Lecture Notes in Artificial Intelligence 2953

Edited by J. G. Carbonell and J. Siekmann

Subseries of Lecture Notes in Computer Science

3

Berlin
Heidelberg
New York
Hong Kong
London
Milan
Paris
Tokyo

Karsten Konrad

Model Generation for Natural Language Interpretation and Analysis

1 3

Series Editors

Jaime G. Carbonell, Carnegie Mellon University, Pittsburgh, PA, USA
Jörg Siekmann, University of Saarland, Saarbrücken, Germany

Author

Karsten Konrad
XtraMind Technologies GmbH
Stuhlsatzenhausweg 3, 66123 Saarbrücken, Germany
E-mail: konrad@xtramind.com

Cataloging-in-Publication Data applied for

A catalog record for this book is available from the Library of Congress.

Bibliographic information published by Die Deutsche Bibliothek
Die Deutsche Bibliothek lists this publication in the Deutsche Nationalbibliografie; detailed bibliographic data is available in the Internet at <http://dnb.ddb.de>.

CR Subject Classification (1998): I.2.7, I.2, F.4.1, I.7, J.5

ISSN 0302-9743
ISBN 3-540-21069-5 Springer-Verlag Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Olgun Computergrafik
Printed on acid-free paper SPIN: 10988374 06/3142 5 4 3 2 1 0

Foreword

This monograph is a revised version of Karsten Konrad's doctoral dissertation. It focuses on a topic of rapidly increasing importance in computer science: the development of inference tools tailored to applications in Natural Language Processing. Technology in mathematical theorem proving has undergone an impressive development during the last two decades, resulting in a variety of attractive systems for applications in mathematical deduction and knowledge processing. Natural Language Processing has become a theme of outstanding relevance in information technology, mainly due to the explosive growth of the World-Wide Web, where by far the largest part of the information is encoded in natural language documents. The book appears at a pivotal moment, when much attention is being paid to the task of adding a semantic layer to the Web, and representation and processing of NL-based semantic information pops up as the primary requirement for further technological progress.

Konrad's book argues for the Model generation paradigm as a framework, which supports specific tasks of natural language interpretation and NL-based inference in a natural way. It presents extensions in several respects: restricted techniques of model generation for higher-order logics, which are useful for the construction of semantic representations, as well as refined methods for minimal model generation. The latter support the use of world knowledge in natural language inference and realize a concept of preferential reasoning. The proposed method is similar in its results to NL applications of weighted abduction, but it obtains these results in a more systematic and transparent way.

Konrad applies his variant of model generation to selected topics in NL semantics: the well-known problem of reference resolution for anaphoric definite noun phrases, and the selection of readings for reciprocal pronouns. His method provides natural and general solutions for a variety of phenomena, which hitherto had to be treated by enumeration of variants. The inference problems are delegated to the Kimba model generator, an innovative implementation of the inference method in the constraint programming framework.

Certainly the book does not offer final solutions, but it opens a perspective on a fascinating and highly relevant field of future research, and it offers tools to start out with. The book is nicely written, it provides motivation and background to readers who are unfamiliar with part of this interdisciplinary research area. I strongly recommend it to readers both from the NLP and the deduction communities.

Manfred Pinkal

Table of Contents

1 Motivation

To understand a proposition means to know what is the case if it is true.
(Wittgenstein, *Tractatus*)

1.1 The Subject of This Volume

Computational semantics describes and determines the meaning of human language using computational tools. The research reported in this thesis investigates the use of a new such tool in the interpretation and analysis of logic-based semantic representations: model generation.

Model generation refers to the automatic construction of interpretations that satisfy logical specifications. Model generation is a successful area in automated theorem proving, and systems for generating models have found various applications outside of linguistics. However, model generation has, until recently, not been a topic of research in the computational semantics community. While conventional automated theorem provers enjoy some interest as inference modules in natural-language processing, the potential of systems that prove satisfiability by generating models is still widely unknown. Two reasons for this can be identified.

First, model generation is a surprisingly small research area. In the forty years that automated theorem proving has been investigated, work has concentrated on deductive methods that prove unsatisfiability and entailment in classical first-order logics. These methods have been developed originally for proving mathematical theorems without human interaction. In this context, model generation has been considered only as a specialised tool for computing counterexamples of dubious theorems, or for proving the existence of certain small structures in finite mathematics. Hence, a linguist who wants to apply some techniques developed in automated theorem proving is likely to find much more material on first-order deduction than on model generation.

Second, natural-language semantics concentrates on representation rather than on inference. With the explicit goal of developing formal representations that are suitable for language-oriented inference, semanticists define and investigate formal representation languages for the meaning of human language. However, semanticists almost never implement inference systems for these languages. Only recently, there has been a growing interest in using the well-developed inference techniques from automated theorem proving for building natural-language systems. Still, only a small group of semanticists actually work on inference in semantics, and there is too effort for exploring new techniques that have been developed elsewhere.

This thesis aims at presenting model generation to a larger audience of researchers who are interested in logic-oriented computational semantics. The subject of this thesis is the use of automatically generated models in the interpretation and analysis of semantic representations, and the computational methods that are needed for generating such models.

1.2 Interpretation, Analysis, Computation

One of the principal goals of modern linguistic semantics, as initiated by Frege and Russell, has been to develop a computational theory of the conditions under which a statement can be uttered truthfully. Behind this interest lies the insight that the meaning of a statement uttered in a situational context can often be captured completely by such. The correspondence theory of meaning and truth is not a universal one because utterances have a wide range of purposes that cannot be conceived by knowing their truth value alone. Nevertheless, many modern semantic theories determine the meaning of natural-language utterances by a relationship between linguistic form and the truth conditions of some formal semantic representations.

The relation of form and meaning can be given as a translation from syntactical structures into semantic representations that preserves truth and entailment with regard to the original utterance. In semantic formalisms such as Discourse Representation Theory (DRT) by Kamp and Reyle [1], the representations computed from natural-language sentences are logical formulas. The process that determines the meaning of such formulas is called interpretation.

1.2.1 Interpretation

Since Tarski formulated the first mathematically sound truth-conditional semantic for a logic, interpretations are conceived as recursively defined mathematical structures that assign truth values to the well-formed formulas of a logical language. Interpreting a formula in a logic simply means to apply such an interpretation.

In logic-based natural-language semantics, interpretation is understood as a related, but different concept. Here, interpretation refers to processes that compute the truth conditions for a natural-language sentence L uttered in a given context consisting of situational and world knowledge.

More formally, interpreting a semantic representation ϕ for L within in context Γ means to compute facts Δ that must be assumed to be true if ϕ is true in Γ. The assumptions Δ that we look for formulate "explanations" why L is true in the situation described by Γ. Computing semantic interpretations is an inference process that identifies suitable sets of facts Δ for ϕ and Γ such that the following condition holds:

$$\Gamma \cup \Delta \models \phi$$

Suppose that there exists a logical interpretation $\mathcal{M}$ such that $\mathcal{M}(\Gamma \cup \{\phi\})$ is true, i.e., that there is a model $\mathcal{M}$ that satisfies both Γ and the representation ϕ. If we could represent $\mathcal{M}$ as a set of facts $\Delta_{\mathcal{M}}$, we could prove immediately that the following condition holds:

$$\Gamma \cup \Delta_{\mathcal{M}} \models \phi$$

This implies that one can use models $\mathcal{M}$ as semantic interpretations if there is a way to represent $\mathcal{M}$ within the logical language as a set of facts $\Delta_{\mathcal{M}}$. For an important class of models, the Herbrand models, this is indeed possible. A Herbrand model can be represented unambiguously as a set of ground literals. Furthermore, every first-order specification $\Gamma \cup \{\phi\}$ that is satisfiable at all must also be satisfiable by at least one Herbrand model.

Unfortunately, not every Herbrand model $\mathcal{M}$ of $\Gamma \cup \{\phi\}$ is suitable as an explanation for the truth of a semantic representation ϕ in context Γ. There are, in general, many models that interpret a logical specification $\Gamma \cup \{\phi\}$ to be true even though they do not have a correspondence to the truth conditions that we look for in a semantic interpretation.

What we actually want to perform here is abduction, i.e., that kind of inference that finds the best explanation. One of the topics of this thesis is to determine the relationship between abduction and the generation of models in the interpretation of semantic representations.

1.2.2 Analysis

Semantic analysis identifies the valid semantic representations within a set of possible readings by common-sense reasoning. For this, too, our models $\mathcal{M}$ for ϕ and Γ are interesting. In fact, the existence of a model $\mathcal{M}$ for a specification is a valuable information on its own. If a semantic representation ϕ is inconsistent within a context Γ, then $\Gamma \cup \{\phi\}$ will have no model at all. "Be consistent!" is one of the most important constraints for human communication and most discourses obey this maxim. If a semantic representation has no model within its situational context, we can assume that either the representation itself is faulty, or there is a discourse anomaly. Consider for instance the discourse *Katja's husband knows her sister. She is not married.* A natural-language parser might provisionally propose two different resolutions for the pronoun *She*:

(1.1) ***Katja***$_1$*'s husband knows her sister.* ***She***$_1$ *is not married.*

(1.2) *Katja's husband knows her* ***sister***$_1$. ***She***$_1$ *is not married.*

Only the second reading makes sense since the first reading leads to an inconsistency—we know that Katja is married because this information was implicitly given. A parser which has no access to common-sense reasoning will not be able to distinguish a valid from an inconsistent reading.

Model generation is useful for distinguishing valid and invalid semantic representations. Given a sentence L, a listener will in general have an intuition for the assumptions Δ that must hold if L is to be true in a context Γ. A computational tool that enumerates sets of assumptions Δ' for the semantic representations ϕ of L can be used to investigate whether the assumption sets Δ' meet the listener's expectations or not. If not, the sets Δ' can help us to detect flaws in the original representation.

For this form of semantic analysis, we need inference for proving the satisfiability of semantic representations in context. In this thesis, I will discuss some existing model generation methods that have been developed originally for other applications than linguistics. As we will see, there is no off-the-shelf method available that is sufficient for the task at hand. For semantic analysis, we need specialised technical machinery, and I will show how to tailor the existing tools to our needs.

1.2.3 Computation

Technically, this thesis is about developing a model generation method that is suitable for the interpretation and analysis of logical semantic representations. One of the stumbling stones for conventional model generation in this context is higher-order logic. Model generation, like automated deduction, has concentrated on first-order predicate logic or fragments thereof. The standard language of classical first-order predicate logics can be very awkward in places, especially for representing natural language. Since the early 1970s, when Montague [2] initiated the use of the simply typed λ-calculus in natural-language semantics, semanticists use, among other means, higher-order logics with specific linguistic quantifications as a more convenient form of representation. The following are two possible representations of the sentence *Two women love John*, one with a Montague-style syntax and linguistic quantification, and one in a standard first-order form.

$$\textsc{Two}(\mathsf{w})(\mathsf{love}(\mathsf{j})) \tag{1.3}$$

$$\exists x_1\ \exists x_2\ \mathsf{w}(x_1) \wedge \mathsf{w}(x_2) \wedge x_1 \neq x_2 \wedge \mathsf{love}(\mathsf{j})(x_1) \wedge \mathsf{love}(\mathsf{j})(x_2) \tag{1.4}$$

It is easy to see that the Montague-style representation (1.3) is both closer and more natural with respect to the original sentence than the formula in standard syntax (1.4). The use of such Montague-style formalisations can have both theoretical and practical advantages for first-order inference, as has been shown by McAllester and Givan [3]. Linguists use higher-order logics because of its expressivity and because the semantics of natural language utterances can be constructed in a compositional way.

All model generation methods expect their input in a form similar (1.4). As natural-language semantics makes use of logical languages that are more expressive than classical first-order predicate logic, a model generation method that is suitable for natural-language processing must be able to deal with such languages too. However, the use of more expressive logics may change the computational tractability of inference. Although the extra expressive power is often worth the price, we would like to have efficient techniques that can be used in practice and as the basis of scientific evaluation in computational semantics. The mutually exclusive design goals of expressivity and computational tractability can be dealt with in systems that use the more expressive power of the richer formalism, but in fact inhabit a fragment that has better computational properties. One of the contributions of this thesis is the development of an efficient model generation method for Montague-style higher-order logical specifications that has the same complexity as first-order model generation.

1.3 Acknowledgments

Starting at the beginning, I would like to thank Michael Kohlhase and Jörg Siekmann for introducing me to the exciting interdisciplinary research in automated reasoning and formal semantics. They gave me the opportunity to be a part of a very active group which, among other privileges, provided me with access to knowledge of logical systems and how to implement them. The knowledge that I needed desperately was donated by several people, but I have to thank especially my colleagues Michael Kohlhase and Christoph Benzmüller who spend a lot of their valuable time by initiating me into the mysteries of higher-order logics, both with an amount of patience that I still find incredible.

My other collegues at the computer science department, Lassaad Cheikhrourou, Detlef Fehrer, Armin Fiedler, Helmut Horacek, Andreas Meier, Erika Melis, Martin Pollet, and Volker Sorge not only let me be a part of their workload sometimes, but also accepted my “exotic” interest in natural language semantics—they are after all in a group whose primary goal is to develop techniques for planning and presenting mathematical proofs. I suspect that most of them believe that my research interests were only a clever scheme of mine to avoid doing any real work, but they did not let it show too much.

In the computational linguistics department, my colleague Claire Gardent provided the linguistic background that was essential for the present work. Our discussions and close cooperation strongly influenced what I did and how I did it. Some of my “best” ideas died an instant death as soon as I gave her the opportunity to examine them from an experienced linguist’s point of view. I am therefore not only grateful for the work that we did together, but also for a lot of unnecessary work that she prevented me from doing.

Johan Bos kindly let us plug Kimba into the Doris system as an inference module and wrote the translation routine that was necessary for this. Of our students at the computer science department, I would like to thank Andreas Franke for giving me more than a hand with Kimba's integration into Mathweb such that it could be used as part of the Doris system. Carsten Brockmann at the computational linguistics department was equally helpful for the modification of Kimba into a web-based applet.

David Wolfram of the computer science department at The Australian National University, Canberra, gave me the opportunity to further develop the idea of higher-order model generation as a guest of his university for ten weeks. François Bry and his group at the Ludwig-Maximilian University in Munique also were several times my hosts. They showed me the beauty of "lean" model generation and have answered some of my questions about the connections of model generation and abduction. Martin Müller and Christian Schulte of the Programming Systems Lab, Saarbrücken, were the ones I could always turn to for technical support with Oz.

My girlfriend Katja often had to notice that whatever she had just told me did not find a way into my mind: it was still busy with such important questions as whether *giving measles to each other* should be interpreted by IAO or IAR reciprocity. There is also no question that she had to endure bad moods and inexplicable behavior beyond whatever can be expected from a PhD student's girlfriend. Sorry love, I promise that I will never write a dissertation again.

Michael Kohlhase, Christoph Benzmüller, Jörg Siekmann, and Manfred Pinkal had a closer look at parts of this thesis at various stages of its development. I am more than grateful for the constructive criticism and helpful remarks they gave me. Needless to say, all remaining errors are my own.

Finally, I always had and have the support of my family, although I think my dad would like me to stop doing research now. He does not consider it to be proper work at all because there is too much fun involved.

2 Model Generation

Approach your problem from the right end and begin with the answers. Then one day, perhaps you will find the final question. (van Gulik)

Overview: This chapter presents research topics in model generation that will be relevant for later chapters. It also gives an introduction to some of the most popular methods for generating models.

2.1 Introduction

Model generation is important for the automation of reasoning. Its applications are in many different fields such as [4,5], [6,7], [8], [9], [10], software verification [11], [12], and [13]. For this, and other reasons, model generation developed into a rich field of research of its own.

In Section 2.3, we give an introduction to the topics of model generation that are especially interesting for computational semantics and that will play some rôle in later chapters. Such topics are for instance minimal model generation and model enumeration. On the other hand, the methods discussed here are not those that we will actually use for natural-language analysis and interpretation, but survey the standard technical machinery of model generation. The staple techniques of model generation include the use of (hyper-)tableaux calculi and that of efficient propositional decision procedures. In Section 2.4, we discuss both PUHR tableaux and the Davis-Putnam procedure, because these two methods exemplify the two most common approaches to model generation. Section 2.5 gives some pointers to selected works that investigate other methods and topics in model generation.

2.2 Preliminaries

We assume the reader to be familiar with first-order logics and inference procedures such as presented, e.g., in Fitting's textbook on first-order logic and automated theorem proving [14].

α	α_1	α_2	β	β_1	β_2
$\mathsf{A} \wedge \mathsf{B}$	A	B	$\neg(\mathsf{A} \wedge \mathsf{B})$	$\neg\mathsf{A}$	$\neg\mathsf{B}$
$\neg(\mathsf{A} \vee \mathsf{B})$	$\neg\mathsf{A}$	$\neg\mathsf{B}$	$\mathsf{A} \vee \mathsf{B}$	A	B
$\neg(\mathsf{A} \Rightarrow \mathsf{B})$	A	$\neg\mathsf{B}$	$\mathsf{A} \Rightarrow \mathsf{B}$	$\neg\mathsf{A}$	B
$\mathsf{A} \Leftrightarrow \mathsf{B}$	$\mathsf{A} \Rightarrow \mathsf{B}$	$\mathsf{B} \Rightarrow \mathsf{A}$	$\neg(\mathsf{A} \Leftrightarrow \mathsf{B})$	$\neg\mathsf{A} \wedge \mathsf{B}$	$\mathsf{A} \wedge \neg\mathsf{B}$

Fig. 2.1. α- and β-Formulae and Components

γ	$\gamma(t)$	δ	$\delta(t)$
$\forall x\mathsf{F}$	$[t/x]\mathsf{F}$	$\neg\forall x\mathsf{F}$	$[t/x]\neg\mathsf{F}$
$\neg\exists x\mathsf{F}$	$[t/x]\neg\mathsf{F}$	$\exists x\mathsf{F}$	$[t/x]\mathsf{F}$

Fig. 2.2. γ- and δ-Formulae and Instantiations

A **specification** is a finite set of closed formulas, a **theory** a possibly infinite one. Unless indicated otherwise, I use “formula” in lieu of closed formula, i.e., formulas F do by default not contain free variables.

An **atom** a is a formula that does not contain any logical connectives. A **literal** l is either an atom a or its negation $\neg\mathsf{a}$. If a literal l is an atom a, then its **complement** $\bar{\mathsf{l}}$ is $\neg\mathsf{a}$, otherwise its complement is a.

Following Smullyan [15], we classify formulas as implicitly conjunctive (α), disjunctive (β), existentially quantified (δ) or universally quantified (γ). Figure 2.1 shows the components of α- and β-formulae, while figure 2.2 shows the relationship between γ- and δ-formulae and their instantiations. We denote a substitution that instantiates a variable x with a term t with $[t/x]$.

2.3 Topics

In the following, some selected concepts and research topics in model generation are introduced.

2.3.1 Models and Decidability

Models are interpretations of the symbols of a logical theory T such that T becomes true. For a first-order theory, each is associated with a nonempty domain $\mathcal{D}$ of individuals. This domain sometimes is called the.

First-order **interpretations** are pairs $\mathcal{I} = \langle [\![\ ]\!], \mathcal{D} \rangle$ of an interpretation function $[\![\ ]\!]$ that maps predicate and constant symbols to predicates and constants of the appropriate type, and a domain $\mathcal{D}$ of individuals that provides entities for the interpretation of constants. Every interpretation of a term t, $[\![t]\!]$, must be an element of $\mathcal{D}$.

A model $\mathcal{M}$ for a classical propositional logic is simply a mapping $[\![\ ]\!]$ from Boolean constants c into the truth values. We call a problem **combinatorial** if it is equivalent to finding an assignment for finitely many variables with finite domains, and verifying in polynomial time whether the assignment is a solution or not. All combinatorial problems are decidable, because we can use a brute-force algorithm that simply tries out all possible variable assignments. Satisfiability of specifications in classical propositional logic is a combinatorial problem.

The predicate calculus, on the other hand, is undecidable. We know that there is no method that will be able to prove satisfiability in general. Hence, there can also be no procedure that is always able to compute a model for an arbitrary satisfiable first-order specification. However, there exist methods that are complete for finite satisfiability, i.e., that are decision procedures for specifications that either are unsatisfiable or can be satisfied by a finite model (see Section 2.4.5).

2.3.2 Herbrand Models

An important class of models are the **Herbrand models**. Within these, constants and other terms are interpreted as themselves, i.e., $[\![t]\!] = t$ for all terms. This implies that the universe of discourse is a set of the terms that can be built from the signature of the interpreted theory. In a classical first-order logic, a Herbrand model $\mathcal{M}$ and its associated domain $\mathcal{D}$ can be specified completely by the set $\textsc{Ground}(\mathcal{M})$ of all ground literals that are satisfied by $\mathcal{M}$. The domain $\mathcal{D}$ of $\mathcal{M}$ are the terms that occur in this set of literals. We identify Herbrand models $\mathcal{M}$ with their representation as sets of ground literals.

Herbrand models are an important class of models because each satisfiable first-order theory must have at least one Herbrand model. Methods in automated deduction make use of this property when proving unsatisfiability. If a theory is inconsistent, it suffices to show that it cannot have a Herbrand model. This, in general, is simpler to show than the non-existence of arbitrary models. Unless indicated otherwise, we use the term "model" for Herbrand models.

2.3.3 Finite Models

Not all first-order Herbrand models for a formula may have a *finite* representation as a finite set of ground literals. For instance, the formula $\mathsf{p}(f(a)) \land \forall x\ \mathsf{p}(f(x)) \Rightarrow \mathsf{p}(f(f(x)))$ is satisfiable, but only by an infinite model of the form $\mathcal{M} = \{\mathsf{p}(f(a)), \mathsf{p}(f(f(a))), \mathsf{p}(f(f(f(a)))), \ldots\}$. The universe of discourse $\mathcal{D}$ of this model is infinite. A **finite model** is a model with a finite universe of discourse. Most model generation methods are restricted to the generation of such finite models (see Section 2.5).

A serious problem for completeness in model generation is Skolemisation. A Skolem term is a term that represents an equivalence class of δ-formulas up to a renaming of variables [16, 17]. Skolemisation is a convenient method for eliminating existential quantifiers in theorem proving, but for model generation purposes it has serious drawbacks. Skolem terms often introduce new function symbols which extend the domain of the Herbrand models of the input specification. While Skolemisation does not turn a satisfiable specification into an unsatisfiable one, it may transform a finitely satisfiable specification into one that has only infinite Herbrand models. For instance, the formula $\mathsf{p}(a,a) \wedge \forall x\ (\mathsf{p}(x,x) \Rightarrow \exists y\ \mathsf{p}(x,y))$ has a finite Herbrand model $\{\mathsf{p}(a,a)\}$, while a Skolemised form $\mathsf{p}(a,a) \wedge \forall x\ (\mathsf{p}(x,x) \Rightarrow \mathsf{p}(x,f(x)))$ has not. As mentioned before, infinite models are a problem because there can be no method that detects their existence in general.

2.3.4 Representations

A finite representation $\text{Ground}(\mathcal{M})$ still can be awkward in places because of their size. Herbrand models are therefore represented more compactly by their positive part alone, i.e., by the set of positive literals $\mathbf{Pos}(\mathcal{M})$. The interpretation function I that is derived from $\mathcal{M}$ simply interprets all literals P as false if P is not in $\mathcal{M}$. Note that the universe of discourse $\mathcal{D}$ of a might actually be smaller than the terms that occur in the complete model. Throughout this volume, we will usually present models as sets of atoms that leave out all negative literals.

2.3.5 Minimality

An important research topic in model generation is the generation of models with certain properties, the most important property of which is some form of minimality. In the following, we summarize the forms of minimality that have been investigated in the literature.

2.3.6 Subset Minimality

A model $\mathcal{M}$ for a specification Φ is called **subset-minimal** iff for all models $\mathcal{M}'$ of Φ the following holds: $\text{Pos}(\mathcal{M}') \subseteq \text{Pos}(\mathcal{M}) \Rightarrow \text{Pos}(\mathcal{M}') = \text{Pos}(\mathcal{M})$. Subset-minimality is the most important form of minimality and has been the focus of a great deal of attention in the literature. Minimal model reasoning is at the heart of the circumscriptive approach to common sense reasoning. Circumscription, as introduced by McCarthy [18], is one way to formalise non-monotonic logics, and it has been applied to reasoning about actions, planning, and the semantics of logic programs. The logical consequence relation $\models_c$ of circumscriptive reasoning can be formalised as follows: $\Gamma \models_c \Phi$ iff Φ is true in all subset-minimal models of Γ. This consequence relation is stronger than the classical consequence relation $\models$. For instance $\mathsf{A} \wedge \mathsf{B} \models_c \neg\mathsf{C}$, but obviously $\mathsf{A} \wedge \mathsf{B} \not\models \neg\mathsf{C}$. It is non-monotonic because $\mathsf{A} \wedge \mathsf{B} \models_c \neg\mathsf{C}$ holds, but $\mathsf{A} \wedge \mathsf{B} \wedge \mathsf{C} \models_c \neg\mathsf{C}$ does not.

Circumscription needs methods that can verify the subset-minimality of models. Several such methods have been proposed in the literature on circumscriptive reasoning [19–21]. There are also other methods from model generation to generate subset-minimal models [22,23]. Yet, there is no method available that enumerates the minimal models of a specification both in a space- and time-efficient way. Most solutions proposed are not able to handle larger examples because of exponentional worst-case complexity. For instance, the system of Bry and Yahya [22] needs a potentially expensive lookup mechanism for models that have been computed earlier (see Section 2.4.4).

The minimal model reasoning method proposed by Niemelä [24] seems to be the only one on the market that has no exponential space requirements by design. In a tableaux setting, the method is able to locally verify the minimal model property of an open saturated tableau branch, but does not prevent that minimal models are sometimes generated multiple times in different branches. Also, the method is restricted (in its original form) to propositional logics.

2.3.7 Domain Minimality

A second definition of model minimality is domain minimality. A Herbrand model $\mathcal{M}$ for a specification Φ is **domain-minimal** if the size $|\mathcal{D}(\mathcal{M})|$ of its universe of discourse is minimal, i.e., there is no model $\mathcal{M}'$ of Φ such that $|\mathcal{D}(\mathcal{M}')| < |\mathcal{D}(\mathcal{M})|$. Reasoning with domain minimal models has been investigated by Hintikka [25] and Lorenz [26] as an alternative form of circumscription where the focus lies on minimizing the domain size rather the facts. Unlike subset-minimality, verifying domain-minimality is computationally tractable for every model generation method that is guaranteed to find a domain-minimal model first. A method that enumerates models in increasing size of the domain can be derived from the EP calculus presented in Section 2.4.5.

2.3.8 Predicate-Specific Minimality

Let p be a certain predicate symbol, and $\mathsf{p}(\mathcal{M})$ be the subset of atoms in $\textsc{Pos}(\mathcal{M})$ whose head predicate symbol is p. We call a model $\mathcal{M}$ of a specification Φ p-minimal if there is no model $\mathcal{M}'$ of Φ such that $|\mathsf{p}(\mathcal{M})| < |\mathsf{p}(\mathcal{M})|$.

Applications such as diagnosis and planning often employ some forms of p-minimal models. For instance, logical specifications in model-based diagnosis are used for describing the correct behavior of a device, while a certain predicate ab is used for abnormal behavior. The following specification may describe an AND-gate in a circuit:

$$\mathsf{on}(in_1) \wedge \mathsf{on}(in_2) \Rightarrow \mathsf{on}(out_1) \vee \mathsf{ab}(AND_1)$$

The formula specifies that $on(out_1)$, the output of our AND-gate AND_1, must hold if both inputs in_1 and in_2 are on as well, otherwise the device AND_1 is abnormal. A set of such rules could be used to describe the in/out-behavior of an integrated circuit. A **diagnosis** is then a model of such a description Γ together with a set of observations Φ that indicate a faulty device. The observations Φ correspond to the in/out-states of parts of the circuit. The models that are interesting are those that explain the truth of $\Gamma \cup \Phi$ which refer to a minimum of devices in the predicate ab. These minimal models show how the error indicated by Φ can be caused without assuming more faulty devices than necessary. The specific task in diagnosis is to compute models that minimize not all atoms, but only the atoms that depend on the predicate *ab*. Other parts of the model are of no interest. We call models that are minimal with respect to a certain set of predicates **predicate-specific minimal models**.

2.3.9 Enumeration

The minimal models of a logical problem specification usually represent different solutions of the problem. As a consequence, we are often not interested in just computing one single model for a specification, but want to enumerate models instead. This implies that the generation method is complete with regard to the solutions we are interested in. For many applications, the form of completeness that is asked for is completeness for finite satisfiability [27].

2.3.10 Model Enumeration with Theorem Provers

Some theorem provers decide fragments of first-order logics. For instance, the theorem proving system implements a decision procedure for the [28]. Within such decidable fragments, the related proof procedure can sometimes be modified such that a model is constructible from a failed refutation proof attempt. Yet, high-speed theorem proving calculi are not designed for enumerating the models of a specification. The general use of Skolemisation makes it impossible for conventional automated theorem provers to be complete for finite satisfiability.

On the other hand, calculi for model generation theorem proving can simultaneously be refutation complete and complete for finite satisfiability if we do not Skolemise (see Section 2.4.5). The system by Bry and Torge [27] implements this idea. It is the only theorem prover currently available whose underlying hyper-tableaux calculus is provably complete for finite satisfiability. The original [29] system uses Skolemisation, and therefore shows the same enumeration incompleteness with regard to finite satisfiability as other theorem provers.

2.3.11 Enumeration with Finite Model Generators

Unlike model generation theorem provers, are programs whose only purpose is the generation of finite models and not the refutation proof of a theorem. One of the best known systems for finite model generation is Slaney's [30].

Finite model generators apply propositional decision procedures to first-order specifications that have been translated into propositional logic using a given universe of discourse. Programs such as FINDER are frequently used for applications that require the exhaustive search for finite models within very large search spaces [4]. By using efficient decision procedures, finite model generators can solve combinatorial problems that are far beyond the capabilities of first-order methods.

As a welcome side-effect, the use of complete propositional decision procedures results in a finitely complete form of model generation where all finite models of a specification up to a renaming of constant symbols are generated.

Every propositional model is also a unique finite model of the input first-order specification. By iterative deepening over the size of the domain, all finite models can be successively enumerated in a space- and time-efficient way.

2.4 Methods

Most of the calculi developed in automated deduction that prove unsatisfiability can also sometimes be used for proving satisfiability. For many non-classical logics such as a variety of modal logics, the proof procedures available are not only refutation complete, but actually decide theories in these logics. Also, efficient procedures for testing satisfiability in (classical) propositional logics are a major research topic in Automated Reasoning and Artificial Intelligence [31–35].

Model generation as a field distinguishes itself from research on SAT and decision procedures in two aspects. First, the logics considered are undecidable in general and have some unrestricted notion of quantification over individuals. This means that the logical languages used are not tailored to be a decidable fragment of first-order logics such as those used in knowledge representation [36]. Second, the methods generate objects from which models can be derived.

The border line between model generation and other research on satisfiability is not a strict one. For instance, propositional decision and SAT procedures have been used successfully for first-order model generation in planning [8] and finite mathematics [37].

Model generation as a field of research is associated with a variety of tableaux methods, the most prominent being Positive Unit Hyper-Resolution (PUHR). We present this tableaux method in Section 2.4.4 as a standard device in model generation. An important improvement of PUHR with regard to our intended application is completeness for finite satisfiability, which is exemplified in the Extended Positive (EP) tableaux calculus in Section 2.4.5. The core method for model generation that we will use throughout this volume is constraint-based finite model generation. This approach shares many properties with the Davis-Putnam procedure, a propositional decision procedure that we will discuss in Section 2.4.6. Constraint-based model generation is then the main topic of the next chapter.

2.4.1 Analytical Tableaux

Many calculi for model generation are based on analytical tableaux, although conventional tableaux calculi with an unrestricted syntax are rarely used for model generation. In the following, we show why standard tableaux methods are not suitable for finite model generation in general.

2.4.2 Ground Tableaux

The set of expansion rules shown in Figure 2.3 defines a sound and refutation complete theorem proving calculus for first-order predicate logics called **ground tableaux** [38].

$$\frac{\beta}{\beta_1 \,|\, \beta_2} \qquad \begin{array}{c}\alpha\\ \hline \alpha_1\\ \alpha_2\end{array} \qquad \frac{\neg\neg\mathsf{F}}{\mathsf{F}} \qquad \frac{\gamma}{\gamma(t)} \qquad \frac{\delta}{\delta(sko(\delta))}$$

Fig. 2.3. Tableaux expansion in conventional tableaux

In conventional analytical tableaux, an arbitrary input formula is analysed by applying expansion rules, thus creating the well-known tree-structure of tableaux proofs. A branch in such a tableau is closed if it contains an elementary contradiction, such as an atom a and its negation $\neg\mathsf{a}$. A closed tableau is a tableau where all branches are closed. A closed formula proves that the input formula is unsatisfiable. Figure 2.4 shows a closed tableau for the formula $(\forall x\ \neg \mathsf{p}(x)) \wedge (\mathsf{p}(a) \vee \mathsf{p}(b))$ The tableau proof instantiates the γ-formula $\forall x\ \neg\mathsf{p}(x)$ twice, creating two new formulas, $\neg\mathsf{p}(a)$ and $\neg\mathsf{p}(b)$. These are used for showing that the disjunction $\mathsf{p}(a) \vee \mathsf{p}(b)$ can not be satisfied.

$$\begin{array}{c}
(\forall x\ \neg \mathsf{p}(x)) \wedge (\mathsf{p}(a) \vee \mathsf{p}(b))\\
\forall x\ \neg\mathsf{p}(x)\\
\mathsf{p}(a) \vee \mathsf{p}(b)\\
\neg\mathsf{p}(a)\\
\neg\mathsf{p}(b)\\
\begin{array}{c|c}
\mathsf{p}(a) & \mathsf{p}(b)\\
\star & \star
\end{array}
\end{array}$$

Fig. 2.4. A tableau proof for refuting $(\forall x\ \neg \mathsf{p}(x)) \wedge (\mathsf{p}(a) \vee \mathsf{p}(b))$

Branches are frequently identified with the set of formulas they contain. A branch is called **saturated** if no application of an expansion rule can add new elements to this set. In analytical tableaux, the atoms in a saturated open branch define a positive model of the input formula.

The γ-rule of ground tableaux is a serious source of inefficiency. When applying the rule, we must "guess" the right terms and number of instantiations that are necessary in order to close all branches. There is, in principle, no limit for the number of instantiations that might be necessary to close a branch. For model generation, we must use all different terms that occur in the same branch as the γ-formula while Skolemisation in the δ-rule successively might introduce new terms. This quickly leads to infinite branches.

2.4.3 Free Variable Tableaux

The free variable tableaux is a basis of many deduction systems. It is particulary easy to implement, and some systems consist only of a few lines of Prolog code [39]. In a free variable tableaux, the instantiation of γ-formulas by ground terms is replaced by an instantiation with some variable x that is new to the tableau and whose instantiation has to be determined later (see Figure 2.5). A branch containing two formulas E and $\neg\mathsf{F}$ can be closed iff there is a variable assignment, i.e., a unifier σ, such that $\sigma(\mathsf{E}) = \sigma(\mathsf{F})$. The unifier σ can be computed efficiently by first-order **unification**, and σ is then applied to the whole tableau in order to remove all occurrences of the free variables in E and F.

By using unification, the selection of a term that instantiates a γ-formula can be delayed until this instantiation actually helps closing a branch. This is a considerable advantage in practice. Still, the selection can be wrong, because there might be several formula pairs E and F in a branch that are suitable for closing it. Applying a unifier has a global effect on the whole tableau, and a wrong choice in one part of a tableau might require additional γ-instantiations in another part. In order to have a fair expansion of γ-formulas, implementations often use iterative deepening over the number of γ-instantiations that are allowed in one branch.

$$\frac{\gamma}{\gamma(x)}$$

Fig. 2.5. Free variable tableaux expansion

The quantifier rules in a free variable tableaux still are a source of model incompleteness. Once a branch contains one γ-formula, it can be extended by an unlimited number of successive applications of the γ-rule. Hence, even if a branch has a model, we might not be able to detect it because one or several γ-formulas prevent the finite saturation of a branch. In practice, there are nevertheless methods that can be used for detecting models in potentially infinite branches. A formula that contains a variable represents an infinite set of ground Herbrand terms, and if a branch provably cannot be closed, such methods are even able to generate infinite models. However, infinite model generation requires a considerable amount of technical machinery and still cannot prevent model incompleteness because of Skolemisation. For a survey of the state-of-the-art techniques in this area, we refer to Klingenbeck's thesis [40].

2.4.4 Positive Unit Hyper-resolution

Positive Unit Hyper-Resolution () tableaux is the theoretical basis of the theorem prover [29,41]. Introduced by Bry and Yahya [22], this calculus addresses the efficiency problem that the blind instantiation of γ-formulas poses for conventional analytical tableaux calculi. PUHR tableaux is a ground calculus, i.e., the formulas in a branch do not contain free variables.

PUHR tableaux are constructed from an initially empty branch by applying the inference rule in Figure 2.6. The single PUHR tableaux rule is a sound and refutation complete calculus for clausal specifications.

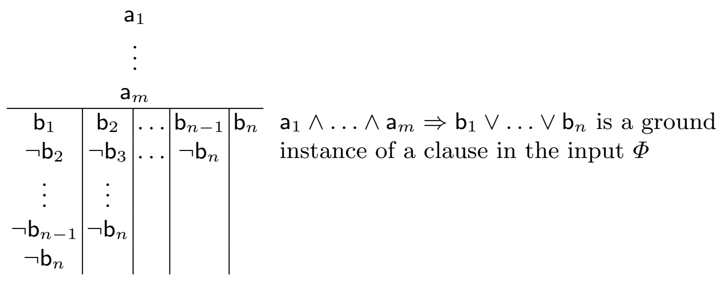
a_1
a_m
b_1
$\neg b_2$
$\neg b_{n-1}$
$\neg b_n$
b_2
$\neg b_3$
$\neg b_n$
$\dots$
b_{n-1}
$\neg b_n$
b_n
$a_1 \wedge \ldots \wedge a_m \Rightarrow b_1 \vee \ldots \vee b_n$ is a ground instance of a clause in the input Φ

Fig. 2.6. The PUHR tableaux expansion rule

If a branch θ contains all body atoms $a_1 \ldots a_m$ of some ground instance of a clause $C = a_1 \wedge \ldots \wedge a_m \Rightarrow b_1 \vee \ldots \vee b_n$ in the input specification Φ, then θ is split into n branches, each extended by some atom b_i. In addition, each branch also contains the complements of the atoms b_j with $j > i$. If C has no atoms b_i, i.e., $n = 0$, then the branch is closed. A branch is also closed if it contains an atom and its negation. PUHR tableaux satisfy a regularity condition that forbids the application of a ground clause C if one of the atoms b_i already occurs in the current branch.

The input to the PUHR calculus is a set of **range-restricted rules** $F_1 \wedge \ldots \wedge F_m \Rightarrow E_1 \vee \ldots \vee E_n$, i.e., first-order clauses with free variables where all variables in $E_1 \vee \ldots \vee E_n$ also occur in $F_1 \wedge \ldots \wedge F_m$. By using a special domain predicate *dom* for introducing variables on the left-hand side, all first-order specifications can be translated into this form. A rule is applied only if the left-hand side as a whole can be instantiated with the atoms that occur in the current branch. PUHR tableaux can avoid the effect of the blind instantiation of γ-formulas because implicative rules need only be instantiated if all their preconditions are met.

A branch of a PUHR tableaux is frequently identified with the set of atoms it contains. These atoms define a partial interpretation. A saturated PUHR tableau Θ for a clausal specification Φ has, among others, the following properties:

- Every open branch of Θ is a positive model of of Φ.
- Every subset-minimal model of Φ appears as a branch of Θ.
- If two models $\mathcal{M}$ and $\mathcal{M}'$ appear in Θ and $\text{Pos}(\mathcal{M}) \subseteq \text{Pos}(\mathcal{M}')$, then the branch containing $\mathcal{M}$ appears left of the branch of $\mathcal{M}'$.

PUHR tableaux can be used for the generation of subset-minimal models if non-minimal models are eliminated. In order to do so, we traverse a tableau in a depth-first left-to-right manner and check for every generated model whether it is not a superset of some earlier computed model. This requires a potentially expensive lookup mechanism. We either have to keep track of all models that have been computed earlier, or we must repeat parts of the earlier tableau proof.

PUHR tableaux, like other clausal calculi, must employ Skolemisation in order to have a clausal input that is equivalent to the original first-order specification. As mentioned before, Skolemisation may cause infinite Herbrand models. A saturated PUHR tableau always has a branch for each minimal model, but this theoretical property is often of no relevance in practice because a tableau that contains infinite models cannot be saturated by finitely many expansion steps. In this case, a branch will be expandable infinitely many times, and we will not be able to verify the existence of its model in general.

2.4.5 A Method Complete for Finite Satisfiability

The simple PUHR calculus presented in the last section is in many ways interesting for both theoretical and practical research and has inspired many related works. One of the various derivations based on the PUHR tableaux addresses the question of how Skolemisation can be eliminated in order to get completeness for the generation of minimal models. The Extended Positive (EP) tableaux calculus by Bry and Torge [27] uses a special δ-rule which avoids Skolemisation. Fig. 2.7 displays EP's expansion rules.

$$\frac{\beta}{\beta_1 \mid \beta_2} \qquad \frac{\alpha}{\begin{array}{c}\alpha_1\\ \alpha_2\end{array}} \qquad \frac{\forall \overline{x}\ (\mathsf{C}(\overline{x}) \rightarrow \mathsf{D})}{\mathsf{D}[\overline{c}/\overline{x}]}$$

$$\frac{\exists x\ \mathsf{F}(x)}{\mathsf{F}[c_1/x] \mid \ldots \mid \mathsf{F}[c_k/x] \mid \mathsf{F}[c_{new}/x]}$$

Fig. 2.7. The EP calculus

EP accepts so-called Positive Restricted Quantification (PRQ) rules as its input. These rules are rules of the form $\forall \overline{x}(\mathsf{C} \rightarrow \exists \overline{y}\ \mathsf{F})$ where C is a conjunction of atoms and F is a disjunction of atoms and PRQ rules. The rules do not contain function symbols. Every first-order specification in classical logic can be translated into a PRQ specification.

In the $\forall$-rule in Fig. 2.7 above, $\overline{c}$ is a tuple of constants occurring in the expanded node. These constants are determined by evaluating C, a conjunction of atoms, against the already constructed interpretation, i.e., the Herbrand model determined by the set of ground positive literals occurring in the current branch. The $\forall$-rule handles both universally and implicative formulas, and, like the original PUHR rule from which it is derived, only extends the current branch if there is a complete match of the implication's left side with the already constructed interpretation.

The $\exists$-rule (originally called the $\delta^\star$-rule in [27]) instantiates existentially quantified formulas x with constants $\{c_1, \ldots, c_k\}$ that occur in the current branch and that form the current domain $\mathcal{D}$. It also extends the tableau with a new branch that instantiates x with a new constant c_{new}. By this, the use of a Skolem term with a potentially dangerous introduction of function symbols is avoided. The reuse of constants also ensures that EP constructs models with minimal domains first when using a left-to-right traversion of the tableau.

With the elimination of Skolem terms, and function symbols as a whole, EP is complete for finite satisfiability. Like PUHR tableaux, EP is refutation complete and simultaneously searches for models and proofs.

2.4.6 The Davis-Putnam Procedure

All tableaux methods presented in Sections 2.4.1–2.4.5 are refutation complete proof calculi for first-order logics. They not only compute models and refutation proofs simultaneously, but also determine the domain of the models they compute. A quite different approach to model generation is the use of where the universe of discourse must be chosen in advance.

Given a domain of individuals $\mathcal{D}$, one translates the input specification Φ into ground formulas or ground clauses by translating δ-formulas into disjunctions and γ-formulas into conjunctions. The result are propositional theories for which efficient decision procedures are available. A propositional model determines the truth value of all ground literals and can be translated back into a model of the first-order input specification. If the specification is unsatisfiable, one simply extends $\mathcal{D}$ by an additional entity and starts the translation into propositional logics again. By iterative deepening on the size of $\mathcal{D}$, all finite models of the input can be enumerated.

A simple, yet efficient decision procedure is the Davis-Putnam procedure[1]. In its original form, as presented in [43], it was intended to be a theorem-proving technique suitable for automation, covering both classical propositional and first-order logics. The first-order version was not as efficient as resolution, which was introduced soon after, but the propositional version is still among the fastest [14].

2.4.7 Calculus and Procedure

The Davis-Putnam procedure works on ground clauses $\mathsf{a}_1 \vee \ldots \vee \mathsf{a}_m \vee \neg\mathsf{b}_1 \vee \ldots \vee \neg\mathsf{b}_n$ that we will write in the following as **literal sets** $\{\mathsf{l}_1, \ldots, \mathsf{l}_{n+m}\}$. Like all sets, a literal set does not contain the same elements twice. Additionally, we remove all clauses from our input that contain both some literal l and its complement $\bar{\mathsf{l}}$, i.e., that are tautologies.

Like a tableaux calculus, the Davis-Putnam procedure manipulates sequences of formulas that we organise into branches. Unlike tableaux, however, we do not have a tree-like proof structure, but each branch is kept separately. A branch can be modified destructively by removing formulas. Like in a tableaux, a saturated open branch proves the satisfiability of the input. A branch is closed if it contains the empty clause $\{\}$, or a unit clause $\{\mathsf{l}_1\}$ together with the complementary clause $\{\overline{\mathsf{l}_1}\}$.

Figure 2.8 shows the rules of the Davis-Putnam calculus. The first rule eliminates clauses $\{\mathsf{l}_1, \ldots \mathsf{l}_n\}$ if there is a unit clause $\{\mathsf{l}_1\}$ in the same branch.

[1] It is more precisely referred to as the Davis-Logeman-Loveland procedure [42].

