Introduction

This book introduces a number of fundamental techniques for computing semantic representations for fragments of natural language and performing inference with the result. Both the underlying theory and their implementation in Prolog are discussed. We believe that the reader who masters these techniques will be in a good position to appreciate (and critically assess) ongoing developments in computational semantics.

Computational semantics is a relatively new subject, and trying to define such a lively area (if indeed it is a single area) seems premature, even counterproductive. However, in this book we take "semantics" to mean "formal semantics" (that is, the business of giving model-theoretic interpretations to fragments of natural language, usually with the help of some intermediate level of logical representation) and "computational semantics" to be the business of using a computer to actually build such representations (*semantic construction*) and reason with the result (*inference*). Thus this book introduces techniques for tackling the following two questions:

1. *How can we automate the process of associating semantic representations with expressions of natural language?*
2. *How can we use logical representations of natural language expressions to automate the process of drawing inferences?*

In the remainder of the Introduction we'll briefly sketch how we are going to tackle these questions, explain where we think computational semantics belongs on the intellectual landscape, suggest how best to make use of the text and the associated software, and (in the Notes at the very end) give a brief historical account of the origins of computational semantics. Some of the discussion that follows may not be completely accessible at this stage, especially if you have never encountered semantics or logic before. But if this is the case, don't worry. Concentrate on the straightforward parts (such as the chapter-by-chapter outline and the advice on using this book), and then go straight on to the chapters that follow. You can return to the Introduction later; by then you will be in a better position to assess the perspective on computational semantics that has guided the writing of this book.

Representation

As regards semantic construction, this book is fairly orthodox (though see below for some caveats). We first introduce the reader to first-order logic, and then show how various kinds of sentences of natural language can be systematically translated into this formalism. For example, the techniques we discuss (and the software we implement) will enable us to take a sentence like

Every boxer loves Mia.

as input and return the following formula of first-order logic as output:

$\forall x(\textsc{boxer}(x) \rightarrow \textsc{love}(x,\textsc{mia}))$.

The technical tool we shall use to drive this translation process is the lambda calculus. We motivate and introduce the lambda calculus from a computational perspective, and show in detail how it can be incorporated into an architecture for building semantic representations.

To put it another way, it's not inaccurate to claim that roughly half of this book is devoted to what is known as Montague semantics. Yes, it's true that we don't discuss a lot of topics that would ordinarily be taught in a first course on Montague semantics (for example, we don't discuss intensional semantics). But in our view Richard Montague was not merely the father of *formal* semantics (or *model-theoretic* semantics, as it is often called), he was also the father of *computational* semantics. Richard Montague made many pioneering contributions to the study of semantics, but in our view the most important was the conceptual leap that opened the door to genuine computational semantics: he showed that the process of constructing semantic representations for expressions of natural language could be formulated *algorithmically*. Many philosophers before Montague (and many philosophers since) have used (various kinds of) logic to throw light on (various aspects of) natural language. But before Montague's work, such comparisons were essentially analogies. Montague showed how to link logic and language in a *systematic* way, and it is this aspect of his work that lies at the heart of this book.

Inference

But this book is not just about representation, it is also about inference. Now, inference is a vast topic, and it is difficult to be precise about what is (and is not) covered by this term. But, roughly speaking, we view inference as the process of making *implicit* information *explicit*. To keep this book to a manageable size we have focused on one particular aspect of this process, namely the making of logical inferences. We have done so by formulating three inference tasks—the querying task, the consistency checking task, and the informativity checking task—and have looked at inference in natural language through the lens they provide. For example, it is intuitively clear that the discourse

Every boxer loves Mia. Butch is a boxer. Butch does not love Mia.

is incoherent. But why is it incoherent? As we shall show in Chapter 1, the consistency checking task gives us an important theoretical handle on this type of incoherency. Moreover in the second half of the book we shall create a computational architecture that makes use of sophisticated automated reasoning tools to give us a useful (partial) grasp on consistency checking (partial, because of the undecidability of first-order logic). We conclude the book by showing how our semantic construction software (part of Richard Montague's legacy) and our inference architecture (the legacy of John Alan Robinson and the other pioneers of automated reasoning) can be integrated.

