LOGICS OF AMBIGUITY

A thesis submitted to the University of Manchester
for the degree of Doctor of Philosophy
in the Faculty of Science and Engineering

2004

By
Nickie J. Player
Department of Computer Science

Contents

Abstract

The broad goal of Natural Language Processing is to enable communication between humans and computers without resorting to tightly constrained artificial languages. Unfortunately, natural language is plagued by ambiguity. In fact, ambiguity is the principal reason why the goal of Natural Language Processing remains out of reach. There are currently two approaches to the ambiguity problem: *resolution* and *underspecification.* Resolution aims to decide which of the possible interpretations of an ambiguous expression is 'correct'; (this is usually achieved using contextual or pragmatic information). By contrast, underspecification aims to mirror ambiguity in natural language by ambiguity in the formal ('underspecified') language. The idea behind underspecification is simple enough: to encode the ambiguous content of natural language expressions in single compact representations. A further development of this scheme is to endow the Underspecified Representations with a semantics, and reason over them: so–called 'Underspecified Logic'.

The focus of the thesis is a study of the plethora of Underspecified Logics and the relationships between them. The preferred view of underspecified semantics presented in the thesis is captured by the slogan: *The meaning of an Underspecified Representation is given by the set of its readings.* With this slogan in mind, the expressive power of current Underspecified Representation Languages is investigated in detail.

It is shown that, in terms of expressive power, all current Underspecified Representation Languages are equivalent. This equivalence is established by mathematically precise translation procedures between each of the languages. Furthermore, ambiguous (weak) satisfiability (with respect to *any* Underspecified Representation Language) is reduced to classical satisfiability. Importantly, the size of this reduction is (polynomially) bounded by the size of the input Underspecified Representations.

The interest of these results resides in the link they establish between a mathematically very manageable Underspecified Representation Language and other more linguistically natural languages. The reduction of (weak) ambiguous satisfiability to classical satisfiability suggests that the enterprise of looking for theorem provers for (weak) ambiguous satisfiability is of little theoretical interest. In short, a study of the relationships between the various systems for representing ambiguity and an appraisal of these systems in terms of expressive power form the main contribution of the thesis.

Keywords: *Ambiguity, Logic, Underspecification, Resolution, Expressive Power.*

Declaration

No portion of the work referred to in this thesis has been submitted in support of an application for another degree or qualification of this or any other university or other institution of learning.

Copyright

Copyright in text of this thesis rests with the Author. Copies (by any process) either in full, or of extracts, may be made **only** in accordance with instructions given by the Author and lodged in the John Rylands University Library of Manchester. Details may be obtained from the Librarian. This page must form part of any such copies made. Further copies (by any process) of copies made in accordance with such instructions may not be made without the permission (in writing) of the Author.

The ownership of any intellectual property rights which may be described in this thesis is vested in the University of Manchester, subject to any prior agreement to the contrary, and may not be made available for use by third parties without the written permission of the University, which will prescribe the terms and conditions of any such agreement.

Further information on the conditions under which disclosures and exploitation may take place is available from the head of Department of Computer Science.

Acknowledgements

I would like to thank the following people.

Dr Ian Pratt–Hartmann for his unyielding guidance, support and enthusiasm.

Dr Ian Pratt–Hartmann and his wife Cornelia for their kind offer of accommodation during my long weekends in Manchester.

My wife Kirsty Lauren (to whom I am deeply indebted) for her love, support, patience and understanding throughout my Postgraduate education.

My son Reuben John and my daughter Corrin Rae for their love, and for providing a playful diversion during the final stages of my research.

My parents Linda and Eddie for their support and for teaching me the value of education.

The Engineering and Physical Sciences Research Council (EPSRC) for their financial support throughout my Postgraduate education.

Finally, I would like to thank an anonymous friend who advised me not to begin the thesis with the words "let ϵ be a large negative number"!

Dedication

This thesis is dedicated to my wife Kirsty Lauren Player, to my son Reuben John Player and to my daughter Corrin Rae Player; their love gives me life. I would also like to dedicate this thesis to the memory of my sister Debbie Jane Gray (1972–2000) who will remain forever in my heart.

Footprints in the Sand

One night a man had a dream. He dreamed he was walking along the beach with the Lord. Across the sky flashed scenes from his life. For each scene, he noticed two sets of footprints in the sand: one belonging to him, and the other to the Lord. When the last scene of his life flashed before him, he looked back at the footprints in the sand. He noticed that many times along the path of his life there was only one set of footprints. He also noticed that it happened at the very lowest and saddest times in his life. This really bothered him and he questioned the Lord about it. "Lord, You said that once I decided to follow You, You'd walk with me all the way. But I have noticed that during the most troublesome times in my life, there is only one set of footprints. I don't understand why when I needed You most You would leave me." The Lord replied, "My son, My precious child, I love you and I would never leave you. During your times of trial and suffering, when you see only one set of footprints, it was then that I Carried You."

(Author unknown)

Chapter 1

Introduction

Time flies like an arrow, fruit flies like a banana.

1.1 Ambiguity

Natural language ambiguity is a well–known and much–studied area of research spanning many disciplines including Linguistics, Philosophy, Cognitive Psychology and Computer Science. We (informally) use the term 'ambiguity' in its usual sense; that is, we call a word or sentence 'ambiguous' if it can be understood or interpreted in more than one way. The existence of such 'multiplicities of meanings' is most often viewed as problematic. Moreover, it is widely believed that ambiguity remains 'the main single obstacle for various Natural Language Processing tasks' (van Deemter, 1996, p.1). The word 'ambiguity' is used (in the literature) to refer to a variety of linguistic phenomena; for example, lexical ambiguity, structural ambiguity, scope ambiguity, distributive ambiguity and ambiguity arising because of anaphora and/or ellipsis. These phenomena are only related in that they all give rise to uncertainty at the level of semantic interpretation; that is, they are all 'types' (or 'sources') of ambiguity. We begin the thesis with a taxonomy of the types of ambiguity.

1.1.1 A Taxonomy of Ambiguities

There are six main types of ambiguity: lexical, structural, scope, ambiguities arising from anaphora, ellipses or the interaction of both. Ambiguity occurring because of multiple word sense is called *lexical ambiguity*. Lexical ambiguity is

divided into three sub–groups: *homonymy*, *polysemy* and *categorical ambiguity*. Homonymous words are those with several *unrelated* meanings. For example, 'bark' can refer to the outer ring of a tree or the noise made by a dog (or even a type of boat if written 'barque'). Polysemous words are those with several *related* meanings, each meaning constituting a partial representation of the overall concept. An example of a polysemous word is 'open' since it has many meanings involving unfolding, expanding, being unobstructed or unengaged, and so on. In a nutshell, the meanings of polysemous words can be traced back to a common etymology, whereas those of homonymous words cannot. Polysemy is often quite subtle; consider the sentences 'I walked through the door' and 'I took the door off its hinges'. The two occurrences of the noun 'door' do not have the same sense here (since the first seems to refer to a space and the second to a physical object). Many words are both homonymous and polysemous, having some senses related and some not; 'ground' and 'jam' are such words. A word suffers from categorical ambiguity if it has multiple senses which belong to different syntactic categories. For example, 'sink' may be used as a noun referring to a plumbing fixture, or a verb meaning to become submerged. The task of deciding the syntactic category of an (occurrence of an) ambiguous word is usually referred to as *(part–of–speech) tagging*.

The second type of ambiguity listed is structural ambiguity; a sentence suffers from *structural* (or *syntactic*) *ambiguity* if there is more than one parse of it. Consider the sentence, 'the police shot the rioters with guns'. Ignoring any contextual or common sense information, this sentence might mean that either 'the police used guns to shoot the rioters' or 'the police shot those rioters who were armed with guns'. The phrase structures of these two readings are depicted in figures 1.1 and 1.2 respectively. Consider also the sentence 'sailors only like blonde girls and ladies'; this sentence is (structurally) ambiguous because we do not know whether it is asserted that 'sailors only like blonde girls and blonde ladies' or that 'sailors only like blonde girls and ladies of any hair colour'. This ambiguity occurs because there is uncertainty in precisely what the conjunct 'and' should coordinate. Dik (Dik, 1972, pp.227–249), among others, supports our view that ambiguities arising as a result of coordination are syntactic phenomena.[1] Another

[1]Dik divides (structural) ambiguities arising as a result of coordination into three sub–groups: *functional ambiguities*, *hierarchical ambiguities* and *relational ambiguities*. We omit any further discussion of these subgroups. Instead, we refer the interested reader to Dik's work (Dik, 1972, pp.227–241).

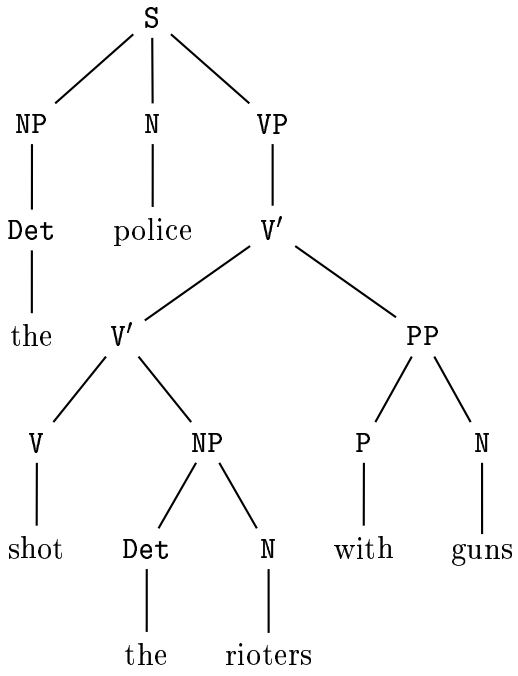


Figure 1.1: A possible phrase structure of the structurally ambiguous sentence 'the police shot the rioters with guns'.

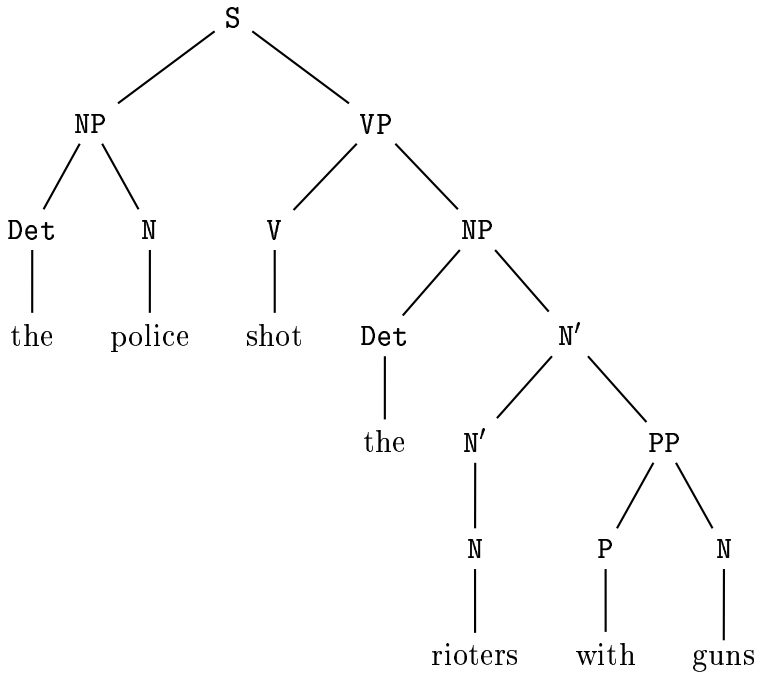


Figure 1.2: An alternative phrase structure of the structurally ambiguous sentence 'the police shot the rioters with guns'.

type of structural ambiguity is *attachment ambiguity*. According to Manning and Schütze: A pervasive problem in parsing natural language is resolving attachment ambiguities (Manning and Schütze, 1999, p.278). A well–known example is the sentence: 'I saw a man with a telescope'. Due to attachment ambiguity, this sentence has two distinct interpretations: An analysis where the prepositional phrase $[\text{with a telescope}]_{\text{PP}}$ is part of the object noun–phrase has the semantics "the man who had a telescope"; an analysis where the PP has a higher attachment (perhaps as daughter of [the] VP) is associated with a semantics where the seeing is achieved by means of a telescope (Hindle and Rooth, 1993, p.1). In fact, many sentences in which a prepositional phrase is preceded by an object noun–phrase suffer from this type of syntactic ambiguity.

The third type of ambiguity listed is scope ambiguity. A sentence suffers from (operator) *scope ambiguity* if it contains more than one operator (for example, negation and a quantifier) and the 'scoping relation' between these operators is uncertain. Consider the sentence, 'every man loves a woman'; this sentence is ambiguous due to operator scope ambiguity (or, more specifically, *quantifier scope ambiguity*). One possible reading is 'there is one woman who is loved by every man' and the other 'every man loves a separate woman'. In the first reading, 'a' *out–scopes* 'every', in the second reading this scoping relation is reversed; we also say that 'a' takes *wide scope* in the first reading and *narrow scope* in the second. Scope ambiguity has become a popular area of inquiry in recent years; however, it is widely accepted that 'while not all scoping can be determined independently of context, certain sentences have strong preferences toward a particular scoping' (Allen, 1995, p.351).[2] Indeed, these preferences may be so strong that we sometimes fail to notice such ambiguities. Closely related to operator scope ambiguity is *distributive* (or *collective*) *ambiguity*, which results from the use of plural quantifiers. Consider for example, the sentence 'two teachers marked three examination papers'. It is not clear whether it is asserted that 'two teachers marked three examination papers each', or that 'two teachers marked three examination papers between them'. It is important to realise that the phrase structure of a scope ambiguous sentence may be unique, and therefore scope ambiguity is not an example of syntactic ambiguity.

Ambiguity may also arise because of *anaphora*; such ambiguities are sometimes called *referential ambiguities*. Consider the sentence 'I saw a cat chase

[2] Allen discusses *preference* at an introductory level (Allen, 1995, pp.159–163).

a mouse; it was black'. Because we are uncertain about what the anaphor 'it' refers to, we cannot be sure whether it is the cat or the mouse which is black. We share Dalrymple's view that anaphoric ambiguity is not an example of structural ambiguity and that:

> The relation between an anaphor and its antecedent is often represented diacritically, by coindexing. It is well–known, though, that this relation is in fact not a syntactic relation, but a particular semantic relation. (Dalrymple, 1993, p.99)

Ellipsis too can cause ambiguity; for example, in sentences like 'Peter knows a richer man than Paul'. Because of ellipsis (that is, omission) we can interpret this sentence as meaning either 'Peter knows a man who is richer than all the people Paul knows' or 'Peter knows some man who is richer than Paul'. Although we have chosen to treat ambiguity arising as a result of ellipsis autonomously, the question of whether this type of ambiguity should be classed as structural or not is controversial (Aronoff and Rees–Miller, 2001, p.426). We will not discuss the argument here, instead we recommend Lappin's work (Lappin, 1996, pp.145–175) as a good starting point. Finally, ambiguity can arise because of the interaction between anaphora and ellipsis; such ambiguities are sometimes referred to as *strict/sloppy ambiguities*. For example, consider the sentence 'John likes his car; Michael does too'. Ambiguity occurs here because of the interaction between the anaphor 'his' and the ellipsis permitted by the phrase 'does too'; so that, we cannot be sure whether to interpret the second clause as 'Michael likes John's car' ('strict' reading), or 'Michael likes his own car' ('sloppy' reading). The standard approach to dealing with ellipsis is based on the following hypothesis:

> The completed phrase structure of the elliptical clause corresponds to the structure of the previous clause. Once the structural correspondence between the two clauses is identified, the semantic interpretation of the previous clause can be updated with the new information in the elliptical clause to produce the new interpretation. (Allen, 1995, p.451)

Suppose that the first clause of the sentence 'John likes his car; Michael does too' has the following semantic interpretation:

$$\text{like}(\text{john}\,,\,\text{car–belongs–to}(\text{john})).$$

To generate the semantics of the second clause, Allen suggests that we must abstract 'john' from the semantic interpretation of the first clause, and there are two distinct ways of doing this:

$$\lambda p\ [\text{like}(p, \text{car–belongs–to}\,(\text{john}))] \quad \text{and}$$
$$\lambda p\ [\text{like}(p, \text{car–belongs–to}\,(p))].$$

resulting in the two distinct interpretations of the original sentence. In the sequel, we indicate the source of ambiguities only where such a specification is significant.

1.1.2 Disambiguation and Resolution

We refer to the process of obtaining the possible readings of a sentence as *disambiguation*. For example, disambiguating the sentence 'the ball was wonderful' yields two readings, resulting from the (lexical) ambiguity of the word 'ball'. Clearly, the information communicated by an ambiguous sentence is very much dependent upon how we disambiguate it. On hearing an ambiguous statement, it is sometimes impossible to determine which reading(s) the speaker intended to communicate. It is important from the outset to be clear about the difference between the processes of 'disambiguation' and 'resolution'. In resolving ambiguity we must decide which particular reading(s) the speaker intended to communicate. In disambiguation, by contrast, we simply list all possible readings without any preferences.

Unfortunately, all natural languages display a remarkable degree of ambiguity. However, in practice many ambiguities are resolvable; for example, Hirst claims that 'although many sentences of English have more than one parse, there is usually a unique preferred parse for a sentence after semantics and discourse are considered' (Hirst, 1987, p.9). Once an ambiguity has been resolved it becomes unproblematic and therefore, faced with an ambiguity, perhaps the best line of attack is resolution. This belief is shared by van Deemter (among others): 'ambiguities that can be resolved, by taking context, prosody, or even common sense knowledge into account are probably better resolved' (van Deemter, 1998, p.24). But what if insufficient information is available to allow resolution of all ambiguities?

1.2 Underspecification

One way of dealing with unresolved ambiguities, which has become popular in Computational Semantics, is (*semantic*) *underspecification*. Indeed, this approach has become so popular that 'it often seems as if semantics has entered an age of underspecification' (Blackburn and Bos, 1999a, p.83). The basic idea behind underspecification is simple enough: to encode the ambiguous content of natural language expressions in single compact (Underspecified) Representations. Underspecification may be summed up as follows:

> The key idea is to derive a single, compact *description* of all readings instead of the (exponential number of) readings themselves. (Egg et al., 2001, p.457)

There are arguments against allowing ambiguities to occur in formal languages. In fact, artificial languages, such as the predicate calculus, are most often devised to eliminate expressions with more than one interpretation. Enderton asserts that 'formal languages allow us to escape from the imprecision and ambiguities of natural language' (Enderton, 1972, p.15). Cann expresses the opinion that avoiding ambiguity is a central feature of the 'logical language' L_p: L_p is 'a logical language into which sentences of English are translated in order to circumvent the problems of ambiguity and underdeterminacy found in the object language' (Cann, 1993, p.27). Thomason goes further still, claiming that 'there is no serious point to constructing an artificial language that is not disambiguated' (Thomason, 1974, p.6). However, Alshawi offers an opposing view:

> ... natural languages exhibit such partiality with respect to quantifier scope, anaphora, underspecified relations, and so on. It would therefore seem reasonable that expressions of formal languages that are meant to capture the meaning of natural language sentences should also exhibit this type of partiality. (Alshawi, 1996, p.146)

Having outlined some of the arguments surrounding underspecified representation we make no further reference to this issue in the sequel. Instead, we concentrate on studying the relationships between the plethora of current Underspecified Representation Languages. In fact, our study of underspecification will go deeper than representation alone; we will study Underspecified Logics. An

Underspecified Logic is (informally) defined as 'a pair consisting of a proper underspecified semantic representation formalism and a deductive component that directly operates on these structures' (König and Reyle, 1996, p.1). Given an ambiguous natural language (declarative) sentence, we would like to find an Underspecified Representation with which we can reason. The basic idea is to reason with ambiguous formulae, reach ambiguous conclusions and worry about resolution afterwards, if at all. To see why such an approach might be successful, consider the scope ambiguous sentence *every man loves a woman.* If we analyse this statement in isolation, then it is not clear whether it is asserted that 'there exists one woman whom is loved by every man' (for example, Marilyn Monroe), or that 'every man loves a possibly separate woman' (for example, his wife). However, if we choose to ignore this ambiguity, we may still be able to draw sensible inferences from this sentence. For example, if it is further asserted that *John is a man*, then we may infer that *John loves a woman*, from either reading of the original sentence. Indeed, it is widely believed that 'not only is it very difficult to resolve quantifier scope, it is also often unnecessary' (Copestake et al., 1999, p.3). Van Deemter makes the more general point that 'partial understanding [of ambiguous sentences] is often enough for making inferences' (van Deemter, 1996, p.204). There are numerous arguments in the literature suggesting why Underspecified Logic might provide a sensible way to proceed; we now turn our attention to an informal discussion of some of these arguments.

Consider the well–known example due to Hobbs:

> A politician can fool most voters on most issues most of the time, but no politician can fool all voters on every single issue all of the time.[3] (Hobbs, 1983, p.1)

Each of the two clauses (separated by the comma) permits 4!=24 different orderings of its quantifiers. Disambiguating the clauses independently yields 4!.4!=576 readings in total. Even though this statement has many readings (though some are equivalent), humans are able to make sense of it with relative ease. Muskens claims that:

> ... ambiguities in natural language can multiply so fast that no person

[3]Hobbs' example is actually "In most democratic countries most politicians can fool most of the people on almost every issue most of the time" which Hobbs claims has 120 different readings, or 'quantifier scopings' (Hobbs, 1983, p.1).

> or machine can be expected to process a text of even moderate length by enumerating all possible [readings]. (Muskens, 1999, p.311)

This observation has led to the conjecture that humans reason in an underspecified manner, since it seems unlikely that we can carry out the calculations required to reason over all readings. Poesio refers to this proposal as the *underspecification hypothesis* (Poesio, 1994, p.4). However, there are arguments supporting the claim that humans find processing ambiguous sentences a more difficult task than processing unambiguous sentences. For example, there is experimental evidence (Mohanty, 1983) that there is a significant increase in a person's heart rate when required to interpret ambiguous sentences (compared to unambiguous sentences)! We now consider two practical reasons for adopting an underspecified approach to ambiguity.

Reasoning over natural language is fraught with difficulties. Given an ambiguous sentence there is sometimes little evidence to suggest which reading(s) may be disregarded. Furthermore, such evidence usually comes from contextual information or semantic insight. This situation presents serious difficulties for computer scientists interested in Natural Language Processing because, in a nutshell, 'computers are *stupid*. We can't appeal to their semantic insight because they don't have any' (Blackburn and Bos, 1999, p.30). As a consequence, many computer systems with natural language input suffer from *interpretational deadlock*. That is, many computer systems require resolution of all ambiguities. If a system ceases operation because of unresolved ambiguity it is said to suffer from interpretational deadlock. It is also well–known that reasoning over all the readings of a sentence leads to a combinatorial explosion.[4] Suppose that we have n sentences, each of which is ambiguous between just two distinct readings. Multiplying up each ambiguity means that, when combined, these sentences will have 2^n readings. Underspecification is a proposed solution to the problem of deadlock. It is also widely believed that underspecification reduces the computational burden experienced when reasoning with ambiguous information. In particular, it is widely believed that reasoning with Underspecified Representations is (computationally) more efficient than reasoning over an enumeration of readings. In fact, underspecification is often described as 'a recent approach to controlling the combinatorial explosion caused by ambiguity' (Egg et al., 2001, p.457). This

[4]Poesio provides an introduction to this problem (Poesio, 1996, pp.1–3).

view is supported by Koller: 'these [underspecified] approaches all aim at controlling the combinatorial explosion of sentences with multiple ambiguities' (Koller et al., 2003, p.1). Indeed, Bos attributes the birth of underspecified semantics to computational considerations and claims that 'the so called Combinatorial Explosion Puzzle, a well–known problem in this area, can be successfully tackled using Underspecified Representations' (Bos, 1995, p.133). However, to date, there is no evidence to support or refute this claim. That is, the issue remains of whether reasoning with Underspecified Representations really is computationally less expensive than reasoning over an enumeration of readings. We will explore a number of computational issues in the sequel; however, an evaluation of the expressive power of the various Underspecified Representation Languages will be the main focus of this thesis.

1.3 Thesis Plan

As we have said, there are currently two approaches to the ambiguity problem: resolution and underspecification. Resolution aims to decide which of the possible interpretations of an ambiguous expression is 'correct'; this is usually achieved using contextual or pragmatic information. By contrast, underspecification aims to mirror ambiguity in natural language by ambiguity in the formal (Underspecified Representation) Language. Our overall aim is to evaluate underspecification. In the next chapter (chapter two) we investigate the legitimacy of this relatively new enterprise; in particular, we report on the success of the older and (notionally) simpler approach – resolution. Once we have established that there is a need for underspecification, our goal will be to gain a clear understanding of the plethora of Underspecified Logics and the relationships between them.

Underspecified Logics consist of two components: a mathematically precise language, the expressions of which (compactly) represent the ambiguous content of natural language sentences, and a framework for reasoning over these expressions. We will treat these components independently to reflect our belief that representation and reasoning are orthogonal issues.

In the third chapter we begin our study of underspecification; in particular, we define the syntax and disambiguation procedure for each current Underspecified Representation Language. We will compare these languages with one another and assess their (relative) expressive power in chapter five. We gain a clear

understanding of the relative expressive power of each language by providing translation procedures between them. However, we will insist that in each case, (the size of) our translated formulas is polynomial in the size of the formulas from which they were translated. Without this restriction our results would in some sense destroy the original motivation behind underspecification – compactness of representation.

In chapter four, we study the various frameworks for reasoning with ambiguous formulas; we will split the current proposals into two groups: 'recursive satisfaction definitions' and 'non–recursive satisfaction definitions'. We call an underspecified semantic framework a *recursive satisfaction definition* if it is such that the interpretation of any non–atomic formula is defined only in terms of the interpretations of its immediate subformulas. By contrast, we shall see that *non–recursive satisfaction definitions* may be captured by the slogan: *The meaning of an Underspecified Representation is given by the set of its readings.* The best–known recursive satisfaction definition is a proposal based on Partial Logic[5] due to van Eijck and Jaspars (van Eijck and Jaspars, 1996, pp.5–11). However, we argue that only non–recursive satisfaction definitions provide sensible semantic frameworks for underspecification. We shall justify this view in the sequel; for now, we note that it is shared by Schiehlen, among others:

> Two things are required for underspecification: (1) a formalism for compact representation of interpretation classes (*Underspecified* Representations) and individual interpretations (*fully specified* representations) and (2) a *disambiguation device* to connect Underspecified Representations with fully specified representations. (Schiehlen, 1997, pp.1–2)

In short, a study of the relationships between the various systems for representing ambiguity and a generic study of the expressive limits of Underspecified Representation Languages form the main contribution of the thesis.

Our aim in the following chapter is to assess the need for underspecification. Actually, we present the next chapter as evidence that ambiguity resolution is limited, and therefore we will argue that underspecification is worth investigating.

[5]Partial Logic was introduced by Blamey (Blamey, 1986).

Chapter 2

Statistical Resolution

The fisherman's catch was miserable.

Ambiguity makes automated natural language understanding (or, more correctly, Natural Language Processing – NLP) a difficult task. Therefore, we would very much like to eliminate all ambiguities. Resolution aims to do exactly this; that is, resolution aims to decide between the possible readings of an ambiguous expression. In this respect, resolution is (notionally) the simplest approach to the ambiguity problem. However, we will argue that resolution is not completely adequate and therefore our study of underspecification is worthwhile. In arguing our case, we provide a flavour of (the best–known) current statistical approaches to resolution and an indication of how successful these approaches are. In this chapter, we will abuse some of the terminology introduced earlier; in particular, we refer to the task of *resolving* lexical ambiguities as 'word sense *disambiguation*'. Our reason for doing so is simply that we wish to use the terminology in the same way that it is currently used in the literature and we believe that our use of the term 'disambiguation' should be clear from context anyway.

2.1 Resolving Lexical Ambiguity: Word Sense Disambiguation

Lexical ambiguities are pervasive even in 'specialised' texts such as the journals *Time* and *Computer Science* (Harper, 1957a), (Harper, 1957b), (Krovetz and Croft, 1992), (Krovetz, 1997). It is usually viewed as desirable, if not essential,

that Natural Language Processing systems can resolve lexical ambiguities. The kind of Natural Language Processing systems we have in mind serve a variety of purposes, including: machine translation (between languages, for example, English and French), information retrieval[1], hypertext navigation (for example, keyword search), automated speech processing and so on.[2]

We begin our presentation with some terminology. The terms 'context window', 'context frame' and 'micro–context' (of a word occurrence) appear frequently in the literature. The concept underlying these terminologies was first elucidated by Weaver:

> If one examines the words in a book, one at a time as through an opaque mask with a hole in it one word wide, then it is obviously impossible to determine, one at a time, the meaning of the words... But if one lengthens the slit in the opaque mask, until one can see not only the central word in question but also say N words on either side, then if N is large enough one can unambiguously decide the meaning of the central word. (Weaver, 1949, p.21)

Weaver does not indicate how large N should be, but this issue has been investigated, first by Kaplan (Kaplan, 1950) and later by Choueka and Lusignan (Choueka and Lusignan, 1985). In the sequel, we will use the term *context* instead of 'context window', 'context frame' or 'micro–context'.

2.1.1 The Problem

The resolution of lexical ambiguities in unrestricted text is one of the most difficult tasks of Natural Language Processing (Dagan and Itai, 1994, p.1). The task of resolving lexical ambiguities is frequently referred to as *word sense disambiguation*. The word sense disambiguation problem can be stated as follows:

Problem 2.1.1 (Word Sense Disambiguation). The aim of word sense disambiguation is to map each word (occurrence) in a particular text to its 'correct sense'; that is, to determine which of the senses of an ambiguous word is invoked by a particular occurrence of that word.

[1] There is evidence to suggest that resolving lexical ambiguities does not significantly improve the performance of information retrieval systems (Krovetz and Croft, 1992), (Krovetz, 1997), and evidence that it does (Strzalkowski, 1995)!

[2] The question of whether or not these endeavours benefit at all from word sense disambiguation is addressed by Kilgarriff (Kilgarriff, 1997).

But what is meant by the term 'sense' (of a word) and how do we know whether or not a sense is 'correct'? Unfortunately, neither of these fundamental questions is easy to answer. Therefore, we now take a brief excursion to discuss each of them in turn.

We view the term 'sense' as being synonymous with the term 'possible reading'; 'the precise definition of a sense is, however, a matter of considerable debate within the community' (Ide and Véronis, 1998, p.3). We would like to define word senses as the 'mental representations' of the meanings of words, (since we are trying to imitate human natural language understanding). However, it has proved very difficult to model such mental representations in practice. Most research following this approach involves experiments in which human subjects are given the quite unnatural task of grouping word occurrences with similar meanings. Unfortunately, it has been shown that agreement on these groupings between human subjects is low (Jorgenson, 1990). In practice, the (lists of) senses of words are most often taken from everyday dictionaries, thesauri or bilingual dictionaries. An immediate problem with all of these sources is the inherent disparity between them; for example, in the (number, grouping and content of) word definitions (or 'senses') given in different dictionaries. That is, the use of these sources raises the question: which particular dictionary should we use, and why?

Suppose for a moment that we are completely satisfied with our set of word senses; the question still remains – what does it mean to say that one (or more[3]) of these senses is 'correct' for a particular word occurrence? Given the subtlety of many natural language ambiguities it is hardly surprising that human agreement on word sense assigning tasks is less than perfect. The actual level of human agreement in experimental conditions ranges between 65% and 99% depending upon whether the words under consideration have distinct senses (high agreement) or closely related senses (low agreement), (Manning and Schütze, 1999, p.234). This fact alone suggests that even the most sophisticated word sense disambiguation algorithm will fail to be completely adequate. However, we would still like to know just how successful state–of–the–art word sense disambiguation algorithms are.

[3]Kilgarriff suggests that it is common for senses to be induced simultaneously in natural language. For example, Kilgarriff claims that the word 'competition' in the sentence 'For better or worse, this would bring competition to the licensed trade' induces two senses: 'the act of competing' and 'the competitors' (Kilgarriff, 1993).

It would seem unfair (but not impossible!) to expect an automated Natural Language Processing system to out–perform humans. Perhaps therefore, we should only measure machine performance (at word sense disambiguation tasks) against human capability (at the same tasks). Indeed, human performance is usually used as an 'upper bound' for evaluating the success of word sense disambiguation procedures. It is also sometimes appropriate to set a lower bound for evaluating word sense disambiguation procedures. The need for a lower bound is most evident for those words with a strongly preferred sense, regardless of context. For example, suppose that some word w has just two senses s and s' and that in some (sufficiently large) corpus 95% of all occurrences of w invoke sense s. It is trivial to write a disambiguation algorithm with an approximate 5% 'error rating' on w, (by mapping all occurrences of w to s regardless of context). Clearly, under these circumstances, any salutary word sense disambiguation algorithm would have to map *at least* 95% of occurrences of w to their correct sense!

Fortunately, for theoretical purposes, we can circumvent the difficulties discussed above with a neat simplification of the problem. The basic idea is to ignore any real ambiguity in the source text and instead, create artificially ambiguous pseudo–words[4], such as 'river–banana' which we define to have just two senses, 'river' and 'banana'. We simply replace all occurrences of 'river' and all occurrences of 'banana' in the original text with the term 'river–banana'. The idea is then to treat the text with the pseudo–words as ambiguous (the source text), and the original text (containing real ambiguous words) as disambiguated. The 'senses' of a pseudo–word are well–defined, they are simply the terms to the immediate left and the immediate right of the hyphen symbol occurring in it. We can also easily manufacture our pseudo–words so that it is always trivial to determine their 'correct' senses.

Having briefly discussed some of the issues surrounding the word sense disambiguation problem, we now report on the success of the best–known solutions to it.

2.1.2 Current Solutions

Quite recently (that is, in the past fifty years), it was widely believed that the word sense disambiguation problem was intractable, and that sense ambiguity could

[4]Pseudo–words were first used in this way by Yarowsky (Yarowsky, 1993).

not be resolved by electronic computer either current or imaginable (Bar–Hillel, 1964, p.175). The modern sceptic (at worst) claims that word sense disambiguation is a so–called *AI–complete* problem, meaning that its solution presupposes "the synthesis of all human–level intelligence". We shall see that current word sense disambiguation procedures are not completely adequate – the best of which resolve approximately 96% to 97% of lexical ambiguities correctly. We will therefore argue that there is room for a fresh approach to lexical ambiguity – namely, underspecification.

Early attempts at automated word sense disambiguation were impeded by the unavailability of machine–readable lexical resources. Indeed, 'hand–crafting' such resources proved to be a prodigious task; the difficulty caused by this problem is sometimes referred to as the 'knowledge acquisition bottleneck'. Fortunately, large scale machine–readable lexical resources (such as dictionaries, thesauri and corpora) became widely available during the 1980's. The arrival of these resources led to the birth of a new breed of word sense disambiguation procedure. Word sense disambiguation algorithms are often split into two categories: those which are described as being *knowledge–driven* and those which are *data–driven* (or *corpus–based*). Knowledge–driven word sense disambiguation techniques use external sources, such as dictionaries, thesauri and encyclopedias. By contrast, data–driven techniques exploit knowledge of previously disambiguated instances of the target words from corpora. We now briefly describe current (well–known) knowledge–driven followed by data–driven word sense disambiguation procedures.

Knowledge–Driven Word Sense Disambiguation

In this section, we describe dictionary–based, thesaurus–based and bilingual–dictionary–based word sense disambiguation techniques. The simplest dictionary–based word sense disambiguation algorithm is one in which we view each alternative dictionary definition of a word as 'describing' a distinct sense of that word. To disambiguate a word occurrence w, we look at the words occurring in each of its dictionary definitions (one definition at a time) and then try to match each of these words (taken from the dictionary definitions) to the words in the context of w. To disambiguate, we simply select (the sense associated with the dictionary definition with) the best match. This method was first formalised by Lesk (Lesk, 1986), who provides the following example:

Example 2.1.2. Suppose that in a particular dictionary the word *cone* is assigned the following two alternative definitions:

1. a mass of ovule–bearing or pollen–bearing scales or bracts in trees of the pine family or in the cycads that are arranged usually on a somewhat elongated axis,

2. something that resembles a cone in shape: such as a crisp cone–shaped wafer for holding ice cream.

(Lesk, 1986, p.24)

If the words 'pine' and 'tree' appear in the context of some occurrence of *cone* then we should assign that occurrence the sense described by definition 1 above; assuming that overall, there is a bigger overlap between (the words in) the context of the occurrence of 'cone' and the words in definition 1 than there is between the context of 'cone' and the words in definition 2.

In general, for any word occurrence w, each sense of w is given a 'score' based on the overlap between the *bag*[5] of words in the dictionary definition[6] of w (describing that sense) and the *bag* of words in the context of w. The sense with the greatest score is then selected as being the preferred (or 'most likely') disambiguation of w. In practice, Lesk's method only achieves between 50% and 60% correct disambiguation (Lesk, 1986, p.26). A simple modification of Lesk's technique is to expand the contexts of word occurrences, by adding to each context all of the synonyms (obtained from a thesaurus) of each of its elements. This modification was first suggested by Pook and Catlett (Pook and Catlett, 1988). However, Pook and Catlett do not evaluate the success of their modification.

Other dictionary–based word sense disambiguation algorithms typically use some measure of 'relatedness' between words (which are sometimes referred to as 'metrics of semantic preference'). For any two words w and w', computing the metric of semantic preference between w and w' invariably involves counting the number of words that co–occur in at least one (dictionary) definition of w and at least one (dictionary) definition of w'. These metrics are then used to compare the

[5] A bag is just a set with repeated elements.

[6] The bag of words occurring in the dictionary definitions of a word w is sometimes referred to as the *signature* of w.

target word with words in the surrounding context. To disambiguate a word w, we select the sense of w which is most closely related to the words in the context of w. This approach is generally attributed to Wilks and Fass (Wilks and Fass, 1990). Unfortunately, Ide and Véronis report that this method achieves just 45% accuracy on sense identification, and 90% accuracy on homograph identification in experiments on the single word 'bank' (Ide and Véronis, 1998, p.9). That is, Ide and Véronis report that Wilk's method is twice as successful at choosing between homonymous (related) senses of the word 'bank' as it is at choosing between polysemous (unrelated) senses.

Another (relatively) popular approach is to exploit 'meta–level knowledge' of some sort, for example, the type (or subject matter) of the target text (Krovetz and Croft, 1989), (Guthrie et al., 1991), (Liddy and Paik, 1993). Usually, this meta–level knowledge is 'written into' the algorithms by assigning every (dictionary definition of each) word a *box code* and a *subject code*. Intuitively, box codes encode 'type restrictions' on nouns, adjectives and on (the arguments of) verbs; examples of box codes are: **abstract**, **human** and **collective**. Subject codes classify words by subject; for example, **physics**, **popular–culture** and so on. Stevenson and Wilks report that an implementation which uses only subject codes achieves 79% precision (Stevenson and Wilks, 1999, p.1). Therefore, we argue that dictionary–based word sense disambiguation techniques are limited; we now turn our attention to the use of thesauri in word sense disambiguation.

The rationale behind the use of thesauri, (for example, Roget's International Thesaurus, sixth edition – Kipfer & Chapman, 2002), for word sense disambiguation can be summarised as follows: 'the categories listed for a word in Roget's index tend to correspond to sense distinctions; thus selecting the most likely category provides a useful level of sense disambiguation' (Yarowsky, 1992, p.1). The basic idea is that the categories (or 'semantic categories') of the words in a context should determine the 'semantic category' of the context as a whole, and the semantic category of the context as a whole determines the sense of the target word(s). Walker was the first author to suggest this method (Walker, 1987, p.254). However, in an experiment on the five (highly) ambiguous words *interest, point, power, state* and *terms*, Black reports only 50% correct disambiguation using Walker's technique (Black, 1988, p.187). Another (well–known) thesaurus–based word sense disambiguation algorithm, due to Yarowsky, consists of the following three steps, which we paraphrase from Yarowsky (Yarowsky, 1992,

p.2):

1. Scan the corpus[7] for occurrences of words which appear as categories in the thesaurus, and collect the (100 word) context of each such occurrence.

2. Identify 'salient' words in the collective context, and weight appropriately. Intuitively, a salient word is one which appears significantly more often in the contexts of a category than at other points in the corpus, and hence is a better than average indicator for that category. This is formalised with a mutual–information–like estimate $\frac{P(w|\mathrm{RCat})}{P(w)}$, the probability of a word w appearing in the context of a Roget category divided by the overall probability that w occurs in the corpus.

3. Use the resulting weights to predict the appropriate (thesaurus) category for the target word(s). When any of the salient words appear in the context of the target word, there is evidence that the target word belongs to the indicated category. If several such words appear, the evidence is compounded. Using Bayes' rule, we sum their weights, over all words in context, and determine the category 'Rcat' which maximises:

$$\sum_{w \in C} \log \left(\frac{P(w|\mathrm{Rcat}).P(\mathrm{Rcat})}{P(w)} \right).$$

On the twelve ambiguous words tested, (*star, ride, gallery, core, bass, bow, taste, interest, issue, duty, sentence* and *slug*) Yarowsky reports an overall accuracy of 92% (Yarowsky, 1992, p.6). Therefore, we argue that thesaurus–based word sense disambiguation is also limited. We now turn our attention to the use of bilingual dictionaries in word sense disambiguation.

The basic idea behind the use of bilingual sources in word sense disambiguation is:

> ... to use a bilingual lexicon to find all possible translations of each lexically ambiguous word in the source sentence, and then use statistical information gathered from target language corpora to choose the most appropriate alternative. ... Our model defines the different "senses" of a source word to be all its possible translations to the

[7] Yarowsky uses (the June 1991 version of) Grolier's Encyclopedia.

> target language, as listed in the bilingual lexicon. (Dagan and Itai, 1991, pp.4–5)

Dagan and Itai map an ambiguous construct from one language to another, in the hope that they will obtain representations in which each sense corresponds to a distinct word. This approach benefits from the fact that typically, an ambiguous word and its translation (into another language) do not share the same senses (this is especially true in unrelated languages); that is, 'this approach exploits the differences between mappings of words to senses in different languages' (Dagan and Itai, 1991, p.1). For example, the (English) word 'duty' has two (related) senses – 'a kind of tax' and 'an obligation'. However, no single word in French has the same ambiguity. Instead, the word 'duty' is usually translated to either of the French words *droit* (the tax) or *devoir* (the obligation), this example is used by Charniak (Charniak, 1996a, p.148). Having translated each ambiguous word into a set of words in a different language using a bilingual dictionary, we still need to choose between these translated words – since the translated words correspond to the senses of the target word. Dagan and Itai's answer is simply to count occurrences of *droit* and *devoir* (in a French corpus) and use their relative frequencies to estimate $P(\text{tax})$ and $P(\text{obligation})$ respectively. That is, 'the plausibility of selecting a target word is determined by the plausibility of the tuples [that is, foreign translations] which are obtained from it. The plausibility of alternative target tuples is in turn determined by their relative frequency in the corpus' (Dagan and Itai, 1991, p.8). Dagan and Itai incorporate confidence intervals into their algorithm (making decisions only when their confidence is 90% or higher) based on the maximum–likelihood estimator (Agresti 1990). When tested on a set of examples of Hebrew to English translations, selected randomly from foreign news sections in the Israeli press, this model was only applicable to 70% of the ambiguous words occurring in the data, and it disambiguated just 92% of these words correctly – amounting to a 64.4% success rating (Dagan and Itai, 1991, p.3). This low value should not surprise us, since Dagan and Itai's method ignores the contexts of the target words and instead uses the relative frequencies of their translations (senses) in a foreign corpus. We have classified Dagan and Itai's method as being knowledge–driven since their technique uses a bilingual lexicon for defining the senses of words. But in fact, since Dagan and Itai use the relative frequencies of words in the foreign corpus to estimate the probabilities of the alternative senses of words, their algorithm is also data–driven.

As we have seen, the usefulness of dictionaries, thesauri and bilingual dictionaries in word sense disambiguation is somewhat limited, mainly because of inconsistencies and the lack of sufficient pragmatic information in these sources. It is widely believed that the kind of pragmatic information needed can be obtained from corpora. Consider for example the word 'ash'. It is quite possible that 'ash' might co–occur frequently with the words 'cigarettes', 'tobacco' and 'tray' in texts and that none of these words occur in any of the dictionary definitions of 'ash'. Data–driven word sense disambiguation is based on the hypothesis that corpora contain useful pragmatic information which cannot be obtained from dictionaries. We now investigate the extent to which such pragmatic information can improve resolution techniques for lexical ambiguity.

Data–Driven Word Sense Disambiguation

The idea behind data–driven word sense disambiguation is to disambiguate words using information gained by 'training' on corpora, rather than using information taken from external knowledge sources such as dictionaries. This training can be carried out on either 'disambiguated' (that is, unambiguous) corpora or 'raw' (that is, ambiguous) corpora. We begin by looking at those word sense disambiguation algorithms which use disambiguated corpora; that is, those techniques which depend upon 'supervised learning' from 'sense–tagged' corpora.

The best–known such algorithm is the Bayesian Classifier (Gale et al., 1992b). As we have indicated, this algorithm assumes that we have a corpus available in which each ambiguous word is labelled with its correct sense.[8] Gale's algorithm is a specialised version of the so–called *Naive Bayes Classifier*. We therefore begin by describing the *Bayes Classifier*, which uses *Bayes' Decision Rule* when choosing a class – the rule that minimises the probability of error (Duda and Hart, 1973). Suppose that we wish to disambiguate some word occurrence w in context c. Suppose also that w has senses $s_1, \ldots, s_n$ where $n \in \mathbb{N}$. Then Bayes' Decision Rule can be applied as follows:

Rule 2.1.3 (Bayes' Decision Rule). Decide s if $P(s|c) > P(s_k|c)$ for all $s_k \neq s$ with $s \in \{s_1, \ldots, s_n\}$ and $1 \leqslant k \leqslant n$.

Rule 2.1.4 (Bayesian Classification). Assign w sense s where

[8] In fact, Gale et al. use a large corpus of parallel English and French, taken from the "Hansards" – the proceedings of the Canadian parliament.

$$\begin{aligned} s &= \arg\max_{s_k} P(s_k|c) \\ &= \arg\max_{s_k} \frac{P(c|s_k).P(s_k)}{P(c)} \\ &= \arg\max_{s_k} P(c|s_k)P(s_k) \quad \text{since } P(c) \text{ is constant.} \end{aligned}$$

We use the notation $\arg\max_x f$ to denote the argument x for which f has maximal value.

The application of the Naive Bayes Assumption in this case amounts to the assumption that the elements of the context c are conditionally independent. That is, we make the following independence assumption:

Rule 2.1.5 (Naive Bayes Assumption). Let v range over the *bag* of words occurring in context c. Then,

$$P(c|s_k) = P(\{v|v\widehat{\in}c\}\,|s_k) = \prod_{v\widehat{\in}c} P(v|s_k).$$

Note that, we use the symbol '$\widehat{\in}$' instead of (the regular) '$\in$' to indicate that we are referring to membership of a bag, not set membership.

Because this model ignores any order and structure in c, it is sometimes referred to as the 'bag of words model'. We can now modify our decision rule using the Naive Bayes Assumption.

