Abstracting over Semantic Theories

Alexander Guy Brockway Holt

Doctor of Philosophy
University of Edinburgh
1993

Abstract

The topic of this thesis is abstraction over theories of formal semantics for natural language. It is motivated by the belief that a metatheoretical perspective can contribute both to a better theoretical understanding of semantic theories, and to improved practical mechanisms for developing theories of semantics and combining them with theories of syntax.

The argument for a new way to understand semantic theories rests partly on the present difficulty of accurately comparing and classifying theories, as well as on the desire to easily combine theories that concentrate on different areas of semantics. There is a strong case for encouraging more modularity in the structure of semantic theories, to promote a division of labour, and potentially the development of reusable semantic modules. A more abstract approach to the syntax-semantics interface holds out the hope of further benefits, notably a degree of guaranteed semantic coherence via types or constraints.

Two case studies of semantic abstraction are presented. First, alternative characterizations of intensional abstraction and predication are developed with respect to three different semantic theories, but in a theory-independent fashion. Second, an approach to semantic abstraction recently proposed by Johnson and Kay is analyzed in detail, and the nature of its abstraction described with formal specifications.

Finally, a programme for modular semantic specifications is described, and applied to the area of quantification and anaphora, demonstrating successfully that theory-independent devices can be used to simultaneously abstract across both semantic theories and syntax-semantics interfaces.

Acknowledgements

Most of all, I owe a great debt to my supervisor, Ewan Klein, for his kindness, tolerance, inspiration and continual encouragement.

It has been a privilege to spend the last five years in the Centre for Cognitive Science. For discussions and arguments that have shaped the ideas which follow I thank all those that I have known here, but especially David Adger, David Beaver, Patrick Blackburn, Lawrence Cavedon, Robin Cooper, Martin Emms, Ian Lewin, Suresh Manandhar, Michael Newton, Mike Reape, Sheila Rock, Catrin Siân Rhys, Peter Ruhrberg and Henk Zeevat. Beyond Edinburgh I would like to thank Mark Ryan, and to record the influence of George Bealer's inspiring lectures at Leuven in 1990.

I am grateful to everyone who has worked to make CCS such a good place to be, in particular Robert Dale, Elisabet Engdahl, Betty Hughes, Ewan Klein and Jeremy Olsen. To Jeremy and the rest of the computing team—thanks for an environment that I have relied on time and time again.

This thesis would have been impossible without financial support from the UK Science and Engineering Research Council, for which I am most grateful. It would have been hard to complete it without the occasional work kindly offered to me by Robert Dale and Jeremy Olsen.

To my family and to my friends, for all your support—thank you very much.

Finally, I thank Chris Mellish and Henk Zeevat, my examiners, for their constructive suggestions.

Contents

1 Introduction 1

1.1 Overview . . . 1
1.2 Formal semantics . . . 2
1.3 Entailment . . . 3
1.4 Summary . . . 5
1.5 Synopsis of chapters . . . 6

2 The Case for Semantic Abstraction 7

2.1 Arguments . . . 7
2.1.1 Comparison and classification . . . 7
2.1.2 Modularity and coherence . . . 8
2.2 Approaches . . . 11
2.3 Forms of abstraction . . . 13
2.4 Summary . . . 15

3 Case Study 1: IL, PT$_2$ and LQC 17

3.1 Intensional entities in semantics . . . 17
3.2 Comparing theories . . . 18
3.3 Formal specifications . . . 19
3.4 Semantic theories versus intensional logics . . . 20
3.5 Intensional identity . . . 21
3.5.1 Abstraction . . . 21
3.5.2 Propositional identity . . . 22
3.5.3 Predication . . . 23
3.6 Russell's property . . . 25
3.7 Summary . . . 26

4 Case Study 2: Johnson and Kay's 'Semantic Abstraction and Anaphora' 28

4.1 Introduction . . . 28
4.2 Grammar . . . 29
4.2.1 Semantic values . . . 30
4.3 Semantic primitives . . . 32
4.3.1 Sorts . . . 32

4.3.2 Interpretations 33
4.4 Implementations 36
4.4.1 'Predicate-logic' 36
4.4.2 'Discourse-representation' 37
4.4.3 'Sets-of-infons' 39
4.5 Semantic abstraction 40
4.5.1 Content and behaviour 41
4.5.2 Constraining inferential behaviour 43
4.6 Specifying correct implementations 45
4.6.1 Discourse representation languages 45
4.6.2 An external specification 47
4.6.3 An internal specification 49
4.7 An anaphoric specification 52
4.7.1 Using external representations 53
4.7.2 An internal approach 53
4.8 Summary 55

5 Towards a Specification of Quantification and Anaphora 57
5.1 Abstraction and modularity 57
5.2 Strategies 58
5.3 Anaphora: resolution and interpretation 58
5.4 Bound and unbound anaphors 59
5.5 Constraints on anaphoric interpretation: scope 62
5.6 Formalizing anaphoric dependency 66
5.7 Formalizing quantifier scope 68
5.7.1 Scope ambiguity 68
5.7.2 Uniform presentation of scope relations 69
5.7.3 Connecting quantifier expressions with quantifiers 70
5.7.4 Connecting constituents with scope domains 70
5.8 Specifications for other areas 72
5.8.1 Anaphora and binding theory 72
5.9 Summary 73

6 Conclusions and Prospects 75
6.1 Review 75
6.2 Future work 76

A Implementing the Russell's property specification 78

B Implementing the Johnson and Kay specifications 86
B.1 The inferential map U_0 and the function D 86
B.2 The anaphoric maps A and P 89

References 91

Chapter 1

Introduction

1.1 Overview

This thesis is about formal semantics for natural language. The activity of formal semantics normally involves making precise statements about the meaning of linguistic expressions. To that extent, I do no formal semantics in what follows: there are no claims that such-and-such a sentence must be analyzed in this way, and must have that meaning, which in turn must be formalized just so. Instead, this thesis is about the theories that do make such claims: its propositions are metatheoretical.

The belief which motivates the following chapters is that there are interesting things to say in formal semantics which can only be said, or can best be said, by taking a viewpoint outside any particular theory. So this work is not committed to any one of the many formal theories of natural language semantics currently on offer. What it is committed to is formal theorizing about linguistic meaning in the tradition that stems from Frege, Russell and Montague, and it places particular emphasis on certain aspects of this tradition, as I shall shortly describe.

From my metatheoretical perspective, I do three things. First, I describe formally how two core semantic notions vary across three representative object theories. Second, I analyze one particular semantic framework that claims to embody some degree of semantic abstraction, evaluating those claims by providing a formal specification for the key notions of the framework. Finally I demonstrate a modular approach that can be used to state constraints on and generalizations about natural language semantics that are truly independent of any particular semantic theory.

In the remainder of this chapter I explain what I mean by ‘formal semantics’, clarify my use of some common terms in the field, and indicate precisely which aspects of formal semantics are addressed by this work.

1.2 Formal semantics

A formalized theory is one couched in terms that rely for their meaning only on their syntactic relations to other terms, grounded by expressions that have a common logical or mathematical sense. Reasoning in such theories may, in principle, be checked fully automatically. I call a theory **formal** when it is formalized in this sense, or is generally held to be capable of being so formalized.

Until Frege's breakthrough at the end of the nineteenth century, formal logic was unable to represent the deductive content of all valid arguments. It is now a commonly-held view that all correct mathematical proofs are ultimately capable of being shown valid through representation in formal logic. Frege and Russell initiated the application of modern logic to meaning and reasoning in natural language, and thereby laid the foundations of formal semantics.

From the beginning of this century until the late 1960s, however, the use of logic to analyze natural language meaning operated in a fashion that discussed the grammatical structure of language largely informally, and which did not attempt to model grammar with the same precision that characterized the logical representation of linguistic meaning. The pioneering work of Noam Chomsky in the 1950s demonstrated that natural language grammar was capable of formalization to a far greater degree than had been realized, but it was Richard Montague who first synthesized formal models of grammar and meaning into a unitary theory of natural language semantics. The last version of this theory, described in Montague 1973, (henceforth, PTQ) includes a formal grammar of a fragment of English (based on the work of Ajdukiewicz rather than Chomsky, as it happens) for which a translation to Montague's intensional logic IL is precisely defined, so that the translation process is capable of automation. Formulas of IL are given a model-theoretic interpretation, whence by putting the two processes together every sentence of English admitted by Montague's grammar is automatically given a formal meaning. The contribution of Montague was not only a unified formal treatment of syntax and semantics, but also, through his intensional logic, a semantic domain rich enough to do justice to the fragment of English which he formalized. In the twenty years since the publication of PTQ, it has come to be regarded as the paradigm—though far from an unquestioned one—of formal semantics, in the sense used throughout this work.

Since this sense encompasses the requirement that there be a formal map from expressions of the natural language fragment to the domain of meanings, then strictly speaking, there can be no formal semantics for any natural language: a formal map is a function from one formally defined domain to another one, and thus the domain of linguistic expressions must in fact be a formal language. This is almost a trivial observation—the point is simply that any formal grammar of a natural language is really the grammar of a formal language that resembles the natural language to a certain degree.

Some terminology: when I talk about **syntax** or **grammar**, I will generally mean a formal grammar of natural language, such as GPSG's account of English (Gazdar et al. 1985), and the rules and representations that comprise it. Where that grammar contains semantic representations (as GPSG does, for instance) I exclude those from consideration when I use phrases such as "present in the syntax". Sometimes I may have no particular grammatical theory in mind, intending my comments to apply to an arbitrary one. Where a theory treats **discourse** consisting of a string of sentences as well as sentences in isolation, I consider the term **syntactic theory** to embrace both sentence syntax and any mechanisms required by the discourse-related aspects of the theory. Occasionally I will talk about the syntax of non-natural languages. When that happens I will generally qualify the sense appropriately, as in "logical syntax". I use the term **semantic theory** to mean a system, like PTQ, that can generate formal meanings from natural language expressions, and that is predominantly concerned with doing that, rather than (as GPSG is) with accurately and revealingly characterizing all and only the grammatical expressions. So I say that PTQ does contain a syntactic theory, but that it is a minor part of the overall system, whereas GPSG is chiefly a theory of syntax, but also contains a semantic theory (in fact, one largely based on PTQ). Thus I do *not* mean by 'semantic theory' a hypothesis about natural language meanings, such as the claim that generalized quantifiers in natural language have the property of conservativity, though it is certainly the case that semantic theories, in my sense, may incorporate hypotheses of that kind. Finally, sometimes I will talk about the **object theory**, to distinguish the semantic theory under consideration (possibly an arbitrary one) from my own theorizing about it.

1.3 Entailment

People expect different things from formal semantics. Let a **logical language** be a formal language somehow equipped with a relation on its formulas that is intended to appeal to our intuition of logical consequence—an entailment relation. As part of their assignment of formal meaning to linguistic expressions, nearly all semantic theories translate natural language sentences (or discourses) into formulas of logical languages, at least at some juncture. The significance of this translation is often disputed. A logical language, by the definition just given, guarantees that a notion of entailment exists for its formulas. This means that any system of translation from natural language into it automatically confers an entailment relation on those natural language sentences that are thus translated (appropriately disambiguated). I maintain, therefore, that the modelling of natural language entailments is a core attribute of a semantic theory—if it can't do that, it's not clear that it is doing formal semantics. It certainly isn't doing it in the sense that I adopt throughout this work.

But many people, while conceding this, think that it is reasonable to expect rather more of a semantic theory. One of the most natural things to want is some insight into the *meaning* of a linguistic expression. A formula of a logical language is not a meaning. Further, one of the characteristic attributes of competent speakers of a natural language is their ability to make true statements about the world. It is not immediately obvious how translating their statements into a language with a formal notion of consequence explains this ability.

The most successful attempt to satisfy these expectations within formal semantics has been an appeal to the **model-theoretic** aspect of many semantic theories, including PTQ. From the perspective of formal logic, if a logical language has a model theory then it has at least (i) a canonical way to define its consequence relation; (ii) the chance of proving completeness for a syntactic—that is, proof-theoretic—presentation of that relation, and (iii) a consistency proof, relative to the consistency of the model-theoretic apparatus itself (which can usually be embedded in any formal set theory).

But from the point of view of natural language semantics, a model theory provides—as a minimum—a way to determine **truth conditions** for every formula of the logical language, and again, by extrapolation, to do the same for the expressions of natural language that are translated into it. I'll indicate briefly how truth conditions work. Given a sentence

(1.1) Mary calls John

and a logical translation, say

$$\texttt{call}(m, j)$$

then if our theory is sufficiently broad and the language that $\texttt{call}(m, j)$ occurs in is sufficiently expressive we can expect it to tell us that asserting (1.1) logically entails sentences such as

Mary calls someone

John is called by Mary

and so on. It will also tell us which sentences in their turn entail (1.1). And that's it, unless there is more to the semantic theory than just entailment. But if the theory has a model-theoretic basis, then we can expect it to tell us (assuming a first-order model) that given a certain state of affairs—any state of affairs where *Mary*, *calls* and *John* mean something—then (1.1) is true if, and only if

> *Mary* and *John* mean two things that stand in exactly the two-place relation on things that the word *calls* means.

This at any rate does some justice to our intuitions about the meanings of words—and the truth conditions for more complex sentences rapidly become less obvious! (Common logical languages, like first-order predicate logic, have standard and established model theories. Consequently I will sometimes loosely refer to first-order formulas as 'truth conditions', taking their standard interpretation for granted.)

It should be apparent that one of the properties of model-theoretic semantics is that it proposes models of structures or entities that linguistic expressions seemingly refer to. In the example, the transitive verb *call* is taken to stand for a two-place relation on things. It is often argued that this feature of model-theoretic semantic theories means that they are able to offer more insight into the nature of linguistic meaning than a theory that talks only of entailments.

This is not the place to carry on that argument, but I hope it is clear that by restricting my interest to entailment—a kind of lowest common denominator among semantic theories—I am choosing to bypass at least one important issue about what makes one semantic theory better than another. There are many others. We may ask, for example, how directly the semantics of an expression follows from its linguistic structure, whether the semantic theory lends any support to apparent linguistic universals, whether it is practical to use the theory to actually compute 'meanings', and even whether the consequence relation for the entities thus computed admits of a computationally tractable interpretation, so that reasoning with them may plausibly be automated. By and large, the work reported here addresses none of these questions.

Finally, I want to emphasize that I am well aware of the many semantic theories which generate semantic structures—I'll generally say **semantic representations**—that cannot simply be viewed as formulas of a logical language. The system examined in chapter 4 is one such. But as I demonstrate there, that does not prevent me extracting from these structures a kind of content that can be so viewed, thereby enabling comparison between these theories and the ones for which *logical* forms are indeed the be-all and end-all of formal semantics.

1.4 Summary

The conception of formal semantics for natural language with which this work is concerned sits squarely in the logic-based tradition that derives from Frege, Russell and Montague. I am interested in formal theories which assign semantic representations to fragments of natural languages, and in analyzing these theories I am chiefly concerned with the entailment relations that may be defined over these representations. It is through such relations on representations that a theory makes empirical claims about entailments between sentences of natural language. While many, perhaps most, semantic theories are justified more broadly than this, often with reference to model-theoretic interpretations, I adopt a narrow entailment-based perspective here because of its universality and the core position of entailment in the formal semantics enterprise.

1.5 Synopsis of chapters

The next chapter argues the general case for abstracting over semantic theories, and considers various ways to go about this, some of which correspond to existing work. Chapter 3 looks at how three different semantic theories choose to express the concepts of intensional abstraction and predication, illustrating how such comparisons may be made formally precise. The following chapter is a second case study, but of a single theoretical framework. The authors of this framework claim that it provides a certain level of abstraction over semantic theories, and that by doing so it is able to describe a variety of them in its own metatheoretical terms. I examine these claims critically and attempt to formally circumscribe the exact scope of their abstraction. In chapter 5 I outline the goal of modular specifications for semantics, applicable across all semantic theories, and I demonstrate in detail how this approach can go some way towards a specification of some constraints on quantification and anaphora. What has been achieved is summarized in chapter 6, where I also consider the future prospects for this approach which this thesis presents.

Finally, I should like to apologize for a parochialism of this work: my topic is the formal semantics of natural language, of which latter there is more than one, yet all of my linguistic examples are from English, and all of my theoretical examples are semantic theories applied to English. It is far from evident that cross-linguistic differences have no significance for formal semantics. To the extent that they do, then, I hope that they will tend to strengthen rather than weaken the theoretical approach I argue for here.

Chapter 2

The Case for Semantic Abstraction

2.1 Arguments

2.1.1 Comparison and classification

Since PTQ, alternatives have been proposed for almost every aspect of Montague's system. These include complete replacement semantic theories for comparable fragments of English, as well as proposals that modify only part of PTQ's apparatus. Some of the latter make sufficiently small changes that it is relatively easy to see what the implications of the changes are for the overall theory. But in general, the variety of semantic theories currently available—many only distantly related to PTQ—can make it hard to determine exactly where they differ.

For example, when comparing two theories that address similar semantic issues one would often like to know precisely which syntactic constructions receive different truth conditions under the competing analyses. Yet this can be difficult to work out, particularly if the two theories employ incompatible formalisms. Another common problem is dealing with the interaction of theories that focus on different phenomena. Suppose that we have two theories, A and B, where A has a detailed treatment of quantification and anaphora, while theory B is primarily concerned with the semantics of plurals and mass terms. We might well want to find a way to unify these theories, so as to get a satisfactory treatment of, for example, plural anaphora. But in such cases it is notoriously hard to find out all the respects in which the two theories overlap, or make incompatible assumptions about the semantic terrain. I suspect that faced with these difficulties, researchers have often decided it would be easier to start a new theory from scratch.

So one of the chief motivations for semantic abstraction is the desire to accurately compare and classify semantic theories. The next chapter demonstrates how this can work, by creating formal specifications for the treatment of intensional abstraction and predication. Given an arbitrary semantic theory, one can then attempt to apply these specifications: some will hold, some won't, and the resulting pattern serves to locate that theory in the theoretical landscape. It also provides concrete evidence of the actual relations and operations that the object theory defines, information that is potentially of relevance to the problem mentioned at the end of the previous paragraph.

One of the hopes held out by this approach is that it might become possible to compile a library of tests, or benchmarks, of semantic behaviour. As well as enshrining consensus views on the semantics of, say, restrictive relative clauses, such benchmarks would help to map out areas of continuing dispute by providing a range of possible behaviours. This is exactly what happens for abstraction and predication in chapter 3.

As I tried to indicate in my introduction, I regard the modelling of entailment as a core criterion for formal semantics, so I have devoted most of my energy to describing the results of semantic theories in those terms. But the techniques of semantic abstraction are certainly not limited to that kind of description. It might well be productive to consider classifications based on computational criteria, for instance.

2.1.2 Modularity and coherence

Contemporary theories of grammar integrate syntax and semantics to varying extents. At one extreme, GB theory (Chomsky 1981), still strongly influenced by the thesis of the autonomy of syntax, places the actual construction of semantic forms firmly outside the domain of syntax proper. Although one of GB's levels of representation is called LF, after 'logical form', the only major difference between the forms of LF and those of S-structure—which is one of the two core syntactic representations—is that quantifiers move at LF (that is, transformations result in quantifiers moving within LF representations) in order to capture quantifier scope ambiguities.[1] (A consequence is that in GB scope ambiguity is unequivocally a matter of *syntactic* ambiguity, since LF is a syntactic representation.) Chomsky has always made it clear that semantic interpretation is something that happens *to* the forms of LF, rather than within LF, and for him the rules that constrain that interpretation are outside the scope of grammar proper.

The theories of categorial grammar and HPSG demonstrate positions on the integration of syntax and semantics that lie at the other extreme, though each in a different way. Categorial grammar (seen as a collection of related theories; see Wood 1993 for an introduction) is notable for insisting on a tight correspondence between the categories of syntactic expression and the kinds of semantic entity. Standardly, the combination of syntactic categories is directly mirrored by the combination of functional types in the semantics. HPSG (Pollard and Sag 1987,

[1]Other forms of activity at LF have been suggested, particularly for languages other than English, but quantifier raising remains the primary motivation. I ignore Chomsky's most recent theoretical proposals (Chomsky 1992), under which LF acquires much more significance.

1993), on the other hand, connects syntax with semantics rather differently, by taking the prime unit of linguistic description to be a conglomerate of syntactic, morpho-phonological and semantic features called a sign. Every linguistic expression is represented by a sign and grammatical principles are stated as constraints on signs. Among other things, this allows syntactic and semantic information to be interdependent.

Evidently, then, categorial grammar and HPSG are theories which intertwine syntax and semantics. But even in GB, semantics is not as cleanly separated from syntax as one is led to believe. For instance, quantifier scope is a phenomenon rooted in semantics, yet as just noted treated entirely with the syntax in GB; the notation of coindexing (discussed further in §5.4) carries semantic import, and there are other areas where distinctions that affect semantic interpretation are regarded as purely syntactic features. So a blurred boundary between syntax and semantics seems fairly characteristic of contemporary grammatical theories.

Is there anything wrong with this state of affairs? After all, it is supposed to be to HPSG's positive advantage that it allows syntactic and semantic constraints to be integrated so easily. There certainly are benefits to be had in linguistic description from such integration. But there are also reasons to doubt that we've yet found quite the right way to do it. Here are three:

1. *Monolithic structure*. The core unification-based theory of HPSG (excluding the components of constituent order and lexical inheritance) is monolithic in the sense that any one of its principles may affect any other one. It is precisely because they do affect each other that it is a powerful theory. Just the same holds of GB—the often-heard claim that GB is modular is true to the extent that the various components of GB (X-bar theory, Case theory, and so on) are stated in isolation from each other, and have theoretical notions that are peculiar to them, but false to the extent that the 'modules' of GB are functionally insulated from each other. They are not: the representations of D-structure, S-structure and the others are *global*—any rule or principle of GB can operate with impunity on these representations at any time.

   In both cases this makes for a powerful, expressive theory. But it also makes it very hard to design and to debug, borrowing a term from software development. Because there is little true functional modularity, it inhibits the division of labour in grammar construction. For a significant natural language grammar, it is tempting to be able to let syntacticians hone the grammar rules, or lexical entries, while the semanticists work out how to implement the necessary semantic operations. Making that work demands that the interface between the two is well-specified, which brings us to the next problem.

2. *Limited type-based security*. Unusually among syntactic theories, and admirably, HPSG imposes a comprehensive type system on its main class of theoretical entities, the signs. This ensures that syntactic attributes can

only possess values that 'make sense', to a certain degree. But this type system has limitations, particularly in the area of syntax-semantics integration. The semantic parts of signs are included in the type system, but the types effectively serve only to ensure that the resulting semantic representations obey the logical syntax of the underlying semantic formalism, in the same way that the syntactic types of 'term' and 'formula' in the predicate calculus operate. Owing to the absence of any truly *semantic* types, what is effectively unconstrained is the construction of semantic values by rules and principles of the grammar—there is nothing in HPSG's type system to cast suspicion on a semantics principle that, say, uniformly assigns the same logically true proposition to every sentence. The risk of this kind of semantic incoherence is a consequence of a theory that allows grammar rules to manipulate semantic values as they please.

3. *Inappropriateness of single formalism.* Another of the elegant aspects of HPSG is the fact that its theory is predominantly couched in a single formalism, whose logical foundations are now quite thoroughly investigated, and which comes with an associated model of computation. But again, when we come to the semantics, we find that the formalism is not quite up to the job. This was well-illustrated in the first HPSG book (Pollard and Sag 1987), by the presence of the functions 'successively-combine-semantics' and 'collect-indices' which formed an essential part of the semantics principle, but could not be completely defined in the underlying unification-based formalism. A system that works well for the syntactic constraints of HPSG appears too limiting for those of the semantics: the difficulty, at least in part, appears to be that the kind of processing required by the semantics is more complex, or at least not of the same kind, as that which is sufficient for the syntax.

I believe that the introduction of an appropriate level of abstraction between syntax and semantics is the best way to solve these problems. Taking a cue from software engineering, we should try to develop truly modular grammars, with an especial concern to lay down clearly the interface between syntax and semantics. In the domain of syntax, efforts have already been made to tackle the problems I alluded to above: Newton (1992) has shown how algebraic specification techniques developed in computer science may be applied to the modular construction of syntactic theories.

The prospect of successfully modularizing semantic theories opens the door to applications such as reusability and incremental development. It might become possible to use a well-tested module for, say, quantification and anaphora, in a variety of natural language grammars. Or logical research might turn up a promising new approach to the logical and semantical paradoxes—if an interface to the relevant semantic notions, such as abstraction and predication, has been appropriately defined one could then simply 'plug in' the new semantic module without disturbing the rest of the grammar.

