Automated Reasoning

Matthias Knorr

Synomyms

Automated Deduction

Glossary

backward chaining: an inference method that searches backwards starting from the goal to be proven.
complete: a complete algorithm returns all correct answers (but may also return others).
deduction calculus: a formal system in mathematical logic to express logical derivations, i.e., proving that a conclusion can be inferred from a given set of assumptions.
direct proof: starts from the known assumptions, and proves the desired conclusion applying rules of inference in a step-wise manner.
forward chaining: an inference method that searches for solutions applying a set of inference rules to the available data.
indirect proof: assumes that the desired conclusion is wrong, and aims at proving that this together with the known information leads to a contradiction.
resolution: a rule of inference used to show that some goal/objective is not derivable.
SAT solver: program that given a propositional formula returns whether it is satisfiable.

Matthias Knorr
NOVA LINCS, Departamento de Informática, Universidade Nova de Lisboa, 2829-516 Caparica Portugal e-mail: mkn@fct.unl.pt

sound: a sound algorithm returns only correct answers (but not necessarily all of them).

Definition

Automated reasoning aims at building computing systems that are able to apply logical reasoning in an automated manner.

Introduction

Reasoning, that is, the process of thinking about something in a rational way in order to form a conclusion or judgment, is arguably among the key characteristics of human nature. It is therefore not too surprising that being able to reason in an automatic manner figured among the main objectives of early research in artificial intelligence, whose ultimate overall goal can be summarized as developing an intelligent agent capable of simulating human behavior. Despite initial success - for example, with the development of formal logic, it became indeed possible to create mathematical proofs automatically for the first time during the 1950s - it turned out that developing such a universal intelligent agent was overwhelmingly difficult due to the sheer amount of information it would need to process. Nevertheless, research on single capabilities such as natural language processing, knowledge representation, automated reasoning, and machine learning continued and eventually grew into fields of its own.

In the case of automated reasoning, research has been developed mainly in automated theorem proving, that is, providing formal proofs in an automated way (and semi-automated for the sub-field of interactive theorem proving) as well as automated proof checking. The former has been applied in mathematics itself as well as in finding proofs/solutions to problems in Artificial Intelligence, e.g., in planning and search, while the latter has been of increasing importance for verification of software in particular in critical areas, such as life sciences or aviation. Today, automated reasoning is a mature area of computer science and mathematical logic with strong ties to Artificial Intelligence and Theoretical Computer Science.

Key Points

- Given a description of the known information and of a goal, i.e., the objective intended to be derived, the idea is to find a formal proof of the goal based on the known information in an automated way.

- To formally describe the known information and the goal, a language/formalism has to be chosen that is adequate to represent the necessary information.
- In addition, a deduction calculus has to be selected that fits the language chosen to represent the necessary information and whose characteristics are most appealing to the problem at hand, balancing for example questions such as: how fast do we want to obtain an answer to the question of whether the goal follows from the known information; and do we always want to obtain a certain answer or do we also admit that no answer is found at the expense of faster answers in the majority of the cases.

Historical Background

Automated reasoning started in the 1950s with the attempt to capture reasoning in the mathematical domain building on the then newly available serious computing power. Basically, two possible kinds of approaches presented themselves. One was trying to capture the human capabilities to create proofs and write programs that would emulate this process. Another would try to build on logics and logical reasoning and base algorithms on these. As pointed out by [Davis(2001)], both approaches had their shortcomings. For the former, available information on the creative thought-process of mathematicians is still today rather sparse. For the latter, the well-known impossibility results by Church and Turing implying that no algorithm might exist for a particular problem as well as the potential huge combinatorial explosion proved to be major obstacles. Ultimately, both paths led to significant advances, and, despite long-lasting controversy and polemics between supporters of both groups, often their combination provided the most successful approaches. In the following, we only briefly discuss some of the very early successes.

In 1954, Martin Davis programmed the algorithm developed by M. Presburger (in 1929) that would be able to determine whether a given sentence in the first-order theory of addition in the arithmetics of integers was true or not. It was not performing very well, but was, for example, able to prove for the first time automatically that the sum of two even numbers is even.

Newell, Shaw, and Simon reported in 1957 on their Logic Theory Machine, a program with which they tried to emulate the process of a person searching for proofs in the propositional calculus of Whitehead and Russell's famous *Principia Mathematica*, a three-volume work on the foundations of mathematics, originally published in between 1910 and 1913, that attempts to describe a set of axioms and inference rules from which all mathematical truth would be provable. Newell, Shaw and Simon provided pioneering work introducing explicitly the concepts of *forward* and *backward chaining*, the generation of useful subproblems, and seeking substitutions that produce desired matches. Their algorithm was however incomplete, a limitation overcome by Wang and Gao's proof system in 1987, which built on deduction calculi developed by Gentzen in the 1930s.