Rule 2.1.6 (Decision Rule for Naive Bayes). Assign w sense s if

$$s = \arg\max_{s_k}\Big[\prod_{v\hat{\in}c} P(v|s_k)\cdot P(s_k)\Big].$$

We compute $P(v|s_k)$ and $P(s_k)$ using our labelled training corpus as follows:

$$P(v|s_k) = \frac{C(v,s_k)}{C(s_k)} \quad \text{and} \quad P(s_k) = \frac{C(s_k)}{C(w)},$$

where $C(v,s_k)$ denotes the number of occurrences of v in all of the contexts of w which are tagged with sense s_k in the corpus, $C(s_k)$ is the number of occurrences of w with sense s_k in the corpus, and $C(w)$ is simply the total number of occurrences of w in the corpus.

Gale et al. report just 90% correct disambiguation on the five words *duty, land, language, position* and *sentence* in the Hansard (English–French) corpus (Gale et al., 1992). Charniak (Charniak, 1996a, pp.147–150) and Manning and Schütze (Manning and Schütze, 1999, pp.235–239) present detailed overviews of Gale et al.'s algorithm.

As we have indicated, Gale's algorithm ignores any structure in contexts, (the Naive Bayes Assumption). We now turn our attention to a technique, due to Brown et al. (Brown et al., 1991), in which we search contexts for a single feature which reliably indicates which sense of the target word is being used. In this respect, Brown's method stands in direct opposition to Gale's (modified) Bayesian Classifier. But what sort of features give us the sort of information we need for word sense disambiguation? Consider the French word *prendre*, which translates into either (of the English words) 'take' or 'make'. If *prendre* occurs with *décision* as its object then it most often translates to 'make'; if *prendre* takes *measure* as its object then it translates to 'take'. Therefore, (in this case) the *object* of the target word *prendre* is a good indicator of its sense. Similarly, the *tense* of the French word *vouloir* is a good indicator of how we should translate it into English, (past tense suggests 'to want' and conditional tense suggests 'to like').[9] Brown et al. achieve a 20% improvement in the performance of a machine translation system as a result of incorporating their algorithm into that system (Manning and Schütze, 1999, p.241). However, even the improved algorithm disambiguates just 45% of sentences correctly.

Apart from high error, these 'labelled–corpus–based' methods suffer from another serious disadvantage: disambiguated corpora are expensive and difficult to obtain. Consequently, these methods are often tested on a very small number of words, usually of order 10. Despite the expense, several attempts have been made to manufacture suitable artificial (that is, disambiguated) corpora. There are two ways of creating artificial corpora: the first is to somehow disambiguate a raw corpus manually and label sense occurrences; the second is to create artificial ambiguities using pseudo–words (as we discussed earlier in this chapter). The first of these methods (sense labelling a raw corpus) is usually achieved using a 'bilingual corpus'. Bilingual corpora are often produced by multi–national organisations such as the European Union, the United Nations or the Canadian

[9]Both of these examples are copied from Brown's paper (Brown et al., 1991).

Parliament (which routinely produce documents in several languages). Such transcripts provide good sentence alignment between translations and have proved to be relatively successful tools in word sense disambiguation. Brown et al. (Brown et al., 1991) and Gale, Church and Yarowsky (Gale, Church and Yarowsky, 1992a) use the Canadian Hansard for word sense disambiguation.

The lack of disambiguated data has led some researchers to experiment with the use of raw (ambiguous) corpora; techniques which use raw corpora are commonly referred to as *unsupervised disambiguation* techniques. Unsupervised disambiguation cannot assign senses (that is, semantic labels) to word occurrences; instead, unsupervised disambiguation aims to achieve word sense 'discrimination' (to discriminate between groups without labelling them). Recall that Gale's algorithm requires us to calculate $P(v|s_k)$, for each word occurrence w to be disambiguated, where v ranges over the elements in the context of w and s_k is a sense of w. This calculation uses empirical evidence taken from a labelled corpus, (that is, a training set). By contrast, since no such training set is available, in Schütze's unsupervised algorithm, $P(v|s_k)$ is calculated as follows: start with a random initialisation of the 'parameters' $P(v|s_k)$ and then re–estimate them using the iterative EM (Expectation–Maximisation) algorithm. Details of the EM algorithm are provided by Manning and Schütze (Manning and Schütze, 1999, pp.518–527). The main advantage of this method is that it does not require any disambiguated material. Another (possible) advantage is that the algorithm tends to split dictionary senses into 'finer–grained' senses. However, the 'cluster groups' identified by unsupervised word sense disambiguation procedures rarely coincide with dictionary senses or human sense assignment. Furthermore, although Schütze's algorithm is good at differentiating between fine–grained polysemous word senses it sometimes fails to spot homonymy. Therefore, Manning and Schütze suggest that unsupervised disambiguation is probably best suited to topic specific texts, such as a Chemistry book (since they hypothesise that, in such sources, polysemy is harder to resolve than homonymy, and homonymy is less frequent than polysemy anyway). A summary of Schütze's algorithm is provided by Charniak (Charniak, 1996a, pp.151–155) and Manning and Schütze (Manning and Schütze, 1999, pp.253–256). A number of other unsupervised algorithms have been suggested by various authors; for example, Zernik (Zernik, 1991), Pereira et al. (Pereira et al., 1993), Dolan (Dolan, 1994), Pederson and Bruce (Pederson and Bruce, 1997) and Chen and Chang (Chen and Chang, 1998).

However, unsupervised disambiguation algorithms are not as successful as we might hope; in general, performance is approximately 5% to 10% lower than that of the best dictionary–based algorithms (Manning and Schütze, 1999, p.256). Indeed, Pederson and Bruce report just 65% correct disambiguation (Pederson and Bruce, 1997) even though, on the corpus used, 73% of the instances could be correctly disambiguated just by (blindly) choosing the most frequent sense of each word! Levow presents a summary of best–known current data–driven word sense disambiguation techniques (Levow, 1997). This concludes our presentation of data–driven word sense disambiguation.

We have seen that current knowledge–driven and (to an even greater extent) data–driven (supervised and unsupervised) word sense disambiguation algorithms are somewhat limited. Other (best–known) word sense disambiguation algorithms include those due to Quillan (Quillan, 1969), Hayes (Hayes, 1978), Hirst (Hirst, 1987), Luk (Luk, 1995) and Yarowsky (Yarowsky, 1995). Unfortunately, other current approaches to word sense disambiguation are (in general) even worse than those which we have discussed. We therefore argue that the problem of lexical ambiguity might profit from the enterprise of underspecification.

Information about state–of–the–art word sense disambiguation algorithms (and how successful they are) is collated and updated as part of the SENSEVAL (the Evaluation of Word Sense Disambiguation Systems) project. Kilgarriff and Palmer (Kilgarriff and Palmer, 2000) and Edmonds (Edmonds, 2002) present brief summaries of this project.[10] Kilgarriff and Palmer claim that the best word sense disambiguation algorithm evaluated by the SENSEVAL project achieves 77% correct disambiguation (compared to human performance of 95% at the same task) and the best unsupervised algorithm tested achieved 63% correct sense discrimination (Kilgarriff and Palmer, 2000). Other useful surveys of word sense disambiguation algorithms have been written by Guthrie (Guthrie et al., 1996), Abney (Abney, 1996), Resnik and Yarowsky (Resnik and Yarowsky, 1997) and Ide and Véronis (Ide and Véronis, 1998).

So far, we have omitted from our discussion any mention of the task of resolving categorical ambiguities. Deciding the grammatical categories of words is known as *part–of–speech tagging* (or just 'tagging'). In the literature, word sense disambiguation and tagging are dealt with separately, 'partly because of the difference between the nature of the problem[s], and partly because of the methods

[10] Further information is also currently available online at `http://www.senseval.org/`.

that have been used to approach them' (Manning and Schütze, 1999, p.231). We now turn our attention to the tagging problem. Having argued that current word sense disambiguation algorithms are limited, we now investigate the success of current tagging techniques.

2.2 Resolving Categorical Ambiguity: Tagging

Many words suffer from categorical ambiguity (that is, at least two of their senses belong to different grammatical categories).[11] In tagging we try to determine which category is most likely for a particular occurrence of a word in a sentence. Therefore, 'tagging is a case of limited syntactic disambiguation' (Manning and Schütze, 1999, p.341). We state the tagging problem as follows:

Problem 2.2.1 (Part–of–Speech Tagging). Given a sentence containing one or more ambiguous words, determine the *most likely* grammatical category for each word.

Obviously, categorical ambiguity is the only reason why tagging is non–trivial, since without it, all we would need to do is map every word to its unique category. It is important to understand why tagging is not the same as 'parsing'. Parsing is the process of reconstructing the derivations (or 'phrase structure trees') that give rise to a particular sequence of words. Tagging, on the other hand, is the process of determining the syntactic categories of the words in a sentence (and not the phrase structure). Tagging is generally considered to be an easier task than word sense disambiguation; however, the most successful tagging algorithms still only achieve about 97% accuracy – even though a simple algorithm, in which we always choose the most frequent categories (determined from tagged corpora[12]) tags 90% of words correctly (Charniak, 1996a, p.49). Therefore, we must evaluate tagging algorithms against a 90% lower bound bench–mark. Clearly, frequency of usage is very useful information in tagging, but what other information is available? One source of information we might use is our knowledge about grammar. For example, consider the sentence 'heavy boats sink'. Suppose that we wish to establish which grammatical category the occurrence of 'sink' should belong to:

[11] DeRose claims that in the Brown Corpus, 4100 (out of 39440) words suffer from categorical ambiguity – about 10% (DeRose, 1988).

[12] We say that a corpus is *tagged* if each word in that corpus is labelled (or 'tagged') with its part–of–speech.

verb (meaning to become submerged) or noun (referring to a plumbing fixture). According to elementary linguistics, and in particular standard grammar, a sentence may be constructed from an adjective followed by a noun (a noun–phrase) followed by a verb (a verb–phrase), but not from an adjective followed by two nouns. That is, grammars (typically) permit us to parse 'heavy boats sink' as follows:

$$[[[\text{heavy}]_{\text{Adj}}\,[\text{boats}]_{\text{N}}\,]_{\text{NP}}\,[\text{sink}]]_{\text{V}}\,]_{\text{S}},$$

but not

$$*\,[[[\text{heavy}]_{\text{Adj}}\,[\text{boats}]_{\text{N}}\,]_{\text{NP}}\,[\text{sink}]]_{\text{N}}\,]_{\text{S}}.$$

We infer that in this sentence, 'sink' must be tagged 'verb'. But how far can such (generic) *syntagmatic* constraints get us? Unfortunately, Greene and Rubin's (syntagmatic) 'rule–based' tagger tags just 77% of words correctly (Greene and Rubin, 1971). It would therefore seem that we need to seek out a more sophisticated solution.

Most tagging algorithms use local contextual information (of each target word). For example, if the word to be tagged is preceded by 'the' then the target word is almost certainly part of a noun–phrase. Suppose that $w_{1,n}$ denotes the sequence of (natural language) words $w_1, \ldots, w_n$. We want to find a sequence of grammatical categories (or 'tags') $t_1, \ldots, t_n$, which we denote $t_{1,n}$, that maximises $P(t_{1,n}|w_{1,n})$. It is not surprising that estimating this probability directly from corpora is in reality (currently) infeasible, due to the size of the computations involved. Fortunately, we can proceed using approximation methods. Before we discuss these methods we re–write the above conditional probability function using Bayes' Rule:

$$P(t_{1,n}|w_{1,n}) = \frac{P(w_{1,n}|t_{1,n})}{P(w_{1,n})} \cdot P(t_{1,n}).$$

Since we want to maximise $P(t_{1,n}|w_{1,n})$ we can eliminate the denominator of the above expression. Therefore, the simplified task is to find a sequence of tags $t_{1,n}$ which maximises

$$P(w_{1,n}|t_{1,n}) \cdot P(t_{1,n}). \tag{2.1}$$

The above probabilities are usually approximated by applying particular independence assumptions. We now discuss (the best–known) current methods of approximating the probabilities in expression 2.1.

A (relatively) well–known way of approximating the required probabilities is to use *n–gram models*, usually *bigram* or *trigram models*. Under the n–gram model, $P(w_{1,n}|t_{1,n})$ is approximated by assuming that the assignment of any particular tag is independent of (the assignment of) all other tags, except the previous $n-1$. In the bigram (or 2–gram) model this means that only the previous two tags matter; that is,

$$P(t_{1,n}) \approx \prod_{i=1}^{n} P(t_i|t_{i-1}),$$

where t_0 is a 'virtual' tag with probability 1. We also assume that a word appears in a category independently of the words in the preceding or succeeding categories, this is approximated by:

$$P(w_{1,n}|t_{1,n}) \approx \prod_{i=1}^{n} P(w_i|t_i).$$

Therefore, 2.1 can be approximated as:

$$P(w_{1,n}|t_{1,n}).P(t_{1,n}) \approx \prod_{i=1}^{n} P(w_i|t_i).P(t_i|t_{i-1}). \tag{2.2}$$

That is, under the bigram model we must find the sequence of tags $t_1, \ldots, t_n$ which maximises 2.2. We cater for the first word of any sentence by creating a 'virtual' tag (or 'pseudo–tag') which we denote '`first`' to take the value of t_0. But how do we compute $P(w_i|t_i)$ and $P(t_i|t_{i-1})$? Let t and t' be (distinct) elements of the tag set. Then,

$$P(t'|t) = \frac{C(t,t')}{C(t)}$$

where $C(t,t')$ is the number of times the tag t is followed by the tag t' in the (tagged) corpus, and $C(t)$ is the number of words tagged by t in the (tagged) corpus. Also, if w is any word in the lexicon then,

$$P(w|t) = \frac{C(w,t)}{C(t)}$$

where $C(w,t)$ is the number of occurrences of w which are assigned tag t in the (tagged) corpus. At best, this approach tags just 95% of word occurrences correctly (Samuelsson and Voutilainen, 1997).

We now take a brief excursion to spell out the relationship between the n–gram models we have been discussing and *Markov models* (Markov, 1913). Both Markov models and n–gram models represent situations which give rise to a sequence of random variables that are not independent, but are such that the value of any variable in the sequence depends on the value of its predecessors. The Markov assumption is that, to calculate the value of the next variable in the sequence we need only to know the value of the current variable (and not all of its predecessors). This assumption is commonly referred to as the 'limited horizon' assumption. Formally, suppose that $X = (X_1, \ldots, X_T)$ is a sequence of random variables taking values in some finite set $S = \{s_1, \ldots, s_n\}$ called the 'state space' such that the so–called 'limited horizon' property holds:

Limited Horizon

$$P(X_{t+1} = s_k | X_1, \ldots, X_t) = P(X_{t+1} = s_k | X_t),$$

then we call X a 'Markov Chain'. Markov chains are used to model the probability of linear sequences of events. For our purposes, we will view the sequence of tags in a text as a Markov Chain. We assume that the tag assigned to each word depends only on the previous tag (and no earlier tags) and that this dependency does not change as we continue to tag other words (so–called 'time invariance'). It should be clear that (n^{th}–order) Markov models are the same as n–grams (the last element of the n–gram being the element that we are predicting). Therefore, in the sequel, we treat the terms 'n–gram Model' and '(n^{th}–order) Markov Model' as being synonymous. End of excursion.

Unfortunately, using approximation 2.2 to tag text is computationally inefficient, since such a scheme requires us to evaluate over all possible taggings. A popular way of refining the 'Markov Model Tagger', which successfully tackles this problem, is to augment the tagger with Viterbi's (iterative) algorithm (Viterbi, 1967). Viterbi's algorithm enables us to efficiently compute two fundamental functions: the first is a function which tells us the probability that tag t is assigned to a word w; the second tells us the most likely tag t_i for word w_i given that (we currently know that) the word w_{i-1} is tagged by t_{i-1}. A detailed discussion about how Viterbi's algorithm increases the efficiency of Markov Model Taggers is provided by Manning and Schütze (Manning and Schütze, 1999, pp.349–351) and Allen (Allen, 1995, pp.202–204).

Unfortunately, even with Viterbi's algorithm, Markov Model Tagging is still not trouble–free. Throughout, we have (implicitly) assumed that all of the words occurring in the sentences we wish to tag also appear in the tagged corpus (that is, in our training set). However, it is quite common for words to appear in the target text but not in the tagged corpus. Clearly, we must find some way of processing these 'unknown words'. Recall that simply selecting the most frequent tag for each word results in 90% accuracy; it should therefore not surprise us that 'the differing accuracy of different taggers over different corpora is often mainly determined by the proportion of unknown words, and the smarts [that is, 'the technical tricks'] built into the tagger to guess the part–of–speech of unknown words' (Manning and Schütze, 1999, p.351). But how should we make these guesses? Well, the most sensible way to proceed is to try to use features of the unknown words (and of their contexts) to make informed guesses as to which (grammatical) categories they should belong to. The information gathered from such features is usually expressed as a set of rules (these rules are sometimes referred to as *context frame rules*). In addition to contextual information, many taggers use morphological information; such as, if a word ends with the string 'ing' and is preceded by a verb then it too must be a verb. Some taggers go further still, using aspects of punctuation and capitalisation to choose the tags of unknown words. In some natural languages this information can be quite profitable; for example, in German, capitalisation is very informative, in particular about how we should tag unknown nouns. Weischedel et al. (Weischedel et al., 1993) implement a tagger which combines elements of these ideas; their algorithm better than halves the error rate for unknown words (from about 40% to about 20%). However, even with refinements, 'the better models typically perform at about the 95% level of correctness: that is, about one word in twenty is given the wrong part–of–speech' (Charniak, 1996a, p.49).

Another approach to tagging is Brill's transformation–based learning algorithm (Brill, 1995). In common with Markov Tagging, Brill's technique needs a tagged corpus as training set. The key idea is to re–tag the entire corpus by assigning each word its most frequent tag (introducing about a 10% error rate). A training model is then applied to our re–tagged version of the corpus until it achieves a tagging which is close to correct – that is, until our re–tagging is almost undone. The transformations that the training model used to perform this can then be applied to a new text (once we have re–tagged it by mapping each word

to its most frequent tag). Brill reports about 97% accuracy using this method (Brill, 1995, p.23).

There are many other tagging algorithms currently in use, including those suggested by the following (non–exhaustive list of) authors: Cherry (Cherry, 1978), Jelinek (Jelinek, 1985), Garside (Garside, 1987), Church (Church, 1988), Benello (Benello et al., 1989), Karlsson (Karlsson, 1990), Kupiec (Kupiec, 1992), Charniak (Charniak et al., 1993), Schmid (Schmid, 1994), Voutilainen (Voutilainen, 1995), Daelemans (Daelemans et al., 1996), Ratnaparkhi (Ratnaparkhi, 1996) and Thorsten (Thorsten, 2000). However, as we have seen, even the most successful tagging algorithms are not completely adequate – tagging at best 97% of words correctly (Manning and Schütze, 1999, p.371). So once again we have seen that state–of–the–art resolution is somewhat limited, supporting our argument that our study of underspecification is worthwhile. This completes our presentation of part–of–speech tagging. We now turn our attention to resolution methods for syntactic ambiguities, so–called *probabilistic parsing* techniques.

2.3 Resolving Syntactic Ambiguity: Probabilistic Parsing and Attachment Resolution

Many sentences have more than one possible phrase structure and therefore more than one parse.[13] Our aim is to use probabilities to predict which of the possible parses of a sentence is most likely, that is, we want to resolve syntactic (or 'structural') ambiguities using probability theory. The possible structures of a sentence are traditionally represented as parse trees. Conventional parsing aims to take a sentence s and compute its parse trees t according to some pre–determined grammar G. Probabilistic (or 'statistical', or even 'stochastic') parsing aims to rank these parse trees by plausibility, or perhaps just to return the most likely parse. That is, probabilistic parsers disambiguate and rate how likely different parses are as they parse; by contrast, conventional parsers simply list all possible parses and perhaps choose between them using semantics or world knowledge afterwards. We define the probabilistic parsing task as follows:

Problem 2.3.1 (Probabilistic Parsing). Given a sentence with more than one possible parse, determine the *most likely* parse for that sentence.

[13] See Church and Patil (1982) and Martin et al. (1987) for an indication of just how bad the situation is!

The simplest approach to probabilistic parsing is to count the number of times each Context–Free Grammar (CFG) rule is used in a (parsed) corpus and then use this information to estimate the probability that the rule will be used in the target text. That is, we estimate the probabilities of rules based on their relative frequency in the (parsed) corpus. Algorithms which exploit this idea are called *Probabilistic Context–Free Grammars* (PCFGs).

2.3.1 Probabilistic Context–Free Grammars (PCFGs)

A Probabilistic Context–Free Grammar is simply a Context–Free Grammar augmented with empirically estimated probabilities attached to each rule; that is, a PCFG is a 'CFG with probabilities added to rules, indicating how likely different rewritings are' (Manning and Schütze, 1999, p.382).[14]

Consider the (fairly typical) CFG rule `VP` $\rightarrow$ `V NP`. Suppose that, in our training corpus exactly 300 constituents are assigned the 'category' `VP` and that exactly 118 of these 'use' the rule `VP` $\rightarrow$ `V NP`. Then, under the PCFG model, we can estimate the probability that this rule will be used in the target text as follows:

$$P(\texttt{VP} \rightarrow \texttt{V NP} | \texttt{VP}) = \frac{118}{300} \approx 0 \cdot 393.$$

But how do these probabilities enable us to select 'most likely' parses? To answer this question we paraphrase an example from Manning and Schütze (Manning and Schütze, 1999, pp.384–385).

Consider the simple PCFG in table 2.1 which we denote G.[15] Let s denote the (syntactically ambiguous) sentence 'astronomers saw stars with ears'. Under grammar G, sentence s has two possible parses which we depict by trees t_1 and t_2 in figures 2.1 and figure 2.2 respectively. To find the probability of a tree in a PCFG model we simply multiply the probabilities of the rules that built its local subtrees; that is, the probability of each parse is the product of the probabilities of all of the rules used in the parse tree. Therefore,

[14] Probabilistic Context–Free Grammars were introduced by Booth and Thomson (Booth and Thomson, 1973).

[15] This table is copied from Manning and Schütze (Manning and Schütze, 1999, p.384).

S → NP VP	1.0	NP → NP PP	0.4
PP → P NP	1.0	NP → astronomer	0.1
VP → V NP	0.7	NP → ears	0.18
VP → VP PP	0.3	NP → saw	0.04
P → with	1.0	NP → stars	0.18
V → saw	1.0	NP → telescopes	0.1

Table 2.1: A simple Probabilistic Context–Free Grammar (PCFG).

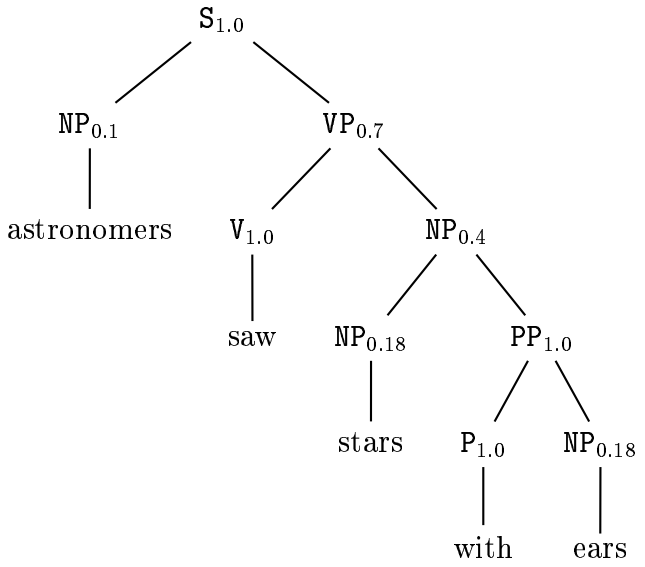


Figure 2.1: One of two possible parses of the structurally ambiguous sentence 'astronomers saw stars with ears'. We denote this tree t_1. Note that non–terminal nodes are subscripted with the probability of the local tree they dominate.

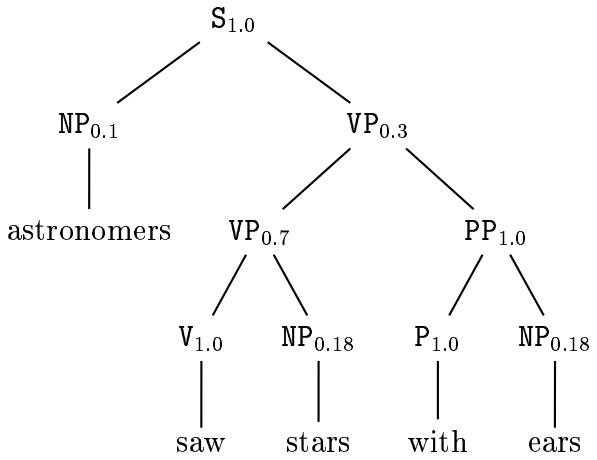


Figure 2.2: One of two possible parses of the structurally ambiguous sentence 'astronomers saw stars with ears'. We denote this tree t_2. Note that, again non–terminal nodes are subscripted with the probability of the local tree they dominate.

$$\begin{aligned} P(t_1|s', G) &\approx 1.0 \times 0.1 \times 0.7 \times 1.0 \times 0.4 \times 0.18 \times 1.0 \times 1.0 \times 0.18 \\ &= 0.0009072, \end{aligned}$$

$$\begin{aligned} P(t_2|s', G) &\approx 1.0 \times 0.1 \times 0.3 \times 0.7 \times 1.0 \times 0.18 \times 1.0 \times 1.0 \times 0.18 \\ &= 0.0006804. \end{aligned}$$

But how can we justify multiplying probabilities in this way to estimate likelihood? According to Charniak, we can derive the product rule for tree probabilities given certain assumptions about subtree independence (Charniak, 1996a, p.76). These assumptions are: the probability of a subtree does not depend on where in the string the words it dominates are (place invariance assumption); the probability of a subtree does not depend on words not dominated by the subtree (context free assumption); and, the probability of a subtree does not depend on nodes in the derivation outside the subtree (ancestor free assumption).[16] We deduce that the most likely parse for *astronomers saw stars with ears* is t_1, since $P(t_1|s', G) > P(t_2|s', G)$. We note in passing that $\sum_t P(t|s', G)$ is the probability that (according to grammar G) sentence s' occurs in the target text (Manning and Schütze, 1999, p.383). Charniak (Charniak, 1996b) uses the relative frequency of local trees in the *Penn Treebank* to assign probabilities to the rules of

[16]These assumptions are listed by Manning and Schütze (Manning and Schütze, 1999, p.384) and a derivation of the multiplication of probabilities rule from these assumptions is provided by Charniak (Charniak, 1996a, pp.78–79).

his PCFG. A *treebank* is exactly what its name suggests it is: a bank (or collection) of (parse) trees. The best–known treebank is the Penn Treebank which consists of (parsed) sentences taken from the *Wall Street Journal*. As we might expect Charniak's technique is somewhat limited; in fact Charniak reports just 88% accuracy (Charniak, 1996b).

One advantage of PCFGs is that they can be used to rule out unnatural computer–generated parses. For example, the sentence 'list the sales of the products produced in 1973 with the products produced in 1972' is reported to have 455 (computer–generated) parses (Martin et al., 1987); however, it seems highly unlikely that any human would generate all of these parses. It is hoped that PCFGs might help to eliminate those parses that are syntactically possible but highly unlikely. An obvious disadvantage of PCFGs is that they ignore all lexical and contextual information and instead use only structural information. It is not surprising therefore that 'these techniques [only] identify the correct parse about 50 percent of the time' (Allen, 1995, p.212). To improve this situation, some authors have tried combining PCGFs with various other theories. For example, Magerman and Mitchell (Magerman and Mitchell, 1991) combine a PCFG with Church's trigram part–of–speech tagger. Magerman and Mitchell report that their ('PEARL') algorithm parses 88% of a 40 sentence text segment correctly. This completes our presentation of PCFGs. Manning and Schütze (Manning and Schütze, 1999, ch.11, pp.381–404) and Charniak (Charniak, 1996a, pp.75–101) present comprehensive introductions to PCFGs.

Other, more general, parsing techniques are split into two categories: *lexicalized* and *non–lexicalized*. A probabilistic parser is said to be lexicalized if it exploits knowledge of lexical dependencies (between words); by contrast, non–lexicalized parsers ignore lexical information. The best–known non–lexicalized parsers are: Probabilistic Context–Free Grammars and Data–Oriented Parsing (Bod, 1996), (Bod, 1998), (Bod et al., 2003), (Sima'an et al., 1994). Unfortunately, even the best non–lexicalized probabilistic parsers are less than 90% accurate. The best–known lexicalized parsers are History–Based Grammars (Black et al., 1993), the `SPATTER` algorithm (Magerman ,1995), Dependency–Based parsing (Collins, 1996) and Attribute–Value Grammars (Abney, 1997). Unfortunately, it is reported that these algorithms only parse between 75% (HBG) and 88% (DBP) of sentences correctly.

2.3.2 Attachment Ambiguity

As we said in the introduction; many sentences in which a prepositional phrase is preceded by an object noun–phrase suffer from attachment ambiguity; for example, 'I saw a man with a telescope'. Fortunately, 'in most cases simple statistics can determine which attachment is the correct one' (Manning and Schütze, 1999, p.279). But where do we obtain these 'simple statistics' from? Our first port–of–call is, as it has been so often, the corpus: we simply count co–occurrences of the verb (*saw* in our example) and the preposition (*with*) and compare this with the number of co–occurrences of the noun (*telescope*) and the preposition (*with*). This idea is explained by Hindle and Rooth:

> Our proposal is to use co–occurrence of verbs and nouns with prepositions in a large body of text as an indicator of lexical preference. Thus, for example, the preposition *to* occurs frequently in the context *send NP* –, that is after the object of the verb *send*. This is evidence of a lexical association of the verb *send* with *to*. Similarly, *from* occurs frequently in the context *withdrawal* –, and this is evidence of a lexical association of the noun *withdrawal* with the preposition *from*. (Hindle and Rooth, 1993, p.3)

In general the goal is to develop a 'procedure to guess whether a preposition is attached to the verb or its object when a verb and its object are followed by a preposition' (Hindle and Rooth, 1993, p.10). Hindle and Rooth make the (simplifying) assumption that, in every case of attachment ambiguity, the preposition attaches either to the verb or the noun (ignoring the fact that the preposition may be licensed by neither – in sentences such as 'The supreme court today agreed to consider reinstating the murder conviction of a New York City man who confessed to killing his former girlfriend after police illegally arrested him at home').[17] For any preposition p, Hindle and Rooth choose to either attach p to the verb v or to the noun n using a likelihood ratio λ as follows:

$$\lambda(v, n, p) = \log_2 \left[\frac{P(\text{Attach}(p) = v | v, n)}{P(\text{Attach}(p) = n | v, n)} \right]. \tag{2.3}$$

We choose verb attachment for large positive values of λ and noun attachment for large negative values. The probabilities in 2.3 are estimated by looking at

[17] This sentence is used by Hindle and Rooth (Hindle and Rooth, 1993, p.8).

relative frequencies (of noun and verb attachment) in corpora.

Unfortunately, Hindle and Rooth report a 20% error rate for their so–called 'Lexical Association' algorithm (compared to a 12% to 15% human error rate for the same task). Further, when tested on a set of 800 sentences (taken from a 13 million word sample of Associated Press stories from 1989), the Lexical Association procedure recognised about 85% of the 586 actual noun attachment examples as noun attachments, and just 70% of the actual verb attachments as verb attachments (Hindle and Rooth, 1993, p.18). In this section, we have seen that current resolution techniques for syntactic ambiguity are limited. In the following section we will assess current resolution methods for operator scope ambiguity.

2.4 Resolving Scope Ambiguity: Scopal Preferences

It is widely accepted that 'scopal ambiguities are problematic for language processing systems; resolving them might lead to a combinatorial explosion' (Gambäck and Bos, 1998, p.433). Consider for example, the sentence 'everyone didn't go to the movie'. We can either interpret the quantifier *everyone* as having scope over the negative *not* (meaning that not one person went to the movie), or we can interpret the negation as having scope over the quantifier (meaning that at least one person didn't go to the movie).[18] The task of resolving scope ambiguity is simply:

Problem 2.4.1 (Resolution of Scope Ambiguity). Given a sentence with more than one operator (for example, a quantifier and a negation or two quantifiers) determine the intended 'scoping relation' between these operators.

It is hypothesised that humans (subconsciously) use some sort of 'rules–of–thumb' when processing scope ambiguous sentences; these rules–of–thumb are usually referred to as *scopal preferences*. In this section we present a (very brief) appraisal of scopal preferences. The simplest example of a scopal preference is the so–called *left–to–right principle* (or sometimes 'linear order principle'): "In simple sentences the leftmost quantified phrase generally takes wide scope". Consider

[18] This example is paraphrased from Manning and Schütze (Manning and Schütze, 1999, p.111).

for example the scope ambiguous sentence 'every kid climbed a tree'. According to the left–to–right principle 'every kid' should (take wide) scope over 'a tree' – suggesting that we should 'prefer' the reading that corresponds to the formula $\forall x(\text{kid}(x) \rightarrow \exists y(\text{tree}(y) \wedge \text{climed}(x, y)))$, which of course we do. It has been suggested that the leftmost quantified phrase 'every kid' takes wide scope over 'a tree' because it is processed first (Lakoff, 1971). However, this principle proves to be too simplistic to resolve an acceptable proportion of scopal ambiguities. For example, the sentence 'I saw a picture of each child' has the preferred reading in which 'each child' takes wide scope, and not narrow scope as the left–to–right principle would predict. Indeed, it is widely accepted that 'order has little to do with the determination of quantifier scope' (Ioup, 1975, p.37). Ioup justifies this claim by taking lots of example sentences from fourteen different natural languages. As an alternative, Ioup proposes two hierarchies to account for scopal preferences across languages. The first hierarchy suggested is: "generally quantifiers whose determiner is 'each' or 'the' (take wide) scope over indefinites which in turn scope over plural quantifiers (such as 'all', 'most' etc.)". However, van Lehn provides empirical evidence (based on human preferences) that this hierarchy is not quite correct and that usually 'each' outscopes 'every' which outscopes 'all' which in turn outscopes plural 'the' (van Lehn, 1978). The second hierarchy suggested by Ioup is: "generally, noun–phrases in subject position should (take wide) scope over noun–phrases in the indirect object position which in turn scope over noun–phrases in object position". In an experiment (on 40 human subjects) Kurtzman and MacDonald found that this preference coincided with human judgement in just 85% of the sentences tested (Kurtzman and MacDonald, 1993).

Another suggestion links the C–command property (from government and binding theory) and scopal preference (Jackendoff, 1972 *and* Reinhart, 1983). Specifically, it is proposed that "a quantified expression takes scope over another quantified expression if the latter is in the C–command domain of the former at s–structure". This principle is sometimes called the *C–command principle* of scopal preference (Kurtzman and MacDonald, 1993). Another scopal preference, due to May, is the so–called *scope constraint* which states that "a quantifier cannot take scope outside of the clause in which it appears" (May, 1977). Unfortunately (and perhaps unsurprisingly), it has been shown, mostly by Kurtzman and MacDonald, that all of these principles are subject to a significant number of exceptions.

Gambäck and Bos (Gambäck and Bos, 1998) present an approach to scopal

resolution in terms of *Underspecified Representation Structures*, (in the sequel we will study these structures in detail). Gambäck and Bos report that their algorithm resolves about 80% of scope ambiguous expressions correctly. Alshawi et al. (Alshawi et al., 1992) use a variety of scopal resolution techniques – including a score which measures how far quantifiers are raised through their QLF structures. Full details of this approach to scope ambiguity (with respect to the Core Language Engine) are presented by Moran (Moran, 1988). Unfortunately, the best (current) techniques for resolving scope ambiguity are less than 90% accurate. Therefore, once more we argue that resolution is limited. This completes our discussion on scopal resolution. We conclude this chapter with a brief assessment of resolution techniques for ambiguities arising as a result of anaphora.

2.5 Resolving Ambiguity arising from Anaphora

Consider the sentence 'I saw a cat chase a mouse; it was black'. Because we are uncertain about what the anaphor 'it' refers to, we cannot be sure whether it is the cat or the mouse that is black.[19] We focus our discussion on Ge's (Ge et al., 1998) approach to this problem, since it is fairly representative of other current techniques. Given an anaphor, a probability is assigned to each of its potential antecedents (or 'referents'); for each potential antecedent its probability reflects the likelihood that it is the correct candidate. We are interested in how these probabilities are estimated. One available source of information is the distance (measured in words) between the pronoun and the candidate antecedent: "the greater the distance the lower the probability". In addition, the actual words appearing in the noun–phrase of the candidate antecedent can provide clues about gender, number and animaticity of the candidate antecedent. Ge's algorithm also exploits empirical observations such as: "noun–phrases that are mentioned repeatedly are preferred". Ge et al. train their model to be responsive to these sources of information on a corpus in which all referential ambiguities are resolved. Ge's algorithm has two separate modules: 'one collects the statistics on the training corpus and the other uses these probabilities to resolve pronouns in the corpus' (Ge et al., 1998, p.5). Unfortunately, Ge reports just 84% accurate pronoun resolution.

[19] Whether or not anybody would utter such an obviously ambiguous sentence in the first place is questionable – see Grice's Maxim of Manner (Grice, 1975); in particular, 'avoid ambiguity'!

An example of a non–statistical approach to anaphora resolution is presented by Mitamura et al. (Mitamura et al., 2002). Their algorithm has two steps:

> The first step is to identify possible antecedents by applying a set of pre–defined constraints. The second step is to eliminate candidates by applying a set of selection rules (heuristics) in a particular order. (Mitamura et al., 2002, p.3)

To select the correct antecedent, a set of ten heuristic rules are applied, these include:

1. Prefer an antecedent that is also an anaphor.
2. If two antecedents occur in the form: ⟨np1⟩ of ⟨np2⟩, prefer ⟨np1⟩. But if ⟨np1⟩ is one of 'type', 'length', 'size', or 'part' then prefer ⟨np2⟩.
3. Prefer antecedents that attach to the same syntactic constituent (or part–of–speech or grammatical function) as the pronoun.
4. Prefer antecedents that are conjunctions.
5. Prefer the most recent antecedent.

(Mitamura et al., 2002, pp.4–5)

Central to this algorithm is that these heuristics are ranked in order of significance (the most significant being first); that is, 'it is important that not every heuristic is tried for each anaphor, and sequential ordering is used to rank the heuristics' (Mitamura et al., 2002, p.4). This is in contrast to other methods which apply every heuristic to every anaphor and use a weighted sum technique to make a final selection.

Mitamura et al. report that, when implemented into the KANTOO system, their algorithm achieves about 90% correct resolution (Mitamura et al., 2002, p.1). Other work on the resolution of anaphoric ambiguity includes that of Carbonell and Brown (Carbonell and Brown, 1988), Dagan and Itai (Dagan and Itai, 1990), Kennedy and Boguraev (Kennedy and Boguraev, 1996) and Mitkov (Mitkov, 1999). Ge et al.'s account provides a good introduction to this topic, including an overview of previous work (Ge et al., 1998, sec.7, pp.9–10). This completes our presentation of resolution techniques. Readers interested in the resolution of

ambiguity arising as a result of ellipsis are referred to the work of Kehler (Kehler, 1993).

In this chapter, we have presented a brief appraisal of current ambiguity resolution techniques. We have seen that none of these algorithms is completely adequate, since in each case there is a considerable degree of error (between about 60% and 98%). Perhaps the situation is so bad that Bar–Hillel's belief is justified:

> ... the quality of fully autonomous mechanical translation, even when restricted to scientific or technological material, will never approach that of qualified human translators. (Bar–Hillel, 1964, p.182)

We argue that, since ambiguity resolution is limited, underspecification (a scheme which stands in opposition to resolution) is worth investigating. We begin our investigation of underspecification with a presentation of current Underspecified Representation Languages (sometimes 'URLs').

Chapter 3

Underspecified Representation Languages

Visiting relatives can be boring.

On first considering the logical properties of ambiguous expressions, we might feel that the apparatus of ordinary first–order logic is adequate. More specifically, we might try to capture the ambiguous content of natural language sentences using disjunction. However, we argue that a disjunctive theory of ambiguity is incompatible with compositional theories of meaning. Consider for example, the sentences:

$$\text{Nick is mad,} \tag{3.1}$$

$$\text{Nick is not mad.} \tag{3.2}$$

Suppose that these sentences have two readings each (due to the lexical ambiguity of the word 'mad' – meaning either 'insane' or 'angry'). Under a disjunctive theory of ambiguity we would encode the ambiguous content of sentence 3.1 using the first–order formula 'insane(nick) $\vee$ angry(nick)'. If we assume that negated natural language sentences should be (systematically) interpreted as the negation of the interpretations of their non–negated counterparts, then to obtain a representation of 3.2 we would simply negate our representation of 3.1 resulting in the formula '$\neg$(insane(nick) $\vee$ angry(nick))'. However, we argue that under a disjunctive theory of ambiguity we would intuitively represent the ambiguous content of sentence 3.2 by the first–order formula '$\neg$insane(nick) $\vee$ $\neg$angry(nick)' which

is not logically equivalent to '$\neg$(insane(nick) $\vee$ angry(nick))'. Clearly, such an interpretation fails to do justice to the way ambiguity interacts with logical constructions in English sentences; so that for example, negation cannot be treated systematically. Therefore we may rule out a disjunctive theory of ambiguity.[1] We now describe the current approaches to underspecified representation, focusing on the syntax and formal disambiguation procedure of each language. In the sequel we will use the word 'disambiguation' to refer to a product as well as a process (since we feel that within context, our use of this term is clear). We call any unambiguous disambiguation a 'total disambiguation'; the term 'total disambiguation' is therefore synonymous with 'possible reading'. Throughout this thesis, we denote the language of ordinary first–order logic $\mathcal{L}$.

3.1 A Logical Connective for Ambiguity

The Underspecified Representation Language described by van Eijck and Jaspars (van Eijck and Jaspars, 1996, pp.5–11) is motivated by a consideration of local ambiguities (for example, lexical or referential ambiguities). A binary connective (called *ambiguation*) denoted '?' is introduced to capture the notion of a sentence being ambiguous between two distinct readings; thus, the formula $\varphi_1 \,?\, \varphi_2$ is ambiguous between the readings (or "disambiguations") φ_1 and φ_2. Ambiguity only occurs in this language as a result of the ambiguation connective. Consider for example the sentence 'every bank is mossy', which we claim incorporates lexical ambiguity, since the lexeme 'bank' might mean either 'mud wall' or 'financial institution'. We can represent the ambiguous content of 'every bank is mossy' in the language of van Eijck and Jaspars by the formula,

$$\forall x \big[\big(\mathrm{mud\_wall}(x) \,?\, \mathrm{financial\_inst}(x) \big) \rightarrow \mathrm{mossy}(x) \big].$$

We shall see that this formula encodes precisely the information we intend it to; that is, we shall see that this formula has exactly two (total) disambiguations '$\forall x(\mathrm{mud\_wall}(x) \rightarrow \mathrm{mossy}(x))$' and '$\forall x(\mathrm{financial\_inst}(x) \rightarrow \mathrm{mossy}(x))$', which correspond to the two possible readings of 'every bank is mossy'.

We denote the first–order ambiguous language $\mathcal{Q}$ (for "Question–Mark").

[1] Van Deemter presents a more detailed argument against a disjunctive theory of ambiguity (van Deemter, 1996, pp.205–208).

Definition 3.1.1 (Syntax of $\mathcal{Q}$). The language $\mathcal{Q}$ is the set of formulas generated by the usual formation rules for $\mathcal{L}$ (the language of ordinary first–order logic) together with the rule:

$$\text{if } \pi,\ \pi' \in \mathcal{Q},\ \text{then } (\pi\,?\,\pi') \in \mathcal{Q}.$$

As indicated earlier, we think of $\pi\,?\,\pi'$ as a formula which is ambiguous between π and π'. For any $\varphi \in \mathcal{Q}$, a *disambiguation* of φ is a formula obtained by replacing zero or more occurrences of the ?–operator in φ by either its left or right operand, not necessarily uniformly. We will use the following definition of disambiguation in $\mathcal{Q}$:

Definition 3.1.2 (Disambiguation). Let $\pi \in \mathcal{Q}$. We define the *disambiguations of* π by induction as follows:

> If π is an atom, π is the only disambiguation of π.
>
> If $\pi = \neg\pi_1$ then φ is a disambiguation of π if and only if $\varphi = \neg\varphi_1$, where φ_1 is a disambiguation of π_1.
>
> If $\pi = \pi_1 \wedge \pi_2$ then φ is a disambiguation of π if and only if $\varphi = \varphi_1 \wedge \varphi_2$, where φ_1 is a disambiguation of π_1 and φ_2 is a disambiguation of π_2.
>
> If $\pi = \pi_1 \vee \pi_2$ then φ is a disambiguation of π if and only if $\varphi = \varphi_1 \vee \varphi_2$, where φ_1 is a disambiguation of π_1 and φ_2 is a disambiguation of π_2.
>
> If $\pi = \pi_1 \rightarrow \pi_2$ then φ is a disambiguation of π if and only if $\varphi = \varphi_1 \rightarrow \varphi_2$, where φ_1 is a disambiguation of π_1 and φ_2 is a disambiguation of π_2.
>
> If $\pi = \pi_1\,?\,\pi_2$ then φ is a disambiguation of π if and only if φ is a disambiguation of π_1 or a disambiguation of π_2. The formula $\pi_1\,?\,\pi_2$ is also a disambiguation of π.
>
> If $\pi = \forall x\pi_1$ then φ is a disambiguation of π if and only if $\varphi = \forall x\varphi_1$, where φ_1 is a disambiguation of π_1.

A formula with no occurrences of '?' is called *unambiguous*. A *total disambiguation* of φ will be any unambiguous disambiguation of φ. For example, the formula $\varphi_1 \rightarrow (\varphi_2\,?\,(\varphi_3 \wedge \varphi_4))$ has total disambiguations $\varphi_1 \rightarrow \varphi_2$ and $\varphi_1 \rightarrow (\varphi_3 \wedge \varphi_4)$. It is important to realise that, by definition, for any $\varphi \in \mathcal{Q}$, all the total disambiguations of φ are well–formed formulas of ordinary first–order

logic ($\mathcal{L}$) and that all formulas of $\mathcal{Q}$ have at least one disambiguation. Furthermore, it is trivial to write a linear time algorithm which can compute a single (total) disambiguation of any $\varphi \in \mathcal{Q}$ (by simply deleting the right operand of each occurrence of the ambiguation connective). We also note that the disambiguations of $\varphi_1 \,?\, (\varphi_2 \,?\, \varphi_3)$ are exactly those of $(\varphi_1 \,?\, \varphi_2) \,?\, \varphi_3$; that is, it is clear (inductively) that the ambiguation connective is associative. Therefore, although '?' is a binary connective, it makes perfect sense to write formulas of the form $\varphi_1 \,?\, \ldots \,?\varphi_n$ for any $n > 1$. The main strength of this language is its mathematical simplicity. However, this simplicity is not without cost; we shall see that $\mathcal{Q}$ is not the most intuitive language for dealing with operator scope ambiguities. This completes our presentation of the language $\mathcal{Q}$.