At this point, a doubter might well cite the formalism of PATR-II (Shieber et al. 1983), particularly in response to the final point above. One could argue that the considerable success of PATR-II in formalizing a wide range of linguistic theories in the 1980s suggests that a unitary framework for syntax and semantics is worth striving for. I contend that the case of PATR-II is special in a number of respects. First, and most importantly, PATR-II primarily addressed *syntactic* theory. The unification-based formalisms of PATR-II, HPSG, and other related theories do indeed seem to be well-suited to expressing the kind of operations required by theories of grammar (though note that PATR-II cannot unaided describe every syntactic theory). Second, PATR-II demonstrated that a grammar formalism with a built-in computational interpretation can still be declarative. This allowed grammar development to share a number of benefits, such as fast prototyping, already enjoyed by logic programming. In other words, PATR-II was both expressive enough to capture a large chunk of syntactic processing, and restrictive enough to admit a transparent computational interpretation. Third, PATR-II built on a growing sense among computational linguists that unification-based approaches were fruitful, crystallizing a convergence that in some respects had already started.

In the more general area of semantics, it seems that these conditions can't be met. Semantic processing is inherently more complex than syntactic processing, and owing to its connections with reasoning and knowledge representation, is much less self-contained than the domain of syntax. Certain areas—such as anaphora resolution—are notoriously ill-constrained, it being trivial to construct examples where carrying out semantic interpretation requires reasoning of considerable complexity over past discourse content. Nor, by contrast with computational syntax, do we appear to be in a position to cite a convergence of research work along lines that might lead to a unitary framework. These, then, are the arguments for semantic abstraction.

2.2 Approaches

There are a number of ways that we could go about finding a more abstract description of semantic theories and of the connection between syntax and semantics. If our prime aim is to abstract over semantic theories, it would be undesirable to discover that the result is an account that is somehow tied to a fixed syntactic theory; whatever there is that is true, or uniform, across all varieties of semantics cannot be dependent on a particular way of doing syntax. This is not to say that in abstracting over semantic theories one should necessarily ignore syntax—the connection between syntax and semantics cannot be left out of the reckoning if we are to satisfactorily address all of the issues raised above. But whether one actually abstracts over this connection depends on the precise approach taken.

In what follows, for a given theory of formal semantics let the "syntax" be the formal grammar and let the "semantics" be the realm of semantic representations, along with whatever operations and relations are defined on it. The "syntax-semantics interface" is the mechanism the theory utilizes to connect the two. I use the term "interface" loosely—in particular there's no requirement that it have the highly constrained Montagovian sense of a function from disambiguated syntax to semantics, and it is really the interface *to the syntax* that is under scrutiny. I argued that any approach to semantic abstraction should apply across a range of syntactic theories. Then one way to classify approaches to semantic abstraction is according to which of the interface and the semantics they leave fixed, and which they allow to vary. This gives rise to four categories:

1. *Fixed interface, fixed semantics.* If both are fixed, then there is no opportunity for semantic variability. All syntactic theories are forced to adopt the same interface to the semantics, which in turn is unchanging. So this category in fact corresponds to no abstraction at all, but rather to a claim that just one semantic interface and underlying formalism is sufficient.

2. *Fixed interface, variable semantics.* The approach of Johnson and Kay (1990) falls into this category. They propose a small set of 'semantic operators', which constitutes a fixed interface for every syntactic theory that wishes to make use of it. The operators, however, are intended to be general enough that alternative implementations of them can be used to model a range of different semantic theories. I analyze Johnson and Kay's theory in chapter 4.

3. *Variable interface, fixed semantics.* On this view the underlying semantic theory is fixed, but alternative syntactic theories are catered for by providing an interface layer for each theory. Ongoing work by Sebastian Millies and Manfred Pinkal is apparently of this kind, with a separate interface defined for each syntactic theory that generates a different variety of syntactic structure.

4. *Variable interface, variable semantics.* If both interface and semantics are allowed to vary freely, then this approach is without content unless some sense is made of the variation. One way to obtain content is to claim that every semantics can be reinterpreted in a single semantic 'metatheory'. Such a theory would obviously have to possess a very rich and expressive semantic vocabulary. Recent work by Robin Cooper on re-expressing discourse representation theory (DRT) within situation theory falls partly into this category.

   A metatheory approach that treats only the semantic representations themselves, however, will in general be blind to the syntax-semantics interface, and will therefore be unable to state generalizations which need to take account of the interface. So remaining within this category, another possibility is to search for descriptions that simultaneously abstract over both interface and semantics. Conceivably one might extend the single metatheory approach to to incorporate a single 'meta-interface'. But it is not necessary to advocate just *one* metatheory or meta-interface in order to belong to this category, as the next section's discussion makes clear.

Approaches which fall into the final category are evidently maximally general, in that they are capable of making statements across both interfaces and semantic domains. It is this kind of semantic abstraction that I choose to advocate here. But I have not yet been precise about the form of abstract statement I envisage making, and it's time to address that issue.

2.3 Forms of abstraction

I am interested in making metatheoretical statements that abstract across a number of object semantic theories. These object theories posit entities, relations on them and operations over them, and my abstract metatheoretical statements will necessarily also be in terms of entities, relations and operations—in general, different ones. How can I arrange for my metatheoretical statements to say something about the object theories? Somehow, I must relate the terms of my metatheoretical statements to the terms of each object theory.

Essentially there are two possibilities. Either the metatheoretical statements are interpreted in the terms of each object theory, or vice versa: the statements of the object theory are interpreted in the terms of the metatheory. At the level of logical theories, this is simply a matter of the direction of embedding. On the first option, the metatheory is embedded in the object one, on the second the direction is reversed. A related perspective comes from the field of formal specification of programs, concerned with manipulating precise specifications of how a program program should behave, and with proving that a program satisfies a given specification. Here, it is often the case that a program is said to satisfy—or implement—a specification just in case the specification can be logically embedded in the formal theory of the program. Thus on this view, our first option corresponds to the object theory implementing the metatheory, and the second to the metatheory implementing the object theory.[2]

Let us consider how these options apply to abstracting over semantic theories. Option 1 sees metatheoretical statements as constituting an abstract specification

[2]This analogy is only partial, owing to the complexity of the field of formal specification. First, it is possible to take the 'meaning' of a specification not to be a logical theory but rather a class of algebras (the algebras—or models—that satisfy it, where a program is modelled as an algebra). Given this approach, the logical framework that a specification is couched in cannot always express every relevant property of classes of algebras, whence logical embedding is not sufficient to characterize implementation. Second, it may well be that a central requirement is the *behavioural equivalence* of the programs that satisfy a given specification. Since models of specifications are not in general closed under behavioural equivalence, logical embedding is also insufficient in this respect. (See Sannella and Tarlecki 1992, pp. 1–5, for an introduction to these issues. Note also that Newton 1992, cited earlier, adopts this model-based style of formal specification when characterizing syntactic theories.)

which every semantic theory should be able to implement by interpreting the specification in its own terms, with the interpreted metatheoretical statements holding in that object theory. This requires that every entity and relation posited by the metatheory is somehow present in each object theory. Note that for this to be true it is not necessary that these metatheoretical entities and relations exist as *primitives* of the object theory, merely that they can constructed or expressed using the apparatus of the object theory. Since on this approach the metatheoretical statements must hold in each object theory, there is a sense in which they will be weak and 'minimal', stating only what is undisputed across all theories.

Option 2, on the other hand, is about a metatheory into which every object semantic theory can be embedded; that is, the metatheory 'implements' all of them. Whatever particular entities or operations some semantic theory may contain, they will all have correlates in the metatheory (again, not necessarily as primitives, but at least constructible). So in a sense, this option presents the metatheory as 'maximal', as a characterization of the entire semantic realm, in which all 'naturally occurring' semantic notions can be expressed. (Cooper's work on interpreting DRT in situation theory, mentioned above, is an instance of option 2.)

Which of these styles of abstraction should we prefer? Option 2 potentially offers a unifying account of every semantic notion, across whichever theories we choose to abstract over. Furthermore, if the metatheory of option 2 is executable, then we would automatically obtain a computational interpretation of each object theory that it encompassed. We know, however, that given the current state of semantic theorizing—with its plethora of incommensurable theories—such a metatheory is unlikely. Note also that there is nothing in this very general characterization of option 2 to forbid a metatheory whose top-level structure is a giant disjunction of subtheories. Such a structure would allow the metatheory to interpret any object theory by effectively including a copy of it as a disjunct. So stronger constraints on the nature of an option 2 metatheory are required before we can have any guarantee of explanatory value.

The chief difficulty with option 1, by contrast, is the prospect of only being able to make weak statements, since they have to be interpretable as true in every object theory. A corollary is that option 1 metatheories are unlikely to yield an immediate computational interpretation. One way to counter the weakness of such statements is to consider not a single option 1 metatheory, but a range of them. By creating a collection of option 1 metatheories there is the potential to address the problems of comparison and classification raised at the beginning of this chapter. Rather than a single metatheory for all of semantics, we can consider separate metatheories for distinct subtopics of semantics, such as generalized quantification, plurals and mass terms, propositional attitude reports, and so on. And within each topic, alternative metatheories can capture unresolved disagreements about empirical modelling. This is the picture I sketched in §2.1.1, and it should now be evident that abstract statements—metatheories—of the option 1 variety are one way to achieve it. It is this kind of semantic abstraction that I promote in the following chapters.

Metatheories of formal semantics are not new, of course. One significant early work is Montague's article 'Universal Grammar' (Montague 1970, henceforth UG), which I now briefly consider. In UG, Montague presents a general framework for languages (as algebras), model-theoretic semantic realms (also algebras), and interfaces from the former to the latter (homomorphisms). Viewed as an abstraction over theories of formal semantics (which was not the role Montague had in mind), it falls under my option 2. As such, it fails to provide a system in which every existing semantic theory can be interpreted—partly, of course, because many of these theories have been developed in explicit disagreement with features of Montague's systems. For example, the logics of Bealer and Turner discussed in the next chapter cannot be given a semantics within the strongly-typed model theory of UG, and the syntax-semantics interface of standard DRT (presented in, for example, Kamp and Reyle 1993) is not that of a homomorphism between syntactic and semantic algebras.

2.4 Summary

The case for abstracting over semantic theories rests on a desire to better compare and classify the semantic theories that we have, and on a claim that we can improve future theories by strengthening their modularity and coherence through the use of appropriate abstractions. At present, it is often difficult to predict exactly where two theories supposedly covering the same data actually disagree, and it is often impossible to find a natural union of two theories which aim to account for disjoint sets of data. Within many contemporary linguistic theories, syntax and semantics are closely intertwined, but in such a way that the monolithic structure of the theory prevents modularization and allows only limited type-based security. The use of single unified formalisms for both syntax and semantics has a certain elegance and superficial computational appeal, but appears to ignore many of the very different needs of semantic theorizing, and to prejudge semantic issues that research has yet to clarify.

I claim that the introduction of modularity, appropriate levels of abstraction and well-specified interfaces (familar from software engineering) can address many of these issues within semantic theory. In considering what to abstract over, there is a distinction to be made between the semantic realm proper and the syntax-semantics interface. There is some existing work that abstracts over one of these but not both, and a little that effectively does abstract over both. Only by taking this latter course can maximum generality be achieved.

The form of abstract metatheoretical statements about concrete object semantic theories must be specified. There are two options: either (option 1) we can interpret the metatheory in the terms of the object theory, or (option 2) we can do the reverse. There is a parallel with formal specification of programs: in option 1 the object theory implements the metatheory, in option 2 the metatheory implements the object one. Option 2 unifies all semantic concepts occurring in the object theories, but since it may do so with a purely disjunctive structure we have no guarantee of explanatory adequacy, and indeed given the range of existing semantic theories, good reason to doubt we can obtain it. Option 1 risks weak statements and lack of computability, but the weakness (at least) can be resolved through factorization into multiple metatheories, first by semantic topic, and second by degree of consensus on contentious issues.

I favour abstract statements of the option 1 variety. The next three chapters demonstrate the power of this approach across different domains.

Chapter 3

Case Study 1: IL, PT_2 and LQC

3.1 Intensional entities in semantics

In this chapter I examine three semantic theories from a classificatory perspective. My aim is to clarify their behaviour in the domain of intensional identity and self-predication by writing formal specifications for the operations of intensional abstraction and predication; I begin with a brief introduction to some of the key theoretical issues that surround them.

The terms **intension** and **extension**, as introduced by Carnap, are parallel to Frege's notions of **sense** and **reference** in his analysis of meaning. We need to admit, or at least speak of, intensional entities in order to adequately describe the meaning of things such as properties and propositions. For example, the property of having a heart (being cardiate) and the property of having a kidney (being renate) are distinct notions, with different meanings, but the set of things in the world that have hearts and that which have kidneys happens to be identical. We say that the intensions of "cardiate" and "renate" are distinct, but their extensions are the same.

First-order logic appears to suffice for the formal modelling of extensions, but there is still considerable disagreement on the best way to describe *intensions* formally, at least for the purpose of representing natural language meaning. In formal semantics, the most influential account of intensions has been that of Montague, which built on Church's formalization of Carnap's description of intension and extension. In Montague's account, as presented, for example, in PTQ, the model-theoretic notion of possible worlds is responsible for characterizing intensions: the intension of an expression is a function from every possible world to the expression's extension in that world.

Montague's notion of intension certainly appears rich enough to make many of the important distinctions in formal semantics. However, the logical system which defines this notion—the intensional logic IL—has come in for considerable criticism over the last twenty years on the grounds that it does not capture sufficiently well much of the behaviour of intensional entities in natural language. There are two central avenues of criticism:

1. *Overly coarse-grained intensional identity.* In Montague's logic IL, two formulas that are logically equivalent (that is, they entail each other) always give rise to identical propositions. Since propositional attitudes such as belief are traditionally modelled as a relation between a subject and a proposition, this aspect of IL is responsible for the problem of so-called logical omniscience: on Montague's theory, to believe in a proposition p is simultaneously to believe in every proposition that is logically equivalent to p, a psychologically unappealing model.

2. *Insufficient type-freedom.* IL is a strongly-typed logic, in which a predication (the ascription of a property to some entity) can only happen when predicate and object have precisely matching types. But it seems that the types of natural language expressions are more flexible than this. There are many cases where, for example, both an individual and a proposition appear to play the same role in a predication, yet IL is unable to represent this.

(See §4.5.1 for examples of each problem.)

The response to these criticisms has been a number of alternative logics and semantic theories. In the remainder of this chapter I focus on two of these, as well as on IL. George Bealer's logic of qualities and concepts (Bealer 1982) is one of them. In developing his theory, Bealer's goals were primarily philosophical: he sought an account of intensional entities that would solve philosophical puzzles such as the paradox of analysis, while remaining fully formal and a plausible 'foundational language' for semantics. The other system is due to Gennaro Chierchia and Ray Turner (Chierchia and Turner 1988). It synthesizes on the one hand Chierchia's desire, as a linguist, for a semantic theory that goes some way to explain the semi-flexible types that natural language exhibits with, on the other hand, Turner's work on property theories (kinds of intensional logic) that minimize type constraints.

3.2 Comparing theories

My intention is to apply the idea of abstract metatheoretical statements developed in the previous chapter to the comparison and classification of these three semantic theories along the dimensions of intensional identity and flexibility of predication. In order to do this I will create a series of small metatheories and investigate whether or not they can be 'implemented' by the three object semantic theories (thus following 'option 1'). I will use the language of many-sorted first-order logic to represent these metatheoretical constraints, following common usage in the formal specification of programs. The next section explains how this works.

The idea is to describe certain properties that these theories may or may not possess, without becoming committed in the description to a particular logical language (or indeed a particular model theory) for the semantic theory in question. In particular, there are two questions that, in effect, I shall ask of each theory:

1. When are two propositions equal?
2. Can Russell's property be expressed?

My reasons for the first question should be clear enough: I want to know the criteria for intensional identity in each object theory. The purpose of the second question is less obvious, so I digress briefly to explain it.

As indicated above, intensional identity and type-freedom are two of the major issues that have driven attempts to improve on Montague's intensional logic. A particularly extreme case of type-freedom occurs if we wish to allow the possibility that some property may be true of itself. There is a strong case, for example, that the (admittedly contrived) property of "being self-identical" is true of itself, as well of everything else. But since Russell's celebrated discovery of an inconsistency in Frege's first attempt to formalize arithmetic it has been well-known that allowing properties to be predicated of themselves is fraught with logical danger. **Russell's property** may be defined as "the property of all things that are not true of themselves". It is not immediately evident that this is a more problematic notion than that of being self-identical. But it turns out to be extremely difficult to construct a logic in which Russell's property may be formalized without also rendering the logic inconsistent. Consider the question: "Is Russell's property true of itself?" First suppose it is true of itself. Then Russell's property must possess "the property of all things that are not true of themselves", whence it it is *not* true of itself. So suppose it is not true of itself. Then it is not the case that it is true of itself, i.e., it is not the case that Russell's property possesses "the property of all things that are not true of themselves", so it must therefore be true of itself! In a completely type-free framework, it requires very little logical machinery to arrive at a contradiction once Russell's property is admitted as a genuine property.

So it is because of the paradoxical role that Russell's property plays in theories with significant type flexibility that I chose to exploit it in the second question that I ask of my three object semantic theories.

3.3 Formal specifications

In order to obtain answers to the questions of the previous section I construct a series of specifications of intensional and predicative behaviour, and then see whether or not the object theories meet these specifications. Given a particular metatheoretical property, the idea is to create a minimal specification of a semantic theory that possesses that property, including in the specification only the machinery required to demonstrate it. Then I say that an object theory has the property of interest if and and if it can implement the minimal specification.

To talk of theories "implementing" specifications is to lean on the analogy with formal specification of programs. In fact, it would be rather more accurate to talk of "partial implementations" or of "refinements" of specifications, since the object theories under examination here do not provide executable definitions for all their predicates and operations. In formal specification theory, the intended behaviour of a program is initially described with a fairly broad specification. As the design of the program is developed, the specification becomes more detailed, perhaps, for example, breaking down into subcases a category of inputs that was simply considered as a unit at an earlier stage. This process is called refinement, and culminates in a specification that is actually executable—a program—which may then be said to implement the original specification.

In formal specification, the intention is to provide formal proofs of every refinement step—of every point at which it is claimed a refined specification or a program satisfies the original specification. In what follows, I shall not provide such proofs, but rather appeal to the behaviour of the object theories directly. It should generally be straightforward to construct corresponding formal proofs.

My specifications are metatheories in many-sorted first-order logic. Here is an example:

$$
\begin{array}{l}
\mathsf{ADDITION} = \\
\quad \textbf{sorts}\ \mathsf{Int} \\
\quad \textbf{opns}\ \mathsf{zero} :\ \to \mathsf{Int} \\
\qquad\qquad \mathsf{succ} : \mathsf{Int} \to \mathsf{Int} \\
\quad \textbf{axioms}\ \forall x\, \mathsf{plus}(\mathsf{zero}, x) = x \\
\qquad\qquad\quad \forall x \forall y\, \mathsf{plus}(\mathsf{succ}(x), y) = \mathsf{succ}(\mathsf{plus}(x, y))
\end{array}
$$

Here **sorts** identifies the different sorts of entities that the specification recognizes. Subsort relations may hold of these; when this happens, the keyword **subsorts** will be used to introduce them. Relations on entities of the various sorts may be either operations (**opns**), yielding another kind of entity, or predicates (**preds**), which implicitly yield Boolean values. A type such as $\mathsf{Int}, \mathsf{Int} \to \mathsf{Int}$ indicates that the operation in question takes two entities of sort Int and generates another entity of that sort as its result. The **axioms** section of a specification is usually where the real content is located: it specifies constraints on the predicates and relations. Finally, note that I will often combine specifications with the symbol '+'. One may consider this to act as if the respective texts of the two specifications have been added together.

3.4 Semantic theories versus intensional logics

As mentioned above, I consider three object semantic theories here:

1. Montague's PTQ;

2. The semantic theory of Chierchia and Turner 1988, relying on the property theory PT_2 (which is closely related to the theory described in Turner 1987);
3. A semantic theory employing Bealer's property theory $L_\omega + \Delta$ (Bealer 1982), subsequently referred to as "LQC" (for the logic of qualities and concepts).

I am only truly concerned, however, with the intensional logics that underlie these theories, whence the title of this chapter mentions only IL, PT_2 and LQC. Not only can the questions raised in the previous section be answered solely by recourse to these theories' intensional logics, Bealer's system does not really constitute a fully-fledged semantic theory at all (in the sense introduced in chapter 1), since he provides no precise model for connecting a fragment of natural language with his property theory—no syntax, and no syntax-semantics interface. So from this point on I talk of logics rather than theories.

I assume some familiarity with PTQ. What is distinctive about both PT_2 and LQC is their rejection of (i) most of IL's type system (all of it in the case of LQC) and of (ii) the 'proposition as set of possible worlds' notion of intensionality that is such a core aspect of IL.

I choose to describe these logics by abstracting over their logical syntax, giving specifications in the language of order-sorted first-order logic (with operation symbols and identity). So when one of my specifications mentions the binary operation on formulas '`and`', this might be realized as '$\wedge$' in the concrete syntax of one logic, while not as an atomic operation in that of another. In the case of the two operations of central interest, '`abs`' and '`pred`', this is precisely what happens, given the way I model them.

By taking this line, I avoid considering model-theoretic objects. This leaves me free—or rather forces me—to characterize the logical nature of these systems simply in terms of a relation of logical consequence on their formulas. How a logic actually grounds its consequence relation is not relevant here: I am modelling their intensional behaviour independent of how consequence is defined.

3.5 Intensional identity

3.5.1 Abstraction

Call **intensional abstraction** any syntactic operation which takes formula-like objects to objects that stand for intensional entities, possibly binding specified variables at the same time. If no variables are bound then the result is a proposition, if a single variable is bound it is a property, and so on. Then this is how intensional abstraction appears in our three object logics:

IL In Montague's system, the operator '$\hat{}$' forms intensional entities. A proposition like $\hat{}\mathtt{love}(j, m)$ is the intension of a formula, and properties are the intensions of functions, which in turn are defined with the lambda-calculus, e.g., $\hat{}\lambda x \mathtt{love}(x, m)$.

PT$_2$ In this system, formulas always denote propositions when used in term-like contexts, so the nullary intensional abstraction operation is just the identity map. Properties exist in two forms: propositional functions (formed by lambda-abstraction), and 'nominalized' individual correlates of these. This latter form is the one I am interested in here, since only in this form can properties in PT$_2$ behave like individuals. A proposition is an expression like $\texttt{love}(j, m)$, and ${}^{\cap}\lambda x_e \texttt{love}(x_e, m)$ a property of the kind I am interested in, where '${}^{\cap}$' is Chierchia and Turner's nominalization operator.

LQC Bealer's language has a single intensional abstraction operation '$[\cdot]_{x_1 \ldots x_n}$' which optionally binds variables. Thus $\texttt{love}(j, m)$ is a formula, $[\texttt{love}(j, m)]$ is a proposition and $[\texttt{love}(x, m)]_x$ is a property.

3.5.2 Propositional identity

If we wish to say something about the operation of intensional abstraction independent of a predication operation, then initially we must restrict our attention to the nullary variant that forms only propositions. So consider a specification of sorts corresponding to formulas and propositions, with the proposition-forming operator just mentioned, and a propositional logic over the formulas:

$$
\begin{array}{l}
\mathsf{PROP} = \\
\quad \textbf{sorts}\ \mathsf{Form}, \mathsf{Prop} \\
\quad \textbf{opns}\ \mathsf{truth} : \ \to \mathsf{Form} \\
\qquad\quad \mathsf{not} : \mathsf{Form} \to \mathsf{Form} \\
\qquad\quad \mathsf{and} : \mathsf{Form}, \mathsf{Form} \to \mathsf{Form} \\
\qquad\quad \mathsf{prop} : \mathsf{Form} \to \mathsf{Prop} \\
\qquad\quad \mathsf{eqprop} : \mathsf{Prop}, \mathsf{Prop} \to \mathsf{Form} \\
\quad \textbf{preds}\ \mathsf{seq} : \mathsf{Form}, \mathsf{Form} \\
\quad \textbf{axioms}\ \forall f \forall g\, \mathsf{seq}(\mathsf{truth}, \mathsf{eqprop}(\mathsf{prop}(f), \mathsf{prop}(g))) \Rightarrow \mathsf{seq}(f, g) \wedge \mathsf{seq}(g, f)
\end{array}
$$

In the preceding specification I have added an identity relation on propositions, eqprop, to the object logic. The predicate seq is intended to capture the relation of logical consequence on formulas. What is missing from this specification, of course, is anything which constrains seq to actually behave like propositional consequence. But specifying that is not a problem peculiar to intensional logic. Therefore, I assume some solution to it. (The simplest one in this framework might just be a Hilbert-style axiomatization.) The axiom imposes the basic constraint that if the propositions corresponding to two formulas are identical, then those formulas must at least be logically equivalent.