This is sound because every interpretation that validates $\{\mathsf{l}_1\}$ must also validate $\{\mathsf{l}_1, \dots \mathsf{l}_n\}$. The second rule is propositional unit resolution. If both $\{\mathsf{l}_1, \mathsf{l}_2, \dots \mathsf{l}_n\}$ and $\{\overline{\mathsf{l}_1}\}$ are in the same branch, we can simplify $\{\mathsf{l}_1, \mathsf{l}_2, \dots \mathsf{l}_n\}$ to $\{\mathsf{l}_2, \dots \mathsf{l}_n\}$ because we know already that literal l_1 must be interpreted as false in the current branch.

$$\frac{\begin{array}{c}\{\mathsf{l}_1, \dots \mathsf{l}_n\}\\ \{\mathsf{l}_1\}\end{array}}{\text{remove } \{\mathsf{l}_1, \dots \mathsf{l}_n\}} \qquad \frac{\begin{array}{c}\{\mathsf{l}_1, \mathsf{l}_2, \dots \mathsf{l}_n\}\\ \{\overline{\mathsf{l}_1}\}\end{array}}{\begin{array}{c}\text{remove } \{\mathsf{l}_1, \dots \mathsf{l}_n\}\\ \text{add } \{\mathsf{l}_2, \dots \mathsf{l}_n\}\end{array}} \qquad \frac{\begin{array}{c}\{\mathsf{l}_1, \dots \mathsf{l}_n\}\\ \{\overline{\mathsf{l}_1}, \dots, \mathsf{l}'_m\}\end{array}}{\{\mathsf{l}_1\} \mid \{\overline{\mathsf{l}_1}\}}$$

Fig. 2.8. The Davis-Putnam calculus

The final rule splits a branch if there is a literal l_1 that occurs both positively and as its complement in some non-unit clauses. This results in two new branches, one that contains the unit clause $\{\mathsf{l}_1\}$ and one that contains the unit clause $\{\overline{\mathsf{l}_1}\}$.

The Davis-Putnam procedure defines an effective order of rule applications for the calculus. The first two rules are always applied exhaustively on a branch until no further simplification is possible. Then, a literal is selected for splitting, and the process of simplification that is defined by the first two rules starts again. As a heuristic, the procedure selects literals that occur in as many other formulas as possible in order to maximize the effect of the simplification rules.

2.4.8 Branches as Models

The output of the Davis-Putnam procedure is a finite set of branches. If at least one of them is open, then the input is satisfiable. Unlike the tableaux calculi discussed before, a single branch can actually represent several models. Consider for instance the set of literal sets $\{\{\mathsf{a}, \mathsf{b}\}, \{\mathsf{d}, \mathsf{e}, \mathsf{c}\}, \{\neg\mathsf{c}\}\}$. This set can be simplified by applying unit resolution to the last two formulas, and we obtain the saturated open branch $\{\{\mathsf{a}, \mathsf{b}\}, \{\mathsf{d}, \mathsf{e}\}, \{\neg\mathsf{c}\}\}$. The models represented in a branch are defined by all atoms that occur in paths through the set of clauses that select one literal in each clause. In our example, we obtain the following positive models:

$$\{\mathsf{a}, \mathsf{d}\}, \{\mathsf{a}, \mathsf{e}\}, \{\mathsf{b}, \mathsf{d}\}, \{\mathsf{b}, \mathsf{e}\}$$

As in the case of first-order models, the interpretation I derived from a positive model $\mathcal{M}$ meets the following condition: $I(\mathsf{c}) = \mathit{true}$ iff $\mathsf{c} \in \mathcal{M}$ for all atoms c.

With a slight variation of the splitting rule, the Davis-Putnam procedure actually enumerates all models of its input in separate branches. Figure 2.9 shows the rule that can now split a branch on literal l_1 if this literal occurs in any non-unit formula in the branch.

$$\frac{\{\mathsf{l}_1, \ldots \mathsf{l}_n\}}{\{\mathsf{l}_1\} \mid \{\bar{\mathsf{l}}_1\}}$$

Fig. 2.9. The modified splitting rule for enumeration

2.4.9 Efficiency

The Davis-Putnam-procedure is the basis of a family of model generation programs (see Section 2.3.11) that all have been used with some success in finite mathematics. The Davis-Putnam programs have been applied to combinatorial problems with very large search spaces that are beyond the capabilities of first-order methods.

At a first glance, it is not obvious why the Davis-Putnam procedure is more effective for propositional decision problems than so many other proof procedures. The splitting rule for instance requires that the set of clauses is split into independent partitions that can be manipulated destructively while a tableaux allows us to keep common parts of different branches in the same branch the tree-like proof structure. In fact, if the modified splitting rule would be applied for all literals before simplification takes place, a Davis-Putnam proof would have the same complexity than a truth-table method, i.e., there would always be 2^n branches with n being the number of different literals in the input problem.

The success of the Davis-Putnam procedure can be explained by the potential of simplification due to the the first two rules. Any unit clause in the current branch is used for simplifying all other clauses in which it occurs. If a literal l occurs in the same polarity in a clause C in the branch, then C is removed immediately, which shortens the branch. If l occurs as $\bar{\mathsf{l}}$ in some non-unit clause C', then C' can be shortened at least by one literal. These shortened clauses are possibly unit clauses, and can be used immediately for simplification. Splitting on some literal introduces two new branches, each of these simplifiable. As an effect, many potential splittings on literals may never become necessary, either because the literals involved are eliminated from the branch entirely, or because the literal or its complement remain as a unit clause in the branch. In this case, the interpretation of the literal is already given.

2.5 Related Work

There currently is no overview article or an annotated bibliography available that gives a throughout introduction to the various topics, methods, and applications in model generation. Hasegawa [44] focuses on model generation theorem proving such as the PUHR tableaux approach. For finite model generation, the article by Slaney, Fujita and Stickel [4] sketches the techniques of different programs, while Zhang and Stickel [45] explain in detail how the Davis-Putnam procedure can be implemented efficiently.

There are several model generation programs available that have been developed in the automated deduction community. A family of model generators is based on the Davis-Putnam procedure, for instance, [45], and [37]. Both [46] and its predecessor [47] use techniques from constraint solving instead or in addition to those that have been pioneered by Finder [30].

[48] is a model generation theorem prover that is based on a PUHR-like tableaux calculus. The system implements some interesting refinements of PUHR, for instance an optimised proof procedure for horn clauses.

While PUHR tableaux as used in Satchmo and MGTP is a ground calculus, the hyper-tableaux derivation of PUHR by Baumgartner et al. uses free variables [49, 50] in order to suppress unnecessary ground instantiations when proving theorems. Also for theorem proving applications, variants of Satchmo have been developed that apply rules in a more goal oriented way by ordering them with respect to relevancy measure [51]. Abdennadher and Schütz [52] present a PUHR tableaux calculus that can handle existentially quantified variables whose domain of instantiation is restricted by constraints.

This volume concentrates on the generation of *finite* Herband models for natural-language semantic representations. There are some methods available in other application areas that generate certain classes of infinite models. The methods proposed by Klingenbeck [40] and Peltier [53] investigate that have finite representations. For an overview of techniques in infinite model generation, we refer to Klingenbeck [40].

There are many model generation methods that are related to some form of first-order (hyper)resolution (see [54] for a summary).

Unlike proof presentation in theorem proving, the computation of an appropriate representation of models or parts thereof is a relatively new research topic in model generation. Horacek and Konrad [55] propose linguistically motivated techniques for presenting finite Herbrand models in a human-oriented way.

3 Higher-Order Model Generation

By keenly confronting the enigmas that surround us, and by considering and analysing the observations I had made, I ended up in the realms of mathematics.
(M.C. Escher)

Overview: The simply typed λ-calculus can be taken as the basis of a formalism in which we assign meaning to the basic expressions of a natural language and explain the meaning of larger constituents by the composition of the meaning of their parts. However, under a conventional higher-order semantic for the λ-calculus, we cannot give a model generation method based on the well-known techniques of first-order model generation. By weakening the semantic of higher-order logic, we formulate a logical language that has the compositional expressivity of a higher-order logic but the finitely representable models of a first-order one. A generic model generation technique for this language is presented and discussed, and a refinement for minimal model generation is introduced.

3.1 The λ-Calculus in Linguistics

The simply typed λ-calculus has been invented by Church [56] with the goal of providing a uniform language with which to describe functions. In the 1950s, its untyped variant was the starting point for the functional programming paradigm in computer science, notably as the theoretical basis of the programming language LISP. It has become a standard tool of computational semantics at least since the 1970s when Montague [2] first introduced his theory of quantification in natural language.

The simply typed λ-calculus is a an expressive, elegant, and uniform method of composing functions out of more primitive functions. The tools for building its complex expressions are function application and λ-abstraction. The operation of β-reduction is the essence of computation, whereas λ-abstraction is the essence of function definition.

In computational semantics, the λ-calculus can be taken as the basis of a formalism in which we assign a formal meaning to the basic expressions of a natural language and explain the meaning of larger constituents by the composition of the meaning of their parts.

3.1.1 Composition of Meaning

In a standard first-order predicate logic, we can represent the meaning of a sentence like *a man loves a woman* as a formula, e.g., as $\exists x\ \exists y\ \mathit{man}(x) \wedge \mathit{woman}(y) \wedge \mathit{love}(x, y)$ where the non-logical constants man, woman and love are given a postulated meaning that corresponds to the meaning of the associated words in the natural language.

What we cannot represent directly as one first-order logical expression is the contribution that the meaning of the verb phrase has to the meaning of the whole sentence. This means that some constituent like *loves a woman* has no corresponding first-order logical form, at least not without additional pieces of machinery. In this sense, standard first-order predicate logic lacks **compositionality**.

One of the primary goals of semantics construction is to develop a truly compositional method for constructing meaning representations that define the meaning of complex constituents in terms of the meaning of its parts. The stumbling stone for giving any such compositional construction mechanism for natural-language semantics is quantification.

3.1.2 Quantification in Natural Language

Russell [57] notes the disturbing variety among the logical contributions of subject constituents to simple sentences. Consider for instance the following sentences.

Sentence	Logical Form
Peter died	$\mathit{died}(peter)$
A man died	$\exists x\ \mathit{man}(x) \wedge \mathit{died}(x)$
The man died	$\mathit{died}(\iota x\ \mathit{man}(x))$
One man died	$\exists x\ \mathit{man}(x) \wedge \mathit{died}(x) \wedge \forall y\ \mathit{man}(y) \wedge \mathit{died}(y) \Rightarrow x = y$
Every man died	$\forall x\ \mathit{man}(x) \Rightarrow \mathit{died}(x)$

As we can see, structurally small changes in the subject constituent of the natural language sentences lead to very different logical forms. The variety of logical representations seemingly prohibits a compositional method for constructing the semantics of sentences.

The contemporary solution to this problem is based on the use of **generalised quantifiers** and **generalised determiners**. A generalised quantifier applies to a property and produces a truth value. Such quantifiers in natural language are proper names like *Peter* that in the sentence *Peter died* takes a property, *died*, in order to produce the truth of the sentence *Peter died.* A generalised determiner can then be taken to be a relation between two or more sets. In the case of diadic determiners we have two sets, one contributed by the restriction from the noun and one contributed by the predicate that supplies the scope of quantification. In the sentence *every man died*, the word *every* is a diadic determiner that formulates a relation between the set of men and the set of died entities. Like *Peter*, the phrase *every man* is a quantifier, because it applies to a property. What is puzzling in natural language is the infinite variety of such quantifiers that can be constructed easily from the set of available determiners and nouns, and the variety of logical contributions they show when sentences are translated into first-order logic. A compositional treatment of linguistic quantification requires some uniform way of constructing complex generalised quantifiers from their parts. Such a uniform way exists for the λ-calculus.

3.1.3 Quantifiers as Higher-Order Expressions

The simply typed λ-calculus can represent generalised quantifiers as second-order functions that take a set and return a truth value. Indeed, we can even calculate suitable semantic representations for quantifiers by solving higher-order equations. In the following, the quantifiers *every man* and *some man* are treated as higher-order variables $\textsc{EveryMan}^\star$ and $\textsc{SomeMan}^\star$. Typical equations that we have to solve are

$$\textsc{EveryMan}^\star(\mathit{died}) = \forall x\ \mathit{man}(x) \Rightarrow \mathit{died}(x) \tag{3.1}$$

$$\textsc{SomeMan}^\star(\mathit{died}) = \exists x\ \mathit{man}(x) \land \neg\mathit{died}(x) \tag{3.2}$$

The right-hand side of the equation is the logical form that we would like to have for the whole sentence. Solving the equations gives us the two results

$$\textsc{EveryMan}^\star = \lambda P\ \forall x\ \mathit{man}(x) \Rightarrow P(x) \tag{3.3}$$

$$\textsc{EveryMan}^\star = \lambda P\ \forall x\ \mathit{man}(x) \Rightarrow \mathit{died}(x) \tag{3.4}$$

for the first equation and

$$\textsc{SomeMan}^\star = \lambda P\ \exists x\ \mathit{man}(x) \land P(x) \tag{3.5}$$

$$\textsc{SomeMan}^\star = \lambda P\ \exists x\ \mathit{man}(x) \land \mathit{died}(x) \tag{3.6}$$

for the second equation. Of these four solutions, only two, namely (3.3) and (3.5), are linguistically valid. The other two are vacuous abstractions that do not correspond to the semantics of the natural-language quantifiers. The application of result (3.4) to the representation of a different verb shows why we must discard such solutions.

$$\begin{aligned}(3.7)\ & \textsc{EveryMan}^\star(\mathit{sleep}) && =_{def}\\ & (\lambda P\ \forall x\ \mathit{man}(x) \Rightarrow \mathit{died}(x))(\mathit{sleep}) && =_\beta\\ & \forall x\ \mathit{man}(x) \Rightarrow \mathit{died}(x)\end{aligned}$$

Clearly, the resulting first-order logical form is not the intended semantic representation of *every man sleeps.* Vacuous abstractions can be dealt with automatically by extending higher-order equational reasoning with syntactic filter mechanisms [58].

By discarding the linguistically invalid solutions, we obtain some suitable semantics for the linguistic quantifiers *every man* and *some man* in the form of λ-terms. But how are such semantics composed from their parts? Now that we have the logical form of the quantifiers at our hands, we simply use higher-order equational reasoning again for computing the semantics of their parts, namely of the determiners *every* and *some.* Their semantic representations, when applied to the representation of the noun *man*, should yield the non-vacuous quantifiers that we have computed above. Hence, the higher-order equations that we must solve now are as follows.

$$\textsc{Every}^{\star}(\mathsf{man}) = \lambda P\ \forall x\ \mathsf{man}(x) \Rightarrow P(x) \tag{3.8}$$

$$\textsc{Some}^{\star}(\mathsf{man}) = \lambda P\ \exists x\ \mathsf{man}(x) \wedge P(x) \tag{3.9}$$

By solving these, and discarding the vacuous solutions, we obtain definitions in higher-order logic for our determiners.

$$\textsc{Every} \equiv \lambda Q\ \lambda P\ \forall x\ Q(x) \Rightarrow P(x) \tag{3.10}$$

$$\textsc{Some} \equiv \lambda Q\ \lambda P\ \exists x\ Q(x) \wedge P(x) \tag{3.11}$$

With these definitions, we can represent simple sentences like *every man drinks* or *no Greek lies* in a compositional way that is closer to the syntax of the original sentence than a standard first-order logical form. At the same time, we can always reduce the representation to a purely first-order logical form by expanding the definition and performing β-reduction.

$$\begin{array}{ll} \textsc{Every}(\mathsf{man})(\mathsf{drink}) & =_{def} \\ (\lambda Q\ \lambda P\ \forall x\ Q(x) \Rightarrow P(x))(\mathsf{man})(\mathsf{drink}) & =_{\beta} \\ (\lambda P\ \forall x\ \mathsf{man}(x) \Rightarrow P(x))(\mathsf{drink}) & =_{\beta} \\ \forall x\ \mathsf{man}(x) \Rightarrow \mathsf{drink}(x) & \end{array} \tag{3.12}$$

One of the central ideas of Montague's approach is the treatment of proper names such as *Peter.* As mentioned earlier, proper names act as quantifiers in sentences. At the same time, they appear as individual constants in the logical form. This discrepancy is resolved by introducing quantifiers for proper names that, when applied to their scope of quantification, introduce the first-order constant into the logical form. For instance, the proper name *Peter* is represented by a quantifier $\textsc{Peter} \equiv \lambda P\ P(peter)$ where *peter* is an individual constant. By raising the type of proper names to quantifiers, we can treat them as required by a compositional approach.

$$\begin{array}{ll} \textsc{Peter}(\mathsf{drink}) & =_{def} \\ (\lambda P\ P(peter))(\mathsf{drink}) & =_{\beta} \\ \mathsf{drink}(peter) & \end{array} \tag{3.13}$$

3.1.4 First-Order Limitations

It seems that we now have the best of two worlds. While a compositional higher-order logic is used at the representation level, we actually still have a standard first-order logical form that is immediately computable by definition expansion and β-reduction. However, the approach has its limits in that sentences with certain quantifiers cannot be reduced to first-order formulas at all. Consider the following.

(3.14) ***More*** *boys than girls drink.*

MoreThan is a three-place generalised determiner that takes three sets instead of two. Its definition can be derived from the following equation.

(3.15) $\textsc{MoreThan}^{\star}(\mathit{boy})(\mathit{girl})(\mathit{drink}) = |\mathit{boy} \cap \mathit{drink}| > |\mathit{girl} \cap \mathit{drink}|$

The right-hand side of the equation states that the set of boys that drink has a greater cardinality than the set of girls that drink. The linguistically valid solution of the equation is

(3.16) $\textsc{MoreThan} \equiv \lambda P \lambda Q \lambda R\ |P \cap R| > |Q \cap R|$

Unlike as in previous examples, we cannot give a higher-order formalisation of cardinality comparision such that a definition expansion of MoreThan will produce a first-order formula. The reason for this is a result of the compactness of first-order logics which prohibits us to formulate conditions which distinguish finite and infinite sets. From this, it follows that comparing the cardinality of two first-order predicates that might denote arbitrary sets is not directly expressible in first-order predicate calculus. There is, however, an indirect approach that makes use of a first-order axiomatisation of mathematical set theory. Such an axiomatisation[1] treats sets as first-order entities, and specifies their properties in terms of first-order formulas. Axiomatic set theory can be taken as the basis of a formalisation of mathematics as a whole. Naturally, we would therefore be able to give an expression that captures the semantic of the sentence (3.14). Unfortunately, all models that depend on a first-order axiomatisation of set theory have an infinite universe of discourse because they must have a distinct entity in the universe for every set. Hence, for our purposes, such an approach is not practicable.

What is really disturbing about our discovery here is that the statement (3.16) itself does not look more "difficult" to model in any way than previous examples. Given a situation in which we have n boys and m girls who either drink or do not drink, it should be very easy to verify whether the statement holds or not. Yet, we cannot give a self-contained first-order formalisation that expresses the simple cardinality constraint of the set of boys and the set of girls that we have in our example.

[1] A first-order formalisation of set theory that has been developed specifically for automated deduction can be found in Boyer et al. [59].

3.1.5 A Motivation for a New Kind of Logic

The simply typed λ-calculus gives us a formalism for representing not only the semantics of complete sentences, but also of the semantics of their parts. Functional application and β-reduction provide a simple, yet powerful compositional construction mechanism for semantic representations. By using higher-order definitions for quantifiers and determiners, we obtain compositional representations that still are reducible to a standard first-order format in many cases.

Nevertheless, certain forms of quantification do not have first-order formalisations that are suitable for the application of conventional model generation methods. First-order compactness prohibits even simple cardinality constraints such as needed for the determiner *more*. This is disturbing for our intended application of model generation in the interpretation of natural language.

What we would like to have for natural-language semantics is an expressive language in which we can experiment with higher-order definable concepts, such as generalised determiners, in the usual way. For this, as we have seen, we cannot stay in the domain of first-order logic in all cases. On the other hand, there is, to my knowledge, no model generation method at all that can deal with higher-order logic in any form. The technical difficulties of model generation in the context of higher-order semantics will be disussed in Section 3.2.9. Here, I will sketch only one, namely that conventional higher-order models can often not be reduced to the interpretation of predicate symbols over *small* universes of discourse. The performance of all model generation methods presented in Chapter 2 depends on having small domains of individuals, whether they are derived from the Herbrand interpretation of term occurences or being given adjacentily as in the case of finite model generators. First-order models are often small and simple structures because they must only define the interpretation of predicates and functions over such small domains. In higher-order logics, the argument of a function symbol can be any expression of appropriate type, and quantification can range not only over individuals, but also over entities of higher type. Even in cases where we only have a small number of individuals in our universe of discourse, the number of distinct functions that can be derived from them grows exponentionally with both the number of individuals and the arity of the functions involved. Any model computation that must deal with complete domains of functions instead of just a domain of individuals will inevitably face intractable combinatorial problems.

Our intended application requires that we identify an interesting fragment of higher-order logic that captures the linguistically motivated forms of quantification that we have mentioned earlier as well as being open to efficient methods of model generation. This is possible by considering a higher-order logic where the domains of all entities of arbitrary type can be restricted to finite (and small) subsets of the full domain that is given by the standard semantics. In other words, we aim at a higher-order logic whose notion of quantification is not only explicitely limited to finite domains, but whose range of quantification can further be restricted such that higher-order forms of quantification stay tractable. In a model where the denotations of predicate symbols are guaranteed to be finite sets, we will be able to formalise the truth conditions that we need for certain forms of natural-language quantification, and find computational means to deal with them when generating models.

There is a price that we will have to pay for this additional expressivity: we will no longer be able to formally capture the meaning of infinite concepts or discourse situations with infinitely many participants. For the overwhelming majority of natural-language utterances, this restriction plays no rôle. Our working hypothesis is that a concentration on finiteness is even a necessity for exploring practicable model generation methods in computational semantics.

3.2 Higher-Order Logic

In the following, the syntax and semantics of higher-order logics are presented in order to provide a compact reference. Apart from the notational conventions, most of the introduced concepts are as usual, and are discussed in more detail for instance in Barendregt's textbook on the λ-calculus [60].

3.2.1 Syntax

The simply typed λ-calculus is a formal language whose expressions, i.e., whose terms, are composed from constants, variables, function applications and λ-abstractions. Each of these terms carries a type, i.e., a symbolic annotation which makes sure that the functions defined in our language can only be applied to arguments of the appropriate domain.

3.2.2 Types

We have a set BaseType of **base types** that consists of the **type ι of individuals** and the **type o of truth values**. The set Type of **types** then is defined as the smallest set such that

– BaseType $\subseteq$ Type, and
– if $\alpha \in$ Type and $\beta \in$ Type, then $\alpha \rightarrow \beta \in$ Type.

The **type constructor** $\rightarrow$ is right-associative, and a type of the form $\iota \rightarrow (\iota \rightarrow o)$ for instance will be written as $\iota \rightarrow \iota \rightarrow o$.

Unless indicated otherwise, we will use α and β for denoting arbitrary types, and Φ_α denotes any syntactic or semantic structure Φ whose type is α. The basic expressions of the simply typed λ-calculus are typed constants and typed variables. For these symbols we will use the following notational conventions.

Symbol	Type	Denotation
c, jon, $peter$,...	ι	individual constants
$x, y, z, \ldots$	ι	individual variables
p, love, man,...	$\alpha_1 \rightarrow \ldots \rightarrow \alpha_n \rightarrow o$	predicate constants
P, Q, R,...	$\alpha_1 \rightarrow \ldots \rightarrow \alpha_n \rightarrow o$	predicate variables
C_α, D_α,...	α	constants of arbitrary type
X_α, Y_α, ...	α	variables of arbitrary type

3.2.3 Terms

We assume a countable signature Σ of constant symbols, and a countable set of variables ∇. The set $\textsc{Term}_{\nabla}^{\Sigma}$ of well-formed typed **λ-terms** with respect to Σ and ∇ is the smallest set such that

- if $X_\alpha \in \nabla$, then $X_\alpha \in \textsc{Term}_{\nabla}^{\Sigma}$
- if $C_\alpha \in \Sigma$, then $C_\alpha \in \textsc{Term}_{\nabla}^{\Sigma}$
- if $\mathsf{T}_{\alpha \rightarrow \beta} \in \textsc{Term}_{\nabla}^{\Sigma}$ and $\mathsf{U}_\alpha \in \textsc{Term}_{\nabla}^{\Sigma}$, then $(\mathsf{T}_{\alpha \rightarrow \beta}\ \mathsf{U}_\alpha)_\beta \in \textsc{Term}_{\nabla}^{\Sigma}$
- if $\mathsf{T}_\beta \in \textsc{Term}_{\nabla}^{\Sigma}$ and $X_\alpha \in \nabla$, then $(\lambda X_\alpha\ \mathsf{T}_\beta)_{\alpha \rightarrow \beta} \in \textsc{Term}_{\nabla}^{\Sigma}$

If the type of a well-formed λ-term is given by the context or implied by our notation, we avoid its explicit mention. We will use T, U, and W for denoting arbitrary λ-terms. A term of the form $(\mathsf{T}\ \mathsf{U})$ is a called a **function application**, and one of the form $\lambda X\ \mathsf{T}$ a **λ-abstraction**. To ease readability, we follow the usual convention for λ-terms and leave out brackets in every case where the construction of an expression is uniquely determined.

In a λ-abstraction $\lambda X\ \mathsf{T}$, the variable X is a **bound variable**. The set free(T) denotes the free variables in T, i.e., those variables in T which are not bound. A λ-term T is a **closed λ-term** iff free(T) is empty, otherwise it is open.

Some λ-terms are in a structural equality relation that is induced by **$\beta\eta$-reduction**:

$$(\lambda X\ \mathsf{T})\mathsf{U} \longrightarrow_\beta [\mathsf{U}/X]\mathsf{T} \quad (\lambda X\ \mathsf{T}X) \longrightarrow_\eta \mathsf{T}$$

where X is not free in T, and $[\mathsf{U}/X]\mathsf{T}$ denotes the substitution of all free occurrences of X in T by U. It is well known that the reduction relations β, η, and $\beta\eta$ are terminating and confluent, so that we have unique normal forms for λ-terms. We chose the $\beta\eta$-normal form as the standard syntactical form of all λ-terms.

Depending on which notational variant is more elegant or clear, we will sometimes write $(((\ldots(\mathsf{T}\ \mathsf{U}_n)\ \mathsf{U}_{n-1})\ldots)\ \mathsf{U}_1)$ either as $\mathsf{T}(\overline{\mathsf{U}_n})$, as $\mathsf{T}(\mathsf{U}_1, \ldots, \mathsf{U}_n)$, or as $\mathsf{T}(\mathsf{U}_n)\ldots(\mathsf{U}_1)$. Function application is considered to be left-associative.

3.2.4 Semantics

The simply typed λ-calculus can be taken as the basis of different logics. The semantic of each such logic is defined by the postulated meaning of its logical constants. In what follows, we first describe the interpretation of general λ-terms without considering a logical content.

3.2.5 Functional Interpretations

A **frame** $\{\mathcal{D}_\alpha | \alpha \in \textsc{Type}\}$ is a collection of domains for types α. A **function domain** $\mathcal{D}_{\alpha \to \beta}$ is a domain that consists of functions $f : \mathcal{D}_\alpha \to \mathcal{D}_\beta$. It is common practice to assume that the elements of function domains are total functions. Alternatively, frames can also be based on partial functions, as has been done for instance by Farmer [61].

An **interpretation of the simply typed λ-calculus** with respect to a signature Σ is a pair $\mathcal{I} = \langle [\![\ ]\!], \mathcal{D} \rangle$ where $\mathcal{D}$ is a frame and $[\![\ ]\!]$ is an evaluation that assigns to each constant $C_\alpha \in \Sigma$ an object in $\mathcal{D}_\alpha$.

A **variable assignment** σ for the set of variables ∇ is a mapping from variables $X_\alpha \in \nabla$ into a domain $\mathcal{D}_\alpha$ of objects of appropriate type. A variable assignment $\sigma \cup \{X_\alpha := a\}$ denotes a mapping σ' derived from σ where $\sigma'(Y) = a$ if $Y = X$, and $\sigma'(Y) = \sigma(Y)$ otherwise.

The **denotation** $[\![\mathsf{T}]\!]^\sigma_\mathcal{I}$ for an arbitrary λ-term $\mathsf{T} \in \textsc{Term}^\Sigma_\nabla$ with regard to an interpretation $\mathcal{I} = \langle [\![\ ]\!], \mathcal{D} \rangle$ and a variable assignment σ is recursively defined as follows.

- $[\![C]\!]^\sigma_\mathcal{I} = [\![C]\!]$ if $C \in \Sigma$
- $[\![X]\!]^\sigma_\mathcal{I} = \sigma(X)$ if $X \in \nabla$
- $[\![(\mathsf{U}\ \mathsf{W})]\!]^\sigma_\mathcal{I} = [\![\mathsf{U}]\!]^\sigma_\mathcal{I}([\![\mathsf{W}]\!]^\sigma_\mathcal{I})$
- $[\![(\lambda X_\alpha\ \mathsf{U}_\beta)]\!]^\sigma_\mathcal{I} = f \in \mathcal{D}_{\alpha \to \beta}$ such that $f(a) = [\![\mathsf{U}]\!]^{\sigma \cup \{X\ :=\ a\}}_\mathcal{I}$ for all $a \in \mathcal{D}_\alpha$

We assume in the definition of denotations that we are dealing with well-typed terms, e.g., that $(\mathsf{U}\ \mathsf{W})$ denotes the application of a function term $\mathsf{U}_{\alpha \to \beta}$ to a term W_β. The denotation of a closed term does not depend on the initial variable assignment σ, and we will simply write $[\![\mathsf{T}]\!]_\mathcal{I}$ for the denotation $[\![\mathsf{T}]\!]^\sigma_\mathcal{I}$ of a closed term T with respect to some arbitrary assignment σ.

3.2.6 Logical Constants

A **logic** based on the simply typed λ-calculus is a set of definitions that determines the denotation of the set of **logical constants** in the signature Σ.

In a higher-order logic, we will have at least the logical constants $\vee_{o \to o \to o}$, $\neg_{o \to o}$, and $\Pi_{(\alpha \to o) \to o}$ for all types α. All logical constants are given a postulated meaning, i.e., a meaning that fixes their denotation in all interpretations.

For the purpose of this volume, we assume a family of logical constants in Σ that are distinguished by their type. Depending on our linguistic applications, we will later extend our set of logical constants as needed. For now, we have

- **unary logical connectives** of type $o \to o$, e.g., $\neg$,
- **binary logical connectives** of type $o \to o \to o$, e.g., $\vee$, $\wedge$, $\Rightarrow$, $\Leftrightarrow$,
- **monadic quantifier constants** of type $(\alpha \to o) \to o$, e.g., $\forall$ and $\exists$, for all types α,
- **diadic quantifier constants** of type $(\alpha \to o) \to (\alpha \to o) \to o$, e.g., Every, Some, etc., for all types α, and
- an **equality sign** $=$ of type $\alpha \to \alpha \to o$ for all types α.

The somewhat exotic diadic quantifiers are inspired from the linguistic theory of quantification in natural language where they are known as *generalised determiners*. We will later use the diadic quantifiers for exemplifying how linguistic quantification can be encoded as constraints over finite models.

All constants that are not logical constants are called **parameter constants**, short **parameters**.

3.2.7 Defining a Logic

In order to define a logic, we must first fix a domain of truth values $\mathcal{D}_o$ that appears in each frame $\mathcal{D}$ of the logic. For a classical two-valued logic, we chose the domain $\mathcal{D}_o$ to be $\{0, 1\}$ where 1 denotes truth and 0 denotes falsity.

As mentioned earlier, different logics may be based on varieties of logical constants and denotations for them. The semantic of a logical constant for a certain logic can be defined as a constraint that describes how a formula governed by the constant is to be interpreted relative to the formula's parts or instantiations. For instance, a basic higher-order logic $\mathcal{HOL}$ can be defined by giving the constraints of a minimal set of logical constants $\neg$, $\vee$, and $\Pi_{(\alpha \to o) \to o}$ as follows.

- $[\![\neg \mathsf{T}]\!]^\sigma_\mathcal{I} = 1 - [\![\mathsf{T}]\!]^\sigma_\mathcal{I}$
- $[\![\vee(\mathsf{T}, \mathsf{U})]\!]^\sigma_\mathcal{I} = max([\![\mathsf{T}]\!]^\sigma_\mathcal{I}, [\![\mathsf{U}]\!]^\sigma_\mathcal{I})$
- $[\![\Pi_{(\alpha \to o) \to o}(\mathsf{T})]\!]^\sigma_\mathcal{I} = 1$ iff $[\![\mathsf{T}]\!]^\sigma_\mathcal{I}(u) = 1$ for all $u \in \mathcal{D}_\alpha$

To ease our notation, we will from now on use infix notation whenever a logical constant denotes a binary connective, and adopt the common notation for quantification that is known from first-order predicate calculus. For instance, we will write $\mathsf{T} \vee \mathsf{U}$ instead of $\vee(\mathsf{T}, \mathsf{U})$, and $\forall X\ \mathsf{p}(X)$ instead of $\forall(\lambda X\ \mathsf{p}(X))$ or $\forall(\mathsf{p})$.

The set of standard logical constants that are usually found in higher-order logics can be derived from the basic set above by using **higher-order definitions**, i.e., by λ-terms that replace all occurrences of the defined constants and that make use only of other logical constants whose denotation is already given. The definitions are as follows.

- $\wedge \ \equiv\ \lambda X \lambda Y\ \neg(\neg X \vee \neg Y)$
- $\Rightarrow \ \equiv\ \lambda X \lambda Y\ \neg X \vee Y$
- $\Leftrightarrow \ \equiv\ \lambda X \lambda Y\ (X \Rightarrow Y) \wedge (Y \Rightarrow X)$
- $\forall \ \equiv\ \lambda P_{\alpha \to o}\ \Pi(P)$
- $\exists \ \equiv\ \lambda P_{\alpha \to o}\ \neg \Pi(\neg P)$
- $\textsc{Every} \ \equiv\ \lambda P_{\alpha \to o} \lambda Q_{\alpha \to o}\ \forall X_\alpha\ P(X) \Rightarrow Q(X)$
- $\textsc{Some} \ \equiv\ \lambda P_{\alpha \to o} \lambda Q_{\alpha \to o}\ \exists X_\alpha\ P(X) \wedge Q(X)$
- $= \ \equiv\ \lambda Q_\alpha \lambda R_\alpha\ \forall P_{\alpha \to o}\ P(Q) \Rightarrow P(R)$

A **formula** in $\mathcal{HOL}$ is a λ-term of type o. Unless indicated otherwise, all formulas that we consider are closed. An **atom** in $\mathcal{HOL}$ is a formula $\mathsf{h}(\overline{u_n})$ in $\beta\eta$-normal form that does not have a logical constant as its head h. A **literal** in $\mathcal{HOL}$ is a $\mathcal{HOL}$ atom or its negation.

Our definition of the equality signs $=_{\alpha\to\alpha\to o}$ is a higher-order formalisation of equality that has been proposed by Leibniz: two things are equal iff they have the same properties.

3.2.8 Standard Frames and Generalised Interpretations

So far, we have not given a precise notion of the content of the function domains $\mathcal{D}_{\alpha\to\beta}$ in our interpretations for the simply typed λ-calculus. The standard assumption is that every total function from $\mathcal{D}_\alpha$ to $\mathcal{D}_\beta$ is an element of $\mathcal{D}_{\alpha\to\beta}$. The frames that are given by this construction are called **standard frames**.

In a higher-order logic that is based on standard frames, the denotation f of a λ-abstraction $\lambda X_{\alpha\to\beta}\ \mathsf{U}_\gamma$ is a total function from the complete set of total functions $\mathcal{D}_{\alpha\to\beta}$ into the domain $\mathcal{D}_\gamma$. Standard frames for higher-order logic suffer from the drawback that there can be no calculus that axiomatises logical consequence. In other words, there is no complete proof theory for a higher-order logic based on standard frames. A method for demonstrating this incompleteness of higher-order logic is to give an encoding of arithmetic and applying Gödel's incompleteness theorem [62].

Henkin [63] introduced a weaker notion of higher-order semantics where the domains $\mathcal{D}_{\alpha\to\beta}$ may consist only of a subset of all total functions from $\mathcal{D}_\alpha$ to $\mathcal{D}_\beta$. In order to have a denotation for each λ-term, the lower bound for each of these subsets is the set of all functions of type $\alpha \to \beta$ that are expressible as λ-terms, i.e., as computable functions. The frames obtained in this way are called **generalised frames**, and a **generalised interpretation** is an interpretation $\mathcal{I} = \langle [\![\ ]\!], \mathcal{D}\rangle$ whose frame $\mathcal{D}$ is a generalised one.

In a higher-order logic that is based on generalised interpretations, sound and complete proof theories for the classical consequence relation can be given. This even leads to mechanisable calculi, e.g., the variant of extensional higher-order resolution by Benzmüller and Kohlhase [64]. In recent years, Henkin's semantic for higher-order logic has become the standard theoretical basis of higher-order automated deduction.

3.2.9 Model Generation for Generalised Frames?

We call an interpretation $\mathcal{I} = \langle [\![\ ]\!], \mathcal{D}\rangle$ a **model of a set of formulas** Φ if $[\![\mathsf{F}]\!]_\mathcal{I} = 1$ for all formulas F in Φ.

It would be very desirable to have a model generation method for a higher-order logic whose semantic is based on Henkin's generalised interpretations. A model generator for higher-order logic would take a finite set Φ of $\mathcal{HOL}$ formulas as its input and determine interpretations $\mathcal{I} = \langle [\![\ ]\!], \mathcal{D}\rangle$ such that $[\![\mathsf{F}]\!]_\mathcal{I} = 1$ for all F in Φ. Finite model generation for higher-order logic is possible at least in theory because all function domains $\mathcal{D}_{\alpha\to\beta}$ that are based on finite basic domains $\mathcal{D}_\iota$ and $\mathcal{D}_o$ are finite themselves, as has been shown by Andrews [65]. The functions f from a finite domain $\mathcal{D}_\alpha$ into a finite domain $\mathcal{D}_\beta$ have a finite representation, for instance as a table. Hence, we could enumerate all models for a specification Φ simply by enumerating all possible interpretations $\mathcal{I}$ of the constants C that occur in the input and verifying in finite time whether the interpretation is a model or not. Of course, such a method would be intractable in general for all except trivial input specifications.

If we could design a more efficient method, higher-order model generation could be put to use for instance to compute counterexamples in higher-order automated theorem proving. Such a method, if it exists, would also be very handy for exploring the meaning of natural-language semantic representations that have higher-order logical forms, i.e., for the topic of research that this volume is about. Unfortunately, there is some evidence that model generation for generalised models is not mechanisable by using the technical machinery that is available to us.

3.2.10 Equivalency for Higher-Order Atoms

In first-order logics, finite model generation methods can be reduced to methods that compute the interpretation of predicate symbols over some finite domains of first-order entities. A model is determined completely by the interpretation of a set of atoms of the form $p(a_1, \ldots, a_n)$ where the a_n are symbolic, i.e., are constants, ground terms or entities of a finite domain. In the case of SATCHMO-like model generation theorem proving, the a_j can be complex terms t, but these are interpreted as themselves and one does not have to consider the interpretation of function symbols. Hence, it is trivial to decide whether two ground literals of the form $p(a_1, \ldots, a_n)$ and $\neg p(b_1, \ldots, b_n)$ are contradictory. The efficiency of first-order model generation methods relies on having a simple way for detecting such elementary inconsistencies while they construct partial interpretations.

In the simply typed λ-calculus, the arguments of predicates can be arbitrarily complex terms or even formulas. Unlike as in first-order logics, the equality of embedded terms cannot be reduced to some syntactical equality relation. For instance, in a higher-order logic $\mathcal{HOL}$ that is based on generalised frames, the following formulas are **logically equivalent**, i.e., must have the same denotation in all models.

$$p_{o \to o}(a_o) \tag{3.17}$$

$$p_{o \to o}((\lambda x\ a_o)(c_\iota)) \tag{3.18}$$

$$p_{o \to o}(a \wedge (b \vee \neg b)) \tag{3.19}$$

$$p_{o \to o}((b \vee \neg b) \wedge a) \tag{3.20}$$

$$p_{o \to o}(\exists x\ ((q(x) \vee a) \vee \neg q(x))) \tag{3.21}$$

The formulas (3.17) and (3.18) have the same $\beta\eta$-normal form. Their logical equivalency is therefore given by the syntactically determined equivalency of formulas induced by $\beta\eta$-reduction. In contrast to this, a purely syntactical approach is not sufficient for proving the equivalency of (3.17) to all other formulas. The equivalency is given only semantically by the denotations of the logical constants. In our examples, the proof problems are quite simple, but as embedded terms can be arbitrarily complex formulas, the resulting proof problems easily become undecidable. If we have two literals $\mathsf{p}(\mathsf{T})$ and $\neg\mathsf{p}(\mathsf{U})$, we cannot decide in general whether they form an elementary contradiction or not.

There is an analogous problem in higher-order automated theorem proving where the first-order variant is efficiently computable, while the higher-order instance of the same operation is undecidable. The operation we refer to is *unification* which is essential for finding non-trivial proofs in many proof procedures. In higher-order automated theorem proving, unification problems are frequently not solved by simply calling a procedure, but are treated as constraints of the proof problem and thus become part of the overall proof search. By this, one avoids to call an external unification procedure which, in general, might not terminate. In the same way, one could treat the equivalency problem in higher-order model generation as a set of constraints: if two formulas $\mathsf{p}(a_1, \ldots, a_n)$ and $\neg\mathsf{p}(b_1, \ldots, b_n)$ have been computed as part of a partial interpretation, we could simply add some additional formulas to the input specification that make sure that the tuples $\langle a_1, \ldots, a_n\rangle$ and $\langle b_1, \ldots, b_n\rangle$ do not have the same denotation. For instance, if we have two formulas $\mathsf{p}_{o\to o}(a_o)$ and $\neg\mathsf{p}_{o\to o}(b_o)$ in a partial interpretation, we prevent a and b from having the same denotation by adding the formula $\neg(a \Leftrightarrow b)$ to our input. In any model that we obtain for the extended specification, a and b must denote different truth values. In cases where our additional conditions can not be satisfied, we will not be able to compute a model, as required.

Nevertheless, a treatment of equivalence in higher-order models still is intractable in general. If a partially determined interpretation contains n positive occurrences of a formula with head p and m negative occurrences, we have $n \times m$ pairs that generate additional formulas for the input problem. These formulas might further add to the complexity of the model generation problem, because the arguments of a predicate could be formulas that introduce new equivalency conditions themselves. It seems that a simple and efficient treatment of the basic formulas of a logic such as we have in first-order finite model generation is not possible for higher-order logics.

3.2.11 Function Domains and Quantification

One of the criteria in first-order finite model generation for the complexity of a model generation task is the number of ground atoms whose interpretation must be determined. This number is roughly the same for all methods based on propositional satisfiability procedures such as the Davis-Putnam procedure. While only being a crude method for classifying model generation problems, a smaller number of atoms generally indicates problems that are easier to solve. Experiments in propositional planning show that the state-of-the-art exhaustive SAT procedures can handle some hundreds of atoms while methods that employ local probabilistic search sometimes find propositional models with several thousand atoms whose interpretation must be determined [66].

The number of ground atoms that are derived from a first-order model generation problem depends on two orthogonal parameters. First, the number of atoms usually grows with the complexity of the input. A specification that consists of a large number of formulas will in general be harder to treat than one that only consists of a few, although it is quite simple to find notoriously hard problems that have small formalisations. Second, the number of ground atoms derivable from a first-order specification can grow exponentionally in the size of the universe of discourse. For certain problems in quasi-group theory, the search space grows with such speed that even particulay efficient systems like FINDER are able to search exhaustively only in a domain size of a dozen elements.

If the number of atoms that must be considered in an interpretation is taken as an indicator for a problem's complexity, then higher-order model generation for generalised frames will usually be much harder than first-order model generation. Even in cases where we have a small domain $\mathcal{D}_\iota$ of individuals, the number of functions that can be defined from them is quite large. In the presence of higher-order quantification, we can easily formulate statements over the function domains $\mathcal{D}_{\alpha\to\beta}$ that are intractable. As an example, consider a first-order domain $\mathcal{D}_\iota$ of size n, and the domain of diadic relations $\mathcal{D}_{\iota\to\iota\to o}$. The number of different relations in $\mathcal{D}_{\iota\to\iota\to o}$ is 2^{n^2}. In order to build a model for a formula of the form $\forall R_{\iota\to\iota\to o} P(R)$, we will have to make sure that all the 2^{n^2} instantiations of $P(R)$ are true. In a worst-case scenario, P itself could be a formula that quantifies over some function domain.

3.3 A Fragment of Higher-Order Logic

As discussed in Section 3.2.9, a higher-order logic that is based on Henkin semantics cannot be given a practically useful model generation method that uses the same techniques that work for first-order model generation. The size of the function domains $\mathcal{D}_{\alpha\to\beta}$ in generalised frames are an obstacle for a computational treatment of quantification, and an unrestricted syntax will leave us with formulas where even basic contradictions in atoms cannot easily be detected. The solution that I propose for the first problem is to quantify only over small subsets of the function domains $\mathcal{D}_{\alpha\to\beta}$ instead of considering the whole function space. The second problem is attacked by a restricted syntax where the ground atoms that must be considered in the computation of a denotation have a simple syntactic structure. By furthermore adopting a Herbrand-like interpretation for constant symbols, we obtain a linguistically motivated logic $\mathcal{MQL}$ (**M**ontague-style **Q**uantification **L**anguage) that combines the compositionality of a higher-order logic with the compactly representable interpretations of a first-order logic.

3.3.1 Syntax

A formula F of the simply-typed λ-calculus is called **function-free quantified** iff for all atoms $h(\overline{u_n})$ that occur in F, each u_j is either a variable or a constant. A well-formed formula in $\mathcal{MQL}$ is a closed function-free quantified formula of the simply-typed λ-calculus.

The function-free quantified format does not have complex λ-expressions as arguments of non-logical constants. This means that the $\beta\eta$-normal form of an argument u_j in an atom $h(\overline{u_n})$ must always be a symbol. Note that while every atom itself must of course be well-typed, the arguments may be of arbitrary type.

