Non-Minimal Model Building for Computational Semantics

Seminar für Sprachwissenschaft
Eberhard Karls Universität Tübingen

Author:
Johannes Dellert
johannes.dellert@gmx.de

Course:
Computational Approaches to Recognizing Textual Entailment and Semantic Equivalence

Supervisor:
Prof. Detmar Meurers

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

The main goal of this research is to find ways of improving the quality of model building for computational semantics by designing and implementing a new model builder that is able to produce non-minimal models.

This is not limited to developing an algorithm that tends to produce linguistically more adequate models for typical formulae. It also means to open up the model building process, making it more transparent and offering hooks at which linguistic knowledge can be applied.

OletinMB, the model builder motivated and described in this paper, is a first step in this direction. By means of freely definable rankings of element identity assumptions, OletinMB allows to adapt search space exploration and the order of constructed models according to linguistic criteria.

I begin in Chapter 2 with a brief history of the field, explaining how the area that model building techniques were first applied to still determines the behaviour of model builders. Special attention is given to the convention of minimal model building, which was the obvious choice for the original areas of application, but has detrimental effects in natural language semantics.

In Chapter 3, I review the two main approaches to model building and some of their branches, comparing their applicability to computational semantics and explaining the history of the ideas behind OletinMB. Chapter 4 then introduces tableau methods in some detail, laying important theoretical groundwork for the understanding of my model building procedure.

The calculus behind OletinMB is then presented formally in Chapter 5, where I also explain the most important optimizations that turned out to be necessary for an efficient implementation. Chapter 6 presents the Prioritized Identity Assumption (PRIDAS) mechanism, which is OletinMB's main device for receiving external guidance during search space exploration. I also discuss the effects of two very simple PRIDAS functions, and develop ideas for linguistically motivated ranking functions.

In Chapter 7, I evaluate the quality of the models produced by OletinMB on formulae produced by the Nutcracker system for Recognizing Textual Entailment (RTE). I give concrete examples for the benefits of non-minimal model building and discuss the perspectives this opens for further research.

I presuppose a certain familiarity of the reader with first order logic as well as basic knowledge of automated reasoning techniques and computational semantics. Apart from that, experiences with RTE systems are helpful for understanding the implications of my findings.

2 Model Building for Computational Semantics

2.1 Automated Reasoning in Computational Semantics

Research efforts in natural language semantics have traditionally been directed to the question of how the meanings of utterances can be modelled by formal means. The tradition of formal logic, which has a long history in linguistics, has emerged as the most popular framework of expressing natural language semantics in a potentially language-neutral manner. Following the revolutionary ideas of Richard Montague, much research has been done on how natural language utterances can be translated into a logical language that formalizes and correctly predicts semantic differences between utterances.

With the advent of computing technology fast enough to perform symbolic computations on logical formulae, it has in principle become possible to automate the process of converting written language into logical statements.

Building on these advances, the next step has been to develop methods which actually make use of the logical representations to accomplish a range of advanced tasks. It has become possible to analyse the meaning of natural language input well enough to allow reasoning over it, e.g. in order to detect inconsistencies in natural-language descriptions.

For this purpose, usually existing tools from the branch of computer science known as automated reasoning are applied. This includes powerful theorem provers, which are mainly developed to assist mathematicians in finding and checking complex proofs for their theorems. Computational semantics arose with the first attempts to apply such tools to sets of formulae derived from natural language sentences.

2.2 Models as Meanings of Sentences and Discourses

Theorem provers turned out to be extremely useful for mathematics, but of limited applicability in linguistics. This is due to the fact that theorem provers mainly possess the ability to find proofs, which is essential for mathematics, but of limited use for linguistics. It is certainly fascinating to let a computer recognize that a natural language text is inconsistent by letting it try to prove its logical representation and end up with a contradiction.

However, finding contradictions and tautologies is arguably not the main aspect of what we commonly concieve as understanding a text. The meaning of a sentence has traditionally been described as the set of worlds in which a sentence is true. Using a theorem prover, we can only prove that a sentence is true in all worlds (= can be proved) or that it cannot be true because it contradicts itself (= its negation can be proved). But such sentences are barely meaningful from the point of view of communication, as they in general add no new information to the discourse.

The linguistically interesting formulae lie between these two extremes, their truth values depend on the world in which they are evaluated. In logical terms,

we are not interested in validity, but in satisfiability, or in other words, in the existence of models for these formula. A model of a formula can be conceived as a world in which that formula is true. Intuitively, the properties of the worlds in which a formula is true contain a lot more semantic information than whether the formula is always valid.

When we hear a story, a good approximation of what happens in our minds while processing the meaning is that we build a world within our minds. The sentences the story is composed of contain descriptions of such a world, and we adapt our internal model of this world as the story progresses. If we want to simulate this process by computational means, we have to find ways to generate models from the formulae we have derived as representations for the meanings of utterances.

Luckily, the automated reasoning community has in recent years produced a new class of specialised programs called model builders. With the help of such tools, it is possible to directly compute models for sets of formulae, providing for the computational equivalent of creating hypothetical worlds in our minds.

2.3 Usage of Model Builders in Mathematics

To understand why model building as a field has developed in the way it has, it is necessary to know what kind of mathematical problems model builders were constructed to solve. The main reason for the rise of model building as a reasoning task is that mathematicians often define classes of interesting mathematical objects by means of sets of axioms. A model of the axioms taken together is an example of the structures they admit. Building on a few elementary notions like sets and relations, only a few logical axioms are needed to define classes of mathematical objects such as trees, or groups, or lattices.

While systematically exploring the space of possible structures, mathematicians are often confronted with the question which kind of objects are described by a given set of axioms. Often, a well-known set of axioms is slightly adapted to see what consequences small changes in the axioms have on the structures described. For larger sets of axioms, this kind of research is very demanding, at some point even exceeding what a trained mathematician can achieve by hand in reasonable time. To facilitate this kind of research, model builders have been designed to explicitly produce structures admitted by a set of axioms.

Another important application for model builders in mathematics is the automated search for counterexamples. The most elegant way of disproving a conjecture or a mathematical assumption is to exhibit a structure for which the assumption does not hold. Without a good intuition on what the problem cases could be, searching for such counter examples can be very time consuming. For various reasons, mathematicians are also often interested in the largest or smallest counterexamples to a theory.

2.4 The Traditional Emphasis on Minimal Models

With this applicational background in mind, it is not very surprising that most model builders are tuned towards finding **minimal models** for formulae of first-order logic. A model of a formula is minimal iff it is not possible to find another model of the formula with a lower number of elements in its universe. Mathematicians tend to look for the smallest models of their axioms because such models are usually still small enough to be fully understood.

Another reason why the construction of minimal models is the standard behaviour of current model builders is that most satisfiable formulae have a large number of, if not infinitely many arbitrarily complex models, and it is difficult to decide which of the many models can be considered canonical and should be produced by a model builder.

Minimal models are also often the prototypical and most interesting cases for further analysis, because the decisive properties of the object class described by a set of axioms are often visible in them without distractions caused by additional structure that would not be necessary to fulfill the axioms.

2.5 Model Builders in Computational Semantics

Among the first to propagate the use of model builders for computational semantics were Blackburn & Bos (2005) in their seminal work on computational semantics. Their idea was to tackle the problem of first order indecidability by letting a theorem prover and a model builder run in parallel on their logical representations. The theorem prover would try to prove that new sentences followed from the previous discourse, discarding them as not informative if this was the case. In parallel, the model builder would try to find a model of the previous discourse enlarged by the sentence's negation, showing that the new sentence is informative because it does not follow from the previous discourse.

Given the complexity of state-of-the-art theorem provers and model builders, B & B were happy to use off-the-shelf tools produced by the automated reasoning community for their prototype system CURT. The performance of existing solutions was good enough for them not to waste energy on thinking about, let alone implementing automated reasoning tools geared towards their particular class of reasoning tasks.

This approach has been carried over to the first attempts of applying techniques of deep semantics to tasks such as RTE (Recognizing Textual Entailment). The model building technology used by Bos & Markert (2005b) in their Nutcracker RTE system is basically still the same as the one used by Blackburn & Bos in their very first experimental systems.

With OletinMB, I strive to develop a model builder specialized in producing more accurate models for natural language utterances. I try to achieve this by giving up some of the established practices for model builders tailored towards mathematical problems. The most important of these changes is that I systematically produce non-minimal models.

3 Review of Model Building Techniques

In this chapter, I will give a brief overview of the current state of the art in **finite model building for first-order logic**, with a special focus on the applicability of the different approaches and their implementations to computational semantics.

Model building for significantly stronger logics is generally considered intractable, and there has not been much research in that area, whereas infinite models have not yet been found to be of much relevance for natural language semantics. For further discussion of higher-order model building with a linguistic application, the reader is referred to the discussion in Konrad (2004) and the literature cited there. The current state of the art in general-purpose model building is presented in Caferra et al. (2004), where finite descriptions for and the construction of infite models are described as well.

3.1 Goals and Scope of Model Building

As an introductory remark, it is worth mentioning that model building for first-order logic is an extremely difficult task to perform efficiently. There cannot be general solutions because satisfiability of predicate logic over finite structures is undecidable.

In fact, finite model building for first-order logic is not even recursively enumerable, as follows directly from Trakhtenbrot's theorem and the lack of duality for undecidability (see e.g. Hedman (2004), p. 412ff for details). This means that model building is an even harder problem than the notoriously difficult theorem proving, which is also undecidable, but at least recursively enumerable.

Notwithstanding these severe theoretical limitations, automated reasoning methods are very useful as a tool for many areas of research. While on the theoretical side, the situation would seem to be quite hopeless, it still turned out to be possible to implement model builders which are able to build models for many problems occurring in practice. However, there always have to be gaps in coverage, and a method that may perform badly on a certain class of formulae is often the best choice for another class.

The methods developed in the field differ mainly in their degree of optimization to certain classes of problems, whereas conceptually there are only two different major approaches which have had some success, both of which will now be discussed in turn.

3.2 Enumerative Methods

The more successful of the two main approches to model building is the **enumerative approach**, which is based on systematically generating all possible first-order structures of a given domain size and essentially trying them out via an efficient model checking algorithm until a model is found. As brute-force as this type of approach may sound, using a number of intelligent optimizations this actually makes it possible to generate small models very efficiently. The

most important optimizations include ways of quickly finding contradictions which exclude whole classes of models which then do not have to be generated any more, and ways to avoid generating and checking many isomorphic cases.