We are now in a position to consider further constraints on the identity conditions for propositions. In fact, neither PT$_2$ nor LQC make any additional identifications in the realm of propositions, though in both cases the authors of the systems are keen to point out that certain further identifications can easily be stipulated, independent of the other logical machinery. For example, a reasonable requirement might be that 'intensional conjunction' is commutative. For propositions, we can capture this with:

$$\begin{array}{l}\mathsf{PROP\_COMM\_CONJ} = \mathsf{PROP} + \\ \quad \textbf{axioms}\ \forall f \forall g\, \mathsf{seq}(\mathsf{truth}, \mathsf{eqprop}(\mathsf{prop}(\mathsf{and}(f,g)), \mathsf{prop}(\mathsf{and}(g,f))))\end{array}$$

At the other end of the scale, only IL is described by

$$\begin{array}{l}\mathsf{PROP\_COARSE} = \mathsf{PROP} + \\ \quad \textbf{axioms}\ \forall f \forall g\, \mathsf{seq}(f,g) \wedge \mathsf{seq}(g,f) \Rightarrow \mathsf{seq}(\mathsf{truth}, \mathsf{eqprop}(\mathsf{prop}(f), \mathsf{prop}(g)))\end{array}$$

This is a specification of 'coarse-grained' propositions, which in IL give rise to the problem of logical omniscience.

3.5.3 Predication

Adding variables, entities and properties to our specification, let abs be that intensional abstraction operation that creates properties rather than propositions:

$$\begin{array}{l}\mathsf{ABS} = \mathsf{PROP} + \\ \quad \textbf{sorts}\ \mathsf{Var}, \mathsf{Ent}, \mathsf{Pty} \\ \quad \textbf{subsorts}\ \mathsf{Var} \subseteq \mathsf{Ent} \\ \quad \textbf{opns}\ \mathsf{abs} : \mathsf{Var}, \mathsf{Form} \rightarrow \mathsf{Pty} \\ \qquad\qquad \mathsf{eqpty} : \mathsf{Pty}, \mathsf{Pty} \rightarrow \mathsf{Form} \\ \quad \textbf{axioms}\ \forall f \forall g \forall x \forall y\, \mathsf{seq}(\mathsf{truth}, \mathsf{eqpty}(\mathsf{abs}(x,f), \mathsf{abs}(y,g))) \Rightarrow \\ \qquad\qquad\qquad \mathsf{seq}(f,g) \wedge \mathsf{seq}(g,f)\end{array}$$

(In the preceding axiom, I am somewhat unreasonably assuming that free variables in formulas are implicitly universally quantified for the purposes of seq. This should probably be fixed by the introduction of explicit quantification in my model of the object logic.)

Before I can give a specification for the predication operation pred as well, I need to define a substitution operation that understands conventional variable binding. The following specification does this, but it suffers from the defect that it characterizes substitution only in terms of the (minimal) operations introduced by earlier specifications—if object theories implementing specifications introduce further operations then this characterization will be insufficient.[1]

$$\begin{array}{l}\mathsf{SUBST} = \\ \quad \textbf{opns}\ \mathsf{subst} : \mathsf{Ent}, \mathsf{Var}, \mathsf{Form} \rightarrow \mathsf{Form} \\ \quad \textbf{axioms}\ \forall e \forall x\, \mathsf{subst}(e, x, \mathsf{truth}) = \mathsf{truth} \\ \qquad\qquad\quad \forall e \forall x \forall f\, \mathsf{subst}(e, x, \mathsf{not}(f)) = \mathsf{not}(\mathsf{subst}(e, x, f))\end{array}$$

[1]I'm grateful to Henk Zeevat for pointing this out to me. A solution to this problem appears to require a more comprehensive metatheoretical treatment of object language syntax than I can accommodate here.

$$\begin{array}{l}
\forall e \forall x \forall f \forall g\, \mathsf{subst}(e,x,\mathsf{and}(f,g)) = \mathsf{and}(\mathsf{subst}(e,x,f),\mathsf{subst}(e,x,g)) \\
\forall e \forall x \forall y \forall f \forall p\, p \neq \mathsf{abs}(y,f) \Rightarrow \mathsf{subst}(e,x,\mathsf{pred}(p,x)) = \mathsf{pred}(p,e) \\
\forall e \forall x \forall f\, \mathsf{subst}(e,x,\mathsf{pred}(\mathsf{abs}(x,f),x)) = \mathsf{pred}(\mathsf{abs}(x,f),e) \\
\forall e \forall x \forall y \forall f\, x \neq y \Rightarrow \\
\quad \mathsf{subst}(e,x,\mathsf{pred}(\mathsf{abs}(y,f),x)) = \mathsf{pred}(\mathsf{abs}(y,\mathsf{subst}(e,x,f)),e) \\
\forall d \forall e \forall x \forall f\, x \neq d \Rightarrow \\
\quad \mathsf{subst}(e,x,\mathsf{pred}(\mathsf{abs}(x,f),d)) = \mathsf{pred}(\mathsf{abs}(x,f),d) \\
\forall d \forall e \forall x \forall y \forall f (x \neq d \wedge x \neq y) \Rightarrow \\
\quad \mathsf{subst}(e,x,\mathsf{pred}(\mathsf{abs}(y,f),d)) = \mathsf{pred}(\mathsf{abs}(y,\mathsf{subst}(e,x,f)),d)
\end{array}$$

It now only remains to specify that pred satisfy some minimal principle of predication. This is a delicate area, since intensional logics vary widely in their strategies for avoiding the logical paradoxes while getting as close as possible to the naive version of this principle. For the time being, I will choose a very limited axiom here, based on the restriction to atomic properties (those not constructed with abs).

$$\begin{array}{l}
\mathsf{PRED} = \mathsf{ABS} + \mathsf{SUBST} + \\
\quad \textbf{opns}\ \mathsf{pred} : \mathsf{Pty}, \mathsf{Ent} \rightarrow \mathsf{Form} \\
\quad \textbf{axioms}\ \forall f \forall g \forall x \forall y \forall d \forall e \forall p \forall q \\
\qquad\qquad (p = \mathsf{abs}(x,f) \wedge f = \mathsf{pred}(q,d) \wedge q \neq \mathsf{abs}(y,g)) \Rightarrow \\
\qquad\qquad \mathsf{seq}(\mathsf{pred}(p,e),\mathsf{subst}(e,x,f)) \wedge \mathsf{seq}(\mathsf{subst}(e,x,f),\mathsf{pred}(p,e))
\end{array}$$

The above description of predication allows 'fine-grained' distinctions, so that the propositions corresponding to predications before and after substitution may differ. To get a more conventional 'grainedness', we can say in addition

$$\begin{array}{l}
\mathsf{PRED\_EQFULL} = \mathsf{PRED} + \\
\quad \textbf{axioms}\ \forall f \forall x \forall e \forall p\, p = \mathsf{abs}(x,f) \Rightarrow \\
\qquad\qquad \mathsf{seq}(\mathsf{truth},\mathsf{eqprop}(\mathsf{prop}(\mathsf{pred}(p,e)),\mathsf{prop}(\mathsf{subst}(e,x,f))))
\end{array}$$

which forces $\mathsf{pred}(\mathsf{abs}(x,f),e)$ and $\mathsf{subst}(e,x,f)$ to stand for the same proposition. In fact, because of the axiom in PRED, this entails the full principle of predication, and is therefore impossible for any real 'property theory' (though not, of course, for IL). The right description for PT_2 is

$$\begin{array}{l}
\mathsf{PRED\_EQATOM} = \mathsf{PRED} + \\
\quad \textbf{axioms}\ \forall f \forall g \forall x \forall y \forall d \forall e \forall p \forall q \\
\qquad\qquad (p = \mathsf{abs}(x,f) \wedge f = \mathsf{pred}(q,d) \wedge q \neq \mathsf{abs}(y,g)) \Rightarrow \\
\qquad\qquad \mathsf{seq}(\mathsf{truth},\mathsf{eqprop}(\mathsf{prop}(\mathsf{pred}(p,e)),\mathsf{prop}(\mathsf{subst}(e,x,f))))
\end{array}$$

which restricts the identity claim to atomic properties.

Of course if PROP_COARSE holds anyway, as in IL, then these distinctions carry no weight, since propositions are already being individuated more coarsely than this.

I have indicated how abs is 'implemented' in my three sample logics; here's how pred comes out:

IL $\mathsf{pred}(\alpha, \beta) = {}^{\vee}\alpha(\beta)$.

PT$_2$ $\mathsf{pred}(\alpha, \beta) = {}^{\dagger}({}^{\cup}\alpha(\beta))$.

LQC $\mathsf{pred}(\alpha, \beta) = \beta \, \Delta \, \alpha$.

3.6 Russell's property

One measure of the expressiveness, or at least the type-freedom, of an intensional logic might be taken to be whether or not it admits an expression which stands for Russell's property. Here I attempt to describe a formal test for this property. We want a minimal description of a logic that can express Russell's property. I propose:

$$\begin{aligned} \mathsf{RUSSELL} = \ & \textbf{sorts}\ \mathsf{Var}, \mathsf{Ent}, \mathsf{Pty}, \mathsf{Form} \\ & \textbf{subsorts}\ \mathsf{Var} \subseteq \mathsf{Ent}, \mathsf{Var} \subseteq \mathsf{Pty} \\ & \textbf{opns}\ \mathsf{not} : \mathsf{Form} \to \mathsf{Form} \\ & \qquad \mathsf{abs} : \mathsf{Var}, \mathsf{Form} \to \mathsf{Pty} \\ & \qquad \mathsf{pred} : \mathsf{Pty}, \mathsf{Ent} \to \mathsf{Form} \end{aligned}$$

together with a comprehension axiom such as that given above for PRED, and of course some specification of the propositional logic. Then given an element x of sort Var the expression

$$\mathsf{abs}(\mathsf{x}, \mathsf{not}(\mathsf{pred}(\mathsf{x}, \mathsf{x})))$$

will represent Russell's property.

Now there is no way to implement the specification RUSSELL in IL, owing to IL's type constraints; this is perhaps most evident from the **subsorts** specification, which requires that there be a kind of variable that is simultaneously an entity and a property. Because variables are strongly typed in IL, and the types of entity and property are different, there can be no such variable.

On the other hand, both PT$_2$ and LQC will satisfy this specification. Appendix A contains a Standard ML program which partially illustrates this, by providing implementations of the syntax of PT$_2$, LQC and the RUSSELL specification, and demonstrating that there are maps which interpret RUSSELL in those two systems. In the case of PT$_2$ the resulting Russell's property expression is exactly as given on p. 267 of Chierchia and Turner 1988. More detail is provided in the appendix.

These results show that the specification RUSSELL can be successfully used to distinguish between the strongly-typed logic IL and the more type-free logics PT$_2$ and LQC. It's worth pointing out that there are further distinctions to be modelled here. For example, there are logics (or property theories) that lie between these two positions in terms of type-freedom, such as the system of Kamareddine and Klein (1993), in which self-predication can be expressed, but Russell's property cannot be. And it would be possible to distinguish between PT_2 and LQC on the basis of a specification which required that Russell's property can actually be predicated of itself to yield a proposition: this is true in LQC but false in PT_2.

3.7 Summary

The correct modelling of intensional entities remains an unresolved issue in formal semantics. Since PTQ, the central concerns have been criteria for intensional identity and for type-freedom. I chose Montague's PTQ, and theories due to Bealer and to Chierchia and Turner as examples of post-Montagovian semantic theories that address these issues and span a wide range. In order to compare them I considered one question about intensional identity (of propositions), and one about type-freedom (the ability to express Russell's property). I constructed small specificational metatheories corresponding to differing answers to these questions: then the test of whether or not an object theory had the property in question was reduced to whether or not it could implement that specification, borrowing an idea from the formal specification of programs (from where I also borrowed the use of many-sorted first-order logic for the form of specifications).

For these purposes I viewed my three object theories as intensional logics rather than full semantic theories, since this was sufficient for the task in hand and necessary in the case of one system, which fails to count as an entire semantic theory. The style of specification was to abstract over the concrete logical syntaxes of these systems, in effect introducing an abstract syntax. I ignored the question of the correct specification of core *logical* behaviour, that is of logical consequence, since that is a problem that standard model- and proof-theory have independently supplied answers to.

Starting with the modelling of an intensional abstraction that takes formulas to propositions, I introduced a basic minimal specification of propositional identity, allowing differing notions of identity to be captured with a series of more specific (and coarser-grained) descriptions of that identity. Extending to predication by allowing abstraction operations to bind variables, I introduced the required notion of substitution and again modelled varying degrees of intensional identity, this time at the level of propositions that involve predications.

Finally, a core specification of the ability to express Russell's property was simple to devise, and demonstrated the required distinction between Montague's strongly typed theory and the two more recent competitors.

This chapter has demonstrated the potential of abstract descriptions of semantic theories to compare and classify those theories on the basis of formal specifications of their behaviour. The metaphor—and some of the techniques—of formal specification of programs was used to achieve this, but as discussed in chapter 2, we may equivalently talk in terms of logical embeddings. It is apparent that is surprisingly easy to formalize notions of degree of intensional identity and of type-freedom that are so often spoken of informally, and in the process gain better understanding of exactly why it is that some theories have the metatheories properties that they do.

I now move on to consider an existing approach to semantic abstraction, evaluating it with respect to the general framework I discussed in chapter 2.

Chapter 4

Case Study 2: Johnson and Kay's 'Semantic Abstraction and Anaphora'

4.1 Introduction

This chapter is an analysis of Mark Johnson and Martin Kay's 'Semantic abstraction and anaphora' (Johnson and Kay 1990). In their paper, Johnson and Kay show how a grammar that builds semantic representations can be parameterized by a small number of 'semantic operators', thereby causing the grammar to abstract over a class of semantic representations while retaining a fixed interface to the syntax. They give a grammar for a fragment of English, written in Prolog's DCG notation, in which the semantic operators are defined as ordinary Prolog predicates. This grammar covers the core data on quantification and anaphora that was first systematically treated by Kamp's discourse representation theory (DRT, Kamp 1981, see also Kamp and Reyle 1993), and it may be viewed as a generalization of the DRT implementation described in Johnson and Klein 1986. Along with the grammar, three candidate implementations of the semantic operators are provided, corresponding to three notions of semantic representation for the sentences admitted by the grammar.

My concern with this work is how it relates to the general notion of semantic abstraction motivated in chapter 2. I want to answer the following questions about 'Semantic abstraction and anaphora':

- What is the level of semantic abstraction that Johnson and Kay employ?
- What constraints does it impose on the grammar and on the space of possible semantic representations?
- Can these constraints be formalized, so as to give a specification of such a representation?

```
% Operator declaration and parse/2 predicate

:- op(1200, xfx, ==>).

parse(String, ExtSem) :-
    external(IntSem, ExtSem),
    s(IntSem, String, []).

% Grammar

s(S) --> np(VP^S), vp(VP).
np(NP) --> det(N1^NP), n1(N1).
n1(N) --> n(N).
n1(X^S) --> n(X^S1), rc(X^S2),
    { conjoin(S1, S2, S) }.
vp(X^S) --> v(X^VP), np(VP^S).
rc(VP) --> [that], vp(VP).
v(X^Y^S) --> [Verb],
    { verb(Verb, X^Y^Pred),
    atom(Pred, S) }.
n(X^S) --> [Noun],
    { noun(Noun, X^Pred),
    new_index(X, S1),
    atom(Pred, S2),
    compose(S1, S2, S) }.
det((X^Res)^(X^Scope)^S) --> [Det],
    { determiner(Det, Res^Scope^S) }.
np((X^S1)^S) --> [Pronoun],
    { pronoun(Pronoun),
    accessible_index(X, S2),
    compose(S1, S2, S) }.
```

Figure 4.1: The Johnson and Kay grammar

- How exactly does 'propositional' information interact with 'anaphoric' information in the system? To what extent can the two kinds be teased apart?

4.2 Grammar

Figure 4.1 shows Johnson and Kay's grammar, and figure 4.2 their example lexicon. Regarding syntactic coverage, the grammar admits declarative sentences with transitive verb phrases and with both indefinite and universally quantified noun phrases, which may have restrictive relative clauses. NPs may also be pronouns, thereby including in the grammar's language the universally quantified version of Geach's donkey sentence:

Every man that owns a donkey beats it.

```
% Lexicon

pronoun(he).
pronoun(she).
pronoun(him).
pronoun(her).
pronoun(it).

verb(likes, X^Y^likes(X,Y)).
verb(saw, X^Y^see(X,Y)).
verb(beats, X^Y^beat(X,Y)).
verb(owns, X^Y^own(X,Y)).

noun(woman, X^woman(X)).
noun(man, X^man(X)).
noun(donkey, X^donkey(X)).

determiner(a, Res^Scope^S) :-
    conjoin(Res, Scope, S).
determiner(every, Res0^Scope^S) :-
    compose(S1, S2, S),
    subordinate(Res, ResName, S1),
    compose(Res0, Res1, Res),
    subordinate(Scope, ScopeName, Res1),
    atom((ResName ==> ScopeName), S2).
```

Figure 4.2: The lexicon

As it stands, the grammar does not admit sentential conjunction or implication, nor the sequencing of sentences to form discourses (which would permit extra-sentential anaphora). All these are, however, easy to add to the grammar in the spirit of the existing constructions.

The grammar takes the form of a DCG augmented with explicit calls to Prolog predicates using the standard "{*goal*, ... }" syntax (an introduction to DCGs may be found in Shieber 1986). These Prolog calls fall into two categories: that of the 'lexical' predicates pronoun, verb, noun and determiner, and that of the seven so-called semantic operators, listed in table 4.2 and discussed at length in the next section. With the exception of determiner, the lexical predicates serve only to introduce terminal symbols, and are thus merely a convenience, rather than an essential deviation from the form of a pure DCG.

4.2.1 Semantic values

All the non-terminal symbols of the grammar have a single argument, standing for the semantic value of that constituent. In general, these semantic values have the form of Montagovian lambda-terms, with the Prolog functor "^" used to

category	*value*
S	t
N, N1, VP, RC	$e\hat{}t$
V	$e\hat{}(e\hat{}t)$
NP	$(e\hat{}t)\hat{}t$
Det	$(e\hat{}t)\hat{}((e\hat{}t)\hat{}t)$

Table 4.1: Syntactic categories and semantic values

represent lambda abstraction. Table 4.1 illustrates, for each syntactic category of the grammar, the form of its semantic value, where t stands for the type of semantic value assigned to clausal constituents, and e for the underlying entity type. In the next section I give the name `rep` to the type t, and follow Johnson and Kay by using the name `index` for the type e.

In Montague's system, the semantic value of a given constituent is always obtained from the values of its sub-constituents by the operation of function application. In Johnson and Kay's system, the task is split between function application and their distinguished semantic operators. Where function application is to take place, Johnson and Kay utilize the Prolog implementation technique of Pereira and Shieber (1987), in which the lambda conversion that follows function application is captured implicitly (or partially-evaluated away) in each clause of the DCG, via the unification of appropriate sub-parts of semantic values. In addition to this, where one or more semantic operators are to apply, they appear as explicit Prolog calls (as already noted) and operate only on clausal values (things of type t), though those values may well be part of more complex lambda-terms. So in the clause for S, for example, only function application occurs. In that for V, only semantic operators apply. In most of the rest, both kinds of semantic value combination happen.

One important property of this system, then, is that these distinguished semantic operators are only capable of operating on semantic values, or representations, that correspond to clausal constituents.

It is with respect to these seven Prolog predicates that Johnson and Kay's grammar is parameterized: the grammar is not fully defined until each predicate has a definition. When it does, the set of seven definitions constitutes an implementation of the semantic operators, and it is three of these sets that Johnson and Kay provide in their paper, as sample implementations.

Before progressing to a detailed examination of the role of the seven operators, I wish to diverge from Johnson and Kay's terminology and call them by the more neutral term 'primitives' instead. For other than the trivial interpretation of them as operators (functions) from their arguments to truth values, none of them need have a functional interpretation over their arguments. Some of them, indeed,

primitive	*type*
external	rep, ext $\rightarrow$ bool
atom	rep, atprop $\rightarrow$ bool
conjoin	rep, rep, rep $\rightarrow$ bool
compose	rep, rep, rep $\rightarrow$ bool
subordinate	rep, name, rep $\rightarrow$ bool
new_index	index, rep $\rightarrow$ bool
accessible_index	index, rep $\rightarrow$ bool
==>	name, name $\rightarrow$ iprop
indexes	lprop $\rightarrow$ indexset

Table 4.2: The semantic primitives

cannot have that interpretation, in general—consider for example the predicate accessible_index, of two arguments, an index and a semantic representation. Its intended interpretation is that it is true just in case the index is anaphorically accessible from the representation. But one representation can have many accessible indexes, and one index can be accessible from many representations, so it is clearly nonsensical to suggest that the index and the representation could be functionally dependent on each other. For this reason, henceforth I refer to Johnson and Kay's 'semantic operators' as 'semantic primitives'. (One may say, then, that the Johnson and Kay syntax-semantics interface is a *relational* one, not a functional one, as it is PTQ and many other semantic theories.)

4.3 Semantic primitives

4.3.1 Sorts

Ordinary Prolog does not come with a type system, and Johnson and Kay do not attempt to classify the arguments of their semantic primitives according to the type of their arguments. None the less, it is possible to distinguish six sorts of entity at work, of which one sort has two subsorts. I find it helpful to define the primitives over these sorts, as in table 4.2, but note that the sorts are purely a convention of mine, and only implicitly part of the Johnson and Kay system. Note also that some of these sorts are fixed by the grammar, independent of any implementation, while some must be defined by an implementation (of the primitives).

I have given these sorts names: rep, ext, atprop, index, indexset and name. Sort atprop has the subsorts lprop and iprop. In general, there are no other subsort relations among the sorts. The sort rep is the sort of internal semantic representations (for clausal constituents), referred to above as type t. Intuitively, all and any information about the meaning of a constituent is stored in something of sort rep. Sort ext is the 'external' version of rep: the semantic value which the grammar presents to the outside world on completion of a parse. Both rep and ext are defined by the implementation.

The sort atprop stands for 'atomic propositions', which come in two flavours: those that form part of lexical entries, which belong to lprop, and those that are 'implicational' propositions, constructed by the grammar, which belong to iprop. There is something a little odd about calling this latter category 'atomic'; the name comes from the fact that it is only entities of this sort that can be arguments to the primitive atom. This primitive is true of an atprop and a rep when the latter represents the former—it is the basic primitive that introduces propositional information into representations, to be contrasted with those other primitives that manipulate existing representations, such as conjoin and compose. That is the sense in which its first argument is 'atomic'.

The sort lprop is completely defined by the grammar; for the one given in figure 4.1 its membership is

$$\bigcup_{x,y} \left\{ \begin{array}{l} \texttt{likes}(x,y), \texttt{see}(x,y), \texttt{beat}(x,y), \texttt{own}(x,y), \\ \texttt{woman}(x), \texttt{man}(x), \texttt{donkey}(x) \end{array} \right\}$$

where x and y range over the sort index. When lexical propositions contain indexes they stand for the 'holes' that entities can occupy. Again, the sort index is defined by the grammar, and in this case is the set of Prolog variables.[1] A set of indexes is an element of the sort indexset. Finally, the sort name describes the kind of thing which one of the primitives uses to 'name' things in sort rep.

With this final sort, we can partially characterize the members of iprop. They are constructed by the grammar from entities of sort name. It is up to the implementation to say what a name is, but once this is established, all members of iprop are merely pairs of elements from name, where the Prolog functor ==> is used to encode the pairing. It is convenient to regard this functor as a further semantic primitive, albeit one constrained to always act as a pairing, and I will do so from now on, having listed it at the bottom of table 4.2. The other primitive of my invention at the bottom of that table is indexes, which again is implicit in Johnson and Kay's system. Given an lprop, indexes extracts the set of indexes it contains.

4.3.2 Interpretations

In considering the meanings of the semantic primitives, it is worth bearing in mind that an implementation is free to make its semantic representations quite

[1]Using Prolog variables for this purpose is essentially an implementation decision of Johnson and Kay; were ordinary Prolog terms to be used instead, there would be two consequences: (i) the Pereira and Shieber style of lambda reduction could not be employed, and (ii) a number of index equalities would have to be specified explicitly along with calls to the primitives (taking proper account of which would require some additional Prolog machinery).

primitive	*true in case*
external(r, e)	e is the external logical form corresponding to r, where r represents a complete parse.
atom(r, p)	r represents the proposition p.
conjoin(r_1, r_2, t)	t represents the logical conjunction of r_1 and r_2.
compose(r_1, r_2, t)	t represents information in both r_1 and r_2, combined in that order.
subordinate(r_1, n, r_2)	n is a 'name' of r_1, and r_1 is 'subordinate' to r_2.
new_index(x, r)	r represents x as a new index for some (non-anaphoric) NP.
accessible_index(x, r)	x is an anaphorically accessible index for r.

Table 4.3: Meanings of the primitives

complex entities. In particular, it is essential that any implementation which processes anaphora in the style of DRT uses its internal representations to store not only the meaning of the 'current' constituent, but also to record information about all prior constituents. This is exactly what the two anaphorically-sensitive implementations do, as discussed in detail in §4.4.