Figure 3.1 gives some examples for well-formed $\mathcal{MQL}$ formulas and expressions that might occur as semantic representations of natural language. The logical encodings are straightforward. In examples 7–10, we have $\mathcal{MQL}$ formulas that formalise properties of higher-order objects. The constant *color* denotes a second-order predicate of type $(\iota \to o) \to o$, and *color*(*red*) specifies that the constant *red* denotes a first-order set has the property of being a color. In a first-order logic, one would have to reify colors as first-order entities, and could not use them at the same time as predicate symbols and arguments of other predicates.

1	Natural Language	$\mathcal{MQL}$ Representation
2	loves Ken	$\mathit{love}(ken)$
3	loved by Ken	$\lambda x\ \mathit{love}(x)(ken)$
4	loves him/herself	$\lambda x\ \mathit{love}(x)(x)$
5	Every man loves Ken	$\textsc{Every}(\mathit{man})(\mathit{love}(ken))$
6	Ken (as quantifier)	$\lambda P\ P(ken)$
7	Ken loves Ken	$(\lambda P\ P(ken))(\mathit{love}(ken))$
8	Some apples are red	$\textsc{Some}(\mathit{apple})(\mathit{red})$
9	Red is a color	$\mathit{color}_{(\iota\to o)\to o}(\mathit{red})$
10	Some colors are primary colors	$\textsc{Some}(\mathit{color}_{(\iota\to o)\to o})(\mathit{primary})$

Fig. 3.1. Examples for well-formed $\mathcal{MQL}$ expressions

The syntactic restriction that we have here is very similar to that of the EP calculus (cf. 2.4.5) that also uses a function-free quantified input format where no complex term may occur as an argument of a predicate symbol[2]. EP's syntax is motivated primarily by the need to avoid function symbols in the context of Herbrand interpretations. Our main motivation instead is to have only atoms $h(\overline{u_n})$ in the recursive computation of a denotation that have a simple syntactical structure. We do not have to care for embedded function expressions whose denotation must be considered for instance when we check for basic contradictions in a partial interpretation.

[2] This format is not to be confused with the function-free *fragment* of first-order logic, i.e., the set of formulas without function symbols whose prenex form starts with an initial sequence of universal quantifiers followed by a sequence of existential ones [14]. EP's input normal form is equivalent to full first-order logics, while the function-free fragment is decidable.

3.3.2 Semantics

We cannot restrict the function domains $\mathcal{D}_{\alpha \to \beta}$ in a frame at will without potentially violating the **Denotatpflicht**, i.e., the requirement that each expression that is necessary for computing a denotation has a denotation itself.

In order to meet the Denotatpflicht in Henkin's construction of generalised frames, each function domain includes all functions of appropriate type that can be expressed as λ-terms—this property is provided by the so-called **comprehension axioms** [63]. In principle, all functions that are expressible must be made available, as Henkin's semantics must assign a meaning to arbitrary terms. In our simpler syntax, we may not encounter complex functional expressions that must be given an interpretation. In the recursive interpretation of a closed $\mathcal{MQL}$ formula, the Denotatpflicht is already met if we can assign a meaning to the formula's components and instantiations down to the level of simple, ground atoms.

The intuition behind the construction below is that we can further simplify the interpretation if all u_i in each atom $h(\overline{u_n})$ are interpreted "as themselves". The interpretations that we obtain in this way are term interpretations, i.e., a constant c_α that occurs as an argument will be interpreted as some "individual" c of type α rather than some object of the domain $\mathcal{D}_\alpha$. Higher-order quantification and the Denotatpflicht for our term interpretations require that there are appropriate sets of individuals of type α available. These sets are provided in our formal framework by constant frames.

3.3.3 Constant Frames

A **constant frame** $\mathcal{C}$ is a collection $\{\mathcal{C}_\alpha | \alpha \in \textsc{Type}\}$ of sets $\mathcal{C}_\alpha$ that obeys the following conditions.

- $\mathcal{C}_\iota \in \mathcal{C}$, $\mathcal{C}_o \in \mathcal{C}$
- $\mathcal{C}_\iota$ is not empty
- $\{0,1\} \subseteq \mathcal{C}_o$

A **constant frame** $\mathcal{C}_\Phi$ for an $\mathcal{MQL}$ specification Φ is a constant frame where all parameter constants C_α that occur in Φ are elements of $\mathcal{C}_\alpha \in \mathcal{C}$. A constant frame $\mathcal{C}_\Phi$ may also contain additional constants C_α that do not occur in Φ. A constant frame is an **initial constant frame** $\mathcal{C}^0_\Phi$ if it contains only the minimum number of constants with regard to Φ. A **finite constant frame** is a constant frame where all sets $\mathcal{C}_\alpha$ are finite. Obviously, all initial constant frames are finite.

3.3.4 Interpretations and Denotations

What we will do in the following is to define a classical two-valued semantic for our logic $\mathcal{MQL}$ that is based on the concept of a finite constant frame $\mathcal{C}_\Phi$.

Let $\mathcal{C}_\Phi$ be a finite constant frame for a logical specification Φ. An $\mathcal{MQL}$ interpretation $\mathcal{I} = \langle [\![\ ]\!], \mathcal{C}_\Phi \rangle$ for Φ is an evaluation of all constants in $\mathcal{C}_\Phi$ where the evaluation function $[\![\ ]\!]$ obeys the following conditions:

- $[\![1]\!] = 1$, $[\![0]\!] = 0$
- $[\![c_o]\!] \in \{0,1\}$ for all Boolean parameters c_o
- $[\![\mathsf{h}_{\alpha_1 \to \ldots \to \alpha_n \to o}]\!]$ is a function $f : \langle \mathcal{C}_{\alpha_1}, \ldots, \mathcal{C}_{\alpha_n} \rangle \to \{0,1\}$ for all n-ary predicate parameters h

An interpretation $\mathcal{I}_\Phi$ for a specification Φ determines the denotation of all parameter constants in $\mathcal{C}_\Phi$ as needed for defining a denotation of an $\mathcal{MQL}$ formula later on. As usual, we furthermore assume that the logic itself determines an evaluation $[\![C]\!]$ of logical constants C in all interpretations.

Unlike conventional interpretations in higher-order logics, a $\mathcal{MQL}$ interpretation does not give a denotation for individual constants or function constants that are not predicates. This is actually not necessary because we will adopt a Herbrand-like interpretation of constant symbols in cases where constants appear as arguments of predicates. A Herbrand-like interpretation of constants is particulary attractive for applications in linguistics, as has been noted for instance by Baumgartner and Kühn [67]. In logical representations of natural language discourse, we often have the convention that individual constants are interpreted as names for distinct individual entities (e.g., *jon*, *mary*, etc.). The **unique-name assumption**, i.e., the practice of treating different individual constants as different individuals, is one of the basic constraints of interpretation in many logic-based linguistic formalisms. Our logic $\mathcal{MQL}$ simply generalises this idea to constants of higher type.

Let $\mathcal{I} = \langle [\![\ ]\!], \mathcal{C}_\Phi \rangle$ be an interpretation for a specification Φ. The **denotation $[\![\mathsf{F}]\!]_\mathcal{I}$ of a $\mathcal{MQL}$ formula in Φ** is recursively defined as follows.

- $[\![\mathsf{c}_o]\!]_\mathcal{I} = [\![\mathsf{c}_o]\!]$ for all Boolean constants c_o in $\mathcal{C}_\Phi$
- $[\![\mathsf{h}(\overline{u_n})]\!]_\mathcal{I} = [\![\mathsf{h}]\!](\overline{u_n})$ for all predicate parameter constants h in $\mathcal{C}_\Phi$
- $[\![\circ \mathsf{F}\ ]\!]_\mathcal{I} = [\![\circ]\!]([\![\mathsf{F}]\!]_\mathcal{I})$ for all unary logical connectives $\circ$
- $[\![\mathsf{F}_1 \circ \mathsf{F}_2]\!]_\mathcal{I} = [\![\circ]\!]([\![\mathsf{F}_1]\!]_\mathcal{I}, [\![\mathsf{F}_2]\!]_\mathcal{I})$ for all binary logical connectives $\circ$
- $[\![\mathrm{Q}(\mathsf{T}_{\alpha \to o})]\!]_\mathcal{I} =$
  $[\![\mathrm{Q}]\!](\langle [\![(\mathsf{T}\ p_1)]\!]_\mathcal{I}, \ldots, [\![(\mathsf{T}\ p_n)]\!]_\mathcal{I} \rangle)$ for all monadic quantifier constants Q and all $p_j \in \mathcal{C}_\alpha$.
- $[\![\mathrm{Q}(\mathsf{T}_{\alpha \to o})(\mathsf{U}_{\alpha \to o})]\!]_\mathcal{I} =$
  $[\![\mathrm{Q}]\!](\langle [\![(\mathsf{T}\ p_1)]\!]_\mathcal{I}, \ldots, [\![(\mathsf{T}\ p_n)]\!]_\mathcal{I} \rangle)(\langle [\![(\mathsf{U}\ p_1)]\!]_\mathcal{I}, \ldots, [\![(\mathsf{U}\ p_n)]\!]_\mathcal{I} \rangle)$ for all diadic quantifier constants Q and all $p_j \in \mathcal{C}_\alpha$.

The recursion is a bit different from that of conventional higher-order logics. The crucial points are as follows.

First, the denotation of an atom $\mathsf{h}(\overline{u_n})$ is that of the interpretation $[\![\mathsf{h}]\!]$ of the head predicate symbol h applied directly to the arguments $\overline{u_n}$. In a recursive computation of a denotation, we now have only atoms $\mathsf{h}(\overline{u_n})$ where all u_j are constants that actually are interpreted as themselves. The situation is analogous to a first-order Herbrand interpretation where embedded terms t that denote individuals are interpreted as themselves, i.e., $[\![t]\!] = t$. Unlike first-order logics, however, our constants u_j can have any type and might be interpreted as predicates when they occur in a predicate position.

Second, the denotation of quantified formulas is that of the interpretation $[\![\mathrm{Q}]\!]$ of the quantifier constant Q applied to tuples of truth values that come from instantiations over the constant sets $\mathcal{C}_\alpha$. Quantification in $\mathcal{MQL}$ does not range over domains $\mathcal{D}_\alpha$, but only over sets of constants $\mathcal{C}_\alpha$ that name entities of type α. Hence, the truth of a quantified formula is determined by the denotations of the formula's instantiations using constants of appropriate type. A set of constants $\mathcal{C}_\alpha$ that is used for quantification can be small, as long as it contains at least the constants of type α that are part of the input specification Φ that we are about to interpret. Informally, a quantified formula $\Pi P_{\alpha\to o}\ Q(P)$ is interpreted as a specification of properties Q over a set of predicates P that have been given names in the specification Φ. Additionally, $Q(P)$ may also influence the interpretation of other named properties P that do not occur in Φ. We may chose the domains of quantification $\mathcal{C}$ as large as we want to as long as they remain finite. However, we can always start with an initial constant frame $\mathcal{C}$ that is much smaller than a generalised frame $\mathcal{D}$, and whose domains of quantification only includes a basic set of distinguished entities. Their existence is implied by giving them distinct names in the specification Φ.

3.3.5 An $\mathcal{MQL}$ Logic

Our logical language $\mathcal{MQL}$ contains the usual set of logical constants that we find in the same form in conventional higher-order logics such as $\mathcal{HOL}$. Additionally, we may add logical constants that are motivated by our linguistic application, namely certain universal determiners whose semantic cannot be given conveniently in the form of a higher-order definition.

Like the λ-calculus, $\mathcal{MQL}$ can be taken as the basis of different actual logics, all of them derived from different sets of logical constants and interpretations for them. As usual, we assume that $\mathcal{MQL}$ determines for each logical constant one denotation that is mandatory in all interpretations.

3.3.6 Connectives

Because unary and binary connectives are predicates, they must be interpreted now as functions from sets of Boolean constants $\mathcal{C}_t ypeo$ into the set $\{0,1\}$. This has no real consequences because we already include the truth values 0 and 1 into the set $\mathcal{C}_o$ of each constant frame. We can therefore keep the interpretation of unary and binary logical connectives as is, i.e., the functions f that denote the logical connectives in a conventional higher-order logic $\mathcal{HOL}$ work analogously in $\mathcal{MQL}$.

For $\mathcal{HOL}$, we have only given some arithmetic constraints for the semantics of the basic logical connectives $\neg$ and $\vee$. The full set of constraints that defines the denotation of the standard set of logical connectives in all $\mathcal{MQL}$ interpretations is as follows.

- $[\![\neg]\!](x) = 1 - x$
- $[\![\vee]\!](x, y) = max(x, y)$
- $[\![\wedge]\!](x, y) = min(x, y)$
- $[\![\Rightarrow]\!](x, y) = min(1 - x, y)$
- $[\![\Leftrightarrow]\!](x, y) = 1 - |x - y|$

The x and y are the truth values 0 or 1 that are provided by the recursive computation of the denotation of a formula.

3.3.7 Quantifiers

The interpretation of quantifier constants in $\mathcal{MQL}$ must be adapted slightly in comparison to $\mathcal{HOL}$. The denotation of a quantifier constant is now a function from one or two tuples of truth values into a truth value. The tuples of truth values come from the denotation of formulas, and the size of the tuples is always finite. Our notion of quantification is obviously different from that of higher-order logic and first-order logic because quantification in $\mathcal{MQL}$ is restricted to finite sets. However, due to this restriction we are more free to define forms of quantification that are hard to capture in first-order logics. Quantifier constants can now be given an operational semantic by defining their denotations as computable functions over finite sets of truth values. For instance, the semantic of the basic quantifier constants $\Pi_{(\alpha \mapsto o) \mapsto o}$ could be given as follows.

(3.22) $[\![\Pi]\!](\langle k_1, \ldots, k_n \rangle) = min(\langle k_1, \ldots, k_n \rangle)$

Here, *min* denotes the function that selects from a tuple of integers the smallest one. Given a set of truth values $k_j \in \{0, 1\}$, we obviously can compute the value of $[\![\Pi]\!](\langle k_1, \ldots, k_n \rangle)$. The definition (3.22) expresses a constraint in the sense that it relates $[\![\Pi]\!](\langle k_1, \ldots, k_n \rangle)$ to a set of variables k_j.

From Π, we can derive the full set of standard quantifier constants[3] by higher-order definitions as usual. Alternatively, we can give constraints over finite tuples of truth values to define denotations such as the following.

(3.23) $[\![\forall]\!](\langle k_1, \ldots, k_n \rangle) = min(\langle k_1, \ldots, k_n \rangle)$

(3.24) $[\![\exists]\!](\langle k_1, \ldots, k_n \rangle) = max(\langle k_1, \ldots, k_n \rangle)$

(3.25) $[\![\textsc{Every}]\!](\langle k_1, \ldots, k_n \rangle)(\langle l_1, \ldots, l_n \rangle) = 1$ *iff* $k_j \leq l_j$ *for all* $i \leq n$

(3.26) $[\![\textsc{Some}]\!](\langle k_1, \ldots, k_n \rangle)(\langle l_1, \ldots, l_n \rangle) = 1$ *iff* $k_j = l_j = 1$ *for some* $i \leq n$

The encoding of truth conditions into computable functions allows us to define the operational semantics of quantifiers that otherwise could not be formalised. For instance, we want a formula $\mathsf{F} = \textsc{More}(\mathsf{T}_{\alpha \mapsto o})(\mathsf{U}_{\alpha \mapsto o})$ to be true iff the denotation of T has more elements than that of U. We know that F is true in an interpretation if more formulas $\mathsf{T}(C)$ than formulas $\mathsf{U}(C)$ are true

[3] We use the standard first-order quantification in order to improve readability.

for the C that are in $\mathcal{C}_\alpha$. Hence, the sum of the denotations $k_j = [\![\mathsf{T}(C_j)]\!]_\mathcal{I}$ must be larger than that of the denotations $l_j = [\![\mathsf{U}(C_j)]\!]_\mathcal{I}$ for all $C_j \in \mathcal{C}_\alpha$. This condition is expressed by the following constraint.

(3.27) $[\![\textsc{More}]\!](\langle k_1, \ldots, k_n\rangle)(\langle l_1, \ldots, l_n\rangle) = 1$ *iff* $\sum_{j=1}^{n} k_j > \sum_{j=1}^{n} l_j$

In first-order and higher-order logics, constraints such as (3.27) do not define computable denotations for quantifiers, because the domains of quantification can be infinite, and infinitely many truth values k_j and l_j may have to be considered. The More quantifier in $\mathcal{MQL}$ has a computable denotation for all interpretations, but only because we have sufficiently weakened the semantic of our logical language to finite domains.

3.3.8 Definitions

The collection of quantifier constants in $\mathcal{MQL}$ can be extended simply by defining new ones as λ-terms. A definition expansion with subsequent $\beta\eta$-conversion eliminates all occurrences of the defined constants which therefore do not need to have a denotation of their own in an interpretation.

A motivation for having definable logical constants is that we would for instance like to experiment with alternative formalisations of linguistic quantifiers in the context of semantic analysis. For instance, we have a diadic linguistic quantifier where for instance One(*man*)(*drink*) is true iff there is exactly one $x \in \mathcal{C}_\iota$ such that both *man*(x) and *drink*(x) are true. Hence, its denotation in $\mathcal{MQL}$ must be as follows.

(3.28) $[\![\textsc{One}]\!](\langle k_1, \ldots, k_n\rangle)(\langle l_1, \ldots, l_n\rangle) = 1$ *iff there is exactly one i such that* $k_j = l_j = 1$

The quantifier One can be implemented simply by a definition that formalises the truth conditions in (3.28). The higher-order definition scheme for quantifiers One of arbitrary type $(\alpha \to o) \to (\alpha \to o) \to o$ looks as follows.

(3.29) $\textsc{One} \equiv$
$\lambda P \lambda Q\ \exists X (P(X) \wedge Q(X)) \wedge \forall Y\ (P(X) \wedge Q(X)) \Rightarrow X \doteq Y$

For reasons of simplicity, $\mathcal{MQL}$'s logical constants only include monadic and diadic quantifier constants, but the device of definitions allows us to formalise other forms of quantifications over finite sets as well. For instance, in Section 3.1.4, we considered the example (3.30) where the cardinality of two sets is part of the truth condition of a three-place quantifier.

(3.30) $\textsc{MoreThan}(\mathit{boy})(\mathit{girl})(\mathit{drink}) = |\mathit{boy} \cap \mathit{drink}| > |\mathit{girl} \cap \mathit{drink}|$

The quantifier constant MoreThan has a simple definition in $\mathcal{MQL}$ that makes use the More quantifier.

(3.31) $\textsc{MoreThan} \equiv$
$\lambda P \lambda Q \lambda R\ \textsc{More}(\lambda x\ P(x) \wedge R(x))(\lambda x\ Q(x) \wedge R(x))$

We will sometimes approximate the meaning of generalised determiners whose truth conditions cannot be exactly captured in our logic. An example is the universal determiner MOST whose meaning is influenced by pragmatic considerations. That is, the choice of which truth conditions actually hold if a sentence like *most men lie* is true lies to some extend with the speaker. However, a formula $\textsc{Most}(P)(Q)$ in general induces at least that the majority of elements in P will also have the property Q. This truth condition is formalised in the following higher-order definition, again with the help of the MORE quantifier.

(3.32) $\textsc{Most} \equiv$
$\lambda P \lambda Q\ \textsc{More}(\lambda x\ P(x) \wedge Q(x))(\lambda x\ P(x) \wedge \neg Q(x))$

While definitions often allow us to formulate complex truth conditions in a compact and human-oriented way, the technical machinery that later deals with the expanded logical forms can be kept very simple by considering only a basic set of logical constants from which others are derived.

3.3.9 Equality

The equality signs $=_{\alpha \to \alpha \to o}$ in our higher-order logic $\mathcal{HOL}$ have been given a semantic via the following definition scheme.

(3.33) $= \equiv\ \lambda Q_\alpha \lambda R_\alpha\ \forall P_{\alpha \to o}\ P(Q) \Rightarrow P(R)$

In $\mathcal{MQL}$, the same formalisation (3.33) defines a form of equality that is determined only by those properties P in $\mathcal{D}_{\alpha \to o}$ that have a representative constant h in the set $\mathcal{C}_{\alpha \to o}$ with $[\![h]\!] = P$. This implies that the content of our constant frame $\mathcal{C}_\Phi$ plays an important rôle in the way in which equality is interpreted. Syntactically, we cannot use the definition of $=$ to compare arbitrary terms because the function-free quantified format prohibits formulas $P(Q)$ where P is a parameter and Q is not symbolic. Hence, a formula $X = Y$ in $\mathcal{MQL}$ is only well-formed if both X and Y are constants or bound variables.

For practical purposes, we will often use a simpler form of equality, the **Herbrand equality** $\doteq$, that identifies its two arguments X and Y as equal if their ground instantiations are syntactically identical. Herbrand equality obeys the rules of parameter constants, i.e., its arguments must be symbolic. Its semantic is as follows.

(3.34) $[\![\doteq]\!](C_\alpha)(D_\alpha) = 1$ *iff* C *and* D *are the same constant*

3.4 Constructing Models

An interpretation $\mathcal{I}$ in $\mathcal{MQL}$ depends on the adjacentily given semantics of the logical constants and on the interpretation of the constants that occur as predicate symbols. Constants that occur as arguments of non-logical constants have a fixed interpretation as themselves as specified in Section 3.3.4. Variables play no rôle because we only consider closed formulas.

Given an instantiation of a logic as a set of constraints that determines the interpretation of logical constants, and an algorithm that instantiates the recursive denotation scheme in Section 3.3.4, a $\mathcal{MQL}$ interpretation is determined completely by the interpretation of predicate symbols over the sets of constants in a finite constant frame $\mathcal{C}_{\Phi}$. The number of different interpretations for a finite set of formulas Φ and a finite constant frame $\mathcal{C}_{\Phi}$ is finite itself. Hence, we could enumerate all possible models for Φ in a given constant frame by a brute-force algorithm that simply enumerates all interpretations $\mathcal{I}$ and checks whether $\mathcal{I}$ is a model for the formulas of Φ. Such a brute-force algorithm is bound to fail for efficiency reasons already for propositional theories, and we would certainly not be able to compute many models for interesting logical specifications that way.

3.4.1 Determining Models Intelligently

Formulas can be used for restricting the search space for valid interpretations. The intuition of this approach has already been hinted at when we defined the interpretations of logical constants as arithmetic constraints. The general idea is based on treating formulas as constraints over variables that range over truth values, and to use efficient techniques for solving such constraints.

3.4.2 Formulas as Constraints

Each complex formula can be seen as a constraint that defines how the denotation of a formula depends on the denotations of its sub-formulas or instantiations. Consider for instance the propositional formula $\mathsf{F} = (a_o \vee b_o) \wedge c_o$. Following the recursive definition of a denotation that has been given in Section 3.3.4, each model $\mathcal{M}$ for F and some arbitrary constant frame $\mathcal{C}$ must obey the following conditions:

$$\begin{aligned}
[\![(a \vee b) \wedge c]\!]_{\mathcal{M}} &= min([\![a \vee b]\!], [\![c]\!]) \\
[\![a \vee b]\!]_{\mathcal{M}} &= max([\![a]\!], [\![b]\!])
\end{aligned}$$

The interpretation of a, b, and c is constrained by the fact that each must be interpreted as a truth value. Hence, the following set of equations characterise all models $\mathcal{M}$ of F:

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

3.4.3 Solving Constraints

In order to enumerate the models $\mathcal{M}$ for F, we could enumerate all evaluation functions $[\![\ ]\!]$ of the boolean constants in F and check whether the set of equations is satisfied. Alternatively, we could instead try to solve the set of (in)equations that we have computed by expanding the semantic of the logical constants. Each solution to this set corresponds to a valid interpretation of the Boolean constants and hence denotes a model.

Efficiently solving sets of linear (in)equations over finite-domain integer variables or, more generally, any computable set of relations between finite sets of variables, is the realm of constraint solving. Decision procedures for classical propositional logics can be seen as specialised constraint solvers for variables that can only have two values. For instance, the Davis-Putnam procedure is an efficient constraint solver that computes the instantiation of a finite set of boolean variables, i.e., atoms, whose dependencies are given by ground clauses.

The set of inequations given in the last section is an instance of the so-called Integer Programming (IP) Problem. IP has applications in operations research and economics [68], and solving inequations over finite-domain integer variables enjoys a great deal of interest in the constraint solving community. There are some off-the-shelf constraint solvers available that can deal with large sets of inequations in practice. Some of these come as modules for programming languages such as the finite-domain integer package of the constraint programming language Oz [69].

The general idea of a is to have a two-part decision procedure. The first part is responsible for constraint propagation. Propagation refers to a process that first partially determines or restricts the value of variables out of the information that is encoded in a set of constraints and the uses this partly determined instantiation for inferring new constraints or simplifying old ones. For instance, in the Davis-Putnam procedure presented in Section 2.4.6, the rule for unit resolution is a typical propagation rule in that it uses the information given by a unit clause for simplifying all clauses in which this unit occurs. The variable that is determined here is the interpretation of the unit, and this determined value is used for simplifying the constraints given by other ground clauses. For IP and other constraint systems, the propagation problem is a bit more complicated because the value of variables can range over many values instead of only two as in the case of propositional logic.

The second part of a constraint solving procedure is distribution. In general, distribution refers to a selection of a variable and a provisional restriction of its range in order to get new information that can be used for propagation. For completeness, a kind of backtracking mechanism must be able to reset the choice of restriction if it does not lead to a solution. In logical inference systems that have an analytical cut, the cut rule formulates a form of distribution. In the Davis-Putnam procedure we have such a cut-rule that selects a literal and splits into the case where the literal is interpreted as true and one in which it is interpreted as false.

An important paradigm of constraint solving is to delay distribution and to use as much information as possible for adding new constraints on the possible values of variables. The idea behind this is that it is very costly in general to backtrack a wrong decision while the efforts for propagation will pay off in the end by leading to more educated guesses. An important advantage of a larger set of constraints in practice is that we might be able to determine more precisely which variables are important and which are not. Distribution over a variable that occurs in many constraints will also help to simplify many more constraints in the end than determining a variable that only occurs in few or even only one constraint.

For the logical language $\mathcal{MQL}$, we do not have an off-the-shelf constraint solver that is comparable to the Davis-Putnam procedure for ground clause sets. We could give a translation that maps $\mathcal{MQL}$ formulas over a given constant frame into sets of inequations whose variables are the interpretations of ground formulas. Such a set of equations can then be solved by a standard IP constraint solver. However, there are constraint solvers available that can already deal more efficiently with the kind of constraints that we find expressed in logical formulas, and whose constraint languages are more convenient than the sometimes awkward language of arithmetic inequations. We therefore will develop a generic translation from logics into constraints that does not directly depend on the actual constraint solver that is available.

3.4.4 Translating Formulas into Constraints

Our task in this section is to define a translation from a set of $\mathcal{MQL}$ formulas into a set of constraints with regard to a given constant frame $\mathcal{C}$ that defines the range of quantification for different types. Our translation must follow the definition of a $\mathcal{MQL}$ semantic and map each complex formula into a constraint that reflects the relation that the interpretation of the formula has to the interpretation of its parts or instantiations. At the same time, we want to leave open which logical constants we have in our logic—we want to be able to extend the basic set as necessary—as well the actual constraint language that we are using. Our translation therefore is only a scheme that can be instantiated by an actual translation.

Fig. 3.2 shows the translation of $\mathcal{MQL}$ specifications into constraints as a tableaux inference system. We use **signed formulas** $V_{\mathsf{F}}\colon \mathsf{F}$ where F is a formula and V_{F} is a associated with the interpretation $[\![\mathsf{F}]\!]$. For a specification $\Phi = \{\mathsf{F}_1, \ldots, \mathsf{F}_n\}$, we start with an initial tableau $\Theta = \{V_{\mathsf{F}_1} : \mathsf{F}_1, \ldots, V_{\mathsf{F}_n} : \mathsf{F}_n, V_{\mathsf{F}_1} = 1, \ldots, V_{\mathsf{F}_n} = 1\}$. The top-level integer variables V_{F_j} all are restricted to 1 by the equations $V_{\mathsf{F}_j} = 1$.

Each of the rules given in our Constraint Tableaux (CT) system can expand the current tableau branch by new signed formulas and constraints. Tableau expansion stops when the tableau is saturated, i.e., no rule can add new information to the tableau.

The *con*2-rule expands signed formulas $V_{\mathsf{F}}\colon \mathsf{F}_1 \circ \mathsf{F}_2$ that are governed by a binary logical connective $\circ$. The tableau is extended by new signed formulas $V_{\mathsf{F}_1}\colon \mathsf{F}_1$ and $V_{\mathsf{F}_2}\colon \mathsf{F}_2$ for the components F_1 and F_2. The rule adds a constraint $V_{\mathsf{F}} = [\![\circ]\!](V_{\mathsf{F}_1}, V_{\mathsf{F}_2})$ for the connective $\circ$. The constraint describes the relation between the variables V_{F}, V_{F_1} and V_{F_2} and is directly derived from the semantic of the logical connective $\circ$. In the case of a conjunctive formula and an IP constraint solver, this constraint could be the equation $V_{\mathsf{F}} = min(V_{\mathsf{F}_1}, V_{\mathsf{F}_2})$.

$$\frac{V_{\mathsf{F}}\colon \mathsf{F}_1 \circ \mathsf{F}_2}{\begin{array}{c} V_{\mathsf{F}_1}\colon \mathsf{F}_1 \\ V_{\mathsf{F}_2}\colon \mathsf{F}_2 \\ V_{\mathsf{F}} = [\![\circ]\!](V_{\mathsf{F}_1}, V_{\mathsf{F}_2}) \end{array}}\, con2 \qquad \frac{V_{\neg\mathsf{F}}\colon \neg\mathsf{F}}{\begin{array}{c} V_{\mathsf{F}}\colon \mathsf{F} \\ V_{\neg\mathsf{F}} = [\![\neg]\!](V_{\mathsf{F}}) \end{array}}\, con1 \qquad \frac{\begin{array}{c} V_1\colon \mathsf{A} \\ V_2\colon \mathsf{A} \end{array}}{V_1 = V_2}\, uni \text{ (A is an atom)}$$

$$\frac{V_{QT}\colon \mathrm{Q}(\mathsf{T}_{\alpha\to o})(\mathsf{U}_{\alpha\to o})}{\begin{array}{c} V_{(\mathsf{T}\ C^1)}\colon (\mathsf{T}\ C^1) \\ \vdots \\ V_{(\mathsf{T}\ C^n)}\colon (\mathsf{T}\ C^n) \\ V_{(\mathsf{U}\ C^1)}\colon (\mathsf{U}\ C^1) \\ \vdots \\ V_{(\mathsf{U}\ C^n)}\colon (\mathsf{U}\ C^n) \\ \\ V_{QT} = [\![\mathrm{Q}]\!](\langle V_{(\mathsf{T}\ C^1)}, \ldots, V_{(\mathsf{T}\ C^n)}\rangle)(\langle V_{(\mathsf{U}\ C^1)}, \ldots, V_{(\mathsf{U}\ C^n)}\rangle) \end{array}}\, quan2$$

$$\frac{V_{QT}\colon \mathrm{Q}(\mathsf{T}_{\alpha\to o})}{\begin{array}{c} V_{(\mathsf{T}\ C^1)}\colon (\mathsf{T}\ C^1) \\ \vdots \\ V_{(\mathsf{T}\ C^n)}\colon (\mathsf{T}\ C^n) \\ \\ V_{QT} = [\![\mathrm{Q}]\!](\langle V_{(\mathsf{T}\ C^1)}, \ldots, V_{(\mathsf{T}\ C^n)}\rangle) \end{array}}\, quan1$$

Fig. 3.2. The Constraint Tableaux (CT) expansion rules

The *con*1-rule corresponds to the *con*2-rule, but treats unary connectives and will add only one new signed formula.

The *uni*-rule adds an equality constraint for all integer variables that represent the same atom. This unification of variables ensures that we actually compute interpretations that are functions.

Finally, the *quan*2-rule and *quan*1-rules translates formulas governed by diadic and monadic quantifier constants Q. Given a constant frame $\mathcal{C}$, there is a tuple of constants $\mathcal{C}_\alpha = \langle C_1, \ldots, C_n \rangle$, $n \geq 0$ for each type α that occurs in Φ. The order of constants may be arbitrary, but fixed. The *quan*-rules build new formulas by applying each of the terms T (and U) to all constants in $\mathcal{C}_\alpha$ of the appropriate type, and recursively translates these instantiations into constraints. The rule then generates a constraint that relates the integer variable of the whole quantified formula and the integer variables of its instantiations as defined by the denotation of the quantifier in question.

Unlike conventional tableaux systems, the CT calculus does not split the current tableau branch in any way during tableau construction.

3.4.5 An Example

We illustrate our tableaux system CT with a short example that uses arithmetic constraints for implementing the semantics of logical constants.

Consider a constant frame $\mathcal{C}$ that contains a set $\mathcal{C}_\iota = \{john, pete, karl\}$ for individual constants and a set $\mathcal{C}_{\iota \to o} = \{\mathsf{drink}, \mathsf{man}\}$ for first-order predicates. A first-order specification $\{\forall x\ man(x) \Rightarrow drink(x))\}$, i.e., *all men drink*, would be expanded first into the following tableau:

V_1: $\forall x\ man(x) \Rightarrow \mathsf{drink}(x))$
$\star\ V_1 = 1$
V_2: $\mathsf{man}(john) \Rightarrow \mathsf{drink}(john)$
V_3: $\mathsf{man}(pete) \Rightarrow \mathsf{drink}(pete)$
V_4: $\mathsf{man}(karl) \Rightarrow \mathsf{drink}(karl)$
$\star\ V_1 = min(\langle V_2, V_2, V_4\rangle)$

The tableau contains two constraints, indicated by $\star$. The first one, $V_1 = 1$, is part of the initial tableau and restricts the value of the truth variable V_1 to 1 because each top-level formula of a specification must be interpreted to 1 in a model. The second one, $V_1 = min(\langle V_2, V_2, V_4\rangle)$ relates V_1 to the truth variables of the three possible instantiations.

Each of the newly generated signed formulas is governed by a logical connective, and must therefore be expanded further by the *bin*-rule. This second stage of expansions will give us the following tableau:

V_1: $\forall x\ man(x) \Rightarrow \mathsf{drink}(x))$
$\star\ V_1 = 1$
V_2: $\mathsf{man}(john) \Rightarrow \mathsf{drink}(john)$
V_3: $\mathsf{man}(pete) \Rightarrow \mathsf{drink}(pete)$
V_4: $\mathsf{man}(karl) \Rightarrow \mathsf{drink}(karl)$
$\star\ V_1 = min(\langle V_2, V_2, V_4\rangle)$
V_5: $\mathsf{man}(john)$
V_6: $\mathsf{drink}(john)$
$\star\ V_2 = min(1 - V_5, V_6)$
V_7: $\mathsf{man}(pete)$
V_8: $\mathsf{drink}(pete)$
$\star\ V_3 = min(1 - V_7, V_8)$
V_9: $\mathsf{man}(karl)$
V_{10}: $\mathsf{drink}(karl)$
$\star\ V_4 = min(1 - V_9, V_{10})$

After these expansions, no rule is applicable any more. Our translation has computed a set of constraints which can be solved by an IP constraint solver after all variables V_j have been restricted to integer values 0 and 1.

A solution is an instantiation $\sigma = \{V_1 = v_1, \ldots V_{10} = v_{10}\}$ for the integer variables V_j and truth values v_j that satisfies the constraints. For instance, a solution for our example is $\sigma = \{V_1 = 1, V_2 = 1, V_3 = 1, V_4 = 1, V_5 = 0, V_6 = 0, V_7 = 0, V_8 = 1, V_9 = 1, V_{10} = 1\}$. The variables that represent atomic formulas, i.e., the variables $\{V_5, \ldots, V_{10}\}$, can be used for constructing the representation sets for evaluation functions. The solution σ represents the model $\mathcal{M} = \langle\{\mathsf{drink}(pete), \mathsf{man}(karl), \mathsf{drink}(karl)\}, \mathcal{C}\rangle$.

3.4.6 Properties of the Translation

A specification Φ is satisfiable with respect to a constant frame $\mathcal{C}$ for Φ iff Φ has a model $\mathcal{M} = \langle [\![\ ]\!], \mathcal{C}_\Phi \rangle$. A constraint tableau Θ is satisfiable iff its set of constraints can be satisfied by a solution σ such that $[\![\mathsf{F}]\!] = \sigma(V_\mathsf{F})$ for an evaluation function $[\![\ ]\!]$ and all signed formulas V_F: F that occur in Θ.

3.4.7 Refutation Soundness

Theorem 1. *A constraint tableau for a specification Φ and a constant frame $\mathcal{C}_\Phi$ is satisfiable if Φ is satisfiable with respect to $\mathcal{C}_\Phi$.*

Proof. If Φ is satisfiable with respect to $\mathcal{C}_\Phi$, then Φ has a model $\mathcal{M} = \langle [\![\ ]\!], \mathcal{C}_\Phi \rangle$ such that $[\![\mathsf{F}_j]\!] = 1$ for all $\mathsf{F}_j \in \Phi$. An initial tableau Θ_0 for Φ and $\mathcal{C}_\Phi$ consists of pairs V_{F_j}: F_j and equations $V_{\mathsf{F}_j} = 1$. We chose $\sigma_0(V_{\mathsf{F}_j}) = 1$ for all V_{F_j}. Then σ_0 obviously satisfies all constraints in Θ_0 and $[\![\mathsf{F}_j]\!]_\mathcal{M} = \sigma(V_{\mathsf{F}_j})$ holds for all signed formulas V_{F_j}: F_j in Θ_0. Hence, Θ_0 is satisfiable.

The set of constraints in a satisfiable tableau Θ can be satisfied by a solution σ such that $[\![\mathsf{F}]\!]_\mathcal{M} = \sigma(V_\mathsf{F})$ for an evaluation function $[\![\ ]\!]$ in $\mathcal{M}$ and all signed formulas V_{F_j} : F_j that occur in Θ. Let V_F : F be a signed formula in Θ. We show that each expansion Θ' of V_F: F in a satisfiable tableau Θ is satisfiable.

Let $\mathsf{F} = \neg\mathsf{F}'$. Then $[\![\mathsf{F}]\!]_\mathcal{M} = [\![\neg]\!][\![\mathsf{F}']\!]_\mathcal{M}$. The expansion Θ' of Θ by the new signed formula $V_{\mathsf{F}'}$: F' and the constraint $V_\mathsf{F} = [\![\neg]\!](V_{\mathsf{F}'})$ is satisfied by a solution σ' such that $\sigma'(V) = \sigma(V)$ for all $V \neq V_{\mathsf{F}'}$, and $\sigma'(V_{\mathsf{F}'}) = [\![\mathsf{F}']\!]_\mathcal{M} = [\![\neg]\!](V_\mathsf{F})$ else. If $\mathsf{F} = \mathsf{F}_1 \circ \mathsf{F}_2$ then $[\![\mathsf{F}]\!]_\mathcal{M} = [\![\circ]\!]([\![\mathsf{F}_1]\!]_\mathcal{M}, [\![\mathsf{F}_2]\!]_\mathcal{M})$, and an expansion of Θ by the new signed formulas V_{F_1} : F_1 and V_{F_2} : F_2 and the constraint $V_\mathsf{F} = [\![\circ]\!](V_{\mathsf{F}_1}, V_{\mathsf{F}_2})$ is satisfiable by a modification of the solution σ to a solution σ' with $\sigma'(V_{\mathsf{F}_1}) = [\![\mathsf{F}_1]\!]_\mathcal{M}$ and $\sigma'(V_{\mathsf{F}_2}) = [\![\mathsf{F}_2]\!]_\mathcal{M}$.

If $\mathsf{F} = \mathrm{Q}(\mathsf{T}^1)\ldots(\mathsf{T}_n)$, with Q being a quantifier constant, then an interpretation of F must formulate a constraint over all instantiations $\mathsf{T}_i(p_j)$ of constants C_k of appropriate type for each T_i. Again, the expansion of Θ by the aforementioned constraint and the new signed formulas $V_{(\mathsf{T}C_j)}$: $(\mathsf{T}C_j)$ for the instantiations is satisfiable by a suitably extended solution σ' such that $\sigma'(V_{(\mathsf{T}p_j)}) = [\![(\mathsf{T}p_j)]\!]_\mathcal{M}$.

Let F be an atomic formula and Θ be a tableau containing two signed formulas V_1 : F and V_2 : F. Because both are part of a satisfiable tableau, $[\![\mathsf{F}]\!]_\mathcal{M} = \sigma(V_1) = \sigma(V_2)$ holds for an evaluation function $[\![\ ]\!]$, and an extension Θ' of Θ by the constraint $V_1 = V_2$ is obviously satisfied by σ. □

Theorem 1 proves a form of refutation soundness. Starting with a satisfiable specification Φ over a constant frame $\mathcal{C}$, each tableau expansion will create a satisfiable set of constraints. It is trivial to show that a solution σ that determines the interpretation $[\![\mathsf{A}]\!] = \sigma(V_\mathsf{A})$ for each atom A in the tableau also determines the interpretations of all occurences of complex formulas that depend on these. Additionally, a solution must interpret the atoms in a way such that the formulas of Φ become true. Hence, the part of a solution σ

that determines the evaluation of all atoms A in the saturated tableau for Φ is equivalent to a model for the specification Φ relative to a given constant frame $\mathcal{C}_\Phi$. Theorem 1 implies that a constraint translation of a satisfiable specification over a constant frame will lead to a tableau that can be solved. Our method is complete in that we can be sure to find at least one solution that represents a positive model.

3.4.8 Completeness for $\mathcal{MQL}$ Satisfiability

Our translation into constraints also gives us a decision procedure for the existence of models in a tableau. A saturated constraint tableau Θ may contain only finitely many signed formulas, and there can only be finitely many different evaluations σ for the atoms in Θ that represent models of Φ. Hence, we can decide whether a tableau Θ contains a model. In the following, we show that we can use this result for proving $\mathcal{MQL}$ satisfiability.

Lemma 1. *Let Φ be satisfiable with respect to a constant frame $\mathcal{C}_\Phi$. Let $\mathcal{C}'_\Phi$ be a constant frame for Φ with $|\mathcal{C}_\alpha| = |\mathcal{C}'_\alpha|$ for all $\mathcal{C}_\alpha \in \mathcal{C}_\Phi$ and all $\mathcal{C}'_\alpha \in \mathcal{C}'_\Phi$. Then Φ is also satisfiable with respect to $\mathcal{C}'_\Phi$.*

Proof. Let Φ be satisfiable by a model $\mathcal{M} = \langle \mathcal{C}_\Phi, [\![\ ]\!] \rangle$. Then $[\![\ ]\!]$ has a finite representation as a set $\{\mathsf{A}_1, \ldots, \mathsf{A}_k\}$ of ground atoms. We show that Φ then has a model $\mathcal{M}' = \langle [\![\ ]\!]', \mathcal{C}'_\Phi \rangle$ with the representation set of $[\![\ ]\!]'$ being equal to the set representation of $[\![\ ]\!]$ up to a renaming of constant symbols.

Let Θ_0 be an initial constraint tableau for Φ and $\mathcal{C}_\Phi$ and Θ'_0 be an initial constraint tableau for Φ and $\mathcal{C}'_\Phi$. It is easy to see that both tableaux are identical modulo a bijective mapping of constant symbols ρ from $\mathcal{C}_\Phi$ to $\mathcal{C}'_\Phi$ such that $\rho(h_\alpha) = h'_\alpha$ for all h_α that are in $\mathcal{C}_\Phi$ but not in $\mathcal{C}'_\Phi$, and $\rho(h_\alpha) = h_\alpha$ else. We denote the renaming of all constant symbols in a formula F according to ρ as $\rho(\mathsf{F})$.

Every extension of a tableau Θ for $\mathcal{C}_\Phi$ has an equivalent extension in Θ' such that each signed formula $V\colon \mathsf{F}$ in Θ has a corresponding signed formula $V\colon \mathsf{F}'$ with $\mathsf{F} = \rho(\mathsf{F}')$. Because the constraints in both tableaux are isomorphic, each tableaux has the same solutions σ_j. We select one solution σ. Let $V_\mathsf{A}\colon \mathsf{A}$ be an atomic formula in Θ such that $[\![\mathsf{A}]\!] = \sigma(V_\mathsf{A}) = 1$. Then there is an atomic formula A' with $\mathsf{A}' = \rho(\mathsf{A})$ in Θ' such that $\sigma(V'_\mathsf{A}) = \sigma(V_\mathsf{A})$ and hence $[\![\mathsf{A}]\!] = [\![\mathsf{A}']\!]'$. Because A is in the representation set of $[\![\ ]\!]$, A' must be in the representation set of the evaluation function $[\![\ ]\!]'$, and vice versa. Hence the representation sets for $[\![\ ]\!]$ and $[\![\ ]\!]'$ are equal up to a renaming ρ of constant symbols. □

Theorem 2 (Satisfiability Completeness). *There is an algorithm that proves the satisfiability of an $\mathcal{MQL}$ specification Φ.*

Proof. Let $\mathcal{M} = \langle [\![\ ]\!], \mathcal{C}_\Phi \rangle$ be an arbitrary $\mathcal{MQL}$ model for Φ. Starting with a smallest initial constant frame $\mathcal{C}^0_\Phi$, we can enumerate models for extensions $\mathcal{C}'_\Phi$ of $\mathcal{C}^0_\Phi$ by adding arbitrary new constants h_α to sets $\mathcal{C}_\alpha \in \mathcal{C}^0_\Phi$ and solving the constraint tableaux for Φ and $\mathcal{C}'_\Phi$. As long as our extension strategy is fair and complete, we will eventually reach a constant frame $\mathcal{C}'_\Phi$ with $|\mathcal{C}_\alpha| = |\mathcal{C}'_\alpha|$ for all $\mathcal{C}_\alpha \in \mathcal{C}_\Phi$ and all $\mathcal{C}'_\alpha \in \mathcal{C}'_\Phi$. Lemma 1 shows that Φ has a model $\mathcal{M} = \langle [\![\ ]\!]', \mathcal{C}'_\Phi \rangle$ that is equal to $\mathcal{M}$ up to a renaming of constant symbols. Hence, the corresponding constraint tableau Θ is satisfiable and we can prove that Φ is satisfiable by producing a model of Φ relative to $\mathcal{C}'_\Phi$. □

Theorem 2 implies that we are bound to find a model for a satisfiable $\mathcal{MQL}$ specification Φ when applying a fair iterative deepening strategy over the size of the constant frame $\mathcal{C}$. All models of $\mathcal{MQL}$ are finite, and we therefore have a procedure for proving $\mathcal{MQL}$ satisfiability.