Comments and caveats

Well, that's where we heading—but before moving on, two remarks should be made. First, above we talked about semantic construction and inference as if they were independent, but in fact they're not. Indeed, how semantic construction and inference are interleaved is an extremely deep and difficult problem, and we certainly don't claim to have solved it in this book. Nonetheless, we *do* believe that the issues this problem raises need to be explored computationally, and that architectures of the type discussed here—that is, architectures which draw on both semantic construction and inference modules—will become fundamental research tools.

Second, the working definition of computational semantics given above isn't quite as innocent as it looks. Many formal semanticists claim that intermediate levels of logical representation are essentially redundant. Richard Montague himself, in his paper "English as a Formal Language", showed how a small fragment of English could be model-theoretically interpreted without first translating it into an intermediate logical representation. Moreover, in his paper "Universal Grammar", he showed that (given certain assumptions) it is always possible to interpret fragments of natural language in this way.

Nonetheless, we feel justified in emphasising the role of intermediate logical representations. For a start, the move to a *computational* perspective on formal semantics certainly increases the *practical* importance of the representation level. Logical representations—that is, formulas of a logical language—encapsulate meaning in a clean and compact way. They make it possible to use well understood proof systems to perform inference, and we shall learn how to exploit this possibility. Models may be the heart of traditional formal semantics, but representations are central to its computational cousin.

Moreover—and this is something that we hope becomes increasingly clear in the course of the book—we feel that the computational perspective vividly brings out the *theoretical* importance of representations. The success of Discourse Representation Theory (DRT) over the past two decades, the explosion of interest in underspecification (which we explore in Chapter 3) and the exploration of glue languages such as linear logic to drive the process of semantic construction all bear witness to an important lesson: semanticists ignore representations at their peril. Representations are well-defined mathematical entities that (among other things) can be manipulated computationally, explored geometrically, and specified indirectly with the aid of constraints. The fact that representations are theoretically eliminable does not mean they should not be taken seriously.

But why computational semantics?

Our discussion so far has taken it for granted that computational semantics is an interesting subject, one well worth studying. But it is probably a good idea to be explicit about why we think this is so.

We believe that the tools and techniques of computational semantics are going to play an increasingly important role in the development of semantics. Now, semantics has made enormous strides since the pioneering work of Richard Montague in the late 1960s and early 1970s. Nonetheless, we believe that its further development is likely to become increasingly reliant on the use of computational tools. Modern formal semantics is still a paper-and-pencil enterprise: semanticists typically examine in detail a topic that interests them (for example, the semantics of tense, or aspect, or focus, or generalized quantifiers), abstract away from other semantic phenomena, and analyse the chosen phenomenon in detail. This "work narrow, but deep" methodology has undeniably served semanticists well, and has lead to important insights about many semantic phenomena. Nonetheless, we don't believe that it can unveil all that needs to be known about natural language semantics, and we don't think it is at all suitable for research that straddles the (fuzzy and permeable) border between semantics and pragmatics (the study of how language is actually used). Rather, we believe that in the coming years it will become increasingly important to study the *interaction* of various semantic (and pragmatic) phenomena, and to model, as precisely as possible, the role played by inference.

Now, it's easy to say that this is what should be done—actually doing it, however, is difficult. Indeed, as long as semanticists rely purely on pencil-and-paper methods, it is hard to see how this style of research can produce detailed results. In our view, computational modelling is required. That is, we believe that flexible computational architectures which make it possible to experiment with semantic representations, semantic construction strategies, and inference, must be designed and implemented. To give an analogy, nowadays it is possible to use sophisticated graphics programs when studying large molecules (such as proteins). Such programs make it possible to grasp the three-dimensional structure of the molecule, and hence to think at a more abstract level about their properties and the reactions they can enter into. Semanticists need analogous tools. The ability to formulate detailed semantic theories, and to compute rapidly what they predict, could open up a new phase of research in semantics. It could also revolutionise the teaching of semantics.