This basic idea of directly exploiting the finiteness of the set of possible structures over a given domain and a given alphabet was first implemented by Slaney (1992) in his **FINDER** (**FIN**ite **D**omain **E**numerato**R**). Slaney's main contribution was the discovery of feasible enumeration methods for domains of useful sizes, in his cases being around 10 to 15. FINDER achieves this via a backtracking search, avoiding to bracktrack twice for the same reason. The major disadvantage is that FINDER is not refutationally complete, and it does not find Herbrand models, instead explicitly specifying the values of functions on arbitrary domains.

The first widely successful finite model builder for first-order logic was **MACE** (**M**odels **A**nd **C**ounter**E**xamples) by McCune (1994), which relies on flattening the first-order formula to a propositional formula equivalent for a given domain size, effectively reducing model building to a SAT problem which is then solved using an efficient Davis-Putnam method. MACE-style model builders were the first to be successfully applied to hitherto unsolved mathematical problems from lattice theory, and continue to be very popular.

The last major innovation in enumerative model building was the introduction of constraint programming techniques by Zhang & Zhang (1995). Their **SEM** (**S**ystem for **E**numerating finite **M**odels) relies on an efficient implementation of constraint propagation and a heuristic tailored to avoiding consideration of many isomorphic models. SEM performs a lot better than MACE on larger formulae with many equations, while MACE is more of a general-purpose method.

The development of enumerative approaches has continued since then, but improvements have become more gradual. For the most recent large innovation and the current state of the art for MACE-style model building, I refer the reader to Claessen & Sörensson (2003).

Apart from the maturity and performance of the implementations, another reason for the popularity of enumerative approaches is that they tend to have some properties favoured by the mathematician. If an incremental search through domain sizes does not yield any models of a certain domain size, the mathematician can be sure there are no such models, which can already be a very important insight. The problem that there are always some formulae on which a given model builder will not terminate can be alleviated by using an enumerative method and only looking for models of a certain cardinality, which over finite alphabets is a decidable problem.

Although especially MACE has been successfully applied to reasoning tasks in computational semantics, enumerative model builders are not really a natural choice for linguistic applications. The effiency-enhancing features of enumerative methods generally tend to work well for logical languages with function symbols and lots of equations, as is typical for problems in algebra. Formulae occurring in computational semantics tend to exhibit neither function symbols

nor equations to a high degree, diminishing the effect of most optimizations. Fortunately, domain sizes in linguistic models have tended to be rather smallish and well within the scope of enumerative approaches even without much optimization.

3.3 Deductive Methods

The second influential approach to finite model building is usually called the **deductive approach**. Unlike enumerative methods, who essentially build up their models in the form of multiplication tables, deductive methods use **atomic representations** of partial models during the process.

Most deductive methods build on tableau calculi, the most prominent of which is **P**ositive **U**nit **H**yper-**R**esolution (**PUHR**). The approach is based on analytical tableaux and tries to solve the efficiency problem caused by the blind instantiation of γ-formulae (see the next chapter for explanations of the terminology). It does this by converting the formulae into a clausal form and compiling them into **range-restricted rules** of the form $C = a_1 \wedge ... \wedge a_m \Rightarrow b_1 \vee ... \vee b_n$, where $a_1, ..., a_m$ and $b_1, ...b_n$ are atoms.

The resulting tableau construction rules are a lot less complex than the ones I will present, and the efficiency gain can be considerable. However, the conversion into clausal form can be very costly, and it requires skolemization, which hampers some of the otherwise nice properties of the method. For detailed information on the PURC calculus, the best source is Bry & Yahya (2000).

PUHR is in essence a formalization of the behavior of one of the oldest model builders, **SATCHMO** (**SAT**isfiability **CH**ecking by **MO**del generation) by Manthey & Bry (1988). SATCHMO was conceptualized as an enhancement to the Prolog programming language that allows it to handle negative clauses and clauses containing more than one literal. It is thus essentially a variant of Prolog which can handle clauses of the form $C : H_1, ..., H_m \leftarrow B_1, ..., B_n$.

Conversion into range-restricted rules then means that during generation of a proof for a goal, all generated facts will be ground, which results in a collectable atomic representation of a model of the goal. SATCHMO has constantly been improved over time with novel search-space reducing techniques, I give **I-SATCHMO** by He (2001) as an example of recent contributions.

A variant of PUHR that preserves completeness for finite satisfiability is the **EP** (**E**xtended **P**ositive) tableau calculus by Bry & Torge (1998). The input to EP has to be provided in the form of **Positive Restrictive Quantification** (**PRQ**) rules, which are of the general form $\forall x(C \rightarrow \exists y F)$, where C is a conjunction of atoms and F a disjunction of atoms and PRQ rules. It is then possible to apply a tableau method without skolemization, while at the same time preserving much of the efficiency gains of operating on range-restricted rules. The approach I am using in OletinMB could be described as a variant of EP which operates on general first-order formulae, not requiring prior conversion to PRQ rules, but with some loss of efficiency.

In theory, deductive methods should exhibit far better behaviour than enumerative methods especially on large domain sizes because they avoid the combinatorical explosion caused by producing all possible models. However, tableaux can only be represented by rather expensive data structures, and the structure of the search space is generally less well-understood, which makes it a lot easier to optimize enumerative methods.

Although deductive methods have rarely been used in computational semantics so far, I see a lot of potential for such methods to be successfully applied to linguistic model building. A case in point is the constraint-based method developed by Konrad (2004) for a higher-order logic that contains most of the formal mechanisms used in formal semantics, including certain kinds of set and function variables. As Konrad points out, enumerative methods are completely unfeasible here because the domains get inflated with constants denoting flattened higher-order entities and soon become intractably huge. Using sophisticated constraint solving techniques, Konrad's system manages to compute such models in reasonable time, with some interesting linguistic applications.

Apart from the fact that deductive methods are interesting because they can demonstrably tackle (fragments of) higher-order logics, they also have the advantage of being much more open to modifications. Tableau methods in particular tend to preserve a lot more of the original formula structure than the conversions necessary for enumerative methods, which provides entry points for all kinds of tweaks, including steering the tableau building process by external heuristics. This is the direction I am exploring with OletinMB, and we shall therefore first have a closer look at the mechanics of tableau-based model building before exploring my ideas in that area.

4 The Tableau Method

The tableau method is originally a technique of theorem proving, used to refute the negation of a theory we want to prove. As was noticed early in the history of automatic reasoning, the tableau method lends itself well to simultaneous search for proofs and models. This is mainly because the tableau data structure keeps together the information representing a partial model, and unlike e.g. in resolution methods, the structure of these partial models is preserved during the reduction process.

I will quickly remind the reader of the basic ideas and issues of the tableau method for theorem proving, then explaining along the lines of Caferra et al. (2004) and following the ideas of Boolos (1984) why this method only requires minor changes to provide us with a naive, but fully functional model builder.

4.1 The Propositional Case

The tableau method as applied to theorem proving tries to reject a first order formula by deriving a contradiction from it. The basic idea driving the process and determining the shape of the tableau data structure is that of a **recursive distinction of cases**.

Every case distinction we must exhaust leads to an additional branching in a tree structure with the formula to be refuted at the root and its subformulae at the nodes. As the structure is expanded, the formula is eventually broken down into literals, i.e. atomic propositions and their negations. Formulae that have been expanded once become inactive, which allows the method to terminate once all formulae have been extended.

A single branch of the tableau from the root to a leaf always contains formulae that have to be true at the same time. We can thus detect contradictions within branches by looking for pairs of clashing literals that occur together in a branch, a process commonly referred to as **closing a branch**. If we manage to close all branches, we have refuted the formula for all cases, thus proving the invalidity of the root formula and the validity of its negation.

The following rules define the behaviour of a classical propositional tableau:

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

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

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**.

As can be shown by a simple induction on the depth of the formulae being expanded using the fact that every rule application lowers the depth of a formula, the tableau calculus for propositional logic is both sound and complete.

4.2 Dealing with Quantification

In order to extend the tableau method to predicate logic, we need additional rules to reduce the quantifiers. Such rules exist in several variants, all much more complicated than the rules for the propositional case.

The main problem is that resolving universal quantification would in principle require us to introduce an infinite number of new leaves to a branch, because it formally corresponds to a conjunction of infinitely many conjuncts. Analogously, existential quantification can result in infinitely many case distinctions, because it formally corresponds to a disjunction over an infinite set of formulae.

It is possible to partially resolve this problem by introducing a special treatment for both quantifiers. Existential quantification is generally processed using **skolemization**, i.e. the replacement of the variable it binds with a skolem function. The function's value can be determined later by **unification**, making skolemization a systematic way of delaying decisions until we know enough about the domain to avoid infinity. Rules for the treatment of existential quantification in tableaux are traditionally called δ**-rules**.

The traditional solution for γ**-rules**, i.e. the treatment of universal quantifiers, is to let them range only over the constants as well as the set of entities that are produced during skolemization. However, universal formulae still cannot become inactive after being expanded, because they clearly also have to hold for entities that are introduced later e.g. by resolving some deeper-embedded existential quantifier. This problem cannot be avoided, so unlike in the propositional case, we cannot guarantee that at some point the tableau will be completely resolved without any active formulae left, which forces us to live with **non-termination** in some cases, which is also a theoretical limitation.

Two regimes for handling quantifiers have been developed in attempts to maxime speed and coverage of the tableau method for predicate logic. The method of **analytical tableaux** is based on a simple version of skolemization for existential quantification, and non-deterministically substituting ground terms for resolving universal quantification:

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

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

The advantage of analytical tableaux is that we can introduce skolem functions of zero arity or **skolem constants** because the quantification rule eliminates the dependencies on quantifiers with higher scope and their bound variables. This is a nice feature because it allows us to get along entirely without costly unification.

However, in the case of theorem proving, we are interested in closing all branches of a tableau as quickly as possible. During processing of universal quantification this requires us to pick **non-deterministically** from a potentially huge set of ground terms one of the few which will give us a valid counterexample and lead us to the desired contradictions in all cases later in the tableau. Using no guidance by unification here can lead to substantial amounts of unnecessary backtracking. As a result, the performance of the method relies entirely on the quality of early and rather un-informed guesses, with disastrous consequences for performance on certain malicious problem classes.