Table 4.3 indicates the intended interpretation of the seven Johnson and Kay primitives. Of these, the grammar uses the primitives external, atom and conjoin in isolation at various points. The use of the first, in `parse/2` was explained above. The primitive atom is used alone to introduce a lexical verb meaning in the clause for V. There are two places where conjoin is used: in the semantics of a relative clause, to combine the noun and relative clause meanings, and in the meaning of the indefinite determiner a, where the restrictor and scope are combined with conjoin.

It is difficult to give convincing explanations of the meaning of some of the primitives independent of a particular implementation. Hence I will rely on the 'discourse-representation' implementation—which closely follows DRT—in order to give a better feel for the intuitions behind them.

The primitives new_index and accessible_index are responsible, respectively, for introducing and accessing indexes that can potentially play a role as anaphoric antecedents: the model here is the notion of discourse referent from DRT.

The semantics and anaphoric constraints of DRT rely heavily on the notion of one DRS being **subordinate** to another. It is this relation that the subordinate primitive is intended to capture. Rather than being a two-place relation, as one might expect, subordinate is three-place, with the additional argument position being an entity of sort name. The intended interpretation is that when subordinate(r_1, n, r_2) holds, n is a representation of the propositional entity r_1, of a kind that can participate as an argument of the ==> primitive. Allowing for the possibility that the 'name' of a propositional representation may not be the same thing as the representation itself is supposed to allow the modelling of 'non-extensional' semantic theories. So far as I can establish, there is nothing in the use of subordinate which requires that this connection between name and representation happens in subordinate—it could have been provided via a separate 'naming' primitive.

These DRT-oriented primitives do not occur on their own, but only in combination with others. There are three places in the grammar where this happens (the clauses for N, pronouns, and universal quantification), and it is convenient to define new terms for the combined effect of the primitives in these cases:

$$\begin{aligned}\mathsf{cn}(x,p,r) &=_{\text{def}} \mathsf{new\_index}(i,r_1) \wedge \mathsf{atom}(p,r_2) \wedge \mathsf{compose}(r_1,r_2,r)\\ &\quad \wedge \mathsf{indexes}(p) = \{x\}\end{aligned}$$

$$\mathsf{pro}(i,r_1,r_2) =_{\text{def}} \mathsf{accessible\_index}(x,r_3) \wedge \mathsf{compose}(r_1,r_3,r_2)$$

$$\begin{aligned}\mathsf{imp}(r,s,t) &=_{\text{def}} \mathsf{subordinate}(s,n_s,r_1) \wedge \mathsf{compose}(r,r_1,r_2)\\ &\quad \wedge \mathsf{subordinate}(r_2,n_r,t_1) \wedge \mathsf{atom}(n_r \texttt{==>} n_s,t_2) \wedge \mathsf{compose}(t_1,t_2,t)\end{aligned}$$

The term cn is used to describe the semantics of common nouns: $\mathsf{cn}(x,p,r)$ is true when abstracting over x for p results in a property, and when r represents both the proposition p and the information that x is a new NP index. The equation '$\mathsf{indexes}(p) = \{x\}$' captures the condition on abstraction, and is my explicit rendering of a constraint implicit in the grammar—namely that lexical entries for nouns always pair a proposition with the index over which abstraction is intended.

The second term deals with pronouns: $\mathsf{pro}(x,r_1,r_2)$ is true when x is an anaphorically accessible index for r_1, and when r_2 represents this fact plus the information in r_2.

The third term is the mechanism whereby the meaning of conditionals and universal quantifications is constructed. In the definition of $\mathsf{imp}(r,s,t)$, r represents the restrictor in a quantification and s the scope (antecedent and consequent in a conditional, respectively). The intended propositional meaning derives only from the use of atom and ==>. The other primitives are responsible for introducing the notion of 'subordinate' representation in the way required by DRT, so as to permit the primitive accessible_index to take advantage of this information when calculating anaphoric accessibility. Both the two more complex implementations, discussed below, show how this works.

Before passing to the details of the sample implementations, note that the precise syntactic coverage of the grammar is not fixed independent of the implementation of the semantic primitives. It is quite possible for their implementation to rule out parses on semantic, as opposed to syntactic, grounds. In the two more complex implementations, for example, the sentence *He beats him* is not parsable,

```
external(P, P).

atom(P, P).

conjoin(P, Q, P & Q).

compose(P, P, P).

subordinate(Sub, Sub, _).

new_index(_, _).

accessible_index(_, _).
```

Figure 4.3: The ‘predicate-logic’ implementation

because for neither pronoun is there sufficient context to provide an antecedent. The sentence *He beats a man* is accepted, however, with the interpretation that the antecedent of *He* is *a man*. This is a consequence of how Johnson and Kay’s grammar rule for pronouns searches for possible antecedents: the search includes any indexes introduced in the ‘propositional’ part of the pronoun NP meaning (which represents the result of combining the pronoun meaning with the meaning of the phrase it is an argument of). In particular, this means that a subject pronoun may find antecedents anywhere in the following verb phrase (up to subordination constraints).

The following slightly modified pronoun rule improves this behaviour by restricting possible antecedents to occur only in that context which is prior to the ‘propositional’ part of the pronoun meaning:

```
np((X^S2)^S) --> [Pronoun],
    { pronoun(Pronoun),
    accessible_index(X,S1),
    compose(S1,S2,S) }.
```

This change is sufficient to rule out *He beats a man*.

4.4 Implementations

Here I discuss each of the sample implementations of the semantic primitives.

4.4.1 ‘Predicate-logic’

Figure 4.3 illustrates the ‘predicate-logic’ implementation. The final four primitives, which are those that have the potential to determine anaphoric processing, are given degenerate definitions. In particular, whatever kind of information compose

```
external([[]]-[S], S).

atom(P, [B|Bs]-[[P|B]|Bs]).

conjoin(P1, P2, P12) :- compose(P1, P2, P12).

compose(B0s-B1s, B1s-B2s, B0s-B2s).

subordinate([[]|B0s]-[B|B1s], B, B0s-B1s).

new_index(Index, C) :- atom(i(Index), C).

accessible_index(Index, Bs-Bs) :-
    member(B, Bs),
    member(i(Index), B).
```

Figure 4.4: The 'discourse-representation' implementation

deals with, it is the same for every representation; subordinate says nothing about its third argument, the superordinate representation (and therefore encodes no subordinate-superordinate relation) while new_index and accessible_index impose no constraints at all on their arguments (or more logically, they are true for all values of them). Hence this implementation cannot perform anaphor resolution. We see from the definition of external that the representations it does construct, internally, are identical to those it makes available externally.

With this implementation of the semantic primitives the grammar parses the donkey sentence given in §4.2 and provides the following (external) meaning for it:

```
(man(X) & donkey(Y) & own(X,Y)) ==> beat(X,Z)
```

Somewhat puzzlingly, this is not a closed sentence of ordinary predicate logic, since there are no explicit quantifiers for the variables. But the intended interpretation (in conventional first-order predicate logic) is evidently

$$\forall x \forall y (\texttt{man}(x) \wedge \texttt{donkey}(y) \wedge \texttt{own}(x, y) \Rightarrow \exists z (\texttt{beat}(x, z)))$$

Johnson and Kay indicate this much (p. 5), but say no more about a general rule for the proper interpretation of the representations constructed by this implementation. I return to this issue in §4.6.

4.4.2 'Discourse-representation'

The implementation for DRT is shown in figure 4.4. The external representations it constructs are similar to the discourse representations structures (DRSs) of

Kamp's DRT, using a linearization of his box notation. The donkey sentence has the representation

```
[[own(X,Y),donkey(Y),i(Y),man(X),i(X)] ==> [beat(X,Y)]]
```

in which a term of the form `i(`x`)` indicates that in Kamp's notation the index x would be listed at the top of the enclosing DRS (and called a discourse referent).

In this implementation, internal and external semantic representations are not the same. They differ in two ways. First, an internal representation is a list of DRSs, rather than a single one. Second, each DRS in that list does not have the form of an ordinary Prolog list, like that above, but is instead encoded as a 'difference list', in the common Prolog style (and perhaps more accurately described as a *difference pair* of ordinary lists).

The first change enables an internal representation to model DRT's subordinate relation as a hierarchy of DRSs. The second change could be motivated on efficiency grounds alone, since difference lists allow efficient implementation of a number of list operations. But there is a more important reason: the structure of a difference list is exploited to automatically record the semantic content of 'prior' constituents. In combination with the use of a list of DRSs, this means that a given index can be considered to be anaphorically accessible if it occurs in the prior content of any of those DRSs. This captures the DRT notion of occurrence in some superordinate DRS.

(Modelling the structures of DRT in this way is very close to the approach taken in Johnson and Klein (1986). The most important difference, of course, is that in this system the DRT-like manipulation of semantic representations is explicitly detached from the grammar. A further difference is Johnson and Kay's use of Montagovian lambda abstraction to combine sub-clausal semantic values.)

To see how difference lists allow an elegant representation for prior content, recall first that the 'value' of a difference list $A{-}B$ is a list C with the property that $\mathit{append}(A, C) = B$.[2] (If there is no such C, then $A{-}B$ isn't a difference list.) So the value of

```
[man(X),i(X)]-[man(X),i(X),beats(X,Y),donkey(Y),i(Y)]
```

is the list

```
[beats(X,Y),donkey(Y),i(Y)].
```

With this representation we can characterize an *append* relation on difference lists very simply:

$$\forall A \forall B \forall C (\mathit{dl\_append}(A{-}B, B{-}C, A{-}C))$$

[2]It's more common to represent difference lists the other way round—as $B{-}A$ rather than my $A{-}B$. I diverge from this trend only for consistency with Johnson and Kay, who use the latter ordering for their semantic value difference lists in this implementation and the following one.

This constraint is directly usable as an efficient Prolog implementation of the *dl_append* relation. It also has the following important property. Suppose a number of difference lists are appended together thus (writing *dl_append* in infix notation as '$\oplus$'):

$$A_1{-}B_1 \oplus A_2{-}B_2 \oplus \cdots \oplus A_n{-}B_n$$

Then for any r,

$$A_r = v(A_1{-}B_1) + \cdots + v(A_{r-1}{-}B_{r-1})$$

where '$+$' is ordinary append, and $v(A{-}B)$ is the value of the difference list $A{-}B$. This means that each difference list $A_r{-}B_r$ automatically contains the (append of) values of all the difference lists with which $A_r{-}B_r$ has been 'pre-appended'. It is this property that the 'discourse-representation' implementation exploits when it choose to represent meanings as difference lists of DRSs, and to use *dl_append* directly as the implementation of the compose primitive.

It should now be clear how accessible_index works: it arranges for a null-valued difference list to be composed into the append sequence when a pronoun is parsed, and is then able to trawl through prior context by searching through the ordinary list that (by being paired with itself) makes up that difference list. What it looks for is simply a term contributed by new_index at some earlier point in the sequence.

It should also be possible to make more sense of the imp relation in the light of this implementation. To do this we need to understand what subordinate is doing. In this implementation, subordinate(r, n, s) is true when n (the 'name') is the value (or current content) of r, when s consists of the superordinate part of r's meaning, and when r itself is constrained to have no prior context (at the base level). Recall the definition of imp. Then the effect of the first two primitives is to add the restrictor of the quantification (and any context it may carry) to the superordinate part of the scope's representation, thereby allowing anaphora from scope to restrictor. The second use of subordinate combined with the final compose allows any context acquired by the ultimate representation to become superordinate context for the restrictor (and therefore, in virtue of the previous sentence, also for the scope). Finally, as remarked earlier, the use of atom and compose incorporates the truth-conditional content of the quantification or conditional into the outgoing representation.

4.4.3 'Sets-of-infons'

Figure 4.5 shows the 'sets-of-infons' implementation. As Johnson and Kay remark (p. 10) it may be considered a notational variant of the 'discourse-representation' implementation just discussed. Here, a representation consists of a difference list of situation-and-infon pairs, and an ordinary list of situation names, which represents their position in a subordinate ordering of situations. All that is really

```
external(@([Sit],[],Is), Sit:Is) :- gensym(Sit).

atom(P, @([Sit|_],Is,[(Sit:P)|Is])).

conjoin(I1, I2, I12) :- compose(I1, I2, I12).

compose(@(Ss,I0s,I1s), @(Ss,I1s,I2s), @(Ss,I0s,I2s)).

subordinate(@([Sit|Sits],I0s,I1s), Sit, @(Sits,I0s,I1s)) :-
    gensym(Sit).

new_index(Index, S) :- atom(i(Index), S).

accessible_index(Index, @(Ss,Is,Is)) :-
    member(Sit:i(Index), Is),
    member(Sit, Ss).
```

Figure 4.5: The 'sets-of-infons' implementation

different from the 'discourse-representation' implementation is that the subordinate information has been factored out into an ordered list of situations, with which infons are tagged. Infons are then otherwise simply the contents of the previous implementation's DRSs. The single difference list of (tagged) infons is used to record prior context as well as current content in an entirely analogous fashion. Functions to translate one-to-one between the two representations may readily be constructed.

For completeness, here is the external representation of the donkey sentence in this implementation:

```
s0:[s0:s1==>s2,s2:beat(X,Y),s1:own(X,Y),s1:i(Y),
  s1:donkey(Y),s1:i(X),s1:man(X)]
```

4.5 Semantic abstraction

One of the benefits of an appropriately abstract interface between syntax and semantics ought to be that certain kinds of inherently nonsensical or incoherent semantic operations cannot be specified by a syntactic rule. In computer science, this is exactly the protection afforded a programmer by the use of abstract data types to specify program components. Johnson and Kay's approach opens the door to this kind of security for the syntax-semantics interface of a natural language grammar, but fails to go the whole way. Most importantly, Johnson and Kay put no formal constraints on the implementation of their semantic primitives: their specification is left at the informal, intuitive level, guided by their example implementations.

The rest of this chapter consists of my attempt to rectify this. Whether or not it succeeds in that aim, it at least serves to shed some light on the precise nature of the abstraction that Johnson and Kay's system achieves.

4.5.1 Content and behaviour

It has already been observed that the 'predicate-logic' implementation gives degenerate definitions to those primitives that are concerned solely with anaphoric processing, and as a consequence does none. The other two implementations do process anaphora, however, so I shall call them the anaphoric implementations in recognition of this. In §4.4 I showed that anaphora resolution in them is dependent on recording prior discourse content in the internal semantic representation of every constituent. It is natural to distinguish this part of the representation, which one might call the *contextual content*, from the part which represents the intended semantic value of the constituent, the *direct content*. Then we can can relate the two, at least for these implementations, by saying that the contextual content represents the direct content of prior constituents.

In the anaphoric implementations these two kinds of content play different roles. Direct content is the final semantic value, as captured in an external semantic representation, while contextual content serves to resolve ambiguities (those introduced by anaphoric elements) in order to get at a semantic value.

This distinction seems useful, but our only precise characterization of it so far is implementation-dependent. We would like to formally describe distinctions such as this in a way that does not require knowledge of the particular data structures used by an implementation. In order to do this we must speak only of the *behaviour* of the primitives in an arbitrary implementation.

It would appear there are essentially two sorts of behaviour to consider: the construction of proposition-like entities as ultimate semantic values, and the resolution of anaphors. Our intention, as outlined above, is to describe the correct range of these behaviours, so that incorrect implementations can be ruled out. (Thus we would expect the 'predicate-logic' implementation to fail the test for correct anaphoric behaviour, but for all the example implementations to pass that for correct proposition-like behaviour.)

Before proceeding, however, it is worth distinguishing between two separate enterprises in the semantic processing of anaphora. First, there is the task of providing a systematic map from sentences and sequences of sentences in which the antecedent of a pronominal anaphor is clearly indicated ('indexed discourses') to some semantic representation that exhibits the appropriate proposition-like behaviour. Second, there are theories of how that indexing is set up—of the process of anaphora resolution itself.[3] Kamp's DRT makes claims in both areas, although its primary importance is probably that it was the first theory to successfully tackle the former one, for a particular set of data. Subsequent systems, such as that of

[3]These issues are discussed at more length in §5.4.

Dynamic Predicate Logic (DPL, Groenendijk and Stokhof 1991), have tended to be concerned exclusively with this aspect of DRT.

I shall regard the first of these enterprises, that of generating semantic representations for indexed discourses, as within the domain of 'propositional' behaviour. Anaphora resolution itself, however, is clearly anaphoric behaviour.

But there is a lack of agreement on how anaphoric behaviour should be modelled. Anaphora resolution is still poorly understood, and lacks even one tolerably comprehensive formal model. For that reason, I devote most of the rest of this chapter to what I have called 'propositional-like' behaviour; a partial specification of anaphoric behaviour appears in §4.7.

There is not, of course, universal agreement on what constitutes correct behaviour of this kind for disambiguated English discourses. Traditionally, what I have been calling 'propositional-like' (in an attempt at neutrality) is rendered simply as 'truth conditional'. Since to depart from this level of analysis is to give up most of the present consensus I shall not do so, and merely use the word 'inferential' in preference to 'truth conditional', wishing to emphasize that inferential relations between discourses are the focus of discussion.

Even in this area there are certainly semantic theories that differ in the precise inferential behaviour they attribute to particular syntactic constructions. I will give two examples. First, it happens that differing notions of intensional identity give rise to differing entailment patterns for sentences involving propositional attitudes, as in:

$$\begin{array}{ll} & \text{Tom believes that John loves Mary.} \\ \hline \therefore & \text{Tom believes that John loves Mary and someone loves Mary.} \end{array}$$

Second, certain entailments, such as the following argument, depend on the degree of 'type looseness' in a semantic theory:

$$\begin{array}{ll} & \text{Everything that worries Tom worries Sue.} \\ & \text{Peter worries Tom. That John loves Mary worries Tom.} \\ \hline \therefore & \text{Peter worries Sue. That John loves Mary worries Sue.} \end{array}$$

Montague's PTQ theory (Montague 1973) validates the first argument but not the second, while the property theory PT_2 of Chierchia and Turner (1988) does exactly the opposite.

Despite this, there is a broad swathe of agreement across our current range of semantic theories. For the English data encompassed by Johnson and Kay's grammar, restricted to the bound-variable readings favoured by DRT, there is sufficient unanimity about the desired inferential behaviour that I now take that as given and proceed to consider how best to specify it for an arbitrary Johnson and Kay implementation.

4.5.2 Constraining inferential behaviour

In principle, we could describe our intended inferential behaviour merely by specifying a desired consequence relation over indexed English discourses. Then an implementation would be satisfactory just in case this relation could be defined on the external semantic representations which it generated for those discourses. However this approach has drawbacks. One concern is the degree of reliance on the grammar with which the implementation works: if that grammar's language does not include the premises and conclusions necessary to capture the inferential behaviour of a particular construction, then the specification will be incomplete. More fundamentally, there is the problem of ambiguity in plain English sentences. Various levels of supplementary notation may be used to resolve these ambiguities, but since they include questions of quantifier scope and other semantic notions, such notations inevitably tend towards a representation that reflects logical structure at least as much as surface syntax. The case for working with logical forms outright is strengthened by the tradition of using formulas of first-order logic to describe intended inferential behaviour (or truth conditions) for this particular range of English discourses. Finally, it seems reasonable to expect that the specification of a correct implementation of general 'semantic primitives' should not depend on a particular natural language.

I conclude that one approach worth pursuing is to constrain the inferential behaviour of a candidate implementation using a map from the implementation's external semantic representations to formulas of first-order logic. Then we can state constraints in terms of the output of this map, the map itself being supplied by a conforming implementation.

An external representation corresponds to a complete discourse. Hence in this approach, we take some representative set of discourses and assign a 'canonical' first-order formula to each of them. The derived constraint on an implementation (together with its associated first-order map) is then that the external representations it generates for each representative discourse must get mapped to the corresponding canonical formula (or to a logically equivalent one, say). We can dispense with the actual discourse and the syntactic processing, taking the collection of constraints generated by the grammar for that discourse as the starting point of the specification (figure 4.6 shows an implementation which automatically makes such collections).

Evidently there is an issue about just how representative the set of test discourses (or equivalent constraints) can be. Although the goal is to constrain the interpretation of semantic primitives as comprehensively as possible, the only combinations of primitives that can actually be tested in this approach are those that serve to constrain the external representation of a single grammatical discourse. In order to parse these discourses, the candidate implementation will have to be paired with a test grammar. The possibility cannot be excluded that some extension of the test grammar might combine the primitives in a novel way,

```
external(S:X-[external(S,E)], E:X).

atom(P, S:[atom(P,S)|X]-X).

conjoin(S1:X-Y, S2:Y-[conjoin(S1,S2,S3)|Z], S3:X-Z).

new_index(Index, S:[new_index(Index,S)|X]-X).

accessible_index(Index, S:[accessible_index(Index,S)|X]-X).

compose(S1:X-Y, S2:Y-[compose(S1,S2,S3)|Z], S3:X-Z).

subordinate(S1:X-[subordinate(S1,B,S2)|Y], B, S2:X-Y).
```

Figure 4.6: An implementation that collects constraints

still consistent with their intuitive interpretation, but not tested by any of the discourses that the original grammar admits.

Furthermore, no finite set of representative discourses really provides sufficient constraint on an implementation, since we would like to ensure that the semantic primitives are defined uniformly over the domain of 'lexical propositions' (lprop). For example, the implementation should do the right thing for all common nouns, not just *woman*, *man* and *donkey*. We can guarantee this only by including some form of quantification in our constraints, so that no particular members of the non-logical syntactic categories are singled out. This should not be problematic, however.

It appears that the main problem with this approach is its inability to specify the behaviour of each primitive independently, or in any combination other than those determined by particular discourses. Even in such a combination, all information about the effect of the primitives involved is filtered through the external primitive, since it is only via the first-order map on external representations that any constraints can be specified. Directly constraining the primitives, on the other hand, suggests a strategy in which *internal* representations are mapped to a common logical form language, with a corresponding 'canonical' relation on that language being defined for each primitive, and the form of the actual specifications including quantification over the domain of all internal representations (rep). In what follows I shall investigate both these approaches for the Johnson and Kay primitives. We may note in advance, however, that in the second approach we are not necessarily free to choose an arbitrary 'logical form language': it may transpire that an internal representation in this system essentially carries more information than, say, a formula of first-order logic.

4.6 Specifying correct implementations

There are various ways to investigate constraints for the primitives. One option is to assume that since we're interested only in inferential behaviour, Johnson and Kay's 'predicate-logic' implementation should be some kind of boundary case, where the implementations of the primitives only just satisfy our inferential constraints. If so, this is a good place to start.

4.6.1 Discourse representation languages

The 'predicate-logic' implementation does not distinguish between internal and external representations. I begin by taking the first approach to specification outlined above, which requires that we find a way to map these representations to first-order logic. As pointed out in §4.4, they are like first-order formulas which lack explicit quantifiers. However we know that we are expected to interpret

```
man(X) & donkey(Y) & beat(X,Y)
```

as

$$\exists x \exists y (\mathtt{man}(x) \land \mathtt{donkey}(y) \land \mathtt{beat}(x,y))$$

but to interpret

```
man(X) ==> (donkey(Y) & beat(X,Y))
```

as

$$\forall x (\mathtt{man}(x) \Rightarrow \exists y (\mathtt{donkey}(y) \land \mathtt{beat}(x,y)))$$

What's going on here? The answer seems to be that the 'predicate-logic' representations are really expressions in a kind of minimal discourse representation language that lacks explicit marking of discourse referents. By a 'discourse representation language' (or DR language) I mean some linearized version of Kamp's DRS notation. A good study of such languages, and how they relate to first-order logic, has been provided by Zeevat (1989). On page 101 of that article, Zeevat defines a linear form in which the appearance of a variable as a discourse referent (in the 'top box' of a DRS) is indicated with an asterisk. He also admits $-$ (absurdity) as a primitive DRS. In this language, the two representations just given would appear as follows.

$$\mathtt{man}(x^*) \land \mathtt{donkey}(y^*) \land \mathtt{beat}(x,y)$$
$$\mathtt{man}(x^*) \Rightarrow (\mathtt{donkey}(y^*) \land \mathtt{beat}(x,y))$$

Zeevat gives a model-theoretic interpretation for such expressions (in effect, extending Kamp's original DRT semantics to 'incomplete' DRSs) and shows that there exist maps between this DR language and first-order logic (FOL) which preserve satisfaction in each direction. The map to FOL, T_0, is defined with

$$T_0(\phi) = \exists x_1 \ldots \exists x_n\, T_1(\phi) \qquad \text{where } \{x_1, \ldots, x_n\} = V(\phi)$$

and

$$\begin{array}{l} T_1(r(t_1, \ldots, t_n)) = r(t'_1, \ldots, t'_n) \qquad \text{where } t'_i \text{ is } t_i \text{ with any star removed} \\ T_1(\phi \wedge \psi) = T_1(\phi) \wedge T_1(\psi) \\ T_1(\phi \Rightarrow \psi) = \forall x_1 \ldots \forall x_n (T_1(\phi) \Rightarrow \exists y_1 \ldots \exists y_m\, T_1(\psi)) \\ \qquad \text{where } \{x_1, \ldots, x_n\} = V(\phi) \text{ and } \{y_1, \ldots, y_m\} = V(\psi) \\ T_1(-) = - \\ \\ V(r(t_1, \ldots, t_n)) = \{\, x_i \mid x_i^* \text{ is one of } t_1, \ldots, t_n \,\} \\ V(\phi \wedge \psi) = V(\phi) \cup V(\psi) \\ V(\phi \Rightarrow \psi) = \emptyset \end{array}$$

The function V collects discourse markers at the current level, and T_1 uses this information to generate the right quantification for DR language implications. T_0 completes the translation by making sure that top level referents are existentially quantified.