Two comments. First, note that we're *not* claiming that semanticists should abandon their traditional "work narrow, but deep" strategy; this style of research is indispensable. Rather, we are suggesting that it should be augmented by a computer-aided "work broad, and model the interactions" approach. Second—before anyone gets their hopes up prematurely—we would like to emphasise that the software discussed in this book does *not* constitute a genuine research architecture. The design and implementation of the type of "Semantic Workbenches" we have in mind is a serious task, one far beyond the scope of an introduction to computational semantics. But we certainly do hope that the software provided here will inspire readers to design and implement more ambitious systems.

Computational semantics and computational linguistics

So that's our answer to the question "Why computational semantics?". But although this answer might well interest (or enrage!) formal semanticists, there is another group of researchers who may find it unconvincing. Which group? Computational linguists. We wouldn't be surprised to learn that some computational linguists are dubious about our aims and methods. Where is the statistics? Where is the use of corpora? Why analyse sentences in such depth? Does inference really require the use of such powerful formalisms as first-order logic? Indeed, does inference really require logic at all? We would like to make two brief remarks here, for we certainly *do* view the techniques taught in this book as an integral part of computational linguistics.

Firstly, what we teach in this book is certainly *compatible* with statistically-oriented approaches to computational linguistics. In essence, we provide some fundamental semantic construction tools (the use of lambda calculus, coupled with methods for coping with scope ambiguities) and inference tools (an architecture for using theorem provers and model builders in parallel) and put them to work. In this book these components are used via a simple Definite Clause Grammar (DCG) architecture, but they certainly don't have to be. They can be—and have been—combined with such tools as speech recognisers and wide coverage statistical parsers to build more interesting systems.

Secondly, we believe that the methods taught in this book are not merely compatible, but might actually turn out to be *useful* to statistically-oriented work. This book was born from the conviction that formal semantics has given rise to the deepest insights into the semantics of natural language that we currently have—and an accompanying belief that a computational perspective is needed to fully unleash their potential. So we find it natural (and important) to look for points of contact with mainstream computational linguistics. For example, many computational linguists want to extend the statistical revolution of the late 1980s and early 1990s (which transformed such areas as speech processing and parsing) to the semantic domain. We don't see any conflict between this goal and the ideas explored in this book. Indeed, we believe that techniques from computational semantics may be important in exploring statistical approaches to semantics: detailed training corpora will be needed, and the techniques of computational semantics may be helpful in producing the requisite "gold standard" material.

It is true that these remarks are somewhat speculative. Nonetheless, in our view the low cost of massive computational power, the ubiquitous presence of the internet, the sophistication of current automated reasoning tools, and the superb linguistic resources now so widely available, all add up to a new era in semantic research. Understanding how natural language works is one of the toughest (and most interesting) problems there is. It's time to get all hands on deck.

Outline

The key aim of this book is to develop a working toolkit for computational semantics. We develop this toolkit as follows:

Chapter 1. First-Order Logic. We begin by introducing the syntax and semantics of first-order logic, the semantic representation language used in this book. We then define and discuss the three inference tasks we are interested in: the querying task, the consistency checking task, and the informativity checking task. Following this, we implement a first-order model checker in Prolog. A model checker is a program that checks whether a formula is true in a given model, or to put it another way, it is a piece of software that performs the querying task.

Chapter 2. Lambda Calculus. Here we start studying semantic construction. We outline the methodology underlying our work (namely, compositionality) and motivate our use of DCGs (Definite Clause Grammars). We then write two rather naive programs that build semantic representations for a very small fragment of English. These experiments lead us to the lambda calculus, the tool that drives this book's approach to semantic construction. We implement β-conversion, the computational core of the lambda calculus, and then integrate it into the grammatical architecture that will be used throughout the book.

Chapter 3. Underspecified Representations. Here we investigate a fundamental problem for computational semantics: scope ambiguities. These are semantic ambiguities that can arise in syntactically unambiguous expressions, and they pose a problem for compositional approaches to semantic construction. We illustrate the problem, and present four (increasingly more sophisticated) solutions: Montague's use of quantifier raising, Cooper storage, Keller storage, and hole semantics. We integrate storage and hole semantics into our grammar architecture.

