Applications of Automated Reasoning

Ulrich Furbach and Claudia Obermaier

Universität Koblenz-Landau
D56070 Koblenz, Germany
{uli,obermaie}@uni-koblenz.de

Abstract. This paper offers an informal overview and discussion on first order predicate logic reasoning systems together with a description of applications which are carried out in the Artificial Intelligence Research Group of the University in Koblenz. Furthermore the technique of knowledge compilation is shortly introduced.

1 Introduction

Automated theorem proving systems have made increasing progress during the last decades. There was even a prominent open problem, the Robbins problem, which has puzzled logicians since 1930, which was solved by the Automated Reasoner EQP for first order equational logic, developed at Argonne National Laboratory [McC97]. Propositional reasoning systems are very successful in soft- and hardware verification, where the length of formulae which can be processed has grown by orders of magnitude over the last 10 years; today it is very well possible to solve real world verification tasks from hardware design.

In knowledge representation there was a shift from graphic oriented systems like KL-One in the beginning of the 90s towards concept languages or description logic, as it is called nowadays. For the processing of description logics the most commonly used algorithms are basically tableau calculi, which reached a very sophisticated level, allowing the use of description logics for numerous interesting applications (see e.g. [BCM$^+$03]). Because of the close relationship between description logic and modal logic, the fact emerged, that in many cases description logic systems are the more powerful modal logic provers ([Mas99]). This is of importance because modal logic is a decidable fragment of first order predicate logic and thus it plays an important role in computer science.

In this paper we want to demonstrate, that automated reasoning systems are very well ready for real world applications. We are dealing with first order predicate logic systems, accepting its semi-decidability and taking advantage from its higher descriptive power. We are aware that there are many applications of propositional and even higher order interactive reasoning systems; in this paper, however, we want to focus on own experiences and therefore we concert this presentation mainly about first order automated reasoning.

In the following section we briefly depict the state of the art in the development of first order high performance theorem proving, while in the main part we then focus on applications we carried out in the AI Research Group of University of

© Springer-Verlag Berlin Heidelberg 2007

Koblenz and in wizAI GmbH, a spin-off of this research group. In a last section we will introduce some aspects of knowledge compilation.

2 State of the Art in Automated Deduction

In this section we will use a small toy example to clarify and to discuss some aspects of automated reasoning systems. Given the knowledge base from Figure 1(a), most systems start by transforming this set of formulae into a set of clauses form Figure 1(b). This is astonishing because there are lot of arguments against this transformation: most important that the structure of the formulae gets lost despite equivalence transformation. This structure might mirror some properties of the domain which is modeled, and which can possibly be used to control the navigation through the search space while proving a theorem based on this formula. If the reasoning system allows the user to control the proof by interaction, it might be helpful to retain the structure, in order to facilitate navigation for the user. To our knowledge, there are very few systems which directly work with the original formula; two of them are used in a program verification context, where user interaction is often helpful ([BHOS96, ABB$^+$02]). Most high performance theorem proving systems for predicate logic use clause normal form (e.g. [Wei97, Sch04, RV02, Wer03])

$symptom(s) \leftarrow$	$symptom(s)$
$cause(c_1) \vee cause(c_2) \leftarrow symptom(s)$	$cause(c_1) \vee cause(c_2) \vee \neg symptom(s)$
$treatment(t_0) \leftarrow cause(c_1)$	$treatment(t_0) \vee \neg cause(c_1)$
$treatment(t_1) \leftarrow cause(c_1)$	$treatment(t_1) \vee \neg cause(c_1)$
$treatment(t_0) \leftarrow cause(c_2)$	$treatment(t_0) \vee \neg cause(c_2)$
$treatment(t_2) \leftarrow cause(c_2)$	$treatment(t_2) \vee \neg cause(c_2)$
(a) Knowledge base KB	(b) Set of clauses