3.4.9 Enumerating Models

Our translation from logics into constraints is not complete in the sense that we can guarantee to find a representative model $\mathcal{M}$ for every model of the input Φ. The translation only considers formulas that can be expanded in some way from input formulas, and will not determine the interpretation of ground atoms that do not occur in the saturated tableau at all. The interpretation of such atoms can be chosen freely and therefore can be ignored when we are only interested in subset-minimal positive models. Still, we would like to have a method that truly enumerates the $\mathcal{MQL}$ models of a specification.

The following theorem shows that a slight modification of our model generation method generates all isomorphic models of the input.

Theorem 3 (Model Completeness). *There is an algorithm that enumerates all models of a satisfiable specification Φ up to a renaming of constant symbols.*

Proof. Given a specification Φ and an initial constant frame $\mathcal{C}^0_\Phi$, we extend Φ to a specification Φ' in the following way: For each ground atom A that can be build from the constant frame $\mathcal{C}^0_\Phi$, we add the formula $\mathsf{A} \vee \neg\mathsf{A}$ to Φ. The newly added formula obviously is satisfiable under the usual interpretation of the logical constants $\vee$ and $\neg$.

The solutions for a constraint tableau Θ_0 for Φ' and $\mathcal{C}^0_\Phi$ determine the evaluation of all atoms A and represent all isomorphic models in the size of the constant frame $\mathcal{C}^0_\Phi$. By iterative deepening over the size of the constant frame, and by adding the tautologies $\mathsf{A} \vee \neg\mathsf{A}$ for all new atoms that we add, we can now enumerate all $\mathcal{MQL}$ models up to isomorphism by enumerating the solutions σ for each resulting tableau. □

4 Minimal Model Generation

We now define a new class of models for $\mathcal{MQL}$ specification, the locally minimal models. They are an amalgamation of domain minimal models and subset-minimal models known from first-order model generation. The property of being a local minimal model in our logic $\mathcal{MQL}$ is decidable.

4.1 Preliminaries

The individual domain size $|\mathcal{C}_\iota(\mathcal{M})|$ of a $\mathcal{MQL}$ model $\mathcal{M}$ is the number of individuals constants c_i that occur in a finite representation of $\mathcal{M}$ as a set of ground literals.

A model $\mathcal{M}$ for a $\mathcal{MQL}$ specification ϕ is **domain minimal** if its individual domain size $\mathcal{C}_\iota(\mathcal{M})$ is minimal, i.e., there is no model $\mathcal{M}'$ of ϕ such that $|\mathcal{C}_\iota(\mathcal{M}')| < |\mathcal{C}_\iota(\mathcal{M})|$. A model $\mathcal{M}$ for ϕ is a **locally minimal model** iff it is domain minimal and the condition $\textsc{Pos}(\mathcal{M}') \subseteq \textsc{Pos}(\mathcal{M}) \Rightarrow \mathcal{M} = \mathcal{M}'$ holds for all domain minimal models $\mathcal{M}'$ of ϕ.

Local minimality is both stronger and weaker than the classical subset-minimal model property defined in Section 2.3.5. A locally minimal model always has a minimal individual domain while minimal models may have arbitrarily large domains. At the same time, a locally minimal model does not have to be minimal with regard to all other finite models of ϕ, but only in comparison with those that have the same domain size.

4.2 Decidability of Local Minimality

In this section, we prove that local minimality for finite models in $\mathcal{MQL}$ is decidable. The method is based on the observation that the minimal models $\mathcal{M}$ up to a give size of the individual domain can be characterised by the literals that occur negatively within $\mathcal{M}$. Niemelä [24] originally used this characterisation for defining a method for propositional minimal model reasoning.

Lemma 2. *Let $\mathcal{M} = \{\mathsf{A}_1, \ldots, \mathsf{A}_n, \neg\mathsf{B}_1, \ldots, \neg\mathsf{B}_m\}$ be a local minimal model of a $\mathcal{MQL}$ specification ϕ. Then there is no model $\mathcal{M}'$ with $|\mathcal{C}_\iota(\mathcal{M}')| = |\mathcal{C}_\iota(\mathcal{M})|$ that satisfies $\phi' = \phi \wedge \neg\mathsf{B}_1 \wedge \ldots \wedge \neg\mathsf{B}_m \wedge \neg(\mathsf{A}_1 \wedge \ldots \wedge \mathsf{A}_n)$*

Proof. Assume that $\mathcal{M}' = \{\mathsf{A}'_1, \ldots, \mathsf{A}'_k, \neg\mathsf{B}'_1, \ldots, \neg\mathsf{B}'_l\}$ exists. Obviously, $\mathcal{M}'$ is a domain minimal model of ϕ. $\mathcal{M}'$ satisfies all negative literals that are satisfied by $\mathcal{M}$, but only a subset of the positive literals $\{\mathsf{A}_1, \ldots, \mathsf{A}_n\}$. This implies that the set of positive literals of $\mathcal{M}'$, $\{\mathsf{A}'_1, \ldots, \mathsf{A}'_k\}$, is a real subset of the positive literals $\{\mathsf{A}_1, \ldots, \mathsf{A}_n\}$, while the set of negated literals of $\mathcal{M}$, $\{\neg\mathsf{B}_1, \ldots, \neg\mathsf{B}_m\}$, is a real subset of $\{\neg\mathsf{B}'_1, \ldots, \neg\mathsf{B}'_l\}$. Hence, $\mathcal{M}'$ and $\mathcal{M}$ must be different and $\mathcal{M}'$ is a model of ϕ with $\textsc{Pos}(\mathcal{M}') \subseteq \textsc{Pos}(\mathcal{M})$ but $\mathcal{M}' \neq \mathcal{M}$. This contradicts that $\mathcal{M}$ is a locally minimal model. □

Theorem 4 (Decidability of Locally Minimal Models). *Given a model*

$$\mathcal{M} = \{\mathsf{A}_1, \ldots, \mathsf{A}_n, \neg\mathsf{B}_1, \ldots, \neg\mathsf{B}_m\}$$

of a $\mathcal{MQL}$ specification ϕ, it is decidable whether $\mathcal{M}$ is a locally minimal model of ϕ.

Proof. By lemma 2, there can be no model $\mathcal{M}'$ that satisfies $\phi' = \phi \wedge \neg\mathsf{B}_1 \wedge \ldots \wedge \neg\mathsf{B}_m \wedge \neg(\mathsf{A}_1 \wedge \ldots \wedge \mathsf{A}_n)$ in the same domain size as $\mathcal{M}$ if $\mathcal{M}$ is locally minimal. Hence it suffices to show that ϕ' is unsatisfiable within the domain size $|\mathcal{C}_\iota(\mathcal{M})|$. We chose an initial constant frame $\mathcal{C}_0$ for ϕ'. If ϕ' is unsatisfiable, then a saturated constraint tableau Θ for ϕ' and our constant frame $\mathcal{C}_0$ cannot have a solution. Hence, the set of constraints in Θ is unsatisfiable, which itself is a decidable property. □

5

The Analysis of Definites

The use of *the* seems to suggest shared experience or knowledge: the listener/reader 'has been there too'.
(Michael Swan, *Practical English Usage*)

Overview: Model generation is a tool for natural-language interpretation in context. We apply a finite model generator to formalisations of singular definite descriptions and show that minimal model generation permits a computational treatment of definite noun phrases which directly reflects contemporary theories of definiteness. Model generation can be used as a uniform tool for analysing linguistic theories on definites.

5.1 Introduction

Given the semantic representation of a sentence and a specification of a context as a set of formulas, a finite model generator will enumerate a set of finite models satisfying both. The information in certain finite models encodes both the context change as well as the truth conditions of the sentence. In the truth-conditional sense of the word, model generation computes meaning. One of the things that we can use this computed meaning for is to (re)formulate and analyse linguistic theories.

The linguistic guinea pig that we[1] apply our analysis to are singular definite descriptions. Our platform of experimentation is the finite model generator KIMBA (cf. Chapter 8) that implements a translation of $\mathcal{MQL}$ logical specifications into constraints over finite-domain integer variables as well as the computation of minimal models as presented in Chapter 4.

We show that minimal model generation permits a computational treatment of singular definite noun phrases which can combine the theoretical insights from contemporary theories of definiteness with the processing of logically encoded knowledge about the situational context in which definite sentences are interpretated.

[1] The research presented here, as well as that presented in Chapter 6 originates in joint work with Claire Gardent.

5.1.1 The Semantics of Definite Descriptions

Singular **definite descriptions** are expressions that refer to a particular individual entity without referring to its name, as in (5.1)– (5.3).

(5.1) *The President of Germany*

(5.2) *The first man on the moon*

(5.3) *Washington's mother*

The analysis of singular definite descriptions, in the following called **definites**, was one of the starting points of the logic-based approach to language philosophy. Both Russell and Frege investigated the problem how the principle of bivalence, i.e., the property of propositions to be either true of false, can be maintained if a definite refers to a non-existing entity, such as in the following famous example.

(5.4) ***The King of France*** *is bald.*

Russel's treatment of definites in predicate logics makes use of a special iota (ι) operator. The expression $\iota x \mathsf{P}(x)$ denotes the x such that $\mathsf{P}(x)$ holds. Thus, examples (5.4) can be formalised as follows.

(5.5) $\mathsf{bald}(\iota x\ \mathsf{kof}(x))$

This approach fails in cases where such an x does not exist, or when there are too many x. Then, propositions like (5.5) can neither be interpreted as true nor as false. Russell's solution to this problem was a transformation from what he calls the "misleading form" of the formula (5.5) into a logical form that refers only to standard quantifiers $\exists$ and $\forall$. The transformation was first presented in his article "On Denoting" [57]. The result of the transformation for (5.5) into first-order predicate logic looks as follows.

(5.6) $\exists x\ \mathsf{kof}(x) \wedge \mathsf{bald}(x) \wedge \forall y\ \mathsf{kof}(y) \Rightarrow x = y$

In higher-order logics, Russell's formalisation is equivalent to a Montague-style linguistic quantification where THE is defined as in (5.7). This leads to the representation (5.8) for sentence (5.4) which nicely hides the discrepancy between the intended logical form and the "misleading" form of natural language.

(5.7) $\textsc{The} \equiv \lambda P\ \lambda Q\ \exists x\ P(x) \wedge Q(x) \wedge \forall y\ P(y) \Rightarrow x = y$

(5.8) $\textsc{The}(\mathsf{kof})(\mathsf{bald})$

5.1.2 Definites and Deduction

A formal representation such as (5.8) can be used for all kinds of inference. For instance, consider the consequence relation that holds for the sentences in example (5.9).

(5.9) *The King of France is bald* $\models$ *There is a King of France.*

This consequence relation can be verified mechanically by applying a first-order theorem prover to the theorem (5.10).

(5.10) $\textsc{The}(\mathit{kof})(\mathit{bald}) \Rightarrow \exists x\ \mathit{kof}(x)$

Given a logically encoded context, deduction can sometimes be used to verify whether a provisional resolution of an anaphor is valid. By anaphor we mean any natural language expression whose referent can only be determined by relating it to an antecedent referent in the discourse. Example (5.12) is a typical proof task that is equivalent to the resolution of the anaphoric information given informally in (5.11). By proving (5.12), we show that the proposed resolution in (5.11) is correct.

(5.11) *Jon has a rabbit.* ***The rabbit*** *sleeps.* $\models^?$ *The rabbit that Jon has sleeps.*

(5.12) $\mathit{rbt}(r_1) \land \mathit{has}(jon, r_1) \land \textsc{The}(\mathit{rbt})(\mathit{sleep}) \Rightarrow$
$\textsc{The}(\lambda x\ \mathit{rbt}(x) \land \mathit{has}(jon, x))(\mathit{sleep})$

Note that (5.12) has a standard first-order form when expanded, and proving the formula should present no difficulty to any state-of-the-art first-order theorem prover. However, a purely deductive approach to the resolution of definites will fail whenever the definite is not an anaphor. The interpretation of a definite sometimes requires the **accommodation** of new individuals. The term accommodation goes back to Lewis [70] and refers to the process by which the listener adjust her assumptions by adding just enough information to remedy the violation of some felicity condition. Consider the following sentence.

(5.13) ***Jon's rabbit*** *is cute.*

Here, we have no context in which we can resolve the definite *Jon's rabbit*. The interpretation of (5.13) requires that we accommodate the existence of some rabbit that has the property of both being the rabbit of Jon and sleeping. If we would insist on asserting the existence of a suitable referent in the context, then example (5.13) cannot be interpreted as true.

What we would like to have for examples such as (5.13) is a method that interprets a sentence relative to a given context and accommodates new information if necessary. The meaning of a sentences is then given by the change that the interpretative process imposes on the context. A method that provides this is minimal model generation.

5.1.3 How Models Interpret Sentences

A logical model can be conceptualised as a set of basic assumptions under which a given logical specification is true. These assumptions are what we are after when we interpret a sentence. In example (5.13), the minimal set of assumptions can be stated informally as follows.

- There is an individual named Jon who
- owns some rabbit r_1 and
- r_1 has the property of being cute.

The assumptions encode truth-conditional meaning. For natural-language interpretation, we need models whose entailed assumptions are either evident or accommodated as required. When accommodation is necessary, it should be restricted to sets of assumptions that at least are consistent with what we know about the situation at hand. Further, it should be obvious that the sets of assumptions must state all information that is required for validating the truth of the interpreted sentence: the assumptions should explain *completely* why a certain sentence can be true in the given context. Finally, we do not want over-explanation in the form of irrelevant or unjustifiable information. An interpretation must not state more information as is required by the task at hand.

It lies in the nature of a model that all assumptions that can be derived from it are true under the interpretation given by this model. If we use a background theory that contains the necessary world knowledge, a model of this background theory and a semantic representation will not entail logically inconsistent facts.

Further, a model always gives sufficient information for the truth of a theory. Every interpretation in the Tarskian sense of the word unambiguously defines the truth value of the specifications that it validates. If a model lacks any necessary information, it cannot be a model at all.

The elimination of irrelevant information is not an issue of model generation in general, but it is an issue for *minimal* model generation. Minimality constraints for models usually restrict the positive assumptions that are made by a model. The locally minimal models that we investigated in Chapter 4 are models that satisfy a specification both without referring to more individual entities than necessary and without making unnecessary assumptions relative to all other models that can be found in the smallest domain. In the sense of Occam's Razor, locally minimal models are the simplest, and therefore often the best explanations for the truth of a theory. In natural-language interpretation, the locally minimal models are essential because they minimise the accommodated individuals and the accommodated assumptions. For definite descriptions, the minimisation of individuals ensures that a resolution with referents in the context is preferred over the accommodation of new individuals. This preference has been advocated for on empirical grounds for instance by Strawson [71].

5.1.4 Discourse Models

In what follows, we understand the concept of a discourse model as a logical description of a context that is derived from a discourse. In general, the term *discourse model* denotes an abstraction of a situation in a real or hypothetical world. In cognitive psychology, such discourse models have been used to explain inferences that people draw in understanding text [72]. In Artificial Intelligence, Webber proposes them as describing the situation/state which the speaker is talking about [73]. In the dynamic/DRT (Discourse Representation Theory [74]) trend of natural language semantics, it represents the context created by previous discourse and against which subsequent utterances will be interpreted [74, 75].

Discourse models are abstractions in that we restrict some situation to a finitary or otherwise limited view that focuses on the information that is present in a discourse. In natural-language processing, the task of understanding an utterance is frequently modelled by the task of creating a discourse model by a hearer [72]. The crucial observation underlying the concept of a discourse model is that the situation a speaker is talking about influences the way in which discourse is interpreted. This intuition is made precise in contemporary dynamic theories of meaning which view meaning as a relation between contexts: a sentence is interpreted relative to a context and the interpretation of that sentence yields a new context, the context against which the next sentence will be interpreted. The context change that the models of a semantic representation induce is what will be in the focus of our interest, because this is where we expect to find a considerable part of the truth-conditional meaning.

As our formal framework is that of classical logics, we will focus on phenomena where we can widely ignore the effects of dynamic discourse structure. A typical problem that we will not attack is illustrated by example (5.14).

(5.14) ***Jon's rabbit*** *is cute.* ***The rabbit*** *is white.* ***Peter's rabbit*** *is cute, too.* ***The rabbit*** *is black.*

Definite descriptions sometimes act as anaphors whose binding to referents is determined by the structure of the discourse. Hence, the association of the definite **The rabbit** to its referent dynamically changes in example (5.14) with the focus of the discourse. A convincing treatment of dynamic effects in discourse models would require suitable data structures that can deal with structural properties such as dynamic accessibility of and focus. As these problems lie beyond the subject of this volume, our discourse models are representations of situations and not that of discourses. In our analysis, we will concentrate on single sentences that are interpreted with respect to a given logical context.

5.1.5 Models for Definites

In $\mathcal{MQL}$, we can reduce the semantics of the standard semantic representation of the generalised determiner THE to a higher-order definition that relies only on the standard quantifiers $\exists$ and $\forall$ of predicate logic. As an alternative, definition (5.15) makes the two separate conditions in the semantic of Russellian definites more explicit. It is equivalent to the set-theoretical higher-order formulation (5.17) that is sometimes found in the literature [76].

(5.15) $\textsc{The} \equiv \lambda P \lambda Q\ \textsc{Unique}(P) \wedge \textsc{Every}(P)(Q)$

(5.16) $\textsc{Unique} \equiv \lambda P\ \exists x\ P(x) \wedge \forall y\ P(y) \Rightarrow x \doteq y$

(5.17) $\textsc{The} \equiv \lambda P \lambda Q\ |P| = 1 \wedge P \subseteq Q$

Russell's approach to definites leads to a first-order expressible form, and so does an expansion of the definitions (5.15)–(5.17).

We use the Herbrand equality symbol '$\doteq$' in the definition of the second-order unicity predicate UNIQUE. As discussed in Section 3.3.9, the symbol $\doteq$ is interpreted as a two-place predicate on elements of a constant frame $\mathcal{C}$ such that $C \doteq D$ is true iff C and D are identical constants. Under $\mathcal{MQL}$'s unique name assumption for all interpretations, two constants are considered equal iff they have the same name.

The finite model generator KIMBA for $\mathcal{MQL}$ logics will be discussed in detail in Chapter 8. KIMBA computes, among other classes of models, the locally minimal models of an input specification. With THE now at our hands, we can apply KIMBA to sentences with singular definite descriptions and investigate whether our semantic representations have the locally minimal models that we expect. In the following, the logical specification of the discourse model for (5.18) is given in (5.19), the semantic representation of the sentence with the definite is the formula (5.20).

(5.18) *Jon has a rabbit.* ***The rabbit*** *is cute.*

(5.19) $\mathit{rbt}(r_1) \wedge \mathit{has}(jon, r_1)$

(5.20) $\textsc{The}(\mathit{rbt})(\mathit{cute})$

We apply KIMBA to the logical specification(5.21), i.e., the conjunction of the logical description of the discourse model (5.19) and the semantic representation (5.20). A model for specification (5.21) must consider both the discourse model and the semantic representation. The model generation process generates the facts (5.22) under which both parts of the specification are true.

(5.21) $\mathit{rbt}(r_1) \wedge \mathit{has}(jon, r_1) \wedge \textsc{The}(\mathit{rbt})(\mathit{cute})$

(5.22) $\{\mathit{has}(jon, r_1), \mathit{rbt}(r_1), \mathit{cute}(r_1)\}$

The minimal model clearly is an intuitive description of the meaning of the sentence (5.18). In order to make the semantic representation (5.20) true in the discourse model (5.19), the model generator adds the fact $\mathit{cute}(r_1)$ where r_1 has the property of being the only rabbit in the context. In other words, we have computed an interpretation of a the sentence *the rabbit is cute* in a given situational context. The example does not have any other locally minimal models.

Our analysis implements the interpretation of definites as local to a context which is restricted to a discourse. This focus is actually necessary for examples such as (5.23) where we cannot really expect to have a unique rabbit in London.

(5.23) *There once was a rabbit in London. **The rabbit** was Welsh.*

In our approach, quantification is restricted to the domain of discourse and therefore (5.23) is not taken to claim that there is a unique Welsh rabbit in London, but simply that there is a unique Welsh rabbit in London *which the speaker is talking about.* The analysis is here similar to that described by Groenendijk et al. [77]. Like them we relativise uniqueness to the domain of discourse, not to the world.

The definition of THE correctly expands to an unsatisfiable specification when we interpret the definite description in a discourse model where uniqueness does not hold even locally. In the following, we have a specification (5.24) where our model generator is unable to generate any model at all.

(5.24) *Jon has two rabbits. **The rabbit** is cute. (*)*

(5.25) $\mathsf{rbt}(r_1) \wedge \mathsf{has}(jon, r_1) \wedge \mathsf{rbt}(r_2) \wedge \mathsf{has}(jon, r_2) \wedge \textsc{The}(\mathsf{rbt})(\mathsf{cute})$

KIMBA will run infinitely when applied to unsatisfiable specifications such as (5.25). For practical purposes, we restrict the search space to a limited, but reasonable number of extensions in the domain of individuals. The model generation problem then becomes decidable.

5.1.6 Uniqueness and Lots of Rabbits

The definition of the generalised determiner THE in sense of Russell's approach actually defines two separate truth conditions for a definite. Our formulation THE in higher-order logic makes these two conditions explicit.

(5.26) $\textsc{The} \equiv \lambda P \lambda Q\ \textsc{Unique}(P) \wedge \textsc{Every}(P)(Q)$

The first condition is that the set P provided by the noun must have exactly one member. The second condition is that this member of P is also a member of Q, i.e., a member of the set given by the scope of quantification of the verb phrase. We will call the first truth condition the **unicity condition**, and the second condition the **subset condition**. The unicity condition seems to be an essential part of the semantics of singular definite descriptions, but for a variety of linguistic material, we can show that unicity with respect to a given context is a constraint which is too strong.

First, as Heim [78] notes, quantifiers may weaken and even annihilate uniqueness. Thus in (5.27), the definite description *the carrot* is unique only per rabbit: there is one carrot per rabbit but there may be many rabbits and therefore many carrots. In (5.28), uniqueness completely disappears: there may even be several carrots per rabbit.

(5.27) *Most rabbits who see one carrot, eat **the carrot**.*

(5.28) *If a rabbit sees a carrot, **the rabbit** eats **the carrot**.*

Second, definites often have some implicit dependency on some other noun phrase. Examples like the following are discussed for instance by Asher and Wada [79] in the general context of anaphora resolution.

(5.29) *When Jon's rabbit dreams of carrots, **the tail** twitches.*

(5.30) ***The tail** is perhaps the least known of the edible parts of a rabbit.*

Third, as example (5.30) indicates, not all uses of the singular definite article *the* really belong to a definite description. In cases like the following, the description is meant to be general.

(5.31) ***The rabbit** is a vermin in Australia.*

(5.32) ***The baseball cap** is an unlikely place to be for **the magician's rabbit**.*

Fifth, there are sentences such as (5.33) and (5.34) which explicitly deny that the description they contain give unique specifications of existing entities.

(5.33) ***The best way to breed rabbits** does not exist.*

(5.34) *There is no such thing as **the magical rabbit**.*

Finally, the uniquely identifying property may not be given by the definite description itself, but might have to be somehow inferred from the surrounding context. In (5.35) from Haddock [80], there is no unique hat in the context but the definite description *the hat* refers successfully to the hat that contains a rabbit. In (5.36), *the rabbit* actually refers to the rabbit in the hat, because to remove x from y, it must be the case that x is in y.

(5.35) *A magician has two hats and two rabbits. One rabbit is in a hat. The magician says: "Now watch attentively, I will make **the rabbit** in **the hat** disappear."*

(5.36) *Bugs and Bunny are rabbits. Bugs is in **the hat**. John removes **the rabbit** from **the hat**.*

In the face of such overwhelming evidence, it might seem best to give up uniqueness. There are a number of proposals however which manage to reconcile uniqueness with reality and on which we shall base our computational treatment [77, 81, 82]. In such proposals, uniqueness is not solely determined by the property denoted by the common noun occurring in the definite description. Additional contextual information also plays a rôle. In Cooper's approach [81] this property is a free variable whose value is determined by the context of use. Kadmon [82] asserts that it can be "accommodated, implicated or contextually supplied". Other approaches identify the property with the context set of the definite [77, 83]. In what follows, we adopt and experiment with a combination of these analyses.

5.2 Some Representations

We assume that the definite article *the* is assigned the semantic representation (5.1), which makes use of the definition of set intersection given in (5.2).

(5.37) $\textsc{The} \equiv \lambda P_1\ \lambda P_2\ \lambda Q\ \textsc{Unique}(P_1 \cap P_2) \wedge \textsc{Every}(P_1 \cap P_2)(Q)$

(5.38) $\cap \equiv \lambda P\ \lambda Q\ \lambda x\ P(x) \wedge Q(x)$

Here, P_2 is a first-order property that uniquely identifies the individual referent in the set P_1 which is given by the noun. We call the argument P_2 the **identifying property**. This form of representation requires that P_2 be given and therefore somehow determined. Determining such an identifying property is, as Kadmon remarks, an essentially pragmatic process which can involve accommodation, implicature and/or inference. A context will make many properties available, most of them are irrelevant for determining uniqueness. In what follows, we will manually determine and discuss arguments P_2 that give us correct analyses.

5.2.1 Simple Cases

For many definites, we do not need to give special identifying properties in order to obtain a correct analysis. In such cases, uniqueness holds for the context and we can use an unspecific identifying property $\top_\alpha = \lambda X_\alpha\ X \doteq X$ that holds for all entities in all domains $\mathcal{C}_\alpha$ of arbitrary type α. In what follows, $\top$ denotes the unspecific identifying property $\top_\iota$ for first-order individuals.

(5.39) *Jon has a rabbit.* ***The rabbit*** *is cute.*

(5.40) $\mathsf{rbt}(r_1) \wedge \mathsf{has}(jon, r_1) \wedge \textsc{The}(\mathsf{rbt})(\top)(\mathsf{cute})$

The model generation problem for (5.40) is trivial. We already have a suitable referent for our definite at hand, namely the rabbit r_1. The unicity condition constrains the resulting discourse model to one where r_1 is the only rabbit, and we therefore have no choice at all when generating a minimal model (5.40). But what about cases where such an individual is missing in the context?

(5.41) ***Jon's rabbit*** *is cute.*

(5.42) $\textsc{The}(\lambda x\ \mathsf{rbt}(x) \wedge \mathsf{has}(jon, x))(\top)(\mathsf{cute})$

Here, things are a bit more complicated. The minimal model (5.43) that we get could hardly be called intuitive.

(5.43) $\{\mathsf{rbt}(jon), \mathsf{has}(jon, jon), \mathsf{cute}(jon)\}$

What is missing in (5.42) is some necessary world knowledge that prohibits an oversimplification. The possessive *'s* that we modelled by the *has* predicate has an implicit truth condition, namely that the possessor cannot be identical with the possession. Alternatively, we could add the constraint that rabbits cannot possess other rabbits, only humans can. In general, we will simply add additional knowledge about relations as formulas to the specification. When adding an irreflexivity axiom (5.44) for *has*, the model (5.45) that we generate is correct.

(5.44) $\forall x\ \neg \mathit{has}(x,x)$

(5.45) $\{\mathit{rbt}(c_1), \mathit{has}(jon, c_1), \mathit{cute}(c_1)\}$

The constant c_1 is an automatically generated constant that comes from the iterative extension of the first-order universe in our model generator Kimba. There is no model that can satisfy (5.42) with a domain of individuals that only consists of the constant jon. Hence, Kimba extends the universe and computes the necessary facts in (5.45). Without a restriction to minimal models, the following models could be generated as well. It should be clear from these examples why we prefer minimal models in general.

(5.46) $\{\mathit{rbt}(c_1), \mathit{has}(jon, c_1), \mathit{has}(c_1, jon), \mathit{cute}(c_1)\}$

(5.47) $\{\mathit{rbt}(c_1), \mathit{has}(jon, c_1), \mathit{cute}(c_1), \mathit{cute}(jon)\}$

(5.48) $\{\mathit{rbt}(c_1), \mathit{has}(jon, c_1), \mathit{has}(c_1, jon), \mathit{cute}(c_1), \mathit{cute}(jon)\}$

The information that the noun phrase of a definite provides, together with some basic world knowledge, is usually sufficient for deciding between anaphor resolution and accommodation. In (5.49), the adjective *black* in the noun phrase *his black rabbit* makes it impossible to identify the white rabbit with Jon's rabbit in the discourse model—if we add the information that a white entity cannot be a black entity. As a result, Kimba accommodates a new rabbit although we have already introduced a rabbit to the model.

(5.49) *Jon loves his black rabbit.* ***The white rabbit*** *is boring.*

5.2.2 Donkeys, Context Sets, and Anaphoric Use

In the following sections, we consider uses of definites where an analysis based on an unspecific identifying property and minimal models is not sufficient in general.

5.2.3 Quantifiers and Donkey Sentences

Quantifiers can weaken or annihilate uniqueness in definites. Thus in (5.50), the definite descriptions *the rabbit* and *the carrot* cannot be understood as referring to unique entities in the context. There may be many rabbits, and also many carrots. A suitable semantic representation should be equivalent to the first-order formula (5.51).

(5.50) *If a rabbit sees a carrot,* ***the rabbit*** *eats* ***the carrot****.*

(5.51) $\forall x\ \forall y\ \mathit{rbt}(x) \wedge \mathit{crt}(y) \wedge \mathit{see}(x,y) \Rightarrow \mathit{eat}(x,y)$

Sentences such as (5.50) have been introduced by Geach and are known as "donkey sentences" in the literature. Donkey sentences are sentences where the anaphoric connection we perceive between an indefinite and a pronoun seems to conflict with the implicit existential quantification associated with the indefinite. In the following example, the indefinite in question is *some donkey*, the definite pronoun is *it*, and the

(5.52) *If Pedro owns some donkey, he beats it.*

The donkey sentences are the starting point for the dynamic/DRT approach to natural-language semantics. In Kamp and Reyle's DRT, sentences are treated within dynamic semantic representations in which logical connectives and bound variables model to some extend the dynamic behaviour of pronouns in natural language. These Discourse Representation Structures (DRS) have a first-order relativisation, i.e., translation. A semantic construction for a donkey sentence in DRT yields a universal quantification in the antecedent, and our donkey sentence (5.52) has the following relativisation.

(5.53) $\forall x\ \mathsf{donkey}(x) \land \mathsf{owns}(pedro, x) \Rightarrow \mathsf{beat}(pedro, x)$

Example (5.54) shows a semantic representation in the spirit of a DRS for sentence (5.50) where we still use our standard formalisation for definites.

(5.54) $\forall x\ \forall y\ \mathsf{rbt}(x) \land \mathsf{crt}(y) \land \mathsf{see}(x, y) \Rightarrow$
$\textsc{The}(\mathsf{rbt})(\top)(\lambda x\ \textsc{The}(\mathsf{crt})(\top)(\lambda y\ \mathsf{eat}(x, y)))$

This representation clearly is faulty. When we apply Kimba to (5.54) and a discourse model where we have several rabbits that each see a carrot, the specification becomes unsatisfiable. The uniqueness that is derived from the definition of The and the unspecific identifying property $\top$ is too strong. In what follows, we exchange the set $\top$ with identifying properties that are more selective.

5.2.4 Context Set Restrictions

The class of identifying properties that we propose for definites in the scope of quantifiers is derived from a syntactical restriction which originates in Kadmon's B_K set [82]. The B_K set refers to the set of variables that are bound higher up than the variable representing the definite noun phrase. This corresponds to the set of variables that are in the accessibility relation of the definite in DRT. A similar restriction has been referred to as context sets for instance by Westerstahl and others [83]. What we do is to introduce the implicit equality relation that holds between the anaphoric definite and the variables that are bound higher up in the semantic representation.

(5.55) $\forall x\ \forall y\ \mathsf{rbt}(x) \land \mathsf{crt}(y) \land \mathsf{see}(x, y) \Rightarrow$
$\textsc{The}(\mathsf{rbt})(\lambda r\ \ r \doteq x \lor r \doteq y)(\lambda z\ \textsc{The}(\mathsf{crt})(\lambda c\ \ c \doteq x \lor c \doteq y)(\lambda u\ \mathsf{eat}(z, u)))$

The idea is to relativise uniqueness to an equality relation over the set of individuals in the discourse mode, in this case, to the individuals related to the variables bound by a quantifier. The effect here is that uniqueness is relativised to arbitrary rabbit-carrot pairs and hence there is no uniqueness. Given such a specification and a discourse model with one rabbit, say *Bugs*, that sees two carrots, KIMBA will return a model where Bugs sees and eats both carrots. In contrast, given a discourse model where Bugs sees two carrots, but explicitely eats only one, KIMBA will find the specification unsatisfiable and no model will be generated.

The following example is very similar, which is hidden somewhat by the different quantifier. The semantic representation is given in (5.57).

(5.56) *Most rabbits that see a carrot, eat* ***the carrot****.*

(5.57) $\textsc{Most}(\mathit{rbt})(\lambda x\ \forall y\ \mathit{crt}(y) \land \mathit{see}(x,y) \Rightarrow$
$\textsc{The}(\mathit{crt})(\lambda z\ z \doteq x \lor z \doteq y)(\lambda u\ \mathit{eat}(x,u)))$

Again, we have some quantifier variables that are used for identifying a referent in the definite. Because we use linguistic quantification with MOST, the quantifier variable x is the bound variable in the λ-abstraction of the scope given by the relative sentence *that sees a carrot* and the verb phrase. Apart from the fact that it is a challenging task to design a semantic construction method for such sentences, our treatment of the definite remains the same. With the identifying property $\lambda z\ z \doteq x \lor z \doteq y$, uniqueness actually disappears, and we can have for instance a majority of rabbits that see and eat more than one carrot in the discourse model without losing satisfiability.

The semantic representations that we have experimented with so far use identifying properties that are determined only by the context set as a whole. We use the set of all accessible variables because it would be very difficult in general to identify just the right variable automatically. In the examples above, we have some syntactic parallelism between the entities represented by the quantifier and the definite, e.g., *a carrot* and *the carrot*, but this syntactic parallelism is not necessarily sufficient for identifying the right variable in all cases. Example (5.58) is a variant of (5.56) where we would find it impossible to detect the dependency between the quantifier and the definite by syntactic means alone.

(5.58) *Every rabbit that sees a carrot, eats* ***the healthy vegetable****.*

The encoding (5.59) works for instance in the discourse model (5.60) where we have the minimal model (5.61). The formula $\textsc{No}(\mathit{rbt})(\mathit{crt})$ gives the sortal information that rabbits are not carrots, and $\textsc{Every}(\mathit{crt})(\mathit{veg})$ adds the knowledge that carrots are vegetables.

(5.59) $\textsc{Every}(\mathit{rbt})(\lambda x\ \forall y\ \mathit{crt}(y) \land \mathit{see}(x,y) \Rightarrow$
$\textsc{The}(\mathit{veg} \cap \mathit{hlthy})(\lambda z\ z \doteq x \lor z \doteq y)(\lambda u\ \mathit{eat}(x,u)))$

(5.60) $\mathit{crt}(c_1) \land \mathit{rbt}(r_1) \land \mathit{see}(r_1,c_1) \land \textsc{No}(\mathit{rbt})(\mathit{crt}) \land \textsc{Every}(\mathit{crt})(\mathit{veg})$

(5.61) $\{\mathit{crt}(c_1), \mathit{hlthy}(c_1), \mathit{veg}(c_1), \mathit{rbt}(r_1), \mathit{see}(r_1, c_1), \mathit{eat}(r_1, c_1)\}$

The encoding works just as well in cases where we have several rabbits and carrots. In each case, the entity that is both healthy and a vegetable is identified with the carrot mentioned in the antecedent clause for each rabbit-carrot pair.

5.2.5 The Treatment of Names

As we have discussed in the last section, some linguistic theories predict a dependency between an anaphoric definite and the quantifier variables in which scope it occurs. Such a dependency is actually a property of all singular anaphora. If an anaphor cannot be linked to a quantifier variable, it remains unresolved and the discourse becomes ill-formed. What we have not discussed so far is how names of individuals, which can be used just as well for anaphor resolution, fit into the picture.

In DRT, names are modelled by unary predicates and their occurrence introduces existential quantification over the whole representation. In $\mathcal{MQL}$, we have the names of individuals represented simply as constants because we have a Herbrand-like interpretation of constants in all $\mathcal{MQL}$ interpretations. In the context set restriction that we use, names must be considered as potential referents for anaphoric definites. The following example illustrates the extension of the context set to names.

(5.62) *When Jon visits a doctor,* ***the man*** *is happy.*

(5.63) $\forall x\ \mathit{doc}(x) \wedge \mathit{visit}(jon, x) \Rightarrow$
$\textsc{The}(\mathit{man})(\lambda u\ u \doteq x \vee u \doteq jon)(\mathit{happy})$

The discourse is ambiguous because it is not entirely clear to whom *the man* refers to, Jon or the doctor. An identifying property $\lambda u\ u \doteq x \vee u \doteq jon$ that leaves the computation of the actual resolution to the model generator allows us to compute two minimal models, one where Jon is the man and is happy when he visits any doctor, and one where every (male) doctor is happy when Jon visits him.

5.2.6 Restrictions with Knowledge

What is interesting in our minimal-model-based approach to anaphor resolution is that additional contextual knowledge can be added which controls the resolution. For instance in example (5.62), if we add the information that Jon only visits female doctors, the computed models change accordingly. Note that an encoding of definites with context sets makes it impossible to accommodate new individuals for the definite—if we use context sets, we implicitly assume an anaphoric use of the definite. For some definites, the suitable identifying properties that are based on some pragmatically given restrictions still allow accommodation.

5.2.7 Implicit Knowledge and Accommodation

In example (5.64), the uniqueness in the definite *the rabbit* relates to *the rabbit in the hat*. The proposed logical specification is given in (5.65).

(5.64) *Bugs and Bunny are rabbits. Bugs is in the hat. Jon removes* ***the rabbit*** *from* ***the hat.***

(5.65) $\textsc{The}(\mathsf{rbt})(\lambda v\ \mathsf{inhat}(v))(\lambda x\ (\textsc{The}(\mathsf{hat})(\top)(\lambda y\ \mathsf{rmv}(jon, x, y))))$

The context set restriction from the last section would not help here because both Bugs and Bunny are in the context set. Instead, we use an identifying property $\lambda x\ \mathsf{inhat}(x)$ that is a pragmatic restriction inferable only from what we know about the world: removing x out of y implies that x was in y[2]. Hence, the rabbit that is removed from the hat must have been in the hat. The locally minimal model generated by Kimba for specification (5.65) is as expected (5.66).

(5.66) $\{\mathsf{rbt}(bugs), \mathsf{rbt}(bunny), \mathsf{inhat}(bugs), \mathsf{rmv}(jon, bugs, hat)\}$

Now suppose there is no rabbit known to be in a hat. In that case, Kimba accommodates the fact that one of the rabbit was indeed in the hat, and yields two possible minimal interpretations that satisfy the representation. In the first case (5.68), Bugs is in the hat, and in the second case (5.68), Bunny is in the hat.

(5.67) $\{\mathsf{rbt}(bugs), \mathsf{rbt}(bunny), \mathsf{inhat}(bugs), \mathsf{rmv}(jon, bugs, hat)\}$

(5.68) $\{\mathsf{rbt}(bugs), \mathsf{rbt}(bunny), \mathsf{inhat}(bunny), \mathsf{rmv}(jon, bunny, hat)\}$

In general, Kimba will accommodate any fact that is necessary provided that it is consistent with the rest of the specification. An interesting question that would be worth exploring in this context is that of computable constraints on accommodation. We leave the question open for now and return to it later in Section 5.2.9.

Another variation[3] on (5.64) is the following:

(5.69) *Bugs and Bunny are rabbits. Bugs is in the hat. Jon removes* ***the birthday present*** *from the hat.*

In this case, we may infer that the birthday present is Bugs. And so does Kimba in the locally minimal model.

[2] For simplicity, we ignore temporal issues here.
[3] Thanks to Bonnie Webber for this particular version of the example. Zeevat [84] proposes a similar example namely: "A man died in a car accident last night. The Amsterdam father of four had been drinking."

5.2.8 Bridging

Bridging refers to the dependency of a definite to some other noun phrase such as in the following example.

(5.70) *Jon's rabbit dreams.* ***The tail*** *twitches.*

As ***The tail*** here refers to the tail of Jon's rabbit, an acceptable interpretation of ***The tail*** *twitches* must resolve the definite to the tail of Jon's rabbit. A correct treatment of bridging requires that we make explicit the relationship of rabbits and tails by a formula such as (5.71). The formula states that for each rabbit we have a unique tail that is a part of that rabbit.

(5.71) $\forall x\ \mathit{rbt}(x) \Rightarrow \textsc{Unique}(\lambda y\ \mathit{tail}(y) \wedge \mathit{has}(x,y))$

If we add formula (5.71) to the logically encoded context, the locally minimal model of the semantic representation $\textsc{The}(\mathit{tail})(\top)(\mathit{twitch})$ represents the correct reading. The model identifies a tail which is introduced by formula (5.71) as that (only) tail that both belongs to Jon's rabbit and which also twitches. Note that the semantic representation does not give a specific identifying property.

However, in cases such as the following, it is reasonable that a suitable identifying property must somehow introduce the implicit dependency between the definite and the dependent noun.

(5.72) *If a rabbit dreams,* ***the tail*** *twitches.*

Example (5.72) is a donkey sentence that cannot be dealt with by simply relating the definite to the context set because we only have a variable for rabbits, and not a variable for tails. Below, we give the semantic representations that do the trick, and the natural-language sentence that they actually stand for.

(5.73) *When Jon's rabbit dreams,* ***the tail of Jon's rabbit*** *twitches.*

(5.74) $\textsc{The}(\lambda x\ \mathit{rbt}(x) \wedge \mathit{has}(jon,x))(\top)(\mathit{dream}) \Rightarrow$
$\textsc{The}(\mathit{tail})(\lambda u\ \textsc{The}(\lambda v\ \mathit{rbt}(v) \wedge \mathit{has}(jon,v))$
$(\top)(\lambda v\ \mathit{of}(u,v)))(\mathit{twitch})$

(5.75) *If a rabbit dreams,* ***the tail of it*** *twitches.*

(5.76) $\forall x\ \mathit{rbt}(x) \Rightarrow\ \textsc{The}(\mathit{tail})(\lambda u\ \textsc{The}(\mathit{rbt})$
$(\top)(\lambda v\ \mathit{of}(u,v)))(\mathit{twitch})$

As we can see, instead of an identifying property that simply relates two variables, we introduce an *of*-relation between two variables. As in the case of quantifier scoping, bridging introduces a relation, but this relation may be more complex than the simple equality of entities. The *of*-relation is a placeholder for a variety of actual relations that can hold between the definite and the dependent noun. For instance, in (5.77), the waiting room refers to the waiting rooms of the doctors' offices, not the waiting room of each doctor. Still, we can use *of* as long as we only want to make sure that an analysis with local uniqueness remains possible.

(5.77) *If Jon visits a doctor,* ***the waiting room*** *empties.*

(5.78) $\forall x\ \mathit{doctor}(x) \wedge \mathit{visit}(jon, x) \Rightarrow$
$\textsc{The}(\mathit{waitrm})(\lambda v\ \mathit{of}(x, v))(\mathit{empties})$

5.2.9 Simple Cases Revisited

The identifying properties P_2 that we have used so far can be roughly classified into two categories.

- The identifying property is $\top$ for simple cases.
- The identifying property relates the definite to a set of variables that are bound higher up in the semantic representation. A suitable relation cannot be determined from syntactical structure alone, but must sometimes be inferred from world knowledge.

It is not easy to classify the use of a definite as a simple case or as one where we need a non-trivial restriction on P_2. The problem is that the simple cases with the identifying property $\top$ sometimes work for sentences where a correct analysis should yield an unsatisfiable specification. This occurs mainly in connection with accommodation and the dynamic behaviour of definites that should be interpreted exclusively as anaphora.

(5.79) *Jon has no rabbit. The rabbit is cute. (*)*

(5.80) $\textsc{No}(\mathit{rbt})(\lambda x \mathit{has}(jon, x)) \wedge \textsc{The}(\mathit{rbt})(\top)(\mathit{cute})$

What we have here is a sentence (5.79) where a naive modelation of a discourse model in (5.80) will not prevent that we can generate a minimal model. Model generation implements an almost unconstrained form of accommodation. In our example, KIMBA simply accommodates an individual that is a cute rabbit but necessarily not Jon's one. Hence, formalisation (5.80) is at least odd as a semantic representation of (5.79). What is missing in our representation is that the definite should be treated as an anaphor.

5.2.10 Non-resolvable Anaphora in DRT

Solving the problem of anaphor resolution in the presence of dynamic quantifier scope and accessibility relations is the domain of dynamic logics. We might be tempted to say that our problem here is really that of a static logic representation. Arguably, we must be careful to use a suitable relativisation of intrinsically dynamic phenomena, such as the accessibility of discourse referents, in our logic $\mathcal{MQL}$. Consider for instance the following sentence.

(5.81) *Jon has no rabbit. He likes it.*

In DRT, the semantic construction process yields a DRT whose first-order form is given by (5.82). The pronoun *it* is resolved with a referent that is actually not accessible. The equation $y \doteq x$ which states that *it* refers to the non-existing rabbit results in a formula which is not well-formed. DRT on this ground rejects the semantic construction of (5.82).

(5.82) $\neg\exists y\ (\neg\exists x\ \mathit{rbt}(x) \wedge \mathit{has}(jon, x)) \wedge \mathit{like}(jon, y) \wedge y \doteq x$

If we use a DRT relativisation (5.83) for the example (5.79) where the definite is treated in the same spirit as an anaphor, we also have an ill-formed formula.

(5.83) $\neg(\exists x\ \mathit{rbt}(x) \wedge \mathit{has}(jon, x)) \wedge \textsc{The}(\mathit{rbt})(\lambda u\ u \doteq x)(\mathit{cute})$

5.2.11 Definites Are Not Anaphora

The DRT-solution to non-resolvable anaphora relies on a semantic construction process that detects some syntactically ill-formed uses of anaphora before a semantic representation is presented. The ill-formedness in the representation comes from equations where one of the variable occurs free. The equations are introduced by the provisional instantiation of pronouns, i.e., by anaphor resolution.