For this reason, another alternative for the γ- and δ-rules is more popular in general-purpose theorem proving. In so-called **free-variable tableaux** the resolution of universal quantification is delayed by replacing the bound variable by a new one. Existential quantification is then dealt with by skolemizing over the free variables resulting from this process.

$$(\forall)\frac{\forall x.\gamma(x)}{\gamma(x')},\ x' \textit{ a variable not occurring elsewhere in the tableau}$$

$$(\exists)\frac{\exists x.\delta(x)}{\delta(f(x_1,...,x_n)},\ f \textit{ new function symbol},\ x_i \textit{ free variables in } \delta$$

The values of the skolem function are then only determined via unification during attempts to close branches. The unification causes free variables to be

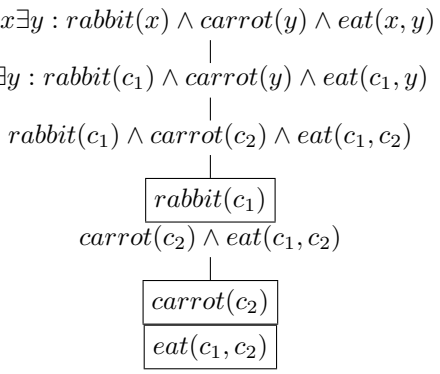

Figure 1: Classical tableau for $\exists x \exists y : rabbit(x) \wedge carrot(y) \wedge eat(x, y)$

assigned to a value, and this value must be respected in other branches as well. This method has the advantage that it avoids backtracking by delaying decisions as much as possible, which is more efficient for theorem proving. The disadvantage is a higher memory usage because the whole tableau must be kept in memory and unification always influences formulae everywhere in the tableau, making the required changes non-local.

4.3 Saturated Open Branches Represent Models

After this sketch of classical tableaux as used in theorem proving applications, we will now tweak the method a little in order to produce models. To see why this is relatively straightforward, consider the tableau in Figure 1.

Note that the framed literals in the only branch can directly be used to construct a complete model of the formula, its domain corresponding to the skolem constants we introduced while extending the branch. We are thus not any longer interested in closed branches that would indicate contradictions, but our goal is to find **saturated open branches**, which cannot be extended any more and failed to be closed while being extended. 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.

In principle, this approach also works with universal quantification, as demonstrated in Figure 2. In this more complex case, it becomes obvious that the method is constructing models of increasing size. If we traverse the branches of this tableau, we start out with small models and can work our way up to models of arbitrary size.

It is also important to note that the models are not fully specified by the literals on the branches. Each set of branch literals merely constitutes a minimal description for a class of models with additional structure. Consider the second branch, which contains the literals $carrot(c_2)$, $eat(c_1, c_2)$, and $\neg rabbit(c_2)$. Out of these, we can construct a model which does not specify whether $carrot(c_1)$ or $\neg carrot(c_1)$, and whether $rabbit(c_1)$ or $\neg rabbit(c_1)$. One could treat any of these combinations as different models, but then one could also integrate all

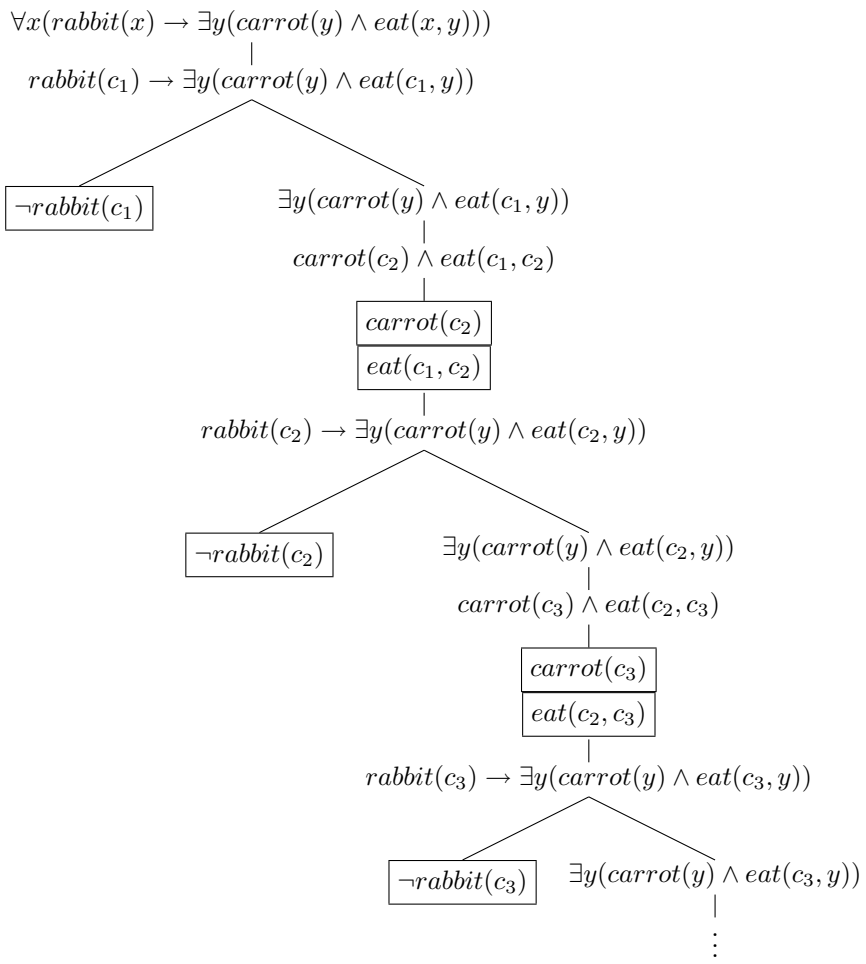

Figure 2: Classical tableau for $\forall x(rabbit(x) \rightarrow \exists y(carrot(y) \wedge eat(x,y)))$

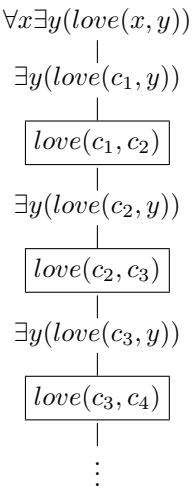

Figure 3: Classical tableau for $\forall x \exists y(love(x, y))$

kinds of other entities and relations to the models that have nothing to do with the formula. In that sense, a model as we are producing it here is always an underspecified description.

As we can see in Figure 3, the naive use of the tableau method for model building can quite easily lead to non-determination although the formula is satisfiable. The problem is that by the default rules, every existential quantifier we meet requires the introduction of a new constant symbol and thus a new element in our models. This makes sense for theorem proving, because we want to find contradictions in the general case, which only requires us to pick a few useful substitutions during γ-rule application.

However, in model building our goal is not to close all branches for a given instantiation by finding contradictions, but to be efficient in finding saturated branches, special cases where the universal formulae hold for all constants contained in the formulae or introduced by δ-rule applications.

If we allow the reuse of introduced skolem constants when expanding existential quantifiers, this amounts to looking at particular cases with additional **identity assumptions**. And indeed, models are nothing other than such particular cases.

This observation leads to the idea of treating existential quantification by allowing skolemization with skolem constants generated at an earlier stage as well as skolemizing with a new constant. With this modification, our variant of the tableau method turns into a valid model building mechanism, as can be seen in Figure 4.

The main problem caused by this change to the tableau building rules is that we now have to non-deterministically choose existential instantiations. As was the case for universal quantification before, it is difficult to guess the correct existential initializations early on, and suboptimal choices can quickly lead to

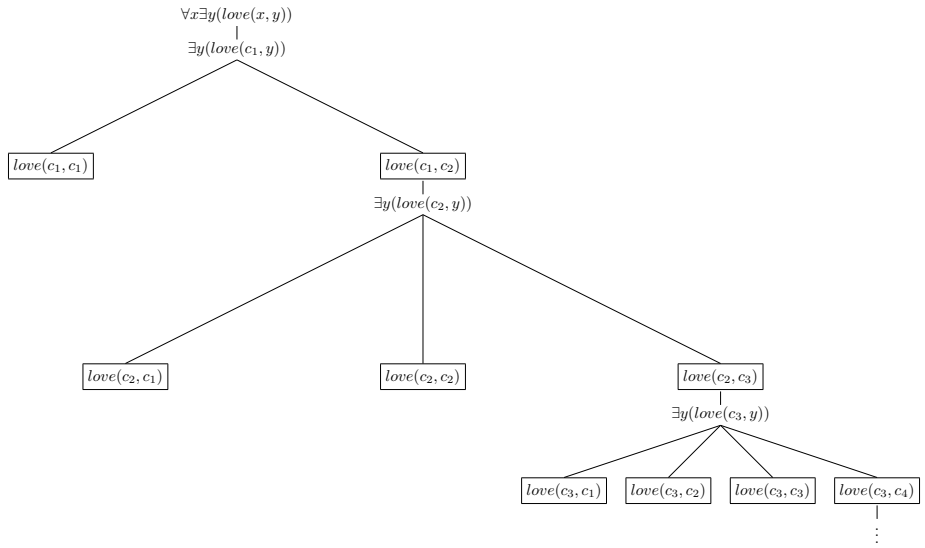

Figure 4: Modified tableau for $\forall x \exists y(love(x,y))$

extensive backtracking and the expansion of intractably many branches that are doomed to failure. Making these choices in an intelligent and well-informed manner, or delaying them by means of constraint systems, have been the main issues in developing refinements to this basic method.

4.4 Completeness for Finite Satisfiability

An important criterion in the evaluation of finite model building methods is whether they are **complete for finite satisfiability**. A model building method has this property if it is guaranteed to find some finite model if any such model exists. This does not imply that the method must be able to find every possibly finite model.

The first person to develop a tableau method that was provably complete for finite satisfiability was Boolos (1984), whose goal was to develop a model building method with that property. The tableau method we just introduced is largely based on his ideas, and this method can be considered the ancestor of all tableau methods for model building.

A problem that has influenced many design decision in tableau-based model building algorithms is that we would have to avoid skolemization with free variables in order to preserve completeness for finite satisfiability. To see why this is so, consider the following example taken from 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. This demonstrates that the introduction of additional function symbols is risky because we lose finite satisfiability. Unfortunately, no variant of skolemization over free variables can avoid this risk

in all cases. To retain completeness for finite satisfiability, we can only use skolem constants as a variant. This is exactly what Bry & Torge (1998) do in their Extended Positive (EP) tableau calculus.

5 Implementation within Oletin

The claims of this paper have been tested by implementing a new model builder within the **Oletin** framework (a Finnish neologism roughly translating as "assumption device"), which is the preliminary name of my self-designed framework for efficient formula and model manipulation, implemented entirely in Java.

Oletin provides utility classes for processing formulae and models in various standard formats, e.g. in the Prolog format used by Blackburn & Bos (2005) and with unicode symbols for pretty pritting. It can read and write formulae and models in MACE format, making it possible for my model builder to completely copy MACE's behaviour, so that it can be used and evaluated with minimum effort in all applications which already provide an interface for MACE.

The framework also features the option of outputting models in the DOT language used by GraphViz[1]. All the models graphs in Chapter 7 were generated using this method.

5.1 Basic Structures and Architecture

To make the process more transparent, I decided not to convert the logical formulae into a particular simplified form. Instead, the tableau implementation in OletinMB operates on arbitrary first-order formulae. Usually, one would not want to do this because it can result in unnecessarily huge tableaux and suboptimal bounding of the search space.

