Automated Deduction Techniques for the Management of Personalized Documents

Peter Baumgartner (peter@uni-koblenz.de)* and Ulrich Furbach (uli@uni-koblenz.de)
Institut für Informatik
Universität Koblenz-Landau
Germany

Abstract. This work is about a "real-world" application of automated deduction. The application is the management of documents (such as mathematical textbooks) as they occur in a readily available tool. In this "Slicing Information Technology tool", documents are decomposed ("sliced") into small units. A particular application task is to assemble a new document from such units in a selective way, based on the user's current interest and knowledge.

It is argued that this task can be naturally expressed through logic, and that automated deduction technology can be exploited for solving it. More precisely, we rely on first-order clausal logic with some default negation principle, and we propose a model computation theorem prover as a suitable deduction mechanism.

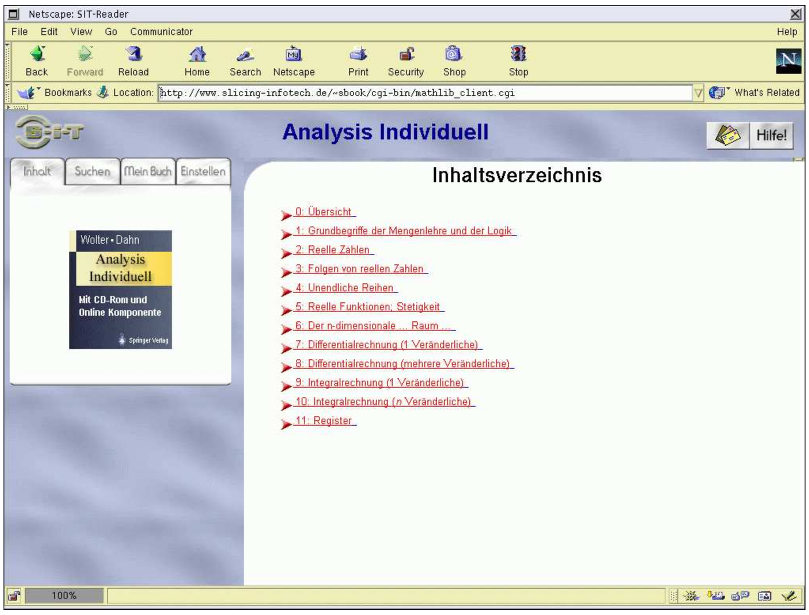
Beyond solving the task at hand as such, with this work we contribute to the quest for arguments in favor of automated deduction techniques in the "real world". Also, we argue why we think that automated deduction techniques are the best choice here.

1. Introduction

This paper is about a "real-world" application of automated deduction. The application is the management of documents (such as mathematical textbooks) that are separated ("sliced") into small units. A particular application task is to assemble a new document from such units in a selective way, based on the user's interest.

The paper concentrates on describing the task to be solved and the formalization in a predicate logic language. We also describe the calculus that we developed for this logic. Although we think that this calculus is new and has some interesting features not found in comparable approaches, its description is rather brief, since not in the main focus of this paper.

* Supported by a BMBF-grant.


Netscape: SIT-Reader
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Shop Stop
Bookmarks Location: http://www.slicing-infotech.de/~sbook/cgi-bin/mathlib_client.cgi
What's Related
Hilfe!
Inhalt Suchen Mein Buch Einstellen
Wolter • Dahn
Analysis Individuell
Mit CD-Rom und Online Komponente
Springer Verlag
100%

Analysis Individuell

Inhaltsverzeichnis

- 0: Übersicht
- 1: Grundbegriffe der Mengenlehre und der Logik
- 2: Reelle Zahlen
- 3: Folgen von reellen Zahlen
- 4: Unendliche Reihen
- 5: Reelle Funktionen; Stetigkeit
- 6: Der n-dimensionale ... Raum ...
- 7: Differentialrechnung (1 Veränderliche)
- 8: Differentialrechnung (mehrere Veränderliche)
- 9: Integralrechnung (1 Veränderliche)
- 10: Integralrechnung (*n* Veränderliche)
- 11: Register

Figure 1. The entry page of a mathematics text book in SBT.

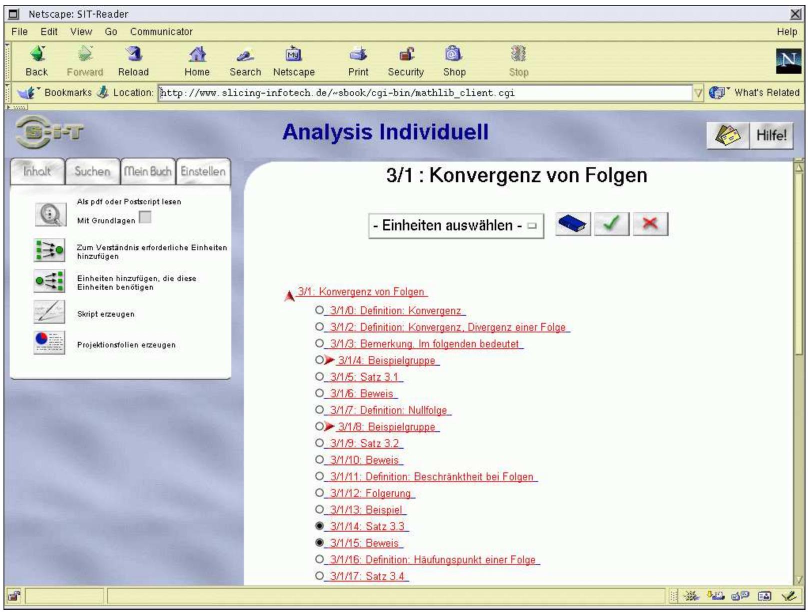
1.1. Tool and Project Context

Before describing how such a task can be formalized with logic and be solved with automated deduction techniques, it is helpful to briefly describe the tool context this work is embedded in.

This context is the *Slicing Information Technology (SIT)* tool for the management of personalized documents. With SIT, 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, and so on. 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; this is depicted in Figure 1 for `http://www.slicing-infotech.de/slib/analysis01.html` of (Dahn and Wolter, 2000).

Besides many other features the reader can decide to read e.g. Theorem 3.3, which is the slice `3/1/14`, and its proof is in slice `3/1/15`. If the reader is marking these units – as is depicted in Figure 2 – and is then clicking the "read" button on the left of the screen, she will get a PDF document containing just this part of the book.