For definites, there is no equivalent and generally useful filter mechanism does not exist because definites are not always anaphora. As mentioned earlier, definites sometimes presuppose the existence of discourse referents that have not been introduced yet (5.84), or trigger bridging (5.85) which only relates the definite to a previously mentioned referent in some indirect way. Finally, we have the use of singular *the*-phrases like (5.86) that are meant as general statements.

(5.84) ***Jon's new rabbit*** *is black.*

(5.85) *Jon has a new rabbit.* ***The tail*** *is black.*

(5.86) ***The rabbit*** *is a vermin in Australia.*

In any of these cases, if we use identifying properties that restrict the interpretation of definites to anaphoric uses, we will obtain unsatisfiable specifications. Unfortunately, linguistics so far has not provided us with computational methods for distinguishing anaphoric and non-anaphoric uses of definites.

Model generation for locally minimal models prefers anaphoric use over non-anaphoric use, because locally minimal models minimise the universe. This preference seems to match the empirical data. Apart from that, we must allow accommodation in all cases where we have consistency of the assumed facts in the model. In cases where only accommodation of new individuals yields a model, model generation on its own does not really have a means to distinguish valid and non-valid forms of accommodation.

5.2.12 Non-existence

An interesting phenomenon are sentences which explicitely deny the uniqueness that the definite implicitly formulates.

(5.87) ***The golden mountain*** *does not exist.*

According to Russell, sentences such as (5.87) have two possible analyses, one where negation has a wide scope over the quantificational complex which represents the contribution of the definite, and one with narrow scope. The first analysis is equivalent to our formulation (5.88), while the second is equivalent to (5.89) for our example (5.87).

(5.88) $\textsc{The}(\mathsf{goldMountain})(\top)(\lambda x\ \neg\top(x))$

(5.89) $\neg\textsc{The}(\mathsf{goldMountain})(\top)(\top)$

Both analyses are not intuitive, as can be shown by investigating their models. In (5.88), we state that there is a unique golden Mountain that has the property of non-existence—note that $\top$ denotes the property that all individuals in the discourse model have. This formalisation is unsatisfiable, because model generation must accommodate an individual on the one hand, and show that it does not belong to the discourse model at the same time. This is inconsistent.

As Kamp and Reyle argue [1], the second formalisation (5.89) says something which is closer to the intuitive meaning of (5.87). It says that there is no unique object which has the property of being the golden mountain that exist. Unfortunately, our model generator proves that the formalisation still is not correct. It has a non-minimal model (5.90) that instantiates another potentially valid interpretation of the formalisation.

(5.90) $\{\mathsf{goldMountain}(c_1), \mathsf{goldMountain}(c_2)\}$

Informally, Russell's semantic representation can be validated by assuming that there are two golden mountains c_1 and c_2. The model (5.90) reveals the subtle problem that the Russellian encodings for definites do not always preserve entailment. In this special case, we cannot simply fix the problem by giving a suitable identifying property because an identifying property that determines an empty set is not possible.

The example shows that it sometimes pays to consider non-minimal models as well as minimal ones when we use model generation in the analysis of semantic representations.

5.3 What We Have Learned so Far

We have shown that model generation can provide interesting insights from existing semantic theories of definites with a computational interpretation that combines reasoning on linguistic and world knowledge. Given a suitable uniquely identifying property and a logical encoding of the context, the model generator either identifies the referent of a definite noun phrase with some already existing entity (coreference) or adds a new entity to the model (accommodation). We have determined some classes of identifying properties for our formalisation for some well-known examples from the literature. This by itself is not entirely trivial, and we know of no existing approach to the semantics of definites that can treat the range of examples that we have presented so far.

For the analysis of definites, the model generator KIMBA provides an adequate notion of minimality. Locally minimal models validate a specification within the smallest possible domain of individuals. As a result, our method for definites favours coreference over accommodation whenever coreferences can be assumed consistently. As has been stated first by Strawson [71], a correct computational treatment of definites asks for a method that can handle accommodation in this way.

Our approach often makes the meaning postulates explicit which are necessary for natural-language interpretation. A formalisation which leaves out this important part of a meaning quickly leads to oversimplification. We can use model generation as a software engineering tool and apply it to complex formalisations where we may not be sure that we actually have a correct semantic representation. For practical research in model-theoretic semantics, model generation helps to formulate and verify linguistic theories and allows us to experiment with their predictions.

Minimal model generation correctly treats the class of definites that we have called “simple cases”, i.e., where the identifying property can be safely chosen as $\top$, provided that we have a correct logic modelation of the context of an utterance and a suitable representation in which the definite is embedded. These simple case not only include some anaphoric uses of definites, but also some common forms of bridging and accomodation. These often require reasoning about the world which model generation provides as a built-in feature.

For other examples, we have shown that minimal model generation is general enough to allow for an implementation of contemporary linguistic theories [77, 78, 81–83]. Our “identifying properties” can uniformally represent the various proposed constraints on the interpretation of definites.

Unfortunately, these theories do not yield specific methods for computing the identifying properties in general. For instance, as we have seen in Section 5.2.6, the use of a syntactic constraint such as a context set does block accommodation. If we want to employ the context set restriction correctly, we must somehow decide when to use it and when we should stay with a less restrictive formalisation that allows accommodation. It is unlikely that this problem has a sound computable answer. In Section 5.2.4, we have shown for instance that there can be no purely syntactic method that identifies anaphoric uses of definites. It remains unclear how we can then satisfactory model an automatic context set restriction when we cannot even decide whether a definite is used as an anaphor or not. The contemporary theories state that the relation which determines the unicity of a definite must be 'referred from the context' or 'is essentially a process of accommodation, implicature and/or inference'. The semantic theories that we know remain vague on the nature of the consequence relation that determines the inference that is referred to.

However, as Kamp and Reyle [1] note, it would be a non-trivial task to identify and describe the different purposes to which singular *the*-phrases can be put, let alone the actual constraints on their interpretation. Hence, our analysis remains incomplete in that we do not have more reliable computational means which automatically determine identifying properties from the linguistic data. What we have presented in this chapter is only a starting point for a more throughout analysis of definites.

6

Reciprocity

Once, indeed, a problem was brought to me, and I solved it, obtaining very many solutions; I went into it fully, and found that there were 2678 valid answers. I marvelled at this, only to discover - when I spoke of it - that I was reckoned a simpleton or an incompetent, and strangers looked on me with suspicion.
(Abu Kamil)

Overview: Linguistic theories on reciprocal expressions identify a semantic principle, the Strongest Meaning Hypothesis, as the key concept for determining the logical contributions of reciprocals. Based on this theory, we present a new and simple method for computing reciprocal meaning by model generation.

6.1 Introduction

Research on the English reciprocal expressions *each other* and *one another* has uncovered a variety of meaning contributions that a reciprocal can provide. Consider the following sentences.

(6.1) *The students like each other.*

(6.2) *The students gave each other measles.*

(6.3) *The students stare at each other in surprise.*

(6.4) *The students follow each other into the ballroom.*

In each case, the reciprocal implicitly formulates truth conditions for the interpretation of the relation that is expressed by the verb phrase. These truth conditions are different for each example. For instance, we can accept (6.1) to be true only if for each pair x and y of different students holds that x likes y. An analogous interpretation would be invalid in the case of (6.2)–(6.4) where not all pairs in the reciprocal group *the students* can consistently be in the scope relation.

Dalrymple et al. [85] argues that the variety of reciprocal meaning can be reduced to six different classes of reciprocal semantics. The correct choice amongst these classes is determined by a purely semantic principle, the Strongest Meaning Hypothesis. In its most general form, this principle can be stated as follows.

> **Strongest Meaning Hypothesis (SMH)**: *The meaning of an expression S in a context Γ is that meaning which corresponds to the interpretation of the logically strongest semantic representation ϕ available for S that is consistent with Γ.*

The SMH as formulated here lies at the border of semantics construction and natural-language interpretation. Each of the sentences (6.1)–(6.4) exemplifies a separate class of reciprocal semantics and the truth conditions expressed in the other examples are either too strong or too weak to cover the meaning of the reciprocal. The SMH identifies that class of reciprocity as the correct one whose contribution to the scope relation is the strongest one that still is consistent with the information provided by the context.

It can be argued that the variation of meaning in reciprocal expressions lies more in the way in which the scope relations can be interpreted over the antecedent groups rather than in some variation of the semantics of reciprocal expressions. We present a new theory of reciprocals where the SMH can be understood as a principle of interpretation alone. In our approach, we implement the preference for certain interpretations as a form of minimal model reasoning. Our version of the SMH maximises the logical contribution of the reciprocal to the scope relation in the set of logical models of a discourse. We use only one semantic representation for reciprocal expressions and model their complex behaviour by a refinement of a well-known minimal model constraint that has found some use for instance in diagnosis applications. As usual, we will experimentally verify our theory with our model generator KIMBA.

6.2 Exploring the Meaning of *Each Other*

In the linguistic literature, the starting point for the semantic analysis of reciprocals are often sentences such as

(6.5) *Jon and Bill saw each other.*

where two discourse participants are in a reciprocal relation expressed by the verb phrase. Such sentences are particularly easy to analyse. The English reciprocal expressions *each other* and *one another* are frequently represented as diadic quantifiers over some first-order set called the **antecedent group**, and a binary first-order relation called the **scope relation**. In what follows, we will use the symbol RCP for such reciprocal quantifiers. In higher-order logics, example (6.5) is then represented as follows.

(6.6) $\textsc{Rcp}(\{\mathit{jon}, \mathit{bill}\})(\lambda y \lambda x\ \mathsf{saw}(x, y))$

Here, $\{jon, bill\}$ denotes the antecedent group, i.e., a first-order property that holds exactly for jon and $bill$, namely $\lambda x\ (x = jon \vee x = bill)$.

When groups $P_{\iota \to o}$ of just two members are considered, each group member is required to stand in some scope relation $R_{\iota \to \iota \to o}$ to the other member, where R is provided by the verb phrase. This truth condition for reciprocals can be formalised by definition (6.7)[1]. The formalisation uses the set exclusion operator '/' which is specified by the λ-term in definition (6.8).

(6.7) $\textsc{Rcp} \equiv \lambda P \lambda R\ \textsc{Every}(P)(\lambda x\ \textsc{Every}(P/\{x\})(\lambda y\ R(x,y))$

(6.8) $/ \equiv \lambda P \lambda Q \lambda x\ P(x) \wedge \neg Q(x)$

The minimal model (6.9) of the semantic representation (6.6) under the definition (6.7) is as expected, and represents nicely the natural-language interpretation of (6.5).

(6.9) $\{\mathsf{saw}(bill, jon), \mathsf{saw}(jon, bill)\}$

6.2.1 Reciprocals for Larger Groups

The application of the semantic representation (6.7) to larger antecedent groups suggests for instance that

(6.10) *House of Commons etiquette requires **legislators** to address only the speaker of the House and **refer to each other indirectly**.*

states that each legislator is required to refer to every other one indirectly. For example (6.10), this is indeed the intended meaning. However, the research on reciprocals has shown that the truth conditions implied by definition (6.7) turn out to be the wrong ones for many cases where the antecedent group has more than two members. Statement (6.11) from J.M. Barrie's *Peter Pan* exemplifies a meaning of the reciprocal where every member is claimed to relate to at least one other group member, but not necessarily to relate to *every* other one.

(6.11) *"The captain!" said the **pirates**, **staring at each other in surprise**.*

It is impossible for the pirates to stare at each other such that each pirate stares at every other pirate at the same time. The truth conditions for the reciprocal that is implied by this example is obviously weaker than the one that we have for our first example. In other words, the binary relation R whose interpretation is partially determined by the semantics of the reciprocal can be interpreted more freely. As the following sentences further illustrate, R may instantiate a variety of relations.

(6.12) *The blocks are stacked on top of each other.*

(6.13) *Five Boston pitchers sat alongside each other.*

(6.14) *Most people at the party are married to each other.*

[1] An equivalent definition is given by Carpenter [76].

Example (6.12) can be interpreted as true in contexts where the blocks mentioned are stacked in any way that is possible, including for instance a pyramid or a tower of blocks. Sentence (6.12) can never have the literal meaning that each block has the property of being stacked on top of each other one.

Sentence (6.13) describes a situation where five Boston pitchers sit on a bench. Naturally, we expect the ones in the middle each to have just two other pitchers sitting alongside, and the ones at the end to have just one pitcher on their side. The possible relation R for the situation at hand is more constrained than the previous one in that each pitcher must have another pitcher he sits alongside to. In sentence (6.12), the lowest layer of blocks may not on top of any other ones.

Finally, the reciprocal in (6.14) only means that we have a spouse for most persons that are at the party. The binary relation R must be one that assigns to each man in the group of most persons that is referred to a woman he is married to, and vice versa. The relation R in question does not even relate all married people with each other directly or indirectly.

6.2.2 Classifying Reciprocal Meaning

Dalrymple et al. [85] provides a classification system whose classes approximate the various forms of reciprocal meaning. Based on the earlier works of Langendoen [86], they propose six classes of reciprocal meaning: Strong Reciprocity, Strong Alternative Reciprocity, Intermediate Reciprocity, One-Way Weak Reciprocity, Intermediate Alternative Reciprocity, and Inclusive Alternative Ordering. Every occurrence of a reciprocal expression falls under one or several of these categories and each class can be given a precise semantic representation in higher-order logic. In what follows, we describe the aforementioned classes of reciprocal semantics and give their formalisations in higher-order logic.

6.2.3 Strong Reciprocity

The strictest form of reciprocity, **Strong Reciprocity** (SR) refers to reciprocals where the relation R holds for each member x of the antecedent set P and each other member y. As discussed above, the following statement (6.15) is an example for SR.

(6.15) *Legislators must refer to each other indirectly.*

The truth condition for such a reciprocal is twofold. First, the reciprocal can only be true in contexts where the antecedent set—in our case, the set of legislators—has at least two members. Second, we have the aforementioned condition on the binary relation R. In (6.15), R is the property of referring indirectly, and its strong meaning is that $R(x, y)$ holds for each legislator x and each other legislator y. All this leads to the following formalisation for reciprocity.

(6.16) $\text{R}\textsc{cp}_{SR} \equiv$
$\lambda P \lambda R\ \textsc{Card}^{\geq 2}(P) \wedge$
$\textsc{Every}(P)(\lambda x\ \textsc{Every}(P/\{x\})(\lambda y\ R(x,y))$

The quantifier $\textsc{Card}^{\geq 2}$, i.e., CARDinality ≥ 2, has a higher-order definition (6.17) that makes use of standard first-order quantification and equality.

(6.17) $\textsc{Card}^{\geq 2} \equiv \lambda P\ \exists x\ \exists y\ \neg(x = y) \wedge P(x) \wedge P(y)))$

6.2.4 One-Way Weak Reciprocity

The example from *Peter Pan* exhibits a form of reciprocity where each member of the antecedent set must be the subject of the relation R and be related with at least one other member as the object. Strong Reciprocity does not hold.

(6.18) *"The captain!" said the* ***pirates****,* ***staring at each other in surprise****.*

The form of reciprocity in example (6.18) is called **One-Way Weak Reciprocity** (OWR). Its formalisation is as follows.

(6.19) $\text{R}\textsc{cp}_{OWR} \equiv$
$\lambda P \lambda R\ \textsc{Card}^{\geq 2}(P) \wedge$
$\textsc{Every}(P)(\lambda x\ \textsc{Some}(P/\{x\})(\lambda y\ R(x,y))$

One-Way Weak Reciprocity is a strictly weaker form than Strong Reciprocity, i.e., One-Way Weak Reciprocity does hold in all cases where Strong Reciprocity holds, but not necessarily vice versa. The weakening corresponds directly to the exchange of one of the second of the two universal determiners EVERY in SR by the existential determiner SOME.

6.2.5 Inclusive Alternative Ordering

A further weakening of the reciprocity conditions occurs in examples such as (6.20) where not every member of the antecedent set P must be a subject of the relation R. Instead, it suffices that every member of P is at least the object of the relation.

(6.20) *He and scores of other inmates slept on* ***foot-wide wooden planks stacked atop each other****—like sardines in a can—in garage-sized holes in the ground.*

A slight modification of the definition of $\text{R}\textsc{cp}_{OWR}$ suffices to obtain the formalisation of **Inclusive Alternative Ordering** (IAO). IAO is the weakest known form of reciprocity.

(6.21) $\text{R}\textsc{cp}_{IAO} \equiv$
$\lambda P \lambda R\ \textsc{Card}^{\geq 2}(P) \wedge$
$\textsc{Every}(P)(\lambda x\ \textsc{Some}(P/\{x\})(\lambda y\ R(x,y) \vee R(y,x))$

6.2.6 Intermediate Reciprocity

Dalrymple et al. [85] cite the following statement from the *New York Times* for exemplifying a form of reciprocal meaning called **Intermediate Reciprocity** (IR). The statement can be true despite the impossibility of each group member sitting alongside.

(6.22) *As the preposterous horde crowded around, waiting for the likes of Evans and Mike Greenwell, **five Boston pitchers sat alongside each other**: Larry Andersen, Jeff Reardon, Jeff Gray, Dennis Lamp and Tom Bolton.*

Informally, IR states that the the relation R relates all members of P directly or indirectly via a sequence of members of P. In other words, every two different elements x and y must be in the transitive closure of R with respect to P. The IR form of reciprocity is modelled by the following definition.

(6.23) $\textsc{Rcp}_{IR} \equiv$
$\lambda P \lambda R\ \textsc{Card}^{\geq 2}(P) \wedge$
$\textsc{Every}(P)(\lambda x\ \textsc{Every}(P/\{x\})(\lambda y\ \textsc{TransCl}(P)(R)(x, y)))$

The formula $\textsc{TransCl}(R)(P)(x, y)$ denotes that there is a sequence $z_0, \ldots, z_n$ of elements of P such that $z_0 = x$, $z_n = y$ and $R(z_i, z_{i+1})$ for all $i < n$.

6.2.7 Intermediate Alternative Reciprocity

Examples (6.24)[2] and (6.25) exhibit yet another variant of reciprocity whose truth conditions are different from all other ones that we have investigates so far.

(6.24) *The students in Mrs. Smith's class gave each other measles.*

(6.25) *Instead, **countless stones**–each weighting an average of 300 pounds–**are arranged on top of each other** and are held in place by their own mass and the force of flying buttresses against the walls.*

The form of reciprocity here is called **Intermediate Alternative Reciprocity** (IAR) and is characterised by a relation R that connects all members of the antecedent set in the fashion of a strongly connected acyclic graph. It suffices that each member x is related to every other member y via a chain of R-relations where we ignore which way the pairs are connected. In the situation described by sentence (6.25), we have a cathedral which is built of stones arranged in a pattern like a brick wall. The relation we have here is not symmetric, but every brick is part of single connected structure. Dalrymple et al. claim that the sentence would be false in a context where the stones are arranged in a multiplicity of piles.

The formalisation of IAR is a variant of Intermediate Reciprocity where the concept of a transitive closure is weakened.

[2] It can be argued that this example illustrates IAO rather that IAR.

(6.26) $\textsc{Rcp}_{IAR} \equiv$
$\lambda P \lambda R\ \textsc{Card}^{\geq 2}(P) \wedge$
$\textsc{Every}(P)(\lambda x\ \textsc{Every}(P/\{x\})$
$(\lambda y\ \textsc{TransCl}(P)(\lambda u \lambda v\ R(u,v) \vee R(v,u))(x,y)))$

The formula $\textsc{TransCl}(P)(\lambda u \lambda v\ R(u,v) \vee R(v,u))$ denotes that there is a sequence $z_0, \ldots, z_n$ of elements of P such that $z_0 = x$, $z_n = y$ and $R(z_i, z_{i+1})$ **or** $R(z_{i+1}, z_i)$ for all $i < n$.

6.2.8 Strong Alternative Reciprocity

Strong Alternative Reciprocity is a weakened form of Strong Reciprocity and an exceptional class as there still are, as far as we know, no examples that can univocally be identified with it. All known examples of SAR also meet the truth conditions of SR. Hence, the SAR class owes its existence only to the parameterisation and classification scheme of Dalrymple et al. that we describe later in Section 6.2.9. SAR's definition in higher-order logic is as follows.

(6.27) $\textsc{Rcp}_{SR} \equiv$
$\lambda P \lambda R\ \textsc{Card}^{\geq 2}(P) \wedge$
$\textsc{Every}(P)(\lambda x\ \textsc{Every}(P/\{x\})(\lambda y\ R(x,y) \vee R(y,x))$

6.2.9 Parameterisation

The differences between the various definitions of Rcp can be parameterised. This parameterisation was helpful to Dalrymple et al. to discover new forms of reciprocity that have not previously been considered in the literature.

The first parameter of variation determines how the scope relation R should cover the domain P. The examples we have seen each fall into the following categories.

- each pair of different individuals in P may be required to participate in the relation R directly (Ful).
- each pair of different individuals in P may be required to participate in the relation R directly or indirectly (Lin).
- each individuals in P may be required to participate in the relation R with another one (Tot).

The second parameter concerns how Ful, Lin, and Tot operate on the reciprocal's parameter R: whether the relation R that reciprocity requires between individuals in the domain is actually the extension of the reciprocal's scope, or the extension where we ignore the direction in which the relation R holds by adding inverse pairs R^{-1}. By $R^{\vee}$, we denote the parameter where we use the extension $R \cup R^{-1}$ in this way.

6.2.10 The Landscape of Reciprocity

The following table shows how the two parameters determine the various semantics of reciprocals. It presents the complete landscape of reciprocal mean-

$$\begin{tikzcd}
 & \text{SR} \arrow[dl] \arrow[dr] & \\
\text{SAR} \arrow[d] \arrow[drr] & & \text{IR} \arrow[d] \arrow[dll] \\
\text{OWR} \arrow[dr] & & \text{IAR} \arrow[dl] \\
 & \text{IAO} &
\end{tikzcd}$$

Fig. 6.1. The partial order on the logical contribution of reciprocals

ing expressible by the variations of the two parameters. The landscape defines a partial order of the semantic contributions of the reciprocal meanings that is shown in Figure 6.1.

	Ful	Lin	Tot
R	SR	IR	OWR
$R^{\vee}$	SAR	IAR	IAO

The Strong Reciprocity (SR) form of reciprocity requires that each pair of different individuals directly participates in the scope relation R. Intermediate Reciprocity (IR) and One-Way Weak reciprocity (OWR) weaken the first parameter to Lin resp. Tot. SAR, i.e., Strong Alternative Reciprocity, is the weakening of SR that uses the notion $R^{\vee}$ of directedness over the scope relation. Intermediate Alternative Reciprocity (IAR) and Intermediate Alternative Ordering (IAO) are the corresponding weakenings of IR and OWR resp. with regard to $R^{\vee}$.

Of the forms of reciprocity mentioned so far, SAR, OWR and IAR have been discovered by Dalrymple et al. and have not previously been considered in the literature.

6.2.11 Parameterised Definitions

The parameterisation in the last section gives us an elegant and uniform way to present the different formalisations for reciprocal quantification. The variations of the first parameter is given by the following sequence of definitions.

(6.28) $\textsc{Ful} \equiv$
$\lambda P \lambda Q\ \textsc{Every}(P)(\lambda x\ \textsc{Every}(P/\{x\})(\lambda y\ Q(x,y)))$

(6.29) $\textsc{Lin} \equiv$
$\lambda P \lambda Q\ \textsc{Every}(P)$
$(\lambda x\ \textsc{Every}(P/\{x\})(\lambda y\ \textsc{TransCl}(Q)(P)(x,y)))$

(6.30) $\textsc{Tot} \equiv$
$\lambda P \lambda Q\ \textsc{Every}(P)(\lambda x\ \textsc{Some}(P/\{x\})(\lambda y\ Q(x,y)))$

The first parameter of reciprocals is defined as a diadic quantifier over a first-order set P and a binary first-order relation Q. Every definition shows how each of the three types FUL, LIN, and TOT constrains the relation Q that holds between pairs of elements x and y in P. In the case of FUL, each pair must be directly in the relation Q, while LIN specifies that each pair of different elements x and y to be in the transitive closure or Q with regard to P. Finally, TOT states that every x must be in the relation R with some arbitrary y.

The second parameter of reciprocal meaning determines whether the argument Q in the definition of the first element is actually the relation R provided by the verb phrase of the reciprocal, or the relation $R^{\vee}$, i.e., the relation R extended by its inverse R^{-1}. We define an extension operator $\textsc{Rel}^{\vee}$ on binary relations which allows us to use the extension $R^{\vee}$ as part of higher-order formalisations.

(6.31) $\textsc{Rel}^{\vee} \equiv \lambda Q \lambda y \lambda x\ Q(x,y) \vee Q(y,x)$

Every form of reciprocal semantics that Dalrymple et al. consider as valid can now be given a compact higher-order definition as follows.

(6.32) $\textsc{Rcp}_{SR} \equiv \lambda P \lambda R\ \textsc{Card}^{\geq 2}(P) \wedge \textsc{Ful}(P)(R)$

(6.33) $\textsc{Rcp}_{SAR} \equiv \lambda P \lambda R\ \textsc{Card}^{\geq 2}(P) \wedge \textsc{Ful}(P)(\textsc{Rel}^{\vee}(R))$

(6.34) $\textsc{Rcp}_{IR} \equiv \lambda P \lambda R\ \textsc{Card}^{\geq 2}(P) \wedge \textsc{Lin}(P)(R)$

(6.35) $\textsc{Rcp}_{IAR} \equiv \lambda P \lambda R\ \textsc{Card}^{\geq 2}(P) \wedge \textsc{Lin}(P)(\textsc{Rel}^{\vee}(R))$

(6.36) $\textsc{Rcp}_{OWR} \equiv \lambda P \lambda R\ \textsc{Card}^{\geq 2}(P) \wedge \textsc{Tot}(P)(R)$

(6.37) $\textsc{Rcp}_{IAO} \equiv \lambda P \lambda R\ \textsc{Card}^{\geq 2}(P) \wedge \textsc{Tot}(P)(\textsc{Rel}^{\vee}(R))$

6.2.12 Interpreting Reciprocals

The interpretation of reciprocal expressions requires that we correctly identify the truth conditions that the reciprocal provides. As we have seen in the previous sections, reciprocals contribute different meanings in different contexts. The parameterisation in Section 6.2.9 identifies six candidate meanings, each of which will impose different constraints on the reciprocal's relation R. The actual selection amongst this set can be modelled by applying a certain semantic principle.

6.2.13 The Strongest Meaning Hypothesis

Many expressions in natural language, for instance homonyms, may ambiguously refer to different semantics where the choice of which one is actually meant relies with the speaker. As Dalrymple et al. argue, this is not the case for reciprocals where the literal meaning is determined only by the context in which the reciprocal is uttered. Hence, reciprocal meaning is independent from the speaker. The device for identifying the correct reciprocal semantic is a simple principle, the Strongest Meaning Hypothesis (SMH).

> **Strongest Meaning Hypothesis for Reciprocals**: *The semantic representation of a reciprocal expression S in a context Γ is that representation ϕ whose logical contribution to the reciprocal relation is the strongest consitent one with respect to Γ.*

In the case of reciprocals, the semantic representations that are available in general are those which use any one of the six quantifiers SR, IR, OWR, SAR, IAR, and IAO. The "strongest" contribution to the scope relation is determined by the partial order shown in in Figure 6.1. Although this ordering is only a partial one, there are very few linguistic examples where a reciprocal can for instance be interpreted as both OWR and IAR, but not as IR. Likewise, there is no example where both IR and SAR hold, but not SR, simply because SAR never has been attested so far. For practical purposes, we have a total order $SR > SAR > IR > IAR > OWR > IAO$ on the logical contributions of the reciprocal semantics.

It is easy to verify that most of the natural-language examples given earlier obey the rule that the correct interpretation of the scope relation in a given context is the logically strongest one possible. A weakening of the correct scope relation leads to interpretations that are not acceptible to a hearer, while a scope relation that comes from an overly strong reciprocal semantic even leads to inconsistencies with the contextual knowledge.

6.2.14 A Counter-Example

The linguistic theory sometimes predicts a reciprocal semantic that is too strong. The graphs in Figure 6.2 depict discourse situations for example (6.38) with four snipers. The two graphs show two ways in which they could train their rifles at each other.

(6.38) *The snipers train their rifles at each other.*

The example is problematic for the linguistic theory since the lefthand scope relation is an instance of IR. In the figure, each member of the reciprocal group is connected with each other one by a sequence of other group members. If we would accept IR as the reciprocal semantics, then we would have to discard the right-hand relation as a valid interpretation of the scope relation. Example (6.38) is an instance of OWR that is mistaken as IR reciprocity by the linguistic theory.

6.2.15 The SMH Does Not Compute (Yet)

There is, to our knowledge, no computational method that implements an interpretation of reciprocal expressions and that makes use of the theoretical

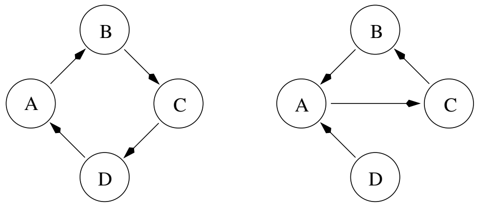

Fig. 6.2. IR or OWR?

insights from the linguistic theories. This is not really surprising, as reciprocals seem to be one of the phenomena of natural language whose computational treatment requires an excessive amount of technical machinery. The most basic problem in this context is that the SMH can only select from a set of reciprocal semantics if it is already known which ones are consistent with the context and which ones are not. Apart from the fact that the problem is undecidable in general, even common reciprocal expressions are very problematic from a computational point of view. For instance, the semantics IR and IAR refer to a computation of transitive closure relations that can be a challenging task once the unverse of discourse becomes larger than a few elements.

Fortunately, the interpretation of reciprocals can be seen from a different point of view where the SMH is understood as a constraint that strengthens the scope relation itself rather than one that chooses the right semantic representation for the reciprocal. As we will see, this alternative use of the SMH leads to an acceptable computational method for interpreting reciprocals by model generation.

6.3 Inference to Best Reciprocal Meaning

The goal of this section is to design a model generation method that can interpret reciprocal expressions. Our approach is based on some insights from the linguistic theory presented in the last section, namely that the reciprocal meaning ranges from a very weak form of reciprocity (IAO) to the strongest one (SR), and that the logical contribution of the reciprocal is determined by the Strongest Meaning Hypothesis. In contrast to the theory of Dalrymple et al., we understand the SMH as a principle of interpretation rather than a principle of semantics construction. The following alternative form of the SMH will be the basis of our computational treatment of reciprocals.

> **Maximise Meaning Hypothesis** (MMH): *The valid interpretations of a reciprocal expression S in a context Γ are those which (a) are consistent both with the weakest form of reciprocal semantics and the context, and (b) whose logical contributions to the scope relation are the strongest.*

The MMH refers only to the weakest semantic representation for reciprocals, namely IAO. The "strongest contribution" of reciprocal interpretations comes from a maximisation of the content of the scope relation over the antecedent set. We argue that this is a precise characterisation of the behavior of reciprocals. Consider again the following examples.

(6.39) *The men like each other. (SR)*

(6.40) *The snipers train their rifles at each other. (OWR)*

The two example sentences are compatible with the weakest form of reciprocity, IAO, which only requires that the antecedent group P that participates in the reciprocal expression has at least two members and that the reciprocal relation R relates each member of the group P with at least one other member. This is, however, not sufficient for a correct interpretation.

In both cases, we have a strengthening of the reciprocal's semantic that is precisely predicted by the MMH. In the SR example, this strenghtening includes all available pairs $\langle x, y\rangle$ to the scope relation where x and y are two different men. The reason for this strenghtening is that the *liking* relation does not have truth conditions that further constrain how we interpret it on a set of men. In contrast to this, the truth conditions for *training a rifle* in the OWR example entail that no sniper x can be the subject of the scope relation for more than one other sniper y. Hence, the strengthening here only makes sure that that each x in P must be the subject of the scope relation R for exactly one other y.

As we can see in these first examples, our understanding of a reciprocal is that of a constituent whose semantic representation is unambiguous, but whose actual range of valid interpretations is determined by some additional constraint on the scope relation and the context. This implies that the truth conditions of the scope relation determine the logical contribution of the reciprocal rather than the other way around.

6.3.1 To Strong Meaning through Minimality

Based on a suitable semantic representation, we will later give the MMH an implementation as a minimal model constraint. This might seem paradoxical at first because the MMH implies a *maximisation* of logical contribution, i.e., assumptions in a model, rather than a minimisation. However, as we will see, both tasks are actually equivalent.

Consider for instance three men a, b, and c in some context Γ. One of the locally minimal models for a semantic representation (6.41) of example (6.39) with respect to Γ is given by (6.42).

(6.41) $\mathit{man}(a) \wedge \mathit{man}(b) \wedge \mathit{man}(c) \wedge \textsc{Rcp}_{IAO}(\mathit{man})(\mathit{like})$

(6.42) $\{\mathit{man}(a), \mathit{man}(b), \mathit{man}(c), \mathit{like}(a, b), \mathit{like}(b, c)\}$

The MMH implies that the weak IAO form of reciprocity is the only semantic representation of reciprocals. The model (6.42) satisfies the logical encoding of reciprocity that we have chosen, but does not represent a meaning that is predicted by the linguistic theory. However, we can turn model (6.42) into a model that corresponds to a valid natural-language interpretation of the reciprocal sentence by adding new consistent assumptions. A completion of (6.42) according to the MMH is given by (6.43).

(6.43) $\{\mathit{man}(a), \mathit{man}(b), \mathit{man}(c), \mathit{like}(a,b), \mathit{like}(b,c), \mathit{like}(a,c), \quad \mathit{like}(b,a), \mathit{like}(c,a)\}$

By adding the new atoms to the set of atoms that is validated by the model, we have actually minimised those atoms that are *not* validated in (6.42). In other words, we have minimised the complement set of the predicate *like*. What we need is a form of minimality that can minimise the denotations of certain predicates or their complements in the set of generated models.

6.3.2 Predicate Minimisation

Predicate-specific minimality, as presented shortly in Section 2.3.5, minimises the occurrences of certain "costly" predicates. In applications of predicate-specific minimality, a model is generally considered to be an explanation for some kind of observation, and the predicates whose occurrences are to be minimised are those that are relevant for distinguishing good explanations from those that are not plausible or may not be attractive for economic reasons.

For instance, an assumption of the form $\mathit{ab}(d)$ in a diagnosis application could be used to encode that the behaviour of the part d of a circuit is faulty. The model of a suitable logical encoding of the circuit and an observed error can then be taken as an explanation how the error is caused. The minimisation of assumptions $\mathit{ab}(d)$ in a model minimises the number of faulty parts that are needed to explain the error in the circuit. By considering only models that are *ab*-minimal, one implements the general principle of diagnosis that the cause of an error lies more probably in the failure of one part of a device than in a simultaneous failure of several parts.

Likewise, predicate-specific minimality can be used to optimise the movement of a robot in a planning application where the movement of a robot from a point x to a point y at a time t corresponds to an assumption $\mathit{go}(x,y,t)$. Each model of the plan task encodes how the robot must move in order to solve a certain task, and by minimising the *go* predicate, one could effectively optimise the (costly) movements of the robot.

In the case of reciprocal expressions, the observation that we want to explain is the truth of the reciprocal sentence, and the assumptions that we want to minimise correspond to those pairs x and y in the reciprocal group that are not in the scope relation. An explanation is best in the sense of the MMH if it explains the truth of a reciprocal sentence while assuming a maximum of pairs $\langle x,y\rangle$ in the scope relation. As indicated earlier, what we minimise here is actually the complement of some binary relation, but by using a suitable logical encoding, we can reduce this problem completely to predicate-specific minimal model generation.

6.3.3 A Logical Encoding of Less Is More

Let ab be an arbitrary but fixed predicate symbol of of type $\iota \to \iota \to o$. For each $\mathcal{MQL}$ model $\mathcal{M}$ of a specification ϕ, the ab-index, in short index, is the number of atoms validated by $\mathcal{M}$ whose head is ab. The index of a model refers to some number of "costly" assumptions that the model implies. The minimisation of the index predicate ab can implement the MMH if we can give a suitable semantic representation that makes explicit the connection between costly assumptions and pairs $\langle x, y \rangle$ that are not in the reciprocal relation,

As usual, we make use of our standard $\mathcal{MQL}$ logic presented in Section 3.3.5. Our higher-order definition of reciprocal semantics has two parts. The first part (6.44) defines the basic constraint on reciprocals that we know from the linguistic theory, namely that each reciprocal must at least meet the truth conditions of IAO reciprocity. We formalise IAO here exactly as we will use it later for model generation.

(6.44) $\textsc{Iao} \equiv$
$\lambda P \lambda R \; \textsc{Card}^{\geq 2}(P) \wedge$
$\textsc{Every}(P)(\lambda x \; \exists y \; \neg(x \doteq y) \wedge P(y) \wedge (R(x,y) \vee R(y,x)))$

(6.45) $\textsc{Card}^{\geq 2} \equiv \lambda P \; \exists x \exists y \; P(x) \wedge P(y) \wedge \neg(x \doteq y)$

The definition is equivalent to the previously given definition of IAO reciprocity in higher-order logic, except that we use Herbrand equality instead of Leibniz equality. In $\mathcal{MQL}$ semantics, we consider two individual constants to be different if they have different names, and Herbrand equality is a computationally very simple form of equality that implements this.

The second part (6.46) of reciprocal semantics formalises the connection between the index of a model and those pairs of group members that are not in the scope relation. Note that we exclude assumptions $\mathsf{ab}(x,x)$; group members that are in the scope relation with themselves do not play any rôle for the index of the model. The higher-order definition (6.47) combines the two parts of the reciprocal's representation.

(6.46) $\textsc{Price} \equiv$
$\lambda P \lambda R \; \textsc{Every}(P)(\lambda x \; \neg \mathsf{ab}(x,x) \wedge$
$\forall y \; (P(y) \wedge \neg(x \doteq y) \wedge \neg R(x,y)) \Leftrightarrow \mathsf{ab}(x,y))$

(6.47) $\textsc{Rcp} \equiv$
$\lambda P \lambda R \; \textsc{Iao}(P)(R) \wedge \textsc{Price}(P)(R)$

6.3.4 A First Attempt at Computation

Definition (6.47) allows us to represent reciprocal sentences according to our theory. Each model of the reciprocal sentence and the context will give us an essential information, the ab-index. The index of a model gives us a means to compare models with respect to a relative satisfaction of the MMH. A model $\mathcal{M}$ that has a smaller index than a model $\mathcal{M}'$ is a better explanation for the truth of the reciprocal sentence according to the MMH. All finite model generation methods can be modified such that their output enumerates models with decreasing index, simply by filtering out all models that have predecessors with a smaller index.

In the case of a domain closure, i.e., whenever we focus only on models that are domain minimal or otherwise restricted in the size of the universe, we can even decide which models have the smallest index. In our model generator Kimba, three methods for the computation of such predicate-specific minimal models are particularly easy to implement.

6.3.5 First Method: Minimality by Proof

The first method is to prove for each generated model that there is no other model that has a smaller index. This method works similar to the computation of locally minimal models as presented in Chapter 4, but requires even less technical machinery. Suppose that ϕ is the logical specification, and $\mathcal{M}$ is a generated model with fixed index n. If n is greater than 0 for $\mathcal{M}$, we simply start Kimba again, but with a constraint on the interpretation of all ab-atoms that their sum must always be less than n. This constraint is actually a finite-domain integer constraint, which we can simply add to the other constraints that Kimba generates while translating the logical input into a system of constraints. If Kimba manages to generate a model $\mathcal{M}'$ with an index $m < n$, then the previously computed $\mathcal{M}$ is not ab-minimal, and we can discard it.

6.3.6 Second Method: Minimality by Bounded Search

The second method is to use branch-and-bound search. In this method, we use the index of each generated model as an upper bound for the index of all models that are generated later. In Kimba, we can compute from each partial interpretation $\mathcal{I}$ a preliminary index that defines a lower bound of the index of those models which are derived from $\mathcal{I}$. If this lower bound becomes greater than the upper bound that we have obtained from the previously computed models, then we know that a further exploration of the search space will not give us better explanations. By discarding those parts of the search space where we cannot expect to find better explanations, Kimba efficiently enumerates models with decreasing index. If the index of a model is minimal, then all models that can be generated from that point on will also be ab-minimal.

6.3.7 Third Method: A Two-Stage Combination

The third method is a a two-stage variation of the second one. In the first stage of the computation, one uses a branch-and-bound search where each computed model carries an index that is truly smaller than the upper bound given by the previously computed models. That is, for each index n, we will compute only one representative model, and all later models must have an index $m < n$. This branch-and-bound constraint restricts the search space more quickly in general than that of the second method, and we will quickly compute a model $\mathcal{M}$ whose index is some minimal index u. Then, in a second stage, we restart the model search again with the second method, but this time we restrict the search from the start to models whose index is u. This restriction will produce only *ab*-minimal models.

While all three methods for computing *ab*-minimal models are equally simple to implement, the second method is more efficient in many cases than the first because the search for the abductive explanations is implemented as a simple constraint that can effectively decrease the search space. The third then is more efficient than the second because it enumerates only the *ab*-minimal models in a much smaller search space that is already constrained by the most minimal index.

6.3.8 An Example

In order to illustrate the effect that the implementation of the MMH has, let us reconsider the example (6.48).

(6.48) *The snipers train their rifles at each other.*

As we know already, this is an example of the OWR form of reciprocity. We now interpret (6.48) in a simple discourse situation where we have a group of three snipers. The formalisation is as follows.

(6.49) $\mathsf{sniper}(peter) \wedge \mathsf{sniper}(paul) \wedge \mathsf{sniper}(mary)$

(6.50) $\text{No}(\mathsf{sniper})(\lambda x\ \text{Card}^{\geq 2}(\lambda y\ \mathsf{train}(x, y)))$

(6.51) $\text{Rcp}(\mathsf{sniper})(\mathsf{train})$

Formula (6.49) formalises that there are three snipers, namely Peter, Paul, and Mary. Formula (6.50) expresses the world knowledge that no sniper trains his or her rifle at more than one other individual. Finally, formula (6.51) represents the reciprocal sentence.

We first apply our model generator Kimba to the formalisation without using any model minimality constraints. A typical model generated by Kimba is the following.

(6.52) $\{\mathsf{ab}(mary, paul), \mathsf{ab}(paul, peter), \mathsf{ab}(peter, mary),$
$\mathsf{ab}(peter, paul), \mathsf{sniper}(mary), \mathsf{sniper}(paul),$
$\mathsf{sniper}(peter), \mathsf{train}(mary, peter),\ \mathsf{train}(paul, mary)\}$

Figure 6.3 shows the search space of the example. Each diamond represents a model, while each circle is a branch point in the search. The search space has 25 such branch points, and leads to 26 different models. These solutions correspond to the different interpretations that one obtains by the IAO form of reciprocity.

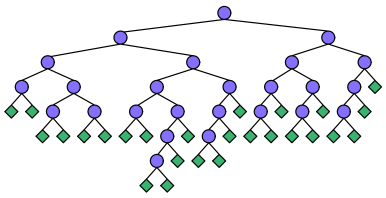

Fig. 6.3. The snipers example without any minimality constraint

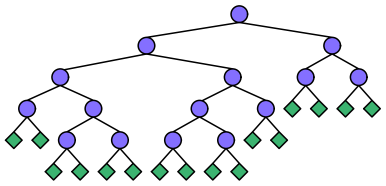

Fig. 6.4. The snipers example with branch-and-bound search

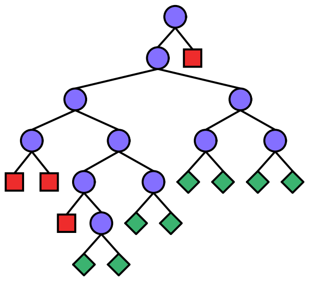

Fig. 6.5. The snipers example with MMH minimality

In Figure 6.4, we have the same example, but this time we have used the branch-and-bound search of KIMBA for eliminating parts of the search space where we know that no *ab*-minimal models can be expected. There are now fifteen branch points in the search space, and sixteen models. Of these, the last eight are *ab*-minimal, and each minimal model contains exactly three *ab*-atoms. The eight minimal models correspond to the eight possible ways in which three snipers can train their weapons at each other.

Finally, in Figure 6.5, we have used the two-stage minimal model computation where those parts of the search space are discarded that lead to models whose index is higher than three. There are now only those models left that are *ab*-minimal. The red squares indicate a failure in the search where a locally computed index becomes too high. Below are the eight different scope relations that are found in the models.

(6.53) $\{\mathsf{train}(mary, peter), \mathsf{train}(paul, peter), \mathsf{train}(peter, mary)\}$

(6.54) $\{\mathsf{train}(mary, peter), \mathsf{train}(paul, peter), \mathsf{train}(peter, paul)\}$

(6.55) $\{\mathsf{train}(mary, peter), \mathsf{train}(paul, mary), \mathsf{train}(peter, mary)\}$

(6.56) $\{\mathsf{train}(mary, peter), \mathsf{train}(paul, mary), \mathsf{train}(peter, paul)\}$

(6.57) $\{\mathsf{train}(mary, paul), \mathsf{train}(paul, peter), \mathsf{train}(peter, mary)\}$

(6.58) $\{\mathsf{train}(mary, paul), \mathsf{train}(paul, peter), \mathsf{train}(peter, paul)\}$

(6.59) $\{\mathsf{train}(mary, paul), \mathsf{train}(paul, mary), \mathsf{train}(peter, mary)\}$

(6.60) $\{\mathsf{train}(mary, paul), \mathsf{train}(paul, mary), \mathsf{train}(peter, paul)\}$

We have mentioned in Section 6.2.9 that the logical contribution of the reciprocal in our example sentence is not identified correctly by the linguistic theory. The scope relation for a larger group that is implied by the classification system is IR, because some of the interpretations meet the truth conditions of IR reciprocity. As we can see above, this does not pose a problem for our approach, as it distinguishes interpretations only by the number of assumptions they provide to the scope relation. Hence, the relation (6.57), an instance of IR reciprocity, occurs as well as a relation like (6.60), that is an example of OWR, in the set of our valid interpretations.