The Davis-Putnam algorithm from 1960 allowed for checking the validity of first-order logic formulas. In fact, its refinement from 1962, the Davis-Putnam-Logemann-Loveland algorithm which requires only a linear amount of memory in the worst case, formed the basis for many systems utilizing *resolution* as introduced subsequently by Robinson in 1965, as well as for the most efficient complete *SAT solvers* as of today.

More information can be found in [Robinson and Voronkov(2001)], not only on these early landmarks, but also on the plethora of approaches to automated reasoning that emerged during the next decades turning automated reasoning into a mature field of its own.

Main Text

Automated reasoning aims at creating algorithms that allow the application of logical reasoning. Many such algorithms exist nowadays, which are based on quite different principles, and the main objective of the following text is to describe the general considerations and characteristics of the *deduction calculi* on which these algorithms are based, illustrating these concepts with resolution in propositional clause logic, and also overview (other) examples.

Describing the Problem

Put in general terms, a program that performs automated reasoning solves problems that can be represented by a set of statements corresponding to the information already available to the program, also termed the *problem's assumptions*, and a statement corresponding to the question being asked, also termed the *problem's conclusion*. Solving such a problem then means providing a proof of the conclusion based on the given assumptions in an algorithmic way.

An important choice to make when designing such programs is to determine the language which is used to represent the problem taking into account, for example, how the problem is represented in actual algorithms, the internal representation of it in the program itself, and even the way the problem and possible solutions are presented to the user. Many different formalisms are available for this purpose such as, for example, first-order predicate logic, propositional logic or clause logic, to name just a few of the more common ones, and the choice is ultimately influenced by the *problem domain*, that is, the class of problems the program is aiming to solve. Often, a specific restriction of such very general formalisms can be used due to the problem domain in consideration, which allows for the usage of so-called *domain axioms*, that is, axioms that only hold within a specific domain and can be leveraged when finding proofs in this domain. For example, when using first-order logic to

reason in the mathematical domain, known axioms of mathematics can be added as domain axioms and used subsequently for deriving desired conclusions.

Here, we briefly describe propositional clause logic, a subset of clause logic. It builds on a set of propositional atoms, similar to propositional logic. A *literal* is either a propositional atom p or its negation $\neg p$, and two literals are complementary if one is the negation of the other, e.g., p and $\neg p$ (where double negation is contracted as usual, that is, we always omit any two immediate negation signs). Then, a *clause* is a finite (possibly empty) set of literals $l_1 \ldots, l_n$, denoted $[l_1, \ldots, l_n]$, representing a disjunction of literals (without repetition). The empty clause in particular, denoted $[]$, does not contain any literals, and is therefore unsatisfiable, that is, always false. The associated problem domain is propositional logic, since any propositional formula can be transformed into *conjunctive normal form (CNF)*, which corresponds precisely to a conjunction of disjunctions, i.e., a conjunction of clauses.

Deduction Calculi

Closely connected to the question of the considered problem domain is also the question of which deduction calculus to use, and again a plethora of approaches is available.

As such, a *deduction calculus* comprises a set of logical axioms and a set of inference rules for deriving new information from known information (including the initial assumptions as well as already derived information). More precisely, given the problem's conclusion α and the set Γ constituted by the set of logical axioms of the calculus, the possibly available domain axioms, and the problem's assumptions, solving the problem means showing that α is derivable from Γ, written $\Gamma \models \alpha$.

As already said, different calculi apply different strategies to achieve this objective. Namely, some try to establish a *direct* step-wise proof of α (from Γ), that is, Γ proves α, written $\Gamma \vdash \alpha$. Alternatively, an *indirect* proof of $\Gamma \models \alpha$ can be established by showing that $\Gamma \cup \neg\{\alpha\} \vdash \bot$, that is, given Γ and the negation of the conclusion α we obtain a contradiction, and therefore α cannot be false, and, hence, α follows from Γ.

Establishing such a proof can also be done in two different ways. We can either explicitly start from Γ ($\Gamma \cup \{\neg\alpha\}$ for indirect proofs) and try to establish that α ($\bot$) holds or we can begin with the conclusion α ($\bot$) and try show that there are deduction rules that eventually allow us to trace the conclusion back to Γ ($\Gamma \cup \{\neg\alpha\}$ resp.). The former approach is called *forward chaining* and the latter *backward chaining* and both have been used extensively.

