Challenges of Model Generation for Natural Language Processing

Seminar für Sprachwissenschaft
Eberhard Karls Universität Tübingen

Author:
Johannes Dellert
johannes.dellert@gmx.de

Course:
Algorithmic Semantics

Supervisor:
PD Dr. Frank Richter

Hiermit versichere ich, dass ich die vorgelegte Arbeit selbstständig und nur mit den angegebenen Quellen und Hilfsmitteln (einschließlich des WWW und anderer elektronischer Quellen) angefertigt habe. Alle Stellen der Arbeit, die ich anderen Werken dem Wortlaut oder dem Sinne nach entnommen habe, sind kenntlich gemacht.

(Johannes Dellert)

Contents

1 Introduction

Model generation is a discipline of automated reasoning that is concerned with the explicit computation of models for logical formulae. The tools developed in the field therefore offer a positive handle on the satisfiability of logical theories. Model generation is potentially very useful because it provides direct and automated access to models of axiomatizations and countermodels of theorems, which often help in developing interesting mathematical insights. Unfortunately, the enormous complexity of the task has so far prevented model generators from becoming ready for wide adoption.

The main purpose of this work is to explore and characterize the two main paradigms of model generation for the purposes of Natural Language Processing (NLP), i.e. analytical tableaux and constraint solving. The focus lies on the severe challenges that every approach to model generation must face, and workaround techniques from both paradigms are presented. Less emphasis will be put on the technical details of existing model generation systems.

The basic notions of the field are introduced in Chapter 2, which also establishes the notational background for the considerations of this paper. Chapter 3 introduces the tableau-based approach to model generation and explains its merits for model generation in NLP. More recent approaches within the framework of constraint programming are the subject of Chapter 4, followed by some discussion of the paradigm's promises and shortcomings.

The last three chapters constitute the core of this work, where the two main challenges of model generation are discussed in some detail. Chapter 5 explores some approaches to mitigating the omnipresent problem of combinatorical explosion, and how it interacts with the two frameworks. Chapter 6 then elaborates on specialized heuristics for model generation in NLP and issues of implementing them within both frameworks. Chapter 7 reviews past attempts to combine the best of both approaches and presents some new ideas in that area with respect to NLP.

Although formal semantics tends to rely on variants of higher-order logic as technical machinery, the developments in computational semantics have concentrated on first-order logic, mainly because it is taken to be the most powerful logic for which inference problems can still be considered computationally tractable in many instances. I will not deviate from this tradition, confining the discussion here to first-order model generation techniques. However, some of the ideas and techniques I will present will also be useful for model generation with other logics.

To understand the main points of this work, some background in computational semantics is required. For a thorough yet accessible introduction to the field, the reader is referred to Blackburn & Bos (2005). In addition, knowledge of at least the basic concepts of automated reasoning is necessary to understand all the technical details.

2 Model Generation

2.1 First-Order Structures and Satisfiability

We start by repeating some basic terminology. The notation that I will use throughout this paper is largely that of Konrad (2004).
Given a first-order language, an **interpretation** is a pair $\mathcal{I} = \langle [\![]\!], \mathcal{D} \rangle$ consisting of an interpretation function $[\![]\!]$ that maps predicate and constant symbols to predicates and constants of the appropriate types, and a domain $\mathcal{D}$ of individuals that provides entities for the interpretation of constants.
A **model** of a logical theory T is an interpretation of the symbols in T such that T becomes true. A **Herbrand model** is a model in which all constants and other terms are interpreted as themselves. In a classical first-order theory, Herbrand models are specified completely by the set of all ground literals that are satisfied in it. This means we can identify each Herbrand model with its representation as a set of ground literals.
We will usually only consider Herbrand models, and define a Herbrand model $\mathcal{M}$ by its set of positive literals $Pos(\mathcal{M})$ alone. All other literals will simply be interpreted as false by the interpretation function of $\mathcal{M}$.
A **finite model** is a model with a finite domain. The model generation methods that will be discussed here are limited to finding such finite models. It is possible to weaken this limitation to a certain degree, but the resulting methods are very complicated and beyond the scope of this work.
A logical theory T is **satisfiable** if it has a model $\mathcal{M}$. T is said to possess the property of **finite satisfiability** if it has a finite model.

2.2 Model Minimality

One of the central concepts in the theory of model generation is the notion of minimality. Many first order theories possess infinitely many models, which makes it worthwile to structure the space of possible models in a way that highlights the kind of models that is usually most useful. For this purpose, different notions of model minimality have been introduced. Here I only repeat the definitions that will become relevant, and refer to Konrad (2004), section 2.4, for an overview of the contexts in which they evolved.
A model $\mathcal{M}$ for a theory T is called **subset-minimal** iff for all models $\mathcal{M}'$ of T we have $Pos(\mathcal{M}') \subseteq Pos(\mathcal{M}) \Rightarrow Pos(\mathcal{M}') = Pos(\mathcal{M})$. Subset minimality is connected to circumscriptive reasoning, whose logical consequence relation can be expressed as evaluation in all subset-minimal models.
A model $\mathcal{M}$ for a theory T is called **domain-minimal** iff there is no model of T with a smaller domain of individuals. The advantage of domain minimality is that its verification is computationally a lot more tractable.
The third notion of model minimality is that of **predicate-specific minimality**. This is similar to subset minimality, but it can be defined for a specific predicate symbol P. Only the number of positive literals headed by the predicate symbol P is then taken into account when determining subset minimality. This type of minimality is often used in artificial intelligence tasks such as system diagnosis, where it mirrors the assumption that e.g. the number of components that act abnormally can be assumed to be the minimum that still explains the erroneous system behavior.

2.3 Models in Natural Language Understanding

The relatively young field of computational semantics has mainly confined itself to the art of automatically computing adequate logical representations of the meanings of natural language sentences in the tradition of formal semantics. The question what can be done with these representations has not yet recieved much attention. This especially applies to model generation, partially because the field is only beginning to produce useful tools, and partially because these tools have some properties that limit their usefulness.
A good overview of the current problems and the promises of model generation for natural language understanding can be gained from Bos (2003).
The first important application area for model generators is that they can can be used in parallel with theorem provers to decide many first-order theories, which makes them a useful tool e.g. in consistency and informativity checking. In addition, the model representations that result from the process are interesting in their own right, as they turn out to contain much of the information required for natural language understanding.
A model of the logical representation of a discourse is often a relevant model of the situation described by the discourse. These models are expected to capture many relevant aspects of the listener's mental representation of the described situation. As a first application, such models have been shown to be useful for answering questions via model checking, as in Blackburn & Bos (2005).

