Automated Deduction: A Technological Point of View

Peter Baumgartner, Ingo Dahn, Jürgen Dix, Ulrich Furbach,
Micha Kühn, Frieder Stolzenburg, Bernd Thomas

AI Research Group, Universität Koblenz, Rheinau 1, D-56075 Koblenz, E-mail: ki-inf@uni-koblenz.de

There are recent successes of automated deduction techniques in various application domains, such as mathematics, *classical* and *nonmonotonic* logics, diagnosis, planning and within software engineering. We will briefly review some of them and observe that these successes have been made only after a careful analysis of the application domain as well as the deduction system under consideration. The purpose of this article is to argue that automated deduction systems can be usefully applied in practice, but it is necessary to have available a variety of deduction methods, to understand their properties and their computational power in order to tailor them for the application under consideration.

In the early days of automated deduction, research concentrated on the development of general purpose deduction systems. “Applications” were toy examples, which did not scale up to realistically sized problems. Nowadays, the theorem prover community discovers applications again. The key to success quite often is the specific knowledge of the application domain which is used to optimise the deduction systems, i.e. the knowledge is used to guide the search for a proof or a model.

Indeed, there are some recent impressive results which show that such deductive systems are even able to handle realistic problems. Domains to be mentioned here include *mathematics* (see Sections 1 and 4.1), *planning* [18] (see Section 3.1), *model checking* (see Section 3.2), *diagnosis* (see Section 3.3), *software reuse* [22] and *verification* (see Section 3.4), or *view deletion in databases* [2]. Another realistically sized problem solved by the Protein prover [5] was the analysis of *banking fees* rule sets, which was successfully tackled with constraints [44] (see Section 3.5).

1 A Success of Automated Deduction

Recently, W. McCune proved with the automated theorem prover EQP that each *Robbins algebra* satisfies S. Winker’s second condition. This implied, that each Robbins algebra is Boolean and settled a problem which had been open for more than 60 years. This success even reached the New York Times [36].

H. Robbins’ conjecture can be formulated very briefly: In 1933, E.V. Huntington presented a basis for Boolean algebras consisting of the associative and commutative law for union and the equation:

$$\overline{\overline{x} \cup y} \cup \overline{\overline{x} \cup \overline{y}} = x$$

Shortly thereafter, H. Robbins conjectured that E.V. Huntington’s equation can be replaced with a simpler one:

$$\overline{\overline{x \cup y} \cup \overline{x \cup \overline{y}}} = x$$

Many mathematicians—among them the famous logician A. Tarski—have worked in order to decide H. Robbins’ conjecture. The proof that solves the problem was found on October 10, 1996, by the theorem prover EQP developed at Argonne [33]. EQP is similar in many ways to the more well known program Otter. The main differences are that EQP has associative-commutative (AC) unification, is restricted to equational logic, and offers paramodulation strategies to handle equality. So, also in this first example, domain specific techniques are exploited.

W. McCune used the output of EQP to guide his deduction system Otter in order to produce a proof. Otter’s proof contained fairly complex terms which were hard to understand or even to print in a readable format. Proofs in mathematics are not a value in their own right. Their main purpose is to help humans to understand complex phenomena. Therefore, several authors tried to find simpler representations of this proof—for proof checking as well as to gain deeper insights into the nature of Robbins algebras.

Therefore

$$\forall(x_1,x_2,x_3,x_4)\ \overline{x_3 \cup x_1 \cup x_2 \cup x_4} \neq x_3 \overline{\cup x_4} \quad \textbf{(1)}$$

Because of commutativity and by Robbins axiom

$$\forall(x_1,x_2)\ (x_1 \rightarrow x_2) \cap (x_2 \cup x_1) = x_2.$$

Hence by commutativity

$$\forall(x_1,x_2)\ (x_2 \cup x_1) \cap x_1 \rightarrow x_2 = x_2. \quad \textbf{(2)}$$

Because of commutativity and by Robbins axiom

$$\forall(x_1,x_2)\ (x_1 \cup x_2) \cap x_1 \rightarrow x_2 = x_2. \quad \textbf{(3)}$$

Fig. 1: A part of the ILF output

A system called ILF, developed by I. Dahn within the German focus program on Deduction, was able to translate automatically the above depicted proof in a way that mathematicians can easily read and check (see Fig. 1). Ultimately, the term depth in the proof could be reduced from 14 to 4 [17]. Using ILF, it is very simple to analyse a proof and to change its presentation. The first presentations generated by ILF reduced the complexity of formulae by introducing abbreviations which are commonly used for Boolean algebras. This facilitated a plausibility check of the proof. Then, automated proof transformations helped to separate the mathematical contents of that proof from the formal details. A system which is also suited for this purpose is the Omega system [10].

It is no question that this proof has been found by means of a calculus which has nothing to do with the way human mathematicians find or communicate proofs.

What we would like to point out with the above example is, that even if it is not the case that reasoning is done human-like, one can aim at a translation of machine-oriented into human-like reasoning. In this way, the increased power of automated reasoning systems is becoming a new tool for the thinking human.

In [46]—also based on the experience with the solution of H. Robbins' problem—L. Wos emphasises the different natures of human and computer reasoning. He provides hints how to combine both successfully to bring new kinds of applications of automated deduction within reach. This is similar to our thesis that will be exemplified in the rest of the text: Automated deduction can be applied successfully in practical applications by using domain-specific techniques.