Important characteristics of deduction calculi are soundness and completeness. *Soundness* determines that the returned answers are always correct. More precisely, for a direct calculus, if $\Gamma \vdash \alpha$ can be verified, then $\Gamma \models \alpha$ holds. For an indirect calculus on the other hand, it means that if $\Gamma \cup \{\neg\alpha\} \vdash \bot$ can be verified, then $\Gamma \models \alpha$ holds. *Completeness* means that the correct answer is always returned. For a direct calculus, this corresponds to ensuring that if $\Gamma \models \alpha$, then $\Gamma \vdash \alpha$. For an

indirect calculus, this corresponds to ensuring that if $\Gamma \models \alpha$, then $\Gamma \cup \{\neg\alpha\} \vdash \bot$. The latter is also termed *refutation complete*.

Decidability and complexity are naturally also of importance for deduction calculi. A deduction calculus is *decidable* if there is a algorithm that, for any Γ and α, answers the question "$\Gamma \models \alpha$" with "Yes" or "No" in finite time. There are undecidable calculi, in which case the focus on decidable fragments becomes interesting w.r.t. implementation. This is also prominently present w.r.t. restrictions on the problem problem, e.g., *description logics (DLs)* [Baader et al(2010)] are commonly decidable fragments of first-order logic, that is, deduction calculi tailored towards DLs are usually decidable whereas general first-order calculi are not. Notably, due to the well-known impossibility results on decidability, calculi that are incomplete are not uncommon and accepted as sometimes unavoidable given the nature of some of the problems considered. On the other hand, unsound calculi are in general not desired at all.

In addition, the efficiency of such a calculus can be measured by the *computational complexity* in terms of time and space. Since many relevant calculi are unfortunately undecidable and are of high computational complexity, finding a trade off between the power of a deduction calculus and its efficiency is often of major importance, which by itself has lead to a large body of research on such calculi.

Propositional Resolution

Deduction calculi that are based on resolution are in fact quite popular. Here, we consider a simplified version restricted to the propositional case utilizing propositional clause logic as described before to ease the understanding.

Essentially, the rule of resolution takes two clauses C_1 and C_2 that both contain one of two complementary literals (w.l.o.g. suppose that C_1 contains p and C_2 $\neg p$). Then the result of resolving C_1 and C_2 is the clause obtained by creating the clause containing all literals from C_1 except p and all literals from C_2 except $\neg p$:

$$\frac{C_1 \qquad C_2}{C_1 \setminus \{p\} \cup C_2 \setminus \{\neg p\}}$$

There are no other logical axioms to be considered in propositional resolution nor any domain axioms in the general case we consider here. Resolution only requires the assumptions Γ and the negated conclusion α in conjunctive normal form (CNF - see the subsection on describing the problem) as input, and then, by means of resolution, we try to derive $\bot$. If we succeed, then α follows from Γ, otherwise it does not. Thus, resolution follows the indirect approach and builds on forward chaining. Also, it has been shown that resolution is sound and complete for the propositional case (in fact refutation complete), and that it is decidable, but with a high exponential complexity in the worst case.

Suppose we know that the following assumptions hold: $\neg a \vee b$, $\neg b \vee c$, $a \vee \neg c$ and $a \vee b \vee c$ and we want to show that $\alpha = a \wedge b \wedge c$ follows. The assumptions are already a set/conjunction of disjunctions, i.e., in CNF, and $\neg\alpha$ becomes $\neg a \vee \neg b \vee \neg c$ in CNF. We can thus apply resolution.

(1)	$[\neg a, b]$	assumption
(2)	$[\neg b, c]$	assumption
(3)	$[a, \neg c]$	assumption
(4)	$[a, b, c]$	assumption
(5)	$[\neg a, \neg b, \neg c]$	assumption ($\neg\alpha$)
(6)	$[\neg a, \neg c]$	resolution on b – (1) and (5)
(7)	$[\neg c]$	resolution on a – (3) and (6)
(8)	$[\neg b]$	resolution on c – (2) and (7)
(9)	$[\neg a]$	resolution on b – (1) and (8)
(10)	$[b, c]$	resolution on a – (4) and (9)
(11)	$[c]$	resolution on b – (8) and (10)
(12)	$[]$	resolution on c – (7) and (11)