However, with a few shortcuts and optimizations which I am going to discuss in the next section, this turned out to be a feasible option, perhaps assisted by an inherent tendency of linguistic formulae to be structured in a friendly manner.

I decided against a free-variable tableau and opted for the analytical variant because I wanted to avoid function symbols in my formulae and thus had to circumvent skolemization, with the added benefit that my system preserves completeness for finite satisfiability.

Choosing the analytical tableau variant also has the advantage that I can save all the relevant information about the active branches in an agenda-like structure without keeping all the inactive formulae in memory.

The established model building method most closely related to mine thus is the Extended Positive tableau calculus by Bry & Torge (1998), which makes the same choices concerning tableau formalism and the avoidance of skolemization. The most relevant difference is that EP requires formula conversion into

[1] see http://www.graphviz.org

a special form, while my implementation works on arbitrary first-order formulae. EP thus chooses efficiency over transparency and modifiability, while for OletinMB I decided to stay as transparent and open for problem-specific modifications as possible at the expense of somewhat reduced performance.

The basic working procedure is that the model builder manages a stack of active branches, leading to a depth-first exploration of the space of possible models. In each step, as long as the stack is nonempty, the next active branch is taken from the top of the stack, and the tableau rules are applied. This results either in a modified branch that gets pushed onto the stack again, or in two new branches which are pushed onto the stack while the old branch is discarded. If a rule results in a closed branch, it will simply not be added to the agenda again. For δ-rule applications, a stack of backtracking points is maintained, with a copy of the respective branch at that moment together with an agenda of yet unconsidered existential instantiations.

5.2 Pruning the Search Space

As was to be expected, my rather naive initial method did not perform as well as desired because it explored the space of possible models in a rather brute-force manner. In my attempts to prune the search space as early as possible without resorting to complicated constraint mechanisms, the following structure for storing active branches $\mathcal{B}$ turned out to be useful:

$\mathcal{B} =< \Phi, \Psi, \Omega, C, A >$ where
Φ is the **non-branching agenda**, a set of active conjunctive formulae,
Ψ is the **branching agenda**, a set of active disjunctive formulae,
Ω is the **universal theory**, a set of active universal formulae,
C is a set of constant symbols, and
A is a set of ground literals together representing a **partial model**.

For ease and clarity of presentation, I enlarge this structure by an additional set of **pending formulae** Π, which makes the formal presentation much more concise. For efficiency reasons, there is no corresponding structure in the code.

With these preparations, the basic ideas of my optimized implementation for the tableau rules can then be described as a calculus in the following way:

Initialization and completion:

$$\frac{buildModel(\phi)}{< \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \{\phi\} >} \qquad \frac{< \emptyset, \emptyset, \Omega, C, A, \emptyset >}{success: \ C \ and \ A \ define \ model}$$

During initialization, the formula to be processed is added to the set of pending formulae, and all other parts of the structure start out empty. Model building is complete if both the non-branching and the branching agendas are empty, and there are no more pending formulae to take into account.

Splitting conjunctives:

$$\frac{<\{\phi_1 \wedge \phi_2\} \cup \Phi, \Psi, \Omega, C, A, \emptyset>}{<\Phi, \Psi, \Omega, C, A, \{\phi_1, \phi_2\}>} \qquad \frac{<\{\neg(\phi_1 \vee \phi_2)\} \cup \Phi, \Psi, \Omega, C, A, \emptyset>}{<\Phi, \Psi, \Omega, C, A, \{\neg\phi_1, \neg\phi_2\}>}$$

$$\frac{<\{\neg(\phi_1 \rightarrow \phi_2)\} \cup \Phi, \Psi, \Omega, C, A, \emptyset>}{<\Phi, \Psi, \Omega, C, A, \{\phi_1, \neg\phi_2\}>}$$

The splitting rules are the main mechanism of extracting subformulae, always processing only one agenda formula at a time and adding its components to Π. All the pending formulae must then be processed before the splitting rule can be applied again, which turns the splitting of conjunctives into an entirely deterministic process.

Distributing conjunctives:

$$\frac{<\Phi, \Psi, \Omega, C, A, \{\phi_1 \wedge \phi_2\} \cup \Pi>}{<\{\phi_1 \wedge \phi_2\} \cup \Phi, \Psi, \Omega, C, A, \Pi>} \qquad \frac{<\Phi, \Psi, \Omega, C, A, \{\neg(\phi_1 \vee \phi_2)\} \cup \Pi>}{<\{\neg(\phi_1 \vee \phi_2)\} \cup \Phi, \Psi, \Omega, C, A, \Pi>}$$

$$\frac{<\Phi, \Psi, \Omega, C, A, \{\neg(\phi_1 \rightarrow \phi_2)\} \cup \Pi>}{<\{\neg(\phi_1 \rightarrow \phi_2)\} \cup \Phi, \Psi, \Omega, C, A, \Pi>}$$

The task of the distribution rules is to correctly assign pending formulae to their respective agendas. The interplay of splitting and distributing rules is thus responsible for the recursive construction of the tableau.

Double negation rule:

$$\frac{<\{\neg\neg\phi\} \cup \Phi, \Psi, \Omega, C, A, \Pi>}{<\Phi, \Psi, \Omega, C, A, \{\phi\} \cup \Pi>}$$

A treatment for double negation is needed to cover all first order formulae in arbitrary form, I simply make use of the tautology $\neg\neg\phi \rightarrow \phi$ here.

Splitting disjunctives:

$$\frac{<\emptyset, \{\psi_1 \vee \psi_2\} \cup \Psi, \Omega, C, A, \emptyset>}{<\emptyset, \Psi, \Omega, C, A, \{\psi_1\}> \qquad <\emptyset, \Psi, \Omega, C, A, \{\psi_2\}>}$$

$$\frac{<\emptyset, \{\neg(\psi_1 \wedge \psi_2)\} \cup \Psi, \Omega, C, A, \emptyset>}{<\emptyset, \Psi, \Omega, C, A, \{\neg\psi_1\}> \qquad <\emptyset, \Psi, \Omega, C, A, \{\neg\psi_2\}>}$$

$$\frac{<\emptyset, \{\psi_1 \rightarrow \psi_2\} \cup \Psi, \Omega, C, A, \emptyset>}{<\emptyset, \Psi, \Omega, C, A, \{\neg\psi_1\}> \qquad <\emptyset, \Psi, \Omega, C, A, \{\psi_2\}>}$$

These are the only rules which can introduce additional branches to the tableau. Observe that they can only be applied if the non-branching agenda is empty. The efficiency gain resulting from delaying branching is the main reason for separating the branching from the non-branching agenda.

Distributing disjunctives::

$$\frac{<\Phi, \Psi, \Omega, C, A, \{\psi_1 \vee \psi_2\} \cup \Pi>}{<\Phi, \{\psi_1 \vee \psi_2\} \cup \Psi, \Omega, C, A, \Pi>} \qquad \frac{<\Phi, \Psi, \Omega, C, A, \{\neg(\psi_1 \wedge \psi_2)\} \cup \Pi>}{<\Phi, \{\neg(\psi_1 \wedge \psi_2)\} \cup \Psi, \Omega, C, A, \Pi>}$$

$$\frac{<\Phi, \Psi, \Omega, C, A, \{\psi_1 \rightarrow \psi_2\} \cup \Pi>}{<\Phi, \{\psi_1 \rightarrow \psi_2\} \cup \Psi, \Omega, C, A, \Pi>}$$

Pending formulae that require branching are moved to the branching agenda.

Closing branches:

$$\frac{< \Phi, \Psi, \Omega, C, \{\neg\phi\} \cup A, \{\phi\} \cup \Pi >}{closed} \qquad \frac{< \Phi, \Psi, \Omega, C, \{\phi\} \cup A, \{\neg\phi\} \cup \Pi >}{closed}$$

Detecting a contradiction between an established literal and a pending literal leads to the closing of the current branch. The model builder first considers other active branches introduced by splitting disjunctives. Then, if all branches resulting from some existential instantiation are closed, the model builder backtracks and tries another instantiation instead.

Adding literals:

$$\frac{< \Phi, \Psi, \Omega, C, A, \{\phi\} \cup \Pi >}{< \Phi, \Psi, \Omega, C, \{\phi\} \cup A, \Pi >} \phi \textit{ is a literal and } \neg\phi \notin A$$

$$\frac{< \Phi, \Psi, \Omega, C, A, \{\neg\phi\} \cup \Pi >}{< \Phi, \Psi, \Omega, C, \{\neg\phi\} \cup A, \Pi >} \neg\phi \textit{ is a literal and } \phi \notin A$$

If there is no contradiction between the partial model and a pending literal, the new literal is used to extend the partial model. This rule is the only device for introducing new facts into the model.

Universal quantification:

$$\frac{< \Phi, \Psi, \Omega, \{c_1, ..., c_n\}, A, \{\forall x(\phi(x))\} \cup \Pi >}{< \Phi, \Psi, \{\forall x(\phi(x))\} \cup \Omega, \{c_1, ..., c_n\}, A, \{\phi(c_1), ..., \phi(c_n)\} \cup \Pi >}$$

$$\frac{< \Phi, \Psi, \Omega, \{c_1, ..., c_n\}, A, \{\neg\exists x(\phi(x))\} \cup \Pi >}{< \Phi, \Psi, \{\forall x(\neg\phi(x))\} \cup \Omega, \{c_1, ..., c_n\}, A, \{\neg\phi(c_1), ..., \neg\phi(c_n)\} \cup \Pi >}$$

Universal quantification is applied as early as possible, introducing instantiations for all the constant symbols defining the universe so far. The universal statements are then stored in the universal theory to be instantiated again as soon as new elements are introduced to the universe.

Existential instantiation with previously introduced constant:

$$\frac{< \Phi, \Psi, \Omega, \{c_1, ..., c_i, ...c_n\}, A, \{\exists x(\psi(x))\} \cup \Pi >}{< \Phi, \Psi, \Omega, \{c_1, ..., c_i, ...c_n\}, A, \{\psi(c_i)\} \cup \Pi >}$$

$$\frac{< \Phi, \Psi, \Omega, \{c_1, ..., c_i, ...c_n\}, A, \{\neg\forall x(\psi(x))\} \cup \Pi >}{< \Phi, \Psi, \Omega, \{c_1, ..., c_i, ...c_n\}, A, \{\neg\psi(c_i)\} \cup \Pi >}$$

Alternatives for existential instantiations, which are the main decision points while exploring the search space, include instantiating existentially quantified formulae with constants introduced during previous existential instantiations.