Can we find a map from our minimal DR language to FOL that similarly preserves satisfaction? We would then have most of what we need in order to constrain the 'predicate-logic' implementation in the first way described in the previous section, since we'd have a map from (external) representations to FOL formulas.

In fact, there is such a map, despite the lack of explicit information about discourse referents in our minimal DR language. For it is sufficient to observe that in a well-formed DRS

1. There are no 'free variables'—every variable (referent) is intended to be quantified;
2. A variable should be existentially quantified unless
   (a) it is already in the scope of a quantifier, or
   (b) it appears in the antecedent of an implication, when it should be universally quantified, subject to the condition just given.

On this basis, we can define a function U_0 to translate our minimal DR language to FOL. The idea is to work top-down and use an auxiliary argument to the main translation function, U_1, to keep track of those variables which have already been quantified over. The only other real change required is to make Zeevat's function V behave as if every variable in its argument DRS was starred, so that it generates all 'candidate discourse referents' for it.

$$U_0(\phi) = \exists x_1 \ldots \exists x_n\, U_1(\phi, \{x_1, \ldots, x_n\}) \qquad \text{where } \{x_1, \ldots, x_n\} = W(\phi)$$

$$
\begin{aligned}
&U_1(r(x_1,\dots,x_n),A) = r(x_1,\dots,x_n)\\
&U_1(\phi\wedge\psi,A) = U_1(\phi,A)\wedge U_1(\psi,A)\\
&U_1(\phi\Rightarrow\psi,A) = \forall x_1\dots\forall x_n(U_1(\phi,B)\Rightarrow\exists y_1\dots\exists y_m\,U_1(\psi,C))\\
&\qquad\text{where } \{x_1,\dots,x_n\} = W(\phi)\setminus A,\quad B=\{x_1,\dots,x_n\}\cup A,\\
&\qquad\qquad\{y_1,\dots,y_m\} = W(\psi)\setminus B,\quad C=\{y_1,\dots,y_m\}\cup B\\
&U_1(-) = -
\end{aligned}
$$

$$
\begin{aligned}
&W(r(x_1,\dots,x_n)) = \{x_1,\dots,x_n\}\\
&W(\phi\wedge\psi) = W(\phi)\cup W(\psi)\\
&W(\phi\Rightarrow\psi) = \emptyset
\end{aligned}
$$

(A Prolog implementation of this map may be found in appendix B. Since writing this chapter I have discovered that Pagin and Westerståhl (1993) discuss a language analogous to my minimal DR language, and prove similar equivalences.)

4.6.2 An external specification

Now that we can map 'predicate-logic' representations to FOL, we may investigate the first of our approaches to specifying the correct behaviour of the primitives. We seek a semantically varied range of sentences admitted by the Johnson and Kay grammar that each give rise to as few constraints as possible.

One such sentence is

A woman owns a donkey.

The standard grammar generates a semantic representation for this sentence by finding a value of e that makes the following formula true:

$$
\begin{aligned}
&\exists x\exists y\exists r_1\dots\exists r_9\,\mathsf{new\_index}(x,r_1)\wedge\mathsf{atom}(\texttt{woman}(x),r_2)\wedge\mathsf{compose}(r_1,r_2,r_3)\\
&\quad\wedge\mathsf{new\_index}(y,r_4)\wedge\mathsf{atom}(\texttt{donkey}(y),r_5)\wedge\mathsf{compose}(r_4,r_5,r_6)\\
&\quad\wedge\mathsf{atom}(\texttt{own}(x,y),r_7)\wedge\mathsf{conjoin}(r_6,r_7,r_8)\wedge\mathsf{conjoin}(r_3,r_8,r_9)\\
&\quad\wedge\mathsf{external}(r_9,e)
\end{aligned}
$$

This collection of constraints exercises all the primitives except accessible_index and subordinate, and it is not possible to find a sentence whose semantic representation involves less primitives. However, if we are prepared to augment the original Johnson and Kay grammar with the following rules for intransitive verbs (and verb phrases), for which the semantic component seems clear, we can obtain a representation that at least depends on fewer instances of the primitives.

```
vp(VP) --> vi(VP).

vi(X^S) --> [Verb],
    { verbi(Verb, X^Pred),
    atom(Pred, S) }.

verbi(walks, X^walk(X)).
verbi(runs, X^run(X)).
```

With these additional rules the sentence

A woman walks.

may be parsed and yields the following constraints:

$$\begin{aligned}&\exists x \exists r_1 \ldots \exists r_5\, \mathsf{new\_index}(x, r_1) \wedge \mathsf{atom}(\mathtt{woman}(x), r_2) \wedge \mathsf{compose}(r_1, r_2, r_3)\\ &\quad \wedge \mathsf{atom}(\mathtt{walk}(x), r_4) \wedge \mathsf{conjoin}(r_3, r_4, r_5) \wedge \mathsf{external}(r_5, e)\end{aligned}$$

To test universal quantification we may take

Every woman walks.

which generates

$$\begin{aligned}&\exists x \exists n_1 \exists n_2 \exists r_1 \ldots \exists r_9\, \mathsf{new\_index}(x, r_1) \wedge \mathsf{atom}(\mathtt{woman}(x), r_2)\\ &\quad \wedge \mathsf{compose}(r_1, r_2, r_3) \wedge \mathsf{atom}(\mathtt{walk}(x), r_4) \wedge \mathsf{subordinate}(r_4, n_1, r_5)\\ &\quad \wedge \mathsf{compose}(r_3, r_5, r_6) \wedge \mathsf{subordinate}(r_6, n_2, r_7) \wedge \mathsf{atom}(n_2 \mathtt{==>} n_1, r_8)\\ &\quad \wedge \mathsf{compose}(r_7, r_8, r_9) \wedge \mathsf{external}(r_9, e)\end{aligned}$$

Using these formulas, the abbreviations cn and imp defined in §4.3, and our map U_0 we see that the following constraints hold of the 'predicate-logic' implementation:

$$\begin{aligned}&\forall p \forall q \forall e[(\exists x \exists r_1 \exists r_2 \exists r_3\, \mathsf{cn}(x, p, r_1) \wedge \mathsf{atom}(q, r_2) \wedge \mathsf{indexes}(q) = \{x\}\\ &\quad \wedge \mathsf{conjoin}(r_1, r_2, r_3) \wedge \mathsf{external}(r_3, e)) \Rightarrow U_0(e) = \ulcorner \exists x(p \wedge q) \urcorner]\end{aligned}$$

$$\begin{aligned}&\forall p \forall q \forall e[(\exists x \exists r_1 \exists r_2 \exists r_3\, \mathsf{cn}(x, p, r_1) \wedge \mathsf{atom}(q, r_2) \wedge \mathsf{indexes}(q) = \{x\}\\ &\quad \wedge \mathsf{imp}(r_1, r_2, r_3) \wedge \mathsf{external}(r_3, e)) \Rightarrow U_0(e) = \ulcorner \forall x(p \Rightarrow q) \urcorner]\end{aligned}$$

where p and q range over lprop. We could try to capture some constraints on the semantics of relative clauses with the additional requirement

$$\begin{aligned}&\forall p \forall q \forall u \forall e[(\exists x \exists r_1 \exists r_2 \ldots \exists r_5\, \mathsf{cn}(x, p, r_1) \wedge \mathsf{atom}(q, r_2) \wedge \mathsf{indexes}(q) = \{x\}\\ &\quad \wedge \mathsf{conjoin} r_1, r_2, r_3) \wedge \mathsf{atom}(u, r_4) \wedge \mathsf{indexes}(u) = \{x\}\\ &\quad \wedge \mathsf{conjoin}(r_3, r_4, r_5) \wedge \mathsf{external}(r_5, e)) \Rightarrow U_0(e) = \ulcorner \exists x(p \wedge q \wedge u) \urcorner]\end{aligned}$$

but this seems not to give us any additional leverage. To generalize away from the 'predicate-logic' implementation it now suffices to demand that every candidate implementation provides some map F from its external representations to FOL such that

$$\begin{aligned}&\forall p \forall q \forall e[(\exists x \exists r_1 \exists r_2 \exists r_3\, \mathsf{cn}(x, p, r_1) \wedge \mathsf{atom}(q, r_2) \wedge \mathsf{indexes}(q) = \{x\}\\ &\quad \wedge \mathsf{conjoin}(r_1, r_2, r_3) \wedge \mathsf{external}(r_3, e)) \Rightarrow F(e) = \ulcorner \exists x(p \wedge q) \urcorner]\end{aligned}$$

$$\begin{aligned}&\forall p \forall q \forall e[(\exists x \exists r_1 \exists r_2 \exists r_3\, \mathsf{cn}(x, p, r_1) \wedge \mathsf{atom}(q, r_2) \wedge \mathsf{indexes}(q) = \{x\}\\ &\quad \wedge \mathsf{imp}(r_1, r_2, r_3) \wedge \mathsf{external}(r_3, e)) \Rightarrow F(e) = \ulcorner \forall x(p \Rightarrow q) \urcorner]\end{aligned}$$

For 'predicate-logic', $F = U_0$. For the other two implementations, it turns out that the required map can be obtained by composing U_0 with a per-implementation map from external representations to the minimal DR-language that U_0 is defined on. Prolog definitions of these implementation-specific maps are provided in appendix B.

4.6.3 An internal specification

The 'external' specification just developed involves many instances of the semantic primitives, and in two very similar configurations. Yet it is clear that basing the specification on more complex sentences will only increase the complexity of the derived constraints in the same limited areas. In order to get a firmer grip on the behaviour of the individual primitives we need less complexity in our constraints and must resort to an approach based on translating *internal* representations into some common language, on which the canonical behaviour of the primitives may be defined.

Let's start again with the 'predicate-logic' implementation, where internal and external representations are the same, so that we can immediately use U_0 to map the former to FOL. Can we then constrain the primitives directly using this map?

We can do so only if we can capture the inferential behaviour of each primitive in terms of a relation on formulas of FOL. For the primitive atom, this is straightforward, at least for members of lprop. We evidently want to say just

$$\forall p \forall r(\mathsf{atom}(p, r) \Rightarrow U_0(r) = \ulcorner p \urcorner)$$

where p is quantified over lprop. What about conjoin? The meaning of conjoin certainly involves logical conjunction, so perhaps we can simply say

$$\forall r_1 \forall r_2 \forall r_3(\mathsf{conjoin}(r_1, r_2, r_3) \Rightarrow U_0(r_3) = \ulcorner U_0(r_1) \wedge U_0(r_2) \urcorner)$$

This is false, however. In interpreting *A woman walks*, the 'predicate-logic' implementation solves the constraint

$$\mathsf{conjoin}(r_1, r_2, r_3)$$

with the following assignments (and translations under U_0):

$$\begin{array}{ll} r_1 = \mathtt{woman}(x) & U_0(r_1) = \exists x\, \mathtt{woman}(x) \\ r_2 = \mathtt{walk}(x) & U_0(r_2) = \exists x\, \mathtt{walk}(x) \\ r_3 = \mathtt{woman}(x) \wedge \mathtt{walk}(x) & U_0(r_3) = \exists x(\mathtt{woman}(x) \wedge \mathtt{walk}(x)) \end{array}$$

which contradicts the proposed specification. But this should not come as a great surprise: it has already been noted that the representations of the 'predicate-logic' implementation may be seen as formulas of a minimal DR-language, in which the symbols "$\wedge$" and "$\Rightarrow$" do not have their FOL interpretation. None the less, the possibility remains that we may be able to mimic DR-conjunction (and implication) in FOL via syntactic operations on FOL formulas other than the the natural binary operations of FOL-conjunction and FOL-implication. In fact, we can do this.

Let S be that function on FOL formulas that strips out all quantifiers, but leaves everything else untouched. Then the following constraint on conjoin *does* hold for the 'predicate-logic' implementation:

$$\forall r_1 \forall r_2 \forall r_3(\mathsf{conjoin}(r_1, r_2, r_3) \Rightarrow U_0(r_3) = \ulcorner U_0(S(r_1) \wedge S(r_2)) \urcorner)$$

We can constrain imp, the set of primitives which implement DR-implication, in an analogous fashion:

$$\forall r_1 \forall r_2 \forall r_3 (\mathsf{imp}(r_1, r_2, r_3) \Rightarrow U_0(r_3) = \ulcorner U_0(S(r_1) \Rightarrow S(r_2))\urcorner)$$

The question is, is this a legitimate approach? I claim that it is not. As indicated above, these specifications only 'mimic' DR-conjunction and DR-implication. The crucial point about DR languages is that their formulas code up the semantics of first-order formulas *and* information about the identity of top-level existentially quantified variables (discourse referents, or indexes). The specifications above rely on interpreting FOL formulas in ways that are at odds with their standard semantics. In FOL, the formulas $\exists x\, \mathtt{woman}(x)$ and $\exists y\, \mathtt{woman}(y)$ have identical inferential content—but they will behave differently under the operations introduced above, since x and y are distinct variables. This means that the above approach breaks down as soon as one exploits some of the standard identities for FOL formulas. Imagine post-composing the map U_0 with a function that rewrites FOL formulas into a kind of canonical form by renaming all bound variables, and possibly performing other satisfaction-preserving operations. Then these specifications immediately fail.

To see why a primitive like conjoin must ultimately be interpreted as DR-conjunction, consider again the role it plays in the Johnson and Kay grammar. It is used in obtaining the semantic value of relative clauses and the indefinite article *a*. In the former case the underlying semantic operation is logical conjunction at the level of properties. The interpretation of *man that owns a donkey* is a property formed through the conjunction of the property corresponding to *man* and that corresponding to *that owns a donkey*. In the latter case, *a* is interpreted as a generalized quantifier that operates on properties to yield a proposition; the meaning of *a woman walks* is the (generalized) existential quantification of the properties corresponding to *woman* and to *walks*. But DR-languages contain no overt mechanism for defining or operating on properties. What they do have, however, is a way of connecting a formula with a set of variables. Thus it is that in a DR-language, properties (and relations of higher arity) must be encoded by using selected variables (in Johnson and Kay's terminology, indexes) to record an implicit abstraction operation. This is exactly what the Johnson and Kay grammar does. In order to implement property-level conjunction, then, it is necessary to record the abstracted-over variable (the grammar does this with the "^" operator), to ensure that both such variables are identical for the two properties (the grammar does this in the head of the rule that rewrites `n1` as a relative clause), and finally to ensure that the operation of conjunction itself respects the identities of variables occurring within the conjuncts. This latter is precisely what DR-conjunction buys us, and FOL-conjunction does not. The story is similar for the operations of generalized quantification; suffice it to point out that in the grammar the corresponding variable-identification constraint occurs in the rule for `det`. (Such observations about the nature of DR-language operations apply equally to Kamp's original DRT, and systems based thereon.)

The conclusion of these arguments is that any 'internal' specification of correct behaviour for the primitives must be stated in terms of a (minimal) DR-language, and hence that the map which a conforming implementation is required to provide must be a map from its internal representations to this DR-language. We may as well take the language of the 'predicate-logic' implementation's representations as our common, minimal, discourse referent-free DR-language. Then, finally, we can give a correct specification for the inferential behaviour of each primitive. In the following formulas, I use D to stand for the DR-language map provided by a conforming implementation.

First of all, we may stipulate that the domain of D can be extended to external representations in such a way that under D internal and external representations are identical. Both representations are the semantic values of clauses, sentences or discourses; the correct inferential behaviour (which is what D is supposed to extract) must surely be the same both internally and externally. Hence:

$$\forall r \forall e (\mathsf{external}(r, e) \Rightarrow D(r) = D(e))$$

We can resurrect essentially the same specification for atom as we mooted earlier. In this version I've made explicit the requirement that p be in lprop:

$$\forall p \forall r \forall n_1 \forall n_2 ((\mathsf{atom}(p, r) \wedge p \neq n_1 \texttt{==>} n_2) \Rightarrow D(r) = \ulcorner p \urcorner)$$

Now conjoin has the following natural specification:

$$\forall r_1 \forall r_2 \forall r_3 (\mathsf{conjoin}(r_1, r_2, r_3) \Rightarrow D(r_3) = \ulcorner D(r_1) \wedge D(r_2) \urcorner)$$

and similarly for imp

$$\forall r_1 \forall r_2 \forall r_3 (\mathsf{imp}(r_1, r_2, r_3) \Rightarrow D(r_3) = \ulcorner D(r_1) \Rightarrow D(r_2) \urcorner)$$

That deals with all the primitives that affect inferential behaviour. I have chosen not to give individual specifications for compose, subordinate and atom applied to a non-lexical proposition. This is because they do not have a fixed inferential interpretation across all implementations—only when combined as in the definition of imp do they convey a consistent meaning, namely that of DR-implication.[4]

[4]In both the 'predicate-logic' and the 'discourse-representation' implementations compose does not contribute to the inferential behaviour of imp, and for these implementations it would be possible to replace the specification of imp just given with, for example:

$$\forall r_1 \forall r_2 \forall n (\mathsf{subordinate}(r_1, n, r_2) \Rightarrow D(r_1) = D(n))$$

$$\forall n_1 \forall n_2 \forall r (\mathsf{atom}(n_1 \texttt{==>} n_2, r) \Rightarrow D(r) = \ulcorner D(n_1) \Rightarrow D(n_2) \urcorner)$$

The 'sets-of-infons' implementation breaks these constraints, because of its internal division of inferential content between lists of tagged infons and the tags (situation labels) themselves;

It remains to consider new_index and accessible_index. These primitives have no effect on inferential behaviour. We can capture this by demanding that D maps the representations that are constrained by these primitives into some tautologous DR formula, such as $- \Rightarrow -$:

$$\forall x \forall r((\mathsf{new\_index}(x, r) \vee \mathsf{accessible\_index}(x, r)) \Rightarrow D(r) = \ulcorner - \Rightarrow - \urcorner)$$

This is not quite the whole story, however: in the Johnson and Kay grammar these representations are always combined via compose; for new_index this happens in the collection of primitives that I've called cn, for accessible_index it happens in the grammar's pronoun rule. There are two ways to deal with this: either we retain the constraint just presented and give a general specification for compose too, or we treat specially the two contexts in which new_index and accessible_index occur.

Both approaches are possible. On the one hand, it would seem desirable to constrain the smallest possible unit, which argues for specifying compose individually. In this case, we may view compose simply as DR-conjunction. On the other hand, it seems more accurate to say that compose essentially has *no* inferential interpretation, and that therefore the right approach is to constrain cn and the combination of accessible_index and compose used in the pronoun rule with two separate specifications. Taking this latter approach we obtain:

$$\begin{aligned}&\forall x \forall p \forall r_1 \forall r_2 \forall r_3 \forall n_1 \forall n_2((\mathsf{new\_index}(x, r_1) \wedge \mathsf{atom}(p, r_2) \wedge p \neq n_1 \texttt{==>} n_2 \\ &\qquad \wedge \mathsf{compose}(r_1, r_2, r_3)) \Rightarrow D(r_3) = \ulcorner p \urcorner)\end{aligned}$$

$$\begin{aligned}&\forall x \forall r_1 \forall r_2 \forall r_3((\mathsf{accessible\_index}(x, r_2) \wedge \mathsf{compose}(r_1, r_2, r_3)) \\ &\qquad \Rightarrow D(r_1) = D(r_3))\end{aligned}$$

Since we arranged for D to map to the minimal DR-language from which the 'predicate-logic' implementation's representations are drawn, suitable implementations of D for the two anaphoric implementations already exist in appendix B (§B.1).

4.7 An anaphoric specification

Let us now suppose that we wish to develop a specification of the primitives which ensures their anaphoric behaviour is reasonable. This comes in three parts:

1. Requiring that certain primitives correctly record past discourse content;

it is only the latter which are used as the names of representations in subordinate, and the implementation relies on the operation of compose on 'superordinate' representations to keep the tagged infons available. (I note here that this structure does nothing to cash out the Johnson and Kay's claim (p. 18) that the 'sets-of-infons' implementation illustrates a "non-extensional" theory of meaning.)

2. Requiring that certain other primitives correctly determine possible antecedents from such content, and

3. Stipulating that the primitives not involved at all in anaphoric processing do not interfere with the first two tasks.

4.7.1 Using external representations

As in the inferential case, we should examine to what extent we can achieve our goals by looking only at an implementation's external representations. The picture here is bleaker, however: come the external representation, all decisions about anaphor resolution have already been taken, and we can only judge the result. Nonetheless, one might think that given a map from ext to first-order formulas (after the style of §4.6), then the external representation for

A donkey beats it.

would necessarily map to a FOL formula equivalent to

$$\exists x(\texttt{donkey}(x) \wedge \texttt{beat}(x, x))$$

—and that this effectively puts constraints on anaphoric behaviour. There are numerous problems here, though. Most obviously, the above example (with the intended reading) is not even grammatical, although it is admitted by the Johnson and Kay grammar. If we claim, on the other hand, that

Every man that owns a donkey beats it.

maps only to a single FOL formula, we conflate judgements on anaphoric processing with many others. It seems necessary to work at the level of internal representations.

4.7.2 An internal approach

Returning to three tasks listed above, let us first determine what notion of 'past discourse content' is required. We would like this content to be the minimum needed for the correct anaphoric behaviour of the primitives. The only primitive that actually extracts antecedents from past content is accessible_index, and what it extracts is an index. Hence it should be possible to treat 'past discourse content' as a set of indexes for the purposes of an anaphoric specification. We should, however, distinguish between the set of indexes set up as antecedents by a particular representation, and those that have been set up prior to that representation.

So let A and P be functions from internal representations to sets of indexes (that is, from rep to indexset). The informal interpretation of $A(r)$ is the set of antecedents (indexes) in r, and that of $P(r)$ is the set of indexes that have arisen in discourse prior to r. We can begin our specification by requiring that when an interpretation representation is related to an external representation by $\mathsf{external}$, then the internal representation can record no prior indexes:

$$\forall r(\exists e\, \mathsf{external}(r,e) \Rightarrow P(r) = \emptyset)$$

This will have the effect (given the constraints below) of ensuring that the representation of first constituent in a discourse will also see no prior indexes.

Taking the other primitives in turn, all we can say of atom is that it should introduce no new antecedents—and this holds whether atom's second argument is in Iprop or not:

$$\forall r(\exists p\, \mathsf{atom}(p,r) \Rightarrow A(r) = \emptyset)$$

Given A and P the constraints on $\mathsf{new\_index}$ and $\mathsf{accessible\_index}$ are obvious:

$$\forall x \forall r\ \mathsf{new\_index}(x,r) \Rightarrow A(r) = \{x\}$$

$$\forall x \forall r\ \mathsf{accessible\_index}(x,r) \Rightarrow (x \in P(r) \wedge A(r) = \emptyset)$$

It remains to regulate the conversion of current antecedents into prior ones, and the propagation of antecedents across 'subordinate' divisions. Both $\mathsf{conjoin}$ and $\mathsf{compose}$ perform the former activity, and it appears that they must have the same anaphoric specification. When $\mathsf{c}(r_1, r_2, r_3)$ holds of three representations, where c may be either $\mathsf{conjoin}$ or $\mathsf{compose}$, three things must hold:

1. The antecedents introduced by the combined representation, r_3, must be the union of those introduced in r_1 and r_2;

2. The prior antecedents of r_2, the second of the two components, must be the union of the prior antecedents of the first component, and the new antecedents introduced in it.

3. The prior antecedents of the combination r_3 must be exactly the prior antecedents of r_1;

So we have:

$$\begin{aligned}
&\forall r_1 \forall r_2 \forall r_3 (\mathsf{conjoin}(r_1,r_2,r_3) \vee \mathsf{compose}(r_1,r_2,r_3)) \Rightarrow \\
&\qquad (A(r_3) = A(r_1) \cup A(r_2) \\
&\qquad \wedge P(r_2) = P(r_1) \cup A(r_1) \\
&\qquad \wedge P(r_3) = P(r_1))
\end{aligned}$$

Regarding $\mathsf{subordinate}$, it seems that the crucial stipulation is that the prior antecedents of the superordinate and subordinate representation should be equal. We should also require that the superordinate representation created by $\mathsf{subordinate}$ not introduce any new antecedents. This gives us:

$$\forall r_1 \forall r_2 \forall n\, \mathsf{subordinate}(r_1,n,r_2) \Rightarrow (P(r_1) = P(r_2) \wedge A(r_2) = \emptyset)$$

It may be instructive to see what the implications of these constraints are for imp, the combination of primitives used to interpret implication and universal quantification. Suppose $\mathsf{imp}(r, s, t)$ holds. Then according to the definition of imp and the specification just given, we can deduce that

$$P(r) = P(t)$$
$$P(s) = P(r) \cup A(r)$$
$$A(t) = \emptyset$$

This says that the antecedents available to the 'restrictor' representation r are the same as those available to the combined representation t, that the antecedents for the 'scope' s include any antecedents introduced in r, and that the representation as a whole introduces no new antecedents, thereby correctly insulating the effect of any new indexes in r or s from subsequent discourse.