2.4 Catering for Special Needs

The challenges in applying existing model generation technology to computational semantics are manifold, and I will confine myself here to the most pressing issues that have prevented model generation from gaining much popularity. These issues need to be kept in mind when thinking about possible ways of building model generators that are more useful for NLP.
The major technical hurdle is that it would be desirable to be able to generate **models for larger discourses**. Off-the-shelf model generators cannot generate models with domain sizes larger than about 20, which effectively limits the kind of situations that can be modeled to mere toy examples. Model generation technology as it stands therefore cannot be used to generate models of interesting discourses, which severely limits is usefulness.
Domain-minimal model generation as implemented by every model generator on the market seems very useful for semantics, as we would normally want to ensure that no redundant structure is present in the models of a discourse. However, a domain-minimal model generator tries to introduce as few entities as logically possible. It will therefore always assume entities to be identical as long as this causes no contradiction, which tends to lead to undesired results.
Often, we already have a lot of knowledge about a situation that can easily be expressed by a model, and which we only want to extend by the information contained in some natural language description. This kind of **incremental model generation** is unfortunately not possible using the existing tools. This means that to enlarge a model by new information, we have to describe the model via a restrictive first-order theory, enlarge that theory by the new information, and apply the model building process again. This is of course very inefficient and inevitably transcends the capacities of current model generators very quickly.

3 Tableau-Based Approaches

Tableaux calculi are one of the most influential and successful approaches to automated reasoning. For a thorough introduction that also includes a lot of background information, the reader is referred to Hähnle (2001). Some basic concepts of automated reasoning, such as skolemization and the ideas behind a tableau calculus for refutation, must be presupposed here for reasons of brevity.

3.1 Analytical Tableaux

In the notation that I will use here, the following rules define the propositional fragment of a classical tableau calculus:

$$\frac{\neg\neg\phi}{\phi} \qquad \frac{\phi \wedge \psi}{\begin{array}{l}\phi\\ \psi\end{array}} \qquad \frac{\neg(\phi \vee \psi)}{\begin{array}{l}\neg\phi\\ \neg\psi\end{array}} \qquad \frac{\neg(\phi \rightarrow \psi)}{\begin{array}{l}\phi\\ \neg\psi\end{array}}$$

$$\frac{\phi \vee \psi}{\phi \qquad \psi} \qquad \frac{\neg(\phi \wedge \psi)}{\neg\phi \qquad \neg\psi} \qquad \frac{\phi \rightarrow \psi}{\neg\phi \qquad \psi}$$

The first group of rules is for obvious reasons also called **conjunctive** or non-branching, while members of the second group of rules are called branching or **disjunctive** rules. Non-branching rules are commonly referred to as α**-rules**, while the branching rules are called β**-rules**.
Rules for the treatment of universal and existential quantification in tableaux are traditionally called γ**-rules** and δ**-rules**, respectively. The γ-rule of **analytical tableaux** is based on introducing **skolem constants**, whereas the γ-rule non-deterministically substitutes ground terms:

$$(\forall)\frac{\forall x.\gamma(x)}{\gamma(t)},\ \textit{where } t \textit{ is an arbitrary ground term}$$

$$(\exists)\frac{\exists x.\delta(x)}{\delta(c)},\ \textit{where } c \textit{ is new constant symbol}$$

The advantage of analytical tableaux is that we have skolem functions of zero arity because the quantification rule eliminates the dependencies on quantifiers with higher scope and their bound variables.
Unfortunately, analytical tableaux have serious efficiency issues mainly caused by the very non-deterministic γ-rule. At the point of application, not much information is available that could lead to a useful guess for the instantiation. This problem was mitigated a lot by the development of **free variable tableaux**, which rely on skolem functions and **unification** to guide instantiations in ways that close as many tableaux branches as possible.

3.2 Tableau-Based Model Generation

A **saturated branch** of an analytical tableau is a branch that cannot be extended any more and failed to be closed during tableau expansion. The fact that the literals on such branches can directly be interpreted as descriptions of models is the main observation made by Boolos (1984) in his pioneering work. The domain of such a model corresponds to the skolem constants we introduced

while extending the branch. This means we are not any longer interested in closed branches that would indicate contradictions, but in finding saturated branches, each of which represents a model.
Many of the optimization techniques that were developed for tableaux in order to speed up refutations are not helpful in model generation. This largely also applies to the techniques that helped to make automated theorem proving feasible, e.g. to skolemization and unification. That these methods cannot be used to speed up model generation mirrors the fact that first-order satisfiability is inherently a harder problem than first-order validity, being not even recursively enumerable, as e.g. shown in Hedman (2004, p. 412ff).
But the main reason against using full free-variable tableaux in model building is that whereas skolemization with skolem functions maintains refutational completeness, it unfortunately breaks **completeness for finite satisfiability**. An instance of this is the following example by Konrad (2004): The formula $P(a,a) \wedge \forall x(P(x,x) \rightarrow \exists y(P(x,y)))$ obviously has a finite Herbrand model. But skolemization turns this into $P(a,a) \wedge \forall x(P(x,x) \rightarrow P(x,f(x)))$, which only has infinite Herbrand models.
For ease of exposition, I introduced tableaux for arbitrary first-order formulae here. In order to shorten lenghty completeness proofs and to get more efficient implementations, state-of-the-art tableaux calculi usually convert input formulae into some kind of clausal form and then only operate on (ground) clauses.
A **hyper tableau** is a ground clause tableau where extension steps for a branch must be weakly connected to the current branch. The first calculi known under the name of model generation (Manthey & Bry (1988), Fujita & Hasegawa (1991)) were essentially variants of hyper tableau calculi. Clauses in a hyper tableau calculus have a rule format, and literals are selected via a fair selection function. A typical selection function for many hyper tableau calculi selects exactly the negative literals in a clause. This variant runs under the name of **positive hyper tableaux** and was first presented in Bry & Yahya (2000).
EP tableaux as defined by Bry & Torge (1998) are a variant of positive hyper tableaux for non-clausal first-order logic without function symbols that is complete for finite satisfiability. Their calculus operates on PRQ formulas (Positive with Restricted Quantifications), which removes the need for free variables. A modified δ-rule that enumerates the current model domain provides the branches with enough information for explicit model construction. Using a fair computation rule, it is possible to ensure that the method detects all finite term domain models.
The **OletinMB** system as presented in Dellert (2010) is in essence an implementation of the EP calculus that works on arbitrary input formula at the expense of some performance. It is an experimental system with a very flexible interface for defining various heuristics, yet it also includes optimizations that allow it to compete with the performance of other model generation systems on many theories over function-free signatures.

3.3 Why Tableaux Are Useful

The main reason for the popularity of tableaux approaches is that they are one of the most human-readable ways of representing automated inferences. The clear data structure and explicit representation make the mechanisms involved

in finding proofs or models a lot more transparent than within other calculi, though often at the cost of a somewhat redundant notation.
In the context of model building, tableaux methods are especially popular because each branch can be interpreted as a **partial model**, and tableaux expansion thus corresponds to stepwise model expansion. This means we can observe many properties of models already while they are being generated, which is conducive to the development and implementation of advanced heuristics.
Analytical tableaux additionally have rather advantageous properties in the context of incremental model building. Unlike in free-variable tableaux, branches can be modified and extended independently of the other branches, which means that it is not necessary to keep the entire tableau in memory. Instead, a simple agenda of open branches is sufficient to represent all the relevant information contained in an analytical tableau at any point.
This means that the relation between open tableau branches and partial models also holds in the other direction. Partial models can be interpreted as open tableau branches which can be inserted into the agenda in order to be subsequently enriched with the information from an extended theory, emulating the tableau construction process that would have resulted if the added formulae had been part of the theory all along.