Existential instantiation with new skolem constant:

$$\frac{< \Phi, \Psi, \{\forall x(\phi_1(x)), ..., \forall x(\phi_m(x))\}, \{c_1, ..., c_{n-1}\}, A, \{\exists x(\psi(x))\} \cup \Pi >}{< \Phi, \Psi, \{\forall x(\phi_1(x)), ..., \forall x(\phi_m(x))\}, \{c_1, ..., c_n\}, A, \{\psi(c_n), \phi_1(c_n), ..., \phi_m(c_n)\} \cup \Pi >}$$

$$\frac{< \Phi, \Psi, \{\forall x(\phi_1(x)), ..., \forall x(\phi_m(x))\}, \{c_1, ..., c_{n-1}\}, A, \{\neg\forall x(\psi(x))\} \cup \Pi >}{< \Phi, \Psi, \{\forall x(\phi_1(x)), ..., \forall x(\phi_m(x))\}, \{c_1, ..., c_n\}, A, \{\neg\psi(c_n), \phi_1(c_n), ..., \phi_m(c_n)\} \cup \Pi >}$$

Introducing a new skolem constant makes it necessary to not only enlarge the partial universe by a new constant symbol, but also to apply the universal theory stored in the branch to the new element.

Universal quantification is thus handled by applying any new universal statement to all constants introduced so far, and by applying to every new constant all the universal formulae introduced so far. This means that at any given point in the process, universal quantification is fully accounted for, in this way the model builder uses much of the available information at all stages, vastly reducing the number of branches that have to be explored.

Given this rather straightforward implementation of the full analytical tableau calculus, it is not surprising that my model builder was still considerably less efficient than other implementations, which generally have a lot more sophisticated theories behind them, all with the goal of using the information available at the branches in a much more thorough fashion.

Severe performance problems on the larger formulae I wanted to test my approach on forced me to develop additional techniques in order to make use of that information. The guideline for these additional measures was to detect inconsistencies as early as possible to tame the enormous search spaces.

My current implementation, which is by no means the end of the rope, includes a special treatment for implications with literal antecedents. Such implications are stored separately in another structure similar to the universal theory from the calculus discussed above.

This implicational theory is then used to compute what I call the **transitive implicational closure** of every introduced literal, which is essentially the set of all literals that follow from introducing the new literal by recursive application of the implicational theory until there are no new literals left to process. If a literal from this closure clashes with the established facts in the partial model, we can close the branch already at this stage. If, on the other hand, no clash between the closure and the partial model arises, we can add the whole closure to the set of pending formulae, often considerably enlarging the partial model and bypassing a lot of dead ends in the search space.

The second major optimization in the current version is the usage of **partial model checks** when formulae are distributed into the agendas and when partial models are modified by introducing new constants or literals.

My version of partial model checking is able to detect whether a formula can be guaranteed not to hold in any model which contains the current partial model as a substructure. The advantage is that model checking is a rather cheap operation in comparison to tableau building, which makes it worthwile to invest some time in detecting contradictions between the partial model of a branch and the formulae on its agendas.

With these optimizations, OletinMB is able to find models of most largish linguistic formulae as they tend to occur in RTE applications in less than a second. For practical purposes in linguistic applications, the performance of my model builder is thus comparable to that of MACE, though there are a few caveats concerning formulae with few models. OletinMB is for various reasons more of a model *builder* than a model *finder*. Further discussion of what this means can be found in Sections 7.3 and 7.5.

5.3 Guiding Exploration via Prioritized Identity Assumptions (PRIDAS)

The main hook for customizing the model builder's behaviour is that while backtracking on existential initialization decisions, it is possible to influence the order in which the different alternatives are evaluated.

The basic idea is that the backtracking point generated by the application of a δ-rule holds an ordered list of alternative initializations which are considered in linear order. The user has the possibility to influence this ordered list directly or more elegantly by means of a weighting function. For reasons of descriptivity, I will call this mechanism **PRioritized IDentity ASsumptions (PRIDAS)**. The only condition for the weighting function is that it must return values between 0 and 1 which represent a confidence measure for possible instantiations. The function can be computed by means of all the knowledge available at the active branch and in its formulae as well as arbitrary external sources of information. Initializations with higher confidence measures will then be considered earlier than initializations with lower confidence measures.

6 Constructing Non-Minimal Models

An important reason why the minimal model property is so popular in model builders is that this property makes it a lot easier to prove the formal properties of a model building algorithm. Moving away from model minimality thus always includes a risk of breaking some desired properties, such as completeness for finite satisfiability.

To avoid this, I employ a variant of a model building strategy that has already been proven complete. What I can influence without risking to break any formal property is the speed and the order in which the search space is traversed. I will show that even with these limited possibilities, there is a lot of potential for improving the quality of model building for linguistic purposes.

The main idea I am pursuing is that in linguistic model building, the advantages of constructing minimal models are far outweighed by the problems this choice causes. Chapter 7 gives justification for this opinion in the form of a few examples from the Nutcracker RTE system. In this chapter, I will first explain more thoroughly how model building can be steered by the PRIDAS mechanism, and motivate why I consider it a promising way of allowing the user to interact with the model building process.

6.1 Advantages of Externally Guided Exploration

Two sources of information promise to be of value for influencing search space traversal. The first is **process-internal data** that can be used to define more efficient heuristics in order to first explore areas of the search space with a higher probability. Such probabilites can for example be estimated via overlaps of the atomic facts in which the predicates occur. Intuitively, if we have two entities with similar properties, it makes more sense to particular case of the two entities being identical.

The second, much more variable source of information is **domain-specific knowledge**. Such knowledge can come in the form of special information about the predicate symbols of the logical language, which tend to have a lot of meaning that is not coded into the formula to be satisfied. Often, it is possible to state that e.g. if an entity has property P, it is not very likely to also have property Q, making separate instantiations the preferred choice when exploring the model space.

As we will see in Chapter 7, in a linguistic context such information can be used to reduce the amount of world knowledge axioms necessary to prevent identity assumptions between entities which can never be identical (such as places and events or rabbits and carrots). This bears potential for vast improvements to linguistic model building because axioms that enforce such constraints tend to inflate the formulae fed into the model builders, making it very hard to build larger models for efficiency reasons.

In essence, PRIDAS turns the major weakness of the analytic tableau method for model building into a strength: at the point where the algorithm has to non-deterministically choose the next alternative to be considered, we receive a hook to influence the behavior and help the process with educated guesses that can be built on external (e.g. linguistic) information.

6.2 Ranking Functions for Element Identity Assumptions

We will now have a more formal look at the PRIDAS mechanism. For clarity of presentation, I will use the following definition of PRIDAS weighting functions:

A function $\rho : C \cup \{c\} \rightarrow [0, 1]$ is a **PRIDAS function** iff C is the universe of a partial model, and c is a new constant symbol, i.e. $c \notin C$. As a notational convention, I will always give the same names to the elements of the universe: $C := \{c_1, ..., c_n\}$, and I will retain the name c for the newly introduced constant.

With these notations, we can now state backtracking behaviour by giving a more explicit version of the existential instantiation rules. To describe the backtracking process, I need to introduce an additional **set of instantiation choices** D to some branch representations:

$$\frac{< \Phi, \Psi, \Omega, C, A, \{\exists x(\psi(x))\} \cup \Pi >}{< \Phi, \Psi, \Omega, C, A, \{\exists x(\psi(x))\} \cup \Pi, C \cup c >} \; c \notin C$$

$$\frac{< \Phi, \Psi, \Omega, C, A, \{\exists x(\psi(x))\} \cup \Pi, D >}{< \Phi, \Psi, \Omega, C, A, \{\exists x(\psi(x))\} \cup \Pi, D \backslash \{\hat{c}\} >< \Phi, \Psi, \Omega, \{\hat{c}\} \cup C, A, \{\psi(\hat{c})\} \cup \Omega(\hat{c}) \cup \Pi >},$$

where $\hat{c} := \text{argmax}_{c \in D}\, \rho(c)$, and $\Omega(\hat{c}) := \begin{cases} \{\phi(\hat{c}) \mid \forall x(\phi(x)) \in \Omega\} & if\ \hat{c} \notin C \\ \emptyset & if\ \hat{c} \in C \end{cases}$

The right branch is then explored first, until a model is found or all resulting branches are closed. Only if the latter case happens (or if it is looking for multiple models), OletinMB will backtrack to explore the left branch.

Note that each time we backtrack, the set of instantiation choices D shrinks in the left branch. Finally, when all the possible instantiations were exhausted without finding a model, the whole branch fails:

$$\frac{< \Phi, \Psi, \Omega, C, A, \{\exists x(\psi(x))\} \cup \Pi, \emptyset >}{closed}$$

.

6.3 Ranking Functions for Producing Minimal Models

If we want to simulate the behaviour of model builders which always produce minimal models, we can impose a very simple condition on the ranking function to be used for that purpose.

For a PRIDAS function ρ_{min} to produce minimal models, it suffices to ensure that $\rho_{min}(c_k) > \rho_{min}(c)$ for $k = 1, ..., n$. The priority ordering of the other existential instantiations is of no importance for this property, leaving space for additional heuristics to make performance-improving choices.

Using such a ranking function where the introduction of a new constant always recieves the lowest confidence value, OletinMB will only introduce new elements to the universe if everything else failed. This results in a minimal model because the interpretation with the fewest introduced constant symbols and thus the smallest universe will be found first.

6.4 Ranking Functions for Producing Maximal Models

The second class of PRIDAS functions we shall consider follow the equally simple constraint $\rho_{max}(c_k) < \rho_{max}(c)$ for $k = 1, ..., n$. I call such functions ρ_{max} because of the symmetry in definitions to ρ_{min}.

The models produced by ρ_{max} are maximal in the sense that OletinMB is not able to produce any larger model. In formulae where no universal quantifier ranges over an existential quantifier, the size of these "maximal models" is bounded by (and will usually be equal to) the number of existential quantifiers in the formula.

The term **maximal model** is not properly defined otherwise, since many formulae possess models of arbitrary size. For this reason, ρ_{max} easily leads to non-termination, since we never identify any elements of the universe, but introduce a new skolem constant for each existential quantifier we meet. The

behaviour of OletinMB steered by ρ_{max} will in effect be identical to that of a classical analytical tableau, cf. Figure 3.

Even such a simple PRIDAS function can lead to considerable improvements in the adequateness of constructed models, as we shall see in the next chapter. However, such a function is also more than slightly dangerous because we have a severe risk of non-termination even on smallish formulae.