3.2 Holes and Constraints

In this section we present four reasonably well–known (in the sub–culture of Computational Semantics) Underspecified Representation Languages: Predicate Logic Unplugged (Bos, 1995), Minimal Recursion Semantics (Copestake et al., 1995), the Constraint Language for Lambda Structures (Egg et al., 1998) and Underspecified Discourse Representation Theory (Reyle, 1993). Henceforth we use the acronyms PLU, MRS, CLLS and UDRT to refer to these languages. We group PLU, MRS, CLLS and UDRT together to mirror our belief that they are the same, excepting some cosmetic differences. We begin by presenting a language which we base on Bos' PLU. In doing so, we aim to capture the key features of PLU, MRS, CLLS and UDRT. We call this language $\mathcal{H}$ (for "Hole Semantics"); however, it is important to stress that we will be rather liberal (for the sake of clarity and generality) in our interpretation of Bos' Hole Semantics (Bos, 1995). In the sequel, we will specify exactly how each of PLU, MRS, CLLS and UDRT differ from $\mathcal{H}$ and why we believe that these differences need not concern us.

3.2.1 The Language $\mathcal{H}$

Consider the declarative sentence 'every man loves a woman'. We may translate this sentence into ordinary predicate logic in either of two ways (because it is

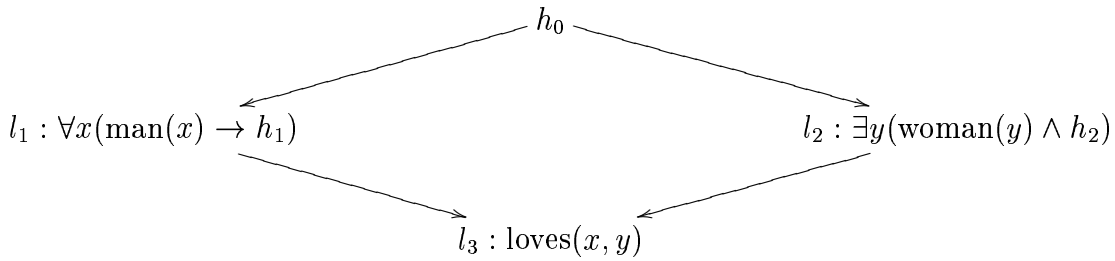
scope ambiguous), namely

$$\forall x\big(\text{man}(x) \rightarrow \exists y(\text{woman}(y) \wedge \text{loves}(x,y))\big) \text{ or}$$
$$\exists y\big(\text{woman}(y) \wedge \forall x(\text{man}(x) \rightarrow \text{loves}(x,y))\big).$$

Using the language $\mathcal{Q}$, we could 'glue' these two readings together with the ambiguation connective '?'. However, doing so would mean disambiguating the original sentence and then recreating the ambiguity from these disambiguations. This strategy is viewed as unnatural by van Eijck and Jaspars (van Eijck and Jaspars, 1996, p.11) and is certainly inefficient from a computational perspective, since in general, the size of such a formula will be exponential in the number of readings of the original sentence. Van Eijck and Jaspars express a desire to 'represent scope underspecification by means of a notation that allows unscoped operators as ingredients of formulas' (van Eijck and Jaspars, 1996, p.11), $\mathcal{H}$ is such a language.

A clear (and efficient) informal description of Hole Semantics must involve the use of pictures and examples. We therefore begin with a diagram representing the formula in $\mathcal{H}$ which captures the ambiguous content of the (scope ambiguous) sentence 'every man loves a woman'.

Example 3.2.1. We would like to write an expression of $\mathcal{H}$ with the disambiguations $\forall x(\text{man}(x) \rightarrow \exists y(\text{woman}(y) \wedge \text{loves}(x,y)))$ and $\exists y(\text{woman}(y) \wedge \forall x(\text{man}(x) \rightarrow \text{loves}(x,y)))$. We claim that exactly these disambiguations are encoded by the following $\mathcal{H}$ expression, which we (informally) depict using a diagram:



The formulas occurring at the nodes of this diagram are taken from the syntax of ordinary first–order logic, except that any subformula can be replaced by a *hole* (the holes in our diagram are represented by the symbols h_0, h_1 and h_2). These formulas are sometimes referred to as *h–formulas*. The key idea is that holes can be filled (or 'plugged') by other formulas (which may themselves contain

holes); that is, we will be able to substitute formulas for holes. Therefore, in a sense, holes may be viewed as meta–variables over formulas. Each formula in the diagram is assigned a unique label (l_1, l_2,...) to avoid the difficulties which would otherwise arise as a result of duplicate copies of formulas. The arrows in our diagram represent constraints. The existence of an arrow between two nodes means that under any plugging, the formula at the head of the arrow must be a subformula of the formula at its tail. To disambiguate an $\mathcal{H}$–expression, we must first find a one–to–one map (or 'plugging') between holes and (the labels of) formulas which 'satisfies' the constraints of that $\mathcal{H}$–expression. We then use this plugging to (recursively) replace holes with formulas, until eventually there are no holes left. The result is a (completely assembled) hole–free formula (an element of $\mathcal{L}$) which we call a 'disambiguation'. Note that, in general, we have no guarantee that we will be able to do this. In a nutshell, our diagram represents a partial order on holes and formulas; disambiguation aims to make this order more specific. That is, disambiguation can be viewed as the process by which such constraint diagrams are 'collapsed' by 'plugging' holes with formulas in agreement with the constraints.

The only pluggings which satisfy the constraints in example 3.2.1 are p, where $p(h_0) = l_1$, $p(h_1) = l_2$ and $p(h_2) = l_3$ and p', where $p'(h_0) = l_2$, $p'(h_1) = l_3$ and $p'(h_2) = l_1$. By inserting formulae into holes according to these pluggings, we obtain $\forall x(\text{man}(x) \rightarrow \exists y(\text{woman}(y) \wedge \text{loves}(x, y)))$ and $\exists y(\text{woman}(y) \wedge \forall x(\text{man}(x) \rightarrow \text{loves}(x, y)))$ respectively. We now make these ideas more concrete, starting with the formal syntax of $\mathcal{H}$.

Let H and L be countably infinite sets, such that H is disjoint from all the formulas in $\mathcal{L}$ and H does not contain h_0. Denote by $\mathcal{L}^H$ the set of formulas obtained by regarding the elements of H as (syntactically speaking) additional proposition letters (0–ary predicate symbols). We will always write $H = \{h_1, h_2, \ldots\}$ and $L = \{l_1, l_2, \ldots\}$.

Definition 3.2.2 (Unconstrained Hole Representation). An *Unconstrained Hole Representation* (or UHR) is a function $\xi : L' \rightarrow \mathcal{L}^H$ where L' is a finite, nonempty subset of L. If ξ is a UHR, we denote the domain L' of ξ by $L(\xi)$ and the set $\{h \in H | h \text{ occurs in } \xi(l) \text{ for some } l \in L(\xi)\}$ by $H(\xi)$.

We can now represent the sentence 'every man loves a woman' more formally:

Example 3.2.3. Let ξ be a UHR with

$$\begin{aligned} \xi(l_1) &= \forall x(\text{man}(x) \to h_1), \\ \xi(l_2) &= \exists y(\text{woman}(y) \wedge h_2), \\ \xi(l_3) &= \text{loves}(x, y). \end{aligned}$$

Then $L(\xi) = \{l_1, l_2, l_3\}$ and $H(\xi) = \{h_1, h_2\}$. For ease of reading we display ξ in the following way:

$$\{l_1 : \forall x(\text{man}(x) \to h_1),\ l_2 : \exists y(\text{woman}(y) \wedge h_2),\ l_3 : \text{loves}(x, y)\}.$$

We refer to the elements of L as *labels* and to the elements of H as *holes*. Intuitively, labels identify fragments of logic and holes identify places in those fragments into which material can be inserted.

Definition 3.2.4 (Constrained Hole Representation). A *Constrained Hole Representation* (CHR) is a pair (ξ, C) where ξ is a UHR and $C \subseteq H(\xi) \times L(\xi)$. Denote by $\mathcal{H}$ the set of CHRs.

Example 3.2.5. Let ξ be as in example 3.2.3. Then $(\xi, \{\langle h_1, l_2\rangle\})$, $(\xi, \{\langle h_2, l_1\rangle\})$ and $(\xi, \emptyset)$ are all CHRs. The first corresponds (as we shall see below) to the unambiguous formula $\forall x(\text{man}(x) \to \exists y(\text{woman}(y) \wedge \text{loves}(x, y)))$; the second corresponds to the unambiguous formula $\exists y(\text{woman}(y) \wedge \forall x(\text{man}(x) \to \text{loves}(x, y)))$; the third is ambiguous between the two. Note that, $(\xi, \emptyset)$ is the same CHR as we depicted pictorially in example 3.2.1 on page 56.

Our next goal is to specify how the elements of $\mathcal{H}$ are to be disambiguated.

Notation 3.2.6. Let h_0 denote a fixed object, called a *root*; recall that $h_0 \notin H$.

Blackburn and Bos explain that 'the hole h_0 will play a special role ... it's used to state constraints, but it *won't* be used in formulas' (Blackburn and Bos, 1999a, p.84). We will see how the hole h_0 is used as a technical device to ensure that, under substitution, our 'top level' formula is completely assembled. If $H' \subseteq H$, denote by H'_0 the set $H' \cup \{h_0\}$. Similarly, if ξ is a UHR, $H_0(\xi) = H(\xi) \cup \{h_0\}$. We refer to h_0, in addition to the elements of H, as a hole.

Definition 3.2.7 (Plugging). Let ξ be a UHR. A *plugging* for ξ is a function $p : H_0(\xi) \to L(\xi)$.

Notation 3.2.8. Given any plugging p for a UHR ξ, we define an associated map $\widehat{p} : \mathcal{L}^{H_0} \to \mathcal{L}^{H}$ by:

$$\begin{array}{lcl}
\widehat{p}(h) & = & \xi(p(h)) \text{ if } h \in H_0(\xi), \\
\widehat{p}(\alpha) & = & \alpha \text{ if } \alpha \text{ is an atom but } \alpha \notin H(\xi), \\
\widehat{p}(\neg\varphi) & = & \neg\widehat{p}(\varphi), \\
\widehat{p}(\varphi \wedge \psi) & = & \widehat{p}(\varphi) \wedge \widehat{p}(\psi), \\
\widehat{p}(\varphi \vee \psi) & = & \widehat{p}(\varphi) \vee \widehat{p}(\psi), \\
\widehat{p}(\varphi \to \psi) & = & \widehat{p}(\varphi) \to \widehat{p}(\psi), \\
\widehat{p}(\forall x\varphi) & = & \forall x\widehat{p}(\varphi).
\end{array}$$

We denote the result of n iterations of a function f by $f^{(n)}$. If, for some n, $\widehat{p}^{(n)}(\psi) = \widehat{p}^{(n+1)}(\psi)$, we write $p^*(\psi) = \widehat{p}^{(n)}(\psi)$; otherwise $p^*(\psi)$ is undefined.

Definition 3.2.9 (Descendant). Let (ξ, C) be a CHR and p be a plugging for the UHR ξ. Given $h, h' \in H(\xi)$, we say that h' is a *child* of h (relative to ξ and p) if h' occurs in $\widehat{p}(h) = \xi(p(h))$. Likewise, we say that h' is a *descendant* of h if there is a sequence $h' = h_0, \ldots, h_m = h$ $(m \geqslant 1)$ such that h_i is a child of h_{i+1} for all i $(0 \leqslant i < m)$.

We will use our definition of descendant to characterise exactly those pluggings which describe well–formed formulas of $\mathcal{L}$. Obviously, any such plugging must fill all holes with labels. In addition, we must avoid infinite loops in the plugging process; that is, we will need to rule out those pluggings which are cyclic, otherwise the plugging process will never terminate. The following definition makes these ideas more formal:

Definition 3.2.10 (Admissible Plugging). The plugging p is said to be *admissible* for (ξ, C) if the following conditions hold:

1. $p : H_0(\xi) \to L(\xi)$ is a bijection,
2. No $h \in H(\xi)$ is a descendant of itself,
3. For all $\langle h, l\rangle \in C$, either $p(h) = l$, or $p(h') = l$ for some $h' \in H(\xi)$ such that h' is a descendant of h.

Intuitively, a plugging specifies which fragments of ξ (identified by labels) are to be inserted into which holes in $H_0(\xi)$. As we have said, an admissible plugging

has the property that this insertion process assembles all the fragments of ξ into a single formula of $\mathcal{L}$ which gets inserted into the root h_0. Formally, we have:

Lemma 3.2.11. *Let (ξ, C) be a CHR and p an admissible plugging. Then $p^*(h_0)$ exists and is an element of $\mathcal{L}$. Moreover, $p^*(h_0)$ contains at least one copy of every formula $\xi(l)$ where $l \in L(\xi)$.*

Proof of Lemma 3.2.11. Note that the constraints C play no role in this result. That $p^*(h_0)$ exists and that $p^*(h_0) \in \mathcal{L}$ follow from the fact that $H(\xi)$ is finite and no $h \in H(\xi)$ is a descendant of itself. That is, because $H(\xi)$ is finite and no $h \in H(\xi)$ is a descendant of itself, each of the formulas $\widehat{p}^{(n)}(h_0)$ for fixed $n \in \mathbb{N}$ must have bounded depth, and (since $H(\xi)$ is finite) we know that the sequence of formulas $\widehat{p}^{(n)}(h_0)$ for $n = 1, 2, \ldots$ must converge, therefore $p^*(h_0)$ exists. That every fragment $\xi(l)$ gets included in $p^*(h_0)$ follows from the fact that p is onto and again from the fact that no $h \in H(\xi)$ is a descendant of itself. □

It is important to realise that some CHRs may have no admissible pluggings. This arises if, for example, $|H_0(\xi)| \neq |L(\xi)|$ or if the constraints in C force some hole to be a descendant of itself. However, we make no attempt to ban these representations from $\mathcal{H}$. This leads us to the following definition:

Definition 3.2.12 (Disambiguation). Let $(\xi, C) \in \mathcal{H}$. Then a *disambiguation* of (ξ, C) is any formula $p^*(h_0)$, where p is an admissible plugging for (ξ, C).

From what has just been said, it is clear that some elements of $\mathcal{H}$ have no disambiguations. For example, neither of the CHRs $\xi = \{l_1 : p \vee h_1,\ l_2 : q \wedge h_2\}$ with $C = \emptyset$, or $\xi' = \{l_3 : p \vee h_3,\ l_4 : q \wedge h_4,\ l_5 : r\}$ with $C = \{\langle h_4, l_3\rangle, \langle h_3, l_4\rangle\}$ have any admissible pluggings.

Example 3.2.13. Consider the CHR $(\xi, \emptyset)$ where ξ is defined in example 3.2.3. The pluggings

$$p_1 : \quad \cancel{h_0} \not\trianglerighteq l_1, \quad h_1 \mapsto l_2, \quad \cancel{h_2} \not\trianglerighteq l_3 \quad \text{and}$$
$$p_2 : \quad \cancel{h_0} \not\trianglerighteq l_2, \quad h_1 \mapsto l_3, \quad \cancel{h_2} \not\trianglerighteq l_1.$$

are the only admissible pluggings of $(\xi, \emptyset)$. It is simple to verify that

$$\begin{aligned} p_1^*(h_0) &= \forall x(\mathrm{man}(x) \rightarrow \exists y(\mathrm{woman}(y) \wedge \mathrm{loves}(x, y))) \quad \text{and} \\ p_2^*(h_0) &= \exists y(\mathrm{woman}(y) \wedge \forall x(\mathrm{man}(x) \rightarrow \mathrm{loves}(x, y))) \end{aligned}$$

are therefore exactly the disambiguations of $(\xi, \emptyset)$.

Note that the CHR $(\xi, \{\langle h_1, l_2\rangle\})$ has p_1 as its only admissible plugging, since $p_2(h_1) = l_3 \neq l_2$ and h_1 has no descendants (relative to p_2). Likewise, the CHR $(\xi, \{\langle h_2, l_1\rangle\})$ has p_2 as its only admissible plugging.

We now take a brief excursion to investigate the computational complexity of checking CHRs for the existence of admissible pluggings. We refer to this checking problem (that is, the problem of determining whether a CHR has any disambiguation at all) as the *plugging problem.*[2]

Lemma 3.2.14. *The plugging problem for expressions of $\mathcal{H}$ is NP–complete.*

We will prove the NP–hardness of the plugging problem for $\mathcal{H}$ by showing that the bin packing problem[3] can be encoded in terms of the plugging problem. More precisely, the problem that we will use is actually a special case of bin packing called 'k–partition'; details of which are provided by Papadimitriou (Papadimitriou, 1994, p.216, problem 9·5·32). We repeat this problem below:

Problem 3.2.15. Let $k \geqslant 2$ be a fixed integer. The *k–partition problem* is the following special case of bin packing: we are given $n = km$ integers $a_1, \ldots, a_n$ adding up to mC, such that $\frac{C}{k+1} < a_i < \frac{C}{k-1}$ for all i. That is, the numbers are such that their sum fits exactly in m bins, but no $k+1$ of them fit into one bin, neither can any $k-1$ of them fill a bin. The question is then: can we find a partition of these numbers into m groups of k, such that the sum in each group is precisely C?

For $k = 3, 4, \ldots$, the k–partition problem is NP–complete. In fact, the bin packing problem and the k–partition problem are *strongly* NP–complete. An NP–complete problem is strongly NP–complete if, after restricting any instance of length n to contain integers of size at most $p(n)$ (where p is polynomial), it is still NP–complete. A strongly NP–complete problem remains NP–complete even if we choose to represent its instances using inefficient notation such as unary notation – since the use of such inefficient notation only increases the size of the reduction by a polynomial amount. From a detailed consideration of the NP–hardness proof of the bin packing problem (Papadimitriou, 1994, pp.204–206, Thm.9·11) we may take the size of a problem instance of the k–partition problem to be given by $\sum_{i=1}^{n} a_i$. In other words, we do not assume binary encoding. (This will be important for our purposes). We are now ready to prove lemma 3.2.14.

[2]The plugging problem is sometimes referred to as the *satisfiability problem* – for example, by Koller et al. (Koller et al., 2000).

[3]Papadimitriou discusses bin packing in detail (Papadimitriou, 1994, pp.204–206).

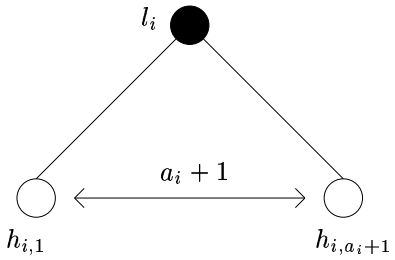


Figure 3.1: The encoding of the i$^{\text{th}}$ item of $\langle a_1, \ldots, a_n, m, C\rangle$.

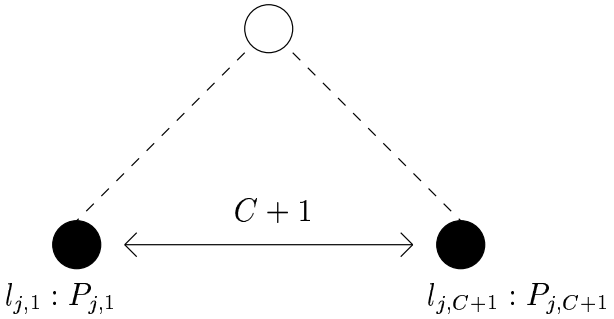


Figure 3.2: The encoding of the j$^{\text{th}}$ bin of $\langle a_1, \ldots, a_n, m, C\rangle$.

Proof of Lemma 3.2.14. We will encode the 4–partition problem in terms of the plugging problem for $\mathcal{H}$. Let $\langle a_1, \ldots, a_n, m, C\rangle$ be the 4–partition problem instance we wish to encode. We must construct a formula $r(\langle a_1, \ldots, a_n, m, C\rangle) \in \mathcal{H}$ which has a disambiguation if and only if $\langle a_1, \ldots, a_n, m, C\rangle$ has answer 'yes'. We encode the i$^{\text{th}}$ item a_i (for $1 \leqslant i \leqslant n$) of $\langle a_1, \ldots, a_n, m, C\rangle$ by the labelled formula $l_i : h_{i,1} \wedge \ldots \wedge h_{i,a+1}$ which we represent pictorially in figure 3.1. We encode the j$^{\text{th}}$ bin of $\langle a_1, \ldots a_n, m, C\rangle$, which has capacity C for $1 \leqslant j \leqslant m$, by the labelled formulas $l_{j,1} : P_{j,1}, \ldots, l_{j,C+1} : P_{j,C+1}$ together with the set of constraints $\{\langle h_j, l_{j,k}\rangle \,|\, 1 \leqslant k \leqslant C+1\}$. Each $P_{j,k}$ (for $1 \leqslant k \leqslant C+1$) is just a proposition letter. We depict our encoding of the j$^{\text{th}}$ bin in figure 3.2. Putting these fragments together, we construct a CHR which has the form illustrated in figure 3.3. This reduction is formally expressed as follows:

$$\begin{aligned} r(\langle a_1, \ldots, a_n, m, C\rangle) \;:=\; & \langle \{l_i : h_{i,1} \wedge \ldots h_{i,a_{i+1}} \,|\, 1 \leqslant i \leqslant n\} \cup \\ & \{l_0 : h_{0,1} \wedge \ldots \wedge h_{0,m}\} \cup \\ & \{l_{j,k} : P_{j,k} \,|\, 1 \leqslant j \leqslant m\,;\, 1 \leqslant k \leqslant C+1\}, \\ & \{\langle l_{j,k}, h_{0,j}\rangle \,|\, 1 \leqslant j \leqslant m\,;\, 1 \leqslant k \leqslant C+1\}\rangle. \end{aligned}$$

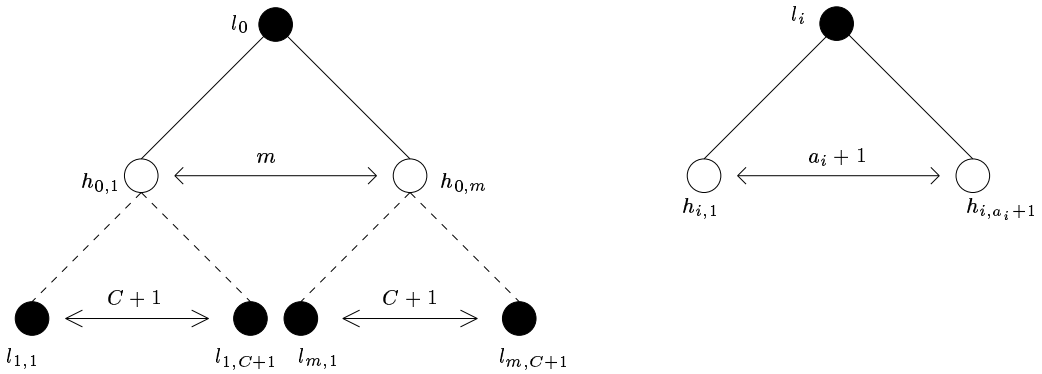


Figure 3.3: The encoding of $\langle a_1, \ldots, a_n, m, C\rangle$. (Note that there should be n copies of the right–hand fragment; one for each $1 \leqslant i \leqslant n$).

Given a plugging p, we can define an assignment of items to bins: assign item a_i to bin j if and only if l_i is a descendant of h_j according to the plugging p. We claim that p is an admissible plugging only if our assignment is a solution to the 4–partition problem instance $\langle a_1, \ldots, a_n, m, C\rangle$. Suppose that p is an admissible plugging of $r(\langle a_1, \ldots, a_n, m, C\rangle)$ and suppose that h_j (for some $1 \leqslant j \leqslant m$) has descendants $l_{i_1}, \ldots, l_{i_k}$ for some $i_1, \ldots, i_k \in [1, n]$. Then the number of holes created under h_j by $\xi(l_{i_1}), \ldots, \xi(l_{i_k})$ (that is, by the formulas labelled by $l_{i_1}, \ldots, l_{i_k}$ respectively) is precisely $a_{i_1} + \cdots + a_{i_k} + 1$. But since these holes are plugged by $l_{1,1}, \ldots, l_{1,C+1}$ we have $a_{i_1} + \cdots + a_{i_k} + 1 = C + 1$; that is, $a_{i_1} + \cdots + a_{i_k} = C$. Hence assigning $a_{i_1}, \ldots a_{i_k}$ to bin j fills it exactly. The proof of the converse is similar; that is, we manufacture an admissible plugging of $r(\langle a_1, \ldots, a_n, m, C\rangle)$ using a successful assignment (of items to bins) for $\langle a_1, \ldots, a_n, m, C\rangle$ by plugging $\xi(l_i)$ into any hole dominated by $h_{0,j}$ if a_i is assigned to bin C_j. We know that this reduction is polynomial because the size of the problem instance of 4–partition is given by $\sum_{i=1}^{n} a_i$. □

We note in passing that if we choose to restrict the syntax of $\mathcal{H}$ so that each $\mathcal{H}$–formula may contain at most one hole, then the plugging problem becomes much simpler. In fact, under such circumstances, the plugging problem amounts to looking for a linear order (subject to a 'no–cycle' constraint) and it is therefore in P. This completes our presentation of the Underspecified Representation Language $\mathcal{H}$. We now discuss the differences between our version of Hole Semantics $\mathcal{H}$ and Bos' (1995) PLU.

3.2.2 Predicate Logic Unplugged

Our version of Hole Semantics is very similar to Bos' PLU (Bos, 1995). Indeed, we view $\mathcal{H}$ and PLU merely as notational variants of one another, accordingly, we will not make a full presentation of PLU here.[4] Instead, we will specify how the syntax of PLU differs from that of $\mathcal{H}$. Readers uninterested in the details of Bos' PLU may skip to the next section. There are two basic differences between the syntax of PLU and $\mathcal{H}$:

1. As we have seen, CHRs in $\mathcal{H}$ are pairs (ξ, C). By contrast, Underspecified Representations in PLU are triples $\langle H, L, C\rangle$, where the first element H is a set of holes; the second element L is a set of labelled formulas (these formulas may take the elements of H as subformulas – Bos calls the elements of L *h–formulas*); and the third element C is a set of pairs taken from $(H \cup L)^2$. In fact, the set C (called the set of 'constraints') takes the form: $C = \{k_1 \leqslant k_2 \mid k_1, k_2 \in H \cup L\}$. By contrast, the constraints in the CHRs of $\mathcal{H}$ are all of the form (h, l) with $h \in H(\xi)$ and $l \in L(\xi)$. Therefore, PLU seems to allow a freer use of constraints than $\mathcal{H}$; that is, the elements of a PLU constraint can be either holes or labels, whereas in $\mathcal{H}$ the first element of a constraint pair must be a hole and the second must be a label.

2. Bos restricts the syntax of PLU in an (abortive) attempt to characterise exactly those expressions which have at least one disambiguation. To do this, Bos imposes a 'meta–constraint' on the syntax of PLU, permitting only those expressions whose constraint diagram forms an upper semi–lattice.[5] An upper semi–lattice is defined as follows:

   Definition 3.2.16 (Upper Semi–Lattice). An *upper semi–lattice (*or *join semi–lattice)* is a poset $\langle A, \leqslant\rangle$ (where $\leqslant$ is a partial order over set A) such that for all $a, b \in A$, there exists a supremum (least upper–bound) of a and b, and this supremum is in A.

   By contrast, we do not impose any such restriction on the syntax of $\mathcal{H}$.

Fortunately, nothing really important hinges on these differences between $\mathcal{H}$ and PLU; we justify this view by addressing the two points above in turn:

[4] A complete presentation of PLU is presented by Bos (Bos, 1995) and Monz and de Rijke (Monz and de Rijke, 1998).

[5] Landman presents a gentle introduction to lattices (Landman, 1991, p.234–283).

1. Let $\Psi = \langle H, L, C \rangle$ be an Underspecified Representation in PLU. Since H denotes the set of holes used in Ψ, this set is equivalent to the set $H(\xi)$ in the CHR (ξ, C) with $\xi : L' \rightarrow \mathcal{L}^H$. Also, because L is just the set of labelled h–formulas of Ψ it is equivalent to the set $L(\xi)$, provided we let L' be exactly the set of labels used in Ψ. Finally, the freer use of constraints in PLU (compared with $\mathcal{H}$) does not give PLU superior expressive power. This claim hinges on the fact that it makes no sense to allow formulas without holes to dominate anything. Therefore, instead of putting a label l into the first element of a constraint pair, we can put the holes occurring in the formula labelled by l in its place. Also, since all formulas are uniquely labelled, we can express that a hole h is dominated by some other hole h' by asserting that the label(s) of the formula(s) in which h occurs is (are) dominated by h'.

2. Unfortunately, Bos does not prove that every PLU expression whose constraint diagram forms an upper semi–lattice has at least one admissible plugging, nor that admissible pluggings always yield well–formed formulas of $\mathcal{L}$ as disambiguations. Moreover, it is easy to show that Bos' upper semi–lattice restriction is necessary but not sufficient. It is interesting to note that similar constraints are discussed by van Genabith and Crouch, but they claim that the lattice constraint stems from purely linguistic considerations (van Genabith and Crouch, 1996, pp.5–6); by contrast, Bos' meta–constraint has no linguistic basis. We therefore dismiss Bos' (not very interesting or useful) restriction on the syntax of PLU.

In general, our version of $\mathcal{H}$ is mathematically cleaner than Bos' PLU. For example, Bos does not specify how the Underspecified Representations of PLU should be disambiguated (at least not in a mathematically precise way). In this respect, our version of Hole Semantics provides a clearer and more precise alternative to PLU. We now turn our attention to MRS.

3.2.3 Minimal Recursion Semantics

We claim that MRS is just a notational variant of a subset of Hole Semantics; accordingly, we do not make a full presentation of MRS here.[6] Instead, we now

[6] A complete presentation of MRS is presented by Copestake et al. (Copestake et al., 1999).

turn to the task of describing how the syntax of MRS differs from that of $\mathcal{H}$. Readers uninterested in the details of MRS can skip to the next section.

In common with Underspecified Representations in PLU, Underspecified Representations in MRS contain three elements: a 'top hole' (like h_0 in $\mathcal{H}$), a set of labelled h–formulas (called *elementary predications* in MRS) and a set of constraints (which, in common with the constraints of $\mathcal{H}$, we view as a set of pairs). There are five principal differences between $\mathcal{H}$ and MRS. We begin by listing these differences and then we will argue that they are not important.

1. The syntax of MRS allows 'h–formulas' to be co–indexed, the syntax of $\mathcal{H}$ does not. That is, an expression of MRS may contain two (or more) h–formulas $l_i : \varphi$ and $l_i : \psi$ sharing a common label; by contrast, every h–formula in a $\mathcal{H}$–expression is assigned a unique label.

2. The notation used to represent quantified noun–phrases in MRS is different to that in $\mathcal{H}$. For example, in MRS the quantifier 'every' may be represented by 'every(x, h_1, h_2)' where h_1 is a 'scope' (or 'restriction') argument place–holder and h_2 is a 'body' (or 'focus') argument place–holder; such expressions are not permitted by the syntax of $\mathcal{H}$.

3. Instructions for 'surface level' pluggings can be encoded within MRS h–formulas; by contrast, this is not possible in $\mathcal{H}$. For example, it is permissible for an MRS expression to contain both of the labelled h–formulas $l_i : \text{every}(x, h_j, h_k)$ and $h_j : \text{man}(x)$. Because $\text{man}(x)$ is labelled by h_j, which occurs as a hole in $\text{every}(x, h_j, h_k)$ any plugging of the MRS expression containing both of these h–formulas must fill h_j with $\text{man}(x)$. This notation cannot be exactly copied in $\mathcal{H}$, because in any expression (ξ, C) of $\mathcal{H}$, the sets $H(\xi)$ and $L(\xi)$ are disjoint.

4. Although syntactically speaking the constraints of both MRS and $\mathcal{H}$ are pairs, MRS constraints are not interpreted as 'dominance constraints' (as their counterparts are in $\mathcal{H}$). Instead, constraints in MRS are interpreted in terms of the 'equality modulo quantifiers' (QEQ) condition:

   Definition 3.2.17 (QEQ Condition). A hole h is *QEQ (Equal Modulo Quantifiers)* to some label l just in case $h = l$ or there is some (non–repeating) chain of one or more labelled (quantified) h–formulas $E_1, \ldots, E_n$ such that h is equal to the label of E_1, l is equal to the 'body argument'

hole of E_n and, for all pairs in the chain E_m, E_{m+1} the label of E_{m+1} is equal to the 'body argument' of E_m. If h is QEQ l then we write $h \leqslant_q l$. (Note that, despite its name, this relation is not symmetric).

5. The notion of 'pluggings' does not occur in MRS. Instead, in MRS holes (called 'handle arguments') are associated with h–formulas by 'equating' holes and labels. If a hole h is 'equal' to label l in the MRS expression ψ (that is, all instances of l in ψ are replaced by h) then to (partially) disambiguate ψ, we must substitute the conjunction of the formulas now labelled by h into each occurrence of the hole h. It is hoped that this process will 'collapse' expressions of MRS into well–formed formulas of $\mathcal{L}$.

There are no other differences between MRS and $\mathcal{H}$. We will now argue that all of these differences may be trivially dismissed. We deal with each of the differences in turn:

1. Co–indexing of h–formulas in MRS basically amounts to a notational variant of conjunction. That is, the way in which co–indexed h–formulas are interpreted in MRS means that, we can replace any two co–indexed h–formulas $l_i : \varphi$ and $l_i : \psi$ occurring in an MRS expression with the single labelled h–formula $l_i : \varphi \wedge \psi$ which has the same interpretation as the original pair of MRS h–formulas. Since terms like $l_i : \varphi \wedge \psi$ are permitted by the syntax of (labelled) h–formulas in $\mathcal{H}$, this apparent difference between MRS and $\mathcal{H}$ need not concern us.

2. Although the notation used to represent quantified noun–phrases in MRS looks different to that in $\mathcal{H}$ we can easily interchange between these notational alternatives. That is, the MRS h–formulas 'every(x, h_1, h_2)' and 'some(x, h_1, h_2)' have the same interpretation as the h–formulas '$\forall x(h_1 \rightarrow h_2)$' and '$\exists x(h_1 \wedge h_2)$' of $\mathcal{H}$ respectively. Therefore, this difference too can be ignored.

3. We can deal with any 'surface level' pluggings encoded in an expression of MRS simply by doing the substitution that they encode. For example, if both of the labelled h–formulas $l_i : \text{every}(x, h_j, h_k)$ and $h_j : \text{man}(x)$ occur in some MRS expression we can replace them with the single labelled h-formula $l_i : \forall x(\text{man}(x) \rightarrow h_k)$. Thus we have managed to capture exactly

the information encoded in the two original labelled h–formulas in a single labelled h–formula which is recognised by the syntax of $\mathcal{H}$.

4. Clearly, dominance (or synonymously, ancestorship) is not the same as the QEQ relation. The natural question to ask then is: Which of "$\leqslant$" (dominance) and "$\leqslant_q$" (QEQ) is more expressive? Copestake et al. claim that 'the "$\leqslant_q$" relation corresponds to an equality condition or a very specific form of [the] outscopes [or dominance] relation' (Copestake et al., 1999, p.10). Indeed, it is quite transparent that for any pair (h, l), with h a hole and l a label, if $h \leqslant_q l$ then $h \leqslant l$. However, the converse of this statement is false. So what class of constraints can be expressed using dominance ($\leqslant$) but not QEQ ($\leqslant_q$)? Well, the "$\leqslant_q$" relation can only be satisfied by those pairs whose elements are equal or whose elements occur as the labels and holes of h–formulas in a quite specific way. But this restriction is not really interesting, since in terms of expressive power it has only a slight (negative) impact. For example, it is difficult to represent expressions in which scope ambiguity arises as the result of the interaction of negation and any other operator(s) using QEQ constraints. Thus, we claim that, MRS is a notational variant of a (not very interesting) *subset* of $\mathcal{H}$. Having made this observation, we do not feel that it merits any further discussion.

5. In $\mathcal{H}$ and MRS, to disambiguate a formula, every hole in that formula must be assigned a unique label. In $\mathcal{H}$ we use a bijection (or 'plugging') between holes and labels to describe this assignment; by contrast, in MRS holes are associated with (unique) labels by equating each hole to some unique label. Clearly, this difference between the way holes are associated with labels in $\mathcal{H}$ and the way they are associated in MRS is not significant. Another possible alternative would be to define 'pluggings' as permutations of holes over a fixed ordering of labels (or visa versa).

We have argued that MRS is a notational variant of a subset of $\mathcal{H}$. The interpretation of constraints in MRS is a (not very useful) restriction of the notion of ancestorship used in $\mathcal{H}$. All of the syntactic differences between the two languages are cosmetic and we have shown how to (trivially) interchange between expressions in one language and expressions in the other. We now turn our attention to another constraint language, CLLS, in which formulae are regarded as trees and Underspecified Representations are defined in terms of partially specified trees.

3.2.4 The Constraint Language for Lambda Structures

Since its conception, CLLS (Egg et al., 1998) has been studied in some detail; (most of this research has been conducted as part of the CHORUS project).[7] We claim that all of the differences between CLLS and $\mathcal{H}$ are immaterial. Before we list these differences, we need to acquaint ourselves with the basic technical apparatus of CLLS. As usual, we begin with an informal overview.

Underspecified Representations in CLLS encode meta–level constraints over object–level formulas (that is, formulas of ordinary first–order logic $\mathcal{L}$). These Underspecified Representations are interpreted by a special class of models. A model is a finite (labelled) tree structure; we allow only those labelled tree structures which are (notational variants of) well–formed formulas of $\mathcal{L}$ to be models in CLLS. For any Underspecified Representation φ of CLLS, the set of models which satisfy φ is the same as the set of well–formed formulas which satisfy the (meta–level) constraints encoded by φ. Therefore, the disambiguations of an Underspecified Representation in CLLS are its models. The syntax of CLLS allows us to express five types of (meta–level) constraint: dominance, labelling, inequality, scope parallelism and lambda binding. For our purposes, it will be possible to restrict this syntax: we will allow only three types of constraint, dominance (denoted '$\lhd^*$'), labelling (denoted ':') and inequality (denoted '$\neq$').[8] Informally, if X and Y are tree nodes then we say that 'X dominates Y' (written $X \lhd^* Y$) if the 'path' beginning at the (unique) 'root' node and terminating at node Y passes through node X. The root node dominates everything. Informally, the relation symbol ':' will be used as follows: we will write $X : f(X_1, \ldots, X_n)$ to indicate that the node X is labelled by the symbol 'f' and has children (from left to right) $X_1, \ldots, X_n$ respectively. The use of the these symbols and the basic idea behind CLLS are best explained using an example. Consider once more the scope ambiguous sentence 'every man loves a woman'.

Example 3.2.18. We claim that the CLLS Underspecified Representation

$$\begin{aligned}\varphi \;=\; & v_0 \lhd^* v_1 \wedge v_0 \lhd^* v_2 \wedge v_1 : \forall x(v_3) \wedge v_3 :\rightarrow (v_4, v_5) \wedge v_4 : \mathrm{man}(v_6) \\ & \wedge v_6 : x \wedge v_2 : \exists y(v_7) \wedge v_7 : \wedge(v_8, v_9) \wedge v_8 : \mathrm{woman}(v_{10}) \wedge v_{10} : y \\ & \wedge v_5 \lhd^* v_{11} \wedge v_9 \lhd^* v_{11} \wedge v_{11} : \mathrm{loves}(v_{12}, v_{13}) \wedge v_{12} : x \wedge v_{13} : y\end{aligned}$$

[7] See `http://www.coli.uni_sb.de/cl/projects/chorus.html` for information about the CHORUS project.

[8] A complete account of CLLS is presented by Egg et al. (Egg et al., 2001).

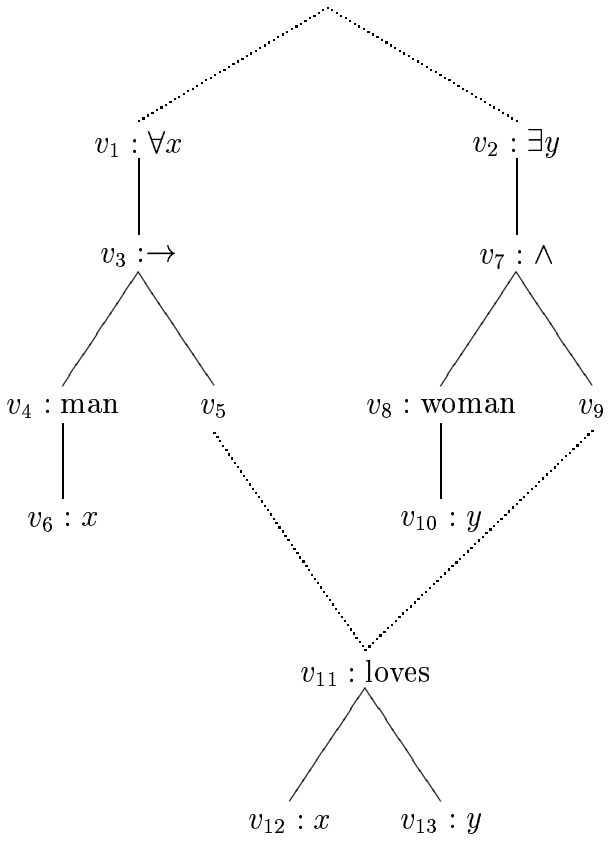


Figure 3.4: A representation of the CLLS Underspecified Representation which encodes the ambiguous content of the sentence 'every man loves a woman'.

encodes the disambiguations of the sentence 'every man loves a woman'. This formula may be depicted diagrammatically as shown in figure 3.4.

The exact relationship between CLLS Underspecified Representations and partial tree diagrams (like figure 3.4) is made clear by Koller and Niehren:

> A constraint graph is a directed graph with node labels and [two] kinds of edges: solid edges and dotted edges. Nodes of the graph stand for variables in a constraint; node labels together with solid edges stand for labelling constraints ... and dotted lines stand for dominance constraints. (Koller and Niehren, 1999, p.33)

To find a model of φ we must find a labelled tree structure which satisfies all of the (dominance and labelling) constraints encoded by φ. In doing so, we must assign labelled nodes to 'meta–variables', (the meta–variables in our example are $v_1, \ldots, v_{13}$). In our example, there are two structure/assignment pairs which satisfy φ, namely the labelled trees corresponding to the formulas $\forall x(\text{man}(x) \rightarrow \exists y(\text{woman}(y) \wedge \text{loves}(x, y)))$ and $\exists y(\text{woman}(y) \wedge \forall x(\text{man}(x) \rightarrow \text{loves}(x, y)))$. We now make these ideas more formal; we begin with the syntax of CLLS.

Definition 3.2.19 (CLLS Constraint). Let $X, X_1, \ldots, X_n$ and Y be elements of an infinite set of meta–variables and f be an n–ary function symbol. A *CLLS constraint* φ is a conjunction of atomic constraints; there are atomic CLLS constraints for dominance, labelling and inequality:

$$\varphi ::= X \lhd^* Y \mid X : f(X_1, \ldots, X_n) \mid X \neq Y \mid \varphi \wedge \varphi'.$$

To define what we mean by the term 'disambiguation' within the context of CLLS we must first specify precisely which class of structures we will accept as models of CLLS constraints. To do this we will need the familiar notion of a finite tree: $V = \{u, v, w, \ldots\}$ is a finite set of nodes and $E \subseteq V \times V$ is a set of edges, such that (V, E) is a graph with the in–degree of each node being at most 1 and exactly one node having in–degree 0 (called the *root*). Let (V, E) be a finite tree and Σ be some set of symbols such that $|V| = |\Sigma|$. We call any bijection $L : V \rightarrow \Sigma$ a *labelling function* of tree (V, E) over Σ. We call any pair consisting of a tree and a labelling function over Σ a *labelled tree* over Σ. We refer to the set Σ as the *node labels* of the labelled tree $((V, E), L)$. In short, a labelled tree is a tree and an assignment of symbols to the nodes of that tree.

Definition 3.2.20 (Satisfaction). Let T be a labelled tree with node labels $\texttt{nodes}(T)$ and $\texttt{var}$ be a set of meta–variables. A tree structure $\mathfrak{T} = (T, v)$ where $v : \texttt{var} \to \texttt{nodes}(T)$ is said to *satisfy* a CLLS constraint if

1. $\mathfrak{T} \models X \lhd^* Y$ if and only if $v(X)$ dominates $v(Y)$, and
2. $\mathfrak{T} \models X : f(X_1, \ldots, X_n)$ if and only if $v(X)$ is labelled by f and has children $v(X_1), \ldots, v(X_n)$, and
3. $\mathfrak{T} \models X \neq Y$ if and only if $v(X) \neq v(Y)$.

We restrict the class of tree structures further still, to only those which are labelled in such a way that they represent well–formed formulas of $\mathcal{L}$. We call such tree structure *formula structures*. We are now ready to define what we mean by the term 'disambiguation' within the context of CLLS.

Definition 3.2.21 (Disambiguation). Let φ be a CLLS constraint and $\mathfrak{T}$ be a formula structure. If $\mathfrak{T} \models \varphi$ then we call (the well–formed formula represented by the tree) $\mathfrak{T}$ a *disambiguation* of φ.

It is easy to see that the formulas $\forall x(\text{man}(x) \to \exists y(\text{woman}(y) \wedge \text{loves}(x, y)))$ and $\exists y(\text{woman}(y) \wedge \forall x(\text{man}(x) \to \text{loves}(x, y)))$ are disambiguations of the CLLS constraint φ from example 3.2.18.

We must now specify how CLLS differs from $\mathcal{H}$. At first sight this task may appear hopeless, since CLLS is a first–order constraint language over formulas whereas $\mathcal{H}$ consists of CHR structures and a completely separate disambiguation module (involving admissible pluggings and so on). However, nothing important hinges upon this difference. Indeed, the only real difference between CLLS and $\mathcal{H}$ is that in CLLS every Underspecified Representation has either no disambiguations (or "solutions") or infinitely many, by contrast, this is not true of $\mathcal{H}$. This is because CLLS allows new symbols to be artificially introduced in order to find solutions to the constraints encoded in its formulas. For example, the CLLS constraint represented in figure 3.4 has infinitely many solutions, for example $e(f(a, b), g(c, d))$, where e is any new (artificially introduced) symbol. By contrast, for any CHR ξ, if $\hat{\xi}$ is a disambiguation of ξ then any symbol occurring in $\hat{\xi}$ must also occur in ξ. That is, we do not permit artificial constructions in $\mathcal{H}$. This discrepancy is noted by Koller:

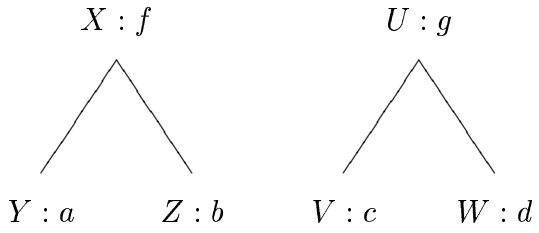


Figure 3.5: A CLLS constraint which has infinitely many solutions.