Chapter 4. Propositional Inference. Here we turn to the second major theme of the book: inference. Our approach to inference will be based on first-order theorem proving and model building, and in this chapter we lay the conceptual foundations for the work of Chapter 5 in the simpler setting of propositional logic. We introduce a signed tableau and a resolution system for propositional calculus, implement both in Prolog, and conclude with a discussion of a number of theoretical issues.

Chapter 5. First-Order Inference. In this chapter we explore inference in the setting of full first-order logic—and, as we swiftly learn, that's a computationally demanding setting. We extend the propositional and resolution theorem provers to deal with first-order logic, but we also see that home-brewed theorem provers simply don't have the muscle for tackling the consistency and informativity checking tasks in a serious way. So we change tack: instead of building our own, we show the reader how to integrate the sophisticated inference tools created by the automated reasoning community into an inference architecture for computational semantics. By the end of the chapter, we have constructed an architecture for consistency and informativity checking that works by calling (sophisticated) theorem provers and model builders in parallel.

Chapter 6. Putting It All Together. In this chapter we bring together the software developed in earlier chapters and develop a series of programs bearing the name Curt (which stands for Clever Use of Reasoning Tools). Curt starts out as a real baby—it can build semantic representations and detect scope ambiguities, but alas, nothing more. No matter! By making use of our inference architecture, (and the model checker developed in Chapter 1), we are able, step-by-step, to extend Curt's abilities so that it can handle consistency checking, informativity checking, eliminate logically equivalent readings, incorporate background knowledge, and answer simple questions.

Here's how the chapters fit together:

$$\begin{tikzcd}
 & \boxed{\textbf{Chapter 1}} \arrow[dl] \arrow[dr] & \\
\boxed{\textbf{Chapter 2, 3}} \arrow[dr] & & \boxed{\textbf{Chapter 4, 5}} \arrow[dl] \\
 & \boxed{\textbf{Chapter 6}} &
\end{tikzcd}$$

That is, Chapter 1 provides the foundation for everything that follows. Chapter 2 and 3 are the representation track of the book, and should be read together. Similarly, Chapters 4 and 5 are the inference track of the book, and again, these two chapters should be read together. The representation and inference tracks are independent of each other, and can be read in either order. Finally, Chapter 6 draws on all that has gone before.

Each chapter concludes with Notes that list references and briefly discuss more advanced topics. Four appendices at the end of the book provide background information.

Using this book

We have tried to make this book relatively self-contained. In fact, there is only one real prerequisite that is *not* introduced here, namely the Prolog programming language. To gain the most out of the computational side of this book, you will need to have some knowledge of this language, and access to a Prolog interpreter. Many good books on Prolog are available, but we'd like to draw your attention to *Learn Prolog Now!*, by Patrick Blackburn, Johan Bos, and Kristina Striegnitz. Written in parallel with the present book, it contains everything needed to understand the code presented here. *Learn Prolog Now!* is available free on the internet at

`http://www.learnprolognow.org/`

For the Prolog interpreters that handle our software, see Appendix A.

Apart from Prolog, we believe that this book covers everything most readers will need. Indeed, we believe that even readers with fairly modest backgrounds in linguistics and logic should be able to follow the discussion. We have taught this material both to linguistics students (with rather weak logical and computational backgrounds) and to computer science students (with stronger logical backgrounds, but no prior exposure to linguistics). Our experience suggests that as long as the instructor is sensitive to the type of background knowledge the students have, it is possible to successfully teach this material. Moreover, the Notes at the end of each chapter provide many references for supplementary reading. So if you are using this book for self study and get stuck at some point, try looking at these.

But there is one point we would like to strongly emphasise to all our readers: *please take the computational component seriously.* Yes, it is certainly possible to read this book with only half an eye on the computational developments. Moreover, we'll also admit that even if you're allergic to computers and computer programming, but want to learn about semantics in a way that emphasises inference, you can use this book for that purpose without worrying too much about the Prolog programs.