6.3.9 Conservative Minimality

The minimisation of a certain predicate *ab* so far seems to be an adequate form of minimality that identifies those models whose content each describes a valid logical contribution of the reciprocal. However, as discussed in some detail in Chapter 5, we have also argued for the use of *local minimality* for obtaining a correspondence of logical models and some valid natural-language interpretations. This form of minimality must be considered as well in the interpretation of reciprocal expressions, for instance in the following sentence.

(6.61) *Peter, Paul, and Mary like each other.*

The *liking* relation implies SR reciprocity, and in this form of reciprocity, all members of the reciprocal group are in the reciprocal relation. SR reciprocals therefore have only one possible scope relation once the reciprocal group is determined. KIMBA's method for computing the *ab*-minimal models yields model (6.63) as one of the interpretations of the logical encoding (6.62), as required. Unfortunately, we also have the models (6.64)–(6.69).

(6.62) $\textsc{Rcp}(\{peter, paul, mary\})(\mathsf{like})$

(6.63) $\{\mathsf{like}(mary, paul), \mathsf{like}(mary, peter), \mathsf{like}(paul, mary),$
$\mathsf{like}(paul, peter), \mathsf{like}(peter, mary), \mathsf{like}(peter, paul)\}$

(6.64) $\{\mathsf{like}(mary, paul), \mathsf{like}(mary, peter), \mathsf{like}(paul, mary),$
$\mathsf{like}(paul, peter), \mathsf{like}(peter, mary), \mathsf{like}(peter, paul),$
$\mathsf{like}(mary, mary)\}$

(6.65) $\{\mathit{like}(mary, paul), \mathit{like}(mary, peter), \mathit{like}(paul, mary),$
$\mathit{like}(paul, peter), \mathit{like}(peter, mary), \mathit{like}(peter, paul), \mathit{like}(paul, paul)\}$

(6.66) $\{\mathit{like}(mary, paul), \mathit{like}(mary, peter), \mathit{like}(paul, mary),$
$\mathit{like}(paul, peter), \mathit{like}(peter, mary), \mathit{like}(peter, paul), \mathit{like}(peter, peter)\}$

(6.67) $\{\mathit{like}(mary, paul), \mathit{like}(mary, peter), \mathit{like}(paul, mary),$
$\mathit{like}(paul, peter), \mathit{like}(peter, mary), \mathit{like}(peter, paul),$
$\mathit{like}(mary, mary), \mathit{like}(paul, paul)\}$

(6.68) $\{\mathit{like}(mary, paul), \mathit{like}(mary, peter), \mathit{like}(paul, mary),$
$\mathit{like}(paul, peter), \mathit{like}(peter, mary), \mathit{like}(peter, paul),$
$\mathit{like}(paul, paul), \mathit{like}(mary, mary)\}$

(6.69) $\{\mathit{like}(mary, paul), \mathit{like}(mary, peter), \mathit{like}(paul, mary),$
$\mathit{like}(paul, peter), \mathit{like}(peter, mary), \mathit{like}(peter, paul),$
$\mathit{like}(peter, peter), \mathit{like}(paul, paul)\}$

(6.70) $\{\mathit{like}(mary, paul), \mathit{like}(mary, peter), \mathit{like}(paul, mary),$
$\mathit{like}(paul, peter), \mathit{like}(peter, mary), \mathit{like}(peter, paul),$
$\mathit{like}(peter, peter), \mathit{like}(paul, paul), \mathit{like}(mary, mary)\}$

The last seven models are *ab*-minimal, but they contain information that is not implied by the natural-language sentence. This is because the model generator is free to assume a fact like $\mathit{like}(mary, mary)$ unless it is not inconsistent. The unwanted assumptions are easy to identify because they do not occur in any locally minimal model of the input.

On the other hand, the model (6.63) that we want to obtain as an interpretation of the natural-language input is itself not a locally minimal model of our semantic representation. It contains assumptions that are unnecessary for validating the truth of the input with respect to local minimality. Our theory of reciprocal meaning explicitely requires the presence of such non-minimal assumptions.

In order to identify models for reciprocal sentences that correspond to natural-language interpretations, we arguably must identify those *ab*-minimal models that are locally minimal *relative* to all other *ab*-minimal models. In our example, the first model is locally minimal relative to all other predicate-minimal models, and hence is the only one that should be accepted. What we have in mind is a new form of minimality that we define formally as follows.

Conservative Minimality: *Let Φ be a set of satisfiable $\mathcal{MQL}$ formulas. Then there exists a nonempty set of models $\boldsymbol{D}$ whose domain $\mathcal{D}_\iota$ of first-order individuals has a minimal size. Let $\boldsymbol{C}$ be the set of those models in $\boldsymbol{D}$ that are* ab*-minimal. A model $\mathcal{M} \in \boldsymbol{C}$ is conservative minimal iff* $\textsc{Pos}(\mathcal{M}') \subseteq \textsc{Pos}(\mathcal{M})$ *implies $\mathcal{M}' = \mathcal{M}$ for all models $\mathcal{M}' \in \boldsymbol{C}$.*

Conservative minimality is the desired combination of local minimality and *ab*-minimisation. We have chosen the term "conservative" because this form of minimality is a conservative extension of local minimality. If **D** does not contain models with *ab*-atoms, then all conservative models are also locally minimal models. The new form of minimality is compatible for instance with our earlier approaches for computing the interpretations of singular definite descriptions.

But how exactly can we compute such models? This can be done by a variation of the standard search for locally minimal models. The models that we want to obtain are domain minimal models that (a) are predicate-specific minimal and (b) that are subset-minimal with respect to all other predicate-specific minimal models. The constraint (a) is met by identifying the predicate-specific models $\mathcal{M}$ of the input that have some smallest possible domain $\mathcal{D}_\iota$. For (b), we use the same technique as for locally minimal models and prove for each model $\mathcal{M}$ that there is no other model $\mathcal{M}'$ that satisfies the input with a true subset of the assumptions in $\mathcal{M}$ **and** that is also predicate-specific minimal. Hence, a predicate-specific minimal model is compared only to other predicate-specific minimal models. A modification of KIMBA's minimal model search is straightforward and has been caried out for the following examples.

6.4 Experiments

In what follows, we will investigate how our approach computes the logical contributions of the reciprocal in the context for some selected examples from the literature. As we will see, a correct analysis of reciprocals sometimes requires a surprisingly complex specification of the properties of the scope relation.

6.4.1 Pitchers and Pearls

IR hold for scope relations R where each group member x is related to each other group member y by a sequence of members $c_1, \ldots, c_n$ such that $x = c_1$, $y = c_n$, and $R(c_i, c_{i+1})$ for all $i < n$. Some linguistic examples for IR are discourses such as (6.1) or (6.2) where the scope relation is equivalent to a linear or circular ordering of individuals.

(6.71) *The Boston pitchers sit alongside each other.*

(6.72) *The pearls in the necklace are separated from each other by semi-precious jewels.*

The truth conditions that IR formulates on its own are quite different from those that can hold in discourse (6.71) and (6.72). In Figure 6.6, we have four graphical representations of IR scope relations, each with four discourse participants. Only the first, leftmost one would be acceptable as a relation for (6.71) in a discourse situation with four Boston pitchers. The situation implied by sentence (6.71) requires a relation that is symmetric. The relations 2 and 3 do not meet this requirement. Relation 4 depicts a situation where one group member is related to three other ones, which conflicts with the world knowledge that a person cannot sit alongside more than two persons.

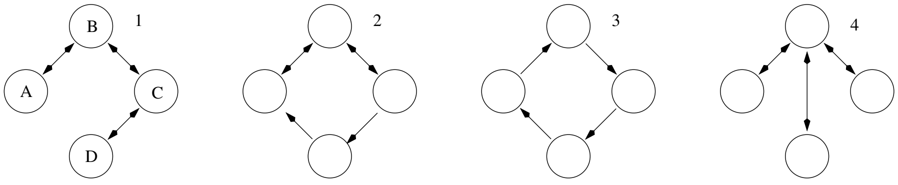

Fig. 6.6. IR relations

6.4.2 The Boston Pitchers

A correct interpretation of example (6.71) requires a suitable analysis of the *sitting alongside* relation. The properties of this relation can be captured by relating it to a *position*, i.e., a total ordering of individuals. In our case, this position could denote the position of the pitcher on the bench on which they sit. A formalisation of a *sitting alongside* relation *sital* must then express the following properties.

- For each two individuals x and y, if *sital*(x, y) holds, then x is either in a predecessor or a successor position of y.
- At each possible position we have at most one individual x.
- An individual cannot be at two different positions.
- For each position i, if some x is in a position i for a given relation, then there are exactly i individuals whose position p is smaller or equal to i.

The complete specification must also define concepts such as a successor relation and some total order on positions, but otherwise is straightforward. Section A.2 gives a logical encoding that can be taken as the input for our model generator Kimba, and evaluates the results for a discourse situation with four Boston pitchers.

6.4.3 Pearls

In example (6.73), the logical specification of the properties of the scope relation is much easier than in the "Pitcher" example.

(6.73) *The pearls in the necklace are separated from each other by semi-precious jewels.*

If we assume that we have a simple, round necklace, then it suffices to specify that no pearl can be separated by semi-precious levels from more than two other pearls, and that the relation itself is symmetric. There also are far fewer models than in the case of the pitchers—we have three models in a "necklace" with four pearls. All of them are instances of the relation depicted in Figure 6.7.

What is interesting here is that the interpretation of the relation provided by the verb phrase is heavily influenced by properties implied by the reciprocal group. The "pearls in the necklace" arguably imply a circular relation that is not part of the truth conditions of the scope relation on its own.

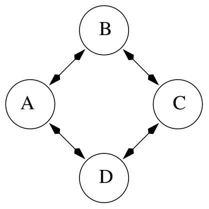

Fig. 6.7. The IR relation for pearls in a necklace

6.4.4 Measles

The linguistic theory describes IAR scope relations as relations where each group member is related directly or indirectly via the reciprocal relation without considering the direction of the relation.

(6.74) *The students gave each other measles.*

The formalisation of IAR in Higher-Order Logics relies on a specification that uses the transitive closure relation of the scope relation. In our approach, it suffices to specify the characteristic properties of the scope relation for a discourse situation. For this, we again use a position relation that describes the distance of some person that has measles to the one who was a source of the measle infection in the discourse situation. The properties of the *giving measles* relation are then as follows.

- No one can be given measles by more than one other person.
- If someone has measles, then there must be at least one person that is the source of the measles, i.e., whose position is 1.
- If some person x gives measles to a second person y, then x's position is the successor of the position of y.
- If some person x gives measles to a second person y, then both persons have measles.

Our specification of what *giving measles* means is quite different from what IAR formulates as the truth conditions of the scope relation. Indeed, it is easy to show that IAR insufficiently describes the reciprocal relation in (6.74). Figure 6.8 shows a selection of IAR relations over a group of four elements that all would not be acceptable as instances of the *giving measles* relation while those relations in Figure 6.9 are.

The model generator KIMBA generates 64 different conservative minimal models for the "Measle" example with four discourse participants. These models all are instances of the relations depicted in Figure 6.9. While relations 1 and 2 each have 24 instances, i.e., permutations, the relation 3 has twelve, and relation 4 only four instances.

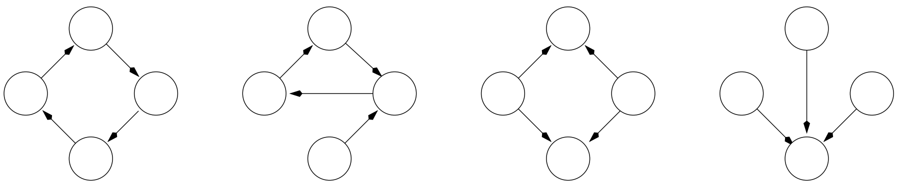

Fig. 6.8. IAR relations that are not compatible with measles

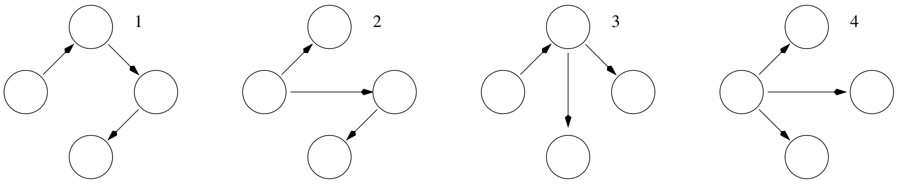

Fig. 6.9. IAR relations that are compatible with measles

6.4.5 Marriages

Example (6.74) exemplifies the OWR form of reciprocity.

(6.75) *The people in the room all are married to each other.*

A formalisation of the relation expressed by the verb phrase includes the following facts.

- If some person x is married to some person y, then y is also married to x.
- No one can be married to more than one other person.

In a discourse situation with four discourse participants, we have three models that all are an instance of the relation depicted in Figure 6.10.

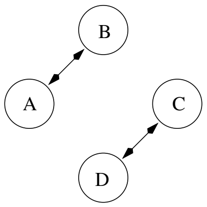

Fig. 6.10. The OWR relation for being married with each other

Interestingly, if we add a fifth person to the discourse, then the model generation problem becomes unsatisfiable for this universe of discourse. The model generator solves this problem by accommodating a new individual who becomes the spouse of the "odd" person.

6.5 Loose Ends

There are some cases where the strong meaning predicted by the SMH as well as the MMH seems to be too strong. Consider the following example from Philip [87] that has also been discussed in detail in Winter [88].

(6.76) *The three boys tickle each other.*

Although it can happen that a boy tickles two objects at the same time, the natural interpretation of (6.76) is one where each boy tickles only one other boy. As Winter notes, this potential counter-example of meaning maximisation might come from some gap in our world-knowledge about the predicate *to tickle*. There seems to be a default assumption that one tickles only one other object in a situation. However, this "uniqueness" assumption can be overridden, i.e., is not a hard constraint on the interpretation of *to tickle*.

Winter also mentions a counter-example for the SMH where the truth conditions are actually too weak. This example is as follows.

(6.77) *Mary and Sue gave birth to each other. (*)*

In (6.77), both SMH and MMH predicts semantics/interpretations where either Mary gave birth to Sue or Sue gave birth to Mary. This is not the intended meaning of the clearly inconsistent statement. There seems to be a lower bound for the weakening of logical contribution at least in the case of the "giving birth" relation. It can be argued that sentence (6.77) implies a discourse anomaly as one of Grice's maxims on conversation states that a contribution should not give redundant information. The weakening of the reciprocal's contribution leads to such a redundancy, as the contribution of the reciprocal becomes nil.

6.6 How We Can Understand Each Other

The approach of Dalrymple et al. [85] to reciprocal meaning exemplifies a formal, theoretic solution that tries to classify ambiguity in meaning by some collection of semantic representations and a non-computational method that models how the correct reading is selected. In theory, the method is not computational because the criteria that must be verified for making a selection are undecidable.

In contrast to this, the approach by model generation strengthens the logical contribution of the weak semantic representation as a process based on *computational* constraints. Once any model can be computed at all for our logical form, the process of interpretation is an inference problem that is guaranteed to terminate. The algorithm may fail for complexity reasons, but in principle it should be able to give at least one linguistically acceptable interpretation if the reciprocal's logical form is satisfiable with respect to its context and a restriction to finite domains.

Empirically, the predictions are the same in most cases because maximising the scope relation often results in yielding a logically stronger meaning on the entailment scale. In particular, the present approach captures the meanings postulated by Dalrymple et al. But the examples also show that there are cases where the predictions differ. Intuition dictates that the sentence *the snipers train their rifles at each other* has a natural OWR interpretation, but the classification scheme incorrectly gives us IR as the strongest meaning. Model generation, however, not only gives us the correct interpretations, it is also linguistically more appealing as it needs only one semantic representation. The model generator is the only computational tool that we need for determining preferable readings from this single representation.

One crucial feature of minimal model generation is that it permits comparing and ranking of natural-language interpretations against each other. In the case of reciprocals, the ranking is given mainly by the size of the scope relation. We have already found other useful ranking criteria as well. For instance, Chapter 5 shows that in the case of definite descriptions, the ranking defined by local minimality permits capturing the preference of binding over bridging over accommodation. Similarly the work of Baumgartner and Kühn [67], which we will discuss in Section 7.2, shows that a predicate minimisation together with a preference for logically consequent resolutions can be used to model the preferences in the interpretation of pronominal anaphora.

This suggests that one of the most promising application of model generators is as a device for developing and testing *preference systems* for the interpretation of natural language. Inference and knowledge based reasoning are needed in natural-language processing not only to check for consistency and informativity as illustrated for instance in Blackburn et al. [89], but also to express preferences between possible interpretations. For this, model generation seems to be a natural tool to use.

7 Abduction

Overview: Abduction is inference to the best explanation. It is widely recognised as a form of inference that can conceptualise the problem of interpreting discourses. Model generation as we have used it is a form of abduction. We discuss the relation between our approach to abduction to others that have been applied to linguistics.

7.1 What Is Abduction?

The term "**abduction**" was first used by Pierce [90] who defined it as follows.

> *The surprising fact, C, is observed; but if A were true, C would be a matter of course. Hence, there is reason to suspect that A is true.*

In other words, abduction originally refers to an unsound inference rule that concludes A from the facts C and $A \Rightarrow C$. Abduction in this sense is inference to some *arbitrary* explanation A for C. Without some further restrictions, abductive inference would have few applications, as there usually are too many explanations that can be obtained from some rules which have C as their consequence. The actual method of inference is in general only secondary to what could be called the **overgeneration problem of abduction**. The usefulness of an explanation depends on additional application-dependent constraints of suitable explanations. The term abduction, as it is commonly used today, refers to all forms of inference that yield "best", i.e. suitable, explanations for observed facts in some way.

Since Hobbs et al. presented their inspiring work, abduction is widely recognised as a form of inference that can conceptualise the problem of interpreting discourses. In this context, the best explanations sought for are sets of assumptions that explain the truth of a speaker's utterances in a way that is compatible with contextual information and the common ground of knowledge shared by speaker and hearer.

Model generation becomes a form of abduction whenever some models with desirable properties are identified as best solutions for a problem. Thus, in previous chapters, we have used model generation as a form of abduction where the best explanations are minimal models. For the interpretation of singular definite descriptions, we have argued for local minimality as that desirable property that characterises best explanations. In the case of reciprocal expressions, we have proposed conservative minimality which combines the minimal model constraints of logical minimality, domain minimality, and predicate-specific minimality.

The principal goal of this chapter is to discuss how minimal model generation fits exactly into the larger picture of abduction in natural-language processing, and how our approach compares to other work that uses inference in natural-language semantics.

7.1.1 A Formal Definition of Abduction

Let ϕ be a logical formula called **observation**, Γ be a set of formulas called **context** or **background theory**, and Ψ be a set of formulas called **allowable hypothesis**. An **abductive explanation** is a set Δ of assumptions such that the following holds [67].

- $\Delta \subseteq \Psi$
- $\Gamma \cup \Delta \models \phi$
- $\Gamma \cup \Delta$ is satisfiable

The term **abduction** refers to all forms of inference that generate best abductive explanations from specifications Γ, ϕ, and Ψ. The definition of an abductive explanation that has been given above is a general one that fits different forms of abduction without exactly capturing the characteristic properties that distinguish *best* abductive explanations from those explanations that are not considered suitable for the intended application.

For instance, in diagnosis applications, the allowable hypothesis Ψ is a set of assumptions that describes the cause a failure in some complex device described by Γ, and ϕ is a specification of an observable faulty behavior. An abductive explanation Δ is then a set that exlains why Γ shows that behavior. A characteristic property of abductive explanations in diagnosis applications is a minimisation of Δ such that the failure of the device is explained by a minimal number of faulty device parts.

In our approach of semantic interpretation as minimal model generation, the formula ϕ consists of some semantic representation for a sentence, and Γ is the logical specification of a discourse model and the necessary world knowledge. The set of allowable hypothesis Ψ is the set of all atoms that can be built from a suitable signature. The set Δ of assumptions is a set of atoms that defines a model of $\Gamma \cup \{\phi\}$. We have characterised best explanations as minimal models whose forms of minimality have been found suitable as constraints of valid natural-language interpretations.

There is some other work that has investigated how different forms of abduction can be used for conceptualising the interpretation of discourses. Baumgartner and Kühn [67] presents an abductive solution to anaphora resolution in a model generation framework. Their approach uses predicate-specific minimality. In Hobbs et al. [91], we find **weighted abduction**, a variation of logic programming. A best explanation is an explanation where the set Δ of assumptions is minimal with respect to a numerical cost obtained from Δ itself and from the computation process that leads to Δ. Weighted abduction has been used for the interpretation of anaphora, compound nominals, syntactic ambiguities and metonomies.

In what follows, we will compare the results of this work with what can be achieved by the form of minimal model generation that we have developed.

7.2 Models for Anaphora Resolution

Baumgartner and Kühn [67] proposes model generation as a method for computing the resolution of anaphoric expressions. More specifically, they address the problem of establishing the links between pronominal and definite anaphora to their referents.

The model generation calculus that is used to attack the problem is based on hyper-tableaux as presented in Section 2.4.4. While a considerable part of the authors' efforts go into the design of a hyper-tableaux method that incrementally computes best abductive explanations, the logical encoding and the abductive inference method is more general and can be used as well for other model generation approaches. In the following, we exemplify abductive anaphoric resolution as proposed by Baumgartner and Kühn.

7.2.1 Chasing the Criminal

Example (7.1) outlines the general problem of abductive anaphoric resolution. We have a definite description *the criminal* that has two possible referents, namely a politician and a gangster. Both referents are introduced by the antecedent sentence. It is not inconsistent per se to assume that a politician is a criminal, but we know that each gangster is a criminal. Hence, the reading where *the criminal* refers to the gangster in the context is preferred. The resolution of the anaphoric definite obviously is an abductive inference task: we have two linguistically consistent[1] interpretations, of which only one is the best, i.e., preferred one.

(7.1) *A politician chased a gangster.* ***The criminal*** *died.*

[1] Note that is a consequence of the weak logical encoding. The representation (7.4) does not formalise unicity which we have identified as one of the essential truth conditions of many definites in Chapter 5.

For example (7.1), we use a logical encoding that formalises the information given in the first sentence by the first-order formula (7.2), the world knowledge by formula (7.3), and the sentence with the definite by formula (7.4).

(7.2) $\mathsf{pol}(p) \wedge \mathsf{gan}(g) \wedge \mathsf{chase}(p, g)$

(7.3) $\forall x\ \mathsf{gan}(x) \Rightarrow \mathsf{crim}(x)$

(7.4) $\exists x\ \mathsf{crim}(x) \wedge \mathsf{anaph}_1(x) \wedge \mathsf{die}(x)$

The predicate anaph_1 is used to indicate in the model which discourse participant has been used for resolving the anaphor expressed by the definite description.

$$\begin{array}{c}
\mathsf{pol}(p) \\
\mathsf{gan}(p) \\
\mathsf{chase}(p, g) \\
\mathsf{crim}(g) \\
\begin{array}{c|c}
\mathsf{crim}(p) & \\
\mathsf{die}(p) & \mathsf{die}(g) \\
\mathsf{anaph}_1(p) & \mathsf{anaph}_1(g)
\end{array}
\end{array}$$

Fig. 7.1. Model construction for discourse (7.1)

The first-order Herbrand models of the formulas (7.2)–(7.4) are given by the tableaux in Figure 7.1. There are two branches, each containing the atoms that define an interpretation. While both branches are consistent and therefore are models of the input, only the second one corresponds to the preferred reading.

Baumgartner and Kühn investigate how resolutions can be eliminated that do not correspond to preferred interpretations. The anaphor resolution performed in the first branch leads to the assumption that the politician is a criminal. This is not a logically consequence of the information given by the discourse. In the second model, we have that the gangster is a criminal, a fact which logically follows from the world knowledge. This difference can be used to characterise the preference for the second model as a reading of the discourse.

7.2.2 Explaining Resolutions

Baumgartner and Kühn argue that the semantic content of the anaphoric expression should be *implied* by the context, and not only be consistent. For our example, the semantic content of the definite *the criminal* is given by formula (7.5).

(7.5) $\exists x\ \mathsf{crim}(x) \wedge \mathsf{anaph}_1(x)$

If we take (7.5) as an observation that must be explained, then we can instantiate the abduction scheme for our example as follows.

(7.6) $\Gamma = \{\mathit{pol}(p), \mathit{gan}(p), \mathit{chase}(p,g), \forall x\ \mathit{gan}(x) \Rightarrow \mathit{crim}(x)\}$

(7.7) $\phi = \exists x\ \mathit{anaph}_1(x) \wedge \mathit{crim}(x)$

(7.8) $\Psi = \{\mathit{anaph}_1(p), \mathit{anaph}_1(g)\}$

There are three candidate explanations.

(7.9) $\Delta_1 = \{\}$

(7.10) $\Delta_2 = \{\mathit{anaph}_1(p)\}$

(7.11) $\Delta_3 = \{\mathit{anaph}_1(g)\}$

(7.12) $\Delta_4 = \{\mathit{anaph}_1(p), \mathit{anaph}_1(g)\}$

The abductive explanation (7.9) can be eliminated because it does not satisfy the condition that $\Gamma \cup \Delta_1 \models \phi$. Explanation (7.12) is eliminated by linguistic knowledge. It does not give a unique resolution of the anaphoric definite. Explanation (7.10) is not acceptable as well because it does not satisfy $\Gamma \cup \Delta_2 \models \phi$. The observation ϕ becomes a logical consequence of $\Gamma \cup \Delta_2$ only if we add the additional assumption $\mathit{crim}(p)$ which, however, is not part of the background theory Γ. Only (7.11) remains as a valid abductive explanation for the truth of observation ϕ.

Baumgartner and Kühn give an extension of the standard hyper-tableaux calculus that maintains a minimal model property such that each open branch is a anaph_i-minimal model of the input. Additionally, it can be verified at any time whether the semantic information of an anaphoric expression is implied by the background information. Thus, it can be tested whether a model provided by the tableaux meets the conditions of a best abductive explanation. This ensure that each model computed corresponds to a valid natural-language interpretation of the input. As their analysis of anaphors is based on closed, finite domains, Baumgartner and Kühn intuitively define an interesting new form of minimality for propositional logics that combines predicate minimisation and logical conditions on the consequence relation of models and input theories.

7.2.3 Discussion

The ambiguity that natural language shows on various levels leads to a combinatorial explosion when computing the candidate interpretions of discourses. In this context, a tableaux approach for model generation in general has the advantage that it can be space efficient. A tableaux expansion may restrict the search space to only one model candidate at a time. If an interpretation given by a tableaux branch is discarded by ading new information, then backtracking can be used to compute a new consistent reading in a different branch. Additionally, a tableaux can be built such that already computed knowledge does not have to be recomputed completely when new data is added. Thus, model generation by tableaux, as proposed by Baumgartner and Kühn, offers both a solution to the combinatorial explosion of alternative readings and *incrementality*, the ability to deal with new discourse information.

7.2.4 Incremental Inference instead of Generate-and-Test

Baumgartner and Kühn identify those resolutions as preferable that explain the semantic content of the anaphoric expression by logical consequence from the context. Their approach works well for instance for singular pronouns as in example (7.13).

(7.13) *Peter loves Mary. He*$_1$ *sends her*$_2$ *letters every day.*

Pronouns provide almost no semantical information of their own. The observations that must be explained in (7.13) are the simple formulas below.

(7.14) $\exists x\ \mathsf{male}(x) \wedge \mathsf{anaph}_1(x)$

(7.15) $\exists x\ \mathsf{female}(x) \wedge \mathsf{anaph}_2(x)$

The background information includes the information that Peter is male and Mary is female. It is then no problem to identify the resolution of the pronouns with the best and only explanation $\Delta = \{\mathsf{anaph}_1(peter), \mathsf{anaph}_2(mary)\}$. The method of how explanations are computed preserves one of the main advantage of the tableaux framework, namely that the computation of a model remains local with respect to the currently expanded branch. This is especially useful in cases where we have ambiguities.

(7.16) *Peter knows Mary's cousin. He*$_1$ *is married to her*$_2$ *sister.*

Here, we have one best explanation where the first pronoun refers to Peter and the second pronoun refers to Mary. This leads to the preferred reading where Peter is married to Mary's sister. Two alternative readings remain accessible in the tableau, one where Peter is married to the sister of Mary's cousin, and one where Mary's cousin is married to Mary's sister. These two readings are not investigated further since the semantic content of the pronouns is not implied by the background knowledge. Hence, a further tableaux expansion can provisionally concentrate on one branch that contains the preferred reading. However, a sentence like (7.17) may later close the current branch by adding inconsistent background knowledge. The method then must backtrack and chose a new interpretation of the discourse from the available open branches. This new branch is then expanded up to the point where it is either closed or saturated.

(7.17) *Peter is now forty, and he still is a single.*

While the example appears to be trivial, it should be made clear that even many contemporary approaches to anaphora resolution use inference only for *verifying* that a provisional coreference of a pronoun to its referent is consistent with the context. The approach by Baumgartner and Kühn actually *computes* the resolutions of anaphoric expressions, and it does so in an incremental way that is more efficient than a generate-and-test approach which must in the end consider all possible mappings of pronouns to referents. The approach makes use of the ability of tableaux to deal with incremental information, which could be a serious advantage in comparison to those methods that must recompute all inferences when a discourse is extended.

7.2.5 An Alternative by Conservative Minimality

The linguistic theory behind abductive anaphora resolution considers preferable readings as abductive explanations for the semantic content of the anaphoric expression. This theory can be implemented by other forms of model generation as well, for instance by conservative minimality. The intuitive idea is to *minimise* the logical contribution that each anaphora provides to the model of the discourse.

In the case of discourse (7.17), we can use the formulas (7.19)–(7.21) as a logical specification that approximates its meaning. Further, we add the formula (7.22) that adds a *ab*-assumption for each discourse participant that meets the semantic content of the definite description *the criminal.*

(7.18) *A politician chased a gangster.* ***The criminal*** *died.*

(7.19) $\mathsf{pol}(p) \land \mathsf{gan}(g) \land \mathsf{chase}(p, g)$

(7.20) $\forall x\ \mathsf{gan}(x) \Rightarrow \mathsf{crim}(x)$

(7.21) $\exists x\ \mathsf{crim}(x) \land \mathsf{die}(x)$

(7.22) $\forall x\ \mathsf{crim}(x) \Rightarrow \mathsf{ab}(x)$

There are two locally minimal models for the formulas (7.19)–(7.22).

(7.23) $\mathcal{M}_1 = \{\mathsf{pol}(p), \mathsf{gan}(g), \mathsf{chase}(p, g), \mathsf{crim}(g), \mathsf{die}(g),\ \mathsf{ab}(g)\}$

(7.24) $\mathcal{M}_2 =$
$\{\mathsf{pol}(p), \mathsf{gan}(g), \mathsf{chase}(p, g), \mathsf{crim}(g), \mathsf{crim}(p), \mathsf{die}(p), \mathsf{ab}(g), \mathsf{ab}(p)\}$

Only the first model $\mathcal{M}_1$ is a conservatively minimal model that has a minimum *ab*-cost. Our encoding implies a penalty for each individual that has the property of being a criminal. Hence, a model where the definite is resolved with a discourse participant that already meets the properties described by the definite itself has necessarily a lower *ab*-cost than one where the properties for the referent must be accommodated. Intuitively, the definite description is resolved with that individual that "fits better" in the given discourse situation. This concept of a preferred resolution is analogous to that in the tableaux approach, only that it is implemented by a minimisation of *ab*-costs rather than additional inference steps in the model computation. The presence of anaph_i-predicates in the encoding is not necessary any more because the minimisation of anaphoric links is already implemented by the *ab*-minimisation.

We make use of these insights and propose a new universal determiner THE as an alternative to that known from Chapter 5. Instead of unicity, i.e., a strict constraint on the truth conditions of a definite, we now imply abductive resolution by *ab*-predicate minimisation. The definition in our $\mathcal{MQL}$ logic is as follows.

(7.25) $\textsc{The} \equiv \lambda P \lambda Q\ (\exists x P(x) \land Q(x)) \land \forall x\ P(x) \Rightarrow \mathsf{ab}(x)$

The sentence *The criminal died* then has the representation $\textsc{The}(\mathsf{crim})(\mathsf{die})$, and conservative minimality ensures that we identify the correct preferred reading in the set of models.

Baumgartner and Kühn consider pronominal anaphora to be a special case of definite anaphora. The determiner THE can in this sense be used to represent pronouns as well. The first argument of THE contains the semantic contribution of the anaphor. This semantic contribution is the gender gender information in the case of pronouns. Thus, we can have the logical representations (7.27)–(7.29) for the discourse (7.26). Formula (7.29) is the semantic representation of *He sends her letters every day.*.

(7.26) *Peter loves Mary. He*$_1$ *sends her*$_2$ *letters every day.*

(7.27) $\mathsf{love}(peter, mary)$

(7.28) $\textsc{No}(\mathsf{male})(\mathsf{female}) \land \mathsf{male}(peter) \land \mathsf{female}(mary)$

(7.29) $\textsc{The}(\mathsf{male})(\lambda x\ \textsc{The}(\mathsf{female})(\lambda y\ \mathsf{sendLet}(x, y)))$

The conservatively minimal model of the input contains the assumption $\mathsf{sendLet}(peter, mary)$, which corresponds to the correct resolution of the pronouns as required.

The alternative approach of abductive resolution by conservative minimality does not correspond exactly to the hyper-tableaux approach. Baumgartner and Kühn concentrate on anaphora. They do not consider definite descriptions where an accommodation of new discourse participants is necessary. We have discussed examples such as (7.30) in detail in Chapter 5. The tableaux approach is based on a domain closure assumption, i.e., all inferences are restricted to a given universe of discourse that is defined by the first-order constants in the current branch. While new linguistic data may add new discourse participants, the abductive inferences that establish the links between anaphora (or definites) and their referents will not. In contrast to this, the minimal model approach allows accommodation as needed for instance in (7.30), but on the other hand will not indicate a discourse anomaly when a pronoun does not have a referent such as in example (7.31).

(7.30) ***Jon's rabbit*** *is cute.*

(7.31) ***She*** *hates my house. (*)*

It is no technical problem to add a domain closure assumption to a model generator, either by a logical encoding or a simple modification in the search for models. However, if such a domain closure is used, an example like (7.29) cannot be treated correctly any more.

7.3 Weighted Abduction

Hobbs et al. [91] considers the interpretation of natural-language texts as a process that yields one preferable explanation of why a text is true. This process is modelled by **weighted abduction**, a form of abductive inference that is based on abductive logic programming.

7.3.1 Logic Programming and Abduction

Logic Programming refers to forms of computation where the programs and the input that define a particular computation are given as logical statements, and the process of executing a computation is made explicit by inference. The relation between logic and algorithms is summed up by Robert Kowalski's equation

$$\mathit{Algorithm} = \mathit{Logic}{+}\mathit{Control}$$

Logic programming systems can be characterised as theorem provers where the way in which proofs are computed is controlled by some fixed control strategy that can be used by a programmer for defining algorithms. The logical language used usually is less expressive than first-order predicate logic, but may be extended by non-logical commands that control the inference process and the input-output of a computation. The best-known example of a logic programming language is Prolog whose logical part is restricted to first-order Horn clauses.

A **query** to a logic programming system is some formula ϕ that is refuted with respect to a **database** $\varGamma$ which consists of logically encoded knowledge. An **answer** to a query is some instantiation σ of free variables in ϕ such that $\sigma(\neg\phi)$ is inconsistent with respect to $\varGamma$.

Abductive logic programming systems are logic programming systems where facts that are needed for a proof be assumed as hypothesis instead of being proved. Abductive logic programming systems answer queries ϕ not only by returning an instantiation, but also by some set of additional hypothesis $\varDelta$. The relation between the query ϕ, the background knowledge $\varGamma$ and the assumptions $\varDelta$ meet the formal definition of abductive explanations given in Section 7.1.

Weighted Abduction computes a set of assumptions that explains the truth of a semantic representation as a "best" answer of a query to an abductive logic programming system. In weighted abduction, the literals in the query and in the clauses in the database carry numerical values called **weights** which determine a system of preference for which facts can be taken as allowable hypothesis. Weighted abduction is a method of abductive inference that is closely related to Prolog-style logic programming. That is, the query and the database are restricted to first-order Horn logic and answers are computed by the usual backward-chaining of a Prolog system. The Horn clauses in the database themselves carry numerical values which determine **inference costs** for deriving information when using them.

7.3.2 Abductive Explanations

Before we investigate the intuition behind weighted abduction, we first give a simple example for a set of explanations that can be computed by abductive logic programming.

Let (7.2) be the logical form of sentence (7.1) that we want to interpret in a context (7.3) where we have some car a and some red object b, and the world knowledge that every car is a vehicle.

(7.32) *The vehicle is red.*

(7.33) $\phi = \mathit{vehicle}(x) \wedge \mathit{red}(x)$

(7.34) $\Gamma = \{\mathit{car}(a), \mathit{red}(b), \forall x\ \mathit{car}(x) \Rightarrow \mathit{vehicle}(x)\}$

Let us further assume that the set of allowable hypothesis is the set of atoms that can be built from all predicates in ϕ and Γ and the individuals in background theory Γ. Thus, the set of allowable hypothesis is given in (7.35). The answers (7.36)–(7.43) each give some abductive explanation why ϕ could be a logical consequence of Γ, and at the same time instantiate the query ϕ.

(7.35) $\Psi = \{\mathit{red}(a), \mathit{red}(b), \mathit{car}(a), \mathit{car}(b), \mathit{vehicle}(a), \mathit{vehicle}(b)\}$

(7.36) $\Delta_1 = \{\mathit{vehicle}(a), \mathit{red}(a)\}$

(7.37) $\Delta_2 = \{\mathit{vehicle}(b), \mathit{red}(b)\}$

(7.38) $\Delta_3 = \{\mathit{vehicle}(a), \mathit{red}(a), \mathit{car}(a)\}$

(7.39) $\Delta_4 = \{\mathit{vehicle}(a), \mathit{red}(a), \mathit{car}(b)\}$

(7.40) $\Delta_5 = \{\mathit{vehicle}(b), \mathit{red}(b), \mathit{car}(a)\}$

(7.41) $\Delta_6 = \{\mathit{vehicle}(b), \mathit{red}(b), \mathit{car}(b)\}$

(7.42) $\Delta_7 = \{\mathit{vehicle}(a), \mathit{red}(a), \mathit{car}(b), \mathit{vehicle}(b)\}$

(7.43) $\Delta_8 = \{\mathit{vehicle}(b), \mathit{red}(b), \mathit{car}(b), \mathit{vehicle}(a)\}$

For our purpose of natural-language interpretation, there are too many explanations. The situation is very similar to model generation where the number of models for a semantic representation is in general much higher than the set of valid natural-language interpretations. However, some explanations can be eliminated immediately as they use redundant assumptions—these explanations correspond to the models that are not minimal in model generation.

In many cases, redundant assumptions will never be considered as hypothesis by an abductive logic programming system. A fact can only become a candidate hypothesis if it either is used directly for refuting a literal of the query or can be reached by some chain of rule applications as a literal that must be refuted. For instance, Δ_8 will actually never be generated by a Prolog-style abductive logic programming system, as there is no sequence of backward-chaining applications of rules in Γ such that all assumptions Δ_8 can be reached. Only the explanations Δ_1–Δ_4 may actually occur as answers.

For natural-language interpretation, we must find a way to eliminate all explanations that are not linguistically valid. For this, there are at least three approaches available.

The first method is to use a more appropriate logical encoding. The definite description in our example is actually modelled by an implicitly existentially quantified formula that does not include the uniqueness condition of singular definites. In a richer logical encoding that models unicity, such as the one discussed in Chapter 5, no explanations except Δ_3 and Δ_7 model the observation. Hence, all other explanations violate one of the conditions for abductive explanations.

A second approach to the problem is that by Baumgartner and Kühn that we have discussed in Section 7.2. Here, that explanation for the resolution of an anaphor is preferred that can be explained by logical consequence from the context. In our example, the definite *the vehicle* triggers an additional inference process that prefers the logically consequent link of the definite *the vehicle* to the car a instead of the accommodated link to the red object b. The link is a logical consequence of an information given by the context Γ, namely that every car is a vehicle.

The third method is to encode the linguistic knowledge implicitly into a system of preferences for abductive explanations. We have implemented this idea for model generation in Section 7.2.5 where the ab-index of each minimal model is used to characterise best explanations. In weighted abduction, this system of preference is implemented by the weights and costs of the logical encoding.

7.3.3 Weights and Costs

The linguistic information that we must implicitly encode in our example is as follows. First, as Hobbs et al. argue, the determiner 'the' indicates an entity that is "*the most salient, mutually identifiable entity of that description*". In other words, the part of the query that represents this entity should be difficult to assume, and rather be shown as some logical consequence of the contextual knowledge. We can encode this implicitly by giving this part of the query high assumption costs, say 100. Then, the cost of that part the query that does not depend on the definite description must be lower, say 20. We consider each fact in the background knowledge to be easily accessible, and give it a minimum cost of 1. The rule that formalises that each car is a vehicle must have some low weight, as it is a rule that encodes easily accessible world knowledge. We choose a weight of 5. If something is a vehicle, we may assume with some low probability that it also is a car. For this form of abductive inference, we assign an assumption cost of 70; it is less than the cost for assuming that some arbitrary object is a car, but still high. We write costs and weight as superscript values, and our query and the background knowledge is now as follows.

(7.44) $\phi = \mathit{vehicle}^{100}(x) \land \mathit{red}^{20}(x)$

(7.45) $\Gamma = \{\mathit{car}^{1}(a), \mathit{red}^{1}(b), \forall x\ \mathit{car}(x)^{70} \Rightarrow^{5} \mathit{vehicle}(x)\}$

In order to answer the query ϕ, the weighted abduction system will refute the negation of ϕ using the background knowledge Γ. If an atom cannot be proven, it is taken as a hypothesis where its assumption cost goes into the overall costs of the proof. If a rule is used to derive new atoms that must be proven, then the rule's weight goes into the cost of the whole proof. For our example, the following explanations can be computed by abductive logic programming proof attempts.

(7.46) $\Delta_1 = \{\mathit{vehicle}^{100}(a), \mathit{red}^{20}(a)\}$

(7.47) $\Delta_2 = \{\mathit{vehicle}^{100}(b), \mathit{red}^{1}(b)\}$

(7.48) $\Delta_3 = \{\mathit{vehicle}^{5}(b), \mathit{car}^{70}(b), \mathit{red}^{1}(a)\}$

(7.49) $\Delta_4 = \{\mathit{vehicle}^{5}(a), \mathit{car}^{1}(a), \mathit{red}^{20}(a)\}$

Solution Δ_1 simply takes $\mathit{vehicle}(a)$ and $\mathit{red}(a)$ as two hypothesis that obviously are abductive explanations of the query. However, the assumptions costs are $100 + 20 = 120$, which makes this explanation the most expensive that can be computed by weighted abduction. Solution Δ_2 is slightly cheaper (101) since it only assumes $\mathit{vehicle}(b)$ and then uses the easily accessible knowledge that b is red. Then, Δ_3 includes some abductive inference where $\mathit{vehicle}(b)$ is "proven" by the assumption that b is a car. The derivation cost is 5, as our rule has a weight of 5, and the assumption cost for $\mathit{car}(b)$ is 70. Again, we make use of the fact that b is red. Hence, the cost of the answer Δ_3 is $5 + 70 + 1 = 76$. The best solution, however, is Δ_4 which derives that a is a red vehicle by proving that it is a vehicle at a very low cost of 6, and assuming that a is red with cost 20. The overall cost of 26 is the lowest possible for our example.

Our example illustrates the intuition behind weighted abduction as a method that uses numerical annotations in the logical specification to control the search for a best abductive explanation. By branch-and-bound-search, such a best explanation can often be determined quickly. Weighted abduction implements abductive inference in natural-language interpretation in a way that suppresses some effects of the combinatorial explosion caused by the large number of abductive explanations. However, as Hobbs et al. state themselves, the actual numerical values are ad hoc, and the high amount of interaction during computation it very difficult to engineer some universal and reliable system of annotations. There is, at present, no linguistic theory behind weights and costs. Nevertheless, weighted abduction has been used with some success in the TACITUS natural language system on a range of examples.

7.3.4 Applications

Hobbs et al. [91] gives a variety of applications for (weighted) abductive inference in natural-language interpretation. In the following, we will shortly present some of these applications.

7.3.5 Definite Reference

Weighted abduction can deal with different kinds of definite reference such as we have discussed in some detail in Chapter 5. Weighted abduction implements the empirically testified preference of anaphoric resolution over accommodation by high assumption costs of definite descriptions in the query. Examples of bridging, such as (7.50), are dealt with in a similar way as in our analysis. The existence of the referent for the definite *the engine* is proven by using[2] a rule (7.51) that has some low inference cost. In other words, from the existence of a car we can follow with low inference costs that there also is an engine that belongs to the car. This engine is used for proving the existence of the engine in sentence (7.50).

(7.50) *I bought a new car last week.* ***The engine*** *is already giving me trouble.*

(7.51) $\forall x\ \mathsf{car}(x) \Rightarrow \exists y\ \mathsf{engine}(y) \wedge \mathsf{of}(x,y)$

Note that the logical form here is actually not Horn. Formula (7.51) must be translated into a Horn clause that approximates the semantics of (7.51) and that is acceptable by the abductive logic programming system. This may for instance require a special predicate that implements negation as failure.

Weighted abduction can correctly model examples such as (7.52) where the definite description is actually a determinative definite noun phrase that includes all information needed for its resolution. In examples such as this, there is no way to prove the existence of the definite referent from the context. Hence, the abductive explanation that simply assumes the existence of Jon's rabbit should be the cheapest one possible.

(7.52) ***Jon's rabbit*** *is cute.*

Weighted abduction gives a clear preference for only one single solution, as there is not really some tractable way of determining "second best" solutions.

In both weighted abduction and minimal model generation, sentences such as (7.53) are a problem because there is an ambiguity that cannot be resolved by the usual heuristics. Does *the old man* refer to Konrad himself or maybe his supervisor? Domain-minimal model generation prefers the reading where Konrad is the old man, while weighted abduction may chose some arbitrary reading depending on how the weights and costs are chosen.