If she now discovers that this information is not sufficient to understand the theorem and its proof, she can decide to include all the


Netscape: SIT-Reader
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Shop Stop
Bookmarks Location: http://www.slicing-infotech.de/~sbook/cgi-bin/mathlib_client.cgi
What's Related
Analysis Individuell
Hilfe!
Inhalt Suchen Mein Buch Einstellen
- Einheiten auswählen -

Als pdf oder Postscript lesen
Mit Grundlagen
Zum Verständnis erforderliche Einheiten hinzufügen
Einheiten hinzufügen, die diese Einheiten benötigen
Skript erzeugen
Projektionsfolien erzeugen

3/1 : Konvergenz von Folgen

3/1: Konvergenz von Folgen
- ○ 3/1/0: Definition: Konvergenz
- ○ 3/1/2: Definition: Konvergenz, Divergenz einer Folge
- ○ 3/1/3: Bemerkung. Im folgenden bedeutet
- ○ 3/1/4: Beispielgruppe
- ○ 3/1/5: Satz 3.1
- ○ 3/1/6: Beweis
- ○ 3/1/7: Definition: Nullfolge
- ○ 3/1/8: Beispielgruppe
- ○ 3/1/9: Satz 3.2
- ○ 3/1/10: Beweis
- ○ 3/1/11: Definition: Beschränktheit bei Folgen
- ○ 3/1/12: Folgerung
- ○ 3/1/13: Beispiel
- ● 3/1/14: Satz 3.3
- ● 3/1/15: Beweis
- ○ 3/1/16: Definition: Häufungspunkt einer Folge
- ○ 3/1/17: Satz 3.4

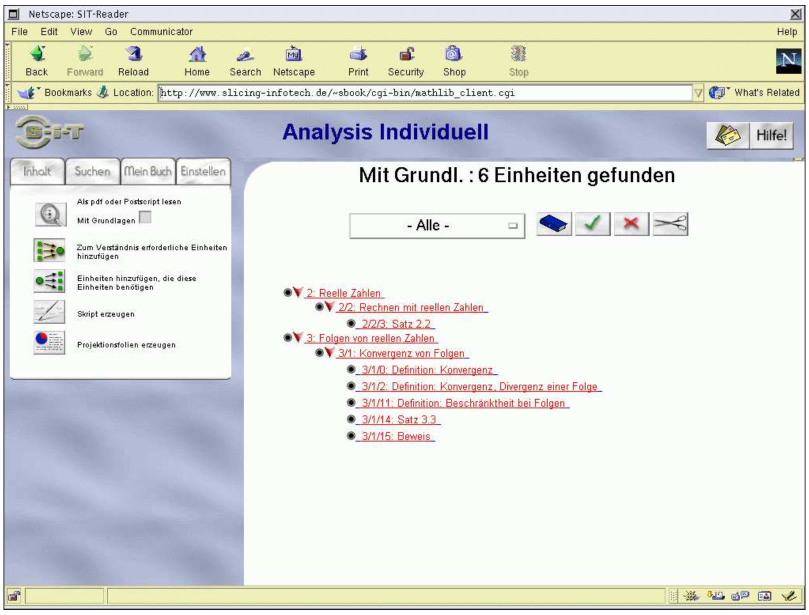
Figure 2. Selection of some units.

material necessary for the understanding of the units presented until now. Again, after clicking the appropriate button she gets the view in Figure 3, offering all the material which is computed as prerequisites for this theorem.

After marking all or some of these units, she can decide to get the corresponding document as a PDF document. Part of it is depicted in Figure 4; on the right margin of the document the identifiers of the slices are given.

In a similar way the reader can choose to include all material which uses the slices marked until then, e.g. to understand how and where a certain property is used in the rest of the chapter or the entire book. Furthermore, she can store information what parts of the book she does not want to get presented anymore, because she knows it very well. All such reasoning and computations are only possible, because besides the LaTeX sources of the units there is also a lot of meta-data stored on the server. This includes besides a glossary and keywords also information about relations of a unit to other units. Hence, we have not only the text of the book, we have an entire knowledge base about the material, which can be used by the reader.

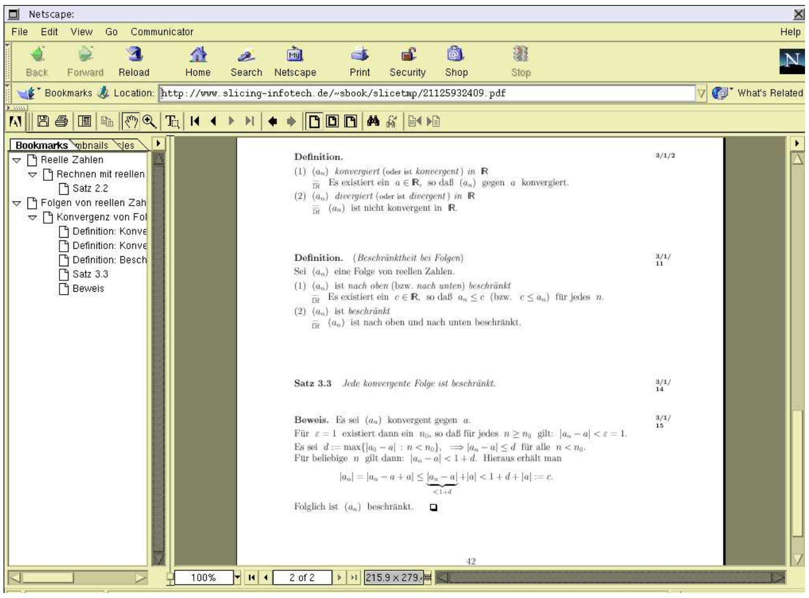
Since SIT is applied in the "real world", this knowledge base together with its reasoning mechanism has to be robust, reliable and of course efficient. SIT is applied until know to several mathematics text books, which are explored commercially by the Springer Verlag. Furthermore,


Netscape: SIT-Reader
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Shop Stop
Bookmarks Location: http://www.slicing-infotech.de/~sbook/cgi-bin/mathlib_client.cgi
What's Related
Analysis Individuell
Hilfe!
Inhalt Suchen Mein Buch Einstellen
Als pdf oder Postscript lesen
Mit Grundlagen
Zum Verständnis erforderliche Einheiten hinzufügen
Einheiten hinzufügen, die diese Einheiten benötigen
Skript erzeugen
Projektionsfolien erzeugen
- Alle -

Mit Grundl. : 6 Einheiten gefunden

- 2: Reelle Zahlen
  - 2/2: Rechnen mit reellen Zahlen
    - 2/2/3: Satz 2.2
- 3: Folgen von reellen Zahlen
  - 3/1: Konvergenz von Folgen
    - 3/1/0: Definition: Konvergenz
    - 3/1/2: Definition: Konvergenz, Divergenz einer Folge
    - 3/1/11: Definition: Beschränktheit bei Folgen
    - 3/1/14: Satz 3.3
    - 3/1/15: Beweis

Figure 3. A personalized view by including prerequisites of the selected unit.