Fig. 1. Knowledge base KB and corresponding set of clauses

Linear Deduction. Most textbooks on Artificial Intelligence (AI) present a resolution calculus to reason about knowledge bases (see e.g. [PMG97] or an overview on different textbooks [Fur03]). In the 70s of the previous century this was indeed the main approach to process logical formulae in AI and a very common understanding was at that time that goal oriented linear deduction should be used to prove logical consequences from a set of formulae. Assume for example the knowledge base KB from Figure 1 together with the task to prove that there is a treatment given that special situation; the existence of a treatment can be modeled by the additional formula $\exists X treatment(X)$, which is called a goal. Altogether we have to prove $KB \models \exists X treatment(X)$. Since resolution is a refutational calculus, we have to negate the goal and after a slight equivalent preserving transformation we get the clause set $KB \wedge \neg treatment(X)$, where all variables are implicitly universal quantified. In order to find a refutation of this, it seems to be very natural to start with the goal and to work "backwards" until one reaches the empty clause $\square$, indicating that the clause set is unsatisfiable and hence the goal logically follows from the knowledge base. This would lead to a sequence of resolvents $\neg treatment(X), \neg cause(c_1), cause(c_2) \vee \neg symptom(s), cause(c_2), treatment(t_2), \square$. One of the first calculi advocating this goal oriented linear approach are model elimination ([Lov68]) and linear resolution ([KK71]) and a model elimination theorem prover SETHEO even won the CASC competition (which will be discussed later). It was much later in the 90s where we really understood that these calculi are not based on resolution – they are much closer to tableau calculi, at least if one takes the treatment of variables as a discriminating parameter[1].

One appealing aspect of linear refutation is that it is very close to the concept of logic programming: one starts from a call of the program and then works through a sequence of intermediate computations until $\square$ is found. The answer to such a computation can be constructed from the unifiers used during the inference steps. In [BF97] it is shown how the logic programming paradigm can be used not only for Horn clause programming, but also for full first order logic by means of a variant of model elimination. Another argument for this goal oriented search for a refutation usually was its higher potential in search space pruning. We intentionally use past tense, because nowadays most high performance theorem provers are working in a saturation based manner, which is explained in the next section.

Saturation. Assume again the proof task $KB \wedge \neg treatment(X)$ from the previous discussion. Instead of assigning some of the clauses, e.g. the goal clause, a particular importance, we just take the clause set as it is given by this task. If we further assume that we have resolution as the inference rule at hand, we simply add new resolvents to this set. Starting with the initial set $S = KB \cup \{\neg treatment(X)\}$ one can derive by this the new set $S' = S \cup \{cause(c_2) \vee \neg symptom(s) \vee treatment(t_0)\}$. Such an extension is done until the set contains the empty clause or (in special cases) until there are no new clauses to derive, i.e. the set is saturated.

There are some issues to solve if one tries to do saturation based proving. The amount of clauses which are generated from a given set of clauses, can increase dramatically. Therefore it is mandatory to avoid generating in some sense useless clauses and to get rid of redundant clauses. A very powerful technique to this end is the use of term ordering to control the generation of new clauses. For an overview of this technique in the context of resolution the reader is pointed towards [BG01]; theorem provers which are successful with this techniques are, among others, Otter, Spass and Vampire ([McC90, Wei97, RV02]). The use of ordering for controlling the generation of new clauses has another advantage: it also helps in handling equality. If the formulae to be handled by the reasoning system contain an equality predicate, there are basically two different methods

[1] In resolution variables are treated as being implicitly universal quantified, whereas in tableau calculi they usually are rigid, i.e. placeholders for a yet unknown constant.

to handle this. Either one adds axioms to the set of clauses describing the usual properties of equational logic or an additional inference rule, like paramodulation, is used to handle the equations. The latter approach also raises the problem of generating too many new clauses, which, however, can be controlled as well by term ordering.

It is not only resolution which is available as an inference rule in saturation based theorem proving. If the entire formula which is given as the proof task, is transformed into an equivalent formula in equational logic, superposition together with ordering restriction can be used. This approach is followed in the theorem prover E ([Sch04]) or in the Waldmeister-system ([HL02]).

It is no doubt that the systems employing saturation techniques nowadays belong to the most successful high performance first order systems; this aspect will be discussed in more detail below.

Tableaux. Although introduced more or less at the same time as the resolution principle (1950 – 1960), there was only evidence in the 1980th that tableau calculi offer an alternative approach with very interesting properties. These are in particular, that parts of the history of the current proof attempt are coded into the proof object and that the variables are treated rigidly. We will explain this on a special form of tableaux, the so called *Hypertableaux*, which are introduced in [BFN96] and which is used in KRHyper, a theorem prover, which is the basis throughout our applications. The calculus is a clause normal form tableau calculus and hence we start constructing a tableau from a given set of clauses, which are regarded as implications (negative literals are the premises, positive literals the conclusion). In our example from Figure 1 there is one single fact, i.e. implication with empty premise. Hence we construct the tableau consisting of one node, namely $symptom(s)$. The only inference rule works as follows: we take a branch from the tableau and a clause from the clause set; if all literals from the premise of the clause are contained in the branch (in the case of variables it is slightly more complicated), then the branch can be extended by the literal in the conclusion. If there is more than one literal in the conclusion, the branch is split; if there is no conclusion in the clause, the branch is closed. A clause set is unsatisfiable, if a tree constructed by this method only contains closed branches. An interesting property o f this method can be seen if one omits the goal clause $\neg treatment(X)$, which is a clause without positive literal, i.e. without conclusion. The tableau from Figure 2 is an exhausted (i.e. maximal) tableau which can be constructed from the clauses in our example. In such a case we not only have a proof object, containing information from the proof search, we also can read two models of the clause set, namely the atoms from each of the two branches. Hence tableaux are also very helpful for constructing models for satisfiable clauses. This is of particular importance in a non-monotonic setting, where minimal models have to be computed as a basis of a closed world assumption; an overview of such approaches can be found in [DFN01].