In a very straightforward fashion, it is possible to alleviate this danger by falling back to a non-maximal ρ if universal quantifiers range over existential quantifiers in the formula. This might sound like a quite crude approach, but the fact is that many or even most formulae as they turn up in computational semantics do not exhibit the problematic structure (cf. Figure 5 for a typical model builder input for the RTE task), so why should we renounce on the benefits of non-minimal model building if we can circumvent the inherent dangers by a simple superficial check of the formula we want to process?

For my first practical experiments, I have chosen to stick to this simple variant. A huge variety of more elegant and theoretically appealing solutions are possible by using more sophisticated PRIDAS functions, as we shall see now.

6.5 Towards Linguistically Motivated Ranking Functions

In some sense, ρ_{min} and ρ_{max} represent the two extremes of how we can influence the model building process using PRIDAS functions. More interesting functions can be computed by using some sort of external knowledge, and such functions will end up somewhere in the continuum between ρ_{min} and ρ_{max}.

It is possible to imagine all kinds of sophisticated PRIDAS functions for special applications, but I will limit myself here to giving a few ideas on what kind of functions could be useful for computational semantics. The crucial question to ask is how we can know as early as possible whether two elements of a partial model are likely to be identical.

Assume we have an active branch $\mathcal{B}$ with partial universe C and a set of established ground literals A, and we are about to decide how to instantiate some existential formula $\exists x\phi(x)$.

On the basis that entities with similar properties are more likely to be identical, an obvious idea would be to measure the **overlap** of the properties of x in the existential formula with the properties of each single element from the partial universe. We can define the set of **properties of an entity c at the current branch** $\mathcal{B}$ as $Prp(c, \mathcal{B}) := \{(\phi, i) \mid \phi(c_1, ..., c_{i-1}, c, c_{i+1}, ..., c_n) \in A\}$, collecting pairs of predicate names and argument positions.

Extracting a meaningful set of properties $Prp(c, \phi)$ for an instantiation of $\exists x\phi(x)$ is more difficult because at that point we have not yet split the existential formula into literals. Naively, we could substitute each candidate constant c for x and then use the literal subformulae of $\phi(c)$ as the basis for property extraction.

This obviously leads to problems, because we thereby basically assume that $\phi(c)$ has the form of a conjunction of literals, which is a bad assumption for the general case. If, however, we take a look at the structure of a typical first-order theory from computational semantics (see e.g. Figure 5), it becomes apparent that extracting literal subformulae could still be a good heuristics.

To achieve a more general solution, it is possible to extract a set (but not all) of the literals guaranteed to hold in a formula by means of a **polarity automaton**. Thoroughly introducing this concept is beyond the scope of this work, I therefore postpone its discussion to future work on PRIDAS functions.

Whichever way we compute $Prp(c, \mathcal{B})$ and $Prp(c, \phi)$, we now have the means to define a new PRIDAS function $\rho_{overlap}(c) := \frac{Prp(c,\mathcal{B}) \cap Prp(c,\phi)}{Prp(c,\mathcal{B}) \cup Prp(c,\phi)}$ to rank the different possible existential instantiations in our situation.

Until now, we have defined property overlap on the grounds of predicate name identitites. However, our predicates have a linguistic meaning, and we ideally would want $\rho_{overlap}$ to recognize that two entities c_1 and c_2 with $Prp(c_1, \mathcal{B}) = \{(red, 1), (house, 1)\}$ and $Prp(c_2, \mathcal{B}) = \{(crimson, 1), (building, 1)\}$ are better candidates for an identity assumption than, say, two entities c_3 and c_4 with $Prp(c_3, \mathcal{B}) = \{(little, 1), (girl, 1)\}$ and $Prp(c_4, \mathcal{B}) = \{(old, 1), (man, 1)\}$.

This is possible by enriching the computation of predicate name similarity by an external source of lexical semantic information. Measures of conceptual relatedness are a common tool in shallow semantics, and readily available toolkits such as **WordNet::Similarity** by Pedersen et al. (2004) offer implementations of a range of different methods for computing WordNet-based similarity measures.

For now, I will formalize such methods as a function $sim : \mathcal{P} \times \mathcal{P} \rightarrow [0, 1]$, where $\mathcal{P}$ is the set of predicate symbols in our signature. Having a continuum of values instead of just a binary distinction for predicate similarity requires a more involved definition of a PRIDAS function based on lexical similarity:

$$\rho_{sim}(c) := \prod_{(\psi,i) \in Prp(c,\phi)} \max_{(\chi,i) \in Prp(c,\mathcal{B})} sim(\psi, \chi)$$

Approaches to linguistically guided PRIDAS computation need not be based on predicate similarity only. An example of an additional factor that could contribute to a promising heuristics is that in linguistic applications, it does not make much sense to assume identity of two arguments of a predicate.

To see the rationale behind this, consider that a sentence like "a woman loves a woman", as a formula $\exists x \exists y(woman(x) \wedge woman(y) \wedge love(x, y)))$, is never interpreted to mean $woman(c) \wedge love(c, c)$ or "a woman loves herself". A good computational semantics system would already have translated that sentence into $\exists x(woman(x) \wedge love(x, x))$, only leaving open the desired interpretation. It therefore makes sense to punish identity assumptions between two entities that occur as arguments of the same predicate.

Numerous other such phenomena could be observed and made use of for lin-

guistically guided model building, and we have only just started to explore the possibilities. Future research will aim at finding many more such heuristics.

As a last idea, I would like to briefly mention what I call **dynamic PRIDAS functions**. Such functions would make use of the backtracking mechanism, thereby going beyond exploiting only what is available at the current branch. For the computation of $Prp(c, \phi)$, dynamic PRIDAS functions could for example take into account literals that were produced during previously tried instantiations. This would help to enrich the decision base for $\rho_{overlap}$ and ρ_{sim}, helping OletinMB to make even more informed instantiation decisions.

7 Evaluation within the Nutcracker System

7.1 The Problem of Evaluating Model Builders

Assessing the performance of algorithms or systems that cannot provide a solution in all cases is always a difficult task because we cannot rely on classical analysis techniques. Non-termination in the worst case has to be a common feature of all model builders, which makes it impossible to compare algorithms in terms of their worst-case complexity. The best we can do formally is to prove completeness and estimate runtimes for certain classes of formulae, but this tends to be a tedious task, and there are often no comparable results for other model builders about the same classes of formulae.

If we try to rely on benchmarking for practical evaluation, the problem is that unlike for theorem proving, we will not even find standardized sets of test cases. Until such test sets exist (and for our purposes, we would specifically need one containing examples from computational semantics), we have to rely on qualitative comparison on a few test cases. The **Nutcracker RTE system** by Bos & Markert (2005a) turned out to be a good source for such example cases.

7.2 Overview of the Nutcracker System

The Nutcracker system was one of the earliest submissions to the RTE challenge. In this competition, systems are evaluated according to how well they can predict textual entailment on a test set of text-hypothesis pairs.

Unlike most other RTE systems, Nutcracker does not rely only on shallow heuristics, but tries to perform proper logical inference on formal semantic representations. To make this possible, the system relies on a rather long toolchain consisting of a wide-coverage CCG parser, a word sense disambiguation module, the Boxer system for DRS generation, a DRS-to-FOL converter, and a WordNet-based world knowledge database.

If it were possible to reliably translate natural language sentences into appropriate first-order representations, it would be possible to decide entailment by simply applying automated theorem proving to an implication $T \rightarrow H$ with the **text representation** T as the antecedent and the **hypothesis representation** H as the consequent.

While this worked in a few example cases, for most text-hypothesis pairs the first-order representations produced by the Nutcracker toolchain were not accurate enough to reliably do that. As a fallback mechanism, Bos & Markert developed the idea of generating models for T and of $T \wedge H$ and comparing their sizes, with the rationale that a large difference in model size between models of T and of $T \wedge H$ means that information was added by the hypothesis, making it unlikely that T entails H.

The size ratios between the models are then used together with other, more shallow parameters to decide on the entailment prediction. Nutcracker was competitive with other RTE systems in the challenge, but unfortunately it turned out that the shallow parameters alone could account for that result, and the effects of adding the deep method were not significant.

On the other hand, in their error analyis, Bos & Markert (2006) found out that model size comparison alone worked almost as good with and without the shallow parameters. This means that, quite surprisingly, the two very different approaches can decide roughly the same class of problems, without complementing each other much. Bos & Markert explain the disappointing performance of the deep approach with a general lack of appropriate background knowledge, the traditional limiting factor in computational semantics.

Unlike other RTE systems, Nutcracker is freely available with source code for non-commercial purposes[2], making it ideal for building on an existing infrastructure and conducting experiments with advanced techniques without having to build up the toolchain by oneself. The internals of the system are not extremely well-documented, but the architecture is nicely modular and allows for significant modifications without having to understand in detail all the various components of the system.

7.3 The Structure of Linguistic Formulae and Useful Specialized Techniques

If we have a look at typical first-order representations generated by Nutcracker (see Figures 5 and 8), it becomes apparent that these formulae have some interesting syntactic and semantic properties. The most obvious syntactic constraints are that we do not have any function symbols, and that we see existential and universal quantifiers, but almost never nested quantification of different types.

The first observation means that a model builder specialized in linguistic formulae need not support function symbols, which makes existential instantiation a lot easier to implement. The second observation removes most risks of nontermination, allowing us to use otherwise dangerous heuristics such as depth-first search of the model space or ρ_{max} to guide existential instantiations. It also implies that we are usually dealing with finitely satisfiable formulae.

[2] at http://svn.ask.it.usyd.edu.au/trac/candc/wiki/nutcracker

On the semantic side, we notice that the set of possible models for a formula is usually huge because the formulae are not very restrictive and do not contain much negation. We can thus find valid models in all corners of the search space, the problem is to find a meaningful and linguistically appropriate one.

Put differently, model building on such formulae can often be achieved by a rather direct traversal of the search space up to some leaf. To establish a concise terminology, I propose to call OletinMB and similar systems with such a behaviour **model builders** in a stricter sense. Model construction systems optimized for restrictive axioms I will instead call **model finders**.

It should be mentioned, however, that it is easily possible to express any first-order formula in natural language. This includes axiomatization of infinite structures like the natural numbers, which may serve as a warning that the structural properties we just saw are by no means universal. They merely hold for almost all formulae Nutcracker produces on typical sentences from the RTE task.

To the formal semanticist, it might sound quite curious that one of the most-discussed problems of her trade, ambiguities arising from nested quantification, rarely play a role in real-life data. But the most urgent problems of semantic inference on real-life language data lie in other areas, as we shall see now.

7.4 Comparing OletinMB and MACE

As its standard model builders, the Nutcracker system uses MACE or another MACE-style model builder named Paradox. Integrating an additional model builder which can behave exactly like MACE into the system turned out to be rather straightforward. This allowed me to quickly create an environment for testing OletinMB's performance and model quality on real-life examples.