> In Hole Semantics, the interpretation is given by means of pluggings, where the holes and labels are identified. In contrast, dominance constraints are interpreted by embedding descriptions into trees that may contain *more* material. (Koller et al., 2003, p.1)

This anomaly does give CLLS one (manufactured) advantage over $\mathcal{H}$: it allows us to establish that for a particular fragment of dominance constraints (called *normal* dominance constraints) the 'plugging problem' (that is, the problem of determining whether a CLLS constraint has a solution) is solvable in polynomial time (Koller et al., 2000, p.10, Thm.4·7). However, the implications of artificially 'inventing' new symbols on demand (particularly within linguistic contexts) are not made explicit by Koller et al. (Koller et al., 2000). Indeed, the whole enterprise of looking for a class of Underspecified Representations for which the plugging problem has low complexity is questionable. To begin with, for any ('real–life') natural language sentence S, it is (in general) easy to answer the question: does S have a disambiguation? Secondly, ('real–life') linguistic ambiguity problems do not get large enough to make complexity issues important. For our purposes, it is simple to 'fix' the unnatural ability in CLLS to invent new symbols – for any CLLS formula φ, we simply insist that $\mathfrak{T} \models \varphi$ only if v maps $\mathtt{var}(\varphi)$ onto $\mathtt{nodes}(T)$.[9] With this sensible restriction, CLLS is equivalent to $\mathcal{H}$.

Koller presents a simple back–and–forth encoding between Hole Semantics and CLLS dominance constraints (Koller et al., 2003, p.4, Thm.4). We now take a brief excursion to reproduce Koller's description of the encoding. Roughly, any variable occurring only on the left–hand side of a labelling constraint is called a 'root'; any variable occurring only on the right–hand side of a labelling constraint is called a 'hole'. Koller claims that any formula of PLU can be mapped onto a CLLS constraint with the same interpretation as follows:

[9] This restriction is the equivalent to Koller's 'constructive solutions' restriction (Koller et al., 2003, p.3).

> We first encode every labelled formula $l : f(h_1, \ldots, h_n)$ as the labelling constraint $l : f(h_1, \ldots, h_n)$. We encode every $\mathcal{H}$ constraint $l \leqslant h$ as a dominance constraint $h \lhd^* l$ – except if h is the unique top hole and does not occur as a hole in a labelled formula. Finally, we add a constraint $l \neq l'$ for every label l. (Koller et al., 2003, p.4)

Similarly, any CLLS constraint φ can be mapped onto a PLU formula with the same interpretation as follows:

> We first split the variables $\mathtt{var}(\varphi)$ into holes and labels: roots become labels and holes become holes. Then we encode every labelling constraint $X : f(X_1, \ldots, X_n)$ as the labelled formula $X : f(X_1, \ldots, X_n)$, and we encode every dominance constraint $X \lhd^* Y$ as the constraint $Y \leqslant X$. Finally, we add a top hole symbol h_0 and the constraint $l \leqslant h_0$ for every label l. (Koller et al., 2003, p.4)

In order to state the exact relationship between CLLS and PLU Koller places extra restrictions on each of these languages. The following definitions are used for this purpose:

Definition 3.2.22 (Proper PLU Underspecified Representation). An Underspecified Representation U in PLU is called *proper* if it has the following properties:

1. U has a unique *top element,* from which all other nodes in the graph can be reached.
2. The graph of U is *acyclic.*
3. Every hole and every label occurs exactly once in the labelled h–formulas of U.

Definition 3.2.23 (Normal CLLS Constraint). A CLLS constraint φ is called *normal* if it has the following properties:

1. Every variable occurs in a labelling constraint.
2. Let X be any variable. Then X occurs at most once on the right–hand side of a labelling constraint and X cannot occur on the left–hand side of two different labelling constraints. Variables that do not occur on the left–hand side of any labelling constraint are called *holes*; variables that do not occur on a right–hand side of any labelling constraint are called *roots.*

3. If $X \lhd^* Y$ occurs in φ then X is a hole and Y is a root.

4. If X and Y are not holes then there is a constraint $X \neq Y$ in φ.

Definition 3.2.24 (Compact CLLS Constraint). A CLLS constraint is called *compact* if no variable occurs in two different labelling constraints.

Definition 3.2.25 (Constructive Solutions). Suppose that the tree structure $\mathfrak{T} = (T, v)$ satisfies the CLLS constraint φ. Then $\mathfrak{T}$ is called a *constructive solution* if every node in $\mathfrak{T}$ is denoted by a variable in $\mathtt{Var}(\varphi)$ on the left–hand side of a labelling constraint, where $\mathtt{Var}(\varphi)$ denotes the set of meta–variables occurring in φ.

Intuitively, a solution is called constructive if it consists only of material mentioned in the labelling constraint. Once the syntax of CLLS and PLU have been refined using these definitions it is possible to show that CLLS and PLU are essentially the same:

Theorem 3.2.26 (Koller). *Compact normal dominance constraints φ with acyclic graphs and proper PLU Underspecified Representations U can be encoded into each other, in such a way that the pluggings of U and the constructive solutions of φ correspond.*

Proof of Theorem 3.2.26. A detailed proof is presented by Koller (Koller et al., 2003, p.4, Thm.4). □

The framework of CLLS is a mathematically precise Underspecified Representation Language for which key complexity issues are well understood. We conclude this section by (briefly) reviewing some of the more interesting aspects of the research completed by the CHORUS team to date. We have seen that the disambiguations of CLLS constraints are exactly the solutions to the constraints encoded in that constraint (where solutions are labelled trees). But how do we solve such constraints, and how hard is it in general to determine whether or not a given set of dominance constraints has a solution at all? Both of these questions are addressed by Koller et al. (Koller et al., 1998). The plugging problem[10] for CLLS (that is, the problem of determining whether or not a set of dominance constraints actually has a solution) is shown to be NP–complete (Koller et al., 1998). We now take a brief excursion to sketch the proof of this claim.

[10]The CHORUS team use the term 'satisfiability problem' instead of 'plugging problem'.

The proof that the plugging problem for CLLS is in NP is presented by Koller et al. (Koller et al., 1998); we begin by outlining the key features of the algorithm used in this proof. The algorithm consists of three steps. The first step is to guess (nondeterministically) for each pair X, Y of variables in φ whether X dominates Y or not:

$$\text{(Choice)} \quad \mathbf{true} \rightarrow X \lhd^* Y \ \mathbf{or} \ \neg X \lhd^* Y,$$

where **or** stands for nondeterministic choice. In the second step φ is saturated according to seven *propagation* rules: In the third step unsatisfiable constraints are detected by applying four *clash* rules: This algorithm is described by Koller:

> After the initial guessing step, the algorithm applies all instances of all propagation and clash rules. We call a constraint to which no clash rule can be applied *clash–free*, the result of applying all possible rules to a constraint for as long as the constraint is clash–free its *saturation*, and a constraint which is its own saturation *saturated*. The algorithm outputs that its input is satisfiable if it can find a clash–free saturation (that is, can apply the guessing step in such a way that subsequent propagation and clash rules won't produce **false**); otherwise, it outputs that the input is unsatisfiable. (Koller et al., 1998, p.5)

Koller uses this algorithm to prove the soundness and completeness of CLLS:

Proposition 3.2.27 (Soundness). *A satisfiable dominance constraint has a clash–free saturation.*

Proposition 3.2.28 (Completeness). *A saturated and clash–free constraint is satisfiable.*

We are most interested in the result that the plugging problem for CLLS is in NP:

Proposition 3.2.29. *The plugging problem of the positive existential fragment over dominance constraints (and, of course, all smaller languages) is in NP.*

Complete details of the proof of proposition 3.2.29 are provided by Koller et al. (Koller et al., 1998, pp.4–11).

To show that the plugging problem is NP–hard, we must construct a polynomial transformation from some (already known) NP–complete problem to the

plugging problem in CLLS. Koller et al. show NP–hardness by reduction from 3–SAT (Koller et al., 1998, p.11, Thm.1 *and* Koller, 1999, p.19, Thm.4·19). That is, Koller et al. show that: any formula ϕ in 3–CNF is satisfiable (that is, there is a valuation satisfying ϕ) if and only if there is a tree structure which satisfies the encoding of ϕ in CLLS. Complete details are presented by Koller et al. (Koller et al., 1998, pp.11–14) and Koller (Koller, 1999, pp.79–91). We note in passing that Koller and Niehren also present an approach to solving (general) CLLS constraints using concurrent programming in OZ (Koller and Niehren, 1999 pp.70–82), along with a brief introduction to OZ (Koller and Niehren, 1999, pp.41–66).

We conclude our discussion of CLLS by sketching the details of the main result reported by Koller et al. (Koller et al., 2000, p.10, Thm.4·7): that there exists a fragment of dominance constraints (that is, the class of 'normal' constraints) for which the CLLS plugging problem is solvable in deterministic polynomial time.

Definition 3.2.30 (Normal Constraint). A CLLS constraint φ is called *normal* only if all four of the following conditions hold:

No overlap $X \neq Y$ is a constraint in φ if and only if X and Y are distinct variables that are both labelled in φ (possibly with the same label).

Tree shaped fragments Any variable X occurring in φ appears at most once as a parent and at most once as a child in any labelling literal of φ.

Dominances go from holes to roots (We call any unlabelled variable which appears in the constraints of φ a *hole* of φ). If $X \lhd^* Y$ is a constraint in φ then X and Y are unlabelled in φ, furthermore, Y must occur in some child position in φ.

No empty fragments If $X \lhd^* Y$ is a constraint in φ then there exist Z and f such that $Z : f(\ldots X \ldots)$ is a constraint in φ; that is, every hole in φ occurs in some child position.

Before we can outline the proof that the plugging problem is solvable in deterministic polynomial time for (the fragment consisting of) normal dominance constraints, we require some definitions involving cycles. Recall that, in an undirected graph, a *simple cycle* is a tour (along edges) which starts and ends at a particular node but otherwise visits each node at most once. We say that two edges are *adjacent* in a cycle if one is traversed immediately after the other. For our purposes the important definition is that of a *hypernormal cycle*:

Definition 3.2.31 (Hypernormal Cycle). A cycle in an undirected graph is called *hypernormal* if it does not contain two adjacent edges that 'emanate' from the same node. (An edge 'emanates' from the node at its tail).

The key result obtained by Koller is:

Proposition 3.2.32. *A normal dominance constraint is satisfiable if and only if its undirected constraint graph has no simple hypernormal cycles.*

This result is significant because Althaus et al. show that, for any normal constraint graph $G(\varphi)$, we can test $G(\varphi)$ for the presence of simple hypernormal cycles by solving a *perfect weighted matching problem* on a specific auxiliary graph of $G(\varphi)$ (Althaus et al., 2000, p.7, Lem.5·1).[11] Fortunately the perfect weighted matching problem can be solved in deterministic polynomial time (Althaus et al., 2000, p.8, Thm.5·1).[12]

In a nutshell, the plugging problem for the CLLS fragment consisting of only normal dominance constraints is reduced to the (deterministic polynomial time) perfect weighted matching problem, via hypernormal cycles. Therefore, given a normal constraint of CLLS, we can test in polynomial time whether or not it has a solution (and therefore we can test for the existence of disambiguations). A graph algorithm is provided by Althaus et al. which enumerates the *solved forms* of satisfiable normal dominance constraints (Althaus et al., 2000, pp.16–22). Solved forms describe infinite families of solutions; in practical terms, the solutions in a solved form are all equivalent. The idea is that each solved form of ϕ is a disambiguation of ϕ. Solved forms are used to eliminate the problem caused by the ability to invent new symbols in CLLS; namely the problem that every CLLS constraint has either zero or infinitely many solutions.[13]

Koller et al. express the opinion that normal dominance constraints form 'a natural fragment of dominance constraints whose restrictions should be unproblematic' (Koller et al., 2000, p.1). However, as we have discussed, CLLS is not free of peculiarities. Indeed, the proof that the plugging problem for normal constraints is in P is dependent upon the (unnatural) ability of CLLS to accept solutions which incorporate new 'invented' symbols. It should therefore not surprise us that, even if we restrict the syntax of $\mathcal{H}$ so that all CHRs are normal,

[11] Koller et al. provide a detailed account of this auxiliary graph (Koller et al., 2000, Sec.4·2, pp.5–6).

[12] This result was first proved by Galil et al. (Galil et al., 1986).

[13] Note that 'solved forms' and 'constructive solutions' serve the same purpose.

the plugging problem for $\mathcal{H}$ remains NP–complete (since, unlike CLLS, we are unable to invent new symbols in $\mathcal{H}$).

We have omitted some of the less important features of CLLS from our presentation for the sake of clarity. For example, CLLS is equipped with specific apparatus that deals with the interaction of scope ambiguity and ellipsis; that is, CLLS has specific apparatus for dealing with strict/sloppy ambiguities. The ellipsis in sentences suffering from strict/sloppy ambiguities enforces a 'parallelism' in the scopes of the clauses; so that, for example, in the sentence 'every student read a book, several researchers did too' the noun–phrase 'every student' has wide (narrow) scope if and only if the noun–phrase 'several researchers' has wide (narrow) scope. CLLS attempts to capture this parallelism. Moreover, the complete framework of CLLS includes apparatus for λ–binding and quantifier–binding constraints. The details of how λ–binding and quantifier–binding constraints are incorporated into the framework of CLLS are presented by Egg et al. (Egg et al., 2001, pp.3–22). Egg et al. also present an algorithm which maps natural language sentences onto CLLS constraints (Egg et al., 2001, Sec.5, pp.22–27); unfortunately, an analysis of the relationship between natural language and the various Underspecified Representation Languages is beyond the scope of this thesis.

Finally, we note that MRS structures can be 'efficiently translated' into CLLS dominance constraints (Niehren and Thater, 2003). However, this result is not a surprise to us since we claim that PLU, MRS, CLLS and UDRT are essentially all the same anyway. This completes our discussion about CLLS, we now turn our attention to UDRT.

3.2.5 Underspecified Discourse Representation Theory

The (total) disambiguations of an expression in any of the Underspecified Representation Languages we have seen so far are well–formed formulas of $\mathcal{L}$; by contrast, the disambiguations of expressions in UDRT are well–formed expressions in the language of ordinary Discourse Representation Theory (DRT). We therefore begin by sketching the syntax of ordinary DRT.[14]

[14]Kamp and Reyle present a detailed account of DRT (Kamp and Reyle, 1993, ch.1 & ch.2); we are only interested in DRT as a prerequisite to UDRT.

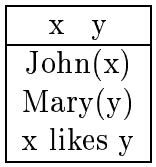


Figure 3.6: A DRS representing the sentence 'John likes Mary'.

Discourse Representation Theory

The expressions of DRT are called *Discourse Representation Structures* (DRSs). Consider the sentence 'John likes Mary'. This sentence may be represented by the DRS in figure 3.6. The variables in the top section of the box are called *discourse referents*; the top section of the box itself is referred to as the *universe*, and it may only contain discourse referents. The bottom section of DRSs consists of *conditions*. Intuitively, the condition 'John(x)' means 'x stands for the individual denoted by John'. The condition 'x likes y' is to be understood in the obvious way under instantiation of 'x' with 'John' and 'y' with 'Mary'. To save space we will linearise the DRS notation, writing $[x, y \mid \text{John}(x), \text{Mary}(y), x \text{ likes } y]$ instead of the usual 'box notation' demonstrated in figure 3.6; that is, we use a vertical bar to separate the universe from the conditions instead of a horizontal line.[15] Consider also the expression 'John knows prolog, he loves it'. This expression may be represented by the DRS:

$$[x, y, u, v \mid \text{John}(x), \text{prolog}(y), x \text{ knows } y, u = x, v = y, u \text{ loves } v]$$

The discourse referents 'u' and 'v' are used as 'markers' for the anaphora permitted by 'he' and 'it' respectively. The '=' sign is used to assign appropriate antecedents to each of these markers. Finally, simple DRSs like those we have seen may be combined as follows: if χ and χ' are DRSs then so are $\boxed{\neg\chi}$, $\boxed{\chi \vee \chi'}$, $\boxed{\chi \wedge \chi'}$ and $\boxed{\chi \Rightarrow \chi'}$. Blackburn and Bos present a detailed account of how DRSs are constructed from natural language inputs (Blackburn and Bos, 1999b, pp.37–57). The order in which the rules of this algorithm are applied is completely determined by the syntactic structure of the sentence being processed.

Finally, we note that it is trivial to translate simple DRSs into first–order logic. To translate a simple DRS into a formula of first–order logic we simply take the conjunction of its conditions and prefix this conjunction by a string of existential quantifiers – one for each discourse referent (changing any predicates

[15] We borrow our linear notation from Muskens (Muskens, 1996, pp.146–147).

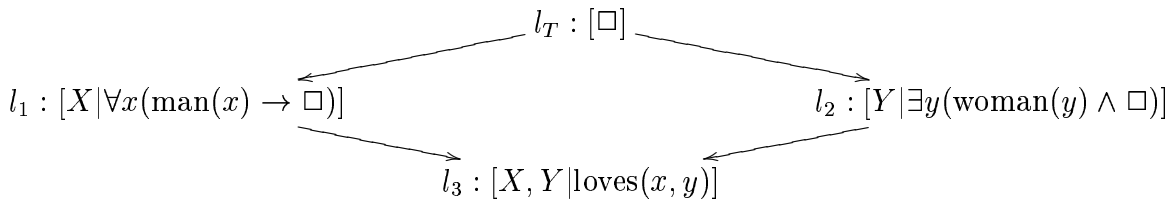


Figure 3.7: A UDRS encoding the ambiguous content of the sentence 'every man loves a woman'.

in infix notation to the more standard prefix notation). For example, the DRS representing the sentence 'John likes Mary' becomes $\exists xy(John(x) \wedge Mary(y) \wedge likes(x, y))$. Embedded DRS are translated in the obvious way (Blackburn and Bos, 1999b, pp.22–27). Now that we know a little about ordinary DRT we are ready to describe the underspecified version – UDRT.

Underspecified Discourse Representation Theory

An Underspecified Discourse Representation Structure (UDRS) consists of one or more DRSs together with some structural information. In fact, the key features of UDRT are very similar to those of Bos' PLU (Bos, 1995). Consider for example, the sentence 'every man loves a woman'. The ambiguous content of this sentence may be captured by the UDRS depicted in figure 3.7 As before, we insert fragments of formulas into occurrences of the hole symbol (this time '$\Box$' instead of $h_0, h_1, h_2, \ldots$) subject to a partial order over the fragments. Since UDRT and PLU are so alike, we do not make a full presentation of UDRT here; instead we catalogue the differences between UDRT and $\mathcal{H}$.[16]

1. A fully assembled UDRS (or more correctly a 'disambiguated' UDRS) is a DRT; by contrast, all disambiguations in $\mathcal{H}$ are well–formed formulas of $\mathcal{L}$.

2. Only one 'hole' symbol is recognised by the syntax of UDRT ('$\Box$'); whereas, an infinite stock of hole symbols ($h_0, h_1, h_2, \ldots$) is available to the h–formulas of $\mathcal{H}$.

3. The 'structural constraints' in UDRT are expressed as a partial order '$\leqslant$' over the set of labels; however, the constraints in any CHR ξ are pairs (h, l) with $h \in H(\xi)$ and $l \in L(\xi)$.

[16] A full account of UDRT is presented by Reyle (Reyle, 1995).

4. As with Bos' PLU (Bos, 1995), the syntax of UDRT is restricted so that every UDRS forms an upper semi–lattice; by contrast, we place no such restriction on the syntax of $\mathcal{H}$.

We will now argue that, for our purposes, nothing important hinges on any of these differences. As before, we deal with each of them in turn.

1. Since we have already said that it is trivial to translate DRSs into formulas of $\mathcal{L}$ this difference need not concern us.

2. To allow the holes occurring in UDRSs to be recognised by the syntax of $\mathcal{H}$ we simply need to number each occurrence of '$\Box$'; we must also replace the symbol 'l_T' by the special hole symbol 'h_0'.

3. Suppose that $l \leqslant l'$ is a 'constraint' in some UDRS which we wish to encode as a CHR. Suppose also that the (newly numbered) holes $\Box_1, \ldots, \Box_n$ occur in the DRT labelled by l. Then the set of constraints $\{(\Box_1, l'), \ldots, (\Box_n, l')\}$ contains exactly the same information as the original constraint $l \leqslant l'$. Therefore, we can easily 'translate' any UDRT constraint into an equivalent set of $\mathcal{H}$–constraints.

4. Finally, although the syntax of UDRT is restricted so that only upper semi–lattices are recognised, we have already discussed and dismissed this restriction with respect to PLU on page 65 (point 2).

Therefore we claim that UDRT is a notational variant of a sub–language of $\mathcal{H}$ (just as PLU is a notational variant of a sub–language of $\mathcal{H}$). That is, UDRT does not, in any interesting sense, provide us with an alternative Underspecified Representation Language to $\mathcal{H}$. This completes our presentation of UDRT.

We have seen that $\mathcal{H}$, PLU, MRS, CLLS and UDRT are all notational variants of one another (or at least notational variants of sub–languages of each other), modulo some not very interesting differences. We have also seen that some of the theoretical results in CLLS depend upon the (unnatural) ability of this language to construct (artificial) new nodes. We now turn our attention to three Underspecified Representation Languages which are all closely related to the earliest form of underspecification – *Storage*.

3.3 Raising

In this section, we report on three more Underspecified Representation Languages: Ambiguous Predicate Logic (van Eijck and Jaspars, 1996, pp.11–19), Storage (Cooper, 1983) and Quasi–Logical Form (Alshawi, 1992). Once more, we group these three languages together to reflect our belief that they share a common underlying structure. We begin by presenting Ambiguous Predicate Logic since we believe that this language (probably) subsumes the others.

3.3.1 Ambiguous Predicate Logic

Ambiguous Predicate Logic (APL) is an Underspecified Representation Language in which we can represent scope underspecification by means of a notation that allows unscoped operators as the ingredients of formulas (van Eijck and Jaspars, 1996, p.11). The basic idea behind this language is best conveyed using an example. Consider once again the sentence 'every man loves a woman'. We claim that the ambiguous content of this sentence can be represented by the formula:

$$(\forall x(\text{man}(x) \rightarrow \Box), \exists y(\text{woman}(y) \wedge \Box))\text{loves}(x, y).$$

In common with UDRT the symbol '$\Box$' is used in APL to denote a 'hole' or 'free–slot' into which other material may be inserted. The expressions '$\forall x(\text{man}(x) \rightarrow \Box(x))$' and '$\exists x(\text{woman}(x) \wedge \Box(x))$' are therefore called *operators*. Syntactically, operators are generated by the usual formation rules for well–formed formulas of $\mathcal{L}$, except that in each operator exactly one (token of a) subformula must be replaced by the '$\Box$' symbol. To disambiguate a formula in this language we simply 'pull out' all of its operators (to the front of the subformula in which they occur) one–by–one; ambiguity is introduced by the indeterminacy in the order in which we do this. Suppose that in our example, we choose to pull out the operator $\forall x(\text{man}(x) \rightarrow \Box)$ first. Doing so gives us:

$$(\exists y(\text{woman}(y) \wedge \Box))\forall x(\text{man}(x) \rightarrow \Box)\text{loves}(x, y).$$

Once an operator has been pulled out it is then applied. Applying an operator means to replace the symbol '$\Box$' occurring in it by the material to the immediate

right–hand side of the operator; in our example, we obtain:

$$(\exists y(\text{woman}(y) \wedge \Box))\forall x(\text{man}(x) \rightarrow \text{loves}(x, y)).$$

A final application results in the $\mathcal{L}$–formula:

$$\exists y(\text{woman}(y) \wedge \forall x(\text{man}(x) \rightarrow \text{loves}(x, y))).$$

It is not difficult to imagine that pulling the operators out in the opposite order yields:

$$\forall x(\text{man}(x) \rightarrow \exists y(\text{woman}(y) \wedge \text{loves}(x, y))).$$

We now make these ideas formal, beginning with the syntax of APL.

The syntax of APL consists of *terms* t (defined as in ordinary first–order logic), *formulae* φ (defined as in first–order logic, except that there are possibly underspecified operations), *operators* α (expressions which contain a free–slot $\Box$ into which formulae may be placed), and *contexts* C (which are *ordered* lists of operators). We reproduce the BNF definitions presented by van Eijck and Jaspars (van Eijck and Jaspars, 1996, p.13); letting $n \geqslant 1$,

$$\begin{array}{lcl} t & ::= & c \,|\, v, \\ \varphi & ::= & R(t_1, \ldots, t_n) \,|\, \neg\varphi \,|\, (\varphi_1 \wedge \varphi_2) \,|\, (\varphi_1 \vee \varphi_2) \,|\, \exists v\varphi \,|\, \forall v\varphi \,|\, (C_1, \ldots, C_n)\varphi, \\ \alpha & ::= & \Box \,|\, \neg\alpha \,|\, (\alpha \wedge \varphi) \,|\, (\varphi \wedge \alpha) \,|\, (\alpha \vee \varphi) \,|\, (\varphi \vee \alpha) \,|\, \exists v\alpha \,|\, \forall v\alpha \,|\, (C_1, \ldots, C_n)\alpha, \\ C & ::= & \alpha \,|\, C\alpha. \end{array}$$

A context is either an operator (a *simple context*) or a finite non–empty (ordered) list of operators (a *complex context*). The idea is that we may choose between contexts (ordered lists of operators), but once we have selected a context we are forced to pull out the bottom operator in that context (list). That is, unordered lists of contexts are used to express (a restricted class of) partial orders over operators. An unordered list of contexts may be viewed as analogous to a cigarette machine. Each context is comparable to a column of the cigarette machine, while operators are analogous to individual packets of cigarettes. We may choose between columns (contexts), but we may only select the bottom packet (operator) from any column (context) we choose. However, we note in passing that this scheme does not allow us to express all partial orders:

Observation 3.3.1. *Contexts do not allow us to express all partial orders.*

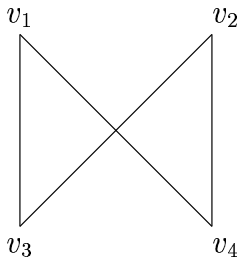


Figure 3.8: A partial order which cannot be expressed using contexts.

Proof of Observation 3.3.1. The partial order depicted in figure 3.8 cannot be expressed in terms of contexts (because we cannot duplicate elements). □

It should be clear that, a formula in APL is ambiguous only if it contains a non–singleton list of contexts; the ambiguity arises as a result of the indeterminate order in which the contexts should be applied. Before we can define a disambiguation procedure for APL we must specify the exact process for substituting formulas into operators. The list of contexts $C_1, \ldots, C_n$ may be abbreviated to $\widetilde{C_n^1}$. More generally, we will use the notation $\widetilde{C_j^i}$ to denote $C_i, C_{i+1}, \ldots, C_{j-1}, C_j$.

Definition 3.3.2 (Substitution Map). Let φ and ψ be formulas of APL, α be an operator, and $\widetilde{C_n^1}$ be a (possibly singleton) list of contexts. We define a *substitution map* $s : APL \to APL$ recursively, as follows:

$$\begin{aligned}
s[(\Box)\varphi] &= \varphi, \\
s[(\neg\alpha)\varphi] &= \neg s[(\alpha)\varphi], \\
s[(\alpha \wedge \psi)\varphi] &= s[(\alpha)\varphi] \wedge \psi, \\
s[(\psi \wedge \alpha)\varphi] &= \psi \wedge s[(\alpha)\varphi], \\
s[(\exists x\alpha)\varphi] &= \exists x s[(\alpha)\varphi], \\
s[(\forall x\alpha)\varphi] &= \forall x s[(\alpha)\varphi], \\
s[(\widetilde{C_n^1}\alpha)\varphi] &= (\widetilde{C_n^1}) s[(\alpha)\varphi].
\end{aligned}$$

We are now ready to spell out how we disambiguate formulas of APL.

Definition 3.3.3 (Disambiguation). Let $\widetilde{C_n^1}$ be an unordered list of contexts with $C_i = (\alpha_{i1}, \ldots, \alpha_{im_i})$ for $1 \leqslant i \leqslant n$ and let φ be a formula of APL. Then we

define the *set of disambiguations* of $(\widetilde{C_n^1})\varphi$ to be:

$$d((\widetilde{C_{i-1}^1}, (\alpha_{i1} \ldots \alpha_{im_i}), \widetilde{C_n^{i+1}})\varphi) = \left\{ d((\widetilde{C_{i-1}^1}, (\alpha_{i1} \ldots \alpha_{im_{i-1}}), \widetilde{C_n^{i+1}})s[(\alpha_{im_i})\pi]) \middle| 1 \leqslant i \leqslant n,\ \pi \in d(\varphi) \right\}.$$

Clearly, for any formula φ in APL the set of disambiguations of φ is non–empty. We conclude our presentation of APL with an example.

Example 3.3.4. Consider the (scope) ambiguous sentence 'every child is not clever'. We may represent the disambiguations of this sentence by the predicate calculus formulas $\forall x(\text{child}(x) \rightarrow \neg\text{clever}(x))$ and $\neg(\forall x(\text{child}(x) \rightarrow \text{clever}(x)))$. We claim that, the Underspecified Representation of 'every child is not clever' in APL should be $\varphi = (\forall x(\text{child}(x) \rightarrow \Box), \neg\Box)\text{clever}(x)$. The set of disambiguations of φ are derived as follows:

$$\begin{aligned}
d((\forall x(\text{child}(x) \rightarrow \Box), \neg\Box)\text{clever}(x)) = & \\
& \Big\{ d((\forall x(\text{child}(x) \rightarrow \Box))s[(\neg\Box)\text{clever}(x)]), \\
& \qquad d((\neg\Box)s[(\forall x(\text{child}(x) \rightarrow \Box))\text{clever}(x)]) \Big\} \\
= & \Big\{ d((\forall x(\text{child}(x) \rightarrow \Box))\neg\text{clever}(x)), d((\neg\Box)\forall x(\text{child}(x) \rightarrow \text{clever}(x))) \Big\} \\
= & \Big\{ s[(\forall x(\text{child}(x) \rightarrow \Box))\neg\text{clever}(x)], s[(\neg\Box)\forall x(\text{child}(x) \rightarrow \text{clever}(x))] \Big\} \\
= & \Big\{ \forall x(\text{child}(x) \rightarrow \neg\text{clever}(x)), \neg(\forall x(\text{child}(x) \rightarrow \text{clever}(x))) \Big\}.
\end{aligned}$$

There are no other disambiguations of φ and therefore we have obtained exactly the disambiguations we had hoped for.

This concludes our presentation of APL. We now turn our attention to two purely linguistic approaches to underspecification.

3.3.2 Quantifier Raising and Storage

Operator scope ambiguity is often dealt with using Quantifier Raising or Storage techniques. The best–known examples of these approaches are Cooper Storage (Cooper, 1983), which was later modified by Keller (Keller, 1986), and Quasi–Logical Forms (Alshawi, 1992). Unfortunately, Underspecified Representation Languages like these are all too often technically ill–defined. Therefore we will

not make a complete presentation of Storage and Quasi–Logical Forms (though we will give an overview); instead, we will present our attempt at a rational reconstruction of the key features of these languages (which we will call $\mathcal{R}$).

An Overview of Storage

Storage techniques provide a natural approach to underspecification. The basic idea behind Storage is: for any natural language sentence, we assign a *Store* to each node of its parse tree. Each Store incorporates a list of quantifiers, which are associated with nodes lower down in the tree. Given a scope ambiguous sentence, the order in which the quantifiers are retrieved from the Store determines the disambiguations of that sentence. Consider the sentence 'every man loves a woman'. We may represent this sentence by the Cooper Store

$$\langle \text{loves}(z_1, z_2), (\lambda P.\forall x(\text{man}(x) \rightarrow Px), 1), (\lambda P.\exists y(\text{woman}(y) \wedge Py), 2))\rangle.$$

Notice that quantified noun–phrases are represented by indexed λ–expressions (called 'operators') and these λ–expressions fill the argument places of the verb–phrase 'loves'. The process of disambiguating Stores is usually referred to as 'retrieval'. Informally, retrieval means to pull each of the operators out to the front of the Store (one–by–one) while at the same time performing the necessary β–reductions. Cooper Stores can be ambiguous precisely because the order in which the operators within a Store are pulled out is not necessarily fixed. The rule for pulling out operators is:

Rule 3.3.5 (Cooper Retrieval). Let σ_1 and σ_2 be (possibly empty) sequences of indexed operators and let $\langle \varphi, \sigma_1, (\beta, i), \sigma_2\rangle$ be a Store. Then the Store $\langle \beta\lambda z_i.\varphi, \sigma_1, \sigma_2\rangle$ is a (partial) disambiguation of $\langle \varphi, \sigma_1, (\beta, i), \sigma_2\rangle$.

Applying the rule to the operator indexed by '1' in our example (that is, $\langle \text{loves}(z_1, z_2), (\lambda P.\forall x(\text{man}(x) \rightarrow Px), 1), (\lambda P.\exists y(\text{woman}(y) \wedge Py), 2)\rangle$) gives us:

$$\langle \lambda P.\forall x(\text{man}(x) \rightarrow Px)\lambda z_1 \text{loves}(z_1, z_2), (\lambda P.\exists y(\text{woman}(y) \wedge Py), 2)\rangle.$$

By β–converting the first element twice we obtain:

$$\langle \forall x(\text{man}(x) \rightarrow \text{loves}(x, z_2)), (\lambda P.\exists y(\text{woman}(y) \wedge Py), 2)\rangle.$$

Applying the retrieval process to the λ–expression indexed by '2' yields:

$$\langle(\lambda P \exists y(\text{woman}(y) \wedge Py))\lambda z_2 \forall x(\text{man}(x) \rightarrow \text{loves}(x, z_2))\rangle.$$

Two further β–conversions give us:

$$\langle \exists y(\text{woman}(y) \wedge \forall x(\text{man}(x) \rightarrow \text{loves}(x, y)))\rangle.$$

Retrieving the two λ–expressions in the opposite order (that is, '2' then '1') would give us the Cooper Store with the single element $\forall x(\text{man}(x) \rightarrow \exists y(\text{woman}(y) \wedge \text{loves}(x, y)))$.

A well–known modification of Cooper Storage is to allow the nesting of Stores, so–called *Keller Storage* (Keller, 1986). We will delay any discussion about the nesting of operators until our presentation of the language $\mathcal{R}$ (which we believe captures the key features of Storage and Quasi–Logical Form).

An Overview of Quasi–Logical Form

Quasi–Logical Form (QLF) was the first Underspecified Representation Language to be used in a real–world Natural Language Processing application – namely, the Core Language Engine (CLE). The syntax of QLF is quite complex and many of the details are not relevant to our present discussion (for example, QLFs contain detailed linguistic information concerning tense, aspect, plurality and so on). We therefore discuss Moran's (heavily) simplified version of QLF (Moran, 1988). The basic idea is as follows: for any natural language sentence we can construct a QLF which encodes the meaning of that sentence by treating each verb–phrase as a predicate, and noun–phrases as terms in the argument positions of those predicates. Each term (representing a noun–phrase) is assigned a unique index and different scope relations (over those noun–phrases) are represented by specifying an order over the indices. In an unresolved (ambiguous) QLF, the order between all of these indices is (at least partly) unspecified; by contrast, a QLF is resolved (disambiguated) once the order between these indices is fixed. Perhaps the best way to get a feel for QLF is by looking at an example. Consider once again the sentence 'every man loves a woman'. We claim that the ambiguous content of this sentence may be captured by the following (heavily simplified) QLF:

$$\texttt{s} : \texttt{loves}(\texttt{term}(+\texttt{m}, \forall, \lambda\texttt{X}.\texttt{man}(\texttt{X})), \texttt{term}(+\texttt{w}, \exists, \lambda\texttt{Y}.\texttt{woman}(\texttt{Y}))).$$

Notice that the two noun–phrases are represented as terms that are the arguments of their syntactic mother, the `loves` verb–phrase. The first element of each of these terms is its unique index, the second element specifies the type of quantifier the term represents, and the third element is the so–called 'restriction' of the quantifier. The initial symbol of our example QLF '`s`' is a 'meta–variable' which under disambiguation must be instantiated with an ordered list of indices (which specifies an order of the noun–phrases). There are just two ways of instantiating `s` in our example (that is, just two ways of ordering `+m` and `+w`) and these instantiations specify two distinct disambiguations in the obvious way:

$$[+\mathtt{m}, +\mathtt{w}] : \mathtt{loves}(\mathtt{term}(+\mathtt{m}, \forall, \lambda\mathtt{X}.\mathtt{man}(\mathtt{X})), \mathtt{term}(+\mathtt{w}, \exists, \lambda\mathtt{Y}.\mathtt{woman}(\mathtt{Y}))) = \forall\mathtt{x}(\mathtt{man}(\mathtt{x}) \rightarrow \exists\mathtt{y}(\mathtt{woman}(\mathtt{y}) \wedge \mathtt{loves}(\mathtt{x}, \mathtt{y}))).$$

$$[+\mathtt{w}, +\mathtt{m}] : \mathtt{loves}(\mathtt{term}(+\mathtt{m}, \forall, \lambda\mathtt{X}.\mathtt{man}(\mathtt{X})), \mathtt{term}(+\mathtt{w}, \exists, \lambda\mathtt{Y}.\mathtt{woman}(\mathtt{Y}))) = \exists\mathtt{y}(\mathtt{woman}(\mathtt{y}) \wedge \forall\mathtt{x}(\mathtt{man}(\mathtt{x}) \rightarrow \mathtt{loves}(\mathtt{x}, \mathtt{y}))).$$

This completes our overview of QLF.

Now that we know a little about Storage and QLF we can present our attempt at a rational reconstruction of these languages, which we call $\mathcal{R}$ (for "raising").

3.3.3 The Language $\mathcal{R}$

As with our presentation of $\mathcal{H}$, we begin our presentation of the language $\mathcal{R}$ with our running example: 'every man loves a woman'. We can represent the ambiguous content of this sentence using the following $\mathcal{R}$–formula:

$$\mathrm{loves}(\forall x(\mathrm{man}(x) \rightarrow \Box(x)), \exists x(\mathrm{woman}(x) \wedge \Box(x))).$$

The symbol '$\Box$' is used in $\mathcal{R}$ to denote a 'hole' or 'free–slot' into which other material may be inserted. The expressions '$\forall x(\mathrm{man}(x) \rightarrow \Box(x))$' and '$\exists x(\mathrm{woman}(x) \wedge \Box(x))$' are therefore called *operators*. Syntactically, operators are generated by the usual formation rules for well–formed formulas of $\mathcal{L}$, except that in each operator exactly one (token of a) subformula must be replaced by the '$\Box$' symbol. Disambiguation is achieved in $\mathcal{R}$ by *raising* these operators (one–by–one). Ambiguity is introduced by the indeterminacy in the order in which operators are

raised. The raising operation consists of two steps: 'movement' and 'application'. Suppose that we choose to raise the operator $\forall x(\text{man}(x) \to \Box(x))$ first. To raise an operator to predicate P we begin by 'moving' it to the left–hand side of P. In our example, having decided to raise the operator $\forall x(\text{man}(x) \to \Box(x))$ first, we have no further choices to make (since there is only one predicate which we can raise it to); hence we obtain:

$$\forall x'(\text{man}(x') \to \Box(x))\text{loves}(\boldsymbol{x'}, \exists x(\text{woman}(x) \wedge \Box(x))).$$

Note that, upon movement, the name of the 'main variable' (or *major variable*) in the operator is changed; (in our case from x to x'). Notice also that the argument position in the predicate 'loves' left vacant as a result of the movement operation is filled by a copy of (the new name for) the main variable, (shown in bold font); this copy of the main variable is sometimes called its *trace*. Now we can 'apply' $\forall x'(\text{man}(x') \to \Box(x))$ to $\text{loves}(x', \exists x(\text{woman}(x) \wedge \Box(x)))$; application simply consists of replacing an occurrence of the symbol $\Box$ by the expression to its right–hand side. We therefore obtain $\forall x(\text{man}(x) \to \text{loves}(x', \exists x(\text{woman}(x) \wedge \Box(x))))$. It remains only to raise (that is, move then apply) the remaining operator $\exists x(\text{woman}(x) \wedge \Box(x))$ to obtain $\exists x''(\text{woman}(x)'' \wedge \forall x'(\text{man}(x') \to \text{loves}(x', x'')))$. Raising the operators in the opposite order yields $\forall x'(\text{man}(x') \to \exists x''(\text{woman}(x'') \wedge \text{loves}(x', x'')))$. We now make these ideas more formal, beginning with the syntax of $\mathcal{R}$.

Definition 3.3.6 (Operators and Formulas). We define *operators* α and *formulas* φ co–recursively:

$$\begin{aligned} \alpha &::= \Box(x) \mid \neg\alpha \mid \varphi \wedge \alpha \mid \alpha \wedge \varphi \mid \varphi \to \alpha \mid \alpha \to \varphi \mid \forall x(\varphi \to \alpha) \mid \exists x(\varphi \wedge \alpha), \\ \varphi &::= P^{(n)}(\xi_1, \ldots, \xi_n) \mid \neg\varphi \mid \varphi \wedge \psi \mid \varphi \to \psi, \end{aligned}$$

where $P^{(n)}$ is an n–ary predicate, ψ is a formula and each ξ_i for $1 \leqslant i \leqslant n$ is either a term (that is, a constant or variable) or an operator.

Before we can formalise the raising procedure, we need a precise definition of the term 'major variable'.

Definition 3.3.7 (Major Variable). Let α be an operator of $\mathcal{R}$, φ be a formula of $\mathcal{R}$ and x be a variable. The *major variable* of an operator is defined recursively as follows:

1. The major variable of $\Box(x)$ is x.

2. The major variable of the operators $\neg\alpha$, $\varphi \wedge \alpha$, $\alpha \wedge \varphi$, $\varphi \rightarrow \alpha$, $\alpha \rightarrow \varphi$, $\forall x(\varphi \rightarrow \alpha)$ and $\exists x(\varphi \wedge \alpha)$ is the major variable of α.

We are now ready to define the raising procedure.

Definition 3.3.8 (Raising). Let φ be a formula and α be a token of an operator occurring in φ. Let x be the major variable of α and x'' be a 'new' variable (that is, a variable which does not occur in φ). Define a function `raise` as follows:

$$\texttt{raise}(\varphi, \alpha) = \alpha[x''/x, \varphi[x''/\alpha]/\Box].$$

As usual $x[y/z]$ denotes the result of replacing all occurrences of z in x with y's. We make the sensible restriction that *free variables cannot be raised beyond their binders.* That is, we only permit raising which does not increase the number of free variables. We shall see why this restriction is necessary in example 3.3.10. Disambiguation in $\mathcal{R}$ is achieved by repeated application of the raising procedure. A formula is totally disambiguated once `raise` has been applied to all of its operators.

Lemma 3.3.9. *Let $\varphi \in \mathcal{R}$ contain $n \geqslant 0$ operators. Repeated application of* `raise` *will terminate, and the result will be a well–formed formula of $\mathcal{L}$.*

Proof of Lemma 3.3.9. Obvious. □

We conclude our presentation of the language $\mathcal{R}$ with an adapted version of a well–known example (Hobbs and Shieber, 1986, p.2).[17]

Example 3.3.10. Consider the scope ambiguous sentence 'every artist who admires a bee–keeper despises a carpenter'. According to Hobbs and Shieber, out of the six combinatorial ways of permuting the three quantifiers ('every', 'a' and 'a'), one reading is 'missing' (or 'disallowed'); namely the reading in which 'every artist' outscopes 'a carpenter' which in turn outscopes 'a bee–keeper' (Hobbs and Shieber, 1986, p.2). We note in passing that to ban unwanted readings Hobbs and Shieber's algorithm incorporates the rule that: 'a quantifier from elsewhere in a sentence cannot come after the quantifier associated with the head noun and before the quantifier associated with a noun–phrase in the head noun's complement

[17] The example we have adapted is 'Every representative of a company saw most samples' (Hobbs and Shieber, 1986, p.2).

[or in a relative clause embedded in any phrase whose head is the head noun]' (Hobbs and Shieber, 1986, p.3). We shall see that this restriction has the same effect as our free variable constraint. We claim that the following $\mathcal{R}$–formula encodes exactly the disambiguations allowed by Hobbs and Shieber:

$$\text{despises}(\forall x(\text{artist}(x) \rightarrow \Box(x)) \wedge \text{admires}(x, \exists y(\text{beekeeper}(y) \wedge \Box(y))), \\ \exists z(\text{carpenter}(z) \wedge \Box(z))).$$

We conclude our presentation of $\mathcal{R}$ by working through the derivation of the disambiguations of our version of the Hobbs and Shieber example. Readers uninterested in the details of this derivation may skip to the next section.

Our representation of 'every artist who admires a bee–keeper despises a carpenter' contains three operators, which we refer to as operators **A**, **B** and **C** as follows:

Operator A: $\forall x(\text{artist}(x) \rightarrow \Box(x)) \wedge \text{admires}(x, \exists y(\text{beekeeper}(y) \wedge \Box(y)))$,

Operator B: $\exists y(\text{beekeeper}(y) \wedge \Box(y))$,

Operator C: $\exists z(\text{carpenter}(z) \wedge \Box(z))$.

Suppose that we choose to raise the operator **A** first. Then the first two steps of the raising process yield:

$$\forall x'(\text{artist}(x' \rightarrow \Box(x)) \wedge \text{admires}(x', \exists y(\text{beekeeper}(y) \wedge \Box(y)))) \\ \text{despises}(x', \exists z(\text{carpenter}(z) \wedge \Box(z))).$$

Which, after application becomes:

$$\forall x'(\text{artist}(x' \rightarrow \text{despises}(x', \exists z(\text{carpenter}(z) \wedge \Box(z))) \wedge \\ \text{admires}(x', \exists y(\text{beekeeper}(y) \wedge \Box(y))))).$$

We now have a choice in the order in which we raise the operators **B** and **C**. Raising **B** before **C** yields:

$$\exists z'(\text{carpenter}(z' \wedge \exists y'(\text{beekeeper}(y') \wedge \forall x'(\text{artist}(x' \rightarrow \\ \text{despises}(x', z') \wedge \text{admires}(x', y'))))))).$$

Whereas, raising **C** before **B** yields:

$$\exists y'(\text{beekeeper}(y') \wedge \exists z'(\text{carpenter}(z' \wedge \forall x'(\text{artist}(x' \rightarrow \\ \text{despises}(x', z') \wedge \text{admires}(x', y')))))).$$

Now suppose that we choose to raise operator **C** first (instead of **A**). Then the first step of the derivation is:

$$\exists z'(\text{carpenter}(z') \wedge \Box(z)x(\text{artist}(x) \rightarrow \Box(x)))\wedge \\ \text{admires}(x, \exists y(\text{beekeeper}(y) \wedge \Box(y)), z')).$$

The second step is:

$$\exists z'(\text{carpenter}(z') \wedge \text{despises}(\forall x(\text{artist}(x) \rightarrow \Box(x))\wedge \\ \text{admires}(x, \exists y(\text{beekeeper}(y) \wedge \Box(y)), z')).$$

As before, we are now faced with a choice; we can either raise operator **A** or operator **B** next. Raising **A** before **B** yields:

$$\exists y'(\text{beekeeper}(y') \wedge \forall x(\text{artist}(x') \rightarrow \exists z'(\text{carpenter}(z')\wedge \\ \text{despises}(x', z') \wedge \text{admires}(x', y'))).$$

Raising **B** before **A** yields:

$$\forall x'(\text{artist}(x') \rightarrow \exists y'(\text{beekeeper}(y') \wedge \exists z'(\text{carpenter}(z')\wedge \\ \text{despises}(x', z') \wedge \text{admires}(x', y')).$$

Finally, we could have chosen to raise operator **B** before all else. Doing so would initially give us:

$$\exists y'(\text{beekeeper}(y') \wedge \Box(y))\text{despises}(\forall x(\text{artist}(x) \rightarrow \Box(x))\wedge \\ \text{admires}(x, y'), \exists z(\text{carpenter}(z) \wedge \Box(z))).$$

An application step yields:

$$\exists y'(\text{beekeeper}(y') \wedge \text{despises}(\forall x(\text{artist}(x) \rightarrow \Box(x)) \wedge \text{admires}(x, y'), \exists z(\text{carpenter}(z) \wedge \Box(z)))).$$

We now seem to have a choice between which of **A** and **C** to raise next. Raising **A** before **C** gives us:

$$\exists z'(\text{carpenter}(z') \wedge \forall x'(\text{artist}(x') \rightarrow \exists y'(\text{beekeeper}(y') \wedge \text{despises}(x', z') \wedge \text{admires}(x', y')))).$$

However, we claim that our algorithm does not allow us to reverse this order; that is, we claim that our free variable condition prevents us from raising **B** then **C** then **A**. Suppose for a moment that we do not realise that raising operator **C** next will lead to a violation of our constraint; raising **C** next would give us:

$$\exists z'(\text{carpenter}(z') \wedge \exists y'(\text{beekeeper}(y') \wedge \text{despises}(\forall x(\text{artist}(x) \rightarrow \Box(x))) \wedge \text{admires}(x, y'), z')).$$

Our final step would then be to raise operator **A** as follows:

$$\forall x'(\text{artist}(x') \rightarrow \Box(x)) \wedge \text{admires}(x', y') \exists z'(\text{carpenter}(z') \wedge \exists y'(\text{beekeeper}(y') \wedge \text{despises}(x', z'))).$$

However, this step raises the first occurrence of the variable y' beyond its binder and therefore according to our free variable condition we cannot proceed. That is, although this formula contains three operators and therefore $3! = 6$ possible orderings of these operators, one of these is banned by our restriction that raising cannot increase the number of free variables. Furthermore, we have verified that our free variable condition bans the same reading that Hobbs and Shieber ban.