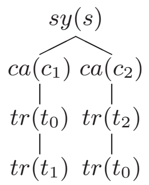
Tableau methods are also the main mechanism for the design of description logic systems, which are gaining increasing importance in the design of the Semantic Web project. A drawback of tableau calculi is the handling of equality;

$\neg tr(X)$
$\neg ca(c_1)$
$ca(c_2) \vee \neg sy(s)$
$ca(c_2)$
$tr(t_2)$
$\bot$

(a) Linear resolution

$ca(c_2) \vee \neg sy(s) \vee tr(t_0)$
$ca(c_2) \vee \neg sy(s) \vee tr(t_1)$
$ca(c_2) \vee tr(t_1)$
$\dots$

(b) Saturation



(c) Hypertableau

Fig. 2. Different calculi – predicates are abbreviated by the first two letters

the variables in a tableau have to be substituted simultaneously in the entire tableau during a unification, which is necessary in an extension step with first oder clauses. This makes the handling of equality very difficult, and, indeed, there are no high performance tableau proofers which are also dealing with equality in a way comparable with saturation based systems.[2]

Empirical Aspects. Two important achievements in automated reasoning research are the commonly used benchmark suite TPTP ([SS98]) and the CASC-competition ([PSS02]). The TPTP (Thousands of Problems for Theorem Provers) problem library is a library of test problems for automated theorem proving (ATP) systems. Currently the TPTP contains 7000 test problems with a large variety in complexity and difficulties. These problems are grouped into domains, like lattice theory, hardware creation and verification and many others. Besides the problem library, the TPTP contains a utility to convert the problems to existing ATP formats; it offers conversions to nearly all systems and thus facilitates the use of the library. The principal motivation for the TPTP project is to move the testing and evaluation of ATP systems from the previous ad hoc situation onto a firm footing. This goal is certainly reached, and, even more, the TPTP idea led to the CASC competition, which is held annually during a deduction conference.

CASC evaluates the performance of sound, fully automatic, classical first order ATP systems. The evaluation is in terms of the number of problems solved and the average runtime for successful solutions. The problems are chosen from the TPTP Problem Library and they are presented together with a specified time limit for each solution attempt. Although there might be the danger that system designers try to tune their provers towards the event and the possible problem set (the TPTP), there are certainly a number of advantages:

– It turned out that different calculi and systems are winning in different problem classes.
– The systems are becoming increasingly robust. They have to run fully automated, to be invoked from batch, such that their developers have no chance to interfere during the entire competition.

[2] In the Hyper tableau calculus the situation is different, because we have universal variables; efficiently equality handling is in development right now.

– The progress of the field becomes transparent, by having the winners from the previous year participate, even if a new version of the system is also an entry into the current competition.

As said above, one way to present the success of automated theorem proving is to refer to TPTP and CASC. However, it is time to point out that applications, of course, are another important measure of success. We experienced that model generation deduction offers a very flexible way to use automated systems in applications and embedded systems. This is what we will exemplify in the following section.

3 Applications

In this section we will focus on application projects we worked through the recent years in the AI Research Group (AGKI) of Koblenz University and in wizAI GmbH, which is a spin-off of the research group. When researchers talk about applications, this can have very different semantics; some mean the application of a theoretical tool or method within the own field, e.g. using a theorem prover for knowledge representation purposes in Artificial Intelligence research. Others mean that a problem for which there was known no solution can be solved by means of the research carried out; e.g. the solution of an open mathematical problem by an automated reasoning system, mentioned in the introduction. In this paper we offer a different understanding of 'application': we have a reasoning system, the KRHyper, based on hyper tableau; this system has been developed during many years, it is tested in various contexts and we assume that it is a very reliable and flexible tool. And this is exactly what we are benefitting from in other projects; we use this tool as part of the software developing process. It is used to quickly and safely solve subproblems during the software engineering process. Of course the problems could have been solved differently by programming it from scratch. By the use of our KRHyper the solution can be achieved quicker, easier to test and more flexible to allow modification in case the requirements of the project change, which is a very likely the case in commercial projects.