Still, while you *can* read the book this way, we feel it's a bit of a shame to do so—after all, this is a book on *computational* semantics, and the reader who does not make the effort to get to grips with the computational issues it discusses is getting (at most) fifty percent of what this book has to offer. Now, there's no denying that for some readers the computational side will be the hard part—readers with weak computational backgrounds will have to put in some extra work (apart from anything else, they'll have to learn something about Prolog). But the effort is well worth making. Thinking about problems computationally often reveals new perspectives on old ideas. For example, our account of the lambda calculus in Chapter 2 makes no appeal to types, function valued functions, or higher-order logic—rather, lambda calculus is presented as a beautiful piece of data-abstraction that emerges naturally from a declarative analysis of semantic construction. To give another example, the various techniques developed for handling scope ambiguities (from Montague's method, through storage methods, to modern underspecification methods) display a conceptually clear evolutionary line when viewed computationally.

Thus the computational side of this book should not be viewed as an optional extra. Indeed, we might say that the ideal reader of this book is someone who treats the text as *documentation*. Such a reader might not wish to understand all the details of the programs provided, but he or she would certainly want to play with them, perhaps by extending the grammars, by experimenting with novel semantic constructions, by applying the inference architecture to novel tasks, or by applying these ideas to other languages. In short, don't think of this as a book. Think of it as a tool-kit for exploring computational semantics. And then put it to work.

Web support

We have set up webpage for this book. The URL is

`http://www.blackburnbos.org/`

There you will find our Prolog programs, pointers to other useful software (such as Prolog interpreters, theorem provers and model builders), and any corrections to the text that need to be made. From time to time we will place material there that extends the present text. For example, Blackburn and Bos (2003), which can be read as a sort of 'alternative introduction' to this book, can be found on the website.

Notes

Modern logic and semantics stem from the work of Gottlob Frege (1848-1925). On the logical side, Frege introduced the use of variable binding quantifiers (we shall see such quantifiers in the following chapter when we introduce first-order logic), and on the semantic side he introduced a number of concepts (such as the distinction between the "sense" and the "reference" of an expression) that are still important today. Readers wanting a taste of Frege's work could try Frege (1892). Translations of this paper have been widely anthologised (for example you can find it in Martinich (1996) under the title "On Sense and Nominatum").

Frege's logical work and his ideas on the foundations of mathematics became increasingly influential from roughly 1900 onwards, and their influence endures till this day. But his pioneering work on semantics took longer to bear fruit. While some important early work was done (for example, by the philosopher Bertrand Russell) the next big steps were not taken till the middle of the 20th century.

With the benefit of hindsight we can see that this delay was not accidental, for a key idea was missing: the notion of *interpretation in a model*. As we shall see in the following chapter, nowadays logic is conceived of as having *two* main components. First, there is some kind of formal logical language (for example, a language of first-order logic that makes use of such symbols such as $\forall$, $\exists$, $\wedge$, $\rightarrow$ and so on). But in addition, crucial use is made of what are known as models, simple mathematical structures that act as pictures of the world (or at least, that part of the world we happen to be interested in for our application). At the heart of much modern logic is the idea of giving a precise mathematical definition of how formal logical languages are linked with models—or to put it more semantically, to specify how formal languages are to be *interpreted* in models (such a definition is usually called a *satisfaction definition*). One of the fundamental facts that any theory of semantics is going to have to get to grips with is that natural languages (like English) can be used to talk about the world around us (for example, English speakers use the word "woman" for adult human females). When we link a logical language with a model via a satisfaction definition we gain a precise mathematical handle on the language-world relationship.

In 1933 Alfred Tarski (1902–1983) gave the first fully explicit satisfaction definition (see Tarski (1935) for a German translation of the Polish original). Its importance was quickly realised by both philosophers and mathematicians, and stimulated serious work on semantics. In particular, Rudolf Carnap (1891–1979), produced an important body of work. For example, his book "Meaning and Necessity" (Carnap, 1947) is still well worth looking at: among other things it discusses the semantics of belief and necessity (two key examples of intensionality in natural language), examines the notion of "meaning postulate" in detail (such postulates would nowadays be thought of as axioms encoding world knowledge or lexical knowledge), and speculates on the possibility of extending Tarski's model-based approach to semantics to handle pragmatics too.

But it was with the work of Richard Montague (1930–1971) that semantics finally came of age. Although Montague only wrote a handful of papers on the subject, they were destined to shape subsequent research.