Definition. 3/1/2

(1) (a_n) *konvergiert* (oder ist *konvergent*) *in* $\mathbb{R}$

$\overline{\overline{\text{Df}}}$ Es existiert ein $a \in \mathbb{R}$, so daß (a_n) gegen a konvergiert.

(2) (a_n) *divergiert* (oder ist *divergent*) *in* $\mathbb{R}$

$\overline{\overline{\text{Df}}}$ (a_n) ist nicht konvergent in $\mathbb{R}$.

Definition. (*Beschränktheit bei Folgen*) 3/1/11

Sei (a_n) eine Folge von reellen Zahlen.

(1) (a_n) ist *nach oben* (bzw. *nach unten*) *beschränkt*

$\overline{\overline{\text{Df}}}$ Es existiert ein $c \in \mathbb{R}$, so daß $a_n \leq c$ (bzw. $c \leq a_n$) für jedes n.

(2) (a_n) ist *beschränkt*

$\overline{\overline{\text{Df}}}$ (a_n) ist nach oben und nach unten beschränkt.

Satz 3.3 *Jede konvergente Folge ist beschränkt.* 3/1/14

Beweis. Es sei (a_n) konvergent gegen a. 3/1/15

Für $\varepsilon = 1$ existiert dann ein n_0, so daß für jedes $n \geq n_0$ gilt: $|a_n - a| < \varepsilon = 1$.

Es sei $d := \max\{|a_0 - a| : n < n_0\}$, $\Longrightarrow |a_n - a| \leq d$ für alle $n < n_0$.

Für beliebige n gilt dann: $|a_n - a| < 1 + d$. Hieraus erhält man

$$|a_n| = |a_n - a + a| \leq \underbrace{|a_n - a|}_{<1+d} + |a| < 1 + d + |a| := c.$$

Folglich ist (a_n) beschränkt. ❑

42

Figure 4. A PDF view of the result in Figure 3.

SIT is the technical basis of the TRIAL-SOLUTION project[1]. The TRIAL-SOLUTION project aims to develop a technology for the generation of personalized teaching materials – notably in the field of mathematics – from existing documents (cf. `www.trial-solution.de` for more details).

Current work on SIT within the TRIAL-SOLUTION context is concerned with techniques to extend the capabilities by handling knowledge coming from various sources. In our previous example from Figure 1, e.g. it could be the case that the reader decides that she wants to get examples related to the material she is studying, however from another book, which she knows to be much more of the introductory kind. By this, she could have a much better chance to understand the material. This approach is motivated by the imagination of a reader standing in front of a shelf of books in a library and searching the material she needs for her work: she is using different sources for her search, like books, table of contents, catalogues or book reviews. In our case of SIT, these sources include (i) different sliced books, (ii) a knowledge base of meta data on content (e.g. by keywords), didactic features and interoperability interfacing, (iii) the user profile, including e.g. information about units known to her, and (iv) thesauri that help to categorize and connect knowledge across different books.

All these sources are to be taken into account when generating a personalized document. So far, no language was available to us to formulate in a "nice" way (convenient, friendly to change and adapt to new needs, efficient, ...) the computation of the personalized documents. Our approach based on logic was heavily motivated to come to a solution here.

1.2. Formalizing the Application Domain

In our approach, the document to be generated is computed by a model generating theorem prover. In Figure 5 the entire knowledge representation and reasoning task is depicted. On the left hand side we see several books, which can be used as sources for the query given by the user on the right hand side of the picture. Besides the various books, the system takes into account a number of different knowledge sources: keywords, an ontology giving relations between the keywords and terms and dependencies between units with respect of a refers and requires relation. All this knowledge is given as metadata of the books and can be used by the deduction system together with additional data about the user to answer the query.

[1] TRIAL-SOLUTION is funded by the EU as part of its Information Society Technologies Programme (IST) within the EU's Fifth RTD Framework Programme.

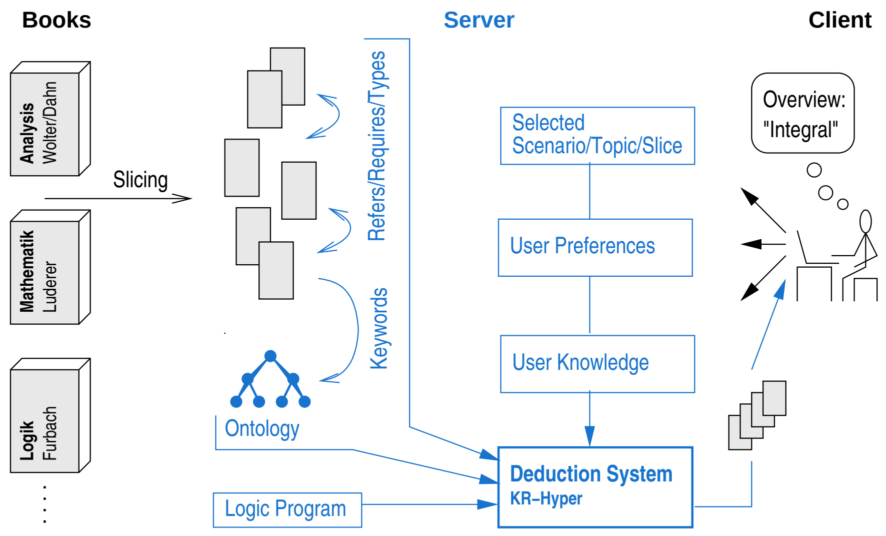


Figure 5. The reasoning part of SBT

In the sequel we describe the formalization with logic as used in our system. Thereby we concentrate on the background theory. This is the part that essentially controls what units go into the final document. We do so in order to highlight some features of the specification, and to motivate our approach taken.

The computation of the final document is triggered by marking some unit U as a "selected unit". The background theory is essentially a specification of units to be included into the generated document. Such a specification is from now on called a *query*, and the task to generate the document from the query and the selected unit is referred to as *solving the query*.

Here is a sample query:

(i) For each keyword K attached to the selected unit U, include in the generated document some unit D that is categorized as a definition of K; in case there is more than such unit, prefer one from book A to one from book B.

(ii) Include all the units that have at least one keyword in common with the keywords of U and that are of explanatory type (examples, figures, etc).

(iii) Include all the units that are required by U.

(iv) Include U.

It is important to note that such queries are *not* intended to be written by the end-user. Instead, the end-user only enters some selected units U and choses one of the several predefined scenarios (like "I want to prepare for a written exam"), and each scenario stands for a query.