2 Theorem Proving Techniques

To apply automated deduction to real-world problems, a sound knowledge of the underlying deductive techniques and their specifics is as important as the knowledge of the application area itself. We will concentrate here on two widespread mainstreams of automated deduction with quite impressive practical results in recent years: classical first-order theorem proving, and nonmonotonic reasoning with its importance for logic programming, databases and other applications. Even in this marked area, there are numerous different methods with individual strengths for certain tasks. We will sketch some of them along with fundamental properties. Examples of useful combinations and refinements of these approaches that adapt deductive techniques to specific fields of application will follow in Section 3.

2.1 Classical Reasoning

High-performance automated theorem provers usually are based on refutations. As in the case of Otter [32], saturation-based resolution calculi are used to generate consequences until a contradiction is found, whereas Setheo [25] or Protein [5] apply variants of model elimination [30], a tableau-based, goal-driven proof procedure. Saturating proof procedures usually are *proof-confluent*, i.e. it is never necessary to undo parts of the derivation process, whereas model elimination procedures employ some kind of backtracking based on *iterative deepening*.

On the other hand, tableau-based procedures often are much less space-consuming and thus handle clause sets where resolution provers might run out of memory. Furthermore, the goal-directed nature of model elimination shows some similarity to *logic programming* and, indeed, the *restart model elimination* variant [4] enables a procedural reading of disjunctive clauses, as in *Prolog* for Horn clauses. It has been demonstrated that these approaches are able to deal with interesting problems, and indeed there is an increasing number of applications of these high-performance automated theorem provers (see Sections 1 and 3).

In contrast to the above mentioned systems which are optimised for finding *refutations*, there is also work on model-based automated reasoning. Here, the idea is to find a proof by means of bottom-up model generation. One of the first systems, which shows that a very simple proof procedure based on model generation is able to outperform (in some cases) refutation-based provers, was SATCHMO [31]. Its main drawback—the limitation to range-restricted clause sets—has been overcome by the hyper-tableau calculus [6].

Model generating calculi are convenient if one is not interested in the unsatisfiability of formulae, but in satisfying interpretations which may be further specialised. Questions of this type are tackled by model-based reasoning in the areas of *mathematics* [23], *automated planning* [28] or *diagnosis* (see Section 3.3). Models may also serve as counterexamples for formulae assumed to be unsatisfiable, useful in domains like e.g. *software verification* where it is not guaranteed beforehand that given specifications are correct (see also Section 4.1).

2.2 Nonmonotonic Reasoning

Originally, nonmonotonic reasoning was intended to provide a *fast* but *unsound* approximation of classical reasoning in the presence of incomplete knowledge. But the complexity results of nonmonotonic logics are located on the second level of the polynomial hierarchy [26], so they are even more complicated than the classical entailment problem. Therefore subsystems of general nonmonotonic logics on syntactically restricted classes of formulae have been investigated, in particular those that are based on logic programming and extend the classical class of Horn programs: either by allowing (1) default negation (normal programs), or (2) disjunction (positive disjunctive programs), or (3) both negation and disjunction (general disjunctive programs).

According to these different classes, various semantics are possible. In Section 3.3 we will see that minimal models are a good choice for diagnosis tasks. In fact, these minimal models stem from theoretical work in deductive databases [35]: the semantics is called *generalised closed world assumption*. Concerning default negation, there are various other possibilities (see [12]). Most notably the well-founded semantics WFS and the stable model semantics Stable. WFS is *consistent* (model always exists), permits *goal-directed* computation and has attractive *complexity* (quadratic in the number of atoms) and it can be computed efficiently using alternating fixpoints. Stable can become inconsistent (programs may have no stable models) and answering queries cannot be restricted to the call-graph below that query.

While Stable is very closely related to default logic, WFS can be seen as an approximation to the intersection of all stable models: an atom that is false in one stable model but true in another one is undefined according to WFS. Let us mention two very interesting systems for computing WFS and Stable for non-disjunctive programs. They constitute the current success stories of logic programming and nonmonotonic reasoning. For the general disjunctive first-order case, a calculus is described in [20].

Firstly, there is *XSB*, a full-fledged programming system, which realizes WFS using tabling (developed from D. Warren and his group [41]). It works for full first-order programs, but, obviously, termination can only be guaranteed in special cases. Nevertheless XSB is terminating for many programs where usual Prolog systems fall into an infinite loop. Tabling operates on low level data-structures but a declarative description has been given in [11].

Secondly, there is *smodels*, which works for range-restricted function-free normal programs [37, 19]. It is written in C++ and can handle realistic size programs (tens of thousands of ground rules) with a potentially large number of stable models.

3 Applications

Many AI critics say: *If it works, it ain't AI.* This section will prove the opposite: There *are* several real world problems successfully solved by deduction-based systems. These applications owe their success to the well founded theoretical background and active research done on the field of deduction, especially to calculi refinements motivated by special problem domains. We will shortly describe how these applications work and which theoretical methods have been used to tackle them.

3.1 Planning

Developing automated methods for generating and reasoning about plans and schedules, has been part of AI research from the very beginning. The need for planning arises naturally when an agent is interested in controlling the evolution of its environment. Algorithmically, a planning problem has as input a set of possible courses of actions, a predictive model for the underlying dynamics, and a performance measure for evaluating the courses of action. The output or solution is one or more courses of action that satisfy the specified requirements for performance. A planning problem thus involves deciding what actions to do, and when to do them.