We conclude this section by examining the differences between APL and $\mathcal{R}$. These differences are:

1. In $\mathcal{R}$ each occurrence of the '$\Box$' symbol is immediately proceeded by a variable; by contrast, the symbol '$\Box$' is used in APL to replace entire subformulas.

2. In $\mathcal{R}$ we cannot raise variables past their binders; however, there is no analogous constraint in APL.

3. The syntax of APL permits unordered lists of contexts (ordered sequences of operators); this unordered list of sequences cannot be replicated in $\mathcal{R}$.

We discuss each of these differences in turn:

1. The reason for attaching variables to operators in $\mathcal{R}$ is to ensure that when an operator is raised it leaves behind the correct 'trace'. In APL, we must construct our formulas so that this information is encoded explicitly. Therefore, the treatment of variables in $\mathcal{R}$ is a little more sophisticated than it is in APL, though nothing really important hinges upon this difference.

2. Since the free variable constraint in $\mathcal{R}$ is essentially a meta–level constraint it can just as easily be used to restrict disambiguations in APL. However, we note in passing that the absence of such a restriction makes it far more difficult to represent sentences like 'every artist who admires a bee–keeper despises a carpenter' in APL compared with $\mathcal{R}$.

3. Because we can express the 'cigarette machine' structure in APL but not in $\mathcal{R}$ we believe that APL is probably more expressive than $\mathcal{R}$ and therefore that APL probably subsumes $\mathcal{R}$.

This concludes our presentation of the language $\mathcal{R}$.

In this section we have presented our rational reconstruction of Storage and QLF, namely: $\mathcal{R}$. The language $\mathcal{R}$ is generalised by APL. Therefore, in the sequel, we concern ourselves only with APL.

3.4 Summary

There are eight relatively well–known (in the sub–culture of Computational Linguistics) Underspecified Representation Languages: A Logical Connective for Ambiguity (van Eijck and Jaspars, 1996, pp.5–11), Predicate Logic Unplugged (Bos, 1995), Minimal Recursion Semantics (Copestake et al., 1995), the Constraint Language for Lambda Structures (Egg et al., 1998), Underspecified Discourse Representation Theory (Reyle, 1993), Ambiguous Predicate Logic (van Eijck and Jaspars, 1996, pp.11–19), Storage (Cooper, 1983 *and* Keller, 1986) and

Quasi–Logical Form (Alshawi, 1992). Until now, little has been known about the relationships between these languages. In this chapter we have begun to address this issue, by reducing the eight languages into just three types, namely : $\mathcal{Q}$, $\mathcal{H}$ and $\mathcal{R}$. Furthermore, the syntax and/or disambiguation procedure of some of these languages (in fact, at least three) had, until now, been poorly defined. For all three types of Underspecified Representation Language we have specified a precise syntax and disambiguation procedure as well as making some interesting observations along the way. Now that we have presented a complete account of state–of–the–art Underspecified Representation Languages, the natural question to ask is: What use are they? In particular, we are interested in how we should interpret (and reason over) Underspecified Representations. We devote the next chapter to an investigation of this issue.

Chapter 4

Interpretation

Police Help Dog Bite Victim!

In the previous chapter we reported on current Underspecified Representation Languages: $\mathcal{Q}$, $\mathcal{H}$ and APL (or, more correctly: $\mathcal{Q}$, PLU, MRS, CLLS, UDRT, APL, Storage and QLF). We will now investigate the various schemes for reasoning over these Underspecified Representation Languages. That is, in this chapter we will evaluate current definitions of ambiguous satisfiability and ambiguous logical consequence. Throughout the chapter, a semantic framework is said to be a *recursive satisfaction definition* if it is such that the interpretation of any non–atomic formula is defined only in terms of the interpretations of its immediate subformulas. The best–known current recursive satisfaction definition is Partial Logic. We also evaluate non–recursive satisfaction definitions. We call a semantic framework a *non–recursive satisfaction definition* if it is such that the meaning of any Underspecified Representation is given by the meanings of its set of readings. There are currently two non–recursive satisfaction definitions, with the notions of strong and weak satisfiability as their basis, respectively.

Our reason for presenting the various semantic frameworks for underspecification separately from our presentation of the current Underspecified Representation Languages is that we believe that these components (that is, representation and interpretation) are orthogonal. We will therefore discuss the current semantic frameworks in terms of the (mathematically) simplest Underspecified Representation Language, $\mathcal{Q}$.

4.1 Partial Logic

One suggestion that has been made for understanding the semantics of Underspecified Representations involves a three–valued logic system which is reminiscent of Blamey's Partial Logic (Blamey, 1986). In fact, we will show that the Ambiguous Logic in question (van Eijck and Jaspars, 1996, pp.5–11) is the same as Partial Logic, despite the fact that its authors claim otherwise (Jaspars, 1997, p.7). Establishing the equivalence of these two logics will allow us to transport many of the results obtained with respect to Partial Logic into Ambiguous Logic. Of particular interest to us is the observation that Partial Logic (and hence also the Ambiguous Logic) is not as much of a departure from classical logic as we might expect; specifically, there is no real difference between partial satisfiability and classical satisfiability. Furthermore, we will argue that van Eijck and Jaspars' Ambiguous Logic is of little use anyway, since their framework fails to capture the intuition which they claim it does.

4.1.1 Partial Logic and Ambiguous Logic

In this section we will make a formal comparison between Partial Logic and the Ambiguous Logic presented by van Eijck and Jaspars.

The Ambiguous Logic

We begin by describing the semantic framework presented by van Eijck and Jaspars (van Eijck and Jaspars, 1996, pp.5–11); in this section, we will generally refer to this framework as 'the Ambiguous Logic'. The structures used to interpret the Ambiguous Logic are exactly the same as those used in Classical Logic. That is, a structure of the Ambiguous Logic consists of:

1. A non–empty set $|\mathfrak{A}|$ called the *domain* or *universe*.

2. For each n–ary predicate R of the language there is an n–ary relation $R^{\mathfrak{A}} \subseteq |\mathfrak{A}|^n$; that is, $R^{\mathfrak{A}}$ is a set of n–tuples whose elements are members of the universe.

3. For each n–ary function f of the language there is an n–ary function on $|\mathfrak{A}|$, $f^{\mathfrak{A}} : |\mathfrak{A}|^n \longrightarrow |\mathfrak{A}|$.

4. For each constant c of the language there is an element of the universe $c^{\mathfrak{A}} \in |\mathfrak{A}|$.

We now specify how formulas are to be interpreted in the Ambiguous Logic; that is, we reproduce exactly the semantic framework proposed by van Eijck and Jaspars (van Eijck and Jaspars, 1996, p.5).

Let $\mathfrak{A}$ be an ordinary first–order model, and let g denote a variable assignment which maps object–level variables to elements of the domain of $\mathfrak{A}$. We define two relation symbols '$\models$' and '$=\!\!|$' as the smallest relations satisfying:

$\mathfrak{A}, g \not\models \bot,$
$\mathfrak{A}, g =\!\!| \bot,$

$\mathfrak{A}, g \models R(t_1, \ldots, t_n)$ if $\mathfrak{A} \models_{cl} R(t_1, \ldots, t_n),$
$\mathfrak{A}, g =\!\!| R(t_1, \ldots, t_n)$ if $\mathfrak{A}, g \not\models R(t_1, \ldots, t_n).$

(The subscript cl indicates that we are referring to the usual classical notion of satisfiability). Non–atomic formulas are interpreted inductively as follows:

$\mathfrak{A}, g \models \neg\varphi$ if $\mathfrak{A}, g =\!\!| \varphi,$
$\mathfrak{A}, g =\!\!| \neg\varphi$ if $\mathfrak{A}, g \models \varphi,$

$\mathfrak{A}, g \models (\varphi_1 \wedge \varphi_2)$ if $\mathfrak{A}, g \models \varphi_1$ and $\mathfrak{A}, g \models \varphi_2,$
$\mathfrak{A}, g =\!\!| (\varphi_1 \wedge \varphi_2)$ if $\mathfrak{A}, g =\!\!| \varphi_1$ or $\mathfrak{A}, g =\!\!| \varphi_2,$

$\mathfrak{A}, g \models (\varphi_1 \,?\, \varphi_2)$ if $\mathfrak{A}, g \models \varphi_1$ and $\mathfrak{A}, g \models \varphi_2,$
$\mathfrak{A}, g =\!\!| (\varphi_1 \,?\, \varphi_2)$ if $\mathfrak{A}, g =\!\!| \varphi_1$ and $\mathfrak{A}, g =\!\!| \varphi_2.$

The set of connectives $\{\bot, \neg, \wedge, ?\}$ will be used throughout; implication and disjunction are defined in the usual way (in terms of negation and conjunction). The interpretation of quantified formulas is the same as it is in Classical Logic:

$\mathfrak{A}, g \models \forall x\varphi$ if $\mathfrak{A}, h \models \varphi$ for all h such that $h(y) = g(y)$ whenever $y \neq x,$
$\mathfrak{A}, g =\!\!| \forall x\varphi$ if $\mathfrak{A}, h =\!\!| \varphi$ for some h such that $h(y) = g(y)$ whenever $y \neq x.$

But why do we claim that this system is three–valued logic? Very roughly, any propositional formula is assigned the value $\top$ if all of its disambiguations are true, $\bot$ if all of its disambiguations are false, and $*$ otherwise. That is, a propositional formula is assigned the value $*$ if and only if the truth–values of its disambiguations are not all the same. Jaspars makes this idea formal by introducing the

following abbreviations (Jaspars, 1997, pp.3–4):

$$\begin{aligned} \bot &\equiv (p \wedge \neg p), \\ \top &\equiv \neg(p \wedge \neg p), \\ * &\equiv (p \,?\, \neg p). \end{aligned}$$

where p denotes a propositional variable. Jaspars observes that:

> It is easy to see how partiality is induced by the ambiguity operator: take $V(p) = 1$ and $V(q) = 0$ then $V \not\models (p \,?\, q)$ and $V \nvDash (p \,?\, q)$. (Jaspars, 1997, p.5)

Jaspars also states that the connectives of the Ambiguous Logic may be interpreted as three–valued Boolean functions with truth–values *true*, *false* and *undefined* (Jaspars, 1997, p.7). The corresponding functions are written as $[\neg]$, $[\wedge]$ and $[\,?\,]$.

$$[\neg](t) = \begin{cases} \text{false} & \text{if} \quad t = \text{true}, \\ \text{true} & \text{if} \quad t = \text{false}, \\ \text{undefined} & \text{if} \quad t = \text{undefined}. \end{cases}$$

$$[\wedge](t_1, t_2) = \begin{cases} \text{true} & \text{if} & t_1 = t_2 = \text{true}, \\ \text{false} & \text{if} & t_1 \text{ or } t_2 = \text{false}, \\ \text{undefined} & \text{otherwise}. & \end{cases}$$

$$[\,?\,](t_1, t_2) = \begin{cases} \text{true} & \text{if} & t_1 = t_2 = \text{true}, \\ \text{false} & \text{if} & t_1 = t_2 = \text{false}, \\ \text{undefined} & \text{otherwise}. & \end{cases}$$

We abbreviate (and extend) this idea (to include quantified formulae) by defining a function $\mathfrak{A}_g : \mathcal{Q} \to \{\top, *, \bot\}$ as follows:

$$\mathfrak{A}_g(\varphi) = \begin{cases} \top & \text{if } \mathfrak{A}, g \models \varphi, \\ \bot & \text{if } \mathfrak{A}, g \nvDash \varphi, \\ * & \text{otherwise}. \end{cases}$$

The earlier definitions of the two turn–style relations can now be written in terms of truth–tables (at least for the propositional fragment of $\mathcal{Q}$) as presented in

φ	ψ	$\neg\varphi$	$(\varphi \wedge \psi)$	$(\varphi \vee \psi)$	$(\varphi \,?\, \psi)$
$\top$	$\top$	$\bot$	$\top$	$\top$	$\top$
$\top$	$*$	$\bot$	$*$	$\top$	$*$
$\top$	$\bot$	$\bot$	$\bot$	$\top$	$*$
$*$	$\top$	$*$	$*$	$\top$	$*$
$*$	$*$	$*$	$*$	$*$	$*$
$*$	$\bot$	$*$	$\bot$	$*$	$*$
$\bot$	$\top$	$\top$	$\bot$	$\top$	$*$
$\bot$	$*$	$\top$	$\bot$	$*$	$*$
$\bot$	$\bot$	$\top$	$\bot$	$\bot$	$\bot$

Figure 4.1: Truth–tables of Van Eijck and Jaspars' Ambiguous Logic.

figure 4.1. Finally, we report on van Eijck and Jaspars' notion of (ambiguous) logical implication:

Definition 4.1.1 (Logical Implication). If $\Gamma,\Delta \subseteq \mathcal{Q}$ then, $\Gamma \models\!\!\!=_{\text{vEJ}} \Delta$ if for all pairs $(\mathfrak{A}, g)$ with $\mathfrak{A}$ a model and g a variable assignment:

if $\mathfrak{A}, g \models \gamma$ for all $\gamma \in \Gamma$, then $\mathfrak{A}, g \models \delta$ for some $\delta \in \Delta$ and,
if $\mathfrak{A}, g =\!\!| \delta$ for all $\delta \in \Delta$, then $\mathfrak{A}, g =\!\!| \gamma$ for some $\gamma \in \Gamma$.

Van Eijck and Jaspars attribute this double–barrelled consequence relation to Blamey's Partial Logic (van Eijck and Jaspars, 1996, p.5).[1] In fact, Blamey defines logical implication in terms of *rejecting* models (Blamey, 1986, p.58). However, it is trivial to show that, for any $\Gamma, \Delta \in \mathcal{Q}$ the pair (Γ, Δ) is *rejected* by all (model/assignment) pairs $(\mathfrak{A}, g)$ if and only if $\Gamma \models\!\!\!=_{\text{vEJ}} \Delta$. In the sequel, we will interchange between the double turn–style notation and the term 'rejecting'. This completes our presentation of the Ambiguous Logic, we now turn our attention to a similar scheme – Partial Logic.

A Comparison between Ambiguous Logic and Partial Logic

Partial Logic was developed to model situations in which sentences are not exclusively either 'true' or 'false'; that is, it is proposed that a *truth–value gap* exists.[2] More precisely, any sentence of Partial Logic may be either 'true' ($\top$), 'false' ($\bot$)

[1] Blamey defines a notion of logical implication which is essentially identical to that above. We shall see that this definition is also closely related to monotonicity in Partial Logic.

[2] Partial Logic has been applied to a number of phenomena including: presupposition, conditional assertion, sortal incorrectness and semantic paradox.

or 'neither–true–nor–false' (∗). Blamey uses the apparatus of three–valued logic to discuss Partial Logic in the natural way. However, not all researchers support such a view; for example, van Fraassen asserts that:

> [It has become] the custom of logicians to treat 'is neither–true–nor–false' on par with 'has a third value which is neither *True* nor *False*'. My thesis will be that this supposition is not quite correct. (van Fraassen, 1966, p.67)

However, Partial Logic is *essentially three–valued* according to the criterion described by Simons (Simons, 1999, p.86). We adopt the view that although truth–value gaps and third truth–values are not (philosophically) equal in status, we may treat them as such for practical purposes. This view is shared by Langholm (Langholm, 1996), among others. Therefore, in the sequel, we treat Partial Logic as being three–valued without any further discussion.

The truth–tables for Partial Logic (Blamey, 1986, p.7) are identical to those we presented in figure 4.1, except that the column heading '$\varphi \,?\, \psi$' is replaced by '$\varphi \between \psi$'; however, the truth–values of $\varphi \between \psi$ are the same as those of $\varphi \,?\, \psi$. The treatment of quantified formulas in Partial Logic is also the same as it is in the Ambiguous Logic. Furthermore, the syntax of Partial Logic is very similar to the syntax of $\mathcal{Q}$. In fact, we may identify any sentence of Partial Logic with a sentence in $\mathcal{Q}$ (and visa versa) by defining a bijection which maps all occurrences of the Partial Logic symbol '$\between$' (*interjunction*) to the symbol '?' (*ambiguation*) of $\mathcal{Q}$, leaving everything else unchanged. Blamey uses additional (Partial Logic) connectives such as *transplication* (/), but all of these may be defined in terms of $\top$, $\bot$, $\neg$, $\wedge$ and $\between$; hence omitting them does not reduce expressive power. The models used in the Ambiguous Logic are exactly those of Classical Logic (with the difference that in Ambiguous Logic formulas can be '∗'); by contrast, 'empty models' are permitted in Partial Logic. Furthermore, in Partial Logic structures may include a non–denoting symbol $\otimes$ in their domains. We do not incorporate the symbol $\otimes$ in our analysis, nor do we consider models with empty domains.[3]

Ignoring these minor details, the obvious question to ask now is: Is the semantics of van Eijck and Jaspars' Ambiguous Logic the same as the semantics of Partial Logic? Jaspars claims that it is not; in a footnote Jaspars asserts that:

[3]Van Orman Quine discusses the implications of permitting empty domains (Van Orman Quine, 1954).

> The clear difference [between Partial Logic and Ambiguous Logic] is that in the case of Partial Logic, propositional variables can be undefined as well, while we [in Ambiguous Logic] enforce bivalence for variables. (Jaspars, 1997, p.7)

That is, atomic propositions may be assigned the value $*$ in Partial Logic but in Ambiguous Logic they may not. However, we shall see that this distinction is less significant than we might expect. We use the following definition to simplify the discussion:

Definition 4.1.2 (Decisive Model). A model $\mathfrak{A}$ is *decisive* with respect to assignment g if, under assignment g, $\mathfrak{A}$ assigns every atomic formula either $\top$ or $\bot$.

In the sequel, we discuss both 'models' (models of Partial Logic) and 'decisive models' (models of Ambiguous Logic). Note that, by definition, any decisive model is also a model.

Definition 4.1.3 (Satisfiable). A formula $\varphi \in \mathcal{Q}$ is *satisfiable* if there exists some model $\mathfrak{A}$ and some assignment function g such that $\mathfrak{A}, g \models \varphi$.

Definition 4.1.4 (Decisively Satisfiable). A formula $\varphi \in \mathcal{Q}$ is *decisively satisfiable* if there exists some decisive model $\mathfrak{A}$ with respect to some assignment g such that $\mathfrak{A}, g \models \varphi$.

We will now make a formal comparison between satisfiability and decisive satisfiability. In doing so, we will exploit the following result which involves an important theme in Partial Logic – monotonicity (Blamey, 1986, Sec.6·2, pp.52–55):

Lemma 4.1.5. *Monotonicity is preserved by the formulas of Partial Logic. That is, for any $\varphi \in \mathcal{Q}$, for models $\mathfrak{A}$ and $\mathfrak{B}$ and variable assignment g, if the atoms occurring in φ are precisely $\theta_1, \ldots, \theta_n$ then the following statement holds: If $\mathfrak{A}_g(\theta_i) = \top \Rightarrow \mathfrak{B}_g(\theta_i) = \top$ and $\mathfrak{A}_g(\theta_i) = \bot \Rightarrow \mathfrak{B}_g(\theta_i) = \bot$ for $1 \leqslant i \leqslant n$ then $\mathfrak{A}_g(\varphi) = \top \Rightarrow \mathfrak{B}_g(\varphi) = \top$ and $\mathfrak{A}_g(\varphi) = \bot \Rightarrow \mathfrak{B}_g(\varphi) = \bot$.*

Proof of Lemma 4.1.5. Routine structural induction on formulas. □

We are now in a position to spell–out the relationship between satisfiability and decisive satisfiability.

Lemma 4.1.6. *For any $\Gamma \subseteq \mathcal{Q}$, Γ is satisfiable if and only if Γ is decisively satisfiable.*

Proof of Lemma 4.1.6. Suppose Γ is satisfied by $\mathfrak{A}$ with respect to assignment g; that is, $\mathfrak{A}, g \models \varphi$ for all $\varphi \in \Gamma$. Let θ range over the atoms which occur in Γ. Define a decisive model $\mathfrak{B}$ with respect to assignment g by:

$$\mathfrak{B}_g(\theta) = \begin{cases} \top & \text{if } \mathfrak{A}_g(\theta) = \top, \\ \bot & \text{otherwise.} \end{cases}$$

It is immediate (from lemma 4.1.5) that $\mathfrak{B}, g \models \Gamma$. The converse is trivial, since any decisive model is a model. □

Lemma 4.1.6 establishes that in $\mathcal{Q}$, if we have a procedure for establishing satisfiability in Ambiguous Logic, then we automatically have a decision procedure for Partial Logic (and visa versa). We might now ask: is the notion of logical implication in the Ambiguous Logic the same as it is in Partial Logic? Before we can answer this question we must introduce some terminology and notation.

Blamey defines the term 'rejects' as follows: a model/assignment pair $(\mathfrak{A}, g)$ *rejects* a pair (Γ, Δ) (where Γ and Δ are sets of sentences) if either $\mathfrak{A}_g(\gamma) = \top$ for all $\gamma \in \Gamma$ and $\mathfrak{A}_g(\delta) \neq \top$ for all $\delta \in \Delta$, or $\mathfrak{A}_g(\gamma) \neq \bot$ for all $\gamma \in \Gamma$ and $\mathfrak{A}_g(\delta) = \bot$ for all $\delta \in \Delta$. We use the notation $\models_{\text{Dec}}$ to represent a relation defined exactly as $\models$, except that, $\models_{\text{Dec}}$ denotes a relation between *decisive* models (equipped with assignment functions) and formulas; we note in passing that $\models_{\text{Dec}} \subseteq \models$. In both Partial Logic and Ambiguous Logic, logical implication is defined as: $\Gamma \models_{\text{PL}} \Delta$ if and only if (Γ, Δ) has *no* rejecting model/assignment pair $(\mathfrak{A}, g)$. The question is then: is decisive logical implication really different (to Blamey's logical implication)? To answer this question we need only to consider the following simple example: Let θ be an atom, then the pair $(\{\theta \wedge \neg\theta\}, \{\bot\})$ has no decisive rejecting models, that is: $\{\theta \wedge \neg\theta\} \models_{\text{Dec}} \bot$. But $(\{\theta \wedge \neg\theta\}, \{\bot\})$ has a (non–decisive) rejecting structure defined by $\mathfrak{A}_g(\theta) = *$. Clearly this observation serves as a counter–example to the hope that for any $\Gamma, \Delta \subseteq \mathcal{Q}$, the pair (Γ, Δ) is rejected in a model if and only if (Γ, Δ) is rejected in a decisive model. However, we do have the following interesting lemmas:

Lemma 4.1.7. *Let $\Gamma, \Delta \subseteq \mathcal{Q}$. Assume without loss of generality that the n–ary predicate symbols occurring in $\Gamma \cup \Delta$ are among $\{p_i\}_{1 \leqslant i}$, and that $\{q_i\}_{1 \leqslant i}$ and*

$\{r_i\}_{1 \leqslant i}$ are sets of n–ary predicates such that all three sets of predicate symbols are (pairwise) disjoint. Define a map $\dagger : \mathcal{Q} \longrightarrow \mathcal{Q}$ as follows:

$$\begin{aligned}
(p_i(x_1, \ldots, x_n))^\dagger &= q_i(x_1, \ldots, x_n) \,?\, r_i(x_1, \ldots, x_n), \\
(\neg\varphi)^\dagger &= \neg\,(\varphi^\dagger), \\
(\varphi \wedge \psi)^\dagger &= \varphi^\dagger \wedge \psi^\dagger, \\
(\varphi \,?\, \psi)^\dagger &= \varphi^\dagger \,?\, \psi^\dagger, \\
(\forall x \varphi)^\dagger &= \forall x (\varphi^\dagger).
\end{aligned}$$

Then (Γ, Δ) is rejected if and only if $(\Gamma^\dagger, \Delta^\dagger)$ is rejected by a decisive model, where $\Gamma^\dagger$ denotes the result of applying the map $\dagger$ to all of the sentences in Γ.

Proof of Lemma 4.1.7. Suppose $(\mathfrak{A}, g)$ rejects (Γ, Δ). Define a decisive model $\mathfrak{B}$ with respect to assignment g as: For any tuple $(a_1, \ldots, a_n) \in |\mathfrak{A}|^n$,

$$\begin{array}{llllll}
\text{if} & p_i^{\mathfrak{A}}(a_1, \ldots, a_n) = 0 & \text{then} & q_i^{\mathfrak{B}}(a_1, \ldots, a_n) = 0 & \text{and} & r_i^{\mathfrak{B}}(a_1, \ldots, a_n) = 0, \\
\text{if} & p_i^{\mathfrak{A}}(a_1, \ldots, a_n) = 1 & \text{then} & q_i^{\mathfrak{B}}(a_1, \ldots, a_n) = 1 & \text{and} & r_i^{\mathfrak{B}}(a_1, \ldots, a_n) = 1, \\
\text{if} & p_i^{\mathfrak{A}}(a_1, \ldots, a_n) = * & \text{then} & q_i^{\mathfrak{B}}(a_1, \ldots, a_n) = 0 & \text{and} & r_i^{\mathfrak{B}}(a_1, \ldots, a_n) = 1.
\end{array}$$

By induction, for any non–atomic formula $\varphi \in \mathcal{Q}$ we have $\varphi^{\mathfrak{A},g} = \varphi^{\mathfrak{B},g}$ where g is any assignment function. Conversely, suppose that $\mathfrak{B}$ is a decisive model which rejects $(\Gamma^\dagger, \Delta^\dagger)$. We define a model $\mathfrak{A}$ with respect to assignment g as:

$$\begin{array}{lll}
p_i^{\mathfrak{A}}(a_1, \ldots, a_n) = 0 & \text{iff} & q_i^{\mathfrak{B}}(a_1, \ldots, a_n) = 0 \text{ and } r_i^{\mathfrak{B}}(a_1, \ldots, a_n) = 0, \\
p_i^{\mathfrak{A}}(a_1, \ldots, a_n) = 1 & \text{iff} & q_i^{\mathfrak{B}}(a_1, \ldots, a_n) = 1 \text{ and } r_i^{\mathfrak{B}}(a_1, \ldots, a_n) = 1, \\
p_i^{\mathfrak{A}}(a_1, \ldots, a_n) = * & \text{iff} & q_i^{\mathfrak{B}}(a_1, \ldots, a_n) = 0 \neq r_i^{\mathfrak{B}}(a_1, \ldots, a_n) = 1.
\end{array}$$

The remainder of the proof is a simple induction over all $\varphi \in \mathcal{Q}$ as before. □

Corollary 4.1.8. Let $\Gamma, \Delta \subseteq \mathcal{Q}$. Then $\Gamma \models \Delta$ if and only if $\Gamma^\dagger \models_{\text{Dec}} \Delta^\dagger$, where $\models$ is the usual double–barrelled consequence relation over all models and $\models_{\text{Dec}}$ is the double–barrelled consequence relation over all decisive models.

Proof of Corollary 4.1.8. Immediate from lemma 4.1.7 and from the definition of $\models_{\text{Dec}}$. □

Corollary 4.1.8 shows how to reduce (general) logical implication to logical implication in a decisive setting. We now establish a reduction in the opposite direction, that is, from logical implication over decisive models to (general) logical implication.

Lemma 4.1.9. *Let θ_i $(1 \leqslant i \leqslant n)$ range over the atoms which occur in $\Gamma \cup \Delta$ where $\Gamma, \Delta \subseteq \mathcal{Q}$ are finite. Define ω to be the sentence of $\mathcal{L}$ given by,*

$$\omega = \bigwedge_{1 \leqslant i \leqslant n} (\theta_i \vee \neg\theta_i).$$

Then $\Gamma \models_{\text{Dec}} \Delta$ if and only if $(\omega \wedge \bigwedge \Gamma) \models (\omega \rightarrow \bigwedge \Delta)$.

Proof of Lemma 4.1.9. Note that, if $\mathfrak{A}$ is a model of $\mathcal{Q}$ and g is an assignment function then $\mathfrak{A}_g(\omega) = \top$ if and only if $\mathfrak{A}$ is decisive (on the atoms $\theta_1, \ldots \theta_n$) with respect to assignment g, otherwise $\mathfrak{A}_g(\omega) = *$. We now present a (contrapositive) proof: Suppose $\mathfrak{A}_g(\omega \wedge \bigwedge \Gamma) = \top$ and $\mathfrak{A}_g(\omega \rightarrow \bigwedge \Delta) \neq \top$. Given $\mathfrak{A}_g(\omega \wedge \bigwedge \Gamma) = \top$ truth–tables imply that $\mathfrak{A}_g(\omega) = \top$ and $\mathfrak{A}_g(\bigwedge \Gamma) = \top$. Because $\mathfrak{A}_g(\omega) = \top$, $\mathfrak{A}$ is a decisive model with respect to g. Also, by assumption $\mathfrak{A}_g(\omega \rightarrow \bigwedge \Delta) \neq \top$ and given that we have $\mathfrak{A}_g(\omega) = \top$ we may use truth–tables to infer that $\mathfrak{A}_g(\bigwedge \Delta) \neq \top$. We now have $\mathfrak{A}_g(\bigwedge \Gamma) = \top$ and $\mathfrak{A}_g(\bigwedge \Delta) \neq \top$; observation of the truth–table for conjunction implies that $\mathfrak{A}_g(\gamma) = \top$ for all $\gamma \in \Gamma$, and that it is *not* the case that $\mathfrak{A}_g(\delta) = \top$ for all $\delta \in \Delta$. Hence $\mathfrak{A}$ is a decisive model with respect to g and the pair $(\mathfrak{A}, g)$ rejects (Γ, Δ), therefore $\Gamma \not\models_{\text{Dec}} \Delta$. Alternatively, suppose that $\mathfrak{A}_g(\omega \rightarrow \bigwedge \Delta) = \bot$ and $\mathfrak{A}_g(\omega \wedge \bigwedge \Gamma) \neq \bot$. By truth–tables $\mathfrak{A}_g(\omega \rightarrow \bigwedge \Delta) = \bot$ implies that $\mathfrak{A}_g(\omega) = \top$ and $\mathfrak{A}_g(\bigwedge \Delta) = \bot$. Thus $\mathfrak{A}$ is decisive with respect to g. Truth–tables show that if a conjunction is non–false then both conjuncts must be non–false. Since $\mathfrak{A}_g(\omega \wedge \bigwedge \Gamma) \neq \bot$ we have $\mathfrak{A}_g(\bigwedge \Gamma) \neq \bot$. So that $\mathfrak{A}_g(\bigwedge \Delta) = \bot$ and $\mathfrak{A}_g(\bigwedge \Gamma) \neq \bot$, and therefore $\Gamma \not\models_{\text{Dec}} \Delta$.

Conversely, suppose $\mathfrak{A}$ is decisive with respect to g, $\mathfrak{A}_g(\gamma) = \top$ for all $\gamma \in \Gamma$, and that $\mathfrak{A}_g(\delta) \neq \top$ for all $\delta \in \Delta$. Truth–tables yield $\mathfrak{A}_g(\bigwedge \Gamma) = \top$ and $\mathfrak{A}_g(\bigwedge \Delta) \neq \top$. Since $\mathfrak{A}$ is decisive with respect to g, $\mathfrak{A}_g(\omega) = \top$. Truth–tables show that $\mathfrak{A}_g(\omega \wedge \bigwedge \Gamma) = \top$ since both conjuncts are true. Similarly, because $\mathfrak{A}_g(\bigwedge \Delta) \neq \top$ and $\mathfrak{A}_g(\omega) = \top$ truth–tables imply that $\mathfrak{A}_g(\omega \rightarrow \bigwedge \Delta) \neq \top$. Thus $(\mathfrak{A}, g)$ rejects $(\{\omega \wedge \bigwedge \Gamma\}, \{\omega \rightarrow \bigwedge \Delta\})$ and therefore $(\omega \wedge \bigwedge \Gamma) \not\models (\omega \rightarrow \bigwedge \Delta)$. Alternatively, suppose $\mathfrak{A}$ is decisive with respect to g, $\mathfrak{A}_g(\delta) = \bot$ for all $\delta \in \Delta$, and $\mathfrak{A}_g(\gamma) \neq \bot$ for all $\gamma \in \Gamma$. Then $\mathfrak{A}_g(\bigwedge \Delta) = \bot$ and $\mathfrak{A}_g(\bigwedge \Gamma) \neq \bot$. Truth–tables yield $\mathfrak{A}_g(\omega \rightarrow \bigwedge \Delta) = \bot$ and $(\omega \wedge \bigwedge \Gamma) \neq \bot$ in a similar manner to that

above. Hence $(\omega \wedge \bigwedge \Gamma) \not\models (\omega \rightarrow \bigwedge \Delta)$. □

In this section, we have established that the Ambiguous Logic of van Eijck and Jaspars is essentially equivalent (in terms of satisfiability and logical implication) to Partial Logic. We might wonder if there is a deeper reason for this connection. Malinowski notes that 'simple Partial Logic is semantically an extension of the Kleene strong system' (Malinowski, 1993) and, Pinkal suggests that 'Kleene proposes a system ... that comes close[r] to the intuitive concept of semantic (linguistic) indefiniteness' (Pinkal, 1985). So perhaps the reason for the relationship between the Ambiguous Logic and Partial Logic might lie in the fact that we regard sentences involving partiality, the indefinite and ambiguity with a similar amount of suspicion!

Now that we know that the Ambiguous Logic is the same as Partial Logic we can transfer results from Partial Logic to the Ambiguous Logic. The main theorems in Blamey's presentation of Partial Logic are:

Theorem 4.1.10 (Blamey). *Every monotonic three–valued boolean connective can be defined in terms of the connectives in the set* $\{\bot, \neg, \wedge, ?\}$.

Theorem 4.1.11 (Blamey). *Let* $\Gamma, \Delta, \Sigma \subseteq \mathcal{Q}$. *Then there exists a model of* Σ *which rejects* (Γ, Δ) *if and only if for every finite subset* Γ_0 *of* Γ, Δ_0 *of* Δ *and* Σ_0 *of* Σ *there is a model of* Σ_0 *which rejects* (Γ_0, Δ_0).

Now that we have made clear the relationship between the Ambiguous Logic and Partial Logic, we ask: How sensible is Partial Logic (and therefore the Ambiguous Logic)? Blamey claims that he provides

> ... two particular accounts of how the triclassificatory semantics of Partial Logic can play a role which does *not*, in any interesting sense, give rise to an alternative to Classical Logic. (Blamey, 1986, p.44)

However, Blamey's discussion is quite philosophical in style, and unfortunately, lacks mathematical clarity. Therefore, we conclude this section, by making a concrete comparison between Partial Logic and Classical Logic.[4]

[4]This issue is explored in some detail by Langholm (Langholm, 1996, pp.32–41).

4.1.2 Partial Logic and Classical Logic

In this section we will investigate the relationship between Partial Logic and Classical Logic; in particular, we will compare partial satisfiability with classical satisfiability. We begin by defining three maps; one of which moves negations inwards in the standard way, the other two simply replace all occurrences of '?' by conjunction and disjunction respectively.

Definition 4.1.12 (Maps N, C & D). We define maps $N, C, D : \mathcal{Q} \to \mathcal{Q}$ as follows:

$$\begin{array}{lcl} \varphi^N & = & \text{for atomic } \varphi, \\ (\varphi \circ \psi)^N & = & \varphi^N \circ \psi^N \text{ for any binary connective } \circ \text{ of } \mathcal{Q}, \\ (Qx\varphi)^N & = & Qx\varphi^N \text{ for any quantifier } Q \text{ and variable } x \text{ of } \mathcal{Q}. \end{array}$$

We define N for formulae prefixed by a negation symbol as follows,

$$\begin{array}{lcl} (\neg\varphi)^N & = & \neg\varphi \text{ if } \varphi \text{ is a literal}, \\ (\neg(\neg\varphi))^N & = & \varphi^N, \\ (\neg(\varphi \wedge \psi))^N & = & (\neg\varphi)^N \vee (\neg\psi)^N, \\ (\neg(\varphi \vee \psi))^N & = & (\neg\varphi)^N \wedge (\neg\psi)^N, \\ (\neg(\varphi \,?\, \psi))^N & = & (\neg\varphi)^N \,?\, (\neg\psi)^N, \\ (\neg\forall x\varphi)^N & = & \exists x((\neg\varphi)^N), \\ (\neg\exists x\varphi)^N & = & \forall x((\neg\varphi)^N). \end{array}$$

For any $\varphi \in \mathcal{Q}$ we denote by φ^C the result of replacing all occurrences of '?' in φ with '$\wedge$'. Similarly φ^D denotes the result of replacing all occurrences of '?' in φ with '$\vee$'.

Lemma 4.1.13. *Let $\mathfrak{A}$ be a partial model and g be an assignment function. Then for all $\varphi \in \mathcal{Q}$:*

1. *$\mathfrak{A}, g \models \varphi$ if and only if $\mathfrak{A}, g \models' (\varphi^N)^C$ and*
2. *$\mathfrak{A}, g =\!| \varphi$ if and only if $\mathfrak{A}, g \models' \neg(\varphi^N)^D$*

where $\models$ and $=\!|$ denote the Partial Logic relations (defined in 4.1.1, page 99) and $\models'$ denotes the usual classical relation "$\models$".

Proof of Lemma 4.1.13. We prove this lemma in two steps:
Step 1: We show by induction on the complexity of formulae that for all $\varphi \in \mathcal{Q}$,

$\mathfrak{A}, g \models \varphi$ *if and only if* $\mathfrak{A}, g \models \varphi^N$ *(and* $\mathfrak{A}, g =\!| \varphi$ *if and only if* $\mathfrak{A}, g =\!| \varphi^N$*).* To prove Step 1, we need only to consider formulas which are prefixed by a negation symbol (since for those formulas which are not prefixed by a negation symbol, Step 1 is trivial). We proceed by induction over formulas; though, for brevity, we present just two of the cases. Suppose that $\mathfrak{A}, g \models \neg(\varphi \,?\, \psi)$. By looking at truth–tables we know that in Partial Logic $\neg(\varphi \,?\, \psi) \equiv \neg\varphi \,?\, \neg\psi$ and therefore $\mathfrak{A}, g \models \neg\varphi \,?\, \neg\psi$. Hence, by definition of the map N, we have $\mathfrak{A}, g \models (\neg(\varphi \,?\, \psi))^N$. Similarly, suppose that $\mathfrak{A}, g \models \neg\forall x\varphi$. Then $\mathfrak{A}, g =\!| \forall x\varphi$ by definition of the turn–style relations. Again, by definition of the turn–style relations we have $\mathfrak{A}, g =\!| \forall x\varphi$ if and only if $\mathfrak{A}, h =\!| \varphi$ for some h such that $h(y) = g(y)$ whenever $y \neq x$. By inductive hypothesis we know that $\mathfrak{A}, h =\!| \varphi^N$ for some h such that $h(y) = g(y)$ whenever $y \neq x$. Once again, by definition of the turn–style relations $\mathfrak{A}, g \models \neg\varphi^N$ for some h such that $h(y) = g(y)$ whenever $y \neq x$ and therefore $\mathfrak{A}, g \models \exists x((\neg\varphi)^N)$. The other cases are similar.

Step 2: For all $\varphi \in \mathcal{Q}$ *such that* φ *is in negation normal form (that is,* N *has been applied),* $\mathfrak{A}, g \models \varphi$ *if and only if* $\mathfrak{A}, g \models' \varphi^C$ *(and* $\mathfrak{A}, g =\!| \varphi$ *if and only if* $\mathfrak{A}, g \models' \neg\varphi^D$*).* As with Step 1, we will prove Step 2 by induction on the complexity of formulae. Since φ is in negation normal form the base case of our (inductive) proof is 'φ is literal'. The base case is trivial anyway (since C and D leave literals unchanged). Furthermore, we can omit negated formulae from the inductive steps. Again, for brevity, we consider just two inductive cases. Suppose that $\mathfrak{A}, g \models \varphi \,?\, \psi$. Then $\mathfrak{A}, g \models \varphi$ and $\mathfrak{A}, g \models \psi$ by definition of the Partial Logic relation $\models$. By inductive hypothesis we have $\mathfrak{A}, g \models' \varphi^C$ and $\mathfrak{A}, g \models' \psi^C$ and therefore $\mathfrak{A}, g \models' \varphi^C \wedge \psi^C$; that is, $\mathfrak{A}, g \models' (\varphi \,?\, \psi)^C$. Similarly, suppose that $\mathfrak{A}, g =\!| \varphi \,?\, \psi$ then $\mathfrak{A}, g =\!| \varphi$ and $\mathfrak{A}, g =\!| \psi$. By inductive hypothesis we have $\mathfrak{A}, g \models' \neg\varphi^D$ and $\mathfrak{A}, g \models' \neg\psi^D$. Therefore $\mathfrak{A}, g \models' \neg\varphi^D \wedge \neg\psi^D$ and, by de Morgan's rule $\mathfrak{A}, g \models' \neg(\varphi^D \vee \psi^D)$. By definition of the map D we have $\mathfrak{A}, g \models' \neg(\varphi \,?\, \psi)^D$ as required. The other cases are similar. □

A 'notational variant' of this result is presented by van Eijck and Jaspars (van Eijck and Jaspars, 1996, p.24, Cor.16).

So far in this chapter we have established that the Ambiguous Logic presented by van Eijck and Jaspars is the same as Partial Logic and that Partial Logic can be understood in terms of Classical Logic anyway. We conclude our study of van Eijck and Jaspars' proposal (Partial Logic) by evaluating its suitability to the ambiguity problem.

4.1.3 Is Partial Logic Suited to the Ambiguity Problem?

In this section we assess the suitability of Partial Logic as a framework for interpreting and reasoning over Underspecified Representations. Very roughly, van Eijck and Jaspars claim that their logic (and in particular the double turn–style notation we presented on page 99) characterises the following intuition: an ambiguous formula is 'true' if and only if all of its readings are (classically) true and 'false' if and only if all of its readings are (classically) false. Van Eijck and Jaspars formalise this claim as follows (van Eijck and Jaspars, 1996, p.5):

- $\mathfrak{A} \models S$, means that every reading of the sentence S is true in model $\mathfrak{A}$,
- $\mathfrak{A} \not\models S$, means that every reading of S is false in $\mathfrak{A}$,
- $\mathfrak{A} \nRightarrow S$, means that at least one reading of S is false in $\mathfrak{A}$,
- $\mathfrak{A} \not\Rightarrow S$, means that at least one reading of S is true in $\mathfrak{A}$.

We will show that, unfortunately, the logic proposed by van Eijck and Jaspars fails to coincide with these glosses. The following lemma establishes the relationship between van Eijck and Jaspars' glosses (page 110) and the formal definition of the double turn–style notation (page 99) for *propositional* formulae.

Lemma 4.1.14. *For any model $\mathfrak{A}$ and any propositional sentence $\varphi \in \mathcal{Q}$:*

1. *$\mathfrak{A} \models \varphi$ if and only if $\mathfrak{A} \models' \widehat{\varphi}$ for all total disambiguations $\widehat{\varphi}$ of φ and,*
2. *$\mathfrak{A} =\!| \varphi$ if and only if $\mathfrak{A} \not\models' \widehat{\varphi}$ for all total disambiguations $\widehat{\varphi}$ of φ,*

where $\models$ and $=\!|$ denote the Partial Logic relations and $\models'$ denotes the usual classical relation "$\models$".

Proof of Lemma 4.1.14. Proceed by induction on the complexity of formulae: The base case is trivial since atomic formulae are unambiguous and thus $\varphi = \widehat{\varphi}$. We present just two inductive steps here, since the omitted cases are similar; we begin with negation. Suppose that $\mathfrak{A} \models \neg\varphi$, then by definition of the Partial Logic turn–style relations $\mathfrak{A} =\!| \varphi$. By inductive hypothesis $\mathfrak{A} \not\models' \widehat{\varphi}$ for all total disambiguations $\widehat{\varphi}$ of φ. Finally, it is clear that for any $\varphi \in \mathcal{Q}$ we have $\neg(\widehat{\varphi}) = \widehat{\neg\varphi}$; that is, the set consisting of the disambiguations of φ prefixed with negation symbols is equal to the set of disambiguations of $\neg\varphi$. Therefore, $\mathfrak{A} \models' \widehat{\neg\varphi}$. Similarly, suppose that $\mathfrak{A} =\!| \varphi_1 \,?\, \varphi_2$. Then by definition of the $=\!|$ relation $\mathfrak{A} =\!| \varphi_1$

and $\mathfrak{A} =| \varphi_2$. By inductive hypothesis we therefore have $\mathfrak{A} \not\models' \widehat{\varphi_1}$ and $\mathfrak{A} \not\models' \widehat{\varphi_2}$ for all total disambiguations $\widehat{\varphi_1}$ of φ_1 and for all total disambiguations $\widehat{\varphi_2}$ of φ_2. By definition of 'disambiguation' in $\mathcal{Q}$ we have $\mathfrak{A} \not\models' \widehat{(\varphi_1 \,?\, \varphi_2)}$ for all total disambiguations $\widehat{(\varphi_1 \,?\, \varphi_2)}$ of $(\varphi_1 \,?\, \varphi_2)$. The other cases are similar. □

Unfortunately, the above lemma does *not* hold for quantified formulae. To prove this claim, we need only to construct a formula $\varphi \in \mathcal{Q}$ such that for some model $\mathfrak{A}$ and assignment function g we have $\mathfrak{A}, g \not\models \varphi$ and $\mathfrak{A}, g \models' \widehat{\varphi}$ for all total disambiguations $\widehat{\varphi}$ of φ. We claim that the formula $\varphi = \exists x[(B_1(x) \,?\, B_2(x)) \wedge M(x)]$ is such a counter–example. Suppose that both of the formulas $\exists x(B_1(x) \wedge M(x))$ and $\exists x(B_2(x) \wedge M(x))$ – that is, *all* of the total disambiguations of φ – are (classically) satisfied by $\mathfrak{A}$, but that '$\exists x$' requires a different witness in each case. Then, according to the semantic framework proposed by van Eijck and Jaspars, $\mathfrak{A} \not\models \varphi$. Intuitively, suppose that $\mathfrak{A}$ interprets B_1 as 'is a financial institution', B_2 as 'is a mud wall' and M as the property 'is mossy'. Then $\mathfrak{A}$ corresponds to a (quite realistic) 'possible world' in which there is at least one mossy financial institution and at least one mossy mud wall, but in which there does not exist a mossy financial institution which is also a mud wall! This counter–example shows that the semantics proposed by van Eijck and Jaspars somehow fails to capture the intuition intended. That is, the intuitive and formal definitions of the double turn–style notation do not completely coincide.