Concerning performance, OletinMB can compete with MACE on almost all formulae from my test set. Given that MACE is a much more mature system, I attribute this to the fact that OletinMB is specifically tailored towards typical formulae from computational semantics, while MACE was originally developed for mathematical problems and will probably perform a lot better on the kind of problems it was optimized for.

The following section will assess the quality of models produced by OletinMB and MACE. Specifically, I will argue why I consider OletinMB's PRIDAS mechanism to be more useful for linguistic model building than the minimal model approach that comes with MACE and most other model builders.

7.4.1 Improved Model Quality by Non-Minimality

In model building, we tend to be interested in the smallest structure described by a formula, because we are not interested in superfluous structure that would not be necessary to satisfy it. Therefore, it generally makes sense to produce minimal models because this way we never assume too much structure, not adding any information that is not in the logical specification.

$\exists x_1 \exists x_2 (per1oracle(x_1) \wedge n1form(x_2) \wedge$
$\exists x_3 \exists x_4 (v1fight(x_3) \wedge r1agent(x_3, x_1) \wedge r1theme(x_3, x_4) \wedge n1proposition(x_4) \wedge$
$\exists x_5 \exists x_6 \exists x_7 (v1keep(x_5) \wedge r1agent(x_5, x_1) \wedge r1patient(x_5, x_2) \wedge r1from(x_5, x_6) \wedge$
$v1release(x_7) \wedge r1patient(x_7, x_6) \wedge n1event(x_7) \wedge n1event(x_5)) \wedge n1event(x_3)))$
$\forall x_8 (n1abstract\_entity(x_8) \rightarrow n1entity(x_8))$
$\forall x_9 (v1release(x_9) \rightarrow n1event(x_9))$
$\forall x_{10} (v1keep(x_{10}) \rightarrow n1event(x_{10}))$
$\forall x_{11} (v1fight(x_{11}) \rightarrow n1event(x_{11}))$
$\forall x_{12} (n1event(x_{12}) \rightarrow n1abstract\_entity(x_{12}))$
$\forall x_{13} (n1form(x_{13}) \rightarrow n1abstract\_entity(x_{13}))$
$\forall x_{14} (per1oracle(x_{14}) \rightarrow n1entity(x_{14}))$
$\forall x_{15} (n1proposition(x_{15}) \rightarrow n1abstract\_entity(x_{15}))$
$\forall x_{16} (v1release(x_{16}) \rightarrow \neg v1keep(x_{16}))$
$\forall x_{17} (v1release(x_{17}) \rightarrow \neg v1fight(x_{17}))$
$\forall x_{18} (v1keep(x_{18}) \rightarrow \neg v1fight(x_{18}))$
$\forall x_{19} (n1event(x_{19}) \rightarrow \neg n1form(x_{19}))$
$\forall x_{20} (n1event(x_{20}) \rightarrow \neg n1proposition(x_{20}))$
$\forall x_{21} (per1oracle(x_{21}) \rightarrow \neg n1abstract\_entity(x_{21}))$
$\forall x_{22} (n1proposition(x_{22}) \rightarrow \neg n1form(x_{22}))$

Figure 5: First-order theory produced by Nutcracker for (1)

However, in computational semantics we have other and possibly more relevant criteria for judging the quality of models than their minimality. We have a sense of which models are adequate for a given sentence and which are not, and this judgement is based on linguistic meaning. On these grounds, we dislike models where a rabbit is identified with its own tail or with the carrot it eats. Yet we are still using model builders whose default strategy is to assume that such identities hold, and such identities will turn up in minimal models if we do not explicitly exclude them by introducing world knowledge axioms, whose sole contribution often is to state which elements may not be identified. So the assumptions used in minimal model building are in fact rather strong, and we are at pains to amend the effects of these assumptions.

The problem is that by identifying elements, minimal models actually make some very strong assumptions about the structure of the world described. By default, building minimal models means to assume that all elements are identical, and only to accept more elements if this fails in all cases.

To see in a real-world example why these default identity assumptions are harmful, consider the following sentence from the RTE 1 test corpus:

(1) Oracle had fought to keep the forms from being released.

The logical theory produced by Nutcracker when processing this sentence is shown in Figure 5. When feeding this theory into OletinMB and comparing the result of applying ρ_{min}, which simulates the behavior of a minimal model builder and is identical to MACE's result, to the result of applying ρ_{max}, we see that the results are different. Both results are inadequate in many ways, but this is mainly due to the inadequate logical theory, which fails to capture a

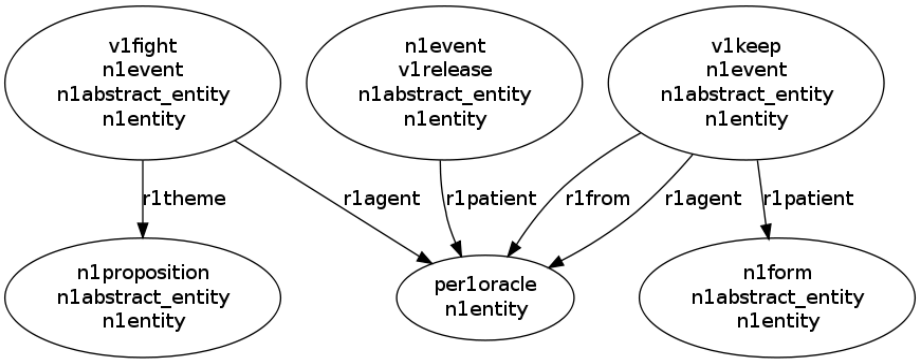

Figure 6: Model for (1) using ρ_{min}; identical to the one produced by MACE

lot of relevant facts. These include that what Oracle is fighting against is the event of the forms being released, and that there is another unmentioned entity which is supposed to be kept from releasing the forms, and that is is not the forms that would have to be kept from doing something.

I have chosen not to use one of the few examples with adequate formulae, also in order to give an impression of the state of the technology, where virtually all formulae produced exhibit major weaknesses. With the given theory, we have no hopes of constructing a completely adequate model. What we can do, however, is to determine which of the models is less inadequate.

We shall first have a look at the result of minimal model building in Figure 6. Quite curiously, the patient of the releasing is identified with Oracle here, such that the model represents the situation where Oracle fights for something, and Oracle keeps the forms from Oracle while being released by somebody.

So where does this wrong identity assumption come from? It is a result of the default identity assumption of minimal model building along with the fact that nothing in the theory excludes this identity. A minimal model builder will identify elements whenever it gets the chance, whereas a linguistically adequate one would need some justification for assuming identity.

The non-minimal model shown in Figure 7 contains one additional element without properties. This element actually corresponds to something which is released. The whole model could be read to describe a situation where Oracle fights for something, and Oracle keeps the forms from (being) something which is released by somebody. We see that this model is arguably the best approximation to the intended meaning that is possible given the problems of the theory it satisfies.

Using ρ_{max} as our PRIDAS function turns out to work remarkably well in this and many other cases. From this we can conclude that non-identity seems to be a much more sensible default assumption for linguistic model building than identity. However, ρ_{max} certainly will not work in many other cases, especially

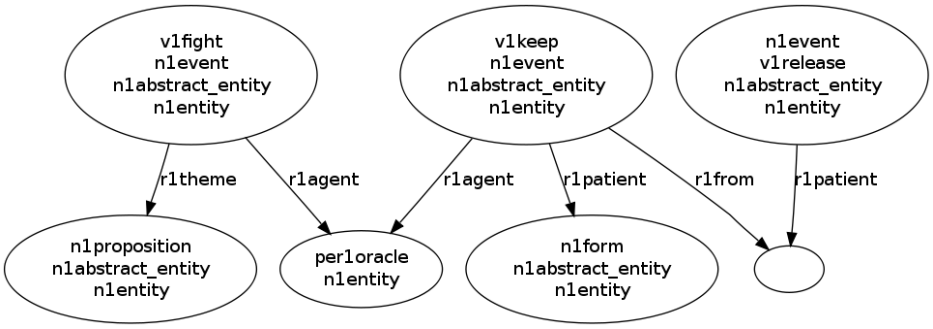

Figure 7: Model for (1) using ρ_{max}; note the additional entity without properties

when we are trying to build semantic models for entire discourses.

That simply assuming all introduced entities to be separate will not work becomes obvious when we consider ubiquitous semantic phenomena such as anaphoras. In a discourse like "The dog chases the cat. The cat flees", maximal model building over the conjunction of the logical interpretations of the two sentences would lead to models containing two cats.

This is why in a good model builder, non-identity would still be the default assumption, but the PRIDAS function employed should be able to rank identity assumptions highly if there are good linguistic reasons. Such reasons could include lexical overlap, agreement e.g. for anaphoric pronouns, or, as in the example case, the definite article, which could be treated by ranking the introduction of a new element so low that it happens only after all the other existential instantiations have failed.

Ranking functions which take such phenomena into consideration could be based on the ideas from Section 6.5. Testing such functions on a large number of cases is a good starting point for future development of linguistic model building.

7.4.2 A Shift in the Importance of World Knowledge

The main reason why world knowledge is needed so urgently in linguistic model building is that it provides us with the axioms necessary to get around the effects of minimal model building. Choosing a PRIDAS function which produces maximal or near-maximal models works so well in many cases because we are overriding the default identity assumption.

An interesting observation is that by using non-identity as the default assumption during model building, we can often recieve acceptable models even without world knowledge. To see this in a typical example, consider the following sentence from the RTE 1 testset:

(2) Singapore scientists reveal that SARS virus has undergone genetic changes.