In our experiments we use sliced versions of two mathematics text books. Here are two sample units, one from each book:

Ident:	`0/1/2/1`
Text:	`Mengentheoretische Grundbegriffe ...`
Book:	`Wolter/Dahn`
Type:	`Merksatz`
Refers:	`0/1/0/2, 0/1/0/3, ...`
Requires:	
Keys:	`set, set intersection, set equality, ...`

Ident:	`1/1/1/1/0`
Text:	`\definition{ \textbf{Definition}: Eine \textbf{Menge} ...`
Book:	`Gellrich/Gellrich`
Type:	`Definition`
Refers:	
Requires:	
Keys:	`set`

The *Ident* field contains a unique name of the unit in the form of a Unix file system subpath, matching the hierarchically organization of the units according to the books sectioning structure. The *Text* field contains the unit's text. The *Book* field contains the name of the book's authors. The *Type* field denotes the class the unit belongs to. The *Refers* and *Requires* fields contain dependencies from other units. The *Keys* field contains a set of keywords describing the contents of the unit.

Now suppose that the unit with Ident `0/1/2/1` is selected, and that the query from above is to be solved. Some aspects in a logical formalization of this task are straightforward, like the representation of the units and the representation of the selected unit identifier as a fact (as in `selected_unit(0/1/2/1)`). There full formalization is not included here. In order to motivate the whole approach, highlighting some parts of it might be helpful, though. Each of the following four parts demonstrates a different aspect of the formalization[2].

1.2.0.1. *First-Order Specifications.* In the field of knowledge representation it is common practice to identify a clause with the set of its ground instances. Reasoning mechanisms often suppose that these sets are finite, so that essentially propositional logic results. Such a restriction should not be made in our case. Consider the following clauses:

[2] Written in a Prolog-style notation. In brief, the `:-` symbol stands for the junctor $\leftarrow$, variables start with an uppercase letter or the symbol `_`; numbers, words starting with a lowercase letter, and strings enclosed in `'`-quotes are constants. A standard textbook on Prolog is (Clocksin and Melish, 1981)

```
wolter_dahn_unit(0/_ALL_).
gellrich_gellrich_unit(1/_ALL_).
equal(X, X).
```

The first two facts test if a given unit identifier denotes a unit from the respective book. The `_ALL_` symbol stands for an anonymous, universally quantified variable. Due to the `/`-function symbol (and probably others) the Herbrand-Base is infinite. Certainly it is sufficient to take the set of ground instances of these facts up to a certain depth imposed by the books. However, having thus exponentially many facts this option is not really a viable one.

None of the stated facts satsifies *range restrictedness* (cf.(Manthey and Bry, 1988)), a syntactical restriction, that many systems impose for programs to be admissible. A workaround, which can be taken then, is to enumerate the Herbrand-Base during proof search. This means to consider all ground terms for the variables, which does not look too prospective.

Some facts could conceivably be handled in such systems by building in the semantics of, e.g., `equal` into the prover. However, there remain cases where this is hardly possible. A typical example is a clause like the following, which seems perfectly plausibly in our scenario:

```
knows_units(User,1/1/_ALL_) :-
    current_user(User),
    learned(User,'set').
```

Another example of this kind, taken from the actual specification, is depicted in Figure 7 below.

In sum, we have an "essential" non-ground specification.

1.2.0.2. *Non-classical Negation.* Consider the following clauses:

```
computed_unit(UnitId) :-          multiple_def(Key) :-
    candidate_def(UnitId,Key),        candidate_def(UnitId1,Key),
    not multiple_def(Key).            candidate_def(UnitId2,Key),
                                      not equal(UnitId1,UnitId2).
```

The `computed_unit` relation shall contain those units (named by unit identifiers `UnitId`) that go into the generated document. According to the left clause, this applies to any candidate definition unit for some keyword `Key` (derived by some other other clauses not described here), provided there is not more than one such candidate definition of `Key`. The second clause states the definition of the `multiple_def`-relation.

What is the intended semantics of `not`? *Classical* semantics is not appropriate, as it allows, for instance, arbitrary different terms to hold in the `equal`-relation. The correct intention, however, of `equal` is to mean syntactical equality. Likewise, classical semantics allows for counter-intuitive interpretations of `multiple_def`.

The *supported model* semantics (cf. Section 2.2), or possibly a stronger semantics (one that allows for more conclusions) seems to be intuitively correct. In brief, every atom true in a supported model must be justified by some clause, which means that the atom occurs in the head of some clause where the body is true in the model.

1.2.0.3. *Disjunctions and Integrity Constraints.* Consider the following two clauses:

```
computed_unit(UnitId1) ;         false :-
computed_unit(UnitId2) :-            computed_unit(UnitId1),
    candidate_def(UnitId1,Key),      computed_unit(UnitId2),
    candidate_def(UnitId2,Key),      candidate_def(UnitId1,Key),
    not equal(UnitId1, UnitId2).     candidate_def(UnitId2,Key),
                                     not equal(UnitId1,UnitId2).
```

The left clause states that if there is more than one candidate definition unit of some `Key`, then at least one of them must go into the generated document (the symbol `;` means "or"). The right clause states that not both of them must go into the generated document.

These clauses demonstrate that Horn clause logic is not sufficient. Any number of positive literals may appear in clauses.

2. Automated Deduction

When employing automated deduction techniques to solve a query like the one at the beginning of Section 1.2, some questions come up immediately: (a) what is an appropriate logic (syntax and semantics)? (b) How does a respective calculus look like? (c) is it efficient enough to solve real-world problems? (d) Why to do it at all with logic, and not, say, write a Prolog-program?

The most effort in our current project work is spent to answer questions (a) and (b). Since not really relevant for this publication, the calculus is only sketched here. Although question (d) is addressed here, some more investigations are certainly necessary in this regard.

In brief, our approach is completely declarative, and hence we believe that there are advantageous over more procedural oriented techniques.

Question (c) has to be answered by practice. At the moment, we have a prototypical implementation in Prolog, which, however, is coupled with a term indexing library for faster access to the base relations. In Section 3 a little more about our current experiences is reported.

2.1. Nonmonotonic and Classical Logics

On a higher, research methodological level the work presented here is intended as a bridging-the-gap attempt: for many years, research in *logic-based knowledge representation and logic programming* has been emphasizing theoretical issues, and one of the best-studied questions concerns the semantics of default negation[3]. The problems turned out to be extraordinarily difficult, which warrant their study in detail. Fortunately, much is known today about different semantics for knowledge representation logics and logic programming. There is a good chance that a logic suitable for a particular application has been developed and studied in greatest detail.