4.1.4 Summary

In this section we have seen that the Ambiguous Logic proposed by van Eijck and Jaspars fails to capture their intended notions of satisfaction (page 111) and that, in terms of satisfiability and logical consequence, their logic is equivalent to Partial Logic (lemma 4.1.6, lemma 4.1.7 and lemma 4.1.9). We have also seen that satisfaction in Partial Logic is reducible to satisfaction in classical logic anyway (lemma 4.1.13). We therefore dismiss this Ambiguous Logic (Partial Logic) and instead concentrate on non–recursive satisfaction definitions.

4.2 Non–Recursive Satisfaction Definitions

We say that a semantic framework is a non–recursive satisfaction definition if it is such that the meaning of any Underspecified Representation is completely

determined by the meanings of its disambiguations. We now evaluate current non–recursive satisfaction definitions.

It is fundamental to this approach that the Underspecified Representation Language, in our case $\mathcal{Q}$, has no semantics of its own beyond the semantics for the unambiguous sub–language $\mathcal{L}$. Thus, we never speak of the truth or falsity of a formula φ in a structure, except where φ is unambiguous. However, we redefine the semantic notions of satisfiability, validity and entailment so that they apply to ambiguous formulas.

4.2.1 Strong and Weak Satisfiability

We begin with satisfiability. In this section we let $\mathcal{A}$ be any Underspecified Representation Language such that $\mathcal{A}$ has a well–defined disambiguation procedure.

Definition 4.2.1 (Satisfiable). Let $\varphi \in \mathcal{A}$. We say that φ is *weakly (strongly) satisfiable* if *some (every)* total disambiguation $\widehat{\varphi}$ of φ is classically satisfiable. Similarly, we say that φ is *weakly (strongly) unsatisfiable* if *some (every)* total disambiguation $\widehat{\varphi}$ of φ is classically unsatisfiable.

Notice that this definition involves quantification over disambiguations, and in some sense quantification over models. For example, $p\,?\,\neg p$ is strongly satisfiable (since the separate occurrences of p may be satisfied by different models) and $\varphi \wedge \neg\varphi$ is not necessarily a contradiction[5], since, for example, if $\varphi = p\,?\,q$ then $p \wedge \neg q$ is a total disambiguation of $\varphi \wedge \neg\varphi$ and $p \wedge \neg q$ is obviously classically satisfiable.

Definition 4.2.2 (Validity & Entailment). Let $\varphi, \psi \in \mathcal{A}$. We say that φ is *weakly (strongly) valid* if, for some (every) disambiguation $\widehat{\varphi}$ of φ, $\widehat{\varphi}$ is classically valid. We say that φ *weakly (strongly) entails* ψ if, for some (every) disambiguation $\widehat{\varphi}$ of φ, and some (every) disambiguation $\widehat{\psi}$ of ψ, $\widehat{\varphi}$ classically entails $\widehat{\psi}$.

Under this view, once we have specified the set of disambiguations of an Underspecified Representation, there is nothing further one needs to do.

Observation 4.2.3. *Let $\varphi, \psi \in \mathcal{A}$. Then φ is weakly (strongly) unsatisfiable if and only if φ is not strongly (weakly) satisfiable; φ is weakly (strongly) valid if and only if $\neg\varphi$ is not strongly (weakly) satisfiable; and φ weakly (strongly) entails ψ if and only if $\varphi \rightarrow \psi$ is weakly (strongly) valid.*

[5] We use the term 'contradiction' here to mean 'strongly unsatisfiable'.

Proof of Observation 4.2.3. Obvious. □

Our notions of strong and weak entailment (first reported by van Deemter) are sometimes denoted '$\models_{\forall\forall}$' and '$\models_{\exists\exists}$' respectively (van Deemter, 1996, p.216). For sure, other possibilites exist, for example van Deemter defines two other relations '$\models_{\forall\exists}$' and '$\models_{\exists\forall}$', but we dismiss these since no analogue of observation 4.2.3 holds. It is clear that non–recursive satisfaction definitions provide a more sensible framework for interpreting Underspecified Representations than Partial Logic. Therefore we will conclude this chapter with a more detailed investigation into non–recursive satisfaction definitions; in particular, we will investigate complexity issues within this setting.

4.2.2 Computational Complexity of Strong and Weak Satisfiability

In this section we will investigate the computational complexity of strong and weak satisfiability. The main result of this section is that strong satisfiability (with respect to $\mathcal{Q}$) can be reduced to ordinary satisfiability. We begin by defining logical equivalence in our semantic framework:

Definition 4.2.4 (Logical Equivalence). Let $\varphi, \psi \in \mathcal{Q}$. We say that φ and ψ are *logically equivalent,* written $\varphi \equiv \psi$, if every disambiguation of φ is logically equivalent to some disambiguation of ψ, and every disambiguation of ψ is logically equivalent to some disambiguation of φ.

In the sequel, it will sometimes be convenient to write formulas in 'negation normal form' as we discussed on page 108. We reproduce the relevant material below:

Definition 4.2.5 (Negation Normal Form). We denote by φ^{N} the result of moving negations inwards using the usual rules for $\mathcal{L}$ together with the additional rule

$$(\neg(\varphi ? \psi))^{\mathrm{N}} = ((\neg\varphi)^{\mathrm{N}} \,?\, (\neg\psi)^{\mathrm{N}}).$$

A formula of the form φ^{N} for $\varphi \in \mathcal{Q}$ is said to be in *negation normal form.*

Example 4.2.6.

$$\neg(\exists x(\neg p(x) ? \neg q(x)) \vee \forall x p(x) \vee \forall x q(x))^{\mathrm{N}} = \forall x(p(x) ? q(x)) \wedge \exists x \neg p(x) \wedge \exists x \neg q(x)$$

Lemma 4.2.7. *Let $\varphi \in \mathcal{Q}$. Then $\varphi \equiv \varphi^{\mathrm{N}}$.*

Proof of Lemma 4.2.7. Follows from the fact that $\neg(\varphi \,?\, \psi) \equiv \neg\varphi \,?\, \neg\psi$. □

Strong Satisfiability

The next lemma contains the main idea in the reduction of strong satisfiability to ordinary satisfiability.

Lemma 4.2.8. *Let φ be a formula in negation normal form containing a subformula α of the form $\psi(\bar{y}) \,?\, \pi(\bar{y})$, where ψ and π are unambiguous and have no free variables other than $\bar{y}$. Let $\varphi[\alpha/p_\alpha(\bar{y}, u)]$ be the result of replacing α in φ by $p_\alpha(\bar{y}, u)$ where p_α is a new predicate letter (of appropriate arity) and u is a variable not appearing in φ. Let φ' be the formula*

$$\forall u \varphi[\alpha(\bar{y})/p_\alpha(\bar{y}, u)] \wedge \forall \bar{y}(p_\alpha(\bar{y}, l_\alpha) \rightarrow \psi(\bar{y})) \wedge \forall \bar{y}(p_\alpha(\bar{y}, r_\alpha) \rightarrow \pi(\bar{y})),$$

where l_α and r_α are new constants. Then φ is strongly satisfiable if and only if φ' is strongly satisfiable.

Proof of Lemma 4.2.8. Suppose φ' is strongly satisfiable and let φ^* be any disambiguation of φ. To show that φ^* is satisfiable, let φ'^* be the disambiguation of φ' obtained by making the same disambiguation choices in the subformula $\varphi[\alpha(\bar{y})/p_\alpha(\bar{y}, u)]$ as were made in the disambiguation φ^*. Suppose also without loss of generality that φ^* disambiguates α in favour of the left–hand reading $\psi(\bar{y})$. Choose $\mathfrak{A}$ and g such that $\mathfrak{A}, g \models \varphi'^*$, so that, in particular,

$$\mathfrak{A}, g \models \forall \bar{y}(p_\alpha(\bar{y}, l_\alpha) \rightarrow \psi(\bar{y})) \wedge \varphi[\alpha(\bar{y})/p_\alpha(\bar{y}, l_\alpha)].$$

Since $p_\alpha(\bar{y}, l_\alpha)$ occurs only positively in $\varphi[\alpha(\bar{y})/p_\alpha(\bar{y}, l_\alpha)]$, it is easy to see that $\mathfrak{A}, g \models \varphi^*$. The converse is even more straightforward. □

Lemma 4.2.8 can now be used to reduce strong satisfiability in $\mathcal{Q}$ linearly to satisfiability in $\mathcal{L}$. Of course that is hardly surprising: first–order logic is so expressive that it is easy to simulate the effects of the ?–operator. Indeed, it is worth noting that the reduction provided by lemma 4.2.8 is an 'expensive' reduction, in that it introduces an extra variable. Thus, the translation in the next theorem which it provides does not preserve membership in certain fragments of

$\mathcal{Q}$, most obviously, the propositional fragment $\mathcal{PQ}$ and the two–variable fragment $\mathcal{Q}_2$.

Theorem 4.2.9. *There exist a number c and a function $f : \mathcal{Q} \to \mathcal{L}$ such that, for all $\varphi \in \mathcal{Q}$, $|f(\varphi)| < c|\varphi|$ and φ is strongly satisfiable if and only if $f(\varphi)$ is satisfiable. Moreover, f is computable in linear time.*

Proof of Theorem 4.2.9. We first convert φ to negation normal form and then apply lemma 4.2.8 to remove occurrences of ? one–by–one, starting from the innermost. In the construction of φ' in lemma 4.2.8, the material in the 'input' formula φ is never duplicated. This explains the linear time bound. □

The non–trivial nature of strong satisfiability becomes clear by looking at the propositional fragment of $\mathcal{Q}$ (henceforth, $\mathcal{PQ}$). Recall that a formula is strongly satisfiable if and only if it is not weakly unsatisfiable (observation 4.2.3).

Theorem 4.2.10. *The problem of determining weak unsatisfiability in $\mathcal{PQ}$ is Σ_2^p–complete.*

Proof of Theorem 4.2.10. To show that a problem Π with input string x is in Σ_2^p, it suffices (Garey and Johnson, 1979, pp.163–164, Thm.7·4) to find a ternary relation $r(x, y, z)$, checkable in polynomial time, such that $\Pi(x)$ if and only if there exists a string y such that for all strings z we have $r(x, y, z)$, where the sizes of y and z are constrained to be polynomially bounded in the size of x. But this characterisation fits the definition of weak unsatisfiability exactly, where $r(x, y, z)$ is given by:

> *$r(x, y, z)$ if and only if y is a disambiguation of x and z is a truth–value assignment in the propositional variables of y such that z falsifies y.*

By another standard result (Garey and Johnson, 1979, p.166, Thm.7·6) the problem B_2 is Σ_2^p–complete, where B_2 is the problem of determining the truth–value of a quantified propositional formula φ of the form

$$\exists p_1 \ldots \exists p_j \forall q_1 \ldots \forall q_k E,$$

with E a Boolean expression in the variables $p_1, \ldots, p_j, q_1, \ldots, q_k$. But B_2 can be reduced to the problem of determining weak unsatisfiability for the ambiguous

propositional calculus. For, given a formula φ of the above form, it is easy to see that the ambiguous propositional calculus formula

$$(p_1 \,?\, \neg p_1) \wedge \ldots \wedge (p_j \,?\, \neg p_j) \wedge \neg E.$$

is weakly unsatisfiable if and only if φ is true. □

Conjecture 4.2.11. *The problem of determining strong satisfiability in $\mathcal{PQ}$ is not polynomially reducible to the problem of determining satisfiability in the propositional fragment of $\mathcal{L}$.*

Proof of Conjecture 4.2.11. Immediate from theorem 4.2.10, assuming that $\Sigma_2^p \not\subseteq$ NP. □

This is good evidence that the introduction of ambiguity operators increases the difficulty of determining (strong) satisfiability. But of course, the best–known algorithms for solving both types of problems run in worst–case exponential time, so the practical implications of this observation are unclear.

The question naturally arises as to whether analogous complexity–class results hold for larger decidable fragments of $\mathcal{L}$ than the propositional fragment. However, very simple considerations show that, in many cases, they do not. Given that an ambiguous formula $\varphi \in \mathcal{Q}$ has at most $2^{|\varphi|}$ disambiguations, in any fragment $\mathcal{L}_0$ where the problem of determining unsatisfiability is EXPTIME–hard adding ambiguous operators will not change the complexity class of the problem of determining either weak or strong unsatisfiability.

Weak Satisfiability

It would be easy to adapt theorem 4.2.9 to obtain an analogous translation reducing weak satisfiability in $\mathcal{Q}$ to satisfiability in $\mathcal{L}$. However, it turns out that, for weak satisfiability, we can do a little better. We begin by repeating the definition of map D (defined previously on page 108) as a reminder:

Definition 4.2.12 (Map D). *Let $\varphi \in \mathcal{Q}$. We denote by φ^{D} the result of replacing all occurrences of ? in φ by $\vee$.*

Example 4.2.13.

$$(\forall x(p(x) \,?\, q(x)) \wedge \exists x \neg p(x) \wedge \exists x \neg q(x))^{\mathrm{D}} = \forall x(p(x) \vee q(x)) \wedge \exists x \neg p(x) \wedge \exists x \neg q(x).$$

In $\mathcal{PQ}$, moving negations in is practically all that is required to reduce weak satisfiability to ordinary satisfiability:

Theorem 4.2.14. *Let $\varphi \in \mathcal{PQ}$. Then φ is weakly satisfiable if and only if the unambiguous formula $(\varphi^{\mathrm{N}})^{\mathrm{D}}$ is satisfiable. Hence, the problem of determining weak satisfiability in $\mathcal{PQ}$ is NP–complete (but of course this is obvious anyway).*

Proof of Theorem 4.2.14. Routine induction on the complexity of φ^{D}. □

Unfortunately, theorem 4.2.14 does not generalise to the whole of $\mathcal{Q}$: the formula

$$\varphi := \neg(\exists x(\neg p(x) \,?\, \neg q(x)) \vee \forall x p(x) \vee \forall x q(x)),$$

is a counter–example. However, the following slightly more complicated construction does work:

Lemma 4.2.15. *Let φ be a formula in negation normal form containing a subformula α of the form $\psi(\bar{y}) \,?\, \pi(\bar{y})$, where ψ and π are unambiguous and have no free variables other than $\bar{y}$. Let $\varphi[\alpha/p_\alpha(\bar{y})]$ be the result of replacing α in φ by $p_\alpha(\bar{y})$, where p_α is a new predicate letter. Let φ' be the formula*

$$\varphi[\alpha/p_\alpha(\bar{y})] \wedge (\forall \bar{y}(p_\alpha(\bar{y}) \to \psi(\bar{y})) \vee \forall \bar{y}(p_\alpha(\bar{y}) \to \pi(\bar{y}))),$$

introduced in lemma 4.2.8, then φ is weakly satisfiable if and only if φ' is weakly satisfiable.

Proof of Lemma 4.2.15. Similar to the proof of lemma 4.2.8. □

Lemma 4.2.15 guarantees that the problem of determining weak satisfiability is linearly reducible to the corresponding problem in unambiguous first–order logic. However, the translation involved in theorem 4.2.16 is 'cheaper' than that for theorem 4.2.9 in that no extra variable is required. In particular, membership in the propositional and two–variable fragments is preserved.

Theorem 4.2.16. *There exist a number c and a function $f : \mathcal{Q} \to \mathcal{L}$ such that, for all $\varphi \in \mathcal{Q}$, $|f(\varphi)| < c|\varphi|$ and φ is weakly satisfiable if and only if $f(\varphi)$ is satisfiable. Moreover, f is computable in linear time, and preserves membership in the propositional fragment, the two–variable fragment and the guarded fragment.*

Proof of Theorem 4.2.16. Similar to the proof of theorem 4.2.9. □

This concludes our investigation into the complexity of strong and weak satisfiability.

In this chapter we have investigated the best–known schemes for interpreting Underspecified Representations: Partial Logic and non–recursive satisfaction definitions. We have argued that Partial Logic may be disregarded because it fails to capture the 'correct' intuition (page 111) and it turns out that partial satisfiability can be reduced to classical satisfiability anyway (lemma 4.1.13). Therefore, we believe that the only sensible underspecified semantic frameworks are those such that the meaning of any Underspecified Representation is completely determined by the meanings of its set of readings. Accordingly, we spent some time investigating the complexity of strong and weak satisfiability.

To recapitulate, in the previous chapter we defined the current Underspecified Representation Languages, and in this chapter we have advocated non–recursive satisfactions (rather than recursive satisfaction definitions) as the most sensible way to interpret expressions in these languages. We are now in a good position to investigate the relationships between the current Underspecified Representation Languages (with respect to our preferred semantic framework). We devote the next chapter to this issue.

Chapter 5

Relative Expressive Power

The landlord painted all the walls with cracks.

> Mathematicians are like Frenchmen: whenever you say something to them they translate it into their own language, and at once it is something entirely different.
>
> GOETHE, Maxims and Reflections (1829)

Our goal in this chapter is to gain a clear understanding of the relative expressive power of each current Underspecified Representation Language. We shall try to satisfy this goal by looking for 'translation procedures' between these languages. By describing how to translate between two Underspecified Representation Languages, we are able to make clear the link between them. For example, the language $\mathcal{Q}$ is motivated by a consideration of local ambiguities (such as, lexical ambiguities); by contrast, the language $\mathcal{H}$ is clearly most suited to operator scope ambiguities. We are therefore interested in the precise relationship between these two languages. Furthermore, establishing that all current Underspecified Representation Languages have the same expressive power might help unify the discussion. There has been some recent interest in encoding Underspecified Representation Languages in terms of one another; in particular, it has been shown that there are back–and–forth maps between Predicate Logic Unplugged and the Constraint Language for Lambda Structures (Koller et al., 2003) and between Minimal Recursion Semantics and the Constraint Language for Lambda Structures (Niehren and Thater, 2003). However, we have argued that $\mathcal{H}$, PLU, MRS,

CLLS and UDRT are all the same anyway, excepting cosmetic differences. Similarly, we argued that APL, Stores, QLF and $\mathcal{R}$ are also the same as each other, again excepting cosmetic differences. Indeed, Koller admits that 'many of these formalisms seem very similar to each other' (Koller et al., 2003, p.1). Our aim is to define back–and–forth translations between those Underspecified Representation Languages which do not look similar to each other; that is, between $\mathcal{H}$ and $\mathcal{Q}$ and between APL and $\mathcal{Q}$.

It is trivial to translate any formula from any Underspecified Representation Language into an ('equivalent') formula of $\mathcal{Q}$, by disambiguating it and then conjoining its disambiguations using the ambiguation connective (assuming that the Underspecified Representation Language is sensible; that is, assuming that we can disambiguate its expressions). However, such a strategy is very inefficient from a computational perspective (because of the combinatorial explosion problem) and would therefore, in a sense, destroy the original motivation for using Underspecified Representations. For this reason, we consider only those translations for which the size of each translated formula is polynomial in the size of the original formula, and henceforth the term 'translation' will be used exclusively for this purpose. We now translate our hole language $\mathcal{H}$ into $\mathcal{Q}$ (and visa versa); we choose $\mathcal{Q}$ because it seems very different to the other Underspecified Representation languages and partly because of its mathematical simplicity.

5.1 Comparing Expressive Power

Our aim is to show that the two very different Underspecified Representation Languages $\mathcal{Q}$ and $\mathcal{H}$ have equivalent expressive power, in the sense that any expression in one of them can be translated (or 'transcribed') by an expression in the other, with at most a polynomial increase in the size of the transcription. We claim that our results apply, with trivial modifications, to PLU (Bos, 1995), MRS (Copestake et al., 1999), CLLS (Egg et al., 1998) and UDRT (Reyle, 1993).

Clearly, before we can proceed we must make our ideas more precise. Of particular interest to us are notions of expressive equivalence.

Definition 5.1.1 (Logical Equivalence). Suppose that $\mathcal{A}$ and $\mathcal{B}$ are Underspecified Representation Languages and let $\alpha \in \mathcal{A}$ and $\beta \in \mathcal{B}$. We say that α and β are *logically equivalent* if

1. every disambiguation of α is logically equivalent to some disambiguation of β, and

2. every disambiguation of β is logically equivalent to some disambiguation of α.

Note that "logically equivalent" is used in (1) and (2) in the sense of ordinary first–order logic.

Observation 5.1.2. *Logical equivalence is an equivalence relation. Moreover, if $\alpha \in \mathcal{A}$ and $\beta \in \mathcal{B}$ are logically equivalent, α is weakly (strongly) satisfiable if and only if β is.*

Proof of Observation 5.1.2. Routine. □

However, logical equivalence is a rather severe measure of expressive power and, in particular, we believe that the following conjecture is correct:

Conjecture 5.1.3. *There is no map $f : \mathcal{H} \to \mathcal{Q}$ (and similarly no map $f : APL \to \mathcal{Q}$) such that for all $\varphi \in \mathcal{H}$ ($\varphi \in APL$), with at least one disambiguation, φ is logically equivalent to $f(\varphi)$ and $|f(\varphi)| < p(|\varphi|)$ where p is a polynomial.*

Fortunately, all is not lost; the following notion appears frequently in first–order logic.

Definition 5.1.4 (Classical Transcription). Let $\varphi, \psi \in \mathcal{L}$. We say that ψ *transcribes* φ if

1. ψ entails φ, and

2. for every structure $\mathfrak{A}$ interpreting only the primitives of φ, and such that $\mathfrak{A} \models \varphi$, there exists a *unique* expansion $\mathfrak{B}$ of $\mathfrak{A}$ interpreting the primitives of ψ such that $\mathfrak{B} \models \psi$.

Thus, transcriptions are just like the formulas they transcribe, except that they may introduce new 'defined' predicates. As an example of the importance of this notion, we mention the result that every formula φ in the two–variable fragment of $\mathcal{L}$ (without equality) has a transcription ψ in the Gödel fragment. This result may be taken to show that the Gödel fragment has at least the expressive power of the two–variable fragment without equality. The classic example is of

course that of Skolemization: If φ is any formula of first–order logic, then its Skolemization is a transcription of φ.

Extending the notion of transcription for underspecification follows the same pattern as logical equivalence:

Definition 5.1.5 (Underspecified Transcription). Let $\alpha \in \mathcal{A}$ and $\beta \in \mathcal{B}$ be Underspecified Representations. We say that β *transcribes* α if

1. every disambiguation of β transcribes some disambiguation of α, and

2. every disambiguation of α is transcribed by some disambiguation of β.

Thus, if β transcribes α, we may take β to express what α expresses.

We spend the rest of this chapter proving that $\mathcal{H}$ can be transcribed in $\mathcal{Q}$ (and visa versa) and that APL can be transcribed in $\mathcal{Q}$ (and visa versa).

5.2 Reducing a Hole Language to $\mathcal{Q}$

We begin by transcribing $\mathcal{Q}$ in $\mathcal{H}$; in the sequel we will transcribe $\mathcal{H}$ in $\mathcal{Q}$.

5.2.1 Transcribing $\mathcal{Q}$ in $\mathcal{H}$

Before we define our transcription we state a useful result about $\mathcal{Q}$:

Lemma 5.2.1. *Every $\varphi \in \mathcal{Q}$ has a transcription $\varphi' \in \mathcal{Q}$ of the form*

$$\varphi_0 \wedge \bigwedge_{1 \leqslant i \leqslant n} \forall \bar{x}_i(\alpha_i(\bar{x}_i) \leftrightarrow (\varphi_i \,?\, \psi_i)), \tag{5.1}$$

where the α_i $(1 \leqslant i \leqslant n)$ are new predicate letters and the formulas φ_i $(0 \leqslant i \leqslant n)$ and ψ_i $(0 \leqslant i \leqslant n)$ are unambiguous.

Proof of Lemma 5.2.1. We say that an occurrence of a connective (or atomic formula) is of *depth* $n+1$ in formula $\varphi \in \mathcal{Q}$ if, when φ is viewed as a tree, n nodes dominate the node labelled by that connective (or atomic formula). Suppose that the deepest occurrence of the ?–operator in φ is of depth n. Note that, we cannot assume that this occurrence of the ?–operator is uniquely the deepest. Therefore, suppose that there are $k \geqslant 1$ occurrences of the ?–operator at depth n. We will refer to these occurrences as $?^{(1)}, \ldots, ?^{(k)}$. Let the left and right operands of $?^{(j)}$

for each $1 \leqslant j \leqslant k$ be denoted φ_j and ψ_j respectively. Obviously $\varphi_j, \psi_j \in \mathcal{L}$ because $?^{(j)}$ is among the deepest ?–operators in φ. Let φ_0 denote the result of replacing each $\varphi_j \, ?^{(j)} \, \psi_j$ (for each $1 \leqslant j \leqslant k$) by a new predicate letter which we denote $\alpha_j(\bar{x}_j)$. Then let

$$\varphi' = \varphi_0 \wedge \bigwedge_{1 \leqslant j \leqslant k} \forall \bar{x}_j (\alpha_j(\bar{x}_j) \leftrightarrow (\varphi_j \, ? \, \psi_j)).$$

Suppose that we disambiguate φ' as:

$$\pi = \varphi_0 \wedge \bigwedge_{1 \leqslant j \leqslant k} \forall \bar{x}_j (\alpha_j(\bar{x}_j) \leftrightarrow \varphi_j)$$

and that $\mathfrak{A}, g \models \pi$. Let $\mathfrak{B}$ be an expansion of $\mathfrak{A}$ defined by: $\alpha_j^{\mathfrak{B}} = \{\bar{a} \mid \mathfrak{A} \models \varphi_j[\bar{a}]\}$. It is then almost immediate that $\mathfrak{B} \models \varphi$. □

Thus, the expressive power of $\mathcal{Q}$ is not enhanced by the ability to embed ?–operators.

Lemma 5.2.2. *Given $\varphi \in \mathcal{Q}$ we can compute (in polynomial time) an element $(\xi, C) \in \mathcal{H}$ such that (ξ, C) is a transcription of φ.*

Proof of Lemma 5.2.2. We may assume without loss of generality that φ is of the form given in formula (5.1). Define

$$\begin{aligned} \xi &= \{l_0 : \varphi_0 \wedge \textstyle\bigwedge_{1 \leqslant i \leqslant n} \forall \bar{x}_i (\alpha_i(\bar{x}_i) \leftrightarrow h_{i1})\} \cup \\ &\quad \{l_{i,1} : \varphi_i \wedge (h_{i2} \vee \top), \\ &\quad l_{i,2} : \psi_i \wedge (h_{i3} \vee \top), \\ &\quad l_{i,3} : \top \mid 1 \leqslant i \leqslant n\}, \\ C &= \{\langle h_{i1}, l_{i,1} \rangle, \ \langle h_{i1}, l_{i,2} \rangle, \ \langle h_{i1}, l_{i,3} \rangle \mid 1 \leqslant i \leqslant n\}. \end{aligned}$$

where l_0 and the $l_{i,j}$ $(1 \leqslant i \leqslant n, \ 1 \leqslant j \leqslant 3)$ are elements of L and the h_{ij} $(1 \leqslant i \leqslant n, \ 1 \leqslant j \leqslant 3)$ are elements of H. The constraints C ensure that each hole h_{i1} is plugged with either of the expressions $\varphi_i \wedge (h_{i2} \vee \top)$ or $\psi_i \wedge (h_{i3} \vee \top)$. Here, the conjuncts $(h_{i2} \vee \top)$ and $(h_{i3} \vee \top)$ act as 'rubbish bins' into which other expressions are 'thrown'. Thus, in assembling $p^*(h_0)$ for a plugging p, the conjunct $\forall \bar{x}_i (\alpha_i(\bar{x}_i) \leftrightarrow h_{i1})$ gets filled as either

$$\forall \bar{x}_i (\alpha_i(\bar{x}_i) \leftrightarrow (\varphi_i \wedge ((\psi_i \wedge (\top \vee \top)) \vee \top))) \text{ or}$$

$$\forall \bar{x}_i(\alpha_i(\bar{x}_i) \leftrightarrow (\psi_i \wedge ((\varphi_i \wedge (\top \vee \top)) \vee \top))),$$

which are logically equivalent to $\forall \bar{x}_i(\alpha_i(\bar{x}_i) \leftrightarrow \varphi_i)$ and $\forall \bar{x}_i(\alpha_i(\bar{x}_i) \leftrightarrow \psi_i)$ respectively. The result is then immediate. □

Unfortunately, we shall see that it is less straightforward to transcribe $\mathcal{H}$ in $\mathcal{Q}$ than it is to transcribe $\mathcal{Q}$ in $\mathcal{H}$.

5.2.2 Transcribing $\mathcal{H}$ in $\mathcal{Q}$

Theorem 5.2.3. *Given $(\xi, C) \in \mathcal{H}$, with at least one disambiguation, we can compute an element $\varphi \in \mathcal{Q}$ such that φ is a transcription of (ξ, C) and $|\varphi| < p(|(\xi, C)|)$ where p is some polynomial function.*

We devote the rest of this section (that is, section 5.2.2) to proving theorem 5.2.3.

Proof of Theorem 5.2.3. Throughout, let (ξ, C) be fixed. Denote by $\bar{u}$ the tuple of variables (in some order) occurring free in any $\xi(l)$ for $l \in L(\xi)$. Expand the signature of ξ with the following primitives:

- for each $h \in H_0(\xi)$ and each $l \in L(\xi)$:
  a proposition letter $p_{h,l}$
- for each $h, h' \in H(\xi)$ and each m $(1 \leqslant m \leqslant |H(\xi)|)$:
  proposition letters $\mathrm{desc}_{h',h}$ and $\mathrm{desc}^m_{h',h}$
- for each $h \in H_0(\xi)$:
  a predicate letter η_h with the same arity as $\bar{u}$

The intuition behind the transcription is as follows. If we think of $p_{h,l}$ as stating that $\xi(l)$ 'plugs' into hole h, then any structure interpreting all the $p_{h,l}$ determines a subset p of $H \times L$. Our translation of (ξ, C) will force (the bijection defined by the set of pairs) p to be an admissible plugging for (ξ, C), and will force the $\eta_h(\bar{u})$ to be satisfied by the same tuples as the corresponding formulas $p^*(h)$. In

the sequel, we adopt the following abbreviations:

$$\text{CHOOSE}^? := \bigwedge_{\substack{l\in L(\xi)\\ h\in H_0(\xi)}} (p_{h,l}\,?\,\neg p_{h,l}),$$

$$\begin{aligned}\text{BIJECTION} := {} & \bigwedge_{h\in H_0(\xi)} \bigvee_{l\in L(\xi)} p_{h,l} \wedge \bigwedge_{l\in L(\xi)} \bigvee_{h\in H_0(\xi)} p_{h,l} \\ & \wedge \bigwedge_{\substack{h\in H_0(\xi)\\ l,l'\in L(\xi)\\ l\neq l'}} \neg(p_{h,l}\wedge p_{h,l'}) \\ & \wedge \bigwedge_{\substack{h,h'\in H_0(\xi)\\ l\in L(\xi)\\ h\neq h'}} \neg(p_{h,l}\wedge p_{h',l}).\end{aligned}$$

Of course, the expression $\text{CHOOSE}^?$ is ambiguous. We will use 'CHOOSE' to refer to an arbitrary disambiguation of $\text{CHOOSE}^?$. We will also use the following abbreviations:

$$\begin{aligned}\text{DESCENDANTS} := {} & \bigwedge_{h,h'\in H(\xi)} \left\{ \text{desc}^1_{h',h} \leftrightarrow \bigvee \left(p_{h,l} \,\middle|\, l\in L(\xi) \text{ s.t. } \xi(l) \text{ contains } h' \right) \right\} \\ & \wedge \bigwedge_{1\leqslant m<(|H(\xi)|-1)} \bigwedge_{h,h'\in H(\xi)} \left\{ \text{desc}^{m+1}_{h',h} \leftrightarrow \bigvee_{h''} \left(\text{desc}^m_{h'',h} \wedge \text{desc}^1_{h',h''} \right) \right\} \\ & \wedge \bigwedge_{h,h'\in H(\xi)} \left\{ \text{desc}_{h',h} \longleftrightarrow \bigvee_{1\leqslant m<|H(\xi)|} \text{desc}^m_{h',h} \right\},\end{aligned}$$

$$\text{NONCYC} := \bigwedge_{h\in H(\xi)} \neg \text{desc}_{h,h},$$

$$\text{CONSTRAINTS} := \bigwedge_{\langle h,l\rangle\in C} \left(p_{h,l} \vee \bigvee_{h'\in H(\xi)} \left(p_{h',l} \wedge \text{desc}_{h',h} \right) \right),$$

$$\text{OKAY} := \text{BIJECTION} \wedge \text{NONCYC} \wedge \text{CONSTRAINTS}.$$

Note that any disambiguation CHOOSE of $\text{CHOOSE}^?$ fixes truth–values for all the $p_{h,l}$. Hence, all structures satisfying a fixed disambiguation CHOOSE and the formula DESCENDANTS give the same truth–values for all $p_{h,l}$ and all $\text{desc}_{h'h}$, and therefore the same truth–value for the formula OKAY.

Observation 5.2.4. *If $p \subseteq H_0(\xi) \times L(\xi)$ is a relation defined by*

$$\langle h, l \rangle \in p \text{ if and only if } \models \text{CHOOSE} \rightarrow p_{h,l},$$

then:

1. *BIJECTION is true if and only if p is a bijection from $H_0(\xi)$ to $L(\xi)$.*
2. *If BIJECTION is true, $\text{desc}_{h',h}$ is true if and only if h' is a descendant of h according to the plugging p.*
3. *If BIJECTION is true, NONCYC $\wedge$ CONSTRAINTS is true if and only if p is an admissible plugging for (ξ, C).*

Let φ_0 be any disambiguation of (ξ, C) and, for $l \in L$, let λ_l be the formula obtained by replacing every $h \in \xi(l)$ by the corresponding atom $\eta_h(\bar{u})$. Then we transcribe (ξ, C) as:

$$\begin{aligned} \varphi \;:=\; & \text{CHOOSE}^? \wedge \text{DESCENDANTS} \\ & \wedge \Big(\text{OKAY} \rightarrow \Big(\eta_0(\bar{u}) \wedge \textstyle\bigwedge_{\substack{h \in H_0(\xi) \\ l \in L(\xi)}} (p_{h,l} \rightarrow \forall \bar{u}(\eta_h(\bar{u}) \leftrightarrow \lambda_l)) \Big) \Big) \\ & \wedge \Big(\neg\text{OKAY} \rightarrow \Big(\varphi_0 \wedge \textstyle\bigwedge_{h \in H_0(\xi)} \forall \bar{u} \neg \eta_h(\bar{u}) \Big) \Big). \end{aligned}$$

It remains to show that φ is indeed a transcription of (ξ, C).

Let φ' be any disambiguation of φ and let CHOOSE denote the (same) disambiguation of the subformula $\text{CHOOSE}^?$. Define a plugging $p : H_0(\xi) \rightarrow L(\xi)$ as:

$$p : h \mapsto l \text{ if and only if } \models \text{CHOOSE} \rightarrow p_{h,l}.$$

Suppose $\mathfrak{A}, g \models \varphi'$ for some model $\mathfrak{A}$ and assignment g. If $\mathfrak{A}, g \models \text{OKAY}$ then by observation 5.2.4 (on page 125) we know that p is admissible and therefore by lemma 3.2.11 (on page 60) p must describe a disambiguation of (ξ, C) which we denote $p^*(h_0)$. Since $\mathfrak{A}, g \models \text{OKAY}$, $\eta_h(\bar{u})$ can only be satisfied by the same tuples as the corresponding formula $p^*(h_0)$ and therefore the conjunct $\eta_0(\bar{u})$ effectively asserts that $\mathfrak{A}, g \models p^*(h_0)$. Otherwise, $\mathfrak{A}, g \models \neg\text{OKAY}$ in which case $\mathfrak{A}, g \models \varphi_0$ where φ_0 is our 'default' disambiguation of φ. Therefore every disambiguation of φ entails some disambiguation of (ξ, C).

Let $p^*(h_0)$ be a disambiguation of (ξ, C) such that $\mathfrak{A}, g \models p^*(h_0)$ with $p : H_0(\xi) \to L(\xi)$ an admissible plugging. Suppose that $\mathfrak{B}$ is the expansion of $\mathfrak{A}$ defined by:

$$p^{\mathfrak{B}}_{h,l} = \top \text{ if and only if } p(h) = l,$$
$$\text{desc}^{\mathfrak{B}}_{h,h'} = \top \text{ if and only if } h' \text{ is a descendant of } h,$$
$$\text{for each } h \in H_0(\xi), \text{ we define } \eta^{\mathfrak{B}}_h = \{\bar{u} \,|\, \text{the tuple } \bar{u} \text{ satisfies } p^*(h) \text{ in } \mathfrak{A}\}.$$

Let φ' be the disambiguation of φ obtained from $p^*(h_0)$ by disambiguating CHOOSE$^?$ as follows: for all h and l select disambiguation $p_{h,l}$ only if $p(h) = l$, select $\neg p_{h,l}$ otherwise. That is, p determines a disambiguation of CHOOSE$^?$ and hence an interpretation for the $p_{h,l}$. It follows that $\mathfrak{B}, g \models \varphi'$. The remainder of the proof is similar. □

We remark that it may not (in general) be possible to compute the above transcription in polynomial *time*, because of the need to generate the "default" disambiguation φ_0. However, this fact is likely to be of little interest in linguistic applications, where default disambiguations (that is, 'in–situ' interpretations of quantifiers) can be polynomially generated. We now turn our attention to the task of transcribing APL in $\mathcal{Q}$ (and visa versa).

5.3 Reducing Quantifier Raising to $\mathcal{Q}$

Our treatment of APL in this section is analogous to our treatment of $\mathcal{H}$ in the previous section. That is, in this section, we present a transcription of APL in $\mathcal{Q}$ (and visa versa) which we attribute to Dr. Ian Pratt–Hartmann (Department of Computer Science, The University of Manchester, England).

5.3.1 Transcribing $\mathcal{Q}$ in APL

The translation from $\mathcal{Q}$ to APL is easy:

Lemma 5.3.1. *There exists a function $f : \mathcal{Q} \to$ APL such that, for all φ, $f(\varphi)$ is a transcription of φ, and $|f(\varphi)|$ is less than some multiple of $|\varphi|$.*

Proof of Lemma 5.3.1. If φ is atomic, we define $f(\varphi) = \varphi$. In addition, we set $f(\neg\varphi) = \neg f(\varphi)$, $f(\varphi \wedge \psi) = f(\varphi) \wedge f(\psi)$, $f(\varphi \vee \psi) = f(\varphi) \vee f(\psi)$, $f(\forall x \varphi) = \forall x f(\varphi)$

and $f(\exists x\varphi) = \exists x f(\varphi)$. Finally, we set

$$f(\varphi\,?\,\psi) = (f(\varphi) \wedge (\bot \to \Box), f(\psi) \wedge (\bot \to \Box))\bot.$$

By induction on the complexity of φ, φ and $f(\varphi)$ are logically equivalent in the sense of definition 4.2.4 (on page 113). Clearly then, $f(\varphi)$ is a transcription of φ. □

5.3.2 Transcribing APL in $\mathcal{Q}$

Having succeeded in showing that every formula $\varphi \in \mathcal{Q}$ has a linear–size transcription (in fact, a logically equivalent formula) in APL in the size of φ, we might hope that the same relation holds in reverse. The following result dashes any such hope.

Lemma 5.3.2. *There exists no function $f : \mathrm{APL} \to \mathcal{Q}$ such that, for all φ, $f(\varphi)$ is a transcription of φ, and $|f(\varphi)|$ is less than some multiple of $|\varphi|$.*

Proof of Lemma 5.3.2. Consider the APL–formulas

$$\varphi_n := (\forall x_1\Box, \ldots, \forall x_n\Box, \exists y_1\Box, \ldots, \exists y_n\Box)r_n(x_1, \ldots, x_n, y_1, \ldots, y_n)$$

for $n \geqslant 1$, with r_n a 2n–ary predicate letter. It is easy to see that φ_n has at least $(n!)^2$ pairwise logically nonequivalent disambiguations, for any two orderings of the quantifiers which alternate between universal and existential will be logically nonequivalent.

Suppose that $f(\varphi_n)$ is a formula of APL of size bounded by cn, where c is a constant. Then $f(\varphi_n)$ contains fewer than cn occurrences of ?, and hence has fewer than 2^{cn} disambiguations. But, for constant c, $n! > 2^{cn}$ where n is large. Since logically nonequivalent formulas of $\mathcal{L}$ cannot have identical transcriptions (obvious), $f(\varphi_n)$ cannot be a transcription of φ_n for all n. □

Although a linear–space translation from APL to $\mathcal{Q}$ is not possible, however, we now proceed to show that a quadratic–space translation is. Very roughly, the idea is to construct a formula of $\mathcal{Q}$ whose disambiguations are exactly the specifications of all strict total orders of a given set $\{a_1, \ldots, a_n\}$. With a little care, we can construct this formula in such a way that its size is quadratic in n.

We need hardly mention that

$$\bigwedge_{1 \leqslant i < j \leqslant n} (a_i < a_j \,?\, a_j < a_i),$$

is not a solution!

In the sequel, we shall be using the following notation. If $n > 1$, we take $A = \{a_1, \ldots, a_n\}$ to be a set of pairwise distinct individuals, and $<_p$ to be a strict partial order on this set. We take l, m to be integers such that $1 \leqslant l < n$ and $0 \leqslant m \leqslant l$. For all such values of l, m, we take $<_{l,m}$ to be a binary relation (but not necessarily a partial order) satisfying the following conditions:

R1 $<_{l,m}$ is a strict total order on the set $\{a_1, \ldots, a_l\}$.

R2 $<_{l,m}$ is a strict total order on the set $\{a_1, \ldots, a_m, a_{l+1}\}$.

R3 No other elements are related by $<_{l,m}$.

R4 $<_{l,m} \cup <_p$ contains no cycles.

If $m = 0$, the set $\{a_1, \ldots, a_m, a_{l+1}\}$ in condition **R2** is to be read as $\{a_{l+1}\}$. It follows that, if $m = l < n - 1$, conditions **R1** and **R2** state that $<_{l,m}$ is a total order on $\{a_1, \ldots, a_{l+1}\}$, and so are unaffected if we replace l by $l + 1$ and m by 0. If $l = m = n - 1$, conditions **R1**, **R2** and **R4** state that $<_{l,m}$ is a total order on $\{a_1, \ldots, a_n\}$ extending $<_p$. In the sequel, we always use the abbreviations:

$$\begin{aligned} \text{irreflex} &:= \forall x \forall y (x < y \to \neg y < x), \\ \text{trans} &:= \forall x \forall y \forall z ((x < y \wedge y < z) \to x < z), \\ \text{subset}_p &:= \forall x \forall y (x <_p y \to x < y). \end{aligned}$$

Given the relations $<_p$ and $<_{l,m}$ satisfying conditions **R1**–**R4**, we construct the formulas $c_{l,m}$, c_p and $\gamma^p_{l,m}$ in the signature with the binary predicates $<$, $<_p$, $<_I$ and names $a_1, \ldots, a_n$ as follows:

$$
\begin{aligned}
c_{l,m} &:= \bigwedge\{a_i < a_j \mid a_i <_{l,m} a_j\},\\
c_p &:= \bigwedge\{a_i <_p a_j \mid a_i <_p a_j\} \wedge \bigwedge\{\neg a_i <_p a_j \mid a_i \not<_p a_j\},\\
c_I &:= \forall x \forall y(x \leqslant_I y \leftrightarrow \bigvee\{x = a_i \wedge y = a_j \mid 1 \leqslant i \leqslant j \leqslant n\}),\\
\delta^p_{l,m} &:= c_{l,m} \wedge c_p \wedge c_I \wedge \text{irreflex},\\
\gamma^p_{l,m} &:= \exists w(a_1 \leqslant_I w \wedge w \leqslant_I a_m \wedge a_{l+1} < w \wedge w < a_{m+1}) \vee\\
&\quad\ \exists w(a_1 \leqslant_I w \wedge w \leqslant_I a_l \wedge a_{l+1} <_p w \wedge w \leqslant a_{l+1}) \vee\\
&\quad\ \exists w(a_1 \leqslant_I w \wedge w \leqslant_I a_m \wedge a_{m+1} < w \wedge w < a_{l+1}) \vee\\
&\quad\ \exists w(a_1 \leqslant_I w \wedge w \leqslant_I a_l \wedge a_{m+1} \leqslant w \wedge w <_p a_{l+1}).
\end{aligned}
$$

Thus, $c_{l,m}$ asserts that the positive instances of $<_{l,m}$ satisfy the predicate $<$; c_p lists the positive *and* negative instances of $<_p$ on A; and c_I fixes the binary predicate $\leqslant_I$ to be exactly the (non–strict) ordering on the a_i given by the indices. The formula $\delta^p_{l,m}$ adds to these the condition that $<$ expresses an irreflexive relation. The formula $\gamma^p_{l,m}$ is a little more complicated. Roughly, it tells us whether the constraints on the ordering of the $a_1, \ldots, a_n$ given by $<_{l,m}$ and $<_p$ together fix the ordering of the 'next' pair of elements, a_{l+1} and a_{m+1}. More precisely, we have:

Lemma 5.3.3. *Let $n \geqslant 2$, $1 \leqslant l < n$, $0 \leqslant m < l$. Let $<_p$ be a strict partial order on the distinct elements $a_1, \ldots, a_n$, let $<_{l,m}$ be a binary relation satisfying **R1**–**R4**, and let $<_c$ be the transitive closure of $<_{l,m} \cup <_p$ (so that $<_c$ is a partial order). Let the formulas $\gamma^p_{l,m}$ and $\delta^p_{l,m}$ be as defined above. Then, if $<_c$ orders a_{l+1} and a_{m+1}, we have $\models \delta^p_{l,m} \rightarrow \gamma^p_{l,m}$; otherwise, we have $\models \delta^p_{l,m} \rightarrow \neg\gamma^p_{l,m}$.*

Proof of Lemma 5.3.3. Suppose $a_{l+1} <_c a_{m+1}$. Then there exists a sequence

$$a_{l+1} = b_0, \ldots, b_N = a_{m+1} \tag{5.2}$$

with the b_h chosen from among the elements $\{a_1, \ldots, a_n\}$ and either $b_h <_p b_{h+1}$ or $b_h <_{l,m} b_{h+1}$ for all h $(0 \leqslant h < N)$. Assume without loss of generality that (5.2) is a shortest such sequence. Let h now be the smallest integer such that $b_h \in \{a_1, \ldots, a_l\}$. (There must be such a b_h, because $b_N = a_m + 1 \in \{a_1, \ldots, a_l\}$; moreover, this integer must be positive, since $b_0 = a_{l+1} \notin \{a_1, \ldots, a_l\}$). By **R4**, $a_{m+1} \not\leqslant_{l,m} b_h$, and so by **R1**, $b_h \leqslant_{l,m} a_{m+1}$. Moreover, since (5.2) is shortest, $h = N - 1$ or $h = N$. Now, either $b_{h-1} <_{l,m} b_h$ or $b_{h-1} <_p b_h$. Suppose that $b_{h-1} <_{l,m} b_h$. Then **R3** ensures that $b_{h-1} = a_{l+1}$ and $b_h \in \{a_1, \ldots, a_m\}$, so that

$h = 1$ and the sequence (5.2) becomes (filling in the ordering relations):

$$a_{l+1} <_{l,m} b_1 <_{l,m} a_{m+1} \tag{5.3}$$

with $b_1 \in \{a_1, \ldots, a_m\}$. Suppose, on the other hand, $b_{h-1} <_p b_h$. Since $b_0, \ldots, b_{h-1} \notin \{a_1, \ldots, a_l\}$, **R3** ensures that $b_0 <_p \ldots <_p b_h$, so that $b_0 <_p b_h$, hence again $h = 1$ and (5.2) becomes:

$$a_{l+1} <_p b_1 \leqslant_{l,m} a_{m+1} \tag{5.4}$$

with $b_1 \in \{a_1, \ldots, a_l\}$. It follows that, if $a_{l+1} <_c a_{m+1}$, then $\models \delta^p_{l,m} \rightarrow \gamma'$, where γ' is the formula:

$$\exists w(a_1 \leqslant_I w \wedge w \leqslant_I a_m \wedge a_{l+1} < w \wedge w < a_{m+1}) \vee$$
$$\exists w(a_1 \leqslant_I w \wedge w \leqslant_I a_l \wedge a_{l+1} <_p w \wedge w \leqslant a_{m+1}).$$

By exactly similar reasoning, if $a_{m+1} <_c a_{l+1}$, then $\models \delta^p_{l,m} \rightarrow \gamma''$, where γ'' is the formula:

$$\exists w(a_1 \leqslant_I w \wedge w \leqslant_I a_m \wedge a_{m+1} < w \wedge w < a_{l+1}) \vee$$
$$\exists w(a_1 \leqslant_I w \wedge w \leqslant_I a_l \wedge a_{m+1} \leqslant w \wedge w <_p a_{l+1}).$$

Thus, if $<_c$ orders a_{l+1} and a_{m+1}, then $\models \delta^p_{l,m} \rightarrow \gamma^p_{l,m}$.