In [28], a successful approach in automated reasoning for planning has been reported. Until then, it was considered folklore that planning required specialised formalisms and algorithms which take into account the special problems from this domain, as e.g. the so called frame problem. It turned out that propositional theorem provers are able to outperform special purpose planning systems. Here, stochastic "greedy" search techniques are important.

More recently, [18] shows how to encode planning problems into normal logic programs in such a way, that stable models of the program correspond to valid plans of the original program. In addition, running *smodels* on the transformed program and thus computing the stable models, outperforms dedicated planning algorithms for the original problem. The approach has some advantages over other approaches because the nonmonotonic semantics allow to handle the frame problem elegantly.

3.2 Model Checking

An important question, given a particular formal specification of a system, is whether this specification possesses certain properties. Often these properties can be formally expressed by temporal logic formulae. Model checking is a particular method (verification technique) to determine such properties of a specification and has been used successfully for finding design errors in real-life systems [14].

Interesting properties of practical value often involve *fairness constraints* and can be naturally computed by evaluating alternating fixpoints. Now one of the most canonical temporal logics to express alternating fixpoints is the modal μ-calculus [29]. Alternating fixpoints turn out to be the link to nonmonotonic reasoning, in particular to the well-founded semantics mentioned above, and lead to another success story of AI. Namely it has been recently shown [40] that a new algorithm, called LFAP, can be elegantly implemented within XSB thereby making use of XSB's specifics. Standard Prolog systems are not suitable for this because they do not terminate and do not allow for negation. In fact the pseudo-code is so clear and concise that a complete proof of the correctness of LFAP can be constructed easily. Even more important: the LFAP under XSB clearly outperforms conventional model checkers on canonical benchmarks [40].

3.3 Diagnosis

Model-based diagnosis of technical systems provides an interesting class of benchmark problems for two reasons. Firstly, the examples which stem from hardware design usually contain a large number of clauses, which have to be handled efficiently in order to solve realistic problems. Secondly, the diagnosis task involves by definition minimal model reasoning. Hence techniques from nonmonotonic reasoning can be applied (see Section 2.2).

Model-based diagnosis techniques were developed by R. Reiter [43]. In this framework, a simulation model of the technical device under consideration is constructed and is used to predict its normal behaviour. By comparing this prediction with the actual, observed behaviour it is possible to derive a diagnosis. This approach uses a logical description of the device, called the system description (*SD*), formalised by a set of first-order formulae. The components of the device are given by a set *COMP*, which come with an associated behavioural mode: $Ab(c)$ denotes that component c is faulty, while $\neg Ab(c)$ means that c is behaving correctly.

For example, a digital circuit containing an OR-gate component *or1* could be specified within *SD* by the formula

$$\neg Ab(or1) \rightarrow (hi(or1,o) \leftrightarrow (hi(or1,i1) \vee hi(or1,i2))),$$

where o, $i1$ and $i2$ stand for the output and the two inputs of the OR-gate, respectively. Besides *SD* , there is a set *OBS* of *observations*. These are formulae which describe the actual, observed behaviour of the system (e.g. that certain inputs and outputs of the the circuit are *high*). A diagnosis task now comes up if the observation contradicts the system description, i.e. if $SD \cup OBS$ is inconsistent. Now, a *diagnosis* of (*SD, COMP, OBS*) is a set $\Delta \subseteq COMP$, such that $SD \cup OBS \cup \{Ab(c) \mid c \in \Delta\} \cup \{\neg Ab(c) \mid c \in COMP - \Delta\}$ is consistent. Δ is called a *minimal diagnosis*, iff it is the minimal set (wrt. $\subseteq$) with this property. There are other interesting ways to define the diagnosis task, e.g. *abductive* diagnosis [15].

The set of all minimal diagnoses can be large for complex technical devices. Therefore, stronger criteria than minimality are often used to discriminate further among the minimal diagnoses. These criteria are usually based on the probability or cardinality of diagnoses. In the remainder of this section, however, we will use restrictions on the cardinality of diagnoses. We say that a diagnosis satisfies the *n-fault assumption* iff $|\Delta| \leq n$. In particular, one-fault diagnoses (n=1) are interesting, because in some scenarios it is plausible that only one component breaks at a time. In [3], it is shown how such diagnosis tasks can be efficiently solved with a general first-order deduction system: the model-generating NIHIL prover, which is based on the hyper-tableau calculus (see Section 2.1). It had to be slightly modified for the computation of *minimal* models, as demanded by the diagnosis task. With this modification alone, however, it turned out that realistically sized examples from a standard benchmark suite [27] could not be solved within acceptable time.

The key to the solution was to take advantage of additional domain dependent pruning techniques. The idea is to use an *initial interpretation* I_0 , which is a model of the correctly functioning device (which, of course, contradicts with the observation *OBS*). It can be computed efficiently by e.g. simulation devices. Now, I_0 can be used in a compilation step to transform the given clause sets for the diagnosis task in such a way that the computation of diagnosis is guided by the *deviations* of *OBS* from I_0 . This can have a huge pruning effect, because those parts of the system whose observed behaviour is in accordance with I_0 never need to be considered in the computation. To sum up, with the *combination* of all the described techniques from the *automated deduction* and the *diagnosis* worlds, the mentioned benchmark examples could be solved by our standard prover; it performed competitive (within one order of magnitude) with dedicated diagnosis systems.