We used KRHyper in the following larger projects:

– Together with Dresdner Bank we developed a prototype of a knowledge management system, which is used for early discovery of reputational risks caused by decisions and statements from own bank divisions (for details see [FGHT$^+$04]). This is presumably the only software system in a major bank, where an automated theorem prover is running its kernel.
– In a PhD-project, which was aiming at the intelligent processing of XML database queries, it turned out, that KRHyper could be used to transform incomplete queries into queries which can be processed efficiently by the underlying database system (details can be found in [BFGHK04])
– In RoboCup we are working towards the use of logic in the simulation league. Until now, we have been working on a soccer team which was programmed in large parts by the use of logic programming techniques. KRHyper was

used to check formal properties of the team, i.e. the multi-agent system. Recently we changed the focus of the project, which is carried out in the DFG Special Focus Program 1125 "RoboCup"; because of the mixture of real valued computation and logical reasoning we are using hybrid automata for model checking of properties ([Hen96]).

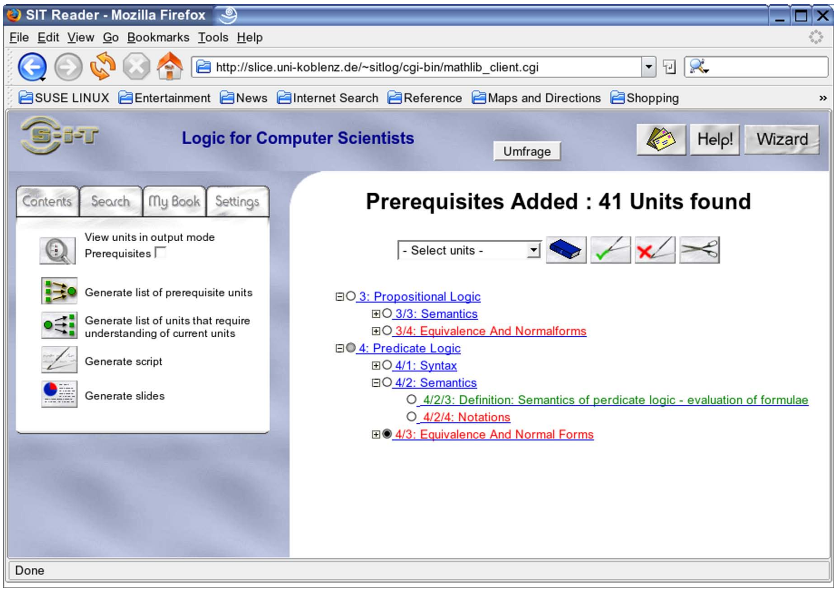
– The Living Book project was carried out over several years; funded by the German Ministry of Research and Education and by the European Comission. We developed a system which allows the development and use of intelligent personalised textbooks via the internet. This project will be discussed in more detail below.
– The Spatial Metro project is an ongoing project carried out together with the city of Koblenz and with two of her twin cities, Norwich and Rouen. It is financed by the European Commission and the State Government of Rheinland-Pfalz. The goal of our part of the project is to use AI techniques for efficient guidance of tourists in the city. This project will be discussed in more detail below.

Living Books. Living Book is a project which was carried out during several years aiming at the development of personalized intelligent books. Intelligent in the sense, that a user is able to work and interact with her book, which is maintained on a central server. The book also contains interactive systems, which can be used for exercises and practice. For access to some books published in this project the reader is referred to `http://www.in2math.de`; in this paper we want to concentrate on the underlying technique, the Slicing Book Technique. By this technique a document, say, a mathematics text book, is separated once as a preparatory step into a number of small units, such as definitions, theorems, proofs, etc. The purpose of the sliced book then is to enable authors, teachers and students to produce personalized teaching or learning materials based on a selective assembly of units. Once a reader is entering the portal of the book in the web, she can login with her account and gets the entry page of the book. There it is possible to select parts of the book from the table of contents and to specify preferences, e.g. to include all prerequisites necessary for the understanding of the selected units or to include all parts were the contents of the selected units is used – such a view is depicted in Figure 3.

Once the user has specified the current view of the book, the system has to provide the appropriate units and compose them in order to receive a final pdf-document. This task is depicted in Figure 4 for the general case, where the user can even select from various books. Assume she is asking for an overview of the notion of "Normal Forms" by selecting the appropriate parts. In addition the user has some preferences, like preferring formal notations or explanations by examples, which the system already knows about the user.