Concerning research in *first-order classical reasoning*, the situation has been different. Not only theoretical issues, but also the design of theorem provers has traditionally received considerable attention. Much is known about efficient implementation techniques, and highly sophisticated implementations are around (e.g. SETHEO (Goller et al., 1994), SPASS (Weidenbach et al., 1999)). Annual, international competitions are held to crown the "best" prover.

In essence, in this work we take as starting point a calculus originally developed for first-order classical reasoning – hyper tableaux (Baumgartner et al., 1996) – and modify it to compute models for a certain class of logic programs.

2.2. Logic Programming Semantics

We are using a sufficiently powerful logic, such that the queries can be stated in a comfortable way. In this section we describe some issues from logic programming and knowledge representation, which we built into the deduction system ((Dix et al., 2001) is an overview article covering the issues discussed here). In particular, including a default negation principle and disjunctions in our logic turned out to facilitate the formalization (cf. Section 1.2 above). As a restriction, we insist on stratified specifications, which turns out not to be a problem. Stratified

[3] Like, for instance, Prolog's *Negation by finite failure* operator.

specifications are also called "logic programs" or simply "programs" in the sequel[4].

We define a special case of the *supported model* semantics [5] as the intended meaning of our programs, and solving the query means to compute such a model for the given program. This point is worth emphasizing: we are thus *not* working in a classical theorem-proving (i.e. refutational) setting; solving the query is, to our intuition, more naturally expressed as a model-generation task.

Fortunately, model computation for stratified programs is much easier than for non-stratified programs, both conceptually and in a complexity-theoretic sense. There is little dispute about the intended meaning of stratified programs, at least for *normal* programs (i.e. programs without disjunctions in the head), and the two major semantics coincide, which are the stable model semantics (Gelfond and Lifschitz, 1988) and the well-founded model semantics (Van Gelder et al., 1991). For propositional stratified normal programs, a polynomial time decision procedure for the model existence problem exists, which does not exist for the stable model semantics for non-stratified normal programs. Being confronted with large sets of data (stemming from tens of thousands of units) was the main motivation to strive for a tractable semantics.

As mentioned above, we do not restrict ourselves to normal programs. Instead, we found it convenient to allow disjunctions in the head of clauses in order to express degrees of freedom for the assembly of the final documents. Also for the disjunctive case (i.e. non-normal programs), stable model semantics and well-founded model semantics have been defined (see e.g. (Dix et al., 2001)). Both semantics agree to assign a *minimal model* semantics to disjunction. For instance, the program

$$A \vee B \leftarrow$$

admits two minimal models, which are $\{A\}$ and $\{B\}$. An equivalent characterization of minimal models is to insist that for each atom true in the intended model, there is a head of a true clause where *only* this

[4] More precisely, the program clauses have the form $A_1 \vee \cdots \vee A_k \leftarrow B_1 \wedge \cdots \wedge B_m \wedge$ not $B_{m+1} \wedge \cdots \wedge$ not B_n, and a program is *stratified* if the call graph of the program does not have a cycle through a negative body literal. A simple example for a non-stratified program is $\{A \leftarrow B,\ B \leftarrow \text{not} A\}$, because A is defined recursively in terms of its own negation.

[5] A model $\mathcal{I}$ of a (stratified and ground) clause set M is a *supported model* of M iff for every $A \in \mathcal{I}$ there is a clause $A \vee A_1 \vee \cdots \vee A_k \leftarrow B_1 \wedge \cdots \wedge B_m \wedge$ not $B_{m+1} \wedge \cdots \wedge$ not B_n in M such that $M \models B_1 \wedge \cdots \wedge B_m \wedge \neg B_{m+1} \wedge \cdots \wedge \neg B_n$.

atom is true. For our task, however, this preferred exclusive reading of disjunctions seems not necessarily appropriate. When assembling documents, redundancies (i.e. non-minimal models) should not be prohibited unless explicitly stated. In order not having to restrict to the minimal model semantics, we find the *possible model semantics* to be appropriate (Sakama, 1990). With it, the above program admits all the obvious models $\{A\}$, $\{B\}$ and $\{A, B\}$. If the inclusive reading is to be avoided, one would have to add the integrity constraint

$$\leftarrow \ A \wedge B \ .$$

Unfortunately, the possible model semantics is costly to implement. At the current state of our developments, our calculus is *sound* wrt. the possible model semantics (i.e. any computed model is a possible model) but *complete* only in a weak sense (if a possible model exists at all, some possible model will be computed). In our current setting, solving the query means to compute *any* model of the program, so the lack of completeness is not really harmful. However, future experiments will have to show if this approach is really feasible.

2.3. The Calculus

One of the big challenges in both classical logic and nonmonotonic logics is to design calculi and efficient procedures to compute models for *first-order* specifications. Some attempts have been made for classical first-order logic, thereby specializing on decidable cases of first-order logic and/or clever attempts to discover loops in derivations of standard calculi (see e.g. (Fermller and Leitsch, 1996; Peltier, 1999; Baumgartner, 2000; Stolzenburg, 1999)).

In the field of logic programming, a common viewpoint is to identify a program with the set of all its ground instances and to apply propositional methods then. Notable exceptions are described (Bornscheuer, 1996; Eiter et al., 1997; Gottlob et al., 1996; Dix and Stolzenburg, 1998). Of course, the "grounding" approach is feasible only in restricted cases, when reasoning can be guaranteedly restricted to a finite subset of the possibly infinite set of ground instances. Even the best systems following this approach, like the S-models system (Niemel and Simons, 1996), quite often arrive at their limits when confronted with real data.

In our application we are confronted with data sets coming from tens of thousands of units. Due to this mass, grounding of the programs before the computation starts seems not to be a viable option. Therefore, our calculus directly computes models, starting from the given program, and without grounding it beforehand. In order to make this work for the case of programs with default negation, a novel technique for the representation of and reasoning with non-ground representations of interpretations is developed.

The calculus developed here is obtained by combining features of two calculi readily developed – *hyper tableaux* (Baumgartner et al., 1996) and *FDPLL* (Baumgartner, 2000) – and some further adaptations for default negation reasoning. These two calculi were developed for *classical* first-order reasoning, and the new calculus can be seen to bring in “a little monotonicity” to hyper tableaux.

The calculus is called *hyper tableau* because it combines two characteristics: the idea of clustering certain basic inference rules into a single one, as it is used in hyper-resolution (Robinson, 1965), and the overall calculus as it was developed for clause normal form tableau (see (Letz, 1998)). Instead of defining the hyper tableau calculus formally we will illustrate it with the following example.