4 Constraint-Based Model Generation

Constraint programming as a research area is concerned with efficiently solving computable sets of relations between finite sets of variables. Constraints offer a very natural way of expressing combinatorial problems in many different areas, and the development of efficient constraint solvers has become one of the major trends in symbolic computing. For an early, but still excellent introduction to the field the reader is referred to Marriott & Stuckey (1998).

4.1 First-Order Satisfiability as a Constraint Problem

One of the most popular types of constraint programming is **finite domain modeling**, where variables range over finite domains. A satisfiability problem for a propositional formula can very straightforwardly be translated into an instance of **Integer Programming**, where variables range over finite sets of integers. This is achieved by introducing for every subformula a variable ranging over integer values that represent the possible truth values, and to impose constraints on these variables that represent the semantics of propositional logic. For example, the following set of conditions is necessary and sufficient for any model $\mathcal{M}$ of the propositional formula $a \vee (b \wedge (c \vee d))$:

$$\begin{array}{lll}
[\![a]\!]_{\mathcal{M}} \geq 0, & [\![a]\!]_{\mathcal{M}} \leq 1, & [\![a \vee (b \wedge (c \vee d))]\!]_{\mathcal{M}} = 1, \\
[\![b]\!]_{\mathcal{M}} \geq 0, & [\![b]\!]_{\mathcal{M}} \leq 1, & [\![a \vee (b \wedge (c \vee d))]\!]_{\mathcal{M}} = \min([\![a]\!]_{\mathcal{M}}, [\![b \wedge (c \vee d)]\!]_{\mathcal{M}}), \\
[\![c]\!]_{\mathcal{M}} \geq 0, & [\![c]\!]_{\mathcal{M}} \leq 1, & [\![b \wedge (c \vee d)]\!]_{\mathcal{M}} = \max([\![b]\!]_{\mathcal{M}}, [\![(c \vee d)]\!]_{\mathcal{M}}), \\
[\![d]\!]_{\mathcal{M}} \geq 0, & [\![d]\!]_{\mathcal{M}} \leq 1, & [\![c \vee d]\!]_{\mathcal{M}} = \min([\![c]\!]_{\mathcal{M}}, [\![d]\!]_{\mathcal{M}})
\end{array}$$

For a given domain size, this approach can be generalized to first-order satisfiability reasoning by means of flattening. As in Konrad (2004, section 3.4.4), translation rules can be notated by means of **signed formulae** $V_\phi : \phi$ where ϕ is a formula and V_ϕ is a variable associated with the interpretation $[\![\phi]\!]$. For

a domain of given size n we introduce the constant symbols $c_1, \ldots, c_n$. The more general scheme presented by Konrad then boils down to the following set of rules for translating the satisfiability problem for a first-order formula ϕ into an instance of Integer Programming, starting with $V_\phi : \phi$ and $V_\phi = 1$:

$$\frac{V_{\phi_1 \wedge \phi_2} : \phi_1 \wedge \phi_2}{\begin{array}{c} V_{\phi_1} : \phi_1 \\ V_{\phi_2} : \phi_2 \\ V_{\phi_1 \wedge \phi_2} : \min(V_{\phi_1}, V_{\phi_2}) \end{array}} \qquad \frac{V_{\phi_1 \vee \phi_2} : \phi_1 \vee \phi_2}{\begin{array}{c} V_{\phi_1} : \phi_1 \\ V_{\phi_2} : \phi_2 \\ V_{\phi_1 \vee \phi_2} : \max(V_{\phi_1}, V_{\phi_2}) \end{array}}$$

$$\frac{V_{\phi_1 \to \phi_2} : \phi_1 \to \phi_2}{\begin{array}{c} V_{\neg\phi_1} : \neg\phi_1 \\ V_{\phi_2} : \phi_2 \\ V_{\phi_1 \to \phi_2} : \max(V_{\neg\phi_1}, V_{\phi_2}) \end{array}} \qquad \frac{V_{\neg\psi} : \neg\psi}{\begin{array}{c} V_{\psi} : \psi \\ V_{\neg\psi} : 1 - V_{\psi} \end{array}} \qquad \frac{\begin{array}{c} V_1 : a \\ V_2 : a \end{array}}{V_1 = V_2}$$

$$\frac{V_{\forall x.\phi(x)} : \forall x.\phi(x)}{\begin{array}{c} V_{\phi(c_1)} : \phi(c_1) \\ \vdots \\ V_{\phi(c_n)} : \phi(c_n) \\ V_{\forall x.\phi(x)} : \max(V_{\phi(c_1)}, \ldots, V_{\phi(c_n)}) \end{array}} \qquad \frac{V_{\exists x.\phi(x)} : \exists x.\phi(x)}{\begin{array}{c} V_{\phi(c_1)} : \phi(c_1) \\ \vdots \\ V_{\phi(c_n)} : \phi(c_n) \\ V_{\exists x.\phi(x)} : \min(V_{\phi(c_1)}, \ldots, V_{\phi(c_n)}) \end{array}}$$

Each solution for the resulting instance of Integer Programming describes a model of ϕ by the values assigned to the variables V_a where a is atomic. Proofs of refutation soundness and of completeness for finite satisfiability can be found in Section 3.4.8 of Konrad (2004).

4.2 Overview of Constraint Technology

With a translation of the model generation problem into some constraint language at hand, the next step is to find an efficient constraint solver for this kind of problem. This section gives an overview of current constraint technology, and contains some discussion of the technological alternatives.

While implementations of pure constraint programming languages exist, many successful constraint solvers are distributed as modules or packages for one or more of the most commonly used programming languages.

Konrad (2004), for instance, used the built-in finite-domain integer package of the functional programming language **Oz** [1] for building his KIMBA model generator. KIMBA is a very adaptable framework for model generation in the spirit of **lean automated reasoning**, which means that it is tailored towards simplicity and adaptability for research instead towards industry-strength performance. Unfortunately, the system has ceased to be available, the author has lost access to the source code (personal communication) and states that it only runs on extremely outdated versions of Oz that are difficult to come by and get to run on current systems.