Conversely, suppose $<_c$ does not order a_{l+1} and a_{m+1}. Then there certainly does not exist an a_i $(1 \leqslant i \leqslant m)$ such that $a_{l+1} <_{l,m} a_i$ and $a_i <_{l,m} a_{m+1}$. By **R1** and **R2**, $<_{l,m}$ is total on both $\{a_1, \ldots, a_l\}$ and $\{a_1, \ldots, a_m, a_{l+1}\}$, so that $c_{l,m} \wedge$ irreflex decides all formulas of the form $a_i < a_j$ within these sets. Given that c_I fixes the extension of $\leqslant_I$ exactly, we must have

$$\models (c_I \wedge c_{l,m} \wedge \text{irreflex}) \rightarrow \neg\exists w(a_1 \leqslant_I w \wedge w \leqslant_I a_m \wedge a_{l+1} < w \wedge w < a_{m+1}).$$

Furthermore, if $<_c$ does not order a_{l+1} and a_{m+1}, there certainly does not exist an a_i $(1 \leqslant i \leqslant l)$ such that $a_{l+1} <_p a_i$ and $a_i <_{l,m} a_{m+1}$. By **R1**, $<_{l,m}$ is total on $\{a_1, \ldots, a_l\}$, so by similar reasoning to the above we must have

$$\models (c_p \wedge c_I \wedge c_{l,m} \wedge \text{irreflex}) \rightarrow \neg\exists w(a_1 \leqslant_I w \wedge w \leqslant_I a_l \wedge a_{l+1} <_p w \wedge w < a_{m+1}).$$

The final two disjuncts of $\gamma^p_{l,m}$ are dealt with similarly, and we have

$$\models \delta^p_{l,m} \to \neg\gamma^p_{l,m}$$

as required. □

Armed with the formula $\gamma^p_{l,m}$, we can construct a formula of $\mathcal{Q}$ whose disambiguations are exactly those formulas specifying total orderings of the $a_1, \ldots, a_n$ extending $<_p$.

Definition 5.3.4. Let $A = \{a_1, \ldots, a_n\}$ be a set (with the a_i pairwise distinct) and let $<_p$ be any partial order on A. If $<_T$ is any strict order on A, let

$$c_T := \bigwedge\{a_i < a_j \mid a_i <_T a_j\},$$

and let the formula c_p be as defined as above. We write $\Omega(<_p, A)$ for the set of all formulas of the form

$$c_p \wedge \text{trans} \wedge \text{irreflex} \wedge \text{subset}_p \wedge c_T$$

where $<_T$ is a total order on A extending $<_p$.

Thus, an element of $\Omega(<_p, A)$ is simply a formula listing some total ordering on A extending $<_p$, conjoined with a specification of $<_p$ and general facts about $<$ and $<_p$.

Lemma 5.3.5. *Let $A = \{a_1, \ldots, a_n\}$ be a set (with the a_i pairwise distinct) and let $<_p$ be any partial order on A. Then we can construct a formula $\omega(<_p, A)$ whose size is bounded by a quadratic function of n, such that: (i) every disambiguation of $\omega(<_p, A)$ is logically equivalent to some formula in $\Omega(<_p, A)$; (ii) for every formula in $\Omega(<_p, A)$, there is at least one disambiguation of $\omega(<_p, A)$ logically equivalent to it.*

Proof of Lemma 5.3.5. Let l and m be integers such that $1 \leqslant l < n$ and $0 \leqslant m \leqslant l$. We define $\omega_{l,m}$ inductively as follows:

$$\begin{aligned}
\omega_{1,0} &:= \top, \\
\omega_{l+1,0} &:= \omega_{l,l} \text{ if } 1 \leqslant l < n-1, \\
\omega_{l,m+1} &:= \omega_{l,m} \wedge (\neg\gamma^p_{l,m} \to (a_{l+1} < a_{m+1} \,?\, a_{m+1} < a_{l+1})) \\
&\qquad \text{if } 1 \leqslant l < n \text{ and } 0 \leqslant m < l.
\end{aligned}$$

We prove by induction that, for all l, m such that $1 \leqslant l < n$ and $0 \leqslant m \leqslant l$:

1. for every disambiguation $\omega^*_{l,m}$ of $\omega_{l,m}$, we can find a relation $<_{l,m}$ satisfying **R1**–**R4** such that

$$\models (c_p \wedge \text{trans} \wedge \text{irreflex} \wedge \text{subset}_p) \rightarrow (\omega^*_{l,m} \leftrightarrow c_{l,m}); \tag{5.5}$$

2. for every relation $<_{l,m}$ satisfying **R1**–**R4** we can find a disambiguation $\omega^*_{l,m}$ of $\omega_{l,m}$, such that (5.5) holds.

Setting

$$\omega(<_p, A) := c_p \wedge \text{trans} \wedge \text{irreflex} \wedge \text{subset}_p \wedge \omega_{n-1,n-1}$$

then proves the lemma.

We proceed by induction on the pair l, m with $1 \leqslant l < n$ and $0 \leqslant m \leqslant l$. *(The symbol $\star$ is used for later reference.)*

Base case: Let $l = 1$ and $m = 0$. The unique $<_{1,0}$ satisfying **R1**–**R4** is the empty relation, so that $c_{1,0} = \top$. Since $\omega_{1,0} = \top$, the result is trivial.

First inductive case: Suppose the result holds for the pair l, m such that $1 \leqslant l = m < n - 1$. If a relation $<_{l+1,0}$ satisfies **R1**–**R4** then the very same relation can be written $<_{l,l}$ also satisfying **R1**–**R4**, and visa versa. Since $\omega_{l+1,0} = \omega_{l,l}$ the result holds for the pair $l + 1, 0$ also.

Second inductive case Suppose the result holds for the pair l, m such that $1 \leqslant l < n$ and $0 \leqslant m < l$. To establish (1), let $\omega^*_{l,m+1}$ be a disambiguation of $\omega_{l,m+1}$, and suppose without loss of generality that $\omega^*_{l,m+1}$ disambiguates the subformula $a_{l+1} < a_{m+1} \,?\, a_{m+1} < a_{l+1}$ of $\omega_{l,m+1}$ as $a_{l+1} < a_{m+1}$. (The other case proceeds analogously). Then we may write:

$$\omega^*_{l,m+1} = \omega^*_{l,m} \wedge (\neg\gamma^p_{l,m} \rightarrow a_{l+1} < a_{m+1}), \tag{5.6}$$

where $\omega^*_{1,m}$ is a disambiguation of $\omega_{l,m}$. By inductive hypothesis, we can find $<_{l,m}$ satisfying **R1**–**R4** such that

$$\models (c_p \wedge \text{trans} \wedge \text{irrefelx} \wedge \text{subset}_p) \rightarrow (\omega^*_{l,m} \leftrightarrow c_{l,m}). \tag{5.7}$$

Now define:

$$<_{l,m+1} := <_{l,m} \cup \{\langle a_{l+1}, a_{m+1}\rangle\}.$$

We need to derive

$$\models (c_p \wedge \text{trans} \wedge \text{irrefex} \wedge \text{subset}_p) \rightarrow (\omega^*_{l,m+1} \leftrightarrow c_{l,m+1}). \tag{5.8}$$

$\star$ Let $<_c$ be the transitive closure on A of $<_{l,m} \cup <_p$. We have two cases, depending on whether $<_c$ orders a_{l+1} and a_{m+1}.

Suppose on the one hand that $<_c$ does order a_{l+1} and a_{m+1}. Then

$$\models (c_p \wedge \text{trans} \wedge \text{subset}_p) \rightarrow (c_{l,m} \leftrightarrow c_{l,m+1}). \tag{5.9}$$

Moreover, by lemma 5.3.3,

$$\models (c_p \wedge \text{irreflex} \wedge c_{l,m}) \rightarrow \gamma^p_{l,m},$$

hence

$$\models (c_p \wedge \text{irreflex}) \rightarrow ((c_{l,m} \wedge \omega^*_{l,m}) \rightarrow \omega^*_{l,m+1}). \tag{5.10}$$

From (5.6), (5.7), (5.9) and (5.10), (5.8) follows easily.

Suppose on the other hand that $<_c$ does not order a_{l+1} and a_{m+1}. By construction, we have

$$c_{l,m+1} := c_{l,m} \wedge a_{l+1} < a_{m+1}. \tag{5.11}$$

Moreover, by lemma 5.3.3,

$$\models (c_p \wedge \text{irreflex} \wedge c_{l,m}) \rightarrow \neg\gamma^p_{l,m},$$

hence

$$\models (c_p \wedge \text{irreflex}) \rightarrow ((c_{l,m} \wedge \omega^*_{l,m+1}) \rightarrow a_{l+1} < a_{m+1}). \tag{5.12}$$

From (5.6), (5.7), (5.11) and (5.12), (5.8) follows easily.

To establish (2) let $<_{l,m+1}$ be a relation satisfying **R1**–**R4**, and suppose without loss of generality that $a_{l+1} <_{l,m} a_{m+1}$. (The other case proceeds analogously). Let

$$<_{l,m} := <_{l,m+1} \setminus \{\langle a_{l+1}, a_{m+1}\rangle\}.$$

By inductive hypothesis, we can find a disambiguation $\omega^*_{l,m}$ of $\omega_{l,m}$ such that (5.7) holds. Now form the disambiguation of

$$\omega_{l,m+1} = \omega_{l,m} \wedge (\neg\gamma^p_{l,m} \rightarrow (a_{l+1} < a_{m+1} \,?\, a_{m+1} < a_{l+1}))$$

by disambiguating $\omega_{l,m}$ as $\omega^*_{l,m}$ and $a_{l+1} < a_{m+1} \,?\, a_{m+1} < a_{l+1}$ as $a_{l+1} < a_{m+1}$. Hence we may again write (5.6). Having established (5.6) and (5.7), our task is again to derive (5.8). From this point, the proof of (2) proceeds from the point marked $\star$ identically as for (1).

□

We note in passing that, in general, more than one disambiguation of $\omega(<_p, A)$ may pick out the same ordering of the $a_1, \ldots, a_n$. On the other hand, $\omega(<_p, A)$ involves exactly $n^2 - n + 2$ occurrences of the ?–operator, and hence has at most 2^{n^2-n+2} disambiguations. Bearing in mind that $n!$ grows at an intermediate rate between 2^n and $2^{(n^2)}$, this means that $\omega(<_p, A)$ is 'efficient', in that it does not introduce too many unnecessary (logically equivalent) disambiguations. In the sequel, we shall write $\omega(<_p; a_1, \ldots, a_n)$ for $\omega(<_p, A)$, where $A = \{a_1, \ldots, a_n\}$.

We now come to the reduction of the language APL to $\mathcal{Q}$. Specifically, we show:

Theorem 5.3.6. *There exist a constant c and a function f : APL $\rightarrow \mathcal{Q}$ such that, for all $\varphi \in$ APL, $f(\varphi)$ is a transcription of φ with $|f(\varphi)| < c|\varphi|^2$.*

Proof of Theorem 5.3.6. We employ the following abbreviations in the proof:

$$\begin{aligned} \text{first}(x) &:= \textstyle\bigwedge_{1 \leqslant i \leqslant n} \neg a_i < x, \\ \text{prec}(x, y) &:= x < y \wedge \textstyle\bigwedge_{1 \leqslant i \leqslant n} \neg(x < a_i \wedge a_i < y), \\ \text{last}(x) &:= \textstyle\bigwedge_{1 \leqslant i \leqslant n} \neg x < a_i. \end{aligned}$$

Thus, with respect to the ordering $<$, first(a_i) states that a_i comes first, prec(a_i, a_j) that a_i immediately precedes a_j and last(a_i) that a_i comes last.

We define functions g : APL $\rightarrow \mathcal{Q}$ and h : APL $\rightarrow \mathcal{L}$ (where $\mathcal{L}$ denotes the

language of ordinary first–order logic) by simultaneous recursion on φ as follows:

$$
\begin{array}{ll}
h(\varphi) := \varphi \text{ if } \varphi \text{ is atomic} & g(\varphi) := \top \text{ if } \varphi \text{ is atomic} \\
\\
h(\neg\psi) := \neg h(\psi) & g(\neg\psi) := g(\psi) \\
h(\psi \wedge \pi) := h(\psi) \wedge h(\pi) & g(\psi \wedge \pi) := g(\psi) \wedge g(\pi) \\
h(\psi \vee \pi) := h(\psi) \vee h(\pi) & g(\psi \vee \pi) := g(\psi) \wedge g(\pi) \\
\\
h(\forall x\psi) := \forall x h(\psi) & g(\forall x\psi) := g(\psi) \\
h(\exists x\psi) := \exists x h(\psi) & g(\exists x\psi) := g(\psi).
\end{array}
$$

Finally,

$$
\begin{array}{lcl}
h((<_p; \alpha_1, \ldots, \alpha_n)\psi) & := & \bigwedge_{1 \leqslant i \leqslant n} (\mathrm{last}(a_i) \rightarrow s_i(\bar{v})), \\
g((<_p; \alpha_1, \ldots, \alpha_n)\psi) & := & \omega(<_p; a_1, \ldots a_n) \wedge \bigwedge_{1 \leqslant i \leqslant n} g(\alpha_i(r_i(\bar{v}))) \wedge g(\psi) \wedge \\
 & & \forall \bar{v}(s_0(\bar{v}) \leftrightarrow h(\psi)) \wedge \\
 & & \bigwedge_{1 \leqslant i \leqslant n} \forall \bar{v}(h(\alpha(r_i(\bar{v})) \leftrightarrow s_i(\bar{v})) \wedge \\
 & & \bigwedge_{1 \leqslant i \leqslant n} (\mathrm{first}(a_i) \rightarrow \forall \bar{v}(s_0(\bar{v}) \leftrightarrow r_i(\bar{v}))) \wedge \\
 & & \bigwedge_{1 \leqslant i,j \leqslant n} (\mathrm{prec}(a_i, a_j) \rightarrow \forall \bar{v}(r_j(\bar{v}) \leftrightarrow s_i(\bar{v}))).
\end{array}
$$

In the clauses defining $h((<_p; \alpha_1, \ldots, \alpha_n)\psi)$ and $g((<_p; \alpha_1, \ldots, \alpha_n)\psi)$, $\bar{v}$ is the tuple of variables appearing free in any of the α_i or in ψ, the $a_1, \ldots, a_n$ are fresh names, the symbols $<$ and $\leqslant_I$ appearing in $\omega(<_p; a_1, \ldots, a_n)$ are fresh binary predicates and the symbols $s_0, \ldots, s_n$ and $r_1, \ldots, r_n$ are fresh predicates with the same arity as $\bar{v}$.

It helps to think of $h(\varphi)$ as the 'actual' translation of φ, with $g(\varphi)$ a conjunction of auxiliary conditions ensuring that this translation is correct. Notice that $h(\varphi)$ is always unambiguous, with any ambiguity in φ having been shunted off into $g(\varphi)$. Moreover, all occurrences of the ?–operator in $g(\varphi)$ are confined to the conjuncts $\omega(<_p; a_1, \ldots, a_n)$, which correspond one–to–one to operator lists $(<_p; \alpha_1, \ldots, \alpha_n)$ occurring in φ.

Every disambiguation of φ can be thought of as an ordering decision for every operator list $(<_p; \alpha_1, \ldots, \alpha_n)$ occurring in φ. Given an ordering decision for the operator list $(<_p; \alpha_1, \ldots, \alpha_n)$, lemma 5.3.5 guarantees that the corresponding conjunct $\omega(<_p; a_1, \ldots, a_n)$ in $g(\varphi)$ will have a disambiguation which orders the

individuals $a_1, \ldots, a_n$ in exactly the same way. Conversely, every disambiguation of $g(\varphi)$ can be thought of as a disambiguation decision for every conjunct $\omega(<_p; a_1, \ldots, a_n)$ occurring in $g(\varphi)$. Lemma 5.3.5 guarantees that any disambiguation decision for the conjunct $\omega(<_p; a_1, \ldots, a_n)$ will order the individuals $a_1, \ldots, a_n$ consistently with $<_p$, and thus induce an ordering of the corresponding operator list $\omega(<_p; \alpha_1, \ldots, \alpha_n)$ in φ. In this way, we can speak about a disambiguation of φ *corresponding to* a disambiguation of $g(\varphi)$. Note that a single disambiguation of φ may correspond to several (logically equivalent) disambiguations of $g(\varphi)$, because one ordering of the individuals $a_1, \ldots, a_n$ may be fixed by several (logically equivalent) disambiguations of $\omega(<_p; a_1, \ldots, a_n)$.

We show by induction on φ that, if φ^* is a disambiguation of φ corresponding to a disambiguation $g(\varphi)^*$ of $g(\varphi)$, then

1. if $\mathfrak{A}$ is any structure not interpreting the fresh symbols introduced in the definition of $g(\varphi)$, then we can expand $\mathfrak{A}$ to a structure $\mathfrak{A}^*$ such that $\mathfrak{A}^* \models g(\varphi)^*$,
2. $\models g(\varphi)^* \to \forall \bar{v}(h(\varphi) \leftrightarrow \varphi^*)$.

Thus, $g(\varphi)^* \wedge h(\varphi)$ is a transcription of φ^*. By definition 5.1.5 on page 122 (and 5.1.4 on page 121),

$$f(\varphi) = g(\varphi) \wedge h(\varphi)$$

is a transcription of φ. Moreover, $|f(\varphi)|$ is visibly quadratic in $|\varphi|$, and so the theorem is proved.

Each part of the claim is proved by structural induction on φ. The only interesting case is the inductive step for $\varphi = (<_p; \alpha_1, \ldots, \alpha_n)\psi$. Other cases are left for the reader to verify.

(1) Suppose that $\mathfrak{A}$ interprets none of the fresh symbols introduced in $g(\varphi)$. If $g(\varphi)^*$ is a disambiguation of $g(\varphi)$, where $\varphi = (<_p; \alpha_1, \ldots, \alpha_n)\psi$, let $g(\psi)^*$, $g(\alpha_i(r_i(\bar{v})))^*$ $(1 \leqslant i \leqslant n)$ and $\omega(<_p; a_1, \ldots, a_n)^*$, be induced disambiguations of the subformulas $g(\psi)$, $g(\alpha_i(r_i(\bar{v})))$ and $\omega(<_p; a_1, \ldots, a_n)$, respectively.

Since the symbols in $\omega(<_p; a_1, \ldots, a_n)$ are fresh, we can certainly choose an expansion $\mathfrak{A}^*$ such that $\mathfrak{A}^* \models \omega(<_p; a_1, \ldots, a_n)^*$. We may suppose without loss of generality that $\omega(<_p; a_1, \ldots, a_n)^*$ fixes $a_1 < a_2 < \cdots < a_n$. (This simplifies the notation). By inductive hypothesis, we can ensure that $\mathfrak{A}^* \models g(\psi)^*$, and, since s_0 is fresh, we can ensure that $\mathfrak{A}^* \models \forall \bar{v}(s_0(\bar{v}) \leftrightarrow h(\psi))$.

Since r_1 is fresh, we can ensure that $\mathfrak{A}^* \models \forall\bar{v}(r_1(\bar{v}) \leftrightarrow s_0(\bar{v}))$. Moreover, by inductive hypothesis, we can ensure that $\mathfrak{A}^* \models g(\alpha_1(r_1(\bar{v})))^*$, and, since s_1 is fresh, we can ensure that $\mathfrak{A}^* \models \forall\bar{v}(s_1(\bar{v}) \leftrightarrow h(\alpha_1(r_1(\bar{v}))))$. Now we can proceed similarly, securing $\mathfrak{A}^* \models g(\alpha_2(r_2(\bar{v})))^*, \ldots, \mathfrak{A}^* \models g(\alpha_n(r_n(\bar{v})))^*$, and fixing the extensions of $r_2, s_2, \ldots, r_n, s_n$. At the end of this process, $\mathfrak{A}^* \models g((\alpha_1, \ldots, \alpha_n)\psi)^*$ as required.

(2) Suppose $\mathfrak{A} \models g(\varphi)^*$, where $\varphi = (\alpha_1, \ldots, \alpha_n)\psi$. As usual, let $g(\psi)^*$, $g(\alpha_i(r_i(\bar{v})))^*$ for $(1 \leqslant i \leqslant n)$ and $\omega(<_p; a_1, \ldots, a_n)^*$ be the induced disambiguations of the relevant subformulas of $g(\varphi)$. Certainly, then, we have $\mathfrak{A} \models g(\psi)^*$, and $\mathfrak{A} \models g(\alpha(r_i(\bar{v})))^*$ for all i $(1 \leqslant i \leqslant n)$. By inductive hypothesis, $\mathfrak{A} \models \forall\bar{v}(h(\psi) \leftrightarrow \psi^*)$, and $\mathfrak{A} \models \forall\bar{v}(h(\alpha(r_i(\bar{v}))) \leftrightarrow \alpha(r_i(\bar{v})))^*$ for all i $(1 \leqslant i \leqslant n)$. Assuming without loss of generality that $\omega(<_p; a_1, \ldots, a_n)^*$ fixes $a_1 < a_2 < \cdots < a_n$, we have

$$\begin{aligned} \models g((<_p; \alpha_1, \ldots, \alpha_n)\psi)^* \quad \rightarrow \quad & \textstyle\bigwedge_{1\leqslant i\leqslant n} g(\alpha_i(r_i(\bar{v}))) \wedge g(\psi) \wedge \\ & \forall\bar{v}(s_0(\bar{v}) \leftrightarrow h(\psi)) \wedge \\ & \textstyle\bigwedge_{1\leqslant i\leqslant n} \forall\bar{v}(h(\alpha(r_i(\bar{v}))) \leftrightarrow s_i(\bar{v})) \wedge \\ & \forall\bar{v}(s_0(\bar{v}) \leftrightarrow r_1(\bar{v}))) \wedge \\ & \textstyle\bigwedge_{1\leqslant i\leqslant n} \forall\bar{v}(r_i(\bar{v}) \leftrightarrow s_{i+1}(\bar{v}))). \end{aligned}$$

Hence we have

$$\begin{aligned} &\mathfrak{A} \models \forall\bar{v}(s_0(\bar{v}) \leftrightarrow \psi^*) \\ &\mathfrak{A} \models \forall\bar{v}(\alpha(r_i(\bar{v}))^* \leftrightarrow s_i(\bar{v})) \quad \text{for all } i\ (1 \leqslant i \leqslant n) \\ &\mathfrak{A} \models \forall\bar{v}(s_i(\bar{v}) \leftrightarrow r_{i+1}(\bar{v})) \qquad \text{for all } i\ (1 \leqslant i \leqslant n). \end{aligned}$$

Performing repeated substitutions, we arrive at

$$\mathfrak{A} \models \alpha_n^*(\alpha_{n-1}^*(\ldots \alpha_2^*(\alpha_1^*(\psi^*)))\ldots),$$

hence

$$\mathfrak{A} \models \varphi^* \leftrightarrow h(\varphi).$$

Thus,

$$\models g(\varphi)^* \rightarrow \forall\bar{v}(\varphi^* \leftrightarrow h(\varphi))$$

as required. □

In this chapter we have established the expressive equivalence of $\mathcal{Q}$, $\mathcal{H}$ and APL, by showing that every formula in one of these languages can be 'translated' (or, more correctly, 'transcribed') into a formula in any of the others. In doing so, we insisted that the size of the translated formulas was not too great compared to the size of the original; otherwise, the problem of finding such a formula would be trivial, and the translations produced would undermine the motivation for employing ambiguous representations in the first place. In the next chapter we will extend our study by investigating the expressive power of Underspecified Representation Languages from a more abstract view–point.

Chapter 6

A Generic Perspective

At this pharmacy we dispense with accuracy.

In this chapter we take a more abstract look at the expressive power of Underspecified Representation Languages. In particular, we reduce ambiguous (weak) satisfiability, with respect to *any* Underspecified Representation Language, to classical satisfiability. That is, we show that: if α is an Underspecified Representation in some Underspecified Representation Language, then we can write a formula $\psi_\alpha \in \mathcal{L}$ such that α is weakly satisfiable if and only if ψ_α is classically satisfiable. Our motivation for doing this is clear; a reduction (of weak satisfiability to classical satisfiability) gives us some indication of where the limitations of underspecification lie (in terms of expressive power at least). Furthermore, our reduction suggests that the enterprise of looking for theorem provers for weak satisfiability is of little theoretical interest. We now make these ideas concrete.

In the sequel, we restrict our discussion to only those Underspecified Representation Languages which are equipped with some formal procedure for disambiguating their elements. The following definition makes this restriction formal:

Definition 6.0.7 (Computable Disambiguation Procedure). Let $\mathcal{A}$ be an Underspecified Representation Language. We say that $\mathcal{A}$ has a *computable disambiguation procedure* if there exists a (deterministic) Turing machine M such that, given any $\alpha \in \mathcal{A}$ as input, M terminates leaving all of the disambiguations of α (separated by colons) on its tape and nothing else.

Clearly, any Underspecified Representation Language for which there does not exist a computable disambiguation procedure is of little use anyway! We will also

insist that for any $\alpha \in \mathcal{A}$, every disambiguation of α involves only the variables $x_0, \ldots x_{p(n_\alpha)}$ where n_α is the size of α, and p is a fixed polynomial. Since all current Underspecified Representation Languages satisfy this condition anyway, we do not believe that it is problematic.

With these sensible restrictions in mind we can state the final theorem of the thesis:

Theorem 6.0.8 (Reduction of Weak Satisfiability). *Let α be an Underspecified Representation in some Underspecified Representation Language $\mathcal{A}$, such that $\mathcal{A}$ has a computable disambiguation procedure and every disambiguation of α involves only the variables $x_0, \ldots x_{p(n_\alpha)}$ where n_α is the size of α, and p is a fixed polynomial. Then there exists a formula $\psi_\alpha \in \mathcal{L}$, of size bounded by $p(n_\alpha)$, such that α is weakly satisfiable if and only if ψ_α is classically satisfiable.*

The remainder of this chapter is devoted to proving theorem 6.0.8; our main task is to define ψ_α, the formula at the heart of this theorem. In the next section we aim to provide an outline of what this will involve.

6.1 Setting the Scene

In writing ψ_α we will be required to axiomatise the notions of 'satisfaction' and 'is–a–disambiguation–of' within a formula algebra. Although we cannot write down axioms whose models are formulas, we can write axioms of a theory of formulas. Thus, our formula algebra is comparable to a term algebra (Hodges, 1992). The basic intuition is simple enough: the expression is–a–disambiguation–of(x, y), for example, is an atomic formula in our formula language in which x is a term denoting formula which is a disambiguation of the ambiguous formula denoted by the term y.

As has been mentioned, the key idea throughout will be that ψ_α is written in a (meta–level) 'formula language' which we denote $\mathcal{L}^\dagger$. Before we can define ψ_α we must explain what the meta–level language is.

Let $\mathcal{L}$ be any first–order language, with signature σ. Assume for simplicity that $\mathcal{L}$ is function–free and that the variables in $\mathcal{L}$ are x_i, for $i \in \mathbb{N}$. Let $\mathcal{L}^\dagger$ be the first–order language having signature σ with the additional items listed in figure 6.1. Since σ has no function symbols, every term t of $\mathcal{L}$ is either a constant or a variable. We would like to use $\mathcal{L}^\dagger$ to 'talk about' the formulas of

Individual constants:	"x_i"	$i \in \mathbb{N}$
	"r"	where r is a predicate letter of σ
	"c"	where c is an individual constant of σ
Function symbols:	"$\neg$"	(unary)
	"$\rightarrow$", "$\vee$", "$\wedge$"	(binary)
	"$\forall$", "$\exists$"	(binary)
	$\{\text{apply}^{(n)} \mid n \geqslant 0\}$	where $\text{apply}^{(n)}$ has arity $n+1$
Predicates:	$\{\text{sat}^{(n)} \mid n \geqslant 0\}$	where $\text{sat}^{(n)}$ has arity $2n+1$
	term-parse	(binary)
	parse	(ternary)
	dism_α	(unary)
	$\{\text{select}^{(n,m)} \mid 0 \leqslant m \leqslant n\}$	of arity $2n+2m$

Figure 6.1: The items to be added to the signature σ.

$\mathcal{L}$; in particular, we would like to 'describe' any formula of $\mathcal{L}$ using a term of $\mathcal{L}^\dagger$. The following definition formalises this idea:

Definition 6.1.1 (Describing Terms). Let $\varphi \in \mathcal{L}$. We define the $\mathcal{L}^\dagger$–term t_φ, *the term describing* φ, inductively as follows:

- if φ is $r(t_1, \ldots, t_n)$ then t_φ is $\text{apply}(\text{"}r\text{"}, \text{"}t_1\text{"}, \ldots, \text{"}t_n\text{"})$, where "$t_i$" is "$x_j$" if t_i is x_j and "t_i" is "c" if t_i is c,
- if φ is $\neg\psi$ then t_φ is "$\neg$"(t_ψ),
- if φ is $\psi \rightarrow \pi$ then t_φ is "$\rightarrow$"(t_ψ, t_π),
- if φ is $\psi \vee \pi$ then t_φ is "$\vee$"(t_ψ, t_π),
- if φ is $\psi \wedge \pi$ then t_φ is "$\wedge$"(t_ψ, t_π),
- if φ is $\forall x_i \psi$ then t_φ is "$\forall$"("x_i", "t_ψ"),
- if φ is $\exists x_i \psi$ then t_φ is "$\exists$"("x_i", "t_ψ").

Now we know what the meta–language $\mathcal{L}^\dagger$ is, we can define our formula ψ_α:

$$\psi_\alpha = \varphi_0 \wedge \exists\nu(\text{dism}_\alpha(\nu) \wedge \exists\bar{y}(\text{sat}^{(N)}(\nu, \bar{y}))),$$

where $N = p(n_\alpha)$ and n_α is the size of α (and p is a fixed polynomial). Under our view, ν is a variable which intuitively ranges over formulas and $\bar{y}$ is an N–tuple of variables which intuitively range over non–linguistic objects. The predicate $\mathrm{sat}^{(N)}$ aims to capture the (usual) notion of satisfaction, and we would like the predicate $\mathrm{dism}_\alpha(\nu)$ to hold if and only if the formula (described by the term) ν is a disambiguation of the Underspecified Representation α. The formula φ_0 will ensure (among other things) that the predicates $\mathrm{sat}^{(N)}$ and dism_α have the 'correct' interpretation. The next section is devoted to constructing φ_0. In the final section (section 6.3) we will prove theorem 6.0.8.

6.2 The Axiomatisations

Our plan in this section is as follows: we will first write a set of axioms to ensure that we can interpret the predicate $\mathrm{sat}^{(N)}$ (which occurs in the formula ψ_α used in theorem 6.0.8) as we have suggested. We then turn to the rather more complicated task of axiomatising the notion of 'is–a–disambiguation–of' (with respect to any Underspecified Representation Language $\mathcal{A}$ with a computable disambiguation procedure). We split this second task into two parts. In part one, we will write a formula of $\mathcal{L}$ which (in effect) encodes runs of the Turing machine $M_\mathcal{A}$ (that is, the Turing machine which writes the disambiguations of $\alpha \in \mathcal{A}$). In part two, we will write axioms for the (ternary) predicate symbol 'parse' which we will use to (in effect) convert the strings written on $M_\mathcal{A}$ into (formula describing) terms of the meta–language $\mathcal{L}^\dagger$.

Our first task is to write a set of axioms which capture the notion of satisfaction.

6.2.1 Axiomatisation of Satisfaction

Before we can write our axioms we must first introduce some technical apparatus:

Definition 6.2.1 (Selection Function). Given any function $s : \{1, \ldots, m\} \to \{1, \ldots n\}$ we can define a function mapping n–tuples from a fixed set A to m–tuples of A as follows:

$$s : \langle a_1, \ldots, a_n \rangle \mapsto \langle a_{s(1)}, \ldots, a_{s(m)} \rangle.$$

Note that, this is actually an abuse of notation. We call such a function a *selection*

function (over A), because it instructs us how to select the elements of the tuple $a_{s(1)}, \ldots, a_{s(m)}$ from the tuple $a_1, \ldots, a_n$.

Result 6.2.2. *Let* $\mathrm{select}^{(n,m)}(x_1, \ldots, x_n, x'_1, \ldots, x'_m, y_1, \ldots, y_n, y'_1, \ldots, y'_m)$ *be the formula given by*

$$\bigwedge_{(1 \leqslant i \leqslant m)} \bigvee_{(1 \leqslant j \leqslant n)} (x'_i = x_j \wedge y'_i = y_j).$$

Then for any structure $\mathfrak{A}$*, any* n*–tuples* $\bar{a}$*,* $\bar{b}$ *and* m*–tuples* $\bar{a}'$ *and* $\bar{b}'$*, we have* $\mathfrak{A} \models \mathrm{select}^{(n,m)}[\bar{a}, \bar{a}', \bar{b}, \bar{b}']$ *if and only if there exists a selection function* $s : \{1, \ldots, m\} \to \{1, \ldots n\}$ *such that* $s : \bar{a} \mapsto \bar{a}'$ *and* $s : \bar{b} \mapsto \bar{b}'$*.*

Proof of Result 6.2.2. Suppose $\mathfrak{A} \models \mathrm{select}^{(n,m)}[\bar{a}, \bar{b}, \bar{a}', \bar{b}']$. For $1 \leqslant i \leqslant m$, define $s(i) = j$ where j is the smallest integer k such that $a'_i = a_k$ and $b'_i = b_k$. Then s is a selection function mapping $\bar{a}$ to $\bar{a}'$ and $\bar{b}$ to $\bar{b}'$. The converse is even easier. □

We are now ready to write an appropriate set of axioms for $\mathrm{sat}^{(N)}$, as before, $N = p(n_\alpha)$.

Definition 6.2.3 (Axioms of Satisfaction). Let $\mathrm{SAT}^{(N)}$ denote the conjunction of the following set of formulas – which we refer to as the *axioms of satisfaction:*

$$\Big\{ \forall\nu\forall\nu'\,\forall\bar{y}(\mathrm{sat}^{(N)}(\text{“}\wedge\text{”}(\nu,\nu'),\bar{y}) \leftrightarrow (\mathrm{sat}^{(N)}(\nu,\bar{y}) \wedge \mathrm{sat}^{(N)}(\nu',\bar{y}))),$$
$$\forall\nu\forall\bar{y}(\mathrm{sat}^{(N)}(\text{“}\neg\text{”}(\nu),\bar{y}) \leftrightarrow \neg\mathrm{sat}^{(N)}(\nu,\bar{y})),$$
$$\forall\nu\forall w\forall y_1 \ldots \forall y_n(\mathrm{sat}^{(N)}(\text{“}\exists\text{”}(w,\nu), y_1 \ldots y_N)$$
$$\leftrightarrow \exists y \bigvee_{1 \leqslant i \leqslant N}(w = \text{“}x_i\text{”} \wedge \mathrm{sat}^{(N)}(\nu, y_1, \ldots, y_{i-1}, y, y_{i+1}, \ldots y_n))),$$
$$\bigwedge_{1 \leqslant i < j \leqslant N} \text{“}x_i\text{”} \neq \text{“}x_j\text{”},$$
$$\bigwedge_{1 \leqslant m \leqslant N} \bigwedge_{r \in \sigma} \forall x'_1 \ldots x'_m \forall\bar{y}(\mathrm{sat}^{(N)}(\mathrm{apply}^{(m)}(\text{“}r\text{”}, x'_1 \ldots x'_m), \bar{y})$$
$$\leftrightarrow \exists y'_1 \ldots \exists y'_m(\mathrm{select}^{(N,m)}(\text{“}x_1\text{”} \ldots \text{“}x_N\text{”}, x'_1 \ldots x'_m, \bar{y}, y'_1 \ldots y'_m)$$
$$\wedge\, r(y'_1 \ldots y'_m))) \Big\}.$$

Recall that σ is the signature of our underlying (object–level) language $\mathcal{L}$ and $\mathrm{apply}^{(m)}$ is a primitive function of our meta–language $\mathcal{L}^\dagger$. Intuitively, $\mathrm{apply}^{(m)}$ represents the result of applying its first argument to its other arguments, and ν and ν' intuitively range over $\mathcal{L}^\dagger$–terms, which in turn describe formulas of $\mathcal{L}$. We are now ready to make a formal statement about how we would like the predicate $\mathrm{sat}^{(N)}$ to be interpreted:

Lemma 6.2.4 (Correctness of $\mathrm{SAT}^{(N)}$). *Let $\varphi(\bar{x})$ be any formula of $\mathcal{L}$ involving only variables from $\bar{x} = x_1 \ldots x_N$. Let t_φ be the term in $\mathcal{L}^\dagger$ describing φ. Finally, let $\delta_\varphi(\bar{y})$ be the $\mathcal{L}^\dagger$–formula $\mathrm{sat}^{(N)}(t_\varphi, \bar{y})$ where $\bar{y}$ is an N–tuple of variables. Then, if $\mathfrak{A} \models \mathrm{SAT}^{(N)}$, we have for any N–tuple $\bar{a}$, $\mathfrak{A} \models \delta_\varphi[\bar{a}]$ if and only if $\mathfrak{A} \models \varphi[\bar{a}]$.*

Proof of Lemma 6.2.4. Proceed by structural induction over φ.

Base Case: Suppose that φ is $r(\bar{x}')$, where $\bar{x}'$ is a selection from $x_1, \ldots, x_N$. Then $\mathfrak{A} \models \varphi[\bar{a}]$ implies that $\mathfrak{A} \models r[\bar{a}']$ where $\bar{a}'$ is the corresponding selection from $\bar{a}$. Let $\overline{\text{“}x\text{”}}$ be the corresponding selection from the constants “x_1”, …, “x_n”. Thus, by Result 6.2.2, $\bar{a}$, $\bar{a}'$ satisfy $\psi(\bar{z}, \bar{z}')$ where $\psi(\bar{z}, \bar{z}') = \mathrm{select}^{(N,m)}(\overline{\text{“}x\text{”}}, \overline{\text{“}x\text{”}'}, \bar{z}, \bar{z}')$. Therefore, if $\delta_\varphi(\bar{y})$ is

$$\exists \bar{y}'(\mathrm{select}^{(N,m)}(\bar{x}, \text{“}x_1\text{”}, \ldots, \text{“}x_N\text{”}, \bar{y}, \bar{y}') \wedge r(\bar{y}'))$$

then $\mathfrak{A} \models \delta_\varphi[\bar{a}]$. Hence $\bar{a}$ satisfies $\mathrm{sat}^{(N)}(\mathrm{apply}^{(N)}(\text{“}r\text{”}, \overline{\text{“}x\text{”}}), \bar{y})$ in $\mathfrak{A}$, because $\mathfrak{A} \models \mathrm{SAT}^{(N)}$. Thus, $\mathfrak{A} \models \delta_\varphi[\bar{a}]$. The converse is similar.

Inductive Steps: We present just two of the inductive cases (negation and existential quantification), since the other cases are similar. Suppose that $\varphi = \neg\psi$ and $\mathfrak{A} \models \delta_\varphi[\bar{a}]$ where $\delta_\varphi = \mathrm{sat}^{(N)}(t_{\neg\psi}, \bar{y})$. Since $\mathfrak{A} \models \mathrm{SAT}^{(N)}$ we have $\mathfrak{A} \models \neg\mathrm{sat}^{(N)}(t_\psi, \bar{y})$. That is, $\mathfrak{A} \models \neg\delta_\psi[\bar{a}]$ where $\delta_\psi = \mathrm{sat}^{(N)}(t_\psi, \bar{y})$. By inductive hypothesis, $\mathfrak{A} \models \neg\psi[\bar{a}]$ and therefore $\mathfrak{A} \models \varphi[\bar{a}]$. The converse is similar. Now suppose that $\varphi = \exists x_{i_0}\psi$ and $\mathfrak{A} \models \delta_\varphi[a_1 \ldots a_N]$ where $\delta_\psi = \mathrm{sat}^{(N)}(t_{\exists x_j \psi}, \bar{y})$. Since $\mathfrak{A} \models \mathrm{SAT}^{(N)}$, there exists a' such that

$$\mathfrak{A} \models \mathrm{sat}^{(N)}[f, a_1 \ldots a_{j-1}\, a'\, a_{j+1} \ldots a_N],$$

where $f = t_\psi^{\mathfrak{A}}$. That is,

$$\mathfrak{A} \models \delta_\psi[a_1 \ldots a_{j-1}\, a'\, a_{j+1} \ldots a_N].$$

By inductive hypothesis,

$$\mathfrak{A} \models \psi[a_1 \ldots a_{j-1}\, a'\, a_{j+1} \ldots a_N],$$

and therefore

$$\mathfrak{A} \models \varphi[a_1 \ldots a_{j-1}\, a_j\, a_{j+1} \ldots a_N].$$

Again, the converse is similar.

□

In this section, we have written a set of axioms $\mathrm{SAT}^{(N)}$ which capture the notion of satisfaction. This axiomatisation ensures that $\mathrm{sat}^{(N)}$ (which is a predicate symbol in our meta–language $\mathcal{L}^{\dagger}$ and a key feature of the formula ψ_{α}) is interpreted 'correctly' (lemma 6.2.4). In the next section, we will write a set of axioms DISM_{α} to capture the notion of 'is–a–disambiguation–of'.

6.2.2 Axiomatisation of 'is–a–disambiguation–of'

In this section we turn our attention to the rather complicated task of axiomatising the notion of 'is–a–disambiguation–of'. Recall that theorem 6.0.8 applies only to those Underspecified Representation Languages with 'computable disambiguation procedures'. An Underspecified Representation Language $\mathcal{A}$ is said to have a computable disambiguation procedure if there exists a (deterministic) Turing machine M such that, given any $\alpha \in \mathcal{A}$ as input, M terminates leaving all of the disambiguations of α (separated by colons) on its tape, and nothing else. In the sequel, we will consider a special Turing machine M_{α} – which is an adapted version of M – such that M_{α} computes only the disambiguations of the specific Underspecified Representation α (and terminates). The main aim of this section is to write an $\mathcal{L}^{\dagger}$–formula $\varphi_{M_{\alpha}}$ such that any model of $\varphi_{M_{\alpha}}$ corresponds to a run of M_{α} (and visa versa). We note in passing that many aspects of our encoding will look similar to those found in standard proofs of the undecidability of first–order satisfiability.[1]

The plan of this section is as follows: We begin by writing down an $\mathcal{L}$–formula $\varphi_{M_{\alpha}}$ which (in effect) encodes runs of M_{α}. We will then write axioms for a predicate 'parse', which we will use to (in effect) convert strings of symbols written on the tape of M_{α} into (formula describing) terms of the meta–language $\mathcal{L}^{\dagger}$. Finally, we will show that all of this material enables us to interpret the predicate dism_{α} as we have suggested.

[1] Garey and Johnson provide a detailed account of such an encoding (Garey and Johnson, 1979, pp.19–22).

$\langle S_1, \sigma_1, S'_1, \sigma'_1, \delta_1 \rangle$
$\langle S_2, \sigma_2, S'_2, \sigma'_2, \delta_2 \rangle$
$\vdots$
$\langle S_n, \sigma_n, S'_n, \sigma'_n, \delta_n \rangle$

Table 6.1: A depiction of the program of M_α.

Encoding the Runs of the Turing Machine M_α

Our aim in this sub–section is to write a formula φ_{M_α} such that there is a one–to–one correspondence between the models of φ_{M_α} and the runs of M_α. That is, we aim to write a formula φ_{M_α} such that any model of φ_{M_α} contains a 'picture' of a run of M_α.

Let $\mathcal{A}$ be any Underspecified Representation Language, $\alpha \in \mathcal{A}$ and M_α be a deterministic Turing machine which terminates leaving all of the disambiguations of α (separated by colons) on its tape, and nothing else. We depict the 'program' of M_α as a finite table (see table 6.1). We interpret any row $\langle S_k, \sigma_k, S'_k, \sigma'_k, \delta_k \rangle$ (for $1 \leqslant k \leqslant n$) of table 6.1 as follows: if M_α is currently in state S_k reading symbol σ_k from the finite alphabet of the language $\mathcal{L}$ (together with the symbol ':') then M_α changes state to S'_k, writes symbol σ'_k (taken from the same alphabet as σ_k) and moves to the successor of the current square if $\delta_k = +1$, or moves to the immediate predecessor of the current square if $\delta_k = -1$.