Consider the following set of clauses, where clauses are given in implication form, such that $B_1 \vee \cdots \vee B_m \leftarrow A_1 \wedge \cdots \wedge A_n$ stands for the clause $B_1 \vee \cdots \vee B_m \vee \neg A_1 \vee \cdots \vee \neg A_n$.

$$\begin{aligned} A &\leftarrow A & (1)\\ B \vee C &\leftarrow A & (2)\\ A \vee D &\leftarrow C & (3)\\ &\leftarrow A \wedge B & (4) \end{aligned}$$

In order to construct a hyper tableau for this clause set, we start with the empty tableau ϵ, which is given in the left part of Figure 6. We will discuss this derivation from left to right: If we consider clause (1), which we can understand as ”in any model A has to hold”, hence we extend our single (empty) branch of the tableau ϵ by the new leaf A. We arrive at a tableau with a branch which contains the (possibly) partial model A. Obviously clause (2) does not hold in this model, because (2) is stating ”if in a model A holds, than B or C has to hold as well”. Let’s repair this, by extending our tableau by these two possibilities; we extend it by the disjunction $B \vee C$, which is expressed in the tableau by a new branching. The left branch $\{A, B\}$ of this new tableau, again is a (possibly) partial model, but now we observe that there is a contradiction to clause (4), which is stating that A and B cannot be true together in any model; hence we know that this branch does not correspond to a partial model – we mark it as closed with an asterisk. The right branch of the tableau, however, although it could be further extended, is a model of the entire clause set (1)– (4).

We demonstrated the calculus only in the propositional case, but it can be extended to a complete and correct calculus for full first order

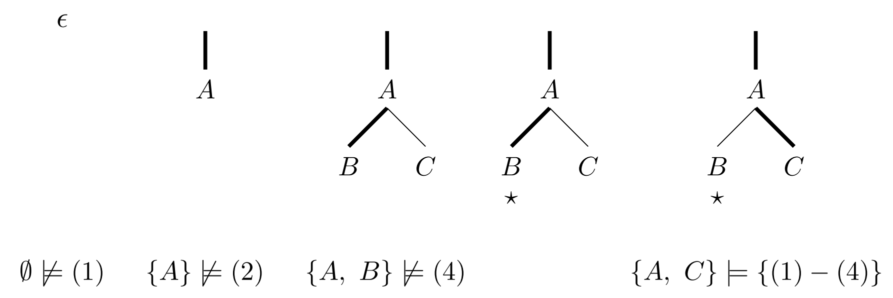

ϵ
A
B
C
$\star$
$\emptyset \not\models (1)$
$\{A\} \not\models (2)$
$\{A,\ B\} \not\models (4)$
$\{A,\ C\} \models \{(1) - (4)\}$

Figure 6. A sample hyper tableau derivation.

clausal logic, and there are various improvements of its basic variant as introduced in (Baumgartner et al., 1996).

Hyper tableau calculi are tableau calculi in the tradition of SATCHMO (Manthey and Bry, 1988). In essence, interpretations as candidates for models of the given clause set are generated one after another, and the search stops as soon as a model is found, or each candidate is provably not a model (refutation). A distinguishing feature of the hyper tableau calculi (Baumgartner et al., 1996; Baumgartner, 1998) to SATCHMO and related procedures is the representation of interpretations at the first-order level. For instance, given the very simple clause set consisting of the single clause

$$\mathrm{equal}(X, X) \ \leftarrow$$

the calculus stops after one step with the model described by the set $\{\mathrm{equal}(X, X)\}$, which stands for the model that assigns true to each ground instance of $\mathrm{equal}(X, X)$.

The hyper tableau calculi developed so far do not allow for default negation. In the present work we therefore extend the calculus correspondingly. At the heart is a modified representation of interpretations. The central idea is to replace atoms – which stand for the set of all their ground instances – by pairs $A - \{E_1, \ldots, E_n\}$, where A is an atom as before, and E is a set of atoms ("Exceptions") that describes which ground instances of A are *excluded* from being true by virtue of A. For instance, if the clause

$$\mathrm{different}(X, Y) \ \leftarrow\ \mathrm{not\ equal}(X, Y)$$

is added, then the search stops with the set

$$\{\mathrm{equal}(X, X) - \{\}, \mathrm{different}(X, Y) - \{\mathrm{different}(X, X)\}\}\ .$$

It represents the model where all instances of $\mathrm{equal}(X, X)$ are true, and all instances of $\mathrm{different}(X, Y)$ are true, except the reflexive ones.

To see how we use the calculus in the context of SBT, we take another excerpt from the knowledge base, and use it to demonstrate the working of the hyper tableau calculus modified for nonmonotonic reasoning. Figure 7 depicts this excerpt; it is from the "user model", and its purpose is to find out by means of the `known_unit_inf` relation whether the user knows some unit in question. The clauses in the

```
%% User knowledge:
known_unit(analysis/1/2/_ALL_).                              (1)
unknown_unit(analysis/1/2/1).                                (2)

%% Book metadata:
refers(analysis/1/2/3, analysis/1/0/4).                      (3)

%% 'known_unit' transitive closure:
known_unit(Book_B/Unit_B) :-                                 (4)
    known_unit(Book_A/Unit_A),
    refers(Book_A/Unit_A, Book_B/Unit_B).

%% 'unknown_unit' transitive closure
unknown_unit(Book_B/Unit_B) :-                               (5)
    unknown_unit(Book_A/Unit_A),
    refers(Book_A/Unit_A, Book_B/Unit_B).

%% Derived:
known_unit_inf(Book/Unit) :-                                 (6)
    known_unit(Book/Unit),
    not unknown_unit(Book/Unit).
```

Figure 7. Excerpt from the knowledge base.

knowledge base in Figure 7 are stratified, i.e. they are organized in a hierarchical way. On the base layer, e.g., clause (2) is expressing the fact that the current user does not know the unit `1/2/1` from the `analysis`-book. Clause (1) is a good example demonstrating the representational advantage of our system over other systems that require grounding of the clauses before computation (this is also mentioned inSection 1.2 above): in clause (1), `_ALL_` is a universally quantified variable, i.e., intentionally, all sub-units of `analysis/1/2` are declared to be known. It is the purpose of clause (6) to resolve the apparent inconsistency behind the just given explanation of clauses (1) and (2) (cf. below).

Clauses (4) and (5) are expressing knowledge about the meta data relations "known_unit" and "unknown_unit".

Clause (6) says that `Book/Unit` is contained in the `knows_unit_inf`-relation (the relation we are interested in), if it is “known” (by means of the `known_unit(Book/Unit)` declaration) *and* this circumstance is not overridden by a an explicit “unknown”-declaration of the same unit (by means of `not unknown_unit(Book/Unit)`). By the use of the default negation technique we realize that “unknown”-declarations should be stronger than “known”-declarations. We think that this is appropriate modelling, as “unknown” units are never withhold from the user.