Turning to state-of-the-art constraint technology instead, we see the market split between high-performance industry-oriented commercial systems and more research-oriented free software libraries. Examples of the latter include the very

[1] `http://mozart-oz.org/`

popular C++-based constraint libary **Gecode** [2] as well as the Java library **JaCoP** [3], which I have been using for my experiments.
Major players in the commercial market include IBM with their very successful ILOG **CPLEX** Optimizer [4], and Gurobi with the **Gurobi** Optimizer[5], which is particularly good at exploiting multi-core processors. An emerging player is the NICTA Constraint Programming Platform project [6] with their system **G12**, which strives to combine existing tools with approaches from automated reasoning to yield efficient solution methods for many hybrid problems.
The tendency for each system to define its own constraint language led to a lack of interoperability between different constraint systems, making comparison and integration of different solvers costly and difficult. To achieve some level of standardization, de la Banda et al. (2006) introduced **Zinc** as a powerful modelling language that subsumes most of the other constraint languages in expressivity, paving the way for a more unified syntax and influencing many systems in their input languages. However, Zinc turned out to be too expressive to be widely adapted and implemented.
This led Nethercote et al. (2007) to the definition of **MiniZinc** as a subset of Zinc that can be compiled into the low-level constraint language **FlatZinc**. MiniZinc has enjoyed growing popularity in recent years, and many systems today offer FlatZinc as an alternative input language in addition to their native constraint languages. As these include popular non-commercial systems such as Gecode and JaCoP, MiniZinc is well on the way to becoming the first lingua franca of constraint programming.

4.3 Advantages of the Constraint Paradigm

The main asset of constraint programming is that it is a **declarative** paradigm, which means that from the user perspective, the focus is on formalizing problems, without a need to put much effort into algorithms for their solution. This also implies that only the problem statement needs to be changed when circumstances change or different variants need to be tried out. Constraint solvers are therefore ideal for rapid prototyping and the ad-hoc solution of practical combinatorical problems as they occur in logistics and other planning tasks.
The promise of using constraint solvers for model generation is that constraint solvers are readily available, highly optimized and mature systems that are usually a lot more performant than what one could hope to achieve with a home-made program for any given combinatorical problem. This especially includes ad-hoc implementations of tableaux calculi, where the effort that has to be put into optimization in order to end up with a workable system heavily detracts from the time that could go into a better understanding of the problem domain and into the development of applications.

[2] `http://www.gecode.org/`
[3] `http://jacop.osolpro.com/`
[4] `http://www.ibm.com/software/integration/optimization/cplex-optimizer/`
[5] `http://www.gurobi.com/`
[6] `http://www.nicta.com.au/research/projects/constraint_programming_platform/`

4.4 Specialization vs. Generality

While constraint solvers are extremely useful as rather general tools that are still reasonably fast at solving combinatorical problems, their generality comes with **abstraction costs**, just as in many other areas of computer science such as database and virtualization technology.
As a paradigm, constraint programming is a little too general to be implemented with near-optimal efficiency for all problem instances by a single solver. The non-commercial libraries suffer from the unavoidable problem that the more freely one can express problems, the higher the burden of implementing the constraint language becomes, and the more inefficient the implementations will get due to limited development time.
On the other hand, commercial systems are tailored towards easy deployment by non-specialist users in a commercial environment. Together with the strict closed-source policy, this turns commercial constraint solvers into **black boxes**, making it hard to integrate additional information that cannot be expressed in the modelling language, but could guide the search for solutions.
As we have seen, the translation of the model generation problem into a constraint problem is only possible for a fixed domain size. This makes it necessary to **estimate the domain size** beforehand, which will not be very precise for larger theories. Therefore, the advantage in performance offered by an industry-strength constraint solver could easily be wasted by many iterations over different domain sizes. This issue also reflects the fact that model generation is not a purely combinatorical problem, which makes it quite a bit more complex than what one would normally try to tackle using constraint technology. A little caution is therefore advised with the assumption that the matureness of constraint solving technology clearly makes constraint-based model generation the most promising method.

5 The Combinatorical Challenge

Even though first-order satisfiability is undecidable in general, this does not need to concern us if we are only interested in finite models. We have seen that it is possible to implement model generation using a tableaux calculus that is complete for finite satisfiability.
Although many models of infinite size are finitely representable and can therefore still be handled by computers, restricting ourselves to finite models arguably suffices for most linguistic applications. Even though even a simple world knowledge database that contains an axiom such as *every person has a father* already causes every sentence that mentions a person to only have infinite models, from a cognitive perspective it makes sense to assume that the mental representation of a sentence is never an infinite structure.
The real problem of model generation for NLP is therefore not its undecidability, but its intractability for many decidable cases, especially for larger problems. Theorem proving algorithms are notorious for scaling rather badly, but in model generation the challenge is even higher. In this chapter, we will get an impression of how severe the problems are, and we will learn about some basic techniques that promise to help in partially circumventing them.

$\exists x_1 \exists x_2 \exists x_3 \exists x_4 (org1singapore(x_1) \wedge r1nn(x_1, x_2) \wedge n1scientist(x_2) \wedge v1reveal(x_3) \wedge r1agent(x_3, x_2) \wedge r1theme(x_3, x_4) \wedge n1proposition(x_4) \wedge \exists x_5 \exists x_6 \exists x_7 \exists x_8 (n1sar(x_6) \wedge r1nn(x_6, x_5) \wedge n1virus(x_5) \wedge a1genetic(x_7) \wedge n1change(x_7) \wedge v1undergo(x_8) \wedge r1agent(x_8, x_5) \wedge r1patient(x_8, x_7) \wedge n1event(x_8)) \wedge n1event(x_3))$
$\forall x_1 (n1abstract\_entity(x_1) \rightarrow n1entity(x_1))$
$\forall x_2 (n1change(x_2) \rightarrow n1event(x_2))$
$\forall x_3 (n1event(x_3) \rightarrow n1abstract\_entity(x_3))$
$\forall x_4 (n1proposition(x_4) \rightarrow n1abstract\_entity(x_4))$
$\forall x_5 (v1reveal(x_5) \rightarrow n1event(x_5))$
$\forall x_6 (n1scientist(x_6) \rightarrow n2being(x_6))$
$\forall x_7 (n2being(x_7) \rightarrow n1object(x_7))$
$\forall x_8 (n1object(x_8) \rightarrow n1entity(x_8))$
$\forall x_9 (org1singapore(x_9) \rightarrow n1object(x_9))$
$\forall x_{10} (v1undergo(x_{10}) \rightarrow n1event(x_{10}))$
$\forall x_{11} (n1virus(x_{11}) \rightarrow n2being(x_{11}))$
$\forall x_{12} (n1abstract\_entity(x_{12}) \rightarrow \neg n1object(x_{12}))$
$\forall x_{13} (n1sar(x_{13}) \rightarrow \neg n1entity(x_{13}))$
$\forall x_{14} (n1change(x_{14}) \rightarrow \neg v1reveal(x_{14}))$
$\forall x_{15} (n1event(x_{15}) \rightarrow \neg n1proposition(x_{15}))$
$\forall x_{16} (org1singapore(x_{16}) \rightarrow \neg n2being(x_{16}))$
$\forall x_{17} (v1undergo(x_{17}) \rightarrow \neg n1change(x_{17}))$
$\forall x_{18} (v1undergo(x_{18}) \rightarrow \neg v1reveal(x_{18}))$
$\forall x_{19} (n1virus(x_{19}) \rightarrow \neg n1scientist(x_{19}))$

Figure 1: Typical first-order theory as produced by Nutcracker

5.1 The Scale of the Problem

In order to understand why combinatorical explosion constitutes such an enormous obstacle to model generation for larger theories, it is worthwile to consider some figures as they arise in a typical example. Take the theory in Figure 1, which the Nutcracker RTE system by Bos & Markert (2005) produces to represent the meaning of the newspaper headline *Singapore scientists reveal that SARS virus has undergone genetic changes*, and which I will use as a typical example of a theory as they arise in computational semantics.