3.4 Software Verification

Proof obligations arising directly in real-world software verification problems are far out of reach of automated deduction systems. Consequently, this is the domain of interactive systems, such as KIV [42] or 3TAP [9]. KIV includes an incomplete deduction system which is called on user request for selected subproblems. According to the developers of KIV, it would be very advantageous to have a more powerful (i.e. “more complete” but still fast enough) automated deduction system instead.

Coupling an automated deduction system to KIV is rather different from “standard” deductive tasks like solving benchmark problems. In the combination with KIV, there is a tough time limit for the proving process. Consequently, the theorems to be proven may not be too hard (and in fact, typically they are not). On the other side, raw proof time is not the ultimate measure. More important is the ability to relieve the KIV user from interaction steps. In a typical case study, 45 theorems could be proven by KIV using the built-in prover after guidance with 52 user interactions.

The first experiments we carried out with Protein and examples extracted from the KIV system did not show optimal performance. The same holds for other provers (Otter, Setheo) which were tried by the KIV developers. One source for improving the situation comes from so-called *simplifier rules*: these are formulae, that, by a special syntax, contain information *how* to use them, namely as (conditional) rewrite rules from left to right. For example, one useful application of a simplifier rule is to express a *definition* like in $X \leq Y \leftrightarrow (X < Y \vee X =$

Y). By this rule, all occurrences of ≤-literals can be eliminated.

Now, the important point is that the KIV system readily includes such simplifier rules and thus need not be artificially discovered. It is obvious that an automated prover should deal with simplifier rules properly, i.e. as conditional rewrite rules, but not as ordinary clauses. Hence we extended the model elimination calculus and the Protein prover by a general mechanism to incorporate such simplifier rules adequately. A detailed description of the whole approach can be found in [7]. In brief, we first employ simplification as a preprocessing step on the given input specification. This set is passed to the prover then. The prover itself uses simplifier rules for the proof search, too. Here, simplifier rules are applied not to clauses, but to the current proof object, i.e. the model elimination tree under construction.

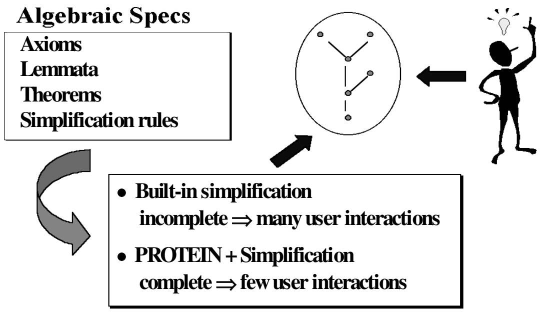

Fig. 2: Verification and AD – KIV

The various possibilities of incorporating simplification into our model elimination framework suggest an experimental evaluation. As mentioned, we extended our Protein prover, and evaluated various strategies for simplification using realistic problems from the KIV domain. In sum, we conclude that simplification has a lot of potential to help here. Interestingly, the most complete setting was also the most successful one. Protein with simplification scored best wrt. the number of problems solved within the time limit of one minute, followed by Setheo, Otter and Protein without simplification. Using Protein with simplification (see Fig.2), the KIV user can be relieved from 23 of 35 interactions to prove the 45 theorems in the mentioned case study!

3.5 Validation of Banking Fees

The practical applications described so far have one thing in common, they solve problems directly related to computer science (software verification, model checking) or technical systems (diagnosis). The following application description is out of the scope of computer science, it is about trading stocks and bonds. In detail, a Swiss credit institute uses a set of natural language rules to calculate the fees for buying and selling stocks and shares for their customers. This rule set is very large and it is nearly not possible to check manually if the rules are *correct*.

Motivated by this real world problem, the validation of rules for the computation of banking fees, we set up a theoretical framework consisting of *constraint logic programming* and *first-order theorem proving* techniques and developed a prototypical application [44]. Once again, domain specific knowledge (here: constraints) are essential for the success of the application. The motivating problem to be solved was to decide whether the calculation rules used by the Swiss credit institute are deterministic and total, i.e., is there at most one fee that can be calculated for a given transaction (determinism), and is it always possible to calculate one fee for any arbitrary transaction (totality). Although these questions are undecidable in general, the restricted form of the investigated rule sets makes them decidable and even tractable in practise.

To tackle this challenge we combined logic programming and constraint solving in an straightforward manner, by making use of the constraint model elimination (CME) calculus [8]. Because of the fact that model elimination [30] is a linear and goal-directed calculus, that is close to the logic programming language Prolog, it allows us to realize this calculus on top of existing Prolog systems by using the Prolog technology theorem prover (PTTP) technique proposed by M. Stickel, incorporated into the Protein system.

In our approach to decide the determinism of the rule set, given in first-order predicate logic, we slightly modified these PTTP technique to compile the rule set into a constraint logic program. There are sub-compilation steps involved in the whole transformation, like the transformation of full first-order formulae into a Prolog program and most important the annotation of domain specific constraints. The result of this automated transformation is a constraint logic program ready to be executed by a Prolog system.