Prolog code that implements A and P for both the 'discourse-representation' implementation and for 'sets-of-infons' may be found in §B.2, appendix B.

4.8 Summary

The focus of this chapter has been the paper Johnson and Kay 1990, in which the authors present a model for semantic abstraction. I explained their system: it consists of a DCG grammar for a fragment of English, with some of the semantic processing factored into explicit Prolog calls, which themselves reduce to seven 'semantic primitives'. This provides a fixed interface to the syntax, but allows for alternative semantic domains via the possibility of different implementations (in Prolog) of the seven primitives. Johnson and Kay themselves provide three such implementations. The most convincing intuitive interpretations of the primitives lean heavily on DRT; the primitives cover both 'inferential' and 'anaphoric' operations, and take as arguments only propositional representations and 'indexes', these latter being analogous to DRT's discourse referents. The interface provided by the primitives is essentially relational, not functional.

I discussed the sample implementations. The 'predicate-logic' one generates minimal logical forms whose interpretation is not fully explained by Johnson and Kay. The 'discourse-representation' implementation constructs DRSs in a manner quite similar to the system of Johnson and Klein (1986), and relies crucially on the properties of difference lists in Prolog to integrate inferential and anaphoric information. The third implementation, 'sets-of-infons', is a notational variant of 'discourse-representation'.

In discussing approaches to modelling the abstraction inherent in the Johnson and Kay system, I argued for distinguishing between inferential and anaphoric behaviour, and further pointed out that there are essentially two approaches: 'internal' specifications that directly constrain the primitives, and 'external' ones that rely only on generated external representations. In the case of inferential behaviour, I showed that external specifications are possible, but unsatisfying.

Development of an internal specification revealed that a single 'minimal discourse representation' language can represent the core inferential content of the primitives, irrespective of implementation. Prolog code that connects this language with each of the sample implementations is collected in appendix B.

Regarding anaphoric behaviour, external specifications are even more inadequate, so I provided a simple internal specification for this too. Appropriate code is also given in the appendix.

In conclusion, note first that in their paper Johnson and Kay do not give any explicit characterization of the kind of semantic abstraction actually provided by their system, and it seems reasonable to desire such a characterization. The approach demonstrated in this chapter not only makes the nature of their abstraction clear, it makes it formal. I have shown that the inferential content of the system is that of a minimal DR language: only semantic theories that re-present or reinterpret DRT can be represented in this framework. Basically, what we have here is an elegant interface from the syntax to DRT.

This is disappointing: it seemed reasonable to hope for something more general. The difficulty of obtaining convincing intuitive explanations for the primitives independent of the DRT implementation turns out to have been an important indicator that the degree of true abstraction was limited, and this was exposed clearly by the abstract specifications developed later in the chapter. We conclude, then, of the Johnson and Kay work, that *this* particular approach to semantic abstraction does not form a satisfactory general basis.

Chapter 5

Towards a Specification of Quantification and Anaphora

5.1 Abstraction and modularity

We have seen how describing semantic theories with abstract specifications can classify them along particular axes of variation (chapter 3) and can help to reveal the underlying nature of their operations (chapter 4). In this chapter I want to investigate to what extent the same approach can provide *modular* specifications for a broad range of semantic theories. I do this by focussing on a specific area of semantic theorizing, and developing the outlines of a modular specification for it.

Modular specifications each address a particular aspect of semantic behaviour, and can be combined with one another so as to build up a series of interlocking constraints on a theory. My goal is to show how one might assemble a collection of such specifications with which to check whether any given theory of semantic interpretation (the object theory) meets common constraints on grammar and meaning. Where there is unanimity on the semantics of particular constructions, or particular operators, then we may have a single specification for them. Where there are currently a number of competing approaches, the aim is to provide a range of specifications accordingly.

This chapter inherits from my general programme the desire to abstract away as much of the presentational character of the object theory as possible. Theories choose different levels of description (often multiple levels at once), employ a great variety of logical frameworks and tend to either assume or define many notions of computational realization. These variations are to be ignored just to the extent that they are independent of the core semantic notions I aim to capture.

I describe first the general approach I adopt, which will be familiar in many respects from the results of the previous two chapters. Then I proceed to consider the topic of quantification and anaphora in detail. Finally, I comment briefly on how extensions to other major topics in semantic theory might work.

5.2 Strategies

In order to eliminate dependence on theory-specific formulations, my approach is to select, for each semantic notion, a minimal language of sorts and relations. These sorts and relations are abstract in the sense that they need not, and in general do not, correspond to any of the concrete sorts, type, relation or operations in particular object theories. Nor is it the case that these are in some sense 'ideal' primitives which object theories should adopt—they are merely the minimum required in order to state the relevant constraints.

The idea, then, is that given a new object theory it should be possible to identify these abstract sorts and relations as combinations of the concrete elements provided by the theory. Determining whether the object theory meets the specifications is a two-stage process: first, one must find some definition of the abstractions introduced below in the terms of the object theory; second one must attempt to show that the abstract specifications of behaviour are satisfied. It's clearly desirable to avoid an appeal to intuition in the first stage—so that any way of defining the abstract notions in terms of the object theory's concepts that formally satisfies certain constraints would suffice to determine whether the theory met the specifications. It remains to be seen whether this is achievable in practice.

Further distinctions can be made at the second stage, based on the kind of constraint that makes up the specifications. One obvious choice is between constraints that say, e.g., "the system can get the right semantics for pronouns outside the scope of a quantifier" and "the system *always* gets the right such semantics".

There are dependencies between modular semantic specifications. For example, some form of predicate logic with identity is assumed for most of the specifications. The description of predication relies on that of abstraction. It is not yet clear how best to formalize these dependencies.

I now move on to the semantic topic under specific investigation—quantification and anaphora—beginning with some discussion of anaphora.

5.3 Anaphora: resolution and interpretation

I shall divide the semantics of anaphora into two parts: **resolution**, the problem of determining the antecedent of an anaphor, and **interpretation**, the method by which anaphoric dependencies in discourses are interpreted. Some theories, such as DRT, address both aspects; more commonly, semantic theories of anaphora attempt only to model what I call interpretation. In this case, their starting point is a representation of sentences or discourses that indicates the resolved anaphoric dependencies, typically the indexed discourse. It is important to remember that the ambiguity associated with anaphora is not always eliminated by resolution alone. It is possible for an anaphor to be unambiguously dependent on a particular syntactic antecedent, and yet for that dependency to have multiple semantic interpretations. Indeed with some kinds of anaphora, such as verb phrase ellipsis, this is the prevalent form of ambiguity.

Anaphora resolution seems to to involve many factors, some of them certainly non-semantic. At the end of this chapter I consider briefly one much-discussed collection of syntactic constraints on resolution, those of binding theory.

In the main, however, my concern is with anaphora interpretation, a primarily semantic task. Unlike resolution, it is hard to describe in isolation from a particular semantic framework, owing to the interaction of anaphora with other semantic phenomena. The best studied of these interactions is that between anaphora and quantification.

5.4 Bound and unbound anaphors

For many years after the invention by Frege and others of bound variable quantification, logicians and linguists alike saw this mechanism as the natural model of anaphoric dependencies in natural language. So in Montague's PTQ, for example, all pronominal anaphora is analyzed as the binding of variables by quantifiers.

It has gradually become clear, however, that not all anaphoric connections have the semantic character of quantifier binding. Not only do some anaphors resist description as bound variables, but for many of those that do not the precise nature of their binding is hard to derive from syntactic realization and context. The debate about just which anaphors *are* to be interpreted as bound variables, and how to give a uniform semantics to discourses containing them whatever they are, is far from over, and it is not my place here to take sides on particular issues. Nonetheless, I believe it is possible to make some general observations about the interaction of quantification and anaphora that apply to every semantic theory which makes use of the two notions.

At this point I will state that I do not wish my analyses to be applicable not only to those theories which rely in their logical syntax on Fregean bound variables to perform quantification. I consider below how to avoid this assumption when formalizing my specifications. Therefore rather than use the term 'bound variable', I shall talk simply of anaphors that are interpreted as **bound** or **unbound** (by a quantifier), leaving open the precise mechanism by which quantification is expressed.

I should clarify my use of certain other terms. Anaphoric dependencies hold between **anaphors**[1] and their **antecedents**, even when the former precede the latter. Many kinds of expression can act as anaphors; in English, these include pronouns, noun phrases with the determiners *this* and *that*, and definite descriptions. An expression is **referential** when it refers to, or denotes, some actual entity. I use the term **quantifier** to mean an element in a logical language, reserving expressions such as **quantifier phrase** for constituents of natural language. I shall restrict my

[1] I do not use the technical GB sense of 'anaphor'.

attention to antecedents and quantifier phrases that are noun phrases in English.

I adopt here a strictly quantificational, or logical, sense of 'binding'. But there is an alternative notion going by the same name which should be carefully distinguished. Linguistic theory has a tradition of assigning indexes to noun phrase constituents in the syntax (and in transformationally-oriented theories, also to positions in syntactic structure that may be empty). One of the uses of such indexes is to indicate so-called binding relations between NPs. Generally, this relation depends both on the syntactic configuration of the NPs and on whether they have been **coindexed**, i.e., assigned the same index. Certain constraints that are based on this relation comprise binding theory; here, it is the relation of coindexing that is relevant, since these theories use coindexing to express anaphoric dependencies. (Though they do not thus represent *all* anaphoric dependencies; those that hold outside some notion of local syntactic domain—be they still intra-sentential—are generally referred to in the literature on binding theory as instances of discourse-level binding or coindexing, beyond the scope of the ordinary theory.)

Now the exact semantic import of this coindexing is a matter of dispute, one that is interwoven with the more general argument about whether particular anaphoric dependencies should be modelled with quantifier binding. It has been common to say that coindexing implies 'coreference'; where in practice that means either true coreference or quantifier binding, depending on the nature of the antecedent ("[We] are concerned rather with properties of LF-representation that enter into interpretations of sentences in terms of intended coreference ... where the 'reference' in question does not carry ontological commitment.", Chomsky 1981, p. 314 n. 2). On the other hand it has been claimed, notably by Lasnik (1976), that it is non-coindexing that carries semantic content, namely that non-coindexed constituents must have *disjoint* reference. Lasnik further argues that every instance of a pronoun coreferring with its antecedent (in the loose sense just noted) comes about through the pragmatic determination of pronoun reference, so that his rule of disjoint reference captures the entire semantic content of indexing. Evans (1980) has countered convincingly that Lasnik's rule must at least be modified to talk of *intended* disjoint reference, and further, that without some notion of intended *co*reference being indicated by coindexing (or some equivalent device) we are unable to explain how it is possible to interpret anaphorically-dependent pronouns in a pragmatically neutral context.

A few examples will serve to illustrate these notions, and my terms for them. I adopt Barwise's notation for representing anaphoric dependencies (Barwise 1987), in which the antecedent receives a superscript index and its anaphor the same index as subscript. Non-lexical constituents are indicated with brackets.

In a case like

John^i believes that he$_i$ is happy.

we may say that *he* is referential, and the truth conditions of this sentence, insofar as they concern the pronoun, are uncontroversial. (Although some have argued that pronouns are never referential, not even in a case like this—notably Geach (1962), whose analysis is that the property 'x believes that x is happy' is being predicated of John. I consider Evans (1977) to have refuted Geach's claim that the only use of pronouns is to construct such properties.)

Where a pronoun is anaphoric to a quantifier phrase such as *Every woman*, as here:

[Every woman]^i believes that she$_i$ is happy.

then the pronoun cannot be referential, and the truth conditions of the sentence are naturally represented with a quantifier for *Every woman* which binds the pronoun. Where a quantifier antecedent serves to pick out a single entity, as in a sentence like

[A certain woman]^i believes that she$_i$ is happy.

then the the pronoun is still bound, even if the sentence is used to assert something of an actual person. That indefinites do not refer is particularly clear when they occur in the scope of some non-existential quantifier, of course, as in Geach's donkey sentence (with Geach's *Any* replaced by *Every*):

(5.1) Every man that owns [a donkey]^i beats it$_i$.

It is the treatment of anaphors with quantifier antecedents, and the conditions under which they are to be interpreted as bound by them, that I wish to concentrate on. (Strictly, I should speak of the *interpretations* of anaphors being bound by the (quantifier) *interpretations* of their antecedents; this distinction will, predictably, become important in the formalization below.) The crucial observation that pronouns with quantifier antecedents are sometimes not bound by them was made by Karttunen (1969) and has been trenchantly analyzed by Evans (1977, 1980), who labelled them 'E-type' pronouns. Consider, for example,

(5.2) John owns [some sheep]^i and Harry vaccinates them$_i$.

On an analysis that has *them* bound by *some sheep*, this sentence is equivalent to

Some sheep are such that John owns them and Harry vaccinates them.

which may be one reading of (5.2), but is arguably not the most natural one. That reading, yielded by taking *them* to be an E-type pronoun, is

John owns some sheep and Harry vaccinates the sheep that John owns.

with the implication that Harry vaccinates *all* of the sheep that John owns.

E-type pronouns occur in many other syntactic contexts. Sells (1986) gives examples which include non-restrictive relative clauses, reflexive pronouns and prenominal modifiers. And it seems clear that one of the readings of the donkey sentence (5.1) is

Every man that owns a donkey beats the donkey that he owns.

which may only be captured with an E-type analysis. (I omit any discussion of the exact nature of the uniqueness requirement in the description that the E-type analysis gives rise to, since it is not relevant to my present purposes.)

Before moving on to consider what constraints may be said to operate in this territory, it may be as well to point out that the categories of 'referential', 'bound' and 'unbound' (or 'E-type') do not appear to exhaust the kinds of anaphoric connection that pronouns may have with their antecedents. For example, Geach used the phrase "pronouns of laziness" (Geach 1962, §76) to describe pronouns whose interpretation is as if their antecedents had been reproduced verbatim in their place. The following well-known example is from Karttunen (1969):

The man who gave [his paycheck]^i to his wife was wiser than the man who gave it$_i$ to his mistress.

It may be noted that Geach in fact interpreted as pronouns of laziness many (instances of) pronouns that I am here considering either referential or bound; see the just-mentioned section of his book for examples.

5.5 Constraints on anaphoric interpretation: scope

For the remainder of this chapter I consider only anaphors that have quantifier antecedents. When can such an anaphor be bound by its antecedent? It is evident that the **scope** of a quantifier antecedent, suitably construed, bears on this issue. Now there are various ways of construing scope. In the linguistics literature it has been popular to define a notion of scope in the terms of a syntactic theory, and to claim that this notion determines the underlying semantic scope of scope-creating operators in the syntax, such as quantifier NPs. There is disagreement over the right way to do this, and for my purposes I file this notion of scope along with the syntactic notion of binding mentioned above.

Ladusaw (1979) took the important step of asserting that the right notion of scope for a linguistic theory is one that is formally connected to the semantic interpretation of scope-forming constituents. It is in this spirit that I should like to promote a notion of **derivative syntactic scope**: derivative, that is, on underlying **semantic scope**. The latter notion is defined on entities at a level of semantic representation; the former is defined on syntactic constituents, but is ultimately parasitic on the latter. The idea is roughly that an expression whose semantic correlate is a scope-creating operator has derivative syntactic scope over those expressions whose semantic correlates are in the scope of that operator, as defined by the semantics.

I formalize this notion below, but some observations about bound anaphora and semantic scope follow immediately. For example, it becomes trivially true that every bound anaphor with a quantifier antecedent must be in the derivative syntactic scope of that antecedent; it doesn't make sense, or at least no sense yet discovered, to talk of binding outside the scope of a quantifier. Of course, some logical languages (such as DPL) have unusual rules for the scope of quantifiers, but it is precisely because this notion is based on the 'true' scope of semantic operators that the generalization just stated always holds. In (5.1), for example, on the DRT 'bound variable' reading

Every man that owns a donkey beats every donkey that he owns.

we will say that *it* is in the derivative syntactic scope of *a donkey*.

The constraint just given is trivial. But there is a non-trivial constraint close by that I believe does hold of every semantic theory that comprehends the relevant notions:

(A) If an anaphor with a quantifier antecedent is in the scope of that quantifier, then it is bound by it.

Cf. "An E-type pronoun evidently cannot have as its antecedent a quantifier with wider scope.", Evans 1977, p. 527. An E-type pronoun is not bound, so Evan's statement follows from (A) by *modus tollens*. The constraint (A) is more general just in that it extends Evans's claim to all non-bound anaphors.

What motivates this claim? For E-type pronouns, the case is quite easy to make, based upon their intended semantics. In Evans's words, "Roughly, the pronoun denotes those objects which *verify* (or that object which verifies) the sentence containing the quantifier antecedent." (ibid., p. 499). So the reference of an E-type pronoun is fixed by a clause that its quantifier antecedent occurs in—but exactly which clause this is needs to be pinned down. Later, Evans says that "the antecedent sentence ... is the smallest sentence which contains the quantifier and everything which it governs." (p. 535). (I ignore certain complexities associated with antecedent sentences when they form part of the restriction of a higher quantifier.) Evans goes on to show how a description may be extracted from this sentence which then serves to fix the denotation of the E-type pronoun in question. What is relevant here is Evans's use of "governs": the sense he has in mind is, I suspect, closer to the idea of derivative syntactic scope introduced above that to the notions of government present in modern syntactic theory, since what matters is that the antecedent sentence should include all the constraints on the quantifier expression that can play a part in the construction of a suitable description. In Evans's sketch of a formalization, all of (i) the main clause into which the antecedent quantifier is inserted, (ii) the common noun in the antecedent quantifier expression, and (iii) any relative clause restricting the quantifier go to make up the description that determines the reference of the E-type pronoun.

So anything in the derivative syntactic scope of an antecedent quantifier expression is potentially able to influence the reference of an E-type pronoun that is anaphorically dependent on it: this is why it is reasonable to stipulate that

E-type pronouns, at least, occur outside that scope—if they occurred within it, they might give rise to a circularity in the description that fixed their reference.

Despite this argument, the formalization of E-type pronoun semantics offered by Sells (1985a) appears to contradict the principle (A). I'll briefly present Sells's analysis, and attempt to show that, in fact, there is no contradiction.

Sells gives his semantic analyses in the notation of standard DRT, augmented with one extra device, the connective '$\to$', which I will come to in a moment. The sentence

(5.3) A man owns [a donkey]^i. It$_i$ grazes.

receives the following DRS:

(5.4)
$$\begin{array}{|l|}\hline x, y, z \\ \hline \texttt{man}(x) \\ \texttt{own}(x, y) \\ \texttt{donkey}(y) \\ \texttt{graze}(z) \\ z \to y \\ \hline \end{array}$$

On the reading in question, *it* is taken to be an E-type pronoun whose antecedent is *a donkey*. In (5.4), y is the discourse referent corresponding to *a donkey*. By the semantics of DRSs, y is interpreted as a existentially quantified variable with a scope that takes in the entire DRS. In particular, then, this scope embraces z, the discourse referent (also existentially quantified) that corresponds to the E-type pronoun *it*, in apparent contradiction of (A), and of Evans's statement quoted above.

But Sells does not intend those conditions of (5.4) which involve z to be interpreted by the normal rules for DRS conditions—indeed he could not, for that would guarantee an extensional treatment of the connective '$\to$' (irrespective of whatever that extension might be) and thereby modify the truth conditions of the first sentence of (5.3) since '$z \to y$' would constrain y as well as z.

So '$\to$' has to be given special treatment, and this is spelled out on pages 20–21 of Sells 1985a. Intuitively, '$y \to x$' means that y is an E-type anaphor whose reference is determined by the quantified variable x (Sells says that x and y are **cospecified**). Sells's idea is to give (5.4) an implicit binary structure, viewing the full DRS as the **anaphoric extension** of one that omits the conditions containing z.[2] Then if K' anaphorically extends K with (among others) the condition $y \to x$,

[2]Sells in fact talks only about omitting the condition whose connective is '$\to$'; this is insufficient, *pace* his example on p. 22, since any conditions left in the 'non-anaphoric' part that involve the E-type discourse referent will serve to generate the wrong description with which to fix the E-type reference (by affecting which functions verify that non-anaphoric part). This is not a matter of circularity, simply of unwanted existential commitments.

A further deficiency in Sells's definition is that for any DRS with two or more conditions involving '$\to$' there is no unique sub-DRS of which it is an anaphoric extension; such cases are not handled.

say that an embedding function g verifies K' if and only if for all f that verify K,

$$\forall a(a \in f(x) \Rightarrow a \in g(y))$$

Here Sells assumes that x and y are plural discourse referents, and his discussion implies that in this context singulars are to be treated as singleton sets.[3]

Returning to (5.4) in the light of this definition, it becomes apparent that in practice the scope of y only extends to those conditions in the DRS that exclude z; the special handling of the condition '$z \to y$' explicitly localizes the scope of y to exactly that sub-DRS. So I do not regard Sells's analysis as a counter-example to principle (A). Furthermore, were Sells's distinction between a sub-DRS and its anaphoric extension to be given some concrete realization in DRS syntax, it would serve to visibly limit the scope of y just as required by (A).

The only kind of non-bound anaphor considered by Evans and Sells is that which has 'E-type' semantics. I explained above the general case for arguing that an E-type pronoun cannot occur in the scope of its antecedent quantifier. But (A) talks about *all* non-bound anaphors, so the point must be made more generally. The core claim is that for an anaphor which is dependent on a quantifier phrase, the only value to being in the scope of the quantifier associated with that phrase is so that the anaphor may be bound by it. Which anaphors might offer counter-examples to this claim?

Non-bound non-E-type anaphors come in assorted flavours. One variety is the pronoun that can be given a 'purely lazy' interpretation, namely as standing for the syntactic repetition of its antecedent. So in the paycheck sentence

> The woman who gave [a lottery prize]^i to her husband was wiser than the woman who gave it$_i$ to her lover.

the lazy pronoun *it* depends on the quantifier phrase *a lottery prize*. Is the pronoun in the derivative syntactic scope of the quantifier phrase? Although it is certainly possible to make subsequent reference to the first woman's prize, such reference is surely always E-type, or non-restrictive—it would not affect the truth conditions of the original sentence. Thus it seems that an anaphor subsequently dependent on *a lottery prize* could not be bound by it, which lends support to the idea that *it* is not in the derivative syntactic scope of *a lottery prize*, as required by (A).

There are, however, other cases of anaphoric dependence involving a quantifier phrase that appear potentially troublesome. In

> Every new cognitive science student that hasn't yet met the rest of them should go to the seminar room at 4 pm.

[3]Under these assumptions, and for singular pronouns or plural pronouns with existentially quantified plural antecedents, Sells's definition does deliver the required E-type uniqueness conditions. For plural reference to non-existentially quantified antecedents the definition appears inadequate; for a start, it's not clear exactly which sub-DRS would qualify as that which is anaphorically extended. But this is not my main point, so I shall not pursue it here.

it seems possible, at least for some speakers, to regard *them* as anaphorically dependent on *Every new cognitive science student*. If this is really what is happening, then such examples do seem to contradict (A), since *them* is clearly not bound by the quantifier, yet is certainly contained in its derivative syntactic scope, given that it contributes to the restriction or range of the quantifier *Every* (these terms are discussed further below). These examples are somewhat marginal, and I have not made a detailed study of them, but I would suggest that (A) may not, in fact, be directly threatened by them, since it is unclear that the dependence of the pronoun *them* is on the entire quantifier phrase. One may construct similar sentences with *Most* in place of *Every* and yet still have the sense of *them* be ‘all new cognitive science students’.

On the basis of the discussion above, I shall maintain principle (A) as a constraint on all semantic theories that are capable of expressing—or implementing—the notions that it refers to. In order that this be a concrete claim, testable for an arbitrary theory, those notions must be made more precise. I now consider how to formalize the two central ideas: dependency and scope.

5.6 Formalizing anaphoric dependency

I talked earlier about the coindexing notation familiar from linguistic theory. This device indicates anaphoric dependency at a level of syntactic structure that is usually fairly close to unelaborated surface structure. For my purposes, it seems best to abstract away from the additional notions (such as binding) that various theories have associated with coindexing and instead take as primitive a relation of anaphoric dependency that may hold of any two syntactic constituents. The notion of ‘syntactic constituent’ is to be provided by the syntactic component of whatever object theory is under consideration.