This theory contains thirteen unary and four binary predicate symbols. If we simply compute the number of combinatorically possible structures over these symbols for a given domain size n, we arrive at a structure space size of 2^{13n+4n^2}, which for the plausible domain size of $n = 8$ already evaluates to 2^{360} structures. An efficient model building algorithm will of course not even come close to generating all of these structures before finding a model. However, in the case where we try to generate models of an unsatisfiable theory, quite a few of these alternatives will indeed have to be tried out.

For a given finite domain size, the satisfiability problem for first order logic can be flattened to a propositional satisfiability problem by replacing universal (and existential) quantifiers with finite conjunctions (and disjunctions) over the whole domain. An algorithm that even in the worst case only needs to test a non-exponential fragment of the search space would therefore have to solve the SAT problem in polynomial time, which would make it a proof of P = NP and

therefore unlikely to be found or even exist.
Given these theoretical limits, we must expect the size of the search space to become completely intractable already at rather low values of n, which means we cannot hope to ever develop a model generator that can reliably generate larger models for arbitrary first-order theories.

5.2 Model Building vs. Model Finding

A first step towards dealing with these limitations is to analyze the nature of the theories arising from the semantic analysis of larger discourses. The basic intuition is that depending on the task we wish to accomplish the model generation problem will surface as one of two typical variants.
In the first variant, the structure space contains many models of the theory. In the tableau case, this means we have many saturated branches, and in the constraint case it means that the constraints are not very restrictive. This means that model generation can proceed very constructively. I therefore propose to use the term **model building** only for model generation in such a situation, and will do so during the rest of the paper.
In the second situation we deal with a restrictive theory that does not possess many models. Almost all branches of a tableau will eventually be closed, and the equivalent constraint problem will be a lot harder to solve. In this situation, model generation will largely become the task of finding one of the few points in structure space that constitute a model, and I will therefore call it the problem of **model finding**.
Although formally, there is no difference between model building and model finding, the optimizations that are most promising in the two situations are obviously of a very different nature. There is of course a continuum of mildly restrictive theories between the two extremes, but the formal representations that we want to build models for in computational semantics do indeed mostly come in one of the two variants. This may be because the thoughts humans utter under normal circumstances are not obviously inconsistent or hard to accomodate. On the other hand, very restrictive theories need to be processed when we need countermodels as e.g. in entailment checking.
The goal of optimization in model building is to speed up the traversal of the structure space, building large parts of the structures as soon as possible and not delaying too many decisions. This is a useful strategy because the probability of ending up in a closed tableau branch will be quite low. Heuristics that are optimized for model building will therefore tend to traverse the structure space mainly in a depth-first fashion.
On the other hand, efficient model finding will tend to employ more of a breadth-first strategy. Model finding is of course inherently more difficult than model building, which means that a model finder will always have to test many different variants and backtrack a lot.
The usefulness of distinguishing these two variants of the model generation problem is that we can develop tools for the two cases separately. Although the combinatorical challenge is present in both situations, we already know that it is a lot less pronounced in model building than in model finding. By focusing on the instances where model building is sufficient, we have a much better chance of arriving at technology that can process the semantics of entire discourses.

5.3 Why Tableaux Are Slower

A common impression among users of automated reasoning technology is that tableau implementations are generally rather slow. This impression is certainly justified if one compares the performance of mostly academic tableau-based reasoning tools with the excellent performance of commercial constraint solvers. The reason behind this problem is not that tableau methods would be inferior in principle, but that a lot of specialized knowledge and excellent programming skills are necessary to implement good systems. The lack of readily available and versatile tableau technology has led many researchers in the field to develop their own ad-hoc implementations, often quite reasonably not with the intention of arriving at competitive systems, but as mere proofs of new concepts.
Of course, even the most sophisticated constraint solving techniques cannot do away with a problem's inherent combinatorical complexity. This gives us a reason to assume that tableau systems could achieve success similar to that of constraint systems if the industry would put a comparable amount of ingenuity into their implementation and optimization.
For our purposes, the main problem of the existing techniques for tableau optimization is that they mostly rely on clause normal forms. But these require the use of implicit quantification and therefore of skolem functions, which makes them problematic for model generation.
On the way towards feasible tableau-based model generation systems, we therefore have to look into possible improvements of analytical tableaux, where the research efforts are a lot more scarce than for optimizations of free-variable tableaux for theorem proving. However, there exist some interesting ideas that seem to indicate that the possibilities of analytical tableau optimization are far from exhausted.
Baumgartner & Kühn (2000) developed a **regularity condition** that can be used to prevent logically redundant expansions on hyper tableaux, an idea that can easily be adapted to the more general analytical tableaux case. Another interesting optimization technique is the **subproof strategy** advocated by Kohlhase & Koller (2001), which can be used to find shortcuts that lead to the immediate generation of additional structure in the current model.
None of these techniques is commonly implemented in off-the-shelf tableau systems, and it is unlikely that the stance of tableau technology in the industry will improve, mostly because of the higher accessibility of the constraint paradigm to non-expert users. Under these circumstances, constraint technology has good chances to remain the primary paradigm for solving combinatorical problems.

5.4 Issues with Constraints

The generality of constraint solvers means that out of the box they cannot be extremely efficient in solving every problem that the user can express using the respective input language. Optimization can be seen as the implementation of some **procedural knowledg**e that cannot be expressed in the declarative input language, which means that it requires some control of the heuristics. Given the combinatorical complexity of the model generation problem, we can be sure that no constraint solver will achieve acceptable performance without some modifications to the default heuristics.
Apart from the expert knowledge that is necessary to understand the heuristics

of a modern constraint solver, enforcing a specific desired behavior is often made difficult because it would interact badly with some low-level implementation detail. We therefore cannot expect to get as much control of the procedural side as would be possible e.g. in logic programming.
The problem is aggravated by the fact that different constraint solvers support different ways of influencing their heuristics. Unlike MiniZinc for problem statements, there is no standardized and implementation-independent way to define specialized heuristics. The need for optimization therefore destroys interoperability as one of the major advantages of current constraint technology. To arrive at an efficient system, one needs to buy into one constraint solver, which can considerably confine the room for experimentation.