So we actually used our PTTP-based deduction system Protein using the CME calculus for the final proof task to show that there is no transaction with two different calculated fees. Because of Protein's capability to calculate answers, a simple query to the constraint logic program of the form $\leftarrow t(X,Fee_1), t(X,Fee_2), Fee_1 \neq Fee_2$ calculates all transactions with more than one possible fee and therefore proves that the investigated rule set is not deterministic. By special visualisation tools to display the model elimination proof tree we are able to pick up the nondeterministic rules as explanation and even more we can generate according reports using the ILF system. The success of this application is due to the combination of constraint logic programming and first-order reasoning, based on the model elimination calculus. Each approach alone is not powerful enough to find the solution in reasonable time. The procedure proposed here can easily be generalised to analysing arbitrary rule sets.

4 Future Perspectives

Surely, there are many more interesting areas where deductive techniques can be applied to, if these techniques are geared to specific requirements of the respective application. Two challenging examples will be presented: the *RoboCup* robot soccer simulation and *web information systems* (see Sections 4.2 and 4.3).

Furthermore, future deductive systems should take into account that they might be used by persons who are *not* experts of automated deduction. Therefore, the systems themselves should provide supporting information for the user, when a task could not be completed automatically. This is discussed in the next Section 4.1.

4.1 Debugging of Specifications

In automated deduction, the final goal is to achieve a fully automatic proof system: given a logical specification of a problem, take a high-performance deduction system, and let it find a solution. Unfortunately, this does not work in practice, not only because deduction systems lack finding the proof within reasonable time, but also because the specification is error-prone. For the latter, in the literature methods are proposed for detecting and verifying errors in logic programs. But, in order to enable such analyses, usually termination of computation is presupposed.

Since termination is not guaranteed in first-order deduction, it should be fruitful to investigate techniques which are also applicable in the case of non-termination, e.g. by employing incremental, model-based techniques as provided by the hyper-tableau calculus [6]. Models are high-level descriptions of what should hold in a given specification and therefore enable a problem-oriented investigation wrt. critical properties like correctness, completeness, and sufficiency of specifications. By virtue of an incremental model construction procedure, this works even in cases where deduction systems usually do not terminate (by success or failure).

In general, the scenario can be pictured as follows (see Fig. 3): given a problem in a certain domain (1), one wants to prove some theorem in it, say in *elementary algebra*. Thus, at first, a formal specification in logics has to be given of the problem space and the theorem at hand (3), maybe using some theory libraries (2). By means of deductive tools then, it is tried to find a proof for the given theorem (4). Now, there are three possibilities: a proof of the theorem is found (6), or it is detected that a proof cannot be found automatically (5), or the theorem prover does not terminate with any answer and loops in (4).

In the first case, the proof might be formatted and output in natural language to get a readable solution to the stated problem, as done by the systems ILF [16] or Omega [10]. In the latter two cases, information about what went wrong in the deduction process is even more important. By inspection of partial models (8) that can be constructed for the given axiomatisation, errors can be detected as well as other insufficiencies of the specification. With this information, a corrected and tuned axiomatisation is formalized (3), and another (now hopefully successful) deduction run started (4). See also [24].

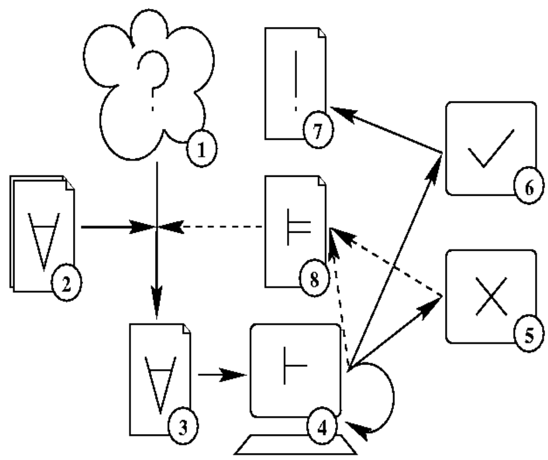

Fig. 3: Deduction life-cycle

One main problem of this approach is the potentially large size of models to be computed and inspected. Therefore, techniques are needed to restrict the model construction to relevant subparts. One could make use of modules or libraries during the specification phase, which are already tested or even proved to be correct. For example, a ring contains an additive group and a multiplicative semi-group. Therefore, one may use libraries specifying these parts and thereby avoid repeated inspection of certified specification modules, as suggested in [10].

Another case study in [24] describes the model-guided tuning of *spatial reasoning* axioms [21] with respect to a specific theorem proving task until an automated proof is eventually possible. Further benefit for debugging by models can be gained by checking specification parts against known model instances automatically. Finally, an important point in this view is the visualisation of partially computed interpretations. Thus, this work can be integrated into interactive proof frameworks with natural language output, such as ILF [16] or Omega [10].

It should be stressed, that the problem of incorrect or insufficient axiomatisations may occur in virtually every application area also mentioned in Section 3, where formal specifications have to be developed, in particular, *mathematics* [24], *software verification* [42], *diagnosis* [43] or *rule set validation* [44]. Automated deduction can provide here engineering support by declarative, model-based debugging.

4.2 Robot Soccer

Recently, there is growing interest in computer soccer. The RoboCup initiative [38] is an attempt to foster AI research by providing a standard problem where a wide range of technologies can be integrated and examined. In the last few years, there were several competitions in different leagues with real robots, and there is also a simulation league. At the University of Koblenz, there is ongoing research on designing clients for the RoboCup simulation league in Prolog and C++. We feel that logic and deduction are appropriate remedies for this task. A single player is part of a team and has to deduce information about the situation. For example, players have to recognise when passing the ball is possible or a player is off-side.