The slices or units, whose collection constitutes the books basically contain LaTex-code. This is connected with appropriate meta data, like the relations according to the prerequisite and refers relation, meta-data stating the type of the unit (example, proof, theorem and things like this) or ontologies which allow the combination of different keyword systems. All this data belonging to the users query are put together and stated as proof task, i.e. a logical formula for the KRHyper system. KRHyper computes a model of the given set of clauses; it is important to note that the formula contains all the slices of the books in a certain representation. From the model for the given query the system can extract the identification numbers of the slices, put together the LaTex parts and generate a pdf document, which can be presented to the user.

There are some lessons we learned from this application: the KRHyper system must be able to process very many, i.e. ten thousands of slices efficiently and it needs non-monotonic negation in order to deal with closed-world assumptions. Another important property is that it must be possible to process description logic parts of the task. For details of all this the reader is referred to [BFGHS04].


SIT Reader - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://slice.uni-koblenz.de/~sitlog/cgi-bin/mathlib_client.cgi
SUSE LINUX
Entertainment
News
Internet Search
Reference
Maps and Directions
Shopping
Logic for Computer Scientists
Umfrage
Help!
Wizard
Contents
Search
My Book
Settings
View units in output mode
Prerequisites
Generate list of prerequisite units
Generate list of units that require understanding of current units
Generate script
Generate slides
- Select units -
Done

Prerequisites Added : 41 Units found

- 3: Propositional Logic
  - 3/3: Semantics
  - 3/4: Equivalence And Normalforms
- 4: Predicate Logic
  - 4/1: Syntax
  - 4/2: Semantics
    - 4/2/3: Definition: Semantics of perdicate logic - evaluation of formulae
    - 4/2/4: Notations
  - 4/3: Equivalence And Normal Forms

Fig. 3. A personalized view of Living Books

Spatial Metro. One goal of this European Commission project is the use of AI techniques for efficient guidance of tourists in a city. For these possible tours within a city the metaphor 'spatial metro' is used. The points of interest in a city are depicted in the form of a metro map: according to the type of these points we can have different 'metro lines'; for example there may be a monument line, a shopping and a culture line. Figure 5 shows two of these lines together with the points of interest they contain.

Each of these points of interests is equipped with a bluetooth access point, which is able to send information about this location. This can be information about buildings, history, a map or even latest offers from a shop. If a tourist is reaching the area of this access point his mobile phone or his PDA can connect with this access point and present the information. Two aspect are of importance: this connection and hence the service is for free, no phone or WLAN fees have to be paid for and, more interesting (at least for this paper), the information which is offered by the access point is processed and filtered by the users phone. For this the user was able to edit a special profile on his phone, which contains preferences and other private information. This information is kept secure within her phone and is compared with the information (and its meta data), such that only those information which are of interest for the user are presented.

The comparison and processing of the information offered by the access point is done by KRHyper. For this we re-implemented the theorem prover in Java ME in order to get it running on a smart phone; presumably its the first theorem prover for first order predicate calculus running on a mobile phone (if the phone is not in use it can be used to solve TPTP-problems); for more information see [KS05b]; more about the entire approach can be found in [KS05a].

The lessons we learned until now from this project: Firstly, implementation language matters! Our KRHyper system is implemented in Ocaml, mainly because this was the Ph.D. student's favorite language; when we tried to get KRHyper running on a smart phone, it became obvious that we need a JAVA version and hence a re-implementation became necessary. Maybe such a porting could have been taken into account from the very beginning. The second lesson is more on the project design, concerning the willingness of users to download a piece of software, i.e. the reasoning machinery, on their mobile phones. In a field study we carried out, it turned out, that users are rather reluctant to do this. In a second phase of the project we are working to get rid of this bottleneck.