6 Special Heuristics for NLP

This chapter focuses on some ideas how more adequate model generation for computational semantics can be achieved. Since all of these ideas boil down to defining specialized heuristics, the design and implementation of specialized heuristics for the purpose will be the main focus of investigation.
After a few short observations about the structure of typical search spaces in NLP, we will first have a closer look at ways to formalize and implement heuristics in analytical tableaux. The discussion will mainly rely on the notion of a **tableaux machine** as presented in Kohlhase & Koller (2000), which permits to describe and compare many useful heuristics. I will then offer a view of my OletinMB system as a version of a tableaux machine, and build up on this idea to elaborate a little more on **Prioritized Identity Assumptions (PRIDAS functions)** as introduced in Dellert (2010). Both approaches will be motivated by possible application areas in computational semantics.
I will then provide an overview of the essential constraint solving heuristics as they are applied in automated reasoning, and also give an impression of the ways in which heuristics can usually be influenced, with JaCoP as the primary example. The chapter concludes with a summary of promising specialized heuristics and an assessment how well these ideas could be implemented with the existing tableau and constraint technology.

6.1 A Closer Look at Search Spaces

If we inspect the search space of an analytical tableau calculus on a typical input theory such as the one from Figure 1, some interesting patterns emerge. Especially interesting is the way in which the information contained in world knowledge axioms of the form $\forall x(\phi(x) \rightarrow \psi(x))$ gets integrated into the partial models. In the calculus as defined in Chapter 3, such an axiom is expanded to $\phi(c_i) \rightarrow \psi(c_i)$ for all skolem constants c_i representing the model's domain. Each of these implications leads to a split of each tableaux branch, leading to a structure that fans out considerably. One such axiom over a domain of size n is factored out to 2^n distinct branches, many of which do not differ very much in relevant information. The need to treat all these branches independently is one of the major reasons for combinatorical explosion.
This effect can be somewhat mitigated by a **forward chaining** rule that applies universal quantification only if the antecedent is already on the branch. Using

this rule one can avoid some of the branching, though at the cost of missing some negative literals that would have been inferred in a true case distinction, potentially leading to delayed closure of some branches. Experiments show that the negative consequences of the forward chaining rule mostly concern model finding, and that it tends to speed up model building tremendously.
Another problematic pattern results from the so-called **universe explosion problem**. A naive analytical tableau has considerable difficulty in handling formulae with the $\forall\exists$-prefix. The problem is that for each entity in the domain, the existential quantifier warrants the introduction of a new entity, to which the formula immediately needs to be applied again, a process which leads to an infinite branch. One of the reasons for the popularity of minimal model generation is that it systematically the problem by avoiding to run into such branches, prioritizing other instantiation choices.

6.2 Tableaux Machines

Kohlhase & Koller (2000) informally introduce a tableaux machine as an automaton whose states are tableaux, and whose accessibility relation is defined by the possible expansions according to some tableau calculus. With closed tableaux as final states, a run of the automaton will correspond to a refutational proof. If every tableau with a saturated branch is a final state, the automaton turns into a model generation machine. The branch and the formula that are selected for expansion at each step can be determined via any computable function.
This basic model of a tableaux machine can then be enhanced by special operations. One idea is to support the introduction of **new formulae** at any state of the machine, which are then added to all branches of the current tableau. Backtracking is seen as an **error recovery** operation that allows the machine to revert to some earlier state and then move into another branch. In addition, the formalism allows to define **theorem-proving sub-tableaux** which need to be closed before the machine can move to another state of the main tableau. This is useful for checking whether some formula is entailed by a branch.
A tableaux machine can be used as a framework for describing a variety of advanced heuristics. For instance, Kohlhase & Koller (2001) suggest a **bounded optimization** technique for determining the expansion strategy. Using costs for rule applications and a function measuring the gain in model quality, the strategy would consist in expanding at low cost and high gain until the costs of further expansion are higher than the expected gain.
The original context for the notion of a tableaux machine was the development of **Resource-Adaptive Model Generation (RAMG)**, a calculus that was introduced by Kohlhase & Koller (2001) as a performance model for natural language understanding. The main idea of RAMG is the guidance of a resource-limited tableau expansion strategy by keeping track of **saliences** for constant symbols. Salience is first approximated using syntactic criteria (e.g. with subjects being more salient than objects) and decays as new sentences are added to the discourse. The re-use of a constant increases its salience. If we now have salience influence expansion rule costs by e.g. making the use of a non-salient entity in a δ-rule very costly, we thereby model the preference for salient entities during anaphora resolution. Kohlhase & Koller (2001) explain by examples how RAMG with saliences could also be used to capture other lin-

guistic phenomena such as accomodation, bridging inferences and the behavior of definite reference.
As Kohlhase & Koller (2001) themselves remark, the claims of their paper are largely based on introspection rather than on evidence. The examples are only shown to work for ad-hoc rule costs, salience assignments and world knowledge axioms. From the perspective of computational semantics, it is a highly relevant question where such numbers and axioms would come from if one were to go beyond the study of a few well-chosen examples and to apply it to a wider range of linguistic data.
Burchardt & Walter (2001) describe a first implementation of RAMG, the **BuGS** system, which they intend as an experimental environment for answering such questions. They manage to ensure a consistent choice of rule costs and saliences by employing a tableau-congruent salience structure as a technical device for handling salience functions. To demonstrate that it is possible to drastically cut the search spaces while still emulating human performance in model generation, the BuGS system forfeits backtracking and always commits to one branch that is locally optimal. This makes BuGS an extreme case of a model builder, one that runs a high risk of not arriving at a finite model although it might exist. To make the consequences of a single wrong expansion decision less drastic than not to arrive at any model at all, **shifting** is introduced as a local repair mechanism that permits the revision of a few very recent expansion decisions.
The main strength of a tableaux machine is that branch and rule selection strategies can be defined in a very modular manner. The variant implemented in BuGS relies on a freely definable **heuristic function** that is used to estimate the expected gain of expansion alternatives at each decision point. Burchardt & Walter (2001) suggest the average salience of the partial model resulting from each expansion step as a first plausible quality measure. Further ideas aim at discouraging model growth in the spirit of domain-minimal model generation, preference of chaining over full evaluation of universal quantification, and equivalence classes of constant symbols that are resolved later.
The flexibility mirrored in these ideas makes BuGS a very valuable platform for experimentation, and the authors propose to use this capability for showing that there is a single choice of strategy and parameters that causes BuGS to handle most of the examples from Kohlhase & Koller (2001) in the desired way. This step seems to never have been taken, and the question how parameters and strategies for wider coverage could be found remains open. Unfortunately, the BuGS system is not available for further experiments, and not even the distribution webpage [7] seems to ever have been launched.

6.3 OletinMB and PRIDAS functions

While minimal model building has its merits because in principle it allows us to avoid assuming too much structure, the resulting models will have a bias towards smaller domains that does not necessarily correspond to human reasoning. For example, the minimal model of $\exists x \exists y(man(x) \wedge car(y) \wedge drive(x, y))$ is a universe with a single entity that is both a man and a car, and which drives itself.