But, almost naturally, tasks to be solved by a team of autonomous agents are many-sided and complex. To achieve a goal, a single agent has to use a set of complementary subtasks. Some of these subtasks consist of solving numerical equations to enable a player to handle tasks like dribbling or actually passing the ball. On the other hand, we have to derive new information from a given set of facts. So we were led to the idea to combine the advantages of procedural and logic programming and decided for a hybrid system. As a result, we implemented the RoboLog Prolog extension. This extension is an enhanced RoboCup soccer server interface for Prolog. Time critical and computational expensive tasks are handled within the RoboLog module, as well as the exchange of data. The developer of a soccer client accesses the calculated data via Prolog predicates. The module provides the atomic soccer server commands and some more complex actions. The Prolog engine handles the player's reasoning and planning, i.e. to a certain degree it models a real soccer player's reasoning. However, it is difficult to say, in how far cognitive actions that are done on a subconscious level by humans, e.g. calculating the amount of power needed to pass (or stop) the ball, can be expressed by logical rules.

Currently, we are investigating the problem of modelling certain situations as patterns by means of logic programs and our Protein system (see [39]). For example, a situation where passing the ball is possible can be described as follows: one player has the ball, and another player can be reached and there is no player in between. We modelled these situations on top of the logical relations *left*, *right* and *between* (see also [21]). A situation pattern is realized as a logical description of spatial—and, to a degree—temporal relations. RoboLog and Prolog predicates will be used to define these relations. To ensure robustness of our approach, further methods can be used to aid the decision process, if no match is close enough. Especially, *spatial reasoning* techniques will be helpful.

4.3 Intelligent Web Information Systems

Overwhelmed by the results the standard WWW search engines produce, it is time to develop more *intelligent* and *autonomous* search and information gathering systems for the web. We need systems that will free the user from the time exhausting work of following thousands of document references, reading each of it in order to extract relevant information manually. Furthermore these systems should provide a set of analytic tools for the comparison of web contents and the discovery of knowledge and relations from the information retrieved.

So we focused our work on techniques for the information extraction from web pages and in particular how to develop a common extraction language in combination with logic programs [45]. The result was a domain independent extraction language. One basic feature of this language besides the concepts like recursion, is its ability to interact with logic programs during the extraction process (*code calls*). Therefore it allows us in conjunction with logic programs to set up relations between web pages, to reason about the contents of pages, to make comparisons on extracted facts and to deduce new facts. The most important point is, that we have a common theoretical well studied layer to work upon: deduction and more specific logic programs. Hence we are able to control the information extraction process with the assistance of deductive reasoning.

Our future plan is to develop methods and information systems that ease the search for facts in the WWW and its analysis. These systems should have the following capabilities: (1) to *reason* and *decide* where to search for good information source, (2) to *deduce* new information from the one they have found on the web, (3) to *learn* how to extract relevant information from new information sources (web pages), (4) to *discover* unknown relationships between the extracted information or between web-pages (5) to guide the search by *background knowledge*, that has to be updated concerning the experiences the system made, (6) to be able to *learn* the user's query behaviour, to offer the user related information due to his interests automatically.

To fulfil these tasks, we can adopt different methods and techniques from the fields of *artificial intelligence*, like *knowledge representation*, *logic programming*, *inductive logic programming*, and *data-mining*. We call a system consisting of the basic parts that are *information extraction*, *deduction* of new facts and the ability to use *background knowledge*, a *LogicRobot*. We have implemented a *LogicRobot* [1] that searches private advertisements of a web vendor (see Fig. 4). The user can set up constraints concerning the columns to search, the description of the item you are looking for, its price and the telephone number of the person selling it. The *LogicRobot* is fast enough to perform a real online search (without caching), so the user always gets offered the latest advertisements.

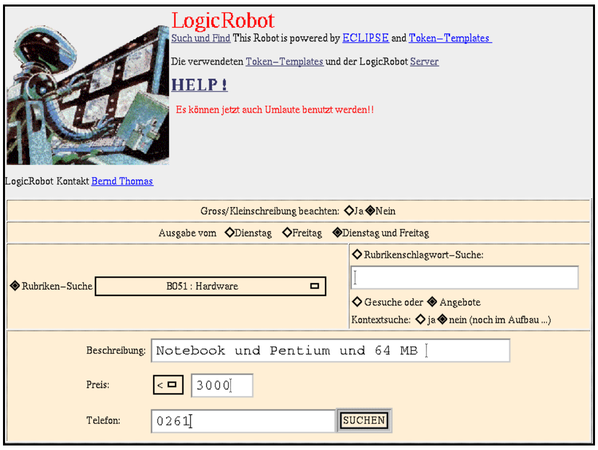

Fig. 4: The LogicRobot´s query-form

Currently we do research on combining the fields of *logic programming*, *inductive logic programming* and *data mining* to enhance our existing robot with the above described capabilities of *learning* extraction patterns and the user's query behaviour and finally to *discover* new *knowledge* from the extracted facts and relations between web pages.

5 Conclusion

In this article, we reviewed some recent successes of automated deduction from an application point of view. We described in detail examples from several different domains. Clearly, this overview is not exhaustive. Nevertheless, all examples corroborated the thesis, that (1) the choice of the respective deduction technology together with (2) a careful inspection of the application domain allows us to solve practical applications by automated deduction systems. Let us briefly summarise the applications mentioned in this article.