Now, the calculus derives from the clauses in Figure 7 in three steps the hyper tableau in Figure 8.

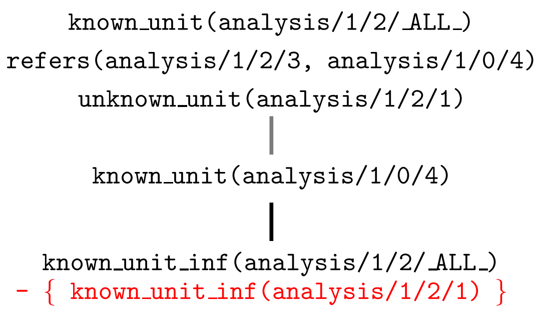

known_unit(analysis/1/2/_ALL_)
refers(analysis/1/2/3, analysis/1/0/4)
unknown_unit(analysis/1/2/1)
known_unit(analysis/1/0/4)
known_unit_inf(analysis/1/2/_ALL_)
- { known_unit_inf(analysis/1/2/1) }

Figure 8. Hyper tableau derivation from the clauses in Figure 7.

The topmost three lines stem from the clauses (1), (3) and (2), respectively. By combining clauses (1), (3) and (4) we conclude that the unit `analysis/1/0/4` should be “known”. This is realized by the hyper tableau derivation in the fourth line. The concluding line in the derivation is obtained by clauses (1), (2) and (6). It says that in the `known_unit_inf`-relation are all sub-units of `analysis/1/2` – more technically: all ground instances of `analysis/1/2/_ALL_` – except for the unit `analysis/1/2/1` (the technique of representing models this way is described above). Observe that the mentioned, apparent contradiction between what clauses (1) and (2) say is eliminated as explained.

2.4. Other Approaches

In the previous sections, specifications in the (disjunctive) logic programming style are advocated as an appropriate formalism to model the task at hand. Undoubtedly, there are other candidate formalisms that seem well-suited, too. In the following we comment on these.

2.4.0.4. *Prolog.* Certainly, one could write a Prolog program to solve a query. When doing so, it seems natural to rely on the `findall` built-in to compute the extension of the `computed_unit` predicate, i.e. the solution to a query. Essentially, this means to enumerate and collect all solutions of the goal `computed_unit(U)` by means of the Prolog built-in backtracking mechanism. In order to make this work, some precautions have to be taken. In particular *explicit loop checks* would have to be programmed in order to let `findall` terminate. Because otherwise, for instance, alone the presence of a transitivity clause causes `findall` not to terminate.

It is obvious that a Prolog program along these lines would be much more complicated than the program in Section 1.2. On the other hand, our approach relieves the programmer from the burden of explicitly programming a loop mechanism, because it is built into the model computation procedure presented below. Indeed, this is a distinguishing and often mentioned advantage of virtually all bottom-up model generation procedures over Prolog.

2.4.0.5. *XSB-Prolog.* One of the view programming languages that works top-down (as Prolog) and that has built-in loop checking capabilities (as bottom-up model generation procedures) is *XSB-Prolog* (Sagonas et al., 2000). XSB-Prolog supports query answering wrt. the well-founded semantics for normal logic programs (Van Gelder et al., 1991). At the heart of XSB-Prolog is the so-called *tabling* device that stores solutions (instantiations) of goals as soon as computed. Based on tabling, it is even possible to compute extensions of predicates (such as `computed_unit`) and return them to the user.

The only problem with XSB-Prolog for our application is the restriction to *normal* programs, i.e. disjunctions in the head of program clauses are not allowed. Certainly, this problem could be circumvented by explicitly coding disjunctions in the program, but possibly at the cost of far less intuitive solution.

2.4.0.6. *Description Logics.* Description logics (DL) are a formalism for the representation of hierarchically structured knowledge about individuals and classes of individuals. Nowadays, numerous descendants of the original $\mathcal{ALC}$ formalism and calculus (Schmidt-Schau and Smolka, 1991, e.g.) with greatly enhanced expressive power exist, and efficient respective systems to reason about DL specifications have been developed (Horrocks et al., 2000).

From the sample query in Section 1.2 it can be observed that a good deal of the information represented there would be accessible to a DL formalization. The concrete units would form the so-called assertional part (A-Box), and general “is-a” or “has-a” knowledge would form the terminological part (T-Box). The T-Box would contain, for instance, the knowledge that a unit with type "example" *is-a* "explanatory unit", and also that a unit with type "figure" *is-a* "explanatory unit". Also, transitive relations like "requires" should be accessible to DL formalisms containing transitive roles.

In Section 1.2 it is argued that disjunctive and non-monotonic reasoning is suitable to model e.g. preference among units to be selected. At the current state of our work, however, it is not yet clear to us if and how this would be accomplished using a DL formalism. Certainly, much more work has to be spent here. Presumably, one would arrive at a *combined* DL and disjunctive logic programming approach. This is left here as future work.

3. Status of This Work and Perspectives

This work is not yet finished. Open ends concern in the first place some design decisions and practical experience with real data on a large scale.

Concerning design decisions, for instance, it is not quite clear what semantics suits our application best. It is clear that a supportedness principle (Section 2.2) is needed, but there is some room left for further strengthenings. Further experiments (i.e. the formulation of queries) will guide us here.

The calculus and its implementation are developed far enough, so that meaningful experiments are possible. The implementation is carried out in Eclipse Prolog. For faster access to base relations, i.e. the currently computed model candidate, the discrimination tree indexing package from the ACID term indexing library (Graf, 1994) is coupled. Without indexing, even our moderately sized experiments seem not to be doable. With indexing, the response time for the sample query in Section 1.2 with a database stemming from about 100 units takes less than a second. A similar query, applied to a full book with about 4000 units takes ten seconds, which seems almost acceptable.

Another topic that has to be addressed seriously is how to design the interface between the logic programming system and the user, i.e. the reader of the document. The user cannot be expected to be acquainted with logic programming or any other formal language (typically, this is what students should *learn* by reading the sliced books). Thus, a simplified, restricted set of commands together with a library of predefined "subroutines" seems to be an option. These would have to become part of the "reader" tool, so that the user can assemble queries by his own in a plug-and-play manner.

The question arises, if the techniques developed within this enterprise can be used in some other context. We plan to do so within one of the projects carried out in our group, the *In2Math* project. This project aims, roughly, to combine documents like sliced books with interactive systems like computer algebra systems and theorem provers. The projects targets at seriously using the resulting systems in undergraduate logic and mathematics courses. We are convinced that the techniques developed here can be used in the In2Math project. Beyond the obvious purpose to help students assemble personalized books, it is maybe possible to *teach* students participating in a logic course the logic programming techniques described here.