Three of his papers, "Pragmatics" (Montague, 1968), "On the Nature of Certain Philosophical Entities" (Montague, 1969), and "Pragmatics and Intensional Logic" (Montague, 1970b) are technically sophisticated developments of the program initiated by Carnap. (And as the titles of two of these papers indicate, Montague was able to extend the model-based approach to semantics to cover a pragmatic phenomenon, namely indexicality.) But it is his last three papers "English as a Formal Language" (Montague, 1970a), "Universal Grammar" (Montague, 1970c), and "The Proper Treatment of Quantification in Ordinary English" (Montague, 1973) that sound a genuinely new note, for it is here that Montague unveils what has become known as the *method of fragments*. What does this mean? Simply that in these papers Montague defined grammars for small portions of English, and showed how the sentences generated by these grammars could be interpreted in models. In "English as a Formal Language" Montague interpreted the English fragment directly (that is, without first translating into an intermediate logical representation) whereas in "Universal Grammar" and "The Proper Treatment of Quantification in Ordinary English" he first translated into higher-order logic. But this difference is far less important than what is common to them all: *in all three papers the interpretation process required is completely explicit*. In essence, all three papers give interpretation algorithms for fragments of English. To be sure, nobody would claim that the interpretations offered by Montague cover all aspects of what we might want to call meaning, and it is also true that the fragments given by Montague were rather small. But such considerations should not blind us to the fact that in these papers something very important has taken place: we see the first glimpse of a possible *mechanism* underlying natural language semantics.

It is not possible here to give a detailed account of developments in semantics since the work of Montague. Perhaps the most important development, and certainly the one of most relevance to computational semantics, was the birth of Discourse Representation Theory (see Heim (1982), Kamp (1984), and Kamp and Reyle (1993)). DRT (as it is usually called) has enabled Montague's program to be extended from the level of sentences to entire discourses, and has also enabled semantics to make further inroads on the domain of pragmatics. But DRT and many other interesting developments are beyond the scope of the book, so we refer the reader to Partee (1997a) and Partee (1997b), two useful discussions of Montague's work and the research it inspired.

We now turn our attention from the development of formal semantics to research conducted in two computational disciplines, namely computational linguistics and Artificial Intelligence (AI). The formal semantic tradition has been the primary source of much that we teach in this book, but the way we view this material has been indelibly marked by ideas from computational linguistics and AI. So let's round out the picture with a quick look at these traditions.

One of the basic themes of this book is the usefulness of logic as a tool for representation and inference. But this is not a new idea—it's a mainstay of classical AI. It's interesting to look through some of the more influential AI textbooks, say Nilsson (1980), Winston (1981), Charniak and McDermott (1985), Rich and Knight (1990), and Russell and Norvig (1995). All take first-order logic as a fundamental framework for representation and inference (and not merely for natural language tasks either), and discuss inference procedures for first-order logic (notably resolution) in varying degrees of detail. Weaker formalisms (such as semantic nets) are sometimes used and (especially in the later texts) the point is explicitly made that such formalisms are essentially fragments of first-order logic with good computational properties (for example, semantic nets are a forerunner of what are nowadays known as description logics; see Baader et al. (2003)).

In short, many of the fundamental ideas on logic and inference underlying these texts are close to those taught here. Indeed, much what divides this book from these earlier introductions (apart from the obvious fact that the texts just mentioned cover a wide range of topics in AI, such as learning, planning, and image recognition, whereas ours focuses exclusively on computational semantics) is simply due to the explosive pace of contemporary research. When we talk about semantic construction, we can draw on ideas (such as constraint-based underspecification) that hadn't been developed when these books were written. And when we advocate the use of first-order logic and theorem proving, we can point the reader to provers (and indeed, newer tools such as model builders) whose performance dwarfs anything available earlier. Moreover, in this book we emphasise the importance of developing architectures by finding the best available components and linking them. This style of development wasn't so practical in the 1980s and early 1990s; nowadays, given the ubiquity of the internet, it seems likely to become the default option.

