots, y_m . P(x_1, \ldots, x_n, y_1, \ldots, y_m)$$

where $P$ is quantifier-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 little engine of proof for each theory [66], by building the theory’s 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 practical, 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 artificial 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 defines 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

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

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 formulæ is expanded:
\[ S \supset S' \]

If it selects a contraction rule, the set of formulæ is contracted:
\[ S \preceq S' \quad S' \prec \text{mul} S \]

where \( \prec \text{mul} \) is the multiset extension of a well-founded ordering on clauses. Strategies that employ well-founded orderings to restrict expansion and contraction are called ordering-based. Ordering-based strategies with a contraction-first 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 \( \neg \psi \). 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, matchings and connections, all eventually understood in the context of clausal normalform tableaux.

The picture is completed by instance-based strategies, that date back to Gilmore’s 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 classification, 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 rst-order prover [12] or a decision procedure [59], to dispatch a rst-order conjecture or a decidable subproblem, respectively.

2.4 Applications

Its intrinsic difficulty notwithstanding, automated reasoning is important in several ways. Its direct applications, such as hardware/software verification 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 veriﬁcation of cryptographic protocols, message-passing systems and software speciﬁcations [72, 65]. Furthermore, automated reasoning contributes techniques to other ﬁelds of artiﬁcial 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].

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 rst-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 notion of nonmonotonic logics. Various approaches have been used to do nonmonotonic reasoning, based on fixpoint 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 speciﬁc 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 efﬁcient 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

Anno III, N° 1/2, Marzo-Giugno 2006
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, which 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 Smoodels [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 \( \varphi \) 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 \( \varphi \) 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 the 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) logic 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 entities, 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 \( p(s) \rightarrow q(\mathrm{result}(a, s)) \) 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 \( \Diamond(p \rightarrow [a]q) \) has the same meaning as the previous one (\( \Box \varphi \) means that \( \varphi \) 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 efficient way what are the unents 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 unent 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 completion construction due to Reiter.

Among other formalisms we can mention the event calculus, an extension of logic programming with explicit time points, and theent 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 defined, 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 agents’ 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 verifi able semantics, i.e. a semantics grounded on the computational models. Given a formal semantics, it is possible to determine what it means for an agent to be respecting the semantics of the communicative action when sending a message. Verification 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 quantified, linear time temporal BDI logic. Properties of MABLE programs can be verified by means of the SPIN model checker, by translating BDI formulas into the form used by SPIN.

The problem of verifying agents’ compliance with a protocol at runtime is addressed in [1]. Protocols are specified in a logic-based formalism based on Social Integrity Constraints, which constrain the agents’ observable behavior.

The paper present a system that, during the evolution of a society of agents, verifies the compliance of the agents’ behavior to the protocol, by checking fulfillment of expectations and violation of expectations.

Another approach for the specification and verification of interaction protocols is proposed in [32] using a combination of dynamic and temporal logic. Protocols are expressed as regular expressions, (communicative) actions are specified by means of action and precondition laws, and temporal properties can be expressed by means of the until operator. Several kinds of verification problems can be addressed in that framework, including the verification of protocol properties and the verification that an 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 be 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, Reasoning on the Web with Rules and Semantics, 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**