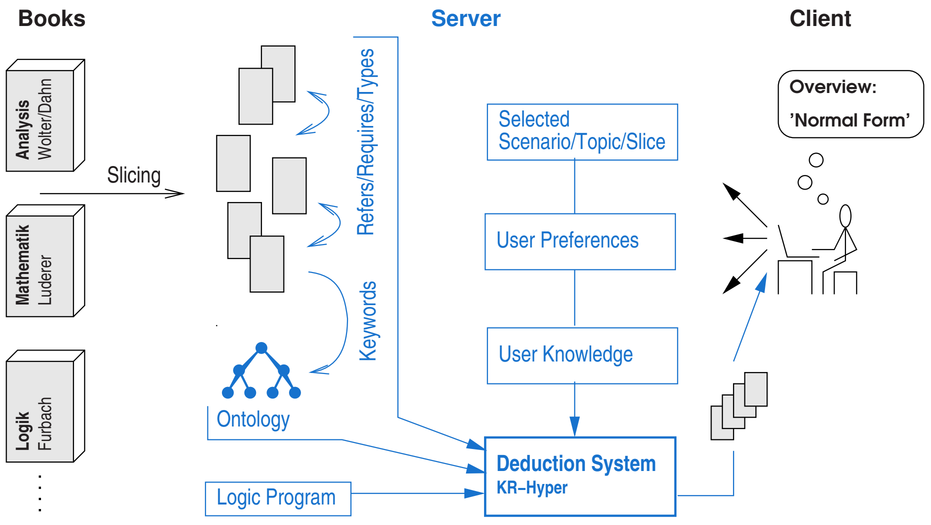
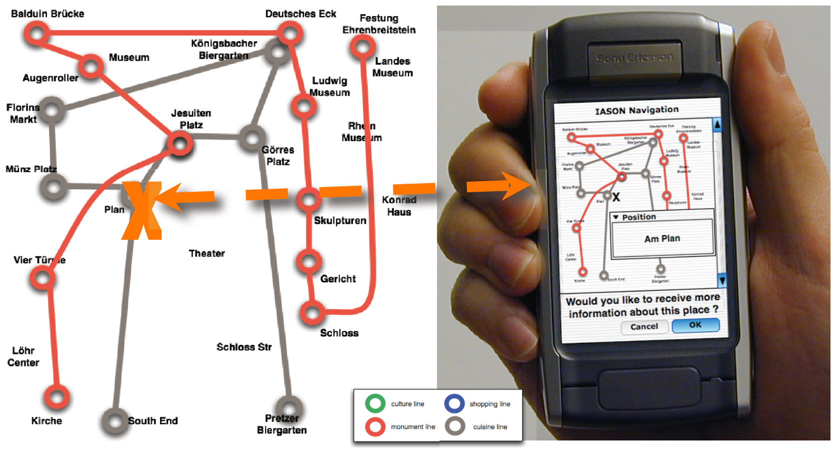


Fig. 4. The reasoning part of Living Books



Would you like to receive more information about this place ?

Fig. 5. A tourist guide on your mobile

4 Knowledge Compilation

In practice we are very often confronted with the following task: given a knowledge base, we want to answer a set of queries from that knowledge base. For example in diagnosis of electrical circuits, the system description of the correctly functioning circuit is used for various different queries. The naive approach to solve this problem would be to answer all the queries independently. But this would cause an exponential complexity for each query. That is why a new approach called knowledge compilation evolved. The basic idea of knowledge compilation is to precompile the knowledge base into a special form. This precompilation step is very costly (meaning of exponential complexity) but has to be performed only once. After that precompilation, some types of queries can be answered in polynomial or even linear time. Usually the formula in the target language of that precompilation has lots of other very nice properties such as the possibility of projecting the formula onto a set of atoms in linear time. Since the costly precompilation only has to be performed once, its complexity is relativized. There is a huge number of target languages for the mentioned precompilation. A rather new target language for knowledge compilation is DNNF. We will now take a closer look at this normal form. In the following, the term formula always means propositional logic formula.

Decomposable Negation Normalform. DNNF is short for decomposable negation normal form and is a special normal form developed in [Dar01]. A formula is in DNNF, if it is in negation normal form (NNF) and additionally satisfies the decomposability property. This property means that for any conjunction which occurs in the formula, the conjuncts do not share atoms. As an example take the set of clauses $F = \{\{a \vee b\}, \{c \vee \neg b\}\}$. This clause set is not in DNNF, because the atom b occurs in both clauses and as usual, the clauses of the set are combined by conjunction.

DNNF has the very nice property, that satisfiability can be decided in time, which is linear to the length of the DNNF. This is a direct consequence of the decomposability property. Because of this property it is possible to perform the satisfiability test on each subformula independently. Another very interesting feature of formulae in DNNF is the possibility to check the minimal cardinality in linear time. The next very interesting feature of formulae in DNNF ist the possibility to project a DNNF on a set of atoms in linear time. A list of other important features can be found in [Dar01].