[7] `http://www.coli.uni-saarland.de/~stwa/bugs/`

Much of this problem can be attributed to missing world knowledge that precludes **unwanted identity assumptions** and can be brought into the picture by introducing a usually rather large number of additional axioms into the theory. This works quite well for small theories, but the huge blow-up in theory size makes models generation for larger discourses even less tractable.
To mention another problem, it is not clear what kind of world knowledge could prevent a minimal model generator from interpreting the sentence "I see a man" with the formal representation $\exists x(man(x) \wedge see(i, x))$ as being uttered by a man who sees himself. The information that a model generator would need to avoid this is that under normal circumstances, two-place predicates that represent the meanings of transitive verbs should preferably not be instantiated with two identical arguments.
The main idea implemented in the heuristics of OletinMB to address both problems is to use **prioritized identity assumptions (PRIDAS)** during the model generation process, and to control these via a **preference function** that can make use of external knowledge. In terms of a tableaux machine, a PRIDAS function is a mechanism to dynamically adapt the costs of different possible applications of the δ-rule.
By default, the input to a PRIDAS function are atomic formulae that directly result from different instantiation decisions. A PRIDAS function then uses this information to rank possible instantiations according to linguistic criteria. Such a function could e.g. rely on an ontology to preclude identity assumption between entities of incompatible types (such as men and cars), or dismiss instantiations that give a reflexive interpretation to transitive verbs.
The OletinMB system offers a flexible interface for user-definable PRIDAS functions, and is currently being extended to handle other types of global heuristics. Futhermore, in order to compensate for the lack of any available platform for experimentation with analytical tableaux, OletinMB will be extended into a full implementation of a tableaux machine, with great advantages for further research in specialized heuristics.

6.4 Heuristics in Constraint Programming

Constraint solver implementations usually rely on a two-part decision procedure operating on sets of constraints and a given set of variables. The **propagation** part is a process that uses the knowledge currently contained in the constraints to (further) restrict the values of the variables. These restrictions are then used in order to infer new constraints or to simplify old ones. The **distribution** part is responsible for provisionally restricting a variable more strongly than warranted by propagation alone in order to enable further propagation. Distribution is a non-deterministic process and the part that makes it necessary to use a backtracking mechanism.
An important principle of efficient constraint solving is to delay distribution as much as possible in order to avoid backtracking, leading to more educated guesses for the provisional restrictions that promise to be of value. Another important technique is to prioritize distribution over variables that occur in many constraints in order to maximize the effect of propagation.
Good constraint solvers offer interfaces for customizing distribution, which are however often not very powerful. As a practical example, we take JaCoP as

a non-commercial and therefore very open system. JaCoP is a rather well-documented and extensive library, and it offers a versatile interface as well as a variety of configuration options for influencing search. As global search modes, JaCoP offers depth-first search, restart search and credit search, as well as combinations of these. Most of the flexibility lies in the possibility to implement **custom search plugins** that can listen to search events such as encountered solutions, performed consistency checks and backtracking. The plugins can then influence the next distribution step by controlling variable selection and value assignments in various ways.
To adapt heuristics beyond these rather local decisions, it would be necessary to completely understand and make changes to the core implementation. Given that the source code for JaCoP is available, this is not impossible, but it would require one to become an expert in the internals of the system. Such an endeavor would inevitably lead to a fork of JaCoP development, precluding the possibility of profiting from improvements in future versions of the main branch.

6.5 Implementing Specialized Heuristics

The parallels between efficient constraint solving heuristics and efficient tableau methods are manifold. In fact, propositional satisfiability is an instance of finite-domain constraint solving where the variables only range over two values. The kind of changes one has to implement when adopting specialized heuristics are therefore similar in both frameworks. The real difference lies in the degree to which they can be easily implemented.
Many of the ideas for improved heuristics we explored rely on the more fine-grained control offered by tableau systems. Distinguishing different rule types makes it easier to assign costs to inference steps in the spirit of RAMG, and open branches that can be interpreted as partial models make it far easier to identify decision points where it pays off to introduce additional knowledge in the style of PRIDAS functions.
In the constraint programming paradigm, it is easier to think in terms of the desired results. The preferred way of modifying the behavior of a constraint solver is to use further constraints for axiomatizing the desired type of models. However, it is often unclear whether and how specific requirements can be translated into the constraint language, which means that we will have to exploit the limited possibilities for influencing the procedural aspect as well, with all the problems mentioned in the last section.
We have seen that the tableaux model has led to some interesting ideas about linguistically motivated model generation heuristics, and it seems that the constraint approach causes too many obstacles to their direct implementation to be attractive for experimental systems. Altogether, it therefore seems more promising to stick to tableau methods for experiments in specialized heuristics.

7 Combining Tableaux and Constraints

In the last two chapters, we have seen that both tableau and constraint methods have some potential for model generation in NLP. The most promising way to advance the state of the art therefore seems to be the development of combined strategies that make good use of the strengths of both paradigms.

7.1 Previous Approaches

There exist previous approaches to combining the two technologies, mainly by enriching tableaux with a constraint system. This strand of research is presented extensively in Caferra et al. (2004), and I will only summarize the main developments here.
The starting point is the idea to speed up model generation by keeping track of conditions that allow or prevent the application of rules in clausal calculi, as e.g. unwanted variable assignments or term equations. Such conditions can be encoded by constraints over the domains of variables occuring in formulae. Apart from restricting partial models, they can also be used to characterize instances that cannot be inferred from the theory. Exploiting such information in the model generation process is also called **disinference**.
Application of these ideas to the tableau calculus led to the method that Caferra et al. (2004) call **Refutation and Model Construction with Extended Tableaux (RAMCET)**, where tableaux are extended with **equational constraints** to substitute for problematic unification. These constraints constitute a compact representation of many models that would otherwise have been constructed redundantly, but handling and propagating the constraints leads to some overhead in implementations.
The basic RAMCET method is then extended in various ways, mostly with the goal of generating a large class of infinite, but finitely representable models. For this, a lot of technical machinery is needed, as e.g. **term schematizations** via integer exponents or tree automata. This kind of research is highly relevant for getting around the universe explosion problem, since e.g. the infinite line of ancestors resulting from the axiom "every person has a mother" can easily be dealt with. However, the resulting models are only represented very implicitly, making it impossible to access them using standard model checking technology. Moreover, the other mentioned challenges of model generation for NLP are arguably more pressing at the moment, and the RAMCET method is not of much use in addressing these issues, especially since the overhead of implicit model representation introduces unnecessary complexity into the finite case.