In principle, there is no imposed order on which atom to resolve first, nor which clauses to use for that. Rather, all possible combinations are as such correct. Still, this creates an unnecessarily large search space and many options are in fact often not helpful. E.g., we could have started above resolving a using (4) and (5) resulting in $[b, \neg b, c, \neg c]$, to which we could have applied then (2). But then, any of the subsequent steps using this clause would contain a or $\neg a$ again. Thus, search strategies are necessary to help reduce the search space. Various such strategies exist, among them for example hyperresolution that allows combining several resolution steps into a single one, but in particular linear resolution. *Linear resolution* always performs resolution using the most recently obtained resolvent. This yields a simple linear structure of the proof (evidently considerably reducing the research space), and, as shown in the literature, without jeopardizing refutation completeness. Other principles, such as unit resolution, that is, in each step one of the resolved clauses has to be a literal, or input resolution, that is, in each step one of the resolved clauses has to come from the original set, do not, despite considerable gains in efficiency. Sometimes, this is acceptable if we are willing to trade completeness for speed of derivation. The example derivation above does in fact apply linear resolution.

We briefly mention that the essential difference in first-order resolution is due to the presence of variables, as resolution is no longer performed on matches of complementary literals, but rather applying unification, that is, find substitutions mapping variables to terms (including other variables) so that two literals (potentially still with variables) become complementary. This raises the difficulty of finding proofs quite a bit also because it introduces a further rule that uses unification to make two literals (in the same clause) match, but it turns out that the resulting calculus is sound and complete as well. Finally, we mention, that *Logic Programming* and *Prolog* builds on a special form of resolution, where the admitted clauses are restricted to those containing only one negated literal. The specific calculus is in

fact backward chaining and sound, but not complete in the first-order case. We refer to the recommended reading pointers for further details.

Other Deduction Calculi

We briefly mention other important deduction calculi here and refer for a detailed comparison and analysis between these to [Bibel and Eder(1993)], but also to [Robinson and Voronkov(2001)] for an even wider scope.

We start with the *sequent calculus* that was developed by Gentzen with the aim of formally replicating mathematical proofs. It uses a relatively simple set of axioms and a relatively large number of deduction rules and applies a direct and forward chaining approach. The idea is starting from tautological axioms to create a proof using conditional tautologies, called sequents with an antecedent and consequent, where the consequent is necessarily true whenever the antecedent is. The sequent calculus is sound and complete for first-order logic.

Natural deduction also due to Gentzen is closely related, but rather intended for emulating the proof construction in a more human-like way. It uses no logical axioms yet a larger set of deduction rules to create proofs in a direct and forward chaining manner. Due to its intended purpose, natural deduction can only rarely be found in actual systems.

Closely related to the sequent calculus is also the method of *tableaux*. Proofs in it can actually be associated to proofs in sequent calculus turned upside-down. Tableaux use forward chaining and are indirect, trying to refute a given negated goal. Often, also the term *analytical* is used to describe that a tableau proof decomposes an initially given formula until either a contradiction ($\bot$) has been found in each of the branches of the proof tree, or no further rules are applicable. In the former case, the desired proof has been found, in the latter, each *open* branch, i.e., a branch without a contradiction, corresponds to a counterexample to the desired conclusion. Tableaux are sound and complete for classical first-order logic, but this is no longer necessarily the case for more expressive logics, such as modal logics, to which the tableaux method has also been frequently applied.

The *matrix connection method* is also an indirect calculus that rearranges the information, both the assumptions and the negation of the desired conclusion, into a matrix, and then aims at trying to verify for all paths through the matrix whether there is a contradiction, i.e., for the propositional case whether in each path there is a pair of complementary literals.

As already hinted, many calculi beyond first-order logic exist. Prominently, in higher-order logic, the typed λ-calculus [Dowek(2001)] and functional programming languages are often used. In the case of non-classical logics, several ways to tackle the problem are common. Either the dedicated non-classical deduction calculus is implemented or the problem is reformulated such that a classical calculus can be used, or even the non-classical logic is embedded into a first-order frame-

work which makes first-order methods applicable. We refer to the further reading material for more details.

Key Applications

Automated reasoning techniques have been applied in many different scenarios and we briefly discuss major ones in the following.