This sentence is converted to the formula in Figure 8, and the world knowledge

$\exists x_1 \exists x_2 \exists x_3 \exists x_4 (org1singapore(x_1) \wedge r1nn(x_1, x_2) \wedge n1scientist(x_2) \wedge v1reveal(x_3) \wedge r1agent(x_3, x_2) \wedge r1theme(x_3, x_4) \wedge n1proposition(x_4) \wedge \exists x_5 \exists x_6 \exists x_7 \exists x_8 (n1sar(x_6) \wedge r1nn(x_6, x_5) \wedge n1virus(x_5) \wedge a1genetic(x_7) \wedge n1change(x_7) \wedge v1undergo(x_8) \wedge r1agent(x_8, x_5) \wedge r1patient(x_8, x_7) \wedge n1event(x_8)) \wedge n1event(x_3))$

Figure 8: First-order formula produced by Nutcracker for (2)

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

Figure 9: World knowledge axioms produced by Nutcracker for (2)

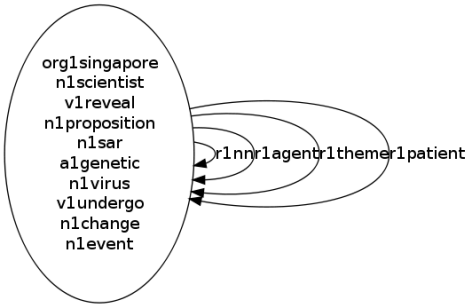

Figure 10: Model for (2) using ρ_{min} without world knowledge

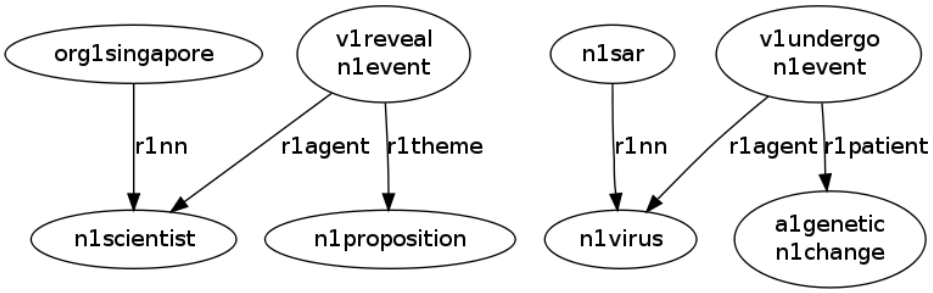

Figure 11: Model for (2) using ρ_{max} without world knowledge

axioms Nutcracker generates for it are listed in Figure 9. We first let MACE or, equivalently, OletinMB with ρ_{min}, work on the formula without adding the world knowledge axioms. The result in Figure 10 illustrates that producing minimal models in linguistic model building is useless without world knowledge: all the elements are assumed to identical, because there is nothing in the formula which would prevent these assumptions.

If instead, we use ρ_{max} within OletinMB, we do not need any world knowledge to get the quite reasonable model in Figure 11. As a matter of fact, the resulting model is essentially the same as the one we get if we include the world knowledge axioms (cf. Figure 12). Essentially the same effect can be observed for virtually all Nutcracker formulae, making world knowledge much less of a necessity for OletinMB than for MACE-style model builders.

This has important implications for the use of world knowledge in model building. I have shown that by using non-minimal PRIDAS functions, the main reason for generating and including world knowledge axioms can be done away with. This means that for many sentences, the logical theories needed to produce acceptable models can become a lot smaller, opening up possibilities for the treatment of more complex sentences whose formal interpretations were intractably huge before.

On the other hand, this is not meant to imply that world knowledge were of no use any longer. It only means that the traditional role of world knowledge as

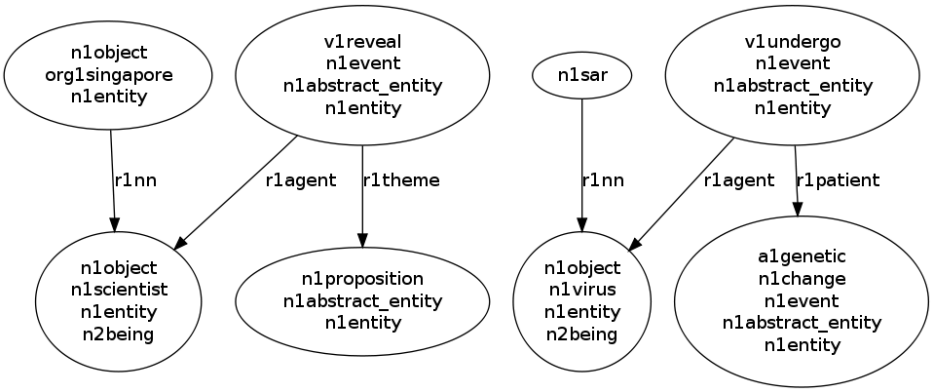

Figure 12: Model for (2) using ρ_{min} or ρ_{max} with world knowledge

a source for simple axioms which prevent unwanted identity assumptions is not as crucial as before. In order to exploit the world knowledge, I propose to put it to good use in computing more sophisticated PRIDAS functions.

Instead of preventing a huge number of useless identity assumptions, world knowledge would then play a role in justifying only a few useful identity assumptions. From the perspective of the model builder, this would mean a much less costly guidance of model space exploration. Instead of inflating its universal theory and forcing it to generate a huge number of instantiations that have to be checked for clashes, world knowledge would only come in at the real decision points, nudging the model builder into the right direction by prioritizing its instantiation choices.

For future work, OletinMB will serve as a very flexible testing platform for such highly specialized PRIDAS functions. My first experiments have shown that this approach has some potential for improving linguistic model building, and future experiments will aim at finding, implementing and evaluating advanced PRIDAS functions for this purpose.

7.5 Problems and Possible Solutions

The experimental nature of OletinMB results in certain shortcomings in comparison to more mature model building systems. The most important problem is a steep decline in speed for largish domain sizes on certain classes of formulae. Given the complexity of the satisfiability problem, this is a problem all model builders face at some point. OletinMB, however, has turned out to run into this kind of problem a little more often than MACE.

On my machine, both model builders behave rather well for theories requiring a minimal domain size of 18 or less. Beyond that, MACE runs into a combinatorical explosion, while OletinMB is usually not overly sensitive to domain size. Where differences in performance occur, they can instead be explained by properties of the space of possible models.

The problem is that by its constructive and little exploratory nature, OletinMB has trouble finding models in the vast but sparse search spaces of very restrictive formulae. OletinMB was developed under the assumption that such problems do not occur very often in linguistic applications. This is a valid assumption as long as we deal with computing the meanings of coherent discourses.

However, Nutcracker systematically requires models for such restrictive formulae when it tries to directly prove $T \rightarrow H$. To improve termination behavior, Nutcracker always searches for a proof and a counter-example in parallel. Finding a counter example to $T \rightarrow H$ is equivalent to building a model of $T \wedge \neg H$. If T and H are semantically close to each other (as is generally the case), this formula is highly contradictive and possesses a rather sparse model space with many dead ends.

The model space structure of such a formula is more akin to that of a mathematical problem with restrictive axioms. It comes as no surprise that OletinMB, which was designed to be more of a model builder than a model finder, has trouble in treating such problems.

For practical purposes, this means that OletinMB is not yet ready to completely replace MACE in a system like Nutcracker. As a preliminary solution, I combine the advantages of MACE and OletinMB by using MACE as the model builder to work in parallel with a theorem prover, and OletinMB for constructing linguistically more adequate models to be used in model size comparison.

However, I am confident that it will be possible to improve Oletin's behaviour as a model finder in future versions. The possibilities for optimization are far from exhausted, and it will be possible to integrate a lot more advanced constraint-based techniques developed for other tableau-based methods.

Another difficulty more on the conceptual side is the construction of meaningful models for negated sentences. By default, a logical representation such as $\neg\exists x(woman(x) \wedge sleep(x))$ for "a woman does not sleep" will instantly result in an empty model, which is logically sound, but not very helpful in a linguistic context. The empty model for negated sentences also breaks Nutcracker's model-size based decision procedure if negation is envolved in either H or T. Enumerative methods have a real advantage here because they operate on given model sizes and do not have to infer on the fly how many entities are needed.

The problem can partly be attributed to the logical representation. The problematic formula could be rephrased as "it is not the case that a woman sleeps", which indeed tells us little to nothing about the structure of the world, making the empty model a good solution. A better logical presentation for the sentence would instead be $\exists x(woman(x) \wedge \neg sleep(x))$, for which OletinMB would build the desired model.

However, referring the problem back to the writer of the transformation rules is a little too easy. Determining the scopus of negations is one of the harder problems in translating sentences into formulae, and we simply cannot expect this to work for all cases. One way would be to simulate the behaviour of an

enumerative model builder by forcing OletinMB to start at a given domain size. This can easily be achieved by feeding the initial branch with constant symbols.

A perhaps more elegant solution would be to devise something like a presupposition handling on the formula level. Extracting literals from formulae as in Section 6.5 could prove to be a worthwile technique in this context as well.

8 Conclusion and Outlook

In this paper, we have seen that off-the-shelf model builders are not the ideal tools for model building in natural language semantics. The main problem is the underlying assumption caused by mathematical applications that minimal models are the models the user is most interested in.

As we have seen, minimal models are almost never linguistically adequate, the purpose of world knowledge in current systems is largely to get around the negative effects of minimal model building.

We have seen it might be worthwile to implement specialized model builders for computational semantics, and we explored various ideas of how world knowledge could be used more efficiently during the process.

With OletinMB, I have developed and implemented a flexible system for larger experiments in non-minimal model building. The central decision mechanism within OletinMB are freely prioritizable identity assumptions.

Future work includes enhacing OletinMB's model finding capabilities by further optimization and implementing some of the ideas on more complex PRIDAS functions, especially different variants using lexical semantic information.

The next major step will then be to find out whether the improvements in model structure are good enough to significantly improve the results achieved by Nutcracker for the RTE task so far.

References

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 & K. Markert (2005a). 'Combining Shallow and Deep NLP Methods for Recognizing Textual Entailment'. In *Proceedings of the PASCAL Challenges Workshop*, pp. 65–68.

J. Bos & K. Markert (2005b). 'Recognising textual entailment with logical inference'. In *Proceedings of Human Language Technology Conference and Confer-*

ence on Empirical Methods in Natural Language Processing (HLT/EMNLP), pp. 628–635.

J. Bos & K. Markert (2006). 'When logical inference helps determining textual entailment (and when it doesn't)'. In *Proceedings of the Second PASCAL Challenges Workshop on Recognising Textual Entailment*, pp. 98–103.

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.

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

K. Claessen & N. Sörensson (2003). 'New Techniques that Improve MACE-style Finite Model Finding'. In *Proceedings of the CADE-19 Workshop: Model Computation - Principles, Algorithms, Applications*.

L. He (2001). 'I-SATCHMO: An Improvement of SATCHMO'. *Journal of Automated Reasoning* **27**(3):313–322.

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.

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.

W. McCune (1994). 'A Davis-Putnam Program and its Application to Finite First-Order Model Search: Quasigroup Existence Problems'. Tech. rep.

T. Pedersen, et al. (2004). 'WordNet::Similarity - Measuring the Relatedness of Concepts'. In D. M. Susan Dumais & S. Roukos (eds.), *HLT-NAACL 2004: Demonstration Papers*, pp. 38–41, Boston, Massachusetts, USA. Association for Computational Linguistics.

J. Slaney (1992). 'FINDER – Finder Finite Domain Enumerator – Notes And Guide'. Tech. rep.

J. Zhang & H. Zhang (1995). 'SEM: a system for enumerating models'. In *Proceedings of the 14th international joint conference on Artificial intelligence - Volume 1*, pp. 298–303, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.