7.2 Combinatorial Subproblems

A popular technique of algorithmics is to reduce a problem to one or more instances of a simpler problem, or at least a problem whose internal structure is well-known and for which good solution methods exist. Propositional satisfiability (SAT) is such a well-understood problem, therefore many popular tableaux calculi rely on resolving quantification as early as possible. The normal form conversions used by many calculi can be conceived as doing this already in a preprocessing step.
While we cannot exclusively operate on clausal forms in the case of model generation, it is still worthwile to prioritize decisions that aim towards propositional subproblems of a very combinatorial nature. In essence, this means to decide on the domain size as early as possible, and this means we want to prioritize the resolution of existential quantification.
If we look at this goal from the perspective of a non-clausal tableaux calculus, an ideal format for an input formula would be one where existential quantification occurs as flatly embedded as possible. This is exactly the defining property of

a formula in **prenex normal form (PNF)**, which takes the form of a string of quantifiers that scope over a quantifier-free formula. Any first-order theory can be cast into PNF, which means we could simply use any full tableaux implementation if we convert input theories into PNF.
These considerations show that identity assumptions constitute the real decision points during the model generation process. Once these decisions have been made, what remains is a SAT problem determined by the respective instantiation choices. The complexity of the problem is of course still there, now mirrored in the fact that the SAT problems resulting from our choices will often be unsolvable, potentially requiring us to backtrack a lot.
To reduce the amount of bracktracking, it is vital to make informed instantiation decisions early on, which is one of the areas where PRIDAS functions have some potential. But this leads us into a tradeoff with the amount of information we already have during model generation. In the extreme case of PNF, the instantiation decisions already take place at a stage where the partial models contained in the open branches do not yet contain any atoms that could help in the instantiation decisions made by a PRIDAS function.
It becomes necessary to implement some kind of **look-ahead** capability that gives us as much information as possible about the future consequences of different instantiation decisions. Fortunately, in many cases it is possible to extract from a propositional formula atoms that will necessarily have to be true if the formula is to hold. This can e.g. by done via a **polarity automaton** that recursively descends into the formula while keeping track of negative scopes and exploits chaining inferences to deal with implication. The atoms thus extracted do not amount to a full solution of the SAT problem, but they offer useful input to a PRIDAS function which could then make informed instantiation decisions early on in the process.

7.3 Constraint Solving To Speed Up Analytical Tableaux

We have seen that tableaux are superior to constraint solvers in their openness to specialized heuristics. On the other hand, constraint solvers excel at solving combinatorical problems, and tableaux do not perform too well on them because of more expansive data structures in experimental implementations.
In the last section, we have taken steps towards a method of splitting the model generation task into one purely heuristic and many purely combinatorical problems. Given the strengths of tableau and constraint technology, this directly suggests a new way of combining the two approaches:
If we keep all the adaptability of heuristics in the tableau system of OletinMB, and resort to constraint solving technology for efficient solution of the resulting combinatorical subproblems, this might well lead us to a novel highly efficient model generation system that retains a lot of flexibility.

8 Conclusion

In this paper, we have encountered and assessed the main challenges in model generation for NLP, exploring how some of these problems might be approached by future model generation technology. We have seen that analytical tableaux are attractive because of their openness to linguistically motivated heuristics,

and that current constraint solving technology is superior in solving combinatorial subproblems. I have described the general idea behind a system that combines both approaches in a novel way, and mentioned some of the potential problems that would have to be solved by an implementation.
Much of the presented considerations remain to be carried out and assessed in practice. This will mainly be done within the framework of OletinMB, as it seems to be the only tableau system currently available that provides the required flexibility in influencing the heuristics. The next steps will be to experiment with linguistically motivated PRIDAS functions, and to experiment with an integration of JaCoP for the solution of the propositional problems that result from the instantiation choices.
In the long run, I hope to extend OletinMB into a full tableaux machine also for other logics. Most of the optimizations mentioned throughout this paper will furthermore have to be implemented sooner or later while I strive for a model generation system that is powerful enough to tackle logical representations as they arise from longer discourses.

References

P. Baumgartner & M. Kühn (2000). 'Abducing Coreference by Model Construction'. *Journal of Language and Computation* pp. 175–190.

P. Blackburn & J. Bos (2005). *Representation and Inference for Natural Language. A First Course in Computational Semantics*. CSLI Publications, Stanford, California.

G. Boolos (1984). 'Trees and finite satisfiability: Proof of a conjecture of Burgess'. *Notre Dame Journal of Formal Logic* **25**:193–197.

J. Bos (2003). 'Exploring Model Building for Natural Language Understanding'. In *Proceedings of the 4th Workshop on Inference in Computational Semantics (ICoS-4)*, pp. 41–55.

J. Bos & K. Markert (2005). 'Recognising textual entailment with logical inference'. In *Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing (HLT/EMNLP)*, pp. 628–635.

F. Bry & S. Torge (1998). 'A Deduction Method Complete for Refutation and Finite Satisfiability'. In *Proceedings of the European Workshop on Logics in Artificial Intelligence*, pp. 122–138, London, UK. Springer-Verlag.

F. Bry & A. Yahya (2000). 'Positive Unit Hyperresolution Tableaux and Their Application to Minimal Model Generation'. *Journal of Automated Reasoning* **25**:35–82.

A. Burchardt & S. Walter (2001). 'BuGS, A Tableau Machine for Language Understanding'. Master's thesis, Universität des Saarlandes.

R. Caferra, et al. (2004). *Automated Model Building*. Applied Logic Series. Kluwer Academic Publishers.

M. de la Banda, et al. (2006). 'The Modelling Language Zinc'. In *Principles and Practice of Constraint Programming - CP 2006*, pp. 700–705. Springer.

J. Dellert (2010). *Non-Minimal Model Building for Computational Semantics*. Term Paper for "Recognizing Textual Entailment". Universität Tübingen.

H. Fujita & R. Hasegawa (1991). 'A Model Generation Theorem Prover in KL1 Using a Ramified-Stack Algorithm'. In *Proceedings of ICLP 1991*, pp. 535–548.

S. Hedman (2004). *A First Course in Logic: An Introduction to Model Theory, Proof Theory, Computability, and Complexity (Oxford Texts in Logic)*. Oxford University Press, New York.

R. Hähnle (2001). 'Tableaux and Related Methods'. In *Handbook of Automated Reasoning*, chap. 3, pp. 101–178. Elsevier Science Publishers.

M. Kohlhase & A. Koller (2000). 'Towards A Tableaux Machine for Language Understanding'. In *Proceedings of Inference in Computational Semantics ICoS-2*.

M. Kohlhase & A. Koller (2001). 'Resource-Adaptive Model Generation as a Performance Model'. *Logic Journal of the IGPL* pp. 435–456.

K. Konrad (2004). *Model Generation for Natural Language Interpretation and Analysis*. Lecture Notes in Computer Science. Springer.

R. Manthey & F. Bry (1988). 'SATCHMO: A Theorem Prover Implemented in Prolog'. In *Proceedings of the 9th International Conference on Automated Deduction*, pp. 415–434, London, UK. Springer.

K. Marriott & P. J. Stuckey (1998). *Programming with Constraints: An Introduction*. MIT Press, Cambridge, Mass.

N. Nethercote, et al. (2007). 'MiniZinc: Towards a standard CP modelling language'. In *Principles and Practice of Constraint Programming - CP 2007*, pp. 529–543. Springer.