References

Baumgartner, P.: 1998, 'Hyper Tableaux — The Next Generation'. In: H. de Swaart (ed.): *Automated Reasoning with Analytic Tableaux and Related Methods*, Vol. 1397 of *Lecture Notes in Artificial Intelligence*. pp. 60–76, Springer.

Baumgartner, P.: 2000, 'FDPLL – A First-Order Davis-Putnam-Logeman-Loveland Procedure'. In: D. McAllester (ed.): *CADE-17 – The 17th International Conference on Automated Deduction*, Vol. 1831 of *Lecture Notes in Artificial Intelligence*. pp. 200–219, Springer.

Baumgartner, P., U. Furbach, and I. Niemelä: 1996, 'Hyper Tableaux'. In: *Proc. JELIA 96*. Springer.

Bornscheuer, S.-E.: 1996, 'Rational Models of Normal Logic Programs'. In: S. H. Günther Görz (ed.): *KI-96: Advances in Artificial Intelligence*, Vol. 1137 of *Lecture Notes in Artificial Intelligence*. pp. 1–4, Springer Verlag, Berlin, Heidelberg, New-York.

Clocksin, W. F. and C. S. Melish: 1981, *Programming in PROLOG*. Springer Verlag, Berlin, Heidelberg, New-York.

Dahn, I. and H. Wolter: 2000, *Analysis Individuell*. Springer Verlag.

Dix, J., U. Furbach, and I. Niemelä: 2001, 'Nonmonotonic Reasoning: Towards Efficient Calculi and Implementations'. In: A. Voronkov and A. Robinson (eds.): *Handbook of Automated Reasoning*. Elsevier-Science-Press, pp. 1121–1234.

Dix, J. and F. Stolzenburg: 1998, 'A Framework to Incorporate Non-Monotonic Reasoning into Constraint Logic Programming'. *Journal of Logic Programming* **37**(1-3), 47–76. Special Issue on *Constraint Logic Programming*. Guest editors: Kim Marriott and Peter J. Stuckey.

Eiter, T., J. Lu, and V. S. Subrahmanian: 1997, 'Computing Non-Ground Representations of Stable Models'. In: J. Dix, U. Furbach, and A. Nerode (eds.): *Proceedings of the 4th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR-97)*. pp. 198–217, Springer-Verlag.

Fermller, C. and A. Leitsch: 1996, 'Hyperresolution and Automated Model Building'. *Journal of Logic and Computation* **6**(2), 173–230.

Gelfond, M. and V. Lifschitz: 1988, 'The Stable Model Semantics for Logic Programming'. In: R. Kowalski and K. Bowen (eds.): *Proceedings of the 5th International Conference on Logic Programming, Seattle*. pp. 1070–1080.

Goller, C., R. Letz, K. Mayr, and J. Schumann: 1994, 'SETHEO V3.2: Recent Developments — System Abstract —'. In: A. Bundy (ed.): *Automated Deduction — CADE 12*. Nancy, France, pp. 778–782, Springer-Verlag.

Gottlob, G., S. Marcus, A. Nerode, G. Salzer, and V. S. Subrahmanian: 1996, 'A Non-Ground Realization of the Stable and Well-Founded Semantics'. *Theoretical Computer Science* **166**(1-2), 221–262.

Graf, P.: 1994, 'ACID User Manual - Version 1.0'. Technical Report MPI-I-94-DRAFT, Max-Planck-Institut, Saarbrcken, Germany.

Horrocks, I., U. Sattler, and S. Tobies: 2000, 'Practical Reasoning for Very Expressive Description Logics'. *Logic Journal of the IGPL* **8**(3), 239–263.

Letz, R.: 1998, 'Clausal Tableaux'. In: W. Bibel and P. H. Schmitt (eds.): *Automated Deduction. A Basis for Applications*. Kluwer Academic Publishers.

Manthey, R. and F. Bry: 1988, 'SATCHMO: a theorem prover implemented in Prolog'. In: E. Lusk and R. Overbeek (eds.): *Proceedings of the 9^{th} Conference on Automated Deduction, Argonne, Illinois, May 1988*, Vol. 310 of *Lecture Notes in Computer Science*. pp. 415–434, Springer.

Niemel, I. and P. Simons: 1996, 'Efficient Implementation of the Well-founded and Stable Model Semantics'. In: *Proceedings of the Joint International Conference and Symposium on Logic Programming*. Bonn, Germany, The MITPress.

Peltier, N.: 1999, 'Pruning the Search Space and Extracting More Models in Tableaux'. *Logic Journal of the IGPL* **7**(2), 217–251.

Robinson, J. A.: 1965, 'Automated deduction with hyper-resolution'. *Internat. J. Comput. Math.* **1**, 227–234.

Sagonas, K., T. Swift, and D. S. Warren: 2000, 'An Abstract Machine for Computing the Well-Founded Semantics'. *Journal of Logic Programming*. To Appear.

Sakama, C.: 1990, 'Possible Model Semantics for Disjunctive Databases'. In: W. Kim, J.-M. Nicholas, and S. Nishio (eds.): *Proceedings First International Conference on Deductive and Object-Oriented Databases (DOOD-89)*. pp. 337–351, Elsevier Science Publishers B.V. (North–Holland) Amsterdam.

Schmidt-Schau, M. and G. Smolka: 1991, 'Attributive Concept Descriptions with Complements'. *Artificial Intelligence* **48**(1), 1–26.

Stolzenburg, F.: 1999, 'Loop-Detection in Hyper-Tableaux by Powerful Model Generation'. *Journal of Universal Computer Science* **5**(3), 135–155. Special Issue on *Integration of Deduction Systems*. Guest editors: Reiner Hähnle, Wolfram Menzel, Peter H. Schmitt and Wolfgang Reif. Springer, Berlin, Heidelberg, New York.

Van Gelder, A., K. A. Ross, and J. S. Schlipf: 1991, 'The well-founded semantics for general logic programs'. *Journal of the ACM* **38**, 620–650.

Weidenbach, C., B. Afshordel, U. Brahm, C. Cohrs, T. Engel, E. Keen, C. Theobalt, and D. Topić: 1999, 'System Description: SPASS Version 1.0.0'. In: H. Ganzinger (ed.): *CADE-16 – The 16th International Conference on Automated Deduction*, Vol. 1632 of *Lecture Notes in Artificial Intelligence*. Trento, Italy, pp. 378–382, Springer.