Compilation of Propositional Logic Formulae into DNNF. A very naive approach is to use Shannon's rule to transform a formula into DNNF. As an example we will transform the clauseset of our previous example F into DNNF. F is not in DNNF, because the variable b occurs in two different clauses. Now we transform F into DNNF by using Shannon's rule:

$$\begin{aligned} dnnf(F) &= b \wedge F_{|b=true} \vee \neg b \wedge F_{|b=false} \\ &= (b \wedge c) \vee (\neg b \wedge a) \end{aligned}$$

In this example the transformation is very short. But one can easily imagine that the compilation of bigger formulae gets a lot more complicated. Because in huge sets of clauses, as occuring in practice, usually a great deal of atoms are shared between different clauses. That is why a variety of algorithms for the compilation into DNNF were developed. Many of these algorithms are based on DPLL ([Dar04],[Dar01]).

Weaving Projection into the Computation of DNNF. Quite often, we are only interested in a special part of our knowledge base F. Meaning for example that we are only interested in the values of a set of atoms S. Hence we want to *project* the knowledge base on this set of atoms S. Let Σ be the set of all propositional atoms occuring in our knowledge base F. Then the projection of F on the atoms in S is dual to *forgetting* all the atoms included in $\Sigma \setminus S$. Given that the projection of a DNNF on a set of atoms is linear, the common procedure is to transform the knowledge base into DNNF and to project on S afterwards. In [Wer06] it is suggested to weave in projection into the precompilation step. It is shown that this leads to an exponential saving of space.

Let's take a closer look at the technique of weaving in projection into the DNNF transformation. As mentioned above, a DPLL based algorithm is used to transform formulae into DNNF. To use this algorithm, the knowledge base is required to be given in CNF. During the computation of the DNNF, it is possible to use a number of rules to get rid of the atoms which are supposed to be forgotten. If one of the atoms we want to forget is pure, we can use pure literal elimination to get rid of this atom. The Isol* rule is another possibility to remove atoms which are supposed to be forgotten. This rule is related to resolution. The application of these rules not only removes atoms which we want to forget, but can also have a positive effect on the transformation into DNNF. Atoms which are shared only between clauses which are removed by this rules do not violate the decomposability property after the application of these rules.

Although we only used a very small example to explain the knowledge compilation approach, it should be clear that such techniques can be very helpful in realistic applications, as they are described in this paper. Currently we are working on separating subproblems where knowledge compilation can be applied.

5 Conclusion

In this paper we gave a very rough overview on first order predicate logic systems and its most recent developments. We then depicted some applications and we tried to demonstrate that automated reasoning systems are a valuable tool to be used in important parts in the application systems. Of course, even if the reasoner is fully developed and tested there is a considerable amount of work to be done for its integration and the design of the appropriate interfaces.

As a last remark concerning the development of commercial real world applications, we want to point out that it is a long way from an academic prototype system towards a real product. There is a lot of manpower to invest, but on the other hand, academic research and development can also benefit from this.

References

[ABB+02] Wolfgang Ahrendt, Thomas Baar, Bernhard Beckert, Martin Giese, Reiner Hähnle, Wolfram Menzel, Wojciech Mostowski, and Peter H. Schmitt. The KeY System: Integrating Object-Oriented Design and Formal Methods. In *Fundamental Approaches to Software Engineering. 5th International Conference, FASE 2002*, LNCS 2306, pages 327–330. Springer, 2002.

[BCM+03] Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications.* Cambridge University Press, 2003.

[BF97] Peter Baumgartner and Ulrich Furbach. Calculi for Disjunctive Logic Programming. In Jan Maluszynski, editor, *Logic Programming - Proceedings of the 1997 International Symposium*, New York, 1997. The MIT Press.

[BFGHK04] Peter Baumgartner, Ulrich Furbach, Margret Groß-Hardt, and Thomas Kleemann. Model Based Deduction for Database Schema Reasoning. In *KI 2004*, volume 3238 of *LNCS*, pages 168–182. Springer Verlag, Berlin, Heidelberg, New-York, 2004.

[BFGHS04] Peter Baumgartner, Ulrich Furbach, Margret Groß-Hardt, and Alex Sinner. Living Book - Deduction, Slicing, and Interaction. *J. Autom. Reasoning*, 32(3):259–286, 2004.

[BFN96] Peter Baumgartner, Ulrich Furbach, and Ilkka Niemelä. Hyper Tableaux. In José Júlio Alferes, Luís Moniz Pereira, and Ewa Orlowska, editors, *JELIA*, volume 1126 of *LNCS*, pages 1–17. Springer, 1996.

[BG01] Leo Bachmair and Harald Ganzinger. Resolution Theorem Proving. In Robinson and Voronkov [RV01], pages 19–99.

[BHOS96] Bernhard Beckert, Reiner Hähnle, Peter Oel, and Martin Sulzmann. The Tableau-based Theorem Prover $_3T^AP$ Version 4.0. In Michael A. McRobbie and John K. Slaney, editors, *CADE*, volume 1104 of *LNCS*, pages 303–307. Springer, 1996.