So there is a broad similarity of aims and methods between what we teach in this book and much that is done in computational linguistics and AI—and given the fundamental role played by logic in these disciplines, we don't find this surprising. If anything, what *is* surprising is how long it has taken for a real alliance between computational linguistics and formal semantics to be forged. For until the 1990s there seem to have been few systematic attempts by researchers in computational linguistics (and AI) to make contact with ideas from formal semantics (and attempts by formal semanticists to make serious contact with ideas from computational linguistics and AI seem to have been even thinner on the ground). That said, there *are* some interesting examples of earlier work in computational linguistics that falls (or almost falls) under our working definition of computational semantics. Let's look at a few.

Hobbs and Rosenschein (1978), a paper entitled "Making Computational Sense of Montague's Intensional Logic", suggests that Montague semantics is best thought of in terms of procedural semantics rather than model-theoretic semantics; to add substance to this idea, the paper contains a translation of Montague's intensional logic into the functional programming language Lisp. In Schubert and Pelletier (1982), motivated by the need to carry out inference, the authors define a simple translation from a fragment of English (specified using a context-free grammar) into what they call conventional logic. In Landsbergen (1982), on the other hand, the motivation is machine translation. Montague's logic is used as an interlingua: the source language is translated into it, and the resulting logical expression is used to help build a sentence in the target language. In Main and Benson (1983), ideas from Montague semantics are used in a question answering system. Also from this period is Gunji (1981), a PhD thesis which is not only a pioneering contribution to computational semantics, but to computational pragmatics as well. Gunji, realising that a database in a computer can be regarded as a model, defines a system in which incoming sentences are translated into Montague-style logic. While the database is essentially read-only memory as far as his semantic procedures are caused (that is, logical formulas are evaluated in the database without altering it, much as in a standard database query) there are also pragmatic procedures that can modify the model (or as Gunji puts it, induce context changes) which may well affect the semantic evaluations of later sentences.

It is also interesting to look through anthologies on computational linguistics for evidence of interest in computational semantics. Actually, one such collection "Computational Semantics. An Introduction to Artificial Intelligence and Natural Language Comprehension" (Charniak and Wilks, 1976) contains in its title the earliest usage of the term "computational semantics" that we know of. A collection of classic papers that is well worth consulting is Grosz et al. (1986), and the more recent collection Rosner and Johnson (1992) also contains much of relevance. Finally, mention must be made of the collected papers of Robert Moore, a researcher who has made many contributions to computational semantics, ranging from work on semantic construction to intensional semantics (see Moore (1995)).

But it was sometime during the 1990s that computational semantics really began to acquire its own identity. For a start, more solid bridges between formal semantics and computational linguistics began to appear. For example, though the *Handbook of Contemporary Semantic Theory* (Lappin, 1997) for the most part contains articles on traditional themes in formal semantics, it also contains an article devoted to the use of attribute-value structure unification (a standard technique in computational linguistics) to build semantic representations (see Nerbonne (1997)). And in the other direction, Jurafsky and Martin (2000), which has established itself as the standard introduction to speech and language processing, teaches an approach to semantic construction that is based on first-order logic and lambda calculus (see in particular Chapters 14 and 15) and refers to the work of Richard Montague and other researchers in formal semantics.

But perhaps the most important coming-of-age landmark was the founding of the International Workshop on Computational Semantics (IWCS) by Harry Bunt. The first was in 1995, and the workshop (which is held in Tilburg in the Netherlands) has taken place every two years ever since. It is the main meeting place for the computational semantics community, and selected proceedings of two of these meetings are available in book form (see Bunt and Muskens (1999) and Bunt et al. (2001)). A more specialised workshop, Inference in Computational Semantics (ICoS), was held in Amsterdam, The Netherlands in 1999, and since then ICoS has been held in Schloss Dagstuhl, Germany (in 2000), in Siena, Italy (in 2001), and in Nancy, France (in 2003). Selected proceedings of the first three meetings are available as special journal issues (see Monz and De Rijke (2000), Bos and Kohlhase (2003) and Kohlhase (2004)). Finally, in 1999 the Association for Computational Linguistics approved the creation of SIGSEM, a Special Interest Group in Computational Semantics. See

```
http://www.aclweb.org/sigsem
```

for further information.