On the one hand, we investigated applications of *classical automated reasoning*. Software verification and analysing rule sets are two examples that are well-suited for goal-oriented top-down reasoning. But in order to become successful in these domains by means of automated deduction technology, we need additional mechanisms, namely simplifier rules and constraints, respectively (see Sections 3.4 and 3.5). For problems in mathematics and diagnosis, model-generating bottom-up approaches seem to be more feasible (see Sections 1 and 4.1 and Section 3.3). Again, domain specific procedures are important, namely equality reasoning and initial interpretations for guiding the search, respectively.

On the other hand, we considered applications of *nonmonotonic reasoning*. Planning problems can be solved by a stable model procedure, model checking is performed successfully by means of the well-founded semantics. But in both cases, a special encoding of the problem at hand is mandatory. For this, the interested reader is referred to the cited literature. In general, we believe that understanding such principles and having a good intuition about them will be the key to the success for automated deduction techniques in domains where dedicated systems are predominant, such as the mentioned *robot soccer* and *intelligent web retrieval* (see Sections 4.2 and 4.3).

References

[1] LogicRobot: http://www.uni-koblenz.de/~bthomas/such_und_find.html.

[2] C. Aravindan and P. Baumgartner. A Rational and Efficient Algorithm for View Deletion in Databases. In J. Maluszynski, editor, *Logic Programming - Proceedings of the 1997 International Symposium*, Port Jefferson, New York, 1997. The MIT Press.

[3] P. Baumgartner, P. Fröhlich, U. Furbach, and W. Nejdl. Semantically Guided Theorem Proving for Diagnosis Applications. In *15th International Joint Conference on Artificial Intelligence (IJCAI 97)*, pages 460–465, Nagoya, 1997. International Joint Conference on Artificial Intelligence.

[4] P. Baumgartner and U. Furbach. Model elimination without contrapositives. In Bundy [13], pages 87–101.

[5] P. Baumgartner and U. Furbach. .PROTEIN: A PROver with a Theory Extension INterface. In Bundy [13], pages 769-773.

[6] P. Baumgartner, U. Furbach, and I. Niemelä. Hyper Tableaux. In *Proc. JELIA 96*, number 1126 in Lecture Notes in Artificial Intelligence. European Workshop on Logic in AI, Springer, 1996.

[7] P. Baumgartner and D. Schäfer. Coupling the software verification system KIV with the theorem prover PROTEIN. In R. Berghammer and Y. Lakhnech, editors, *Tool Support for System Specification, Development, and Verification*, volume 9803. Christian-Albrechts-Universität Kiel, 1998.

[8] P. Baumgartner and F. Stolzenburg. Constraint model elimination and a PTTP-implementation. In P. Baumgartner, R. Hähnle, and J. Posegga, editors, *Proceedings of the 4th Workshop on Theorem Proving with Analytic Tableaux and Related Methods*, LNAI 918, pages 201–216. Springer, Berlin, Heidelberg, New York, 1995.

[9] B. Beckert, R. Hähnle, P. Oel, and M. Sulzmann. The tableau-based theorem prover $_3T^AP$, version 4.0. In *Proceedings, 13th International Conference on Automated Deduction (CADE), New Brunswick, NJ, USA*, volume 1104 of *Lecture Notes in Computer Science*, pages 303–307. Springer, 1996.

[10] C. Benzmüller, L. Cheikhrouhou, D. Fehrer, A. Fiedler, X. Huang, M. Kerber, M. Kohlhase, K. Konrad, E. Melis, A. Meier, W. Schaarschmidt, J. Siekmann, and V. Sorge. Ωmega: Towards a mathematical assistant. In McCune [34].

[11] S. Brass, U. Zukowski, and B. Freitag. Transformation Based Bottom-Up Computation of the Well-Founded Model. In J. Dix, L. Pereira, and T. Przymusinski, editors, *Nonmonotonic Extensions of Logic Programming*, LNAI 1216, pages 171–201. Springer, Berlin, 1997.

[12] G. Brewka, J. Dix, and K. Konolige. *Nonmonotonic Reasoning: An Overview.* CSLI Lecture Notes 73. CSLI Publications, Stanford, CA, 1997.

[13] A. Bundy, editor. *12th International Conference on Automated Deduction*, LNAI 814, Nancy, France, June 26–July 1, 1994. Springer-Verlag.

[14] E. M. Clarke and J. Wing. Formal methods: State of the art and future directions. *ACM Computing Surveys*, 28(4), 1996.

[15] L. Console and P. Torasso. A spectrum of logical definitions of model-based diagnosis. *Computational Intelligence*, 7(3):133–141, 1991.

[16] B. Dahn, J. Gehne, T. Honigmann, and A. Wolf. Integration of Automated and Interactive Theorem Proving in ILF. In McCune [34], pages 57–60.

[17] B. I. Dahn. Robbins Algebras are Boolean – A Revision of McCune's Computer Generated Solution of Robbins' Problem. *J. Algebra*, 1998. To appear.

[18] Y. Dimopolous, B. Nebel, and J. Köhler. Encoding Planning Problems in Nonmonotonic Logic Programs. In *European Conference on Planning (ECP-97)*. Springer, 1997.

[19] J. Dix, U. Furbach, and I. Niemelä. Nonmonotonic Reasoning: Towards Efficient Calculi and Implementations. In A. Voronkov and A. Robinson, editors, *Handbook of Automated Reasoning*. Elsevier-Science-Press, to appear 1999.