We are now ready to write out our formula φ_{M_α} which, as we have said, in effect encodes runs of M_α. We help ourselves to the signature displayed in figure 6.2 (in addition to that mentioned earlier), with the indicated intended (intuitive) interpretations. Note that we treat "squares" and "times" as different objects: the former satisfy the unary predicate square, the latter, the unary predicate time. Technically, it is probably not necessary to separate times and squares; but we do so to make our exposition clearer.

Since M_α only has a finite number of states, we only have a finite number of

$y < z$	y is less than z
$\text{succ}(z, z')$	z' is the successor of z
$\text{square}(x)$	x is a square of M_α
$\text{time}(t)$	t is a time
$H(x, t)$	the head is at square x at step t
$I_S(t)$	M_α is in state S at step t
$T_\sigma(x, t)$	square x contains σ at step t

Figure 6.2: Symbols in the extended version of the signature of σ.

(corresponding) predicates I_S. By assumption, one of these states is the halting state, which we shall denote HALT. Thus, our signature contains the predicate I_{HALT}, and we read $I_{\text{HALT}}(t)$ as "M_α is in the state HALT at step t". In the sequel, t_{end} is a (new) constant symbol which intuitively denotes the time of the first halting state of M_α. Similarly, s_{end} is a (new) constant symbol which intuitively denotes the rightmost square written on by the time t_{end}. Finally, we let 0_{Sq} and 0_{Tm} be constants which intuitively denote the first square and the start time of M_α respectively.

We begin our encoding with some routine background material. The following formula ensures that the predicate succ is interpreted 'correctly':

$$\forall x(\text{square}(x) \rightarrow \neg\,\text{succ}(x, 0_{\text{Sq}})) \wedge \forall x(\text{square}(x) \rightarrow \exists y(\text{square}(y) \wedge \text{succ}(x, y))) \wedge$$
$$\forall x_1 \forall x_2 \forall y((\text{square}(x_1) \wedge \text{square}(x_2) \wedge \text{square}(y) \wedge \text{succ}(x_1, y) \wedge \text{succ}(x_2, y)) \rightarrow$$
$$x_1 = x_2) \wedge$$
$$\forall x \forall y_1 \forall y_2((\text{square}(x) \wedge \text{square}(y_1) \wedge \text{square}(y_2) \wedge \text{succ}(x, y_1) \wedge \text{succ}(x, y_2)) \rightarrow$$
$$y_1 = y_2) \wedge$$
$$\forall y((\text{square}(y) \wedge \neg(y = 0_{\text{Sq}})) \rightarrow \exists x(\text{square}(x) \wedge \text{succ}(x, y))).$$

We will also need a copy of the above formula 'relativised' to time instead of square; that is, we need a duplicate of the above formula with all occurrences of 'square' replaced by 'time' and all occurrences of '0_{Sq}' replaced by '0_{Tm}'. To save space, we will not write out this formula. We will also need a formula which ensures that the symbol '$<$' is interpreted 'correctly'; the following formula serves

this purpose:

$$\begin{aligned}
&\forall x(\text{square}(x) \to \neg(x < x)) \wedge \\
&\forall x \forall y \forall z((\text{square}(x) \wedge \text{square}(y) \wedge \text{square}(z) \wedge x < y \wedge y < z) \to x < z) \wedge \\
&\forall x \forall y((\text{square}(x) \wedge \text{square}(y)) \to (x < y \vee y < x \vee x = y)) \wedge \\
&\forall x(\text{square}(x) \to \neg(x < 0_{\text{Sq}})) \wedge \\
&\forall x \forall y((\text{square}(x) \wedge \text{square}(y) \wedge \text{succ}(x, y)) \to (x < y \wedge \\
&\neg \exists z(\text{square}(z) \wedge x < z \wedge z < y))) \wedge \\
&\neg \text{succ}(0_{\text{Sq}}, 0_{\text{Sq}}).
\end{aligned}$$

Once again, we will also need a copy of this formula in which all occurrences of 'square' are replaced by 'time' and all occurrences of '0_{Sq}' are replaced by '0_{Tm}'.

We now write formulas corresponding to each row in the table of M_α. For each row ($1 \leqslant k \leqslant n$) in table 6.2.2 we write a formula as follows:

If $\delta_k = +1$ then write:

$$\begin{aligned}
\forall x \forall x' \forall t \forall t'(I_{S_k}(t) \wedge T_{\sigma_k}(x, t) \wedge \text{succ}(t, t') \wedge H(x, t) \wedge \text{succ}(x, x') \\
\to I_{S'_k}(t') \wedge T_{\sigma'_k}(x, t') \wedge H(x', t')).
\end{aligned}$$

If $\delta_k = -1$ then write:

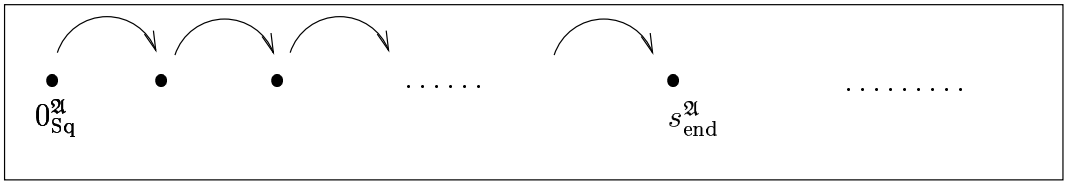
$$\begin{aligned}
\forall x \forall x' \forall t \forall t'(I_{S_k}(t) \wedge T_{\sigma_k}(x, t) \wedge \text{succ}(t, t') \wedge H(x, t) \wedge \text{succ}(x', x) \\
\to I_{S'_k}(t') \wedge T_{\sigma'_k}(x, t') \wedge H(x', t')).
\end{aligned}$$

The following formula states the initial conditions of M_α and that the head of M_α can only be in one place at any given time:

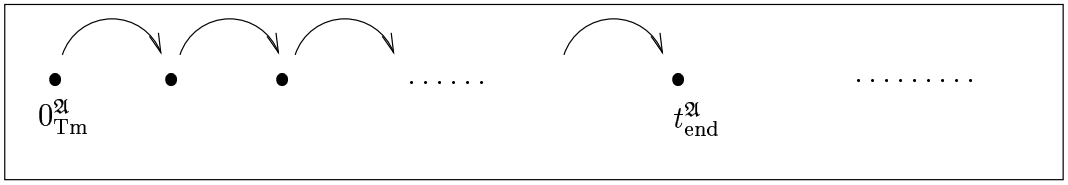
$$\begin{aligned}
&I_0(0_{\text{Tm}}) \wedge H(0_{\text{Sq}}, 0_{\text{Tm}}) \wedge \forall x T_0(x, 0_{\text{Tm}}) \wedge \\
&\forall x \forall x' \forall t((H(x, t) \wedge H(x', t)) \to x = x')
\end{aligned}$$

Finally, we must ensure that our special constant symbols can be interpreted as we want them to be:

$$\forall x(\text{square}(x) \to \forall t((\text{time}(t) \wedge t < t_{\text{end}} \wedge H(x, t)) \to x < s_{\text{end}})).$$



The elements of square$^{\mathfrak{A}}$



The elements of time$^{\mathfrak{A}}$

Figure 6.3: An intuitive depiction of a substructure of $\mathfrak{A}$, where $\mathfrak{A}$ is any model of φ_{M_α}.

The conjunction of all of the above formulas is denoted φ_{M_α}. We claim that there is a one–to–one correspondence between models of the formula φ_{M_α} and runs of the Turing machine M_α.

All of the models of φ_{M_α} contain a substructure which is two copies of the natural numbers, one in the extension of square (with initial element 0_{Sq}) and the other in the extension of time (with initial element 0_{Tm}). In fact, all models of φ_{M_α} have an two initial segments which contain only finite successors of 0, followed by (possibly empty) less 'well–behaved' segments (which are isomorphic to zero or more copies of the integers).[2] The situation is best explained using a diagram. If $\mathfrak{A} \models \varphi_{M_\alpha}$ then $\mathfrak{A}$ contains the substructure depicted in figure 6.3. Therefore, in the sequel we will be able to talk about the domain elements of any structure $\mathfrak{A}$ (such that $\mathfrak{A} \models \varphi_{M_\alpha}$) as 'squares', 'times' (and so on) of M_α because of the obvious one–to–one correspondence between the models of φ_{M_α} and runs of M_α (as illustrated in figure 6.3). For example, T_{σ_k} can be thought of as a relation

[2]The reason for the existence of these less well–behaved substructure lies in our inability to axiomatise the natural numbers.

between squares (elements of square$^{\mathfrak{A}}$) and times (elements of time$^{\mathfrak{A}}$). That is, we interpret the relation $\mathfrak{A} \models T_{\sigma_k}[s,t]$ as 'meaning' that σ_k is written on square s at time t in the run of M_α encoded by the structure $\mathfrak{A}$ (where $\mathfrak{A} \models \varphi_{M_\alpha}$). Now that we have encoded M_α, we turn our attention to the task of axiomatising parsing.

Axiomatisation of Parsing

Given a structure $\mathfrak{A}$ interpreting the Turing machine paraphernalia (that is, $\mathfrak{A} \models \varphi_{M_\alpha}$) and given finite successors s, s' (of 0_{Sq}) and q (of 0_{Tm}), we can make sense of the statement "a formula φ has been written on the tape of M_α at time step q between squares s and s'." It means that there exist objects $s_0, \ldots, s_n$ in the extension of square with $s_0 = s$ and $s_n = s'$ and $\mathfrak{A} \models \mathrm{succ}[s_i, s_{i+1}]$, for $0 \leqslant i \leqslant n$, such that the string of symbols $\sigma_0 \ldots \sigma_n$, (where for all i, σ_i is the unique symbol such that $\mathfrak{A} \models T_{\sigma_i}[s_i, q]$), spells out the formula φ, just as one would write it on a piece of paper. Our task in this section is to write axioms for the (ternary) predicate 'parse', which (in effect) will convert such a string into a term t_φ, describing the formula φ in the meta–language $\mathcal{L}^\dagger$.

Definition 6.2.5 (Axioms of Parsing). Let $\varphi^\alpha_{\mathrm{parse}}$ denote the conjunction of the set of formulas listed below – which we refer to as the *axioms of parsing*:

1. For each variable v mentioned in α:

$$\forall y \forall z (T_v(y, t_{\mathrm{end}}) \to (\text{term-parse}(y,z) \leftrightarrow z = \text{“}v\text{”}))$$

2. For each individual constant c mentioned in α:

$$\forall y \forall z (T_c(y, t_{\mathrm{end}}) \to (\text{term-parse}(y,z) \leftrightarrow z = \text{“}c\text{”}))$$

3. For atomic formulas we have:

$$\begin{gathered}\forall y_0 \ldots \forall y_{2n+1} \forall z_1 \ldots \forall z_n ((T_r(y_0, t_{\mathrm{end}}) \wedge \mathrm{succ}(y_0, y_1) \wedge T_(\,(y_1, t_{\mathrm{end}}) \wedge \mathrm{succ}(y_1, y_2) \\ \wedge \text{term-parse}(y_2, z_1) \wedge \mathrm{succ}(y_2, y_3) \wedge T_,(y_3, t_{\mathrm{end}}) \wedge \\ \mathrm{succ}(y_3, y_4) \wedge \ldots \wedge \text{term-parse}(y_{2n}, z_n) \wedge \mathrm{succ}(y_{2n}, y_{2n+1}) \wedge \\ T_)(y_{2n+1}, t_{\mathrm{end}})) \to (\mathrm{parse}(y_0, y_{2n+1}, z) \leftrightarrow z = \mathrm{apply}^{(n)}(\text{“}r\text{”}, z_1, \ldots, z_n)))\end{gathered}$$

4. Negation is dealt with as follows:

$$\forall y_0 \forall y_1 \forall y_2 \forall z \forall z' (T_{\neg}(y_0, t_{\text{end}}) \wedge \text{succ}(y_0, y_1) \wedge \text{parse}(y_1, y_2, z') \\ \rightarrow (\text{parse}(y_0, y_2, z) \leftrightarrow z = \text{“}\neg\text{”}(z')))$$

5. Similarly, for conjunction we have:

$$\forall y_0 \ldots \forall y_6 \forall z \forall z' \forall z'' ((T_{(}(y_0, t_{\text{end}}) \wedge \text{succ}(y_0, y_1) \wedge \text{parse}(y_1, y_2, z') \wedge \\ \text{succ}(y_2, y_3) \wedge T_{\wedge}(y_3, t_{\text{end}}) \wedge \text{succ}(y_3, y_4) \wedge \text{parse}(y_4, y_5, z'') \wedge \text{succ}(y_5, y_6) \wedge \\ T_{)}(y_6, t_{\text{end}})) \rightarrow (\text{parse}(y_0, y_6, z) \leftrightarrow z = \text{“}\wedge\text{”}(z', z'')))$$

6. Existential quantifiers are dealt with as follows:

$$\forall y_0 \ldots \forall y_5 \forall z \forall z' \forall z'' ((T_{\exists}(y_0, t_{\text{end}}) \wedge \text{succ}(y_0, y_1) \wedge \text{term-parse}(y_1, z') \wedge \\ \text{succ}(y_1, y_2) \wedge T_{(}(y_2, t_{\text{end}}) \wedge \text{succ}(y_2, y_3) \wedge \text{parse}(y_3, y_4, z'') \wedge \text{succ}(y_4, y_5) \wedge \\ T_{)}(y_5, t_{\text{end}})) \rightarrow (\text{parse}(y_0, y_5, z) \leftrightarrow z = \text{“}\exists\text{”}(z', z'')))$$

7. Disjunction and universal quantifiers can be parsed analogously.

We must now check that our axioms $\varphi^{\alpha}_{\text{parse}}$ allow the predicate parse to be interpreted correctly.

Result 6.2.6. *Let $\mathfrak{A}$ be any structure such that $\mathfrak{A} \models \varphi_{M_\alpha} \wedge \varphi^{\alpha}_{\text{parse}}$. If a well–formed formula φ is written on the tape of M_α between the squares s and s' at termination, then* $\text{parse}(y, y', z)$ *is satisfied by the tuple $\langle s, s', a\rangle$ if and only if $a = t^{\mathfrak{A}}_{\varphi}$.*

Proof of Result 6.2.6. Proceed by structural induction on $\varphi \in \mathcal{L}$: The base case is $\varphi = r(\tau_1, \ldots, \tau_n)$ where r is an n–ary predicate symbol in $\mathcal{L}$ and the τ_i (for $1 \leqslant i \leqslant n$) are ordinary terms. Since $\mathfrak{A} \models \varphi_{M_\alpha}$ the antecedent of the formula listed as item 3 in the definition of $\varphi^{\alpha}_{\text{parse}}$ has the interpretation depicted in figure 6.4. It is clear (from the items listed as 1 and 2 in the definition of $\varphi^{\alpha}_{\text{parse}}$) that any pair $\langle s, a\rangle$ satisfies term-parse(y, z) in $\mathfrak{A}$ if and only if $a =$ “τ” where “τ” denotes “x_j” if τ is x_j and “τ” denotes “c” if τ is c. Since the tuple $\langle s, s', a\rangle$ satisfies the antecedent of the formula listed as item 3 in the definition of $\varphi^{\alpha}_{\text{parse}}$, the consequent of the same formula ensures that: the tuple $\langle s, s', a\rangle$ satisfies $\text{parse}(y, y', z)$ in $\mathfrak{A}$

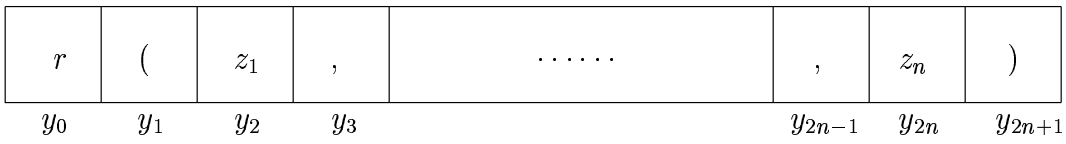


Figure 6.4: An intuitive depiction of item 3 in the definition of $\varphi^{\alpha}_{\text{parse}}$.

if and only if $a = \text{apply}^{(n)}(\text{“}r\text{”}^{\mathfrak{A}}, \text{“}\tau_1\text{”}^{\mathfrak{A}}, \ldots, \text{“}\tau_n\text{”}^{\mathfrak{A}})$. Hence $a = t^{\mathfrak{A}}_{r(\tau_1,\ldots,\tau_n)}$ where $t_{r(\tau_1,\ldots,\tau_n)}$ denotes the $\mathcal{L}^{\dagger}$–term describing $r(\tau_1, \ldots, \tau_n)$. The inductive steps are even easier. □

The Correctness of the Axiomatisation of Disambiguation

We are now ready to complete our axiomatisation of ‘is–a–disambiguation–of’. Let $\varphi^{\alpha}_{\text{dism}}$ denote the formula:

$$\forall\nu(\text{dism}_{\alpha}(\nu) \leftrightarrow \exists x\exists x'(x < s_{\text{end}} \wedge x' < s_{\text{end}} \wedge \text{parse}(x, x', \nu) \\ \wedge \exists y\exists y'(\text{succ}(y, x) \wedge \text{succ}(x', y') \wedge T_{:}(y, t_{\text{end}}) \wedge T_{:}(y', t_{\text{end}}))))$$

Finally, let DISM_{α} denote the conjunction $\varphi_{M_\alpha} \wedge \varphi^{\alpha}_{\text{parse}} \wedge \varphi^{\alpha}_{\text{dism}}$. We must now check that our formula DISM_{α} works; that is, we must verify that the predicate dism_{α} can be interpreted as we want it to be.

Lemma 6.2.7 (Correctness of DISM_{α}). *Let α be an Underspecified Representation in some Underspecified Representation Language $\mathcal{A}$ such that $\mathcal{A}$ has a computable disambiguation procedure. Let the structure $\mathfrak{A}$ interpret the language $\mathcal{L}^{\dagger}$ with $\mathfrak{A} \models \text{DISM}_{\alpha}$ and let $\varphi \in \mathcal{L}$. Then for all $a \in \text{dom}(\mathfrak{A})$, $\mathfrak{A} \models \text{dism}_{\alpha}[a]$ if and only if $a = t^{\mathfrak{A}}_{\varphi}$ for some φ, such that φ is a disambiguation of α.*

Proof of Lemma 6.2.7. Suppose $\mathfrak{A} \models \varphi^{\alpha}_{\text{dism}}$. We know, by definition of $\varphi^{\alpha}_{\text{dism}}$ that there exist domain elements s, s' such that the tuple $\langle s, s', a\rangle$ satisfies the formula $x < s_{\text{end}} \wedge x' < s_{\text{end}} \wedge \text{parse}(x, x', \nu)$ in $\mathfrak{A}$. Because $\mathfrak{A} \models \varphi_{M_\alpha}$ we can interpret this formula as ‘saying’: at termination some disambiguation of α is written on the tape of M_α between squares s and s'. We denote this disambiguation φ. Finally, because $\mathfrak{A} \models \varphi^{\alpha}_{\text{parse}}$ we know that $a = t^{\mathfrak{A}}_{\varphi}$ by result 6.2.6.

Conversely, suppose that $a = t^{\mathfrak{A}}_{\varphi}$ for some $\varphi \in \mathcal{L}$ such that φ is a disambiguation of α. Since M_α writes all of the disambiguations on its tape (and nothing else) there must exist squares s and s' such that φ is written on the tape of M_α

between s and s'. Therefore, by result 6.2.6, $\text{parse}(x, x', z)$ is satisfied by the tuple $\langle s, s', a\rangle$ in $\mathfrak{A}$. Because s_{end} is the last 'written on' square of M_α, the tuple $\langle s, s', a\rangle$ must also satisfy the formula $x < s_{\text{end}} \wedge x' < s_{\text{end}} \wedge \text{parse}(x, x', \nu)$ in $\mathfrak{A}$ and therefore, since the tuple $\langle s, s', a\rangle$ satisfies $\varphi^{\alpha}_{\text{dism}}$ in $\mathfrak{A}$, we have $\mathfrak{A} \models \text{dism}_\alpha[a]$. □

In this section, we have written a set of axioms DISM_α which capture the notion of 'is–a–disambiguation–of', and we have showed that our axiomatisation works (lemma 6.2.7). We are now ready to bring together all of the work we have done in this chapter; that is, we are ready to prove our final theorem (theorem 6.0.8).

6.3 Proving the Reduction Theorem

We are now ready to prove theorem 6.0.8 which we re–state below.

Reduction Theorem. *Let α be an Underspecified Representation in some Underspecified Representation Language $\mathcal{A}$, such that $\mathcal{A}$ has a computable disambiguation procedure and every disambiguation of α involves only the variables $x_0, \ldots x_{p(n_\alpha)}$ where n_α is the size of α, and p is a fixed polynomial. Then there exists a formula $\psi_\alpha \in \mathcal{L}$, of size bounded by $p(n_\alpha)$, such that α is weakly satisfiable if and only if ψ_α is classically satisfiable.*

Proof of the Reduction Theorem (Theorem 6.0.8). Let $\text{SAT}^{(N)}$ and DISM_α be as defined above and let φ_0 denote the conjunction $\text{SAT}^{(N)} \wedge \text{DISM}_\alpha$. We claim that the following formula suffices:

$$\psi_\alpha := \varphi_0 \wedge \exists\nu(\text{dism}_\alpha(\nu) \wedge \exists\bar{y}(\text{sat}^{(N)}(\nu, \bar{y}))).$$

Suppose that α is weakly satisfiable. Then there exists a structure $\mathfrak{A}$ and a tuple $\bar{a}$ such that for some disambiguation $\varphi(\bar{x})$ of α, $\bar{a}$ satisfies $\varphi(\bar{x})$ in $\mathfrak{A}$. Let $\mathfrak{B}$ be an expansion of $\mathfrak{A}$ such that $\mathfrak{B} \models \text{SAT}^{(N)} \wedge \text{DISM}_\alpha$; that is $\mathfrak{B} \models \varphi_0$. We claim that it is easy to modify the disambiguation procedure, if necessary, so that every disambiguation of α does have an expansion satisfying $\text{SAT}^{(N)} \wedge \text{DISM}_\alpha$, without affecting weak satisfiability. This modification need only to relativise all quantification in $\text{SAT}^{(N)} \wedge \text{DISM}_\alpha$ to a domain of non–linguistic objects, thus '$\forall x_i \ldots$' is replaced by '$\forall x_i(\text{real-obj}(x_i) \rightarrow \ldots)$' and '$\exists x_i \ldots$' is replaced by '$\exists x_i(\text{real-obj}(x_i) \wedge \ldots)$'. This avoids problems in

which the disambiguations of α interfere with the technical linguistic apparatus of $\mathrm{SAT}^{(N)} \wedge \mathrm{DISM}_\alpha$. Since $\mathfrak{B} \models \mathrm{DISM}_\alpha$, there must be an element $a \in \mathrm{dom}(\mathfrak{B})$ such that $a = t_\varphi^{\mathfrak{B}}$, where t_φ is the term describing φ. Hence, by lemma 6.2.7, $\mathfrak{B} \models \mathrm{dism}_\alpha[a]$. Furthermore, since $\mathfrak{B} \models \mathrm{SAT}^{(N)}$ and $\mathfrak{B} \models \varphi[\bar{a}]$ by lemma 6.2.4 $\mathfrak{B} \models \delta_\varphi[\bar{a}]$ where $\delta_\varphi(\bar{y}) := \mathrm{sat}^{(N)}(t_\varphi, \bar{y})$. Clearly then, the tuple $\langle a, \bar{a}\rangle$ satisfies the formula $\mathrm{dism}_\alpha(\nu) \wedge \mathrm{sat}^{(N)}(\nu, \bar{y})$ in $\mathfrak{B}$ and by assumption $\mathfrak{B} \models \varphi_0$ and therefore the formula

$$\psi_\alpha := \varphi_0 \wedge \exists\nu(\mathrm{dism}_\alpha(\nu) \wedge \exists\bar{y}(\mathrm{sat}^{(N)}(\nu, \bar{y})))$$

is satisfiable, as required.

Conversely, suppose that ψ_α is satisfiable. Then there exists a structure $\mathfrak{A}$ and a tuple $\langle a, \bar{a}\rangle$, such that $\mathfrak{A} \models \varphi_0$ and $\langle a, \bar{a}\rangle$ satisfies $\mathrm{dism}_\alpha(\nu) \wedge \mathrm{sat}^{(N)}(\nu, \bar{y})$ in $\mathfrak{A}$. Since $\mathfrak{A} \models \varphi_0$ and in particular $\mathfrak{A} \models \mathrm{DISM}_\alpha$ we have $a = t_\varphi^{\mathfrak{A}}$ for some φ such that φ is a disambiguation of α, by lemma 6.2.7. Also, since by assumption $\mathfrak{A} \models \mathrm{sat}^{(N)}[a, \bar{a}]$ we have $\mathfrak{A} \models \varphi[\bar{a}]$, by lemma 6.2.4. Thus, φ is a satisfiable disambiguation of α and so α is weakly satisfiable. □

6.4 Summary

In this chapter, we have demonstrated that weak satisfiability (with respect to *any* Underspecified Representation Language) is reducible to classical satisfiability. Specifically, given any Underspecified Representation α we have demonstrated how to engineer a formula $\psi_\alpha \in \mathcal{L}$ with the following property: α is weakly satisfiable if and only if ψ_α is classically satisfiable. This result gives us some idea of what the (expressive) limitations of underspecification are. It also suggests that the enterprise of looking for theorem provers for weak satisfiability is of little theoretical interest. We conclude this chapter by briefly discussing how theorem 6.0.8 relates to some of our earlier results regarding relative expressive power.

Recall that in chapter 5, we proved that: the Underspecified Representation Language $\mathcal{Q}$ can be 'transcribed' in $\mathcal{H}$ and visa versa (lemma 5.2.1 and lemma 5.2.2); and that $\mathcal{Q}$ can be transcribed in APL and visa versa (lemma 5.3.1 and theorem 5.3.6).[3] At first sight, theorem 6.0.8 might seem to subsume our earlier results; since, if we can 'reduce' any Underspecified Representation Language

[3]It is important to remember that we use the term 'transcription' in a special sense, defined on page 121.

to classical logic, then we must be able to 'reduce' any pair of Underspecified Representation Languages to one another (via classical logic if necessary)! However, in theorem 6.0.8 we prove that the Underspecified Representation α and the corresponding $\mathcal{L}$–formula ψ_α are 'equisatisfiable' (we did not show that they are transcriptions of one another). Trivially, for any formulas α and β, if α and β transcribe one another then α and β must be equisatisfiable; however, the converse of this statement is false. We therefore claim that the results in chapter 5 are not comparable to theorem 6.0.8. In the final chapter we evaluate our results in more detail.

Chapter 7

Conclusion

Statistics show that every 11 seconds a man is mugged here in New York. We are here today to interview him.

Ambiguity poses a real problem for Natural Language Processing. Unfortunately, there are many different sources of ambiguity and all natural languages are highly ambiguous. There are currently two approaches to the ambiguity problem: resolution and underspecification. Resolution aims to select the most likely reading(s) (usually from context); by contrast, the aim of underspecification is to encode the readings of ambiguous expressions in compact Underspecified Representations (enumerating readings upon need). Resolution is (notionally) the simpler option, but we argue that it is somewhat limited and therefore underspecification is worth investigating. An Underspecified Logic consists of two components: a formal language in which we can explicitly encode the various types of ambiguity (a so–called 'Underspecified Representation Language') and a framework for interpreting Underspecified Representations. We present these components separately since we believe that the issues of representation and interpretation are orthogonal.

Although there are currently eight Underspecified Representation Languages (presented in chapter 3), we argue that in fact there are only three types. We also investigate the best–known frameworks for interpreting the expressions of these languages: a recursive satisfaction definition (Partial Logic) and two non–recursive satisfaction definitions (based on strong and weak satisfiability respectively). However, we argue that Partial Logic is of little use, since it fails to capture the intuition intended by its authors, and in any case, partial satisfiability is

reducible to classical satisfiability. Therefore, we prefer non–recursive satisfaction definitions (that is, we prefer the meaning of any Underspecified Representation to be completely determined by the meanings of its set of disambiguations). With this interpretational scheme in mind, we are able to investigate the (relative) expressive power of the various Underspecified Representation Languages. In fact we argue that all current Underspecified Representation Languages have equivalent expressive power. Moreover, we establish that weak satisfiability is always reducible to classical satisfiability (regardless of the Underspecified Representation Language we choose).

Thus, we now have a sound understanding of the relationships between the various Underspecified Representation Languages and of the frameworks used to interpret them.

What is the significance of our contribution? The interest in mapping the current Underspecified Representation Languages into one another resides in the link this establishes between a mathematically very manageable Underspecified Representation Language (namely, $\mathcal{Q}$) and other more linguistically natural languages (for example, $\mathcal{H}$ and $\mathcal{R}$). To establish the expressive equivalence of two Underspecified Representation Languages $\mathcal{A}$ and $\mathcal{A}'$ we show that every formula in one of these languages can be 'translated' into a formula in the other. In translating between two languages it is imperative that the size of the translated formula should be polynomial in the size of the original formula. Without this restriction it would be trivial to map any formula of any current Underspecified Representation Language into an 'equivalent' formula of $\mathcal{Q}$ by disambiguating the original formula and then recreating the ambiguity by joining up the disambiguations obtained (using the ambiguation connective '?'). However, such a strategy is inefficient because of the so–called 'combinatorial explosion problem' and would therefore destroy the original motivation for using Underspecified Representations. The most convincing translation between two formulas $\varphi \in \mathcal{A}$ and $f(\varphi) \in \mathcal{A}'$ would be one in which every disambiguation of φ is the same as (or logically equivalent to) some disambiguation of $f(\varphi)$ and every disambiguation of $f(\varphi)$ is the same as (or logically equivalent to) some disambiguation of φ. However, we believe that there is no such translation between any of the languages $\mathcal{Q}$, $\mathcal{H}$ and APL (conjecture 5.1.3). We therefore use the (less severe) notion of 'transcription' in place of 'logical equivalence' to translate between the formulas of different Underspecified Representation Languages.

The final theorem of the thesis is that weak satisfiability is reducible to classical satisfiability. Our reduction of weak satisfiability to classical satisfiability provides some indication of where the limitations of underspecification lie (in terms of representation). Furthermore, this reduction suggests that the enterprise of designing theorem provers for weak satisfiability should be of little interest to the theorist. The proof of the final theorem requires us to axiomatise the notions of 'satisfaction' and 'is–a–disambiguation–of' within a 'formula algebra'. It is important to realise that the final theorem does not subsume our earlier results (that is, our maps between $\mathcal{Q}$ and $\mathcal{H}$ and between $\mathcal{Q}$ and APL) since 'transcription' is a stronger notion than 'equisatisfiability'. That is, for any Underspecified Representations α and β, if α transcribes β (and visa versa) then α and β are equisatisfiable; however, the converse is false.

There has been recent interest in mapping some of the current Underspecified Representation Languages into each other; in particular, Predicate Logic Unplugged has been mapped into the Constraint Language for Lambda Structures and visa versa (Koller et al., 2003) and Minimal Recursion Semantics has been mapped into the Constraint Language for Lambda Structures and visa versa (Niehren and Thater, 2003). However, these three languages (PLU, MRS and CLLS) all 'look' similar and we claim that they are merely notational variants of one another. We have presented the first back–and–forth map between two very different Underspecified Representation Languages, $\mathcal{H}$ and $\mathcal{Q}$ (and similarly between APL and $\mathcal{Q}$). The significance of this contribution is that it allows us to better understand the landscape of underspecification and, in particular, how the various Underspecified Representation Languages relate to one another. In practical terms, it means that in many cases, results obtained for one language can be interpreted within the context of another.

The focus of future research should be an evaluation of underspecification from a more practical perspective. In particular, the main question remains: is underspecification really more efficient than resolution? In addition, it would be useful to audit the various schemes for mapping natural language expressions onto Underspecified Representations. Currently, such systems are available for PLU (Blackburn and Bos, 1999a, pp.89–91), MRS (Copestake et al., 1995, sec.5, pp.13–24), CLLS (Egg et al., 2001, pp.22–27), UDRT (Reyle, 1993), QLF (Alshawi and Crouch, 1992) and Stores (Blackburn and Bos, 1999a, pp.70–82). But

it is difficult to compare these since each of them is written with a distinct Underspecified Representation Language in mind. A more theoretical issue is the relationship between strong satisfiability and classical satisfiability; we conjecture that strong satisfiability is reducible to classical satisfiability (an analogue of the final theorem). It would also be interesting to further our investigation of strong and weak satisfiability and, in particular, their complexity in various decidable fragments.

In this thesis, we have established that there is really only one sensible way to interpret Underspecified Representations (based on strong/weak satisfiability). Using this scheme, we have proved that all current Underspecified Representation Languages have the same expressive power and that the only sensible notion of ambiguous satisfiability is reducible to classical satisfiability anyway!

Bibliography

Abney, S. (1996). Part–of–Speech Tagging and Partial Parsing. In Young, S. and Bloothooft, G., editors, *Corpus–Based Methods in Language and Speech Processing*, pages 118–136. Kluwer Academic Press, Dordrecht.

Abney, S. (1997). Stochastic Attribute–Value Grammars. *Computational Linguistics*, 23:597–618.

Agresti, A. (1990). *Categorical Data Analysis*. John Wiley & Sons.

Allen, J. (1995). *Natural Language Understanding*. The Benjamin/Cummings Publishing Company Incorporated.

Alshawi, H. (1992). *The Core Language Engine*. MIT Press, Cambridge.

Alshawi, H. (1996). Underspecified First–Order Logics. In Peters, S. and van Deemter, K., editors, *Semantic Ambiguity and Underspecification*, number 15 in CSLI lecture notes, pages 145–158. CSLI, Stanford, CA.

Alshawi, H., Carter, D., Crouch, R., Pulman, S., Rayer, M., and Smith, A. (1992a). CLARE: A Contextual Reasoning and Cooperative Response Framework for the Core Language Engine. Technical Report CRC–028, SRI International, Cambridge.

Alshawi, H., Carter, D., Crouch, R., Pulman, S., Rayner, M., and Smith, A. (1992b). *CLARE: A Contextual Reasoning and Cooperative Response Framework for the Core Language Engine*. SRI International, Cambridge Computer Science Centre, Cambridge, UK. SRI Project 8468.

Althaus, E., Duchier, D., Koller, A., Mehlhorn, K., Niehren, J., and Thiel, S. (2000). An Efficient Graph Algorithm for Dominance Constraints. Submitted.

Aronoff, A. and Rees Miller, J., editors (2001). *The Handbook of Linguistics.* Blackwell Handbooks in Linguistics, number 8. Oxford, Blackwell.

Atwell, E. (1987). Constituent–Likelihood Grammar. In Garside, R., Leech, G., and Sampson, G., editors, *The Computational Analysis of English: A Corpus based Approach.* Longman, London.

Bangalore, S. (1997). Performance Evaluation of Supertagging for Partial Parsing. The 5th International Workshop on Parsing Technologies. Boston.

Bar Hillel, Y. (1964). *Language and Information.* Adison–Wesley.

Benello, J., Mackie, A. W., and Anderson, J. A. (1989). Syntactic Category Disambiguation with Neural Networks. *Computer Speech and Language*, 3:203–217.

Black, E. (1988). An Experiment in Computational Discrimination of English Word Senses. *IBM Journal of Research and Development*, 232:185–194.

Black, E., Jelinek, F., Laferty, J., Magerman, D. M., Mercer, R., and Roukos, S. (1993). Towards History–Based Grammars: Using Richer Models for Probabilistic Parsing. In *Proceedings of the 31st Annual Meeting of the Association for Computational Linguistics (ACL'93)*, pages 31–37.

Blackburn, P. and Bos, J. (1999a). Representation and Inference for Natural Language: A First Course in Computational Semantics. Available online at `http://www.comsen/.org`. vol.1.

Blackburn, P. and Bos, J. (1999b). Working with Discourse Representation Theory: An Advanced Course in Computational Semantics. Available online at `http://www.comsen/.org`. vol.2.

Blamey, S. (1986). Partial Logic. In Gabbay, D. and Guenthner, F., editors, *The Handbook of Philosophical Logic*, volume 3, pages 1–70. D. Reidel Publishing Company.

Bod, R. (1996). Data–Oriented Language Processing: An Overview. Technical Report LP–96–13, Institute for Logic, Language and Computation, University of Amsterdam.

Bod, R. (1998). *Beyond Grammar: An Experience–Based Theory of Language.* CSLI Publications, Stanford CA.

Bod, R., Scha, R., and Sima'an, K., editors (2003). *Introduction to Data–Oriented Parsing.* CSLI Publications, University of Chicago Press.

Bodirsky, M., Erk, K., Koller, A., and Niehren, J. (2001). Underspecified Beta Reduction. In *Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics (ACL'2001)*, Toulouse.

Booth, T. L. and Thomson, R. A. (1973). Applying Probability Measures to Abstract Languages. In *IEEE: Transactions on Computers*, volume C–22, pages 442–450.

Bos, J. (1995). Predicate Logic Unplugged. In *Proceedings of the 10th Amsterdam Colloquium*, pages 133–143.

Brill, E. (1995). Transformation–Based Error–Driven Learning and Natural Language Processing: A Case Study in Part–of–Spech Tagging. *Computational Linguistics*, 21:543–565.

Bronnenburg, W., Hunt, H. C., Landsbergen, S. P. J., Scha, R., Schoenmarkers, W., and van Utteren, E. (1979). The Question–Answering System PHLIQA1. *Natural Communication with Computers*, 2.

Brown, P. F., Della Pietra, S. A., Della Pietra, V. J., and Mercer, R. L. (1991). Word Sense Disambiguation using Statistical Methods. In *Proceedings of the 29th Annual Meeting of the Association for Computational Linguistics (ACL'91)*, volume 29, pages 264–270.

Cann, R. (1993). *Formal Semantics: An Introduction.* Cambridge Textbooks in Linguistics. Cambridge University Press, 1994 edition.

Carbonell, J. and Brown, R. (1988). Anaphora Resolution: A Multi–Strategy Approach. In *Proceedings of the 12th International Conference on Computational Linguistics (COLING'98)*, Budapest, Hungary.

Charniak, E. (1996a). *Statistical Language Learning.* MIT Press, Cambridge, Massachusetts, paperback edition.

Charniak, E. (1996b). Tree–Bank Grammars. In *Proceedings of the 13th National Conference on Artificial Intelligence (AAAI'96)*, pages 1031–1036.

Charniak, E., Hendrickson, C., Jacobson, N., and Perkowitz, M. (1993). Equations for Part–of–Speech Tagging. In *Proceedings of the 11th National Conference on Artificial Intelligence (AAAI'93)*, Menlo Park. MIT Press.

Chen, J. N. and Chang, J. S. (1998). Topical Clustering of MRD Senses based on Information Retrieval Techniques. *Computational Linguistics (Special Issue)*, 24(1).

Cherry, L. (1978). PART: A System for Assigning Word Classes to English Texts. In *AT&T Memorandum*. Bell Laboratories, Murray Hill, New Jersey.

Choueka, Y. and Lusignan, S. (1985). Disambiguation by Short Contexts. *Computers and the Humanities*, 19:147–158.

Church, A. (1956). *Introduction to Mathematical Logic*, volume 1. Princeton University Press, third edition.

Church, K. (1988). A Stochastic Parts Program and Noun Phrase Parser for Unrestricted Texts. In *Proceedings of the 2nd Conference on Applied Natural Language Processing*, Austin, Texas.

Church, K. W. and Patil, R. S. (1982). Coping with Syntactic Ambiguity or How to Put the Block in the Box on the Table. *Computational Linguistics*, 8:139–149.

Collins, M. J. (1996). A New Statistical Parser based on Bigram Lexical Dependencies. In *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics (ACL'96)*, pages 184–191.

Collins, M. J. (1997). Three Generative, Lexicalized Models for Statistical Parsing. In *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics (ACL'97) and Proceedings of the 8th Conference of the European Chapter of the Association for Computational Linguistics (EACL'97)*.

Cooper, R. (1983). *Quantification and Syntactic Theory*. The Syntactic Language Library, number 21, D. Reidel. Dordrecht.

Cooper, R., Crouch, R., van Eijck, J., Fox, C., van Genabith, J., Jaspars, J., Kamp, H., Pinkal, M., Poesio, M., Pulman, S., and Vestre, E. (1994). FraCaS, A Framework for Computational Semantics: Describing the Approaches. CWI Report: LRE 62–051.

Copestake, A., Flickinger, D., Malouf, R., Riehemann, S., and Sag, I. A. (1995). Translation using Minimal Recursion Semantics. In *Proceedings of the 6th International Conference on Theoretical and Methodological Issues in Machine Translation (TMI'95)*, Leuven, Belgium.

Copestake, A., Flickinger, D., Sag, I. A., and Pollard, C. (1999). Minimal Recursion Semantics: An Introduction. Draft.

Daelemans, W., Zavrel, J., Berck, P., and Gillis, S. (1996). MBT: A Memory–Based Part–of–Speech Tagger Generator. In *Proceedings of the 4th Workshop on Very Large Corpora (WVLC'96)*, pages 14–27.

Dagan, I. and Itai, A. (1990). Automatic Processing of Large Corpora for the Resolution of Anaphora References. In *Proceedings of the 13th International Conference on Computational Linguistics (COLING'90)*, volume 3, pages 330–332, Helsinki, Finland.

Dagan, I. and Itai, A. (1994). Word Sense Disambiguation using a Second Language Monolingual Corpus. *Computational Linguistics*, 20:563–596.

Dalrymple, M. (1993). *The Syntax of Anaphoric Binding.* Number 36 in CSLI lecture notes. CSLI Publications.

Dalrymple, M., Shieber, S. M., and Pereira, F. (1991). Ellipsis and Higher–Order Unification. *Linguistics and Philosophy*, 14:399–452.

Dempster, A. P., Laird, N. M., and Rubin, D. B. (1977). Maximum Likelihood from Incomplete Data via the EM Algorithm. *The Journal of the Royal Statistical Society*, 39:1–38.

DeRose, S. (1988). Grammatical Category Disambiguation by Statistical Optimization. *Computational Linguistics*, 14:31–39.

Dik, S. C. (1972). *Coordination: Its Implications for the Theory of General Linguistics.* North–Holland Publishing Company, second edition.

Dolan, W. B. (1994). Word Sense Ambiguation: Clustering Related Senses. In *Proceedings of the 15th International Conference on Computational Linguistics (COLING'94)*, pages 712–716, Kyoto, Japan.

Duda, R. O. and Hart, P. E. (1973). *Pattern Classification and Scene Analysis.* Wiley, New York.

Dummet, M. A. E. (1975). *Truth and other Enigmas.* Duckworth, London.

Edmonds, P. (2002). SENSEVAL: The Evaluation of Word Sense Disambiguation Systems. ELRA Newsletter 7(3). Currently available at `http://www.cs.unt.edu/řada/senseval/publications/senseval.pdf`.

Egg, M., Koller, A., and Niehren, J. (2001). The Constraint Language for Lambda Structures. *Journal of Logic, Language and Information*, 10(2):457–485. Special edition on Underspecification.

Egg, M., Neihren, J., Ruhrberg, P., and Feiyu, X. (1998). Constraints over Lambda Structures in Semantic Underspecification. In *Proceedings of the 17th International Conference on Computational Linguistics (COLING'98) and 36th Annual Meeting of the Association for Computational Linguistics (ACL'98)*, pages 353–359, Montreal Canada.

Enderton, H. B. (1972). *A Mathematical Introduction to Logic.* London Academic Press, New York.

Erk, K., Koller, A., and Niehren, J. (2000). Processing in the Constraint Language for Lambda Structures. *The Journal of Language and Computation*, 0(0):1–37. Oxford University Press.

Fernando, T. (1995). A Logical Connective for Ambiguity Requiring Disambiguation. In Underspecification, Events and More Dynamic Semantics, edited by J. Groenendijk.

Gale, W., Church, K., and Yarowsky, D. (1992a). Using Bilingual Materials to develop Word Sense Disambiguation Methods. In *Proceedings of the 3rd International Conference on Theoretical and Methodological Issues in Machine Translation (TMI'92)*, pages 101–112.

Gale, W. A., Church, K. W., and Yarowsky, D. (1992b). A Method for Disambiguating Word Senses in a Large Corpus. Technical report, AT&T Bell Laboratories, Murray Hill, New Jersey.

Galil, Z., Micali, S., and Gabow, H. (1986). An $\mathcal{O}(ev \log v)$ Algorithm for Finding a Maximal Weighted Matching in General Graphs. *SIAM Journal of Computing*, 15:120–130.

Gambäck, B. and Bos, J. (1998). Semantic–Head Based Resolution of Scopal Ambiguities. In *Proceedings of the 12th International Conference on Computational Linguistics (COLING'98) and Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics (ACL'98)*, pages 433–437.

Garey, M. and Johnson, D. (1979). *Computers and Intractability: A Guide to the Theory of NP–Completeness*. W. H. Freeman and Company.

Garside, R. (1987). The CLAWS Word–Tagging System. In Garside, R., Leech, G., and Sampson, G., editors, *The Computational Analysis of English: A Corpus–Based Approach*. Longman, London.

Ge, N., Hale, J., and Charniak, E. (1998). A Statistical Approach to Anaphora Resolution. In *Proceedings of the 6th Workshop on Very Large Corpora (WVLC'98)*, pages 161–170.

Green, G. (1996). Ambiguity Resolution and Discourse Interpretation. In Peters, S. and van Deemter, K., editors, *Semantic Ambiguity and Underspecification*, number 55 in CSLI lecture notes. CSLI, Stanford, CA.

Greene, B. B. and Rubin, G. M. (1971). Automatic Grammatical Tagging of English. Technical report, Brown University, Providence, RI.

Grice, H. P. (1975). Logic and Conversation. In Cole, P. and Morgan, J. L., editors, *Syntax and Semantics 3: Speech Acts*, pages 41–58. Academic Press, New York.

Guthrie, J. A., Guthrie, L., Wilks, Y., and Aidinejad, H. (1991). Subject Dependant Co–occurrence and Word Sense Disambiguation. In *Proccedings of the 29th Annual Meeting of the Association for Computational Linguistics (ACL'91)*, pages 146–152, Berkeley, California.

Guthrie, L., Pustejovsky, J., Wilks, Y., and Slator, B. M. (1996). The Role of Lexicons in Natural Language Processing. *Communications of the Association for Computing Machinery (ACM'96)*, 39:63–72.

Harper, K. E. (1957a). Contextual Analysis. *Mechanical Translation*, 4(3):70–75.

Harper, K. E. (1957b). Semantic Ambiguity. *Mechanical Translation*, 4(3):68–69.

Hayes, P. J. (1978). Mapping Input into Schemas. Technical Report 29, Department of Computer Science, University of Rochester.

Hindle, D. and Rooth, M. (1993). Structural Ambiguity and Lexical Relations. *Computational Linguistics*, 19:103–120.

Hirst, G. (1987). *Semantic Interpretation and the Resolution of Ambiguity*. Studies in Natural Language Processing. Cambridge University Press.

Hobbs, J., Stickel, M., and Martin, P. (1993). Interpretation as Abduction. *Artificial Intelligence*, 63(1 & 2):69–142.

Hobbs, J. R. (1983). An Improper Treatment of Quantification in Ordinary English. In *Proceedings of the 21st Annual Meeting of the Association for Computational Linguistics (ACL'83)*, pages