[Dar01] Adnan Darwiche. Decomposable Negation Normal Form. *Journal of the ACM*, 48(4), 2001.

[Dar04] Adnan Darwiche. New Advances in Compiling CNF into Decomposable Negation Normal Form. In *Proceedings of the 16th Eureopean Conference on Artificial Intelligence, ECAI'2004*, pages 328–332, 2004.

[DFN01] Jürgen Dix, Ulrich Furbach, and Ilkka Niemelä. Nonmonotonic Reasoning: Towards Efficient Calculi and Implementations. In Robinson and Voronkov [RV01], pages 1241–1354.

[FGHT$^+$04] Ulrich Furbach, Margret Groß-Hardt, Bernd Thomas, Tobias Weller, and Alexander Wolf. Issues Management: Erkennen und Beherrschen von kommunikativen Risiken und Chancen. Fachberichte Informatik 2–2004, Universität Koblenz-Landau, Institut für Informatik,Universitätsstr. 1, D-56070 Koblenz, 2004.

[Fur03] Ulrich Furbach. AI – A Multiple Book Review. *Artificial Intelligence*, 145(1-2):245 – 252, 2003.

[Hen96] Thomas A. Henzinger. The Theory of Hybrid Automata. In *Proceedings of the IEEE Symposium on Logic in Computer Science (LICS 1996)*, pages 278–292, 1996.

[HL02] Thomas Hillenbrand and Bernd Löchner. The Next WALDMEISTER Loop. In Andrei Voronkov, editor, *CADE*, volume 2392 of *LNCS*, pages 486–500. Springer, 2002.

[KK71] R. A. Kowalski and D. Kuehner. Linear Resolution with Selection Function. *Artificial Intelligence*, 2:227–260, 1971.

[KS05a] Thomas Kleemann and Alex Sinner. Decision Support for Personalization on Mobile Devices. In *Proceedings of the 21st International Conference, ICLP 2005*, pages 404–406, 2005.

[KS05b] Thomas Kleemann and Alex Sinner. Krhyper - in your Pocket, System Description. In Robert Nieuwenhuis, editor, *CADE*, volume 3632 of *LNCS*, pages 452–458. Springer, 2005.

[Lov68] D. Loveland. Mechanical Theorem Proving by Model Elimination. *JACM*, 15(2), 1968.

[Mas99] Fabio Massacci. Design and Results of the Tableaux-99 Non-classical (Modal) Systems Comparison. In Neil V. Murray, editor, *TABLEAUX*, volume 1617 of *LNCS*, pages 14–18. Springer, 1999.

[McC90] William McCune. Otter 2.0. In Mark E. Stickel, editor, *CADE*, volume 449 of *LNCS*, pages 663–664. Springer, 1990.

[McC97] William McCune. Solution of the Robbins Problem. *J. Autom. Reasoning*, 19(3):263–276, 1997.

[PMG97] David Poole, Alan Mackworth, and Randy Goebel. *Computational Intelligence: A Logical Approach.* Oxford University Press, 1997.

[PSS02] F. Pelletier, G. Sutcliffe, and C. Suttner. The Development of CASC. *AI Communications*, 15(2-3):79–90, 2002.

[RV01] John Alan Robinson and Andrei Voronkov, editors. *Handbook of Automated Reasoning (in 2 volumes).* Elsevier and MIT Press, 2001.

[RV02] Alexandre Riazanov and Andrei Voronkov. The design and implementation of VAMPIRE. *AI Commun.*, 15(2-3):91–110, 2002.

[Sch04] Stephan Schulz. System description: E 0.81. In David A. Basin and Michaël Rusinowitch, editors, *IJCAR*, volume 3097 of *LNCS*, pages 223–228. Springer, 2004.

[SS98] G. Sutcliffe and C. Suttner. The TPTP Problem Library: CNF Release v1.2.1. *Journal of Automated Reasoning*, 21(2):177–203, 1998.

[Wei97] Christoph Weidenbach. Spass - version 0.49. *Journal of Automated Reasoning*, 18(2):247–252, 1997.

[Wer03] Christoph Wernhard. System Description: KRHyper. Fachberichte Informatik 14–2003, Universität Koblenz-Landau, Institut für Informatik, Universitätsstr. 1, D-56070 Koblenz, 2003.

[Wer06] Christoph Wernhard. Tableaux Between Proving, Projection and Compilation. Technical report, Universität Koblenz-Landau, 2006. In preparation.