[20] J. Dix and F. Stolzenburg. A framework to incorporate non-monotonic reasoning into constraint logic programming. *Journal of Logic Programming*, 37(1-3):47–76, 1998. Special Issue on Constraint Logic Programming, Guest Editors: Kim Marriott and Peter Stuckey.

[21] C. Eschenbach and L. Kulik. An axiomatic approach to the spatial relations underlying *Left-Right* and *in Front of–Behind*. In G. Görz and S. Hölldobler, editors, *KI-97: Advances in Artificial Intelligence – Proceedings of the 21st Annual German Conference on Artificial Intelligence*, LNAI 1303, pages 207–218, Freiburg, 1997. Springer, Berlin, Heidelberg, New York.

[22] B. Fischer and J. Schumann. SETHEO Goes Software Engineering: Application of ATP to Software Reuse. In McCune [34].

[23] M. Fujita, J. Slaney, and F. Bennett. Automatic generation of some results in finite algebra. In *IJCAI-95 — Proceedings of the 14th International Joint Conference on Artificial Intelligence, Montreal.*

[24] U. Furbach, M. Kühn, and F. Stolzenburg. Model-guided proof debugging. Fachberichte Informatik 6/98, Universität Koblenz, 1998.

[25] C. Goller, R. Letz, K. Mayr, and J. Schumann. SETHEO v3.2: Recent developments – system abstract. In Bundy [13], pages 778–782.

[26] G. Gottlob. Complexity results for nonmonotonic logics. *Journal of Logic and Computation*, 2(3):397–425, 1992.

[27] The ISCAS-85 Benchmarks. http://www.cbl.ncsu.edu/www/CBL_Docs/iscas85.html,1985.

[28] H. Kautz and B. Selman. Pushing the envelope: Planning, propositional logic, and stochastic search. In *Proceedings of the 13th National Conference on Artificial Intelligence*, Portland, OR, 1996.

[29] D. Kozen. Results on propositional μ calculus. *Theoretical Computer Science*, 27:333–357, 1983.

[30] D. W. Loveland. Mechanical theorem proving by model elimination. *Journal of the ACM*, 15(2), 1968.

[31] R. Manthey and F. Bry. SATCHMO: a theorem prover implemented in Prolog. In E. Lusk and R. Overbeek, editors, *Proceedings of the 9th International Conference on Automated Deduction*, LNCS 310, pages 415–434, Argonne, IL, 1988. Springer, Berlin, Heidelberg, New York.

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

[33] W. McCune. Robbins Algebras Are Boolean. http://www.mcs.anl.gov/home/mccune/ar/robbins/index.html, 1996.

[34] W. McCune, editor. *Automated Deduction — CADE 14*, LNAI 1249, Townsville, North Queensland, Australia, July 1997. Springer-Verlag.

[35] J. Minker. On indefinite databases and the closed world assumption. *In Proceedings of the 6th Conference on Automated Deduction*, pages 292–308, New York, 1982. Springer, Berlin, Heidelberg, New York.

[36] Computer Math Proof Shows Reasoning Power. New York Times, 10. Dec. 1996.

[37] I. Niemelä and P. Simons. Efficient Implementation of the Well-founded and Stable Model Semantics. In M. Maher, editor, *Proceedings of the Joint International Conference and Symposium on Logic Programming*, pages 289–303, Bonn, Germany, September 1996. The MIT Press.

[38] I. Noda. Soccer server: a simulator for RoboCup. In *JSAI AI-Symposium*, 1995.

[39] O. Obst, J. Murray, F. Stolzenburg, and B. Bremer. Towards deduction in RoboCup. Manuscript.

[40] C. Ramakrishnan and S. Smolka. Fully local and efficient evaluation of alternating fixed points. In *Proceedings of 2nd International Workshop on Tools and Algorithms for the Construction and Analysis of Systems*, LNCS. Springer, 1998.

[41] P. Rao, K. Sagonas, T. Swift, D. S. Warren, and J. Freire. XSB: A System for Efficiently Computing Well-Founded Semantics. In J. Dix, U. Furbach, and A. Nerode, editors, *Logic Programming and Non-Monotonic Reasoning, Proceedings of the Fourth International Conference*, LNAI 1265, pages 430–440, Berlin, June 1997. Springer.

[42] W. Reif, G. Schellhorn, and K. Stenzel. Proving System Correctness with KIV 3.0. In McCune [34], pages 69–72.

[43] R. Reiter. A Theory of Diagnosis from First Principles. *Artificial Intelligence*, 32:57–97, 1987.

[44] F. Stolzenburg and B. Thomas. Analysing rule sets for the calculation of banking fees by a theorem prover with constraints. In W. Bibel and P. H. Schmitt, editors, *Automated Deduction – A Basis for Applications. Volume III: Applications*, Applied Logic Series 10, pages 243–264. Kluwer Academic, Dordrecht, The Netherlands, 1998.

[45] B. Thomas. Intelligent Web Querying with Logic Programs. In J. Dix and S. Hölldobler, editors, *Proceedings of the Workshop on Inference Systems in Knowledge-based Systems, preceding the national German AI conference KI '98, Bremen, Germany*. University of Koblenz, TR 10/98, August 1998.

[46] L. Wos. Programs that offer fast, flawless, logical reasoning. *Communications of the ACM*, 41(6):87–95, 1998.