(7.53) *Konrad's PhD is terrible.* ***The old man*** *should really read it again.*

[2] Here, as in the following examples, we assume that suitable values for weights and costs have been chosen, and leave them out for having a simpler presentation.

7.3.6 Composite Noun Phrases

Weighted abduction has been used in the analysis of complex noun phrases that are composed of given and new information. An example for such noun phrases is given in (7.54) in the form of the definite *the Boston office.*

(7.54) ***The Boston office*** *called.*

Ignoring the problem we must somehow expand the metonymy to "[Some person at] the Boston office called", the primary interpretative problem in sentence (7.55) is to determine the implicit relation between the person that called and the office, and the relation between Boston and the office. The approach proposed by Hobbs et al. treats the sentence roughly[3] as a formula (7.55).

(7.55) $\exists x \exists y\ \mathsf{call}(x) \wedge \mathsf{person}(x) \wedge \mathsf{rel}(x,y) \wedge \mathsf{office}(y) \wedge \mathsf{nn}(y, boston)$

The relation rel is a placeholder for a relation that holds between an individual and a location, while nn stands for some relation that holds between a location and another one. The logical encoding informally states that some person called who somehow is related to some office that somehow is related to Boston. By giving $\mathsf{rel}(x,y)$ and $\mathsf{nn}(y, boston)$ high assumption costs, we enforce an explanation that uses some instances for relations that are derivable from the background knowledge. The following are examples for rules in the database that may be used for such a derivation.

(7.56) $\forall x \forall y\ \mathsf{in}(x,y) \Rightarrow \mathsf{nn}(x,y)$

(7.57) $\forall x \forall y\ \mathsf{works\text{-}for}(x,y) \Rightarrow \mathsf{rel}(x,y)$

Suppose that we have given the additional information in the database that o_1 is an office in Boston and that a person jon works for office o_1. Then (7.58) can be taken as an explanation for (7.55) where $\mathsf{call}(jon)$ is the only new information that must be assumed. The explanation corresponds to a natural-language interpretation of sentence (7.55), namely that jon is the person who called and works for the office o_1 in Boston.

(7.58) $\Delta = \{\mathsf{call}(jon), \mathsf{person}(jon), \mathsf{rel}(jon, o_1), \mathsf{office}(o_1),$
$\mathsf{nn}(o_1, boston), \mathsf{works\text{-}for}(jon, o_1), \mathsf{in}(o_1, boston)\}$

Examples such as (7.54) illustrate how some missing information in the semantic representation of a sentence can be filled in by abductive reasoning. Based on a similar logical encoding, the Kimba model generator is able to reproduce the analysis of the example without using any heuristic control other than the world knowledge that persons are neither cities nor offices. Local minimality suffices here to identify the correct interpretation.

The weights and costs in weighted abduction are often used in a way that simulates the effects of local minimality. In our example, weighted abduction makes sure that a maximum of information that is easily accessible in the

[3] Again, we ignore temporal issues.

database is used, while $\mathit{call}(jon)$ remains the only true assumption. In model generation, we do not distinguish assumptions and provable facts, but the characteristic properties of a minimal model make sure that information which can be derived as a logical consequence of the context are preferred. The effect of the two approaches to abductive interpretation often is the same: the implementation of conversational conventions as computable constraints.

A difficulty with the abductive approach in general is that there may be many rules such as (7.56) and (7.57) that give possible instantiations for the placeholder relations. While weights may control which instances are chosen, it is not entirely clear how any system of weights can guarantee that the selection is correct. Likewise, model generation will produce a variety of models where we cannot decide which ones truly represent preferable interpretations. This is especially true when contextual information is missing in the background knowledge. Model generation seems to be somewhat more sensible to missing knowledge than weighted abduction which only makes assumptions that can be reached by some backward-chaining interpretation of the rules in the database.

7.3.7 Resolving Ambiguity

Common-sense reasoning helps listeners to associate words with the correct concepts. A typical example of word-level lexical ambiguities is given by example (7.59).

(7.59) *The plane taxied to the terminal.*

The interpretative process must identify which semantic concepts are behind such words as "plane", "taxied", and "terminal", all of which carry more than one meaning. We can express a classification of concepts as rules in the database. The following rules give some background knowledge to the ambiguous words.

(7.60) $\forall x\ \mathit{airport\text{-}terminal}(x) \Rightarrow \mathit{terminal}(x)$

(7.61) $\forall x\ \mathit{computer\text{-}terminal}(x) \Rightarrow \mathit{terminal}(x)$

(7.62) $\forall x \forall y\ \mathit{ride\text{-}cab}(x, y) \wedge \mathit{person}(x) \Rightarrow \mathit{taxi}(x, y)$

(7.63) $\forall x\ \mathit{move\text{-}on\text{-}ground}(x) \wedge \mathit{airplane}(x) \Rightarrow \exists y\ \mathit{taxi}(x, y)$

(7.64) $\forall x\ \mathit{wood\text{-}smoother}(x) \Rightarrow \mathit{plane}(x)$

(7.65) $\forall x\ \mathit{airplane}(x) \Rightarrow \mathit{plane}(x)$

Terminal is a hyponym of both airport-terminal and computer-terminal, as stated by the rules (7.60) and (7.61). Taxiing may refer to the process of a person riding a cab, or an air-plane moving on the ground, as stated by rules (7.62) and (7.63). Finally, the word "plane" may refer to a wood-smoother or an airplane.

The task of weighted abduction in the example is to compute an answer for the query (7.66) that somehow makes explicit that the sentence (7.59) refers to an air-plane moving on the ground to an airport terminal. An essential hint that we give to the system is rule (7.67) which expresses the common knowledge that whenever we have an airport, we also have an air-plane and an airport-terminal.

(7.66) $\exists x \exists y\ \mathit{plane}(x) \wedge \mathit{taxi}(x,y) \wedge \mathit{terminal}(y)$

(7.67) $\forall z\ \mathit{airport}(z) \Rightarrow \exists x \exists y\ \mathit{plane}(x) \wedge \mathit{airport\text{-}terminal}(y)$

The analysis intuitively derives a minimal explanation (7.68) as follows. First, if $\mathit{plane}(x)$ in the query is explained by $\mathit{airplane}(c_1)$, then the fact $\mathit{airplane}(x)$ that is part of one of the rule (7.63) for $\mathit{taxi}(x,y)$ has already been assumed. Hence, the interpretation that uses this rule is cheaper than that where $\mathit{taxi}(x,y)$ requires more assumptions with respect to a person and some cab. If we further explain $\mathit{terminal}(y)$ by $\mathit{airport\text{-}terminal}(c_2)$, we can reduce the assumption costs of $\mathit{airport\text{-}terminal}(c_2)$ and $\mathit{airplane}(c_1)$ by using the rule (7.67) that assumes an airport c_3. The cost of assuming a fact in weighted abduction is computed on the basis of the cheapest way to derive it as a candidate hypothesis. Any additional uses of an assumption in the computation of an explanation are free. As a result, the reuse of assumptions is preferred.

(7.68) $\Delta = \{\mathit{plane}(c_1), \mathit{taxi}(c_1,c_2), \mathit{terminal}(c_2), \mathit{airplane}(c_1),$
$\mathit{airport\text{-}terminal}(c_2), \mathit{airport}(c_3)\}$

In model generation, we have no device analogous to the derivation costs in weighted abduction. In our example, the reuse of information makes the explanation (7.68) cheaper than other ones, even though it requires an assumption $\mathit{airport}(c_3)$ which is not required in other explanations. In comparison, the preference for a certain model is entirely determined by the atoms that constitute its representation. When we use minimal model generation on a similar logical encoding, the preferred interpretation that is computed is some model that corresponds to an explanation (7.69). The model uses less individuals as it does not require an airport. The correct interpretation is computed only if we leave out the "hint" that is necessary in weighted abduction. Domain minimality actually prevents us from identifying the most intended interpretation.

(7.69) $\Delta = \{\mathit{plane}(c_1), \mathit{taxi}(c_1,c_2), \mathit{terminal}(c_2), \mathit{airplane}(c_1),$
$\mathit{computer\text{-}terminal}(c_2)\}$

7.3.8 Discussion

Apart from the applications presented in Section 7.3.4, weighted abduction has been put to work on such different phenomena as discourse structure, metonymy, redundancy, and various combinations of pragmatic problems. The examples that have been investigated are real-world texts from equipment failure reports, naval operation reports, and newspaper articles on terrorist activities. For a more thorough presentation of the method and its applications, we refer to Hobbs et al. [91].

In the previous chapters, we have presented model generation as a reasoning method that can be seen as a competitor of weighted abduction as a computational tool for text interpretation. Before we discuss the differences of the methods, we would like to point out that apart from technicalities, the approaches themselves are very similar.

7.3.9 Similarities

First, weighted abduction and minimal model generation conceptualise text interpretation as a task that computes an explanation for truth. The intuition behind this can be taken as a universal approach to natural-language understanding. Even questions or statements with a high pragmatic content can be captured in principle by interpretation as abduction, since interpretation in general explains the truth of a speaker's intentions, emotions, and believes as well as the plain truth of an utterance. Given a suitable logical theory of intentions, emotions, and believes, abductive reasoning explains pragmatic phenomena as well as semantic ones—if we can solve the representational and complexity problems that are inevitable when reasoning about pragmatics. By generating abductive explanations of why a text has a conversationally sensible rhetoric structure, weighted abduction can even be used for discourse structure construction as well as for interpretation. There are few reasons to suspect that minimal model generation may not be useful as well for such inference tasks.

Second, the computation of truth is based on well-known methods in automated reasoning. Minimal model generation and weighted abduction are basically classical first-order inference methods whose purpose is the computation of best explanations. The use of standard tools, or rather of standard tools that have been tweaked to fit a special purpose, leads to a semantic representation language that often can only approximate the meaning of the natural-language input in the sense of a truth-conditional natural-language semantics. However, this disadvantage is paid for by the actual ability to investigate the phenomenon of inference by experimentation.

Third, both model generation and weighted abduction face the same principal problem of abductive inference, namely the combinatorial explosion of explanations for an observation. As natural-language semantics is still dominated by the topic of representation, the criteria that distinguish a valid natural-language interpretation and the methods how these can be determined remain almost entirely unknown. Inference as abduction gives a practical tool for investigating these criteria, either in the form of a model generator or an abductive logic programming system.

7.3.10 Differences and Comparison

Apart from some minor technical details, the main difference of interpretation as weighted abduction to interpretation as model generation is the way in which the two approaches deal with the overgeneration problem of abductive explanations.

In weighted abduction, the main filter mechanism for unwanted explanation is actually the method with which answers for queries are computed. Given a query A and a database $\Gamma = \{B \Rightarrow A, B, C \Rightarrow D\}$, the abductive logic programming system will try to refute $\neg A$ either by assuming A as a hypothesis or by using the rule $B \Rightarrow A$ and the fact B by backward-chaining. The literals C and D cannot be reached at all with query A, and thus will never be part of an answer. In this sense, abductive logic programming is goal-oriented, as it only uses assumptions if they contribute something to the goal of explaining the query. Rules that do not contribute to the computation of an answer will never be executed. This can result in a considerable advantage in tractability as the knowledge base grows larger. However, the downside of the general approach is that without further technical machinery, there is no guarantee that the answers which are computed meet the formal conditions of abductive explanations. That is, abductive logic programming as implemented in weighted abduction sometimes gives sets of assumptions Δ which are not consistent with the database Γ of background knowledge.

In principle, it should be possible to augment weighted abduction with some proof method for finite satisfiability. However, as weighted abduction does not efficiently enumerate explanations, it seems to be a considerable technical problem to combine weighted abduction and a consistency check such that the resulting method still works in practice. The conversational maxim of consistency is not guaranteed by the current implementation. This has some serious disadvantage for natural-language interpretation: as long as consistency is not maintained by the proof procedure, there may be cases where a better logical encoding actually leads to inconsistent "explanations". There is little point in further developing truth-conditional theories in computational semantics if a better logical modelling negatively interacts with the inference process!

In model generation, there is no goal-orientedness or flow of control in principle. Model generation by itself has no operational filter mechanism that eliminates irrelevant literals such as C and D in our example. The answer to this problem is some form of minimality which prefers models that can explain the same observations with some subset of assumptions. The price that we have to pay for maintaining the formal properties of abductive explanations is a complexity problem. There seems to be no way to prevent a rule from interacting with the computation process once it is provided with the database, not even if it seems to be irrelevant for the task at hand. Furthermore, the computation of minimal models is considerably harder than that of simply proving satisfiability.

The weights in weighted abduction are a method for controlling the search for explanations. These weights can define a fine grained system of preferences that includes the plausibility of certain assumptions as well as some measure for the difficulty of inferring certain knowledge. However, weights currently have no theory that determines how they must be selected. As we have shown, our computational treatment of linguistic phenomena by minimal model constraints can be firmly based on contemporary linguistic theories.

8 Implementation

Keep It Simple!
(Charles H. Moore)

Overview: This chapter describes the finite model generator KIMBA.

8.1 Introduction

Natural-language interpretation needs model generation methods that selectively enumerate the models of semantic representations. Finite model generation based on constraint solving provides such enumeration capabilities in an easily accessible way.

The finite model generator KIMBA[1] is based on the $\mathcal{MQL}$ specification language presented in Section 3.3 and its translation into constraints presented in Section 3.4. KIMBA implements an efficient enumeration procedure for finite models and can also enumerate models with certain minimal model properties.

The specification language $\mathcal{MQL}$ itself does not have an efficiently computable clause normal form, but KIMBA can avoid normalisation thanks to the properties of the translation. The basic two-valued logic $\mathcal{MQL}$ described in Section 3.3.5 can be generalised in the implementation to a collection of logics with finitely many truth values. The user may redefine the semantics of logical connectives and thus design her own logics.

In the following, I start by sketching the general system architecture of KIMBA in Section 8.2. Then I discuss both the syntax of the specification language in Section 8.3, and, in Section 8.4, the way how semantics is assigned to it in the form of logic definition structures and propagator procedures. Finally, I outline the control mechanism of the search for models and the variety of proof procedures that are part of KIMBA in Section 8.5 and give some results about KIMBA's performance in Section 8.6.

[1] The original Kimba, the Jungle Emperor, is a small white lion from a Japanese cartoon series in the 1970s. KIMBA is the small brother of LEO [92].

8.2 System Architecture

The system architecture of KIMBA is depicted in figure 8.1. A logical input is first translated into a combinatorial constraint problem over finite-domain integer variables. This translation relates to one certain frame of constants that is determined by the problem itself and a current universe of first-order individuals. Together with a control strategy that describes how and which models are to be generated, the combinatorial problem is handed to the constraint solver.

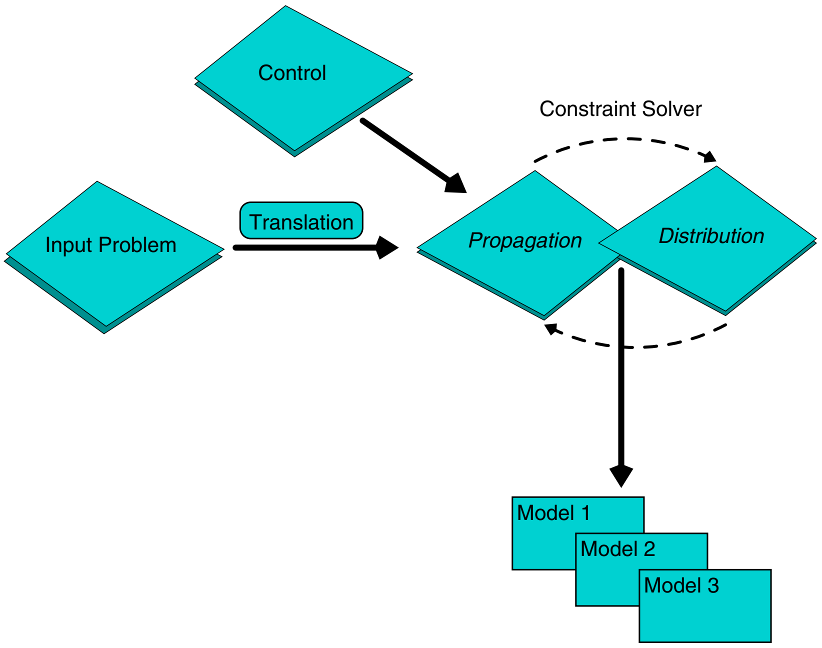

Fig. 8.1. KIMBA's system architecture at a glance

The actual constraint solving process consists of two separate processes that are called iteratively. The first process is **propagation** which restricts the possible values of the finite-domain variables using the knowledge encoded in the constraints. Changing the value of a variable often imposes new constraints to the values of other variables as well, and propagation is applied until the set of constraints is stable, i.e., no further constraints can be derived and added. The second process, **distribution**, selects variables and restricts their values provisionally. The selection of the variable(s) and the actual form of their restriction is done heuristically. After a distribution step, constraints will propagate again, and so forth.

The iterative process of propagation and distribution continues until all variables are uniquely determined or a failure occurs. A model in KIMBA is represented internally as a list for the set of atoms that has been build from the constant frame that defines the range of quantification during model construction. The table's entries are pairs of atoms and finite-domain integer variables whose values each represent the truth value of the associated atom. Each change of a value in this table results in a different interpretation. By distributing over the variables' values, the constraint solver systematically enumerates all finite models with regard to the constant frame from which the set of atoms is derived.

A failure, i.e., an assignment that leads to an unsatisfiable set of constraints, evokes backtracking, and the value of the variable that lead to the failure is constrained differently. The control strategy may both add constraints to the constraint problem itself, or start some additional inference process for each solution that is produced by the constraint solver. For instance, the constraint solver may produce some model for which the control strategy then proves whether it is a locally minimal model.

The constraint solving module of the finite model generator can be seen as independent from the logic and the problem specification considered. Once a suitable translation from the logical language into a constraint language is given, solving a model generation problem reduces to the application of any constraint solver that can deal with the constraint language in question. In KIMBA, the constraint solver used is provided by OZ [69], the constraint logic programming language in which KIMBA is implemented.

The main part of KIMBA is actually the translation from the input into a constraint language for finite-domain integer variables. The other parts of the system consist mainly of the implementation of the basic data structures and operations such as β-reduction. KIMBA has been implemented in the spirit of as exemplified by the leanT^AP tableaux prover [39]. The philosophy of lean deduction is it to have automated reasoning systems that are as small as possible. Lean automated reasoning aims at providing simple systems suitable for research and education rather than high-speed systems where efficiency is bought by complexity in the implementation. The whole KIMBA system is only about 30kBytes of OZ code, including a minimalistic graphical user interface for loading problems, controlling the parameters of the search, and visualising the results.

8.3 The Syntax

The syntax of KIMBA is derived from the $\mathcal{MQL}$ variant of Church's simply typed λ-calculus presented in Section 3.3.

8.3.1 Logical Constants

Figure 8.2 shows the basic set of logical connectives and quantifier constants. Other quantifiers, connectives and determiners can be defined as λ-terms in the problem specification, or permanently added to the system in the logical definition structures (see Section 8.4). The Herbrand equality '$\doteq$' from $\mathcal{MQL}$ is denoted simply by the equality sign =.

$\mathcal{MQL}$	Kimba
$\neg$	`not`
$\vee$	`or`
$\wedge$	`and`
$\Rightarrow$	`implies`
$\Leftrightarrow$	`equiv`

$\mathcal{MQL}$	Kimba
$\forall$	`forall`
$\exists$	`exists`
Every	`every`
Some	`some`
More	`more`

Fig. 8.2. Standard Connectives and Quantifiers

Quantifier	Example	Truth condition
`atLeast`	`[atleast man 5]`	$\lvert man \rvert \geq 5$
`atMost`	`[atMost man 5]`	$\lvert man \rvert \leq 5$
`exactly`	`[exactly man 5]`	$\lvert man \rvert = 5$

Fig. 8.3. Cardinality Quantifiers

Apart from the standard quantifier constants, Kimba provides a set of cardinality quantifiers that can be used to constrain the cardinality of sets. The set of available cardinality quantifiers together with some examples of their use is given in Figure 8.3.

8.3.2 Formulas

Formulas in Kimba follow the syntax of $\mathcal{MQL}$. Hence, the only terms allowed in a $\beta\eta$-normalised formula are parameter constants or bound variables. Applications of the form $h(x_1, \ldots, x_n)$ are written as `[h` x_1 `...`x_n`]`. Likewise, a quantified $\mathcal{MQL}$ formula Q(T)(U) is written as `[Q T U]`.

A λ-abstraction $\lambda x_\iota\ p(x)$ is written as `lam(x#i [p x])`. The bound variable of an abstraction must be given an explicit type, such as in this case by `x#i`.

We have basic types '`i`' for individuals, '`o`' for truth values, and the type '`n`' for numbers. Numbers are used in general only for cardinality constraints. Higher-order types are written in a backward fashion such that the goal type of a function is always the first type. For instance, a Kimba type `[o [o i] [o i o]]` denotes the $\mathcal{MQL}$ type $(o \to o \to o) \to (\iota \to o) \to o$. Kimba can up to some extend deal with polymorphic types, for instance in higher-order definitions of predicate constants whose arguments are polymorphic. A basic type '`x`' is reserved for denoting polymorphic types α.

Fig. 8.4 shows some formulas and some of their possible translations into Kimba's specification language. The redundancy in having a set of standard quantifiers as well as a set of linguistically motivated generalised determiners gives us some freedom in using whatever logical form we think is more elegant or appropriate. For instance, formula 5 is a formula in standard first-order notation whose semantics is equivalent to that of formula 6 which uses a generalised determiner in a more compact representation. Note that the order of arguments of n-ary predicates in Kimba is different from the order of arguments in $\mathcal{MQL}$, for no especially good reason except for compatibility to some older first-order versions of the program.

#	$\mathcal{MQL}$ Formula	KIMBA
1	$\text{EVERY}(\mathit{man})(\lambda x_\iota\ \mathit{eat}(x) \wedge \mathit{sl}(x))$	`[every man lam(x#i [and [eat x] [sl x]])]`
2	$\text{NO}(\mathit{l}(jon))(\lambda x_\iota\ \mathit{l}(jon, x))$	`[no lam(x#i [l x jon]) lam(x#i [l jon x])]`
3	$\text{NO}(\lambda x_\iota\ \mathit{l}(jon, x))(\mathit{l}(jon))$	`[no [l jon] lam(x#i [l x jon])]`
4	$\text{MORE}(\lambda x_\iota\ \neg \mathit{man}(x))(\mathit{man})$	`[more lam(x#i [not [man x]]) man]`
5	$\forall x_\iota\ \mathit{w}(x) \Rightarrow \mathit{l}(jon, x)$	`forall(x#i [implies [w x] [l jon x]])`
6	$\text{EVERY}(\mathit{w})(\lambda x\ \mathit{l}(jon, x))$	`[every w [l j]]`

Fig. 8.4. Specification in KIMBA

8.3.3 Problem Specifications

A **problem specification** in KIMBA is a structure that consists at least of the following parts.

- a declaration of the signature for the non-logical constants
- a declaration of which logic used for formalising the problem
- a list of formulas

Optionally, we can have a set of higher-order definitions, a remark that describes the problem in natural language, and an entry that names a default constraint solving engine (cf. Section 8.5) for the problem.

8.3.4 A Small Example

Figure 8.5 shows a small problem specification. The specification starts with an entry at the label `remark` that describes the problem, in this case a formalisation of the sentence *two girls love jon*. The entry with the label `logic` specifies that the logic we used is the classical two-valued one. The label `engine` specifies the problem solving engine. For our problem, we use the `'local minimal'` engine that enumerates locally minimal first-order models. This means that the solving engine is allowed to extend the initial universe of discourse if necessary, and that only those models are presented which meet the local minimality condition.

The initial constant frame is given by the problem specification. We have parameter constants `girl`, `love`, and `jon`, all of which have different types. We also have a higher-order definition for a new quantifier constant `two` such that `two`$(P)(Q)$ is true iff P has exactly two members and is a subset of Q.

Finally, the set of formulas specifies that Jon is no girl and that two girls love Jon. The problem specification can loaded into KIMBA, and the only locally minimal model that is yield by the model generation process has a representation as the following set of atoms.

```
(8.1)  { [girl C1], [girl C2],
         [love C1 jon], [love C2 jon] }
```

Here, the constants `C1` and `C2` are newly generated ones of type `i` that have been added by the model generator while the first-order domain of constants was extended.

Some larger example specifications can be found in Appendix A.

```
p( remark: 'Two girls love jon.'
   logic: classical
   engine: 'local minimal'
   constants: are(girl:[o i] love:[o i i] jon:i)
   definitions:
     are(two: lam(p#[o i]
                 lam(q#[o i]
                      ['and' [exactly p 2] [every p q]])))
   formulas: [ ['not' [girl jon]]
               [two girl lam(x#i [love x jon])] ])
```

Fig. 8.5. A problem specification for KIMBA

8.3.5 Definitions

Definitions occur as part of a problem specification and are λ-terms that replace the occurrences of defined constants within the specification before the actual model generation process starts. KIMBA uses the standard β-normalisation to reduce expressions containing definitions into ones that have the function free quantified form required by the translation into constraints. Figure 8.6 shows two definitions, one for the determiner THE and one for the determiner MY. The second definition makes use of the first one. A definition may generally refer to other defined symbols as long as the definition expansion terminates, i.e., there are no cyclic definitions.

```
the: lam(p#[o i]
         lam(q#[o i]
             ['and' [exactly p 1] [every p q]]))

my: lam(p#[o i]
        lam(q#[o i]
            [the lam(x#i ['and' [p x] [isOf speaker x]]) q]))
```

Fig. 8.6. Defining the determiners THE and MY by λ-terms

8.4 The Semantics

Logics are, in general, defined by the semantics of their logical constants. In KIMBA, the semantics of a logical constant is given operationally by the definition of a concurrent procedure that manipulates the values of finite-domain integer variables. A user can program these procedures, and combine them into structures that define which logical constants are available within a logic and in which way they behave.

8.4.1 Logic Definition Structures

A logic based on the $\mathcal{MQL}$ specification language can be defined in Kimba by building a (LDS). LDSs are unique to Kimba. Unlike other finite model generators restricted to some fixed (classical) logic, LDSs provide Kimba with a mechanism for implementing different logics in a modular way. The logics in Kimba can vary in the logical constants that are available, their semantics, and in the number of truth values that can occur as the denotation of a formula.

An LDS is a mapping from constant symbols into propagator procedures. The constant symbols mentioned are the logical constants of the logic, for which the propagator procedures define an operational semantic. Figure 8.7 shows a partial LDS for classical 2-valued logic. The constants for the various connectives are mapped directly into predicates that are provided by Oz, for instance '`FD.conj`' as the propagator procedure that treats 2-valued conjunction. LDSs may define the semantics of truth constants such as '`true`' and '`false`' and the semantics of arithmetic predicates on integers such as '`>`'. We also have a slot for defining the equality predicate '`=`'. This allows us to implement special forms of equality if necessary.

Additionally, LDSs also determines the type of all logical constants and the number of truth values and a translation from the truth values into a human-readable form. For instance, we usually want to interpret the highest truth value as "true" and the lowest one as "false" while other truth values might have different purposes. As mentioned earlier, the symbol '`x`' indicates a polymorphic type. A typical example is the definition of the quantifier '`exists`' that has a type $(\alpha \to o) \to o$, or, in Kimba's syntax, a type `[o [o x]]`.

An LDS does not necessarily have to contain all logical constants that are used in other logics. A user may give only a basic set of connectives, quantifiers and determiners (if any) and implement other logical constants by the definition mechanism presented in the previous section.

8.4.2 Propagator Procedures

Implementing a logic in Kimba means to implement the **propagator procedures** for the logical constants. A propagator procedure in Oz is a concurrent procedure that observes and propagates the changes in the values of a set of variables. In our case, these variables are finite-domain integer variables that represent the interpretations of formulas. A propagator procedure operationally defines a relation between the variables it observes. In Kimba, each propagator procedure implements the relation defined by its associated logical constant, and the mapping of logical constants to their associated propagator procedures is defined by the logics' LDS.

8.4.3 Connectives

A propagator procedure for a binary connective will observe three variables, one for the whole formula and two for the two components of the formula.

```
ClassicLDS =
lds(truths: are('false' 'true')
    constants: are('true': (o # 1)
                   'false': (o # 0)
                   '=': ([o x x] # proc {$ FA SA Root}
                                      if FA == SA then
                                         Root = 1 else
                                         Root = 0 end
                                   end)
                   '>': ([o n n] # proc {$ [FA SA] Root}
                                      if FA > SA then
                                         Root = 1 else
                                         Root = 0 end
                                   end)
                    'succ': ([o n n] # proc {$ [FA SA] Root}
                                         if FA == SA+1 then
                                            Root = 1 else
                                            Root = 0 end
                                      end)
                   'not': ([o o] # FD.nega)
                   'and': ([o o o] # FD.conj)
                   'implies': ([o o o] # FD.impl)
                   'exists': ([o [o x]] #
                             proc {$ [Insts] Var}
                                {FD.reified.sum Insts '>:' 0 Var}
                             end)
                   'atLeast': ([o n [o x]] #
                              proc {$ [Insts Card] Var}
                                 {FD.reified.sum Insts '>=:'
                                  Card Var}
                              end)
                   'some': ([o [o x] [o x]] #
                           proc {$ [InstsA InstsB] Var}
                              {FD.reified.sum
                               {List.zip InstsA InstsB
                                fun {$ IA IB} {FD.conj IA IB} end}
                               '>:' 0 Var}
                           end)))
```

Fig. 8.7. A partial LDS for a classical two-valued logic

If any of the values of these variables changes, the values of the other two variables will be modified accordingly, if possible. Otherwise, propagation fails and the variable assignment is rejected.

KIMBA's generic translation of deduction into OZ constraints allows us to design dedicated, optimised propagator procedures for each logic. OZ itself already has an efficient set of propagators for the standard connectives in classical two-valued logic.

Figure 8.8 shows two operationally equivalent propagator procedures for negation, `N1` and `N2`, in a 3-valued logic. The parameters of the procedures are two finite-domain integer variables. Variable `V1` corresponds to the truth value of the formula in the scope of the negation, while `V2` is used for the truth value of the whole formula.

```
proc {N1 V1 V2}                          proc {N2 V1 V2}
   [V1 V2] ::: 0#2                          [V1 V2] ::: 0#2
   thread                                   {FD.minus 2 V1 V2}
      if V1 = 0 then V2 = 2              end
      [] V1 = 1 then V2 = V1
      [] V1 = 2 then V2 = 0
      [] V2 = 0 then V1 = 2
      [] V2 = 1 then V1 = V2
      [] V2 = 2 then V1 = 0
      end
   end
end
```

Fig. 8.8. Two variants of a 3-valued negation

Procedure `N1` first restricts its two parameters to values between 0 ("false") and 2 ("true"), where 1 is used for the truth value "undefined". Then it starts a concurrent process which waits for one of these parameters to be determined and determines the other one accordingly. Propagation works in both directions, so whenever either the truth value of the formula in the scope of the negation or the the truth value of the whole formula becomes determinate, then so becomes the other value.

Procedure `N2` implements the same behaviour, simply by constraining the values of V_1 and V_2 by the equation $V_2 = 2 - V_1$. Experiments show that the "symbolic" propagation used in `N1` is far more effective than the "arithmetic" propagation in `N2` as long as the number of truth values in the logic considered is relatively small.

8.4.4 Monadic Quantifiers

Monadic quantifiers Q such as $\exists$, $\forall$ etc., are used for formulating the properties of single sets. In Kimba, quantified formulas $\mathsf{F} = \mathrm{Q}(\mathsf{P}_{\alpha\to o})$ are translated into constraints over the instantiations $\mathsf{P}(\mathcal{C}_\alpha)$ that are possible using a finite domain $\mathcal{C}_\alpha$ of constant symbols of type α. For classical logics, Oz provides some built-in propagators for **constraint reification** that come handy for implementing these quantifiers. Figure 8.9 shows two procedures that implement existential and uniqueness quantification. The truth conditions implemented by the propagator procedures can be formalised as follows.

$$[\![\mathit{exists}(P_{\alpha\to o})]\!]_{\mathcal{I}} = 1 \textit{ iff } \textstyle\sum_{i=1}^{n} [\![P(X_i)]\!]_{\mathcal{I}} \geq 1 \textit{ for } X_i \in \mathcal{C}_\alpha \tag{8.2}$$

$$[\![\mathit{unique}(P_{\alpha\to o})]\!]_{\mathcal{I}} = 1 \textit{ iff } \textstyle\sum_{i=1}^{n} [\![P(X_i)]\!]_{\mathcal{I}} = 1 \textit{ for } X_i \in \mathcal{C}_\alpha \tag{8.3}$$

The propagator procedures have two parameters, `Var` for the truth value of the whole quantified formula, and a list of truth variables `Insts` for the instantiations $P(X_i)$ over the current domain. Oz only supports reification for 2-valued classical logic, so all variables considered are 0/1-integer variables.

```
proc {Exists Var [Insts]}
  {FD.reified.sum Insts '>:' 0 Var}
end

proc {Unique Var [Insts]}
  {FD.reified.sum Insts '=:' 1 Var}
end
```

Fig. 8.9. Implementing quantification as constraint reification

The procedures each create a reified sum constraint over the list of variables in `Insts`. If the sum constraint becomes determined, i.e., when it is known whether it is satisfied or violated, the reification constraint determines the truth value `Var` as well. If, on the other hand, `Var` becomes determined, then the sum over `Ints` is constrained accordingly.

Oz does not provide constraint reification into many-valued truth variables and in the case of many-valued logics, the semantics of the reified constraints must be programmed in the form of a more complex case analysis. Figure 8.10 shows the implementation of the existential quantification in some 3-valued logic.

8.4.5 Diadic Quantifiers

Diadic quantifiers (also known as generalised determiners) are the logical constants that formulate relations between two sets. The difference between the implementation of a monadic quantifier and that of a diadic quantifier is an additional parameter for the propagator procedure that consists of a list of truth variables for the instantiations of a second set. Figure 8.11 exemplifies how diadic quantifiers can be defined in general. The procedure `More` defines the operational semantics of the diadic quantifier More. The truth values for two instantiations over the domain are given as lists `InstsA` and `InstsB`. The propagator procedure reifies the constraint that the sum of the truth values in `InstsA` must be greater than the sum of the values in `InstsB`. Hence, the truth value of a formula More$(P)(Q)$ is determined by the two sums of the instantiations $P(\Sigma_\alpha)$ and $Q(\Sigma_\alpha)$.

The diadic quantifier No is defined as follows: the formula No$(P)(Q)$ is true iff $P(x) \wedge Q(x)$ is false for all x of the appropriate type. The implementation uses the constraint that the sum of the truth values of all conjuncts $P(x) \wedge Q(x)$ must be 0 iff No$(P)(Q)$ is true. Again, we use constraint reification as an economical way to express the relationship between the truth value

```
proc {Exist3 Var [Insts]}
   thread
      cond Var = 2 then {FD.atLeast 1 Insts 2}
      [] Var = 0 then Insts ::: 0
      [] Var = 1
      then
         {FD.exactly 0 Insts 2}
         {FD.atLeast 1 Insts 1}
      [] Insts ::: 0
      then Var = 0
      [] {FD.exactly 0 Insts 2}
         {FD.atLeast 1 Insts 1}
      then Var = 1
      [] {FD.atLeast 1 Insts 2}
      then Var = 2
      end
   end
end
```

Fig. 8.10. Implementing a 3-valued existential quantification

```
proc {More Var [InstsA InstsB]}
       {FD.reified.sum InstsA '>:'
          {FoldL InstsB fun {$ I Z} {FD.plus I Z} end 0}
        Var}
end

proc {No Var [InstsA InstsB]}
       {FD.reified.sum
          {List.zip InstsA InstsB
                    fun {$ IA IB} {FD.conj IA IB} end}
                    '=:' 0 Var}
end

proc {Some Var [InstsA InstsB]}
        {FD.reified.sum
           {List.zip InstsA InstsB
                     fun {$ IA IB} {FD.conj IA IB} end}
                     '>:' 0 Var}
end
```

Fig. 8.11. Some implementations of diadic quantifiers

of a formula and the instantiations of its parts. The diadic quantifier Some is defined almost exactly as No. Here, this sum over the conjuncts must be bigger than 0 if the top-level formula is to be evaluated to true.

Kimba has a special group of diadic quantifiers that are called **cardinality quantifiers** (see Section 3). Instead of defining the relation of two sets, a cardinality quantifier imposes a constraint on one set and its cardinality. For classical logics, Kimba implements the three cardinality quantifiers AtLeast, AtMost and Exactly using constraint reification. The implementation is shown in Figure 8.12.

```
proc {Exactly Var [Insts Card]}
    {FD.reified.sum Insts '=:' Card Var}
end

proc {AtLeast Var [Insts Card]}
    {FD.reified.sum Insts '>=:' Card Var}
end

proc {AtMost Var [Insts Card]}
    {FD.reified.sum Insts '=<:' Card Var}
end
```

Fig. 8.12. Propagator procedures for cardinality quantifiers

8.4.6 The Translation

The procedure `expand` shown in Figure 8.13 implements the translation from formulas `F` into constraints. The procedure is defined as a method that is inherited to all of Kimba's proof engines (cf. Section 8.5). The main procedure of Kimba simply follows the syntactic structure of an input formula. The input parameters are a formula `F` given in Kimba's syntax, and a root variable `Root` that is the finite-domain integer variable that is associated with the interpretation of `F`. The procedure then splits over the cases that are possible: `F` could be an equality atom, a truth constant, a formula with an unary connective, a formula with a binary connective, a formula dominated by some quantifier, or an atom. In the case of complex formulas, the method `expand` must initiate a further translation of the components and instantiations. In any case, the semantics of the logical constants are all look up in the currently used logic definition structure `self.logic`.

The translation recursively generates a number of propagators by calling the propagator procedures of the logical constants. Some input problems create several thousand concurrent processes that implement the relations between the interpretations of the formulas and their components or instantiations. Propagation is concurrent, so the process of determining variables can partly take place during the translation. An unsatisfiable set of constraints

```
meth expand(F Root)
   case F
   of ['=' A B]                                 %% F is equality
   then {self.logic.constants.'='.2 A B Root}
   elseof H|Rs                                  %% F is application
   then
      if {HasFeature self.logic.constants H} %% F is complex
      then
         HType = self.logic.constants.H.1
         Propagator = self.logic.constants.H.2
      in
         case HType of o then                   %% H is truth const
              Root = Propagator
         elseof [o o]                           %% H is unary
         then C in
            {Propagator C Root}
            {self expand(Rs.1 C)}
         elseof [o o o]                         %% H is binary
         then Ca Cb in
            {Propagator Ca Cb Root}
            {self expand({Nth F 2} Ca)}
            {self expand({Nth F 3} Cb)}
         else                                   %% H is quantifier
            {Propagator                         %% or predicate in LDS
             {Map Rs
               fun {$ I} {self termToDenotation(I $)} end} Root}
         end
      else                                      %% F is complex atom
         {self storeAtomic(F Root)}
      end
   else                                         %% F is symbolic
      {self storeAtomic(F Root)}
   end
end
```

Fig. 8.13. KIMBA's translation from formulas to constraints

can sometimes be detected before any search has been done. After the translation has been finished and the initial propagation has been completed, KIMBA starts the actual search for models where undetermined variables are provisionally restricted further, and the next iteration of propagation starts.

8.5 Proof Engines and Controlling Search

KIMBA is a modular system that has been built in an object oriented way. It consists mainly of a uniform method `expand` for translating the specification using the logic definition structures, and a variety of classes of **proof engines** that implement different algorithms for searching solutions within the search space defined by the constraint satisfaction problems.

8.5.1 Proof Engines

The generic proof engine intuitively prepares the enumeration of models for a given input problem as follows. First, it constructs an initial constraint frame from the problem specification which is a minimal frame of domains for the constant symbols that occur in the problem. This implies for instance that the domain of first-order constants is not empty and contains at least one constant `C1`.

Then, the proof engine applies the the translation method `expand` to all input formulas and constrains all their root variables to the maximum truth value defined by the input's logic. The result is a set of finite-domain integer variables that correspond to the interpretation of the atoms defined by the input and the current constant frame.

In the so-called *First-Order* proof engines, the search space is potentially infinite. If it is not possible to assign to each formula a unique integer value such that the result defines a model, then the input is unsatisfiable over the current constant frame and the first-order proof engine restarts model search by extending the current domain of first-order individuals with a newly generated constant and by building up a new constraint tableau.

The following are the proof engines that are available in the current implementation.

`Propositional`: An engine that leaves out the iterative extension of the first-order part of the constant frame. This engine's main application is the search for models within a given domain of individuals, as is for instance usual in puzzle applications.

`Minimal`: A variant of the `Propositional` engine that rejects models which are not subset-minimal. The engine first searches for propositional models in the usual way and eliminates non-minimal models by proving subset-minimality (see Chapter 4) with the standard Propositional engine.

`First-Order`: The `Propositional` engine with a mechanism for iterative deepening over the universe of discourse. While the engine's results can be controlled by branch-and-bound search, its main application is to enumerate the finite models of an input over an increasing first-order domain.

`Local Minimal`: This first-order engine implements local minimality, i.e., combines the `First-Order` engine and the subset-minimality constraint as implemented by the `Minimal` engine. The main application of this engine have been the analyses of definites as presented in Chapter 5.

`Predicate Minimal`: The minimisation of a certain predicate ab, in KIMBA usually written as `$R`, is an essential part of conservative minimality. The present engine is a sub-engine that only produces ab-minimal solutions for the current translation.

`Conservative`: This is a first-order engine for conservative minimality as presented in Section 6.3.9. It is based on a two stage computation that first determines a *ab*-minimal model using the `Predicate Minimal` engine. The models are enumerated as in the engined `First-Order`, but every model produced must (a) have a minimal *ab*-index and (b) be subset-minimal with respect to all other *ab*-minimal models in the current constant frame.

As a first example, we show the implementation of the class for `Minimal` engines in Figure 8.14. The class inherits from the standard `Propositional` class of engines. It only replaces the top-level proof method `prove`. The `prove` method first constraints the first-order domain size to the size of the current universe of individual constants. By this, it prevents any iterative deepening that is inherited from the generic engine. Next, the method `expandAll` expands all formulas and initiates the first stage of propagation. Next, the **atom weight**, i.e., the number of atoms evaluated to true, is constrained to the sum of the truth values of all atoms in the Herbrand base `bool`. The `FD.distribute` procedure then starts distributing and propagating, controlled by an external search engine. The search process stops when either no model could be found or the set of truth values in `bool` could be determined consistently.

```
class Minimal from Propositional
   meth prove
      self.population =: {Length @universe}
      {self expandAll}
      choice
         %% propagate weights; we ignore ab-index
         {FD.sum @bools '=:' self.weight}
         choice
            {FD.distribute generic(order: nbSusps value: min)
             {List.toTuple vars @bools}}
         end
      end
      %% refuting non-minimal solution
      {RejectNonMinimal self}
   end
end
```

Fig. 8.14. The `Minimal` engine

The difference between the `Minimal` engine and the standard propositional engine is the final call to the procedure `RejectNonMinimal`. This procedure verifies that the currently computed model is a locally minimal one. `RejectNonMinimal` proves that there is no other model within the same domain size that validates the input specification by using a real subset of the

```
class Conservative from PredicateMinimal
   meth computeLowestBound
      proc {SearchAux S}
         S = {New PredicateMinimal init}
         {CopyEngine S self}
         {S prove}
      end
      %% search for ab-minimal, domain-minimal model
      Best = {Search.base.best SearchAux
              proc {$ Ea Eb}
                 Ea.cost >: Eb.cost
                 Ea.population >=: Eb.population
              end}
   in
      %% constrain current ab-index to the best possible
      if Best == nil then fail
      else self.cost =: Best.1.cost end
   end

   meth prove
      choice
         self.population =: {Length @universe}
         {self computeLowestBound} %% ab-minimal model
         {self expandAll}
         choice %% the index are the 'costly' atoms
            {FD.sum @bools  '=:' self.weight}
            {FD.sum @costly '=:' self.cost}
            choice
               {FD.distribute generic(order: nbSusps value: min)
                {List.toTuple vars @bools}}
            end
            {RejectNonMinimal self PredicateMinimal}
         end
      [] %% first-order iterative deepening
         {self addIndividual} {self prove}
      end
   end
end
```

Fig. 8.15. The `Conservative` engine

atoms validated by the current model. This proof problem is a propositional one, and `RejectNonMinimal` simply uses the standard `Propositional` engine for deciding this problem. If the current model is not minimal, then the procedure signals an error and the current model is rejected.

Figure 8.15 shows the first-order engine `Conservative`. It includes the method `computeLowestBound` that implements the first stage of the computation, namely the identification of the minimum of *ab*-predicates that are possible in a models of the current constant frame. The main method `prove` then basically does the same as the `Minimal` engine, except that the search space is opened up to the addition of new individual constants into the current constant frame.

8.5.2 Search

The solutions for a constraint problem can be computed by applying one of Oz's built-in encapsulated search procedures to the constraint problem generated by the proof engine. The search procedures available are for instance depth-first, branch-and-bound, and visual search using the Explorer tool. The Explorer visualises the search spaces as trees; we have used such search trees to visualise the differences in methods for minimal model computation in Section 6.2.12.

The search for models is influenced by the way in which the distribution strategy selects variables for distribution. As a general heuristic guide, Kimba restricts first those variables whose value affects as many other variables and constraints as possible. It also minimises the number of positive literals that are validated in the model at the same time. The process of variable distribution and propagation is repeated until all integer variables have a unique value.

If we use a depth-first left-to-right search strategy, then each first-order engine will always find those models first whose domain of individuals is minimal. Additionally, search can be bounded by the number of literals that are validated. This implements a strong heuristic for minimising the assumptions that are made in a model. By combining this restriction with domain minimality, Kimba can be used for efficiently computing one domain minimal models of the input whose presentation as a finite set of positive literals is as short as possible.