Formally, then, I introduce a relation depend. In order to make relativizing to particular discourses straightforward, I take this to be a ternary relation $\mathsf{depend}(c_1, c_2, d)$ that holds just in case c_1 and c_2 are constituents of discourse d (a discourse may be a single sentence, of course) and c_2 is anaphorically dependent on c_1. I make the assumption that the traditional notion of coindexing coincides with my relation of dependency, except that I do not take my relation to be symmetric. I.e., I shall assume

$$\forall c_1 \forall c_2 \forall d\, \mathsf{coindex}(c_1, c_2, d) \Leftrightarrow (\mathsf{depend}(c_1, c_2, d) \vee \mathsf{depend}(c_2, c_1, d))$$

This relation receives no further formal analysis here, since establishing anaphoric dependency is not the concern of anaphoric interpretation, on my view, but rather of anaphora resolution. It will be as well, however, to point out some possible shortcomings of this model of dependency before moving on.

One area that demands more consideration than I am able to give it here is plural anaphora. It is common for a plural anaphor to anaphorically depend on multiple distinct constituents, as in

John^i met Mary^i at the station. They$_i$ caught a bus home.

I have so far said nothing about depend that would rule out *They* depending both on *John* and on *Mary*. But in order to avoid dealing with the construction of plural entities in the semantics, I shall assume in what follows that the first argument of depend is a function of the second, i.e., that

$$\forall c_1 \forall c_2 \forall d(\mathsf{depend}(c_1, c_2, d) \wedge \mathsf{depend}(c_3, c_2, d)) \Rightarrow c_1 = c_3$$

A deeper challenge to this formalization comes from those anaphors, typically anaphoric uses of definite descriptions, whose interpretation may be dependent either on a syntactic antecedent or on general discourse context, with this latter possibly having no direct representation in the syntax at all. Because such anaphors display a continuous range of usage between the two extremes, it is hard to know at what point to give up on a purely syntactic model of anaphoric dependency, such as my relation depend. One intermediate position is that of relational uses of definite descriptions, such as

John bought a new car. He had to replace the engine.

where *the engine* is intended to refer to the engine of the new car that John bought. Should this kind of relationship be indicated with a device like depend? For the purposes of my analysis, I choose to draw the line at anaphoric uses whose interpretation requires any information not directly present in the semantic representation of discourse; thus I do not expect depend to hold between, say, *a new car* and *the engine* in the last example.

There is one additional issue relevant to this discussion. Pronouns, too, need not derive their interpretation anaphorically, in the sense of being directly dependent on some, generally prior, syntactic constituent. It is common for their reference to be fixed demonstratively, but it is also possible for pronouns to refer to some entity made salient by the linguistic context, without that entity being the referent of any expression. An example might be:

A: Mary has got married again, you know.
B: Really? Who is he?

It has been argued that such 'pragmatic' pronouns introduce ambiguity into sentences that might otherwise seem unambiguous, and furthermore, that this ambiguity plays a role in the explanation of certain other phenomena, most notably verb phrase ellipsis (see, e.g., Partee and Bach 1981, and references therein). So it is claimed that in

John believed that he would win the case, and so did Bill.

the source of the strict/sloppy ambiguity (whether Bill believes that John or Bill would win the case) is a parallel ambiguity in the first clause. If *he* is anaphorically dependent on *John*, then the property picked up by the ellipsis is 'x believes that x will win the case'. Whereas if *he* is a pragmatic pronoun that just happens to refer to John, then the property is 'x believes that John will win the case'.

It happens that some recent work on VP ellipsis (Dalrymple, Shieber and Pereira 1991) locates the strict/sloppy distinction elsewhere, and does not require that the first clause be ambiguous. In any case, what is important for my purposes is that no *truth-conditional* ambiguity is introduced by the possibility of a pragmatic pronoun in the example just given—hence I can assume a single semantic representation, in my terms. This does not, of course, mean that my specifications cannot apply to those theories which choose to recognize an ambiguity at some non-truth-conditional level, even where this has a representation in the model theory.

5.7 Formalizing quantifier scope

In order that principle (A) can be tested for an arbitrary object theory it remains to formalize my notion of derivative syntactic scope. I shall express this via a relation syn-scope, to be true of two syntactic constituents just in case the first one is a quantifier expression whose (actual semantic) scope includes the semantic correlate of the second. There are a number of difficulties to be overcome if this relation is to be definable on a broad range of theories. The chief of these are:

1. Dealing with scope ambiguity;
2. Presenting quantifier scope relations in the semantics in a uniform way;
3. Connecting quantifier expressions with quantifiers;
4. Connecting arbitrary constituents with scope domains.

I consider them in turn.

5.7.1 Scope ambiguity

It would be possible to make syn-scope a ternary relation like depend, saying that syn-scope(c_1, c_2, d) holds just when c_1 and c_2 stand in an appropriate quantifier scope relation to each other as constituents of d, under some scoping of the quantifiers in d. This approach makes scope ambiguity implicit. But it has the disadvantage that one cannot then hold fixed a particular scoping of a discourse and state additional constraints that hold only per fully-scoped interpretation.

Thus I choose to augment syn-scope with an additional argument that stands for a scoped semantic representation, or meaning, so that syn-scope(c_1, c_2, d, m) is to be true just in case m is an interpretation of d, c_1 and c_2 are both constituents of d, and (in a sense to be made precise) the semantic correlate of c_2 is in the scope of the semantic correlate of c_1. (Chapter 3 of Ladusaw 1979 comes to the same conclusion regarding the incorporation of an interpretation argument into the scope relation.)

5.7.2 Uniform presentation of scope relations

At the level of semantic representation, quantifiers can be incorporated into a logical language in a variety of ways. Traditional first-order predicate logic uses the quantifier prefixes $\forall x$ and $\exists x$ to bind the variable x in an open formula $\phi(x)$. Logically equivalent variable-free languages (usefully surveyed in Quine 1976, though see also Grandy 1976 and Cresswell 1990) tend to view quantifiers as operators on syntactic expressions that correspond to predicates or relations, with the application of a quantifier reducing by one the arity of a relation, e.g., $E\phi$.

Natural language, of course, possesses quantifier expressions whose semantics cannot be captured with only the first-order quantifiers (given no additional theory). Such **generalized quantifiers**[4] have the property that they form a proposition by operating on two predicate expressions, rather than one. For variable-free languages, this extension is easily managed, since there must already be provision for binary operators (the truth-functional ones) on predicate expressions; for the traditional kind, it's necessary to add an additional kind of syntactic construction, such as $\mathsf{most}(x, \phi(x), \psi(x))$. If the language has operators of intensional abstraction, on the other hand, then adding generalized quantifiers can be more like the variable-free case: $\mathsf{most}(\lambda x \phi(x), \lambda y \psi(y))$.

In all these cases, a quantifier takes the form either of an operator or a connective, with arity varying from one to three. I shall assume that for an arbitrary object theory the quantificational import of its semantic representations can be expressed via a mapping to a logical language that has one of these forms of quantifier representation. It is for this language, then, that the notion of semantic scope is defined: the semantic scope of a quantifier is simply all of the expressions that it applies to, be that as an operator or a connective. The actual structure of these expressions will vary from object theory to object theory; and I wish to avoid having to know what this structure is, wherever possible. So the relation sem-scope is to be defined for arbitrary symbol instances of the object theory's quantificational language. And as for syn-scope, we must relativize to the particular reading, or scoping, in question. Thus we obtain a relation $\mathsf{sem\text{-}scope}(q, s, m)$, intended to be true just in case the interpretation m maps to a formula of the quantificational language in which q is a quantifier instance and s is a symbol instance that occurs as a substring of one of q's arguments.

A consequence of this definition is that the conventional notions 'restriction'

[4]I follow the usage (contrary to that of Barwise and Cooper 1981) that regards the semantic correlate of, e.g., *most* as a quantifier. The word *most* itself may be called a quantificational determiner.

and 'scope' for a natural language generalized quantifier are subsumed into one domain for the purposes of sem-scope. This is intentional: the arguments above in defence of principle (A) apply equally to the restriction and scope of a quantificational determiner.

5.7.3 Connecting quantifier expressions with quantifiers

From what I have said already, syn-scope(c_1, c_2, d, m) can only hold when c_1 is a quantifier noun phrase. I require an object theory to provide the means for identifying such constituents; this is not a problem for any theory that I know of. More problematic is ensuring that we can connect such a constituent with a quantifier operator or connective in the language postulated above. It is sufficient to obtain a connection between a quantifier NP and a quantifier at the level of the object theory's semantic representation, but this is not always immediately available. While it is true that a meaningful quantifier phrase in the syntax must give rise to a corresponding quantifier at some level of the semantics, processes may intervene to render the connection weak, and possibly to eliminate it altogether in the final representation. For a simple example of the latter, consider the analysis of

Bill is a man.

in Montague's PTQ. The logical form given by Montague is

$$\mathbf{man}'_*(b)$$

in which no quantifier is present. All that has happened is that (in additional to the normal lambda-reductions) some equational reasoning has been applied to eliminate the existential quantifier associated with *a man*. But given my approach, this makes it impossible to determine the scope of *a man*. So it's necessary for me to stipulate that the object theory provide access to semantic representations before conversions that might eliminate quantifiers have been performed on them. For PTQ at least, this requires no changes to the formulation of the theory.

5.7.4 Connecting constituents with scope domains

The only problem that remains before we can claim to have shown how syn-scope may be defined for any object theory is to obtain, for a given discourse interpretation, a connection between an arbitrary constituent of the discourse and a symbol instance in the quantificational formula corresponding to the discourse's semantic representation. We certainly cannot expect to be able to do this for *every* constituent of the discourse—some obvious candidates for which there may be no such connection are dummy elements like the semantically empty *It* of *It's raining*. But in order to make syn-scope capable of expressing the relevant part of principle (A) we do need to be able to do this for all NP anaphors.

The problem is that we cannot assume for every semantic theory that NPs will always acquire some concrete presence—as a symbol, or string of symbols—in that theory's semantic representations. Some of the variable-free logical languages described above simply do not possess the category of singular term, and even those that do will still have no explicit representation for bound pronouns. So it is necessary to take a more indirect route towards the scope domain corresponding to an arbitrary anaphor.

My strategy is first to observe this: the purpose of using an anaphor is to say something about whatever it is that the anaphor stands for. The 'saying something about' is invariably expressed in semantic representations as the use of a predicate or relation, and it is a matter of logic that a predicate and its argument cannot be in different scope domains. Furthermore, it is reasonable to require some explicit semantic representation for every predicate recognized by the object theory. So I propose to establish the scope domain for an anaphor in terms of that for the semantic entity that is predicated of it. The only remaining difficulty is to find a way of reaching the syntactic correlate of that semantic entity from the syntactic location of the anaphor. My claim at this point is that every anaphor is a syntactic argument (complement) of a constituent (its 'immediate syntactic functor') that must have some concrete semantic representation.

Finding this representation is not necessarily straightforward, however. The easy, common case is when the anaphor is the syntactic argument of a open-class lexical head that receives a uniform, concrete semantic interpretation. The difficult cases are when the anaphor's immediate syntactic functor

1. has a syncategorematic interpretation;
2. has a purely syntactic function (is semantically empty);
3. is non-lexical.

An example of (1) is the special treatment of the copula in PTQ (and many other semantic theories): *be* is interpreted not as $\mathbf{be}'_*$, or even as '=', but rather as $\lambda\mathcal{P}\lambda x\,{}^{\vee}\mathcal{P}({}^{\wedge}\lambda y({}^{\vee}x = {}^{\vee}y))$. A typical case of (2) is subcategorized-for prepositions of oblique arguments to verbs. One place that (3) crops up is in combinatory categorial grammar (CCG; see, e.g., Steedman 1988), where an auxiliary VP such as *may wash* is a syntactic functor. In each of these cases, the immediate syntactic functor of an NP does not correspond to the NP's immediate functor, or predicated-of relation, in the semantics.

These difficulties can be overcome if we are prepared to accept that an anaphor's scope domain may have to be determined via a predicate or relation that is not necessarily the immediate predicating entity for the anaphor's interpretation. So (1) and (2) can be dealt with by stipulation: these cases are easy to recognize in a grammar, and we can simply require that in the case of (1), *some* concrete semantic element is associated with the functor (this is simply '=' for PTQ) and that in (2) semantically empty functors don't count, so we look in turn at the immediate syntactic functor of the anaphor's immediate syntactic functor, and so on. For (3), it is sufficient to find any constituent of the non-lexical functor that has a concrete semantic interpretation, and use that to determine the scope—in *may wash him*, the semantic correlates of both verbs must be in the same scope domain as *him*.

In summary, we look for the smallest syntactic constituent containing the anaphor in question that has some concrete semantic presence—not necessarily a well-formed expression of the semantic representation language, merely a symbol instance—and follow that instance through to the quantificational language discussed above, where we use sem-scope to determine whether or not the anaphor is in the scope of the relevant quantifier expression.

5.8 Specifications for other areas

My attempts to develop modular semantic specifications have so far been confined to the domain of quantification and anaphora. However I envisage extending the approach to many other areas. Obvious candidates, given the work described above, are areas such as generalized quantification (with capturing the constraints identified in Barwise and Cooper 1981 being a natural aim) and binding theory, in the sense of syntactic constraints on anaphoric dependence. I include here some preliminary thoughts on the latter topic.

5.8.1 Anaphora and binding theory

That collection of constraints on anaphora that go by the name 'binding theory' are essentially constraints on anaphora resolution.

As a collection of principles, binding theory originates with Chomsky's GB theory, and finds alternative expression in LFG (Kaplan and Bresnan 1982; for a good summary of LFG binding theory see Sells 1985b) and in HPSG.

Here is a first stab at representing the principles of binding theory:

(A) $\mathsf{reflexive\_reciprocal}(x) \wedge \mathsf{constituent}(x,s)$
$\quad \Rightarrow \forall d(\mathsf{binding\_domain}(x,s,d) \Rightarrow \exists x'\, \mathsf{bind}(x',x,d))$

(B) $\mathsf{personal\_pronoun}(x) \wedge \mathsf{constituent}(x,s)$
$\quad \Rightarrow \forall d(\mathsf{binding\_domain}(x,s,d) \Rightarrow \neg\exists x'\, \mathsf{bind}(x',x,d))$

(C) $\mathsf{non\_anaphor}(x) \wedge \mathsf{constituent}(x,s)$
$\quad \Rightarrow \neg\exists x'\, \mathsf{bind}(x',x,s)$

Then we can express the actual constraints of GB, LFG and HPSG as follows:

GB Here $\mathsf{binding\_domain}(x,s,d)$ is true just in case d is a governing category for the syntactic object x (with both x and d contained in the sentence s). The relation $\mathsf{bind}(x, x', d)$ holds when x is in an A-position, x c-commands x', x and x' are coindexed, and both x and x' are contained in d.

We can define reflexive_reciprocal, personal_pronoun and non_anaphor in terms of the standard GB binary features 'anaphoric' and 'pronominal'.

LFG Here $\mathsf{binding\_domain}(x, s, d)$ holds if d is an appropriate **nucleus** for x, where this notion is somewhat language-dependent. Similarly, $\mathsf{bind}(x, x', d)$ is true if x is the antecedent of x' in d, but there may also be stipulations about x being a subject, or f-commanding x'.

HPSG In this theory, $\mathsf{binding\_domain}(x, s, d)$ holds if d is a SUBCAT list on which x occurs. The relation $\mathsf{bind}(x, x', d)$ breaks down into two cases: (i) if d is a SUBCAT list, then the relation is true if x occurs before (is less oblique than) x' on that list; (ii) else if d is a sentence, then the relation is true just in case there exist x'' and SUBCAT list d' such that $\mathsf{bind}(x, x'', d')$ and $\mathsf{dominate}(x'', x', d)$.

This is no more than a sketch, but I hope it indicates one way in which the approach I have been arguing for could be extended.

5.9 Summary

In this chapter I have promoted the idea of modular semantic specifications by saying how they should work, describing a strategy for creating them, and applying this strategy to the particular area of quantification and anaphora.

In considering this domain I started by carefully distinguishing the notions of resolution and interpretation, of binding, coindexing and dependence, and I determined that the natural place for a theory that intentionally abstracts away from details of particular theories of quantification and anaphora to make a substantive claim about their interaction is in the combination of derivative syntactic quantifier scope and anaphoric dependence. I formulated principle (A) relating these notions, defended it empirically, and showed how we might arrive at precise notions of scope and dependence that allow (A) to hold for every extant theory that addresses quantification and anaphora.

To what end? Well, I anticipated some clarification of the notions involved in quantification and anaphora when I embarked on this work, and I believe I've succeeded in that goal. It is very easy, when following the literature in this area, to focus on interactions between the two phenomena that are consequences solely of the theories involved, and which disappear in rival, though perhaps not vastly different, accounts. It is now clear that there are notions of dependence and scope which have validity across all theories, and I believe this work makes it easier to categorize aspects of object theories according to whether they are irrelevant to the interaction between quantification and anaphora, relevant to it in isolation, or relevant but incapable of formalization without consideration of related theory.

What was to some extent unexpected was the particular notions which ended up being relevant, especially that of derivative syntactic scope. The principle (A) was also something that arose out of my investigations, and which appears to capture a worthwhile generalization. Now it is certainly possible that (A) is false, in the sense that some object theory of quantification and anaphora may falsify it, but should that occur I believe the *way* in which it is falsified will shed important light on the theory in question, and that is one of my overriding aims.

Finally, although I have so far only pursued this kind of modular specification in the case of quantification and anaphora, I do not see anything in the approach that restricts it to that domain. I give some evidence of this by looking very briefly at binding theory in a similar way. Indeed, it is worth pointing out that I see potential for this kind of metatheorizing in more overtly logical areas: chapter 3 has already looked at abstraction and predication (where from a modular perspective we will surely find some interesting interfaces with generalized quantification), and I would like to apply this approach to the question of whether it's possible to formally specify something about predicate logic alone—saying nothing about abstraction or the predication of abstracts—that holds of both first-order systems and higher-order ones such as IL.

Chapter 6

Conclusions and Prospects

6.1 Review

I began by clarifying what I meant by “formal semantics”, and on what basis I chose to compare semantic theories. This basis is entailment: for the purposes of this work, a semantic theory must generate semantic representations over which it is possible to define a consequence relation.

In chapter 2 I attempted to answer the question: why should we abstract over semantic theories? Two answers emerged: first, so as to obtain more precise comparisons of and classifications between semantic theories, and second, in order to address the lack of functional modularity (and type-based security) in existing approaches. The second response applies notably to contemporary unification-based formalisms, which while being able to state constraints across both syntactic and semantic domains, do so in the absence of true modular structure and an abstract syntax-semantics interface. To some extent, this response is simply the application of design principles from software engineering to the construction of systems for semantic interpretation.

The second part of chapter 2 analyzed the various forms that metatheoretical statements about semantic theories might take. The primary result is a four-way classification of approaches to semantic abstraction, based on whether the semantic realm itself, or the syntax-semantics interface, is being abstracted over. I opted for abstraction over both, in search of maximum generality. Two key further options were identified: interpretation of metatheory in object theory, or vice versa. The latter option has the attraction of unifying all semantic notions in a ‘universal’ language, but owing to scepticism about finding a truly explanatory such language (as opposed to a mere disjunction of existing ones), I concluded that the first option should be promoted.

This classificatory work was able to categorize the approach of Johnson and Kay (1990) as abstracting over the semantic domain but not the interface, and in chapter 4 I analyzed that paper in some detail. I’ll consider now what we learned about that kind of semantic abstraction from my analysis.

Most importantly, we discovered that Johnson and Kay's framework in fact captures only a very limited kind of abstraction over semantic theories. Every implementation of the Johnson and Kay semantic primitives that 'makes inferential sense' is just a notational variant of Kamp's discourse representation language (or more accurately, of the minimal DR-language defined in §4.6.1) in the sense made precise by the specification of §4.6.3. There is more freedom in the anaphoric component of a Johnson and Kay implementation to depart from DRT behaviour, but the primitives are clearly designed with this behaviour in mind. Furthermore, recall that the semantic primitives must be used in a number of special combinations in order that the inferential and anaphoric elements shall interact appropriately. These combinations are actually established by the way the primitives are embedded in the syntax—as extensions to grammar rules—so it is the case that the syntax is forced to know more about the intended interpretation of the primitives than is desirable.[1]

I regard these results about the Johnson and Kay system as lending support to my decision to adopt a more general approach to semantic abstraction. Chapter 3 applied this approach to just two semantic notions, those of predication and intensional abstraction, but did so for three semantic theories that take very definite positions on the correct treatment of these notions and use quite different formalisms to do so. The outcome was the generation of formal statements of these alternative positions that are in themselves well known, but have not previously to my knowledge been given precise theory-neutral formulations.

Finally, in chapter 5, I speculated about a programme that would embrace all the major areas of semantic theorizing in a modular fashion. I chose to demonstrate this approach for the topic of quantification and anaphora, which does not have a naturally modular character, and so constitutes a revealing test. My results were limited, but went some way in developing consistent metatheoretical notions—such as anaphoric dependence and derivative syntactic scope—that I was able to interpret across semantic theories with apparently incompatible approaches to semantic representation. I obviously cannot claim to have captured all cross-theoretical constraints on quantification and anaphora, but I believe I have shown that non-trivial constraints can be expressed in a formal framework that is applicable to a very wide range of semantic theories. Furthermore, I hope to have illuminated the *kind* of theorizing that will be necessary to sustain truly theory-independent semantic modules.

6.2 Future work

This thesis goes only a little way towards the goal of fully modular reusable semantic specifications. Continuing the theme of chapter 5, I hope in future work to address other topics within formal semantics much more thoroughly, and to further

[1] I owe this observation to unpublished work by Millies and Pinkal.

develop the detailed modelling of semantic formalisms and the syntax-semantics interface.

The approach I have adopted offers generality at the price of no immediate computational interpretation. Tackling this latter deficit is a prime candidate for future research; if we could find a mode of semantic abstraction that offers the combination of sufficient descriptive power and tractable computational properties we would have a powerful tool indeed.

But for now, given the state of semantic theorizing, I prefer to side with generality. By using formal specifications tailored for specific areas of semantics we need neither impose nor propose a single semantic formalism into which all theories are supposed to be translatable. I regard this as a valuable methodological position in a world where consensus on the right semantic theory for natural language still seems a long way off.