- Theorem proving in mathematics and formal logics has obviously benefited a lot from automated reasoning as many different tools are nowadays available to support finding proofs and automating tedious aspects of generating proofs. While such systems are still often not sufficient for complicated problems, interactive theorem proving provides an intermediate solution in which the user interactively guides the search for the proof.
- Software verification has become of major importance for critical services namely wherever malfunctioning software could result in, e.g., loss of human life, threats to security, or unauthorized access to sensitive information. Formal verification can increase the quality of verification, basically, by creating a number of conditions the software must satisfy and prove that it actually does. This has been applied, e.g., in nuclear engineering, aerospace software, and encryption verification.
- Similar to software verification, hardware verification has become of increasing importance where, e.g., design flaws would cause huge losses of money or could cause critical failures. Hardware verification ensures that hardware works correctly as specified and different techniques/deduction calculi of those previously presented have been used in practice.
- Automated reasoning has also been extensively used in applications in the field of artificial intelligence in the wider sense, e.g., in Logic Programming or the Semantic Web. In fact, reasoning in the languages standardized by the World Wide Web Consortium (W3C) such as OWL and RIF heavily relies on automated reasoning techniques. The main idea is that reasoners are used to return answers in problem solving, planning and search with applications in areas as different as systems configuration, circuit verification, health sciences and bioinformatics, and information and decision support systems.

Future Directions

As the number of produced scientific results in mathematics and formal logics keep ever growing, it becomes increasingly more difficult, if not impossible, to be aware of all results. The combined use and interchange of automated/interactive theorem provers and information obtainable on web pages such as *polymath* and *mathover-*

flow, which is currently created based on human interaction only, could help overcome this problem [Martin and Pease(2013)].

Another important point for future research lies in the increasing use of decision support systems, personal assistants, for example on smart phones. One problem with such assistants is that it is not always clear how reliable the provided information actually is. Automated reasoning could help overcome this problem enabling users to verify the truth of provided information using proofs building on annotated data, certified sources of information on the Web and even user's preferences, e.g., with respect to reliability of previously obtained information.

Cross References

Description Logics, OWL, RIF: The Rule Interchange Format,

Acknowledgments

The author acknowledges the support of FCT under strategic project NOVA LINCS (UID/CEC/04516/2013) and grant SFRH/BPD/86970/2012.

References

[Baader et al(2010)] Baader F, et al (eds) (2010) The Description Logic Handbook: Theory, Implementation, and Applications, 3rd edn. Cambridge University Press

[Bibel and Eder(1993)] Bibel W, Eder E (1993) Methods and calculi for deduction. In: Gabbay DM, Hogger CJ, Robinson JA (eds) Handbook of Logic in Artificial Intelligence and Logic Programming (Vol. 1), Oxford University Press, Inc., New York, NY, USA, pp 68–182, URL http://dl.acm.org/citation.cfm?id=185728.185745

[Davis(1983)] Davis M (1983) The prehistory and early history of automated deduction. In: Siekmann J, Wrightson G (eds) Automation of Reasoning. Classical Papers on Computational Logic, Springer, pp 1–28

[Davis(2001)] Davis M (2001) The early history of automated deduction: Dedicated to the memory of Hao Wang. In: [Robinson and Voronkov(2001)], pp 1–28

[Dowek(2001)] Dowek G (2001) Higher-order unification and matching. In: [Robinson and Voronkov(2001)], pp 1009–1062

[Martin and Pease(2013)] Martin U, Pease A (2013) Mathematical practice, crowdsourcing, and social machines. In: Carette J, Aspinall D, Lange C, Sojka P, Windsteiger W (eds) Intelligent Computer Mathematics - MKM, Calculemus, DML, and Systems and Projects 2013, Held as Part of CICM 2013, Bath, UK, July 8-12, 2013. Proceedings, Springer, Lecture Notes in Computer Science, vol 7961, pp 98–119

[Portoraro(2014)] Portoraro F (2014) Automated reasoning. In: Zalta EN (ed) The Stanford Encyclopedia of Philosophy, winter 2014 edn, http://plato.stanford.edu/archives/win2014/entries/reasoning-automated/

[Robinson and Voronkov(2001)] Robinson JA, Voronkov A (eds) (2001) Handbook of Automated Reasoning (in 2 volumes). Elsevier and MIT Press

Recommended Reading

- [Robinson and Voronkov(2001)] describes exhaustively all aspects and major approaches to automated reasoning in a detailed manner.
- [Bibel and Eder(1993)] is a detailed overview and comparison on many of the different deduction calculi.
- [Portoraro(2014)] also provides a rather compact overview on the topic with further pointers to the literature.
- [Davis(1983)] presents an account on the early history of automated reasoning.
- The proceedings of the International Joint Conference on Automated Reasoning (IJCAR) and of the International Joint Conference on Automated Deducation (CADE) as well as the Journal of Automated Reasoning are the main venues for publication on automated reasoning.