Kimba's standard search is based on a branch-and-bound approach where properties of earlier computed solutions are used as upper bounds for the rest of the search space. For instance, the search can be bounded such that each model computed must not use more individuals than a previously computed model. This bound effectively excludes models that are not domain minimal. The search can be restricted by a combination of the following values:

- the number of atoms that are validated within a model
- the number of individual constants in the universe of discourse
- the *ab*-index of the model

Additionally, the bound can be defined as total, i.e., each model must have a smaller bound than all previous ones, or as partial, where a sequence of models can have the same bounds. Totally bounded search restricts the search more quickly and produces the model with the lowest bound faster. However, the number of models that are produced by the enumeration is also reduced greatly, and we might miss models with interesting properties.

To be more specific, Figure 8.16 shows `OneModel`, the generic best-solution search used in KIMBA. The procedure has three arguments, namely `Problem` that is the input problem's specification, `Engine` which names the proof engine to be used, and `WeightP`, the constraint procedure which specifies the branch-and-bound search. The one-model search simply applies the encapsulated branch-and-bound best-solution search that is built into OZ to a procedure `SearchAux`. This auxiliary procedure simply creates a proof engine using both the problem specification and the weight constraint, and the applies the `prove` method to this new engine. `WeightP` may be for instance the procedure `MinimalAtomWeight` which makes sure that the best model found by `OneModel` is one that has a minimal atom weight—the procedure constrains the weight of each successor solution `Eb` to be smaller than that of an earlier solution `Ea`. While `Search.base.best` explores the search space in a depth-first, left-to-right manner, the bound given by `MinimalAtomWeight` makes sure that each model found has a lower atom weight than all earlier ones. The last model found therefore must be the best according to the given bound.

```
fun {OneModel Problem Engine WeightP}
   proc {SearchAux S}
      S = {New Engine create(Problem WeightP)}
      {S prove} %% that's it folks!
   end
in
   {Search.base.best SearchAux WeightP}
end

proc {MinimalAtomWeight Ea Eb}
  Ea.weight >: Eb.weight
end
```

Fig. 8.16. Search for one model, and a branch-and-bound procedure

8.6 System Performance

KIMBA is a research prototype whose main design goal has been simplicity rather than speed. Nevertheless, it is interesting to compare the system's performance with that of automated reasoning systems that can perform similar tasks. KIMBA's intended application in linguistics has been to determine preferred interpretations with specific minimal model properties, a task where it has no real competitors because there are no systems around that implement the forms of minimality that we found is needed. Nevertheless, KIMBA is also a general finite model generator that can for instance be used as a refutation proof procedure for propositional logics as well. In the following, we will discuss KIMBA's performance and weak points.

8.6.1 Identifying Single Solutions

The table below shows KIMBA's performance on some selected problems from the TPTP's puzzle domain [93]. Times were taken on a Sparc Ultra 1.

Problem	PUZ001-1	PUZ005-1	PUZ017-1	PUZ031-1 (8)
Time	0.3s	4.6s	0.7s	1.6s

The original TPTP specifications are first-order clause sets that have been reformalised for KIMBA using first- and higher-order specifications in classical 2-valued logic. The TPTP formalisations of the logical puzzles are unsatisfiable and can be solved by applying first-order refutation procedures. In contrast to this, the KIMBA formalisation for each puzzle is satisfiable, and each model produced corresponds to a solution.

Logical puzzles, such as the example PUZ017-1 given in full in Appendix A.1, can often be solved easily by model generators that are based on constraint solving such as FINDER [30] or KIMBA. The rules given in a puzzle's logic specification are translated in very effective constraints on the combinatorial possibilities. PUZ017-1 is hard for most theorem provers because its first-order clause representations leads to a large search space. KIMBA's exceptionally good performance can be explained by an elegant higher-order formalisation. For instance, the specification *every job is held by at least one person* in PUZ017-1 has a natural formalisation in KIMBA as follows:

$$\textsc{Every}(\lambda J_{(\iota\to o)}\ \mathsf{job}(J))(\lambda J_{(\iota\to o)}\ \textsc{AtLeast}(J)(1))$$

With this form of quantification, KIMBA keeps entities of different type separately. In a standard first-order form, all quantified entities must be individuals. This usually leads to a larger than necessary domain for which model generation and theorem proving becomes exponentially more complex. A similar kind of improvement is possible in conventional deduction systems when using sorted logics. The problem mentioned above presents no problem for other constraint-based model generators which can make use of a sorted formulation (i.e., FINDER and SEM). FINDER can solve PUZ017-1 in about 0.05s.

Puzzles are typical examples of **single-solution problems** for model generators, i.e., combinatorial problems where one arbitrary model suffices as a result. A class of single-solution problems where finite model generators have traditionally been very successful are quasi-group existence problems. A **quasi-group** is a set with a binary operation $\circ$ which satisfies unique solvability of equations. That is, for all a, b there exist unique c, d such that $c \circ a = b$ and $a \circ d = b$. A quasi-group existence problem [4] is finding an n by n multiplication table over the elements $\{1, \ldots, n\}$ such that the table satisfies the properties of a quasi-group and some additional constraints.

For instance, the "QG5" quasi-group problem is finding an table that satisfies the equation $((b \circ a) \circ b) \circ b = a$ for all elements a and b. Furthermore, the table must also be idempotent, that is, that $a \circ a = a$ for all a. The tables, if they exist, are Latin Squares in that there are no repeated entries in any row or column. Quasi-group existence problems are characterised by very large search spaces with very few solutions.

As interesting open questions about quasi-groups can be answered by finite model generators, many systems support the solution of quasi-groups by providing efficient data-structures that have been tailored especially for this task. For instance, FINDER internally represents a quasi-group as a multiplication table of integers whith a built-in Latin-square property. The current implementation of KIMBA has not been optimised for such applications and does not employ any low-level optimisations. As result, it is heavily outperformed by other systems, and is not able to construct models for any but the most trivial finite (quasi)group problems.

8.6.2 KIMBA as a Propositional Theorem Prover

Cook's pigeon-hole problem, MSC007-2 from the TPTP, is a classical benchmark proof problem for propositional theorem provers. The task is to prove that there can be no way of putting $n+1$ pigeons into n pigeon holes such that each hole contains at most one pigeon. Given a set of $n+1$ pigeons which all are individuals, and a set of n holes which are first-order properties denoting the set of pigeons they contain, the formalisation is as follows.

$$\forall x_\iota \; \exists P_{\iota \to o} \; \textsf{\textit{hole}}(P) \wedge P(x) \tag{8.4}$$

$$\forall P_{\iota \to o} \; \textsf{\textit{hole}}(P) \Rightarrow \textsc{AtMost}(P)(1) \tag{8.5}$$

Formula (8.4) states that every individual, i.e., every pigeon, must have a property P that is a hole, i.e., must be in a hole. Formula (8.5) then formalises that every hole can have at most one member, i.e., there can be at most one individual per hole. Of course, with a fixed set of pigeons and holes, the two formulas actually are only a compact higher-order specification of an unsatisfiable set of propositional formulas. For KIMBA, the task is to prove that the two formulas interpreted over a given set n of holes and $n+1$ pigeons are unsatisfiable. As the `Propositional` engine completely enumerates the finite models over the initial constant frame, it can be used to perform this task.

The following table shows the performance of KIMBA on problem instantiations with four to eight holes. It also gives the number of atoms whose interpretations must be determined, and the number of branching nodes in the search space defined by the problem.

Holes	4	5	6	7	8
Time	< 0.05s	0.18s	1.30s	9.80s	85.00s
Atoms	24	35	48	63	80
Nodes	23	119	719	5039	40319

The size of the search space, as indicated by the branching nodes, grows exponentially with the size of atoms and results in an exponentially larger proof time. The naive way in which KIMBA explores the whole search space is not optimal, which implies that KIMBA in general is not well suited for the task of proving the non-existence of models. In particular, the finite model generator SEM is more than two orders of magnitude faster than KIMBA for the pigeon-hole problems. The reason for this gap in efficiency lies in the fact that KIMBA has no mechanism for suppressing **isomorphic models**. The actual order of pigeons in the holes does not matter, as any permutation of $n + 1$ pigeons cannot be put into n holes. Yet, a different permutation results in a different interpretation, and all of these are investigated and refuted by KIMBA. This is relevant for many practical applications, as the same complexity problem frequently occurs whenever a problem specifications allows the permutation of elements with respect to certain properties.

The suppression of isomorphic models as proposed by Zhang and Zhang [46] probably is the most promising single optimisation that is possible in finite model generation. A model is isomorphic if it is equal to another one modulo a renaming of the constant symbols. The SEM system uses an effective heuristic that avoids the generation of isomorphic models without loosing completeness. However, it is not clear how the approach can be implemented in KIMBA because it relies on properties of clause sets that cannot be tested within a system that does not use clause normalisation. The pigeon-hole problems require an exhaustive search in a very large search space that almost entirely consists of isomorphic interpretations. Because of isomorphic models, KIMBA also is not efficient enough for proving the non-existence of certain quasi-groups. Our translation of deduction into constraint solving does not effectively restrict large search spaces for refutation proof problems.

8.6.3 Generating Minimal Models

The property of being a subset-minimal model can only be decided by a reasoning process that must consider a substantial number of interpretations regardless of what is already known about the current one. Hence, the generation of subset-minimal models is in general one step up the complexity hierarchy in comparison to the generation of arbitrary models [24]. Local minimality and conservative minimality both suffer from this problem. In order to verify the minimality constraint, we must initiate a proof process that traverses a potentially very large search space. In the examples for definites given in Chapter 5, the space of interpretations usually was small and thus the effect hardly noticeable—in all examples, the preferred models can be determined in under a second on a conventional Pentium II 266Mhz PC. The interpretation of reciprocal sentences shows more clearly the relations between the search space of interpretations, the number of conservative minimal models, and the cost of determining a single arbitrary model.

Example	Models	Conservative	First	All
Pirates	26/51	23	0.12s	0.45s
Pitchers	24/121	24	1.66s	12.48s
Measles	148/2305	64	5.93s	41.47s
Like	432/863	1	0.12s	0.36s

The two numbers in the row "Models" give the number of valid interpretations that the logical form has and the number of nodes the search space for the problem has. Thus, for the "Pirates" example from page 83, we have 26 models in a search space with 51 nodes, i.e., branchings. The row "Conservative" gives the number of conservative models, which is 23 for the "Pirates" example. As we can see, conservative minimality has eliminated $26 - 23 - 3$ models which are not conservative minimal. The time for determining the first conservative minimal model is $0.12s$, while the time for enumerating all conservative models is $0.45s$. In the Boston "Pitchers" example (cf. page 99), we have 24 valid interpretations and 121 nodes. All models of the logical form also are conservative minimal. The larger search space results in an increased time for finding the first conservative minimal model—the proof procedure must verify for each model whether it is also locally minimal with respect to all other *ab*-minimal models. While the search space in comparison to the "Pirates" example has not even tripled, the time for determining the first conservative model has increased by an order of magnitude. This is the worst case which may not occur in all cases where the search space of interpretations grows larger. In the "Measles" examples (cf. page 100), the search space as a whole is 19 times larger than in the "Pitchers" example with a similar logical encoding. Yet, the time complexity both for determining a first conservative model and all conservative models is below one order of magnitude. In some cases, a larger space of interpretations may even play no rôle at all. In the "Like" example (cf. page 96) that exemplifies SR reciprocity, we have 432 models in a search space with 863 nodes. However, the logical encoding has only one conservative model whith is also very easy to determine, as it is also the first one that the model generator can find in the whole search space. The *ab*-minimality of this model effectively eliminates the major part of the search space. As it takes only 0.12 seconds to find the model and to identify it as the *ab*-minimal one, the remaining search space can be traversed in about 0.36 seconds, which is even less than in the very simple "Pirates" example. Thus, we can conclude that the minimal model constraints on the interpretations may be able to restrict the search space effectively, even though it has been shown that (subset-)minimal model generation may in cases lead to an exponential worst-case complexity. At the same time, the worst case can be testified for some examples. As KIMBA completely enumerates the conservative minimal models, the effect of the combinatorial explosion caused by a larger universe of discourse is plainly visible. The following tables shows the "Pirates" example with three, four, and five pirates who stare at each other.

Example	Models	Conservative	First	All
Pirates (3)	26/51	23	0.12s	0.45s
Pirates (4)	281/561	81	0.27s	4.65s
Pirates (5)	3494/6993	1024	1.99s	87.00s

As we can see, the number of interpretations, the search space, and the number of conservative models explodes with the size of the reciprocal set. This is a direct result of the way in which we interpret reciprocity, as the linguistically valid interpretations must involve permutations of elements with respect to the reciprocal relation. For practical applications, the increase in time for determining a first model may be especially worrisome, as this can be taken as the instance of a preferred linguistic interpretation. A possible counter-measure for the effect would be the elimination of isomorphic models as mentioned in Section 8.6.2, a task that cannot performed by KIMBA in its current form. On the other hand, the theoretical properties of subset-minimality make it impossible to avoid this effect in general.

9 Conclusion

You have always been a cunning linguist, Bond.
(Moneypenny, *Tomorrow never dies*)

The primary hypothesis of computational logic-based semantics is that logic can be used to capture the meaning of natural language. A logic consist of a formal language, i.e., a syntax, and a semantics that maps the expressions of this language into some domain and finally to truth conditions. To define a suitable logical representation for natural language is a worthwhile pursuit on its own right, as such a representation permits us to investigate natural language on a formal, unambiguous level. Works in theoretical logic-based semantics, such as Dalrymple et al.'s research on reciprocity, use a logical device to formally describe some properties of the semantics of natural language. Without inference, however, there is no *computational* logic-based semantics.

9.1 Why Inference Is Worth the Effort

If logic can in fact capture meaning, then there is all reason to suspect that inferences on the logical form can be used to model the process of natural language understanding. It is conceivable that the way in which humans interpret language is determined to a very large degree by some system of rules, that is, is systematic. Otherwise, we would not even be able to detect when the process of communication completely fails.

The research documented in the present work focuses on inference rather than on representation. The starting point was the observation that certain models of a semantic representation and a context consisting of situational and world knowledge can be taken as the meaning of the represented natural-language sentence. Our early analyses were not based on some of the state-of-the-art semantic representation languages that have been designed especially for such phenomena as pronoun resolution or multi-level ambiguities. Instead, we started out with plain first-order predicate logics because these come with a plethora of automatic reasoning systems for model generation and deduction.

Finite model generation is an interesting research tool in computational semantics as it gives us, for the first time, the ability to enumerate a large class of models of the input. This enumeration often makes explicit some subtle errors in semantic representations. It also make clear that semantic representation without contextual knowledge are near to worthless for the purpose of interpretation. There is almost no example where the models of a logical form alone correspond to interpretations that are acceptable. Logic and inference require world knowledge reasoning as the third aspect of a computational logic-based form of natural-language interpretation. Fortunately, common-sense reasoning can often be performed as part of the process of computing a model, and many faulty interpretations can be turned into correct ones by adding the missing knowledge.

Automated reasoning systems are now at a level of performance that allows their application in the real world. Even a simple system like KIMBA, which is based on a lean approach to inference and that is not primarily built with efficiency in mind, profits greatly from the improvements in constraint-solving technology in recent years. Our higher-order logical language $\mathcal{MQL}$ evolved from experimentation with what is still tractable within the limits of a translation of logic into finite-domain integer constraints. The language $\mathcal{MQL}$ as well as the KIMBA system owes much of its appearance to the fact that we started out with a standard approach of inference based on a classical, well-understood form of logic. One of the essential results of the efforts presented here is that the use of efficient inference techniques for classical logics is at least as much an advantage as the logic's restricted expressivity may appear an annoyance to semanticists. The sacrifices that one must bring when using conventional logics or restricted knowledge representation languages are often paid for by the ease of access and the ability for experimentation that the existing well-designed systems provide.

Nevertheless, it is not surprising that our experiments sometimes reveal that a more expressive logical language is desirable not only for theoretical reasons. Some problems on the computational level can be solved without question only by a better representation. As an example, we refer to the improved predictions that a formalisation of unicity gives for singular definite descriptions. The present work in any case avoids topics such as dynamic anaphoric binding or scope ambiguities, as we believe that the questions raised by these phenomena cannot be answered adequately in the formal framework of classical logics that we have chosen. Still, it can be argued that one of the main purposes of a formal representation should be to allow for a simple and possibly tractable form of inference. While a formal language that, for instance, can represent structural ambiguities of quantifier scoping within the logical form may be theoretically attractive, it will be of little use in practice if the representation itself makes the desired forms of inference impracticable. We therefore conclude that both levels of semantics, representation and inference, should no longer be discussed separately.

9.2 Contributions

The present work argues that model generation is a form of reasoning that can be used to conceptualise some inference processes in natural-language understanding. That is, we may use model generation as a formal model of inference processes whose actual forms remain unaccessible to us.

The international conference *Inference in Computational Semantics*, first held in 1999, showed that there is a growing interest in the computational linguistics community to discuss such forms of inference and the computational tools that approximate them. Inference is one of the keys to natural-language understanding, and a formal method that models even small aspects of the interpretation of language therefore may give us new insights in the way in which language works.

The guinea pigs that we have investigated in detail in our linguistic laboratory are singular definite descriptions and reciprocal sentences. Definites were the starting point for a great deal of research in modern computational semantics. Our approach of interpretation as model generation concentrates on the inference tasks that originate in the problem of resolving singular definites in a logically encoded context. Our experiments show that model generation is able to deal with some of the stumbling stones of other approaches, as model generation embeds reasoning about linguistic knowledge into reasoning about the situational context and the world. Unlike as in deductive approaches, we do not have to give some set of entities that may be candidate referents. Domain-minimal model generation implements, in a very natural way, the preference of anaphoric resolution over bridging over accommodation.

Not all questions in connection with definites can be answered by semantics and model generation alone. Definite descriptions show a surprisingly complex behaviour, and the actual truth conditions of a definite can not easily be determined. With the introduction of the identifying property, our theory is no longer a theory of *computational* semantics as we cannot give a formal theory of how identifying properties can be determined.

In contrast to this, the meaning of reciprocal sentences seems to be governed by one principle of interpretation. This principle, the MMH, can be modelled by a certain form of minimal model generation. Our analysis is able to identify the correct interpretations for simple reciprocal sentences within large sets of model candidates. As far as we know, there are no other approaches that can deal equally well with reciprocals. The semantic analysis on the level of interpretations gives us a "continuum of meaning" in contrast to some idiosyncratic class system of reciprocal semantics. In practice, the model generation approach is a convenient means to experiment with the linguistic theory. Theoretically, our treatment of reciprocity is appealing as it requires only one semantic representation for reciprocal expressions and does not treat them as n-way ambiguous constituents. However, as in the case of definites, the present work is cannot be seen as a thorough analysis of the linguistic phenomenon. For instance, a compositional treatment of the interaction of reciprocal expressions and quantifiers requires a different logical representation than the one that we have given. In any way, the maximisation of logical contribution on the level of the models of the logical form currently has no computational alternative, and we believe that model generation is an adequate formal model for the process of reciprocal interpretation.

The reasoning that humans perform when interpreting everyday language is that of reasoning on small and finite sets. Standard first-order predicate logic has its problems with this kind of reasoning, as many properties in connection with finiteness are not first-order expressible, at least not without further technical machinery in the encoding. We have found finite model generation to be appealing because many first-order "inexpressible" truth conditions of natural-language quantifiers can be expressed as computable constraints over finite models. If we interpret the semantic representations of Montague-style higher-order logical semantics only over finite domains, we obtain an interesting fragment of higher-order logics where standard first-order model generation techniques can be used. This fragment inherits the elegant compositional way in which expressions can be composed and defined in the λ-calculus. The language $\mathcal{MQL}$ can be taken as a first model of a formal language that combines desirable properties from higher-order and first-order logics. KIMBA exemplifies that the introduction of concepts like β-reduction, higher-order definitions, or restricted higher-order quantification does not make inference inherently more complex in the way that full higher-order reasoning often does.

9.3 Models as Meaning

Even with all necessary background knowledge present, finite model generation computes in general many models whose content does not reflect the meaning of the logical form. Human conversation implies several constraints for what is an acceptable interpretation and what is not. Model generation must be extended by methods that implement these conventions. Some handy tools for this are available as results of non-linguistic applications of model generation. First and foremost, the notion of a subset-minimal model is central for eliminating those interpretations which give redundant information that cannot be justified by any evidence at all. Then, for such phenomena as definite descriptions, the minimisation of the universe of discourse is convenient for implementing the preference of anaphor resolution over accommodation. Finally, we have used the minimisation of a certain index predicate to model the maximisation of the logical contribution of the scope relation in reciprocal sentences. The forms of minimisation that we have found useful in natural-language interpretation and analysis highly interact, and we must make sure that the intersection of different minimality constraints does not become too strict to be of any use. As a compromise, we have defined a combination of three minimality constraints called conservative minimality that implements a simultaneous minimisation of individuals, logical assumptions, and index value. The minimisation of the universe can be justified on linguistic grounds, but it is also necessary in order to stay in a computable fragment of model generation. There is in general no way of deciding whether some arbitrary first-order model is subset-minimal, but local minimality is decidable because of the domain closure of local minimal models. Likewise, the minimal index can be computed for the set of domain minimal finite models, but not for the set of all models.

The simple techniques that we have used to characterise preferred readings do obviously not exactly model the complex inference processes of natural-language understanding. There can be no question that what any form of minimality will give us is only a heuristic to identify semantic interpretations. As all heuristics, it may sometimes fail. Minimal model generation frequently gives us models which are not preferred readings in any way, or even models whose content has no correspondence to the meaning of the represented utterances at all. However, the term "minimality" is open to any reasoning process that empirically yields better results, and we hope that further research will discover new and better methods that characterise which models of the logical form are linguistically interesting.

A

Some Example Problems

A.1 The Job Puzzle

We consider the logical puzzle PUZ017-1 from the TPTP problem library for automated reasoning systems [93]. Its natural-language formalisation looks like this:

> *There are four people: Roberta, Thelma, Steve, and Pete. Among them they hold eight different jobs. Each holds exactly two jobs. The jobs are: chef, guard, nurse, telephone operator, police officer (either gender), teacher, actor, and boxer. The job of a nurse is held by a male. The husband of the chef is the telephone operator. Roberta is not a boxer. Pete has no education past the ninth grade. Roberta, the chef, and the police officer went golfing together. Question : Who holds which job?*

The problem's formalisation in KIMBA's syntax is given in Figure A.1. Figure A.2 shows the search space of the problem when using the propositional minimal model engine `Minimal` (see Section 8.5.2). The sixteen diamonds represent the minimal models of the problem, while the 53 squares represent failures caused either by some inconsistency or by non-minimality of interpretations. The minimal models prove that the puzzle does not have only one solution, but in fact sixteen. On a Pentium-II/266 workstation, the exhaustive search for all minimal models of the problem takes about 6 seconds.

A.2 Reciprocals: The Boston Pitchers

Figure A.3 shows a logical encoding for the reciprocal sentence *the Boston pitchers sat alongside each other* for a discourse situation with four pitchers: Tom Bolton (`tb`), Jeff Reardon (`jr`), Larry Anderson (`la`), and Jeff Gray (`jg`).

In the encoding, we use some of KIMBA's built-in predicates for formalising the position relation, for instance the predicate `succ` that corresponds to the successor function on natural numbers. The encoding also includes the definitions `iao`, `price`, and `rcp` for the semantic of reciprocals. The predicate `$R` denotes our index predicate.

```
p(remark: 'Who gets the job.'
  logic: classical
  types: are(o:o i:i n:n)
  constants: are(
                roberta:i thelma:i steve:i pete:i
                chef:[o i] guard:[o i] nurse:[o i] operator:[o i]
                officer:[o i] teacher:[o i] actor:[o i]
                boxer:[o i] male:[o i] educated:[o i]
                husband:[o i i] job:[o [o i]])
  definitions: are(female: lam(x#i ['not' ['male' x]])
                   the: lam(p#[o i]
                            lam(q#[o i]
                                ['and' [exactly p 1]
                                 forall(x#i ['implies' [p x] [q x]])])))
  formulas:
     [
      [job chef] [job guard] [job nurse] [job operator]
      [job officer] [job teacher] [job actor] [job boxer]
      ['not' [job male]] ['not' [job educated]]
      %% gender
      [male steve] [male pete] [female roberta] [female thelma]
      %% every job is held by someone
      [every lam(j#[o i] [job j]) lam(j#[o i] [atLeast j 1])]
      %% everyone has exactly 2 jobs
      forall(x#i [exactly lam(j#[o i] ['and' [job j] [j x]]) 2])
      %% the nurse is male, the chef is female
      [the nurse male] [the chef female]
      %% the chef has a husband who is the operator
      [the chef lam(x#i [unique lam(y#i ['and' [husband y x]
                                          [operator y]])])]
      %% husbands are male
      [every lam(x#i exists(y#i [husband x y])) male]
      %% wifes are female
      [every lam(x#i exists(y#i [husband y x])) female]
      %% there is atmost one husband per wife
      forall(x#i [atMost lam(y#i [husband y x]) 1])
      %% all actors are educated and roberta is not a boxer
      [every actor educated] ['not' [boxer roberta]]
      %% Pete has no education past the ninth grade.
      ['not' [educated pete]]
      %% nurses, teachers, and officers are educated
      [every nurse educated] [every teacher educated]
      [every officer educated]
      %% the chef is neither a police officer nor roberta
      [no chef officer] ['not' [chef roberta]]])
```

Fig. A.1. The puzzle PUZ017-1 from the TPTP library [93]

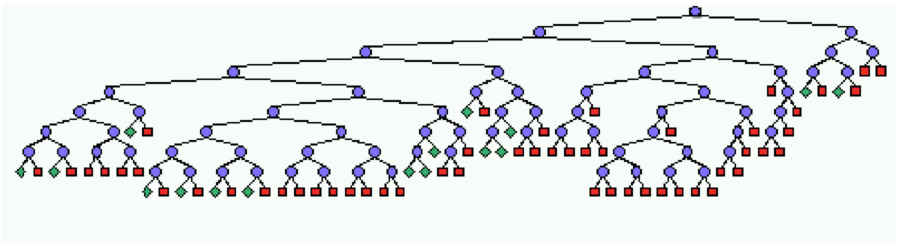

Fig. A.2. The search tree for PUZ017-1 with 16 minimal models.

```
  p(logic: 'classical'  engine: 'conservative'
    constants: are(
                  la:i tb:i jr:i jg:i %% the guys
                  sital:[o i i] %% sitting alongside
                  bospt:[o i]    %% Boston Pitchers
                  pos:[o i n [o i i]]
                  1:n 2:n 3:n 4:n 5:n 6:n 7:n %% some numbers that we might need
                  '$R':[o i i]) %% our index relation
    definitions: are(%% Intermediate Alternative Ordering
                     iao: lam(p#[o i]
                              lam(r#[o i i]
                                  ['and' [atLeast p 2]
                                   [every p
                                    lam(x#i
                                        exists(y#i
                                               ['and' ['and' [p y]
                                                              ['not' ['=' y x]]]
                                                ['or'
                                                 [r x y]
                                                 [r y x]]]))]]))
                     %% The price of not being in the reciprocal
                     price: lam(p#[o i]
                                lam(r#[o i i]
                                    [every p
                                     lam(x#i
                                         ['and' ['not' ['$R' x x]]
                                          [every lam(y#i ['and' [p y]
                                                                ['not' ['=' x y]]])
                                           lam(y#i ['equiv'
                                                    ['not' [r x y]]
                                                    ['$R' x y]])]])]))
                     %% the semantic od reciprocals
                     rcp: lam(p#[o i]
                              lam(r#[o i i] ['and' [iao p r] [price p r]])))
    formulas: [
        %% siting alongside
        forall(x#i forall(y#i ['equiv'
                               [sital x y]
                               exists(p#n
                                      ['and' [pos sital p x]
                                       exists(l#n
                                              ['and'
                                               ['or' ['succ' p l] ['succ' l p]]
                                               [pos sital l y]])])]))
        forall(p#n [atMost lam(x#i [pos sital p x]) 1])
        forall(x#i [atMost lam(p#n [pos sital p x]) 1])
        forall(p#n ['implies' exists(y#i [pos sital p y])
                    forall(l#n ['implies' ['<' l p]
                                exists(x#i [pos sital l x])])])
        %% 4 boston pitchers
        [bospt la] [bospt tb] [bospt jr] [bospt jg]
        %% the Boston pitchers sit alongside each other
        [rcp bospt sital] ])
```

Fig. A.3. The Boston pitchers sat alongside each other

A model generation process yields 24 different conservative minimal models, each of which corresponds to one of the $4! = 24$ different interpretations of *the Boston pitchers sit alongside each other* in a discourse situation where we have four Boston pitchers. In our example, we individual constants for the pitchers *Larry Anderson* (la), *Tom Bolton* (tb), *Jeff Reardon* (jr), and *Jeff Gray* (jg). A typical model appears as follows.

(A.1) $\{$ $\mathit{ab}(jg, jr)$, $\mathit{ab}(jg, tb)$, $\mathit{ab}(jr, jg)$, $\mathit{ab}(jr, tb)$, $\mathit{ab}(la, tb)$, $\mathit{ab}(tb, jg)$, $\mathit{ab}(tb, jr)$, $\mathit{ab}(tb, la)$, $\mathit{bospt}(jg)$, $\mathit{bospt}(jr)$, $\mathit{bospt}(la)$, $\mathit{bospt}(tb)$, $\mathit{pos}(sital, 1, tb)$, $\mathit{pos}(sital, 2, jr)$, $\mathit{pos}(sital, 3, la)$, $\mathit{pos}(sital, 4, jg)$, $\mathit{sital}(tb, jr)$, $\mathit{sital}(jr, tb)$, $\mathit{sital}(jg, la)$, $\mathit{sital}(jr, la)$, $\mathit{sital}(la, jg)$, $\mathit{sital}(la, jr)$ $\}$

The situation described by the model is illustrated by the following table.

position	1	2	3	4
pitcher	Tom Bolton	Jeff Reardon	Larry Anderson	Jeff Gray

All conservative minimal models of the input carry an index of 6, and the conservative minimal models are also identical to the *ab*-minimal models of the input.

References

1. Kamp, H., Reyle, U.: From Discourse to Logic. Kluwer, Dordrecht (1993)
2. Montague, R.: The proper treatment of quantification in ordinary English. In Montague, R., ed.: Formal Philosophy. Selected Papers. Yale University Press, New Haven (1974)
3. McAllester, D., Givan, R.: Natural language syntax and first-order inference. Artificial Intelligence **56** (1992)
4. Slaney, J., Fujita, M., Stickel, M.: Automated reasoning and exhaustive search: Quasigroup existence problems. Computers and Mathematics with Applications **29** (1995) 115–132
5. McCune, W.: Automatic proofs and counterexamples for some ortholattice examples. Information Processing Letters **65** (1998) 285–291
6. Friedrich, G., Nejdl, W.: MOMO—model-based diagnosis for everybody. In: Proceedings 6th IEEE Conference on AI Applications, Santa Barbara, CA, USA (1990) 206–213
7. Baumgartner, P., Fröhlich, P., Furbach, U., Nejdl, W.: Tableaux for diagnosis applications. In Galmiche, D., ed.: Proc. TABLEAUX 97. Number 1071 in LNCS, Pont-a-Mousson, France, Springer (1997) 76–90
8. Kautz, H., Selman, B.: Planning as satisfiability. In: Proc. ECAI 92, Vienna, Austria (1992) 359–363
9. Hilton, A.J.W.: The reconstruction of latin squares with applications to school timetabling and to experimental design. Mathematical Programming Study **13** (1980) 68–77
10. Bry, F., Eisinger, N., Schütz, H., Torge, S.: SIC: Satisfiability checking for integrity constraints. In Fraternali, P., Geske, U., Ruiz, C., Seipel, D., eds.: Proc. 6th International Workshop on Deductive Databases and Logic Programming, DDLP '98. GMD Report 22, Manchester, UK (1998)
11. Jackson, D., Damon, C.: Elements of style: Analyzing a software design feature with a counterexample detector. In: International Symposium on Software Testing and Analysis. (1996)
12. Cunningham, J.: Comprehension by model-building as a basis for an expert system. In: Proc., 5th Technical Conference of the British Society Specialist Group on Expert Systems, University of Warwick (1985)
13. Fujita, M., Hasegawa, R., Shirai, Y., Kumeno, F.: Program synthesis by a model generation theorem prover. Technical report, Institute for New Generation Computer Technology, Tokyo (1991)

14. Fitting, M.: First-Order Logic and Automated Theorem Proving. Springer (1990)
15. Smullyan, R.M.: First-Order Logic. Springer (1968)
16. Beckert, B., Hähnle, R., Schmitt, P.H.: The *even more* liberalized δ-rule in free variable semantic tableaux. In Gottlob, G., Leitsch, A., Mundici, D., eds.: Proceedings of the third Kurt Gödel Colloquium KGC'93, Brno, Czech Republic. Volume 713 of LNCS., Springer (1993) 108–119
17. Giese, M., Ahrendt, W.: Hilbert's epsilon-terms in automated theorem proving. In Murray, N., ed.: Proc. TABLEAUX'99. Number 1617 in LNCS, Saratoga Springs, NY, USA, Springer (1999) 171–185
18. McCarthy, J.: Circumscription – a form of non-monotonic reasoning. Artificial Intelligence **13** (1980) 27–39
19. Nejdl, W., Fröhlich, P.: Minimal model semantics for diagnosis – techniques and first benchmarks. In: Proc., 7th International Workshop on Principles of Diagnosis, Val Morin (1996)
20. Ginsberg, M.L.: A circumscriptive theorem prover. Artificial Intelligence **39** (1989) 209–230
21. Niemelä, I.: Implementing circumscription using a tableau method. In Wahlster, W., ed.: Proc., European Conference on Artificial Intelligence, ECAI 96, Budapest, Hungary (1996) 80–84
22. Bry, F., Yahya, A.: Minimal model generation with positive unit hyper-resolution tableaux. In Miglioli, P., Moscato, U., Mundici, D., Ornaghi, M., eds.: Proc. TABLEAUX'96. Number 1071 in LNCS, Terrasini, Palermo, Italy, Springer (1996) 143–159
23. Schütz, H.: Generating minimal herbrand models step by step. In Murray, N., ed.: Proc. TABLEAUX'99. Number 1617 in LNCS, Saratoga Springs, NY, USA, Springer (1999) 171–185
24. Niemelä, I.: A tableau calculus for minimal model reasoning. In Miglioli, P., Moscato, U., Mundici, D., Ornaghi, M., eds.: Proc. TABLEAUX'96. Number 1071 in LNCS, Terrasini, Palermo, Italy, Springer (1996) 278–294
25. Hintikka, J.: Model minimization – an alternative to circumscription. Journal of Automated Reasoning **4** (1988) 1–13
26. Lorenz, S.: A tableau prover for domain minimization. Journal of Automated Reasoning **13** (1994) 375–390
27. Bry, F., Torge, S.: A deduction method complete for refutation and finite satisfiability. In: Proc. 6th European Workshop on Logics in AI (JELIA). LNAI (1998)
28. de Nivelle, H.: A resolution decision procedure for the guarded fragment. In Kirchner, C., Kirchner, H., eds.: Proc. CADE-15. Number 1421 in LNAI (1998)
29. Manthey, R., Bry, F.: SATCHMO: A theorem prover implemented in Prolog. In: Proc. CADE'88. (1988) 415–434
30. Slaney, J.: FINDER (Finite Domain Enumerator): Notes and guide. Technical Report TR-ARP-1/92, Australian National University Automated Reasoning Project, Canberra (1992)
31. Ernst, M.D., Millstein, T.D., Weld, D.S.: Automatic SAT-compilation of planning problems. In: Proc., of the 15th International Joint Conference on Artificial Intelligence, IJCAI 97, Nagoya, Aichi, Japan (1997) 1169–1176
32. Castell, T., Fargier, H.: Between SAT and CSP: Propositional satisfaction problems and clausal CSPs. In Prade, H., ed.: Proc. 13th European Conference on Artificial Intelligence, Brighton, John Wiley & Sons (1998) 214–218

33. Gallo, G., Urbani, G.: Algorithms for testing the satisfiability of propositional formulae. Journal of Logic Programmming **7** (1989) 45–62
34. Harche, F., Hooker, J.N., Thompson, G.L.: A computational study of satisfiability algorithms for propositional logic. ORSA Journal on Computing **6** (1994) 423–435
35. Jeroslow, R.G., Wang, J.: Solving propositional satisfiability problems. Annals of Mathematics and Artificial Intelligence **1** (1990) 167–187
36. Patel-Schneider, P.F.: A decidable first-order logic for knowledge representation. Journal of Automated Reasoning **6** (1990) 361–388
37. McCune, W.: A Davis-Putnam program and its application to finite first-order model search: Quasigroup existence problems. Technical Memorandum ANL/MCS-TM-194, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL, USA (1994)
38. Beckert, B., Hähnle, R.: Analytic tableaux. In Bibel, W., Schmitt, P., eds.: Automated Deduction: A Basis for Applications. Volume I. Kluwer (1998) 11–41
39. Beckert, B., Posegga, J.: lean$T^A\!P$: Lean tableau-based deduction. Journal of Automated Reasoning **15** (1995) 339–358
40. Klingenbeck, S.: Counter Examples in Semantic Tableaux. Number 156 in DISKI. infix (1996) PhD Thesis.
41. Brüggemann, T., Bry, F., Eisinger, N., Geisler, T., Panne, S., Schütz, H., Torge, S., Yahya, A.: Satchmo: Minimal model generation and compilation (system description). In Imbert, J.L., ed.: Proc., Cinquièmes Journées Francophones de Programmation en Logique et Programmation par Contraintes. Prototypes, JFPLC 96., Clermont-Ferrand (1996) 9–14
42. Davis, M., Logeman, G., Loveland, D.: A machine program for theorem-proving. Communications of the Association for Computing Theory **5** (1962) 394–397
43. Davis, M., Putnam, H.: A computing procedure for quantification theory. Journal of the ACM **7** (1960) 201–215
44. Hasegawa, R.: Model generation theorem provers and their applications. In Sterling, L., ed.: Proceedings of the 12th International Conference on Logic Programming, Cambridge, MA, USA, MIT Press (1995) 7–8
45. Zhang, H., Stickel, M.E.: Implementing the Davis-Putnam algorithm by tries. Technical report, Computer Science Department, The University of Iowa, Iowa City, Iowa, USA (1994)
46. Zhang, J., Zhang, H.: SEM: A system for enumerating models. In: Proc., International Joint Conference on Artificial Intelligence, IJCAI 95. (1995)
47. Zhang, J.: The generation and applications of finite models. Dissertation, Institute of Software, Academia Sinica, Bejiing (1994)
48. Hasegawa, R., Fujita, H., Koshimura, M.: MGTP: A model generation theorem prover – its advanced features and applications. In Baumgartner, P., Hähnle, R., Posegga, J., eds.: Proc., 4th International Workshop on Theorem Proving with Analytic Tableaux and Related Methods, TABLEAUX 95. Number 918 in LNCS, Springer (1995) 1–15
49. Baumgartner, P.: Hyper tableaux — the next generation. In de Swart, H., ed.: Proc., Automated Reasoning with Analytic Tableaux and Related Methods, TABLEAUX 98. Volume 1397 of LNAI., Springer (1998)
50. Baumgartner, P., Furbach, U., Niemelä, I.: Hyper tableaux. In: Logics in AI: European Workshop, JELIA '96. Number 1126 in LNAI (1996)

51. Loveland, D.W., Reed, D.W., Wilson, D.S.: SATCHMORE: SATCHMO with RElevancy. Journal of Automated Reasoning **14** (1995) 325–351
52. Abdennadher, S., Schütz, H.: Model generation with existentially quantified variables and constraints. In Hanus, M., Heering, J., Meinke, K., eds.: Proc., Algebraic and Logic Programming. 6th International Joint Conference, ALP '97 – HOA '97. Number 1298 in LNCS, Springer (1997) 256–272
53. Peltier, N.: Nouvelles Techniques pour la Construction de Modèles finis ou infinis en Déduction Automatique. PhD thesis, Institut National Polytechnique de Grenoble (1997) ftp://ftp.imag.fr/pub/Mediatheque.IMAG/theses/97-Peltier.Nicolas/.
54. Fermüller, C., Leitsch, A., Tammet, T., Zamov, N.: Resolution Methods for the Decision Problem. LNAI 679. Springer (1993)
55. Horacek, H., Konrad, K.: Presenting herband models with linguistically motivated techniques. In: Proc., CIMCA 99. LNCS, forthcoming, Vienna, Austria, *fehlt noch!* (1999)
56. Church, A.: A formulation of the simple theory of types. Journal of Symbolic Logic **5** (1940) 56–68
57. Russell, B.: On denoting. Mind **14** (1905) 479–493
58. Gardent, C., Kohlhase, M., Konrad, K.: Higher-order colored unification: A linguistic application. Technique es science informatiques **18** (1999)
59. Boyer, R., Lusk, E., McCune, W., Overbeek, R., Stickel, M., Wos, L.: Set theory for first-order logic: Clauses for Gödel's axioms. Journal of Automated Reasoning **2** (1986) 287–327
60. Barendregt, H.P.: The Lambda Calculus. North Holland (1984)
61. Farmer, W.M.: A partial-function version of Church's simple theory of types. Journal of Symbolic Logic **55** (1990) 1269–1291
62. Andrews, P.B.: An Introduction to Mathematical Logic and Type Theory: To Truth Through Proof. Academic Press (1986)
63. Henkin, L.: Completeness in the theory of types. Journal of Symbolic Logic **15** (1950) 81–91
64. Benzmüller, C., Kohlhase, M.: Extensional higher order resolution. In Kirchner, C., Kirchner, H., eds.: Proc., 15th International Conference on Automated Deduction (CADE), Lindau, Germany. Number 1421 in Springer LNAI (1998) 56–72
65. Andrews, P.B.: General models descriptions and choice in type theory. Journal of Symbolic Logic **37** (1972) 385–394
66. Kautz, H., Selman, B.: Pushing the envelope: Planning, propositional logic and stochastic search. In: Proc., 13th National Conference on Artificial Intelligence, AAAI 96, Portland, OR, USA (1996)
67. Baumgartner, P., Kühn, M.: Abductive coreference by model construction. In: ICoS-1 Inference in Computational Semantics, Institute for Logic, Language and Computation, University of Amsterdam (1999)
68. Salkin, H., Mathur, K.: Foundations of Integer Programming. North-Holland (1989)
69. Programming Systems Lab Saarbrücken: (1998) Oz Webpage: `http://www.ps.uni-sb.de/oz/`.
70. Lewis, D.: Scorekeeping in a language game. In Bauerle, R., Egli, U., van Stechow, A., eds.: Semantics from different points of view. Springer, Berlin (1979) 172–185

71. Strawson, P.F.: On referring. Mind **59** (1950) 320–344
72. Collins, A., Brown, J., Larkin, K.: Inference in text understanding. In Spiro, R., Bruce, B., Brewer, W., eds.: Theoretical Issues in Language Comprehension. Lawrence Erlbaum Associates, New Jersey (1978)
73. Webber, B.: So what can we talk about now? In Brady, M., Berwick, R., eds.: Computational Models of Discourse. MIT Press, Cambridge MA (1982) 331–371
74. Kamp, H.: A theory of truth and semantic representation. In Groenendijk, J., Janssen, T., Stokhof, M., eds.: Formal Methods in the Study of Language. Mathematisch Centrum Tracts, Amsterdam (1981) 277 – 322
75. Karttunen, L.: Discourse referents. In McCawley, J., ed.: Syntax and Semantics 7. Academic Press, New York (1976) 363–385
76. Carpenter, B.: Type-Logical Semantics. The MIT Press, Cambridge, Massachusetts (1993)
77. Groenendijk, J., Stokhof, M., Veltman, F.: Coreference and modality. In: Handbook of Contemporary semantic theory. Blackwell, Oxford (1995)
78. Heim, I.: The semantics of definites and indefinite noun phrases in English. PhD thesis, University of Massachussetts, Amherst (1982)
79. Asher, N., Wada, H.: A computational account of syntactic, semantic and discourse principles for anaphora resolution. Journal of Semantics **6** (1988) 309–344
80. Haddock, N.: Incremental Semantics and Interactive Syntactic Processing. PhD thesis, University of Edinburgh (1989) advisor: M. Steedman.
81. Cooper, R.: The interpretation of pronouns. In Heny, F., Schnelle, H., eds.: Syntax and semantics 10. Academic Press, New York (1979) 61–92
82. Kadmon, N.: Uniqueness. Linguistics and Philosophy **13** (1990) 273–324
83. Westerstahl, D.: Determiners and context sets. In van Benthem, J., Meulen, A.T., eds.: Generalised quantifiers in natural language. Foris, Dordrecht (1991)
84. Zeevat, H.: Presupposition and accomodation in update semantics. Journal of Semantics **9** (1992) 379–412
85. Dalrymple, M., Kanazawa, M., Kim, Y., Mchombo, S., Peters, S.: Reciprocal expressions and the concept of reciprocity. In Barbosa, P., Fox, D., Hagstrom, P., McGinnis, M., eds.: *Is the Best Good Enough?*: Proc. Workshop on Optimality in Syntax, The MIT Press (1996)
86. Langendoen, D.T.: The logic of reciprocity. Linguistic Inquiry **9** (1978) 177–197
87. Philip, W.: Children who know each other. ms., OTS, Utrech University (1996)
88. Winter, Y.: What does the strongest meaning hypothesis mean? In: Proc., Sixth Semantics and Linguistic Theory Conference. (1996)
89. Blackburn, P., Bos, J., Kohlhase, M., de Nivelle, H.: Theorem proving for natural language understanding. In: Proc., Workshop Problem Solving Methods in Automated Deduction at CADE 98. (1998)
90. Pierce, C.S.: Abduction and induction. In Buchler, J., ed.: Philosophical Writings of Pierce. Dover Books (1955) 150–156
91. Hobbs, J., Stickel, M., Appelt, D., Martin, P.: Interpretation as abduction. Artificial Intelligence **63** (1993) 69–142
92. Benzmüller, C., Kohlhase, M.: LEO, a higher order theorem prover. In Kirchner, C., Kirchner, H., eds.: Proc., 15th Conference on Automated Deduction, CADE 98. Number 1421 in LNAI, Lindau, Germany, Springer (1998) 139–144
93. Suttner, C., Sutcliffe, G.: The TPTP problem library (TPTP v2.2.0). Technical Report 97-04, Department of Computer Science, James Cook University, Townsville, Australia (1998)

Index