Appendix A

Implementing the Russell's property specification

The essence of the two implementations of RUSSELL is contained in the definition of the ML functors `BealerToRussell` and `ChierchiaTurnerToRussell`. What really matter here are the definitions of abs and pred. For the latter, we just adopt the definitions given at the end of the previous section, but we need to take a little more care with abs in the case of PT_2. In order that the subsort relations hold, it is essential that the sort Var becomes the sort of variables of type *nf* in PT_2. But PT_2 only allows lambda-abstraction over variables of type *e*. Hence the actual definition of abs must be

$$\mathsf{abs}(\alpha, \beta) = {}^{\cap}\lambda x_e(\exists \alpha.\, \alpha = x_e \wedge \beta)$$

The last three lines of the following code construct the Russell property in the three cases of the minimal logic itself, the Bealer implementation, and the Chierchia and Turner implementation. After executing these, the following dialogue with the system causes it to display the strings corresponding to the Russell property in each case (lines beginning with the prompt '-' are user input):

```
- MR.rstr;
val it = "abs(x,not(pred(x,x)))" : string
- BR.rstr;
val it = "[~(x @ x)]_x" : string
- CTR.rstr;
val it = "^L x_e[((E x_nf. (x_nf = x_e)) & ~!%x_nf(x_nf))]"
  : string
```

```
(* The RUSSELL specification -- really: just the syntax *)

signature RUSSELL =
    sig
        type Var
        type Ent
        type Form
        type Pty
        val varId : string -> Var
        val entVar : Var -> Ent
        val ptyVar : Var -> Pty
        val not : Form -> Form
        val abs : Var * Form -> Pty
        val pred : Pty * Ent -> Form
        structure print :
            sig
                val var : Var -> string
                val ent : Ent -> string
                val form : Form -> string
                val pty : Pty -> string
            end
    end

(* With something providing the RUSSELL syntax, construct Russell's property *)

functor MakeRussell(R : RUSSELL) =
    struct
        val r =
            let
                val x = R.varId "x";
            in
                R.abs(x,R.not(R.pred(R.ptyVar x,R.entVar x)))
            end
        val rstr = R.print.pty r
    end

(* Provide the RUSSELL syntax directly *)

structure  MinimalRussell : RUSSELL =
    struct
        datatype Var = VarId of string
        datatype Ent = EntVar of Var
        datatype Form = FormNot of Form | FormPred of Pty * Ent
        and Pty = PtyVar of Var | PtyAbs of Var * Form
        fun varId s = VarId s
        fun entVar v = EntVar v
        fun ptyVar v = PtyVar v
        fun not p = FormNot p
        fun abs (v,p) = PtyAbs (v,p)
        fun pred (p,e) = FormPred (p,e)
        structure print =
            struct
                fun var (VarId s) = s
                fun ent (EntVar v) = var v
                fun form (FormNot p) = "not(" ^ form p ^ ")"
                  | form (FormPred (p,e))
                    = "pred(" ^ pty p ^ "," ^ ent e ^ ")"
                and pty (PtyVar v) = var v
                  | pty (PtyAbs (v,p))
                    = "abs(" ^ var v ^ "," ^ form p ^ ")"
            end
    end

(* Syntax of LQC *)

signature BEALER =
    sig
        type Var
        type Term
        type Rel
        type Form
        val varId : string -> Var
        val termVar : Var -> Term
        val termAbs : Form * Var list -> Term
        val relPred : Rel
        val rel : string * int -> Rel
        val formRel : Rel * Term list -> Form
        val formNot : Form -> Form
        val formEq : Term * Term -> Form
        val formAnd : Form * Form -> Form
        val formExists : Var * Form -> Form
        structure print :
            sig
                val var : Var -> string
                val term : Term -> string
                val rel : Rel -> string
                val form : Form -> string
            end
    end

(* Provide it *)

structure Bealer : BEALER =
    struct
        datatype Var = VarId of string
        datatype Rel = RelId of string * int | RelPred
        exception BadRel
        datatype Term = TermVar of Var | TermAbs of Form * Var list
        and Form = FormRel of Rel * Term list
          | FormEq of Term * Term
          | FormNot of Form
          | FormAnd of Form * Form
          | FormExists of Var * Form
        exception BadFormRel
        fun varId s = VarId s
        fun rel (s,n) = if (n > 0) then RelId (s,n) else raise BadRel
        val relPred = RelPred
        fun termVar v = TermVar v
        fun termAbs (f,vl) = TermAbs (f,vl)
        fun formRel (RelId (s,n),tl)
            = if (length tl = n) then FormRel (RelId (s,n),tl)
              else raise BadFormRel
          | formRel (RelPred,[s,t]) = FormRel (RelPred,[s,t])
          | formRel (RelPred,_) = raise BadFormRel
        fun formEq (s,t) = FormEq (s,t)
        fun formNot f = FormNot f
        fun formAnd (f,g) = FormAnd (f,g)
        fun formExists (v,f) = FormExists (v,f)
        structure print =
            struct
                fun var (VarId s) = s
                fun polylist _ [] = ""
                  | polylist f [e] = f e
                  | polylist f (h :: t) = f h ^ "," ^ polylist f t
                fun varlist vl = polylist var vl
                fun rel (RelId (s,_)) = s
                  | rel RelPred = "@"
                fun term (TermVar v) = var v
                  | term (TermAbs (f,[]))
                    = "[" ^ form f ^ "]"
                  | term (TermAbs (f,vl))
                    = "[" ^ form f ^ "]_" ^ varlist vl
                and form (FormRel (RelId (s,n),tl))
                    = rel (RelId (s,n)) ^ "(" ^ termlist tl ^ ")"
                  | form (FormRel (RelPred,[s,t]))
                    = "(" ^ term s ^ " " ^ rel RelPred ^ " "
                    ^ term t ^ ")"
                  | form (FormEq (s,t)) = term s ^ " = " ^ term t
                  | form (FormNot f) = "~" ^ form f
                  | form (FormAnd (f,g))
                    = "(" ^ form f ^ " & " ^ form g ^ ")"
                  | form (FormExists (v,f))
                    = "(E " ^ var v ^ ". " ^ form f ^ ")"
                and termlist tl = polylist term tl
            end
    end

(* Interpret RUSSELL syntax in Bealer *)

functor BealerToRussell(B : BEALER) : RUSSELL =
    struct
        type Var = B.Var
        type Ent = B.Term
        type Form = B.Form
        type Pty = B.Term
        fun varId s = B.varId s
        fun entVar v = B.termVar v
        fun ptyVar v = B.termVar v
        fun not p = B.formNot p
        fun abs (v,p) = B.termAbs (p,[v])
        fun pred (p,e) = B.formRel (B.relPred,[e,p])
        structure print =
            struct
                fun var v = B.print.var v
                fun ent e = B.print.term e
                fun form p = B.print.form p
                fun pty p = B.print.term p
            end
    end

(* Syntax of PT_2 *)

signature CHIERCHIA_TURNER =
    sig
        type Sort
        type Var
        type Cons
        type Ent
        type Func
        val i : Sort
        val u : Sort
        val nf : Sort
        val e : Sort
        val func : Sort * Sort -> Sort
        val varId : Sort * string -> Var
        val cons : Sort * string -> Cons
        val entVar : Var -> Ent
        val entCons : Cons -> Ent
        val funcCons : Cons -> Func
        val funcAbs : Var * Ent -> Func
        val funcDown : Ent -> Func
        val funcTrueDown : Ent -> Func
        val entUp : Func -> Ent
        val entPred : Func * Ent -> Ent
        val entTrue : Ent -> Ent
        val entNot : Ent -> Ent
        val entEq : Ent * Ent -> Ent
        val entAnd : Ent * Ent -> Ent
        val entExists : Var * Ent -> Ent
        structure print :
            sig
                val sort : Sort -> string
                val var : Var -> string
                val cons : Cons -> string
                val ent : Ent -> string
                val func : Func -> string
            end
    end

(* Provide it *)

structure ChierchiaTurner : CHIERCHIA_TURNER =
    struct
```

```
datatype Sort = I | U | NF | E | SortFunc of Sort * Sort
exception BadSort
datatype Var = VarId of Sort * string
exception BadVar
datatype Cons = ConsId of Sort * string
datatype Ent = EntVar of Var
  | EntCons of Cons
  | EntUp of Func
  | EntPred of Func * Ent
  | EntTrue of Ent
  | EntNot of Ent
  | EntEq of Ent * Ent
  | EntAnd of Ent * Ent
  | EntExists of Var * Ent
and Func = FuncCons of Cons
  | FuncAbs of Var * Ent
  | FuncDown of Ent
  | FuncTrueDown of Ent
exception BadFuncAbs
exception BadFuncDown
exception BadFuncTrueDown
exception BadEntUp
exception BadEntPred
exception BadEntNot
exception BadEntAnd
exception BadEntExists
val i = I
val u = U
val nf = NF
val e = E
fun func (SortFunc _,_) = raise BadSort
  | func (s,t) = SortFunc (s,t)
fun source (SortFunc (s,_)) = s
fun target (SortFunc (_,t)) = t
fun le (I,U) = true
  | le (I,E) = true
  | le (U,E) = true
  | le (NF,E) = true
  | le (SortFunc (E,I),SortFunc(E,E)) = true
  | le (s,t) = (s = t)
fun varId (SortFunc _,_) = raise BadVar
  | varId (s,str) = VarId (s,str)
fun cons (s,str) = ConsId (s,str)
fun entVar v = EntVar v
fun entCons c = EntCons c
fun funcCons c = FuncCons c
fun sortEnt (EntVar (VarId (s,_))) = s
  | sortEnt (EntCons (ConsId (s,_))) = s
  | sortEnt (EntUp _) = NF
  | sortEnt (EntPred (f,_)) = target (sortFunc f)
  | sortEnt (EntTrue _) = I
  | sortEnt (EntNot _) = I
  | sortEnt (EntEq (_,_)) = I
  | sortEnt (EntAnd (_,_)) = I
  | sortEnt (EntExists (_,_)) = I
and sortFunc (FuncCons (ConsId (s,_))) = s
  | sortFunc (FuncAbs (_,_)) = SortFunc (E,E)
  | sortFunc (FuncDown _) = SortFunc (E,E)
  | sortFunc (FuncTrueDown _) = SortFunc (E,I)
fun funcAbs (VarId (E,id),e) = FuncAbs (VarId (E,id),e)
  | funcAbs (_,_) = raise BadFuncAbs
fun funcDown e = if (sortEnt e = NF) then FuncDown e
                 else raise BadFuncDown
fun funcTrueDown e = if (sortEnt e = NF) then FuncTrueDown e
                     else raise BadFuncTrueDown
fun entUp f = if (le (sortFunc f, SortFunc (E,E))) then EntUp f
              else raise BadEntUp
fun entPred (f,e) = if (le (sortEnt e, source (sortFunc f)))
                        then EntPred (f,e)
                    else raise BadEntPred
fun entTrue e = EntTrue e
fun entNot e = if (sortEnt e = I) then EntNot e else raise BadEntNot
fun entEq (e,f) = EntEq (e,f)
fun entAnd (e,f) = if ((sortEnt e = I) andalso (sortEnt f = I))
                       then EntAnd (e,f)
                   else raise BadEntAnd
fun entExists (v,e) = if (sortEnt e = I) then EntExists (v,e)
                      else raise BadEntExists
structure print =
    struct
        fun sort I = "i"
          | sort U = "u"
          | sort NF = "nf"
          | sort E = "e"
          | sort (SortFunc (s,t))
            = "<" ^ sort s ^ "," ^ sort t ^ ">"
        fun var (VarId (s,id)) = id ^ "_" ^ sort s
        fun cons (ConsId (s,id)) = id ^ "_" ^ sort s
        fun ent (EntVar v) = var v
          | ent (EntCons v) = cons v
          | ent (EntUp f) = "^" ^ func f
          | ent (EntPred (f,e))
            = func f ^ "(" ^ ent e ^ ")"
          | ent (EntTrue e) = "!" ^ ent e
          | ent (EntNot e) = "~" ^ ent e
          | ent (EntEq (e,f))
            = "(" ^ ent e ^ " = " ^ ent f ^ ")"
          | ent (EntAnd (e,f))
            = "(" ^ ent e ^ " & " ^ ent f ^ ")"
          | ent (EntExists (v,e))
            = "(E " ^ var v ^ ". " ^ ent e ^ ")"
        and func (FuncCons c) = cons c
          | func (FuncAbs (v,e))
            = "L " ^ var v ^ "[" ^ ent e ^ "]"
```

```
                  | func (FuncDown e) = "%" ^ ent e
                  | func (FuncTrueDown e) = "!%" ^ ent e
            end
    end

(* Interpret RUSSELL syntax in ChierchiaTurner *)

functor ChierchiaTurnerToRussell(CT : CHIERCHIA_TURNER) : RUSSELL =
    struct
        type Var = CT.Var
        type Ent = CT.Ent
        type Form = CT.Ent
        type Pty = CT.Ent
        fun varId s = CT.varId (CT.nf,s)
        fun entVar v = CT.entVar v
        fun ptyVar v = CT.entVar v
        fun not p = CT.entNot p
        fun abs (v,p) =
            let
                val x = CT.varId (CT.e,"x")
            in
                CT.entUp (CT.funcAbs (x,CT.entAnd (
                    CT.entExists (v,CT.entEq (entVar v,entVar x)),p)))
            end
        fun pred (p,e) = CT.entPred (CT.funcTrueDown p,e)
        structure print =
            struct
                fun var v = CT.print.var v
                fun ent e = CT.print.ent e
                fun form p = CT.print.ent p
                fun pty p = CT.print.ent p
            end
    end

(* Store the results of these interpretations *)

structure MR = MakeRussell(MinimalRussell);

structure BR = MakeRussell(BealerToRussell(Bealer));

structure CTR = MakeRussell(ChierchiaTurnerToRussell(ChierchiaTurner));
```

Appendix B

Implementing the Johnson and Kay specifications

B.1 The inferential map U_0 and the function D

A Prolog implementation of §4.6's U_0 map follows.

```
:- op(400, xfy, &).
:- op(950, xfx, ==>).

u0(P, exists(A, PP)) :-          % at the top level, existentially quantify
    v(P, A),                     % any unbound variables
    u1(P, A, PP).                % then apply u1

u1(P & Q, A, PP & QQ) :- !,      % recurse over conjunctions
    u1(P, A, PP),
    u1(Q, A, QQ).

u1((P ==> Q), A, forall(XS, (PP ==> exists(YS, QQ)))) :- !,
    v(P, VP),                    % here we must pull out the right variables
    diff(VP, A, XS),             % for overall universal quantification
    union(XS, A, B),             % and for existential quantification
    v(Q, VQ),                    % in the consequent...
    diff(VQ, B, YS),
    union(YS, B, C),
    u1(P, B, PP),                % ...before recursing on the antecedent
    u1(Q, C, QQ).                % and consequent

u1(P, _, P) :-
    atomp(P), !.

atomp(P) :-
    functor(P, N, _),
    N \== '&',
    N \== '==>'.

v(P & Q, A) :- !,
    v(P, VP),
    v(Q, VQ),
    union(VP, VQ, A).

v((_ ==> _), []) :- !.

v(P, A) :-
    atomp(P), !,
    P =.. [_|A].

union([], X, X).

union([H|X], Y, Z) :-
    member(H, Y),
    union(X, Y, Z).

union([H|X], Y, [H|Z]) :-
       \+ member(H, Y),
    union(X, Y, Z).

diff([], _, []).

diff([H|X], Y, Z) :-
    member(H, Y),
    diff(X, Y, Z).

diff([H|X], Y, [H|Z]) :-
       \+ member(H,Y),
    diff(X, Y, Z).

member(X, [X|_]).

member(X, [_|Y]) :-
    member(X, Y).
```

The output of `u0` may contain representations of quantification over empty lists of variables; post-composing `u0` with the predicate `simp` defined below will eliminate these.

```
    simp(P & Q, PP & QQ) :- !,
        simp(P, PP),
        simp(Q, QQ).

    simp(P ==> Q, PP ==> QQ) :- !,
        simp(P, PP),
        simp(Q, QQ).

    simp(exists([],P), PP) :- !,
        simp(P, PP).

    simp(forall([],P), PP) :- !,
        simp(P, PP).
```

```
simp(exists(X,P), exists(X,PP)) :- !,
    simp(P, PP).

simp(forall(X,P), forall(X,PP)) :- !,
    simp(P, PP).

simp(P, P).
```

The following Prolog code implements the map D introduced in §4.6 for the 'discourse-representation' implementation:

```
:- op(400, xfy, &).

d([T0|_]-[T1|_], MDR) :-
    subtract(T0, T1, L1),       % realize difference lists as single list
    noanaph(L1, L2),            % eliminate anaphoric indexes
    andify(L2, MDR).            % turn lists into conjunctions

subtract([], L, L).
subtract(L1, L2, []) :-
    L1 == L2, !.
subtract(_, X, _) :-
    var(X), !, fail.
subtract(T0, [H1|T1], [H1|T2]) :-
    subtract(T0, T1, T2).

noanaph([], []).
noanaph([i(_)|T1], T2) :- !,
    noanaph(T1, T2).
noanaph([(A1==>B1)|T1], [(A2==>B2)|T2]) :- !,
    noanaph(A1, A2),
    noanaph(B1, B2),
    noanaph(T1, T2).
noanaph([H|T1], [H|T2]) :-
    noanaph(T1, T2).

andify([X1], X2) :- !,
    and1(X1, X2).
andify([H1|T], H2 & T2) :-
    and1(H1, H2),
    andify(T, T2).

and1((A1==>B1), (A2==>B2)) :- !,
    andify(A1, A2),
    andify(B1, B2).
and1(X, X).
```

The following version of `d`, together with its additional predicates and those just defined, implements D for the 'sets-of-infons' implementation:

```
d(@([Sit|_],T,L), MDR) :-
```

```
        subtract(T, L, Infons),             % realize difference lists as list
        infons(Sit, Infons, TopLevel),      % extract top-level infons
        expandsits(TopLevel, Infons, TL2),  % and expand situation tags in them
        noanaph(TL2, TL3),                  % eliminate anaphoric indexes
        andify(TL3, MDR).                   % turn lists into conjunctions

    infons(_, [], []).
    infons(Sit, [S:I|T1], [I|T2]) :-
        Sit == S,
        infons(Sit, T1, T2).
    infons(Sit, [S:_|T1], T2) :-
        Sit \== S,
        infons(Sit, T1, T2).

    expandsits([], _, []) :- !.
    expandsits([(S1==>S2)|T1], I, [(L1==>L2)|T2]) :- !,
        infons(S1, I, IL1),
        infons(S2, I, IL2),
        expandsits(IL1, I, L1),
        expandsits(IL2, I, L2),
        expandsits(T1, I, T2).
    expandsits([H|T1], I, [H|T2]) :-
        expandsits(T1, I, T2).
```

B.2 The anaphoric maps A and P

The following Prolog code (additional to that in the previous section) implements A and P for the 'discourse-representation' implementation, demonstrating that it satisfies the above constraints:

```
    a([T0|_]-[T1|_], Is) :-
        subtract(T0, T1, L),
        indexes(L, Is).

    p(P-_, Is) :-
        unvar(P, P1),
        getinds(P1, Is).

    indexes([], []).
    indexes([i(X)|T1], [X|T2]) :- !,
        indexes(T1,T2).
    indexes([_|T1], T2) :-
        indexes(T1,T2).

    unvar(X, []) :-
        var(X), !.
    unvar([], []).
    unvar([H|T1], [H|T2]) :-
        unvar(T1,T2).

    getinds([], []).
```

```
getinds([H|T], L) :-
    unvar(H, H1),
    indexes(H1, Is),
    getinds(T, Js),
    append(Is, Js, L).
```

The ‘sets-of-infons’ implementation also satisfies this specification, as demonstrated by the following code:

```
a(@(S,T,L), Is) :-
    subtract(T, L, Infons),
    unvar(S, Sits),
    indexes(Sits, Infons, Is).

p(@(S,T,_), Is) :-
    unvar(T, Infons),
    unvar(S, Sits),
    indexes(Sits, Infons, Is).

indexes(_, [], []).
indexes(Sits, [S:i(X)|T1], [X|T2]) :-
    mem(S, Sits), !,
    indexes(Sits, T1, T2).
indexes(Sits, [_|T1], T2) :-
    indexes(Sits, T1, T2).

mem(X, [Y|_]) :-
    X == Y, !.
mem(X, [_|T]) :- member(X,T).
```

References

Barwise, J. (1987) Noun phrases, generalized quantifiers and anaphora. In P. Gärdenfors, ed., *Generalized Quantifiers: Linguistic and Logical Approaches*, pp. 1–29. Dordrecht: D. Reidel.

Barwise, J. and R. Cooper (1981) Generalized quantifiers and natural language. *Linguistics and Philosophy* **4**(2), 159–219.

Bealer, G. (1982) *Quality and Concept.* Oxford: Clarendon Press.

Chierchia, G. and R. Turner (1988) Semantics and property theory. *Linguistics and Philosophy* **11**(3), 261–302.

Chomsky, N. (1981) *Lectures on Government and Binding.* Dordrecht: Foris Publications.

Chomsky, N. (1992) A minimalist program for linguistic theory. Occasional Papers in Linguistics 1, MIT, Cambridge, Mass.

Cresswell, M. J. (1990) *Entities and Indices.* Dordrecht: Kluwer.

Dalrymple, M., S. M. Shieber and F. C. N. Pereira (1991) Ellipsis and higher-order unification. *Linguistics and Philosophy* **14**(4), 399–452.

Evans, G. (1977) Pronouns, quantifiers and relative clauses (I). *Canadian Journal of Philosophy* **7**(3), 467–536.

Evans, G. (1980) Pronouns. *Linguistic Inquiry* **11**(2), 337–362.

Gazdar, G., E. Klein, G. Pullum and I. Sag (1985) *Generalized Phrase Structure Grammar.* London: Basil Blackwell.

Geach, P. T. (1962) *Reference and Generality: An Examination of Some Medieval and Modern Theories.* Ithaca, N.Y.: Cornell University Press.

Grandy, R. E. (1976) Anadic logic and English. *Synthese* **32**, 395–402.

Groenendijk, J. and M. Stokhof (1991) Dynamic Predicate Logic. *Linguistics and Philosophy* **14**(1), 39–100.

Johnson, M. and M. Kay (1990) Semantic abstraction and anaphora. In H. Karlgren, ed., *COLING-90: Papers Presented to the 13th International Conference on Computational Linguistics*, Vol. 1, pp. 17–27. University of Helsinki.

Johnson, M. and E. Klein (1986) Discourse, anaphora and parsing. In *Proceedings of the 11th International Conference on Computational Linguistics*, pp. 669–675, Bonn University.

Kamareddine, F. and E. Klein (1993) Nominalization, predication and type containment. *Journal of Logic, Language and Information* **2**, 171–215.

Kamp, H. (1981) A theory of truth and semantic representation. In J. A. G. Groenendijk, T. M. V. Janssen and M. B. J. Stokhof, eds., *Formal Methods in the Study of Language: Part 1*, pp. 277–322. Amsterdam: Mathematisch Centrum.

Kamp, H. and U. Reyle (1993) *From Discourse to Logic: Introduction to Model-theoretic Semantics of Natural Language, Formal Logic and Discourse Representation Theory.* Dordrecht: Kluwer.

Kaplan, R. M. and J. Bresnan (1982) Lexical-Functional Grammar: A formal system for grammatical representation. In J. Bresnan, ed., *The Mental Representation of Grammatical Relations*, pp. 173–281. Cambridge, Mass.: MIT Press.

Karttunen, L. (1969) Pronouns and variables. In R. I. Binnick, A. Davison, G. M. Green and J. L. Morgan, eds., *Papers from the Fifth Regional Meeting of the Chicago Linguistics Society*, pp. 108–116. Chicago, Ill.: Department of Linguistics, University of Chicago.

Ladusaw, W. A. (1979) *Polarity Sensitivity as Inherent Scope Relations.* PhD thesis, University of Texas at Austin. Reproduced by the Indiana University Linguistics Club, 1980.

Lasnik, H. (1976) Remarks on coreference. *Linguistic Analysis* **2**(1), 1–22.

Montague, R. (1970) Universal grammar. *Theoria* **36**, 373–398. Reprinted in R. H. Thomason, ed., *Formal Philosophy: Selected Papers of Richard Montague*, pp. 222–246. New Haven, Conn.: Yale University Press, 1974.

Montague, R. (1973) The proper treatment of quantification in ordinary English. In K. J. J. Hintikka, J. M. E. Moravcsik and P. Suppes, eds., *Approaches to Natural Language: Proceedings of the 1970 Stanford Workshop on Grammar and Semantics*, pp. 221–242. Dordrecht: D. Reidel. Reprinted in R. H. Thomason, ed., *Formal Philosophy: Selected Papers of Richard Montague*, pp. 247–270. New Haven, Conn.: Yale University Press, 1974.

Newton, M. (1992) *Abstract Specification of Grammar.* PhD thesis, Centre for Cognitive Science, University of Edinburgh.

Pagin, P. and D. Westerståhl (1993) Predicate logic with flexibly binding operators and natural language semantics. *Journal of Logic, Language and Information* **2**(2), 89–128.

Partee, B. and E. Bach (1981) Quantification, pronouns, and VP anaphora. In J. A. G. Groenendijk, T. M. V. Janssen and M. B. J. Stokhof, eds., *Formal Methods in the Study of Language: Part 2*, pp. 445–481. Amsterdam: Mathematisch Centrum.

Pereira, F. C. N. and S. M. Shieber (1987) *Prolog and Natural Language Analysis.* Stanford, Ca.: Center for the Study of Language and Information.

Pollard, C. and I. A. Sag (1987) *Information-Based Syntax and Semantics*, Vol. 1: *Fundamentals*. Stanford, Ca.: Center for the Study of Language and Information.

Pollard, C. and I. A. Sag (1994) *Head-Driven Phrase Structure Grammar*. Stanford, Ca. and Chicago, Ill.: CSLI and University of Chicago Press.

Quine, W. V. (1976) Algebraic logic and predicate functors. In *The Ways of Paradox and Other Essays*, rev. and enlarged ed, pp. 283–307. Cambridge, Mass.: Harvard University Press.

Sannella, D. and A. Tarlecki (1992) Toward formal development of programs from algebraic specifications: Model-theoretic foundations. Report ECS-LFCS-92-204, Laboratory for the Foundations of Computer Science, Department of Computer Science, University of Edinburgh.

Sells, P. (1985a) Restrictive and non-restrictive modification. Report CSLI-85-28, Center for the Study of Language and Information, Stanford, Ca.

Sells, P. (1985b) *Lectures on Contemporary Syntactic Theories*. Stanford, Ca.: Center for the Study of Language and Information.

Sells, P. (1986) Coreference and bound anaphora: A restatement of the facts. In S. Berman, J-W. Choe and J. McDonough, eds., *Proceedings of NELS 16*, pp. 434–446. Graduate Linguistic Student Association of the University of Massachusetts at Amherst. Papers presented at the sixteenth annual meeting of the North Eastern Linguistic Society.

Shieber, S. M. (1986) *An Introduction to Unification-Based Approaches to Grammar*. Stanford, Ca.: Center for the Study of Language and Information.

Shieber, S. M., H. Uszkoreit, F. C. N. Pereira, J. J. Robinson and M. Tyson (1983) The formalism and implementation of PATR-II. In B. Grosz and M. E. Stickel, eds., *Research on Interactive Acquisition and Use of Knowledge*, pp. 39–79. Artificial Intelligence Center, SRI International, Menlo Park, Ca.

Steedman, M. (1988) Combinators and grammars. In R. T. Oehrle, E. Bach and D. Wheeler, eds., *Categorial Grammars and Natural Language Structures*, pp. 417–442. Dordrecht: Kluwer.

Turner, R. (1987) A theory of properties. *Journal of Symbolic Logic* **52**(2), 455–472.

Wood, M. M G. (1993) *Categorial Grammars*. London: Routledge.

Zeevat, H. (1989) A compositional approach to discourse representation theory. *Linguistics and Philosophy* **12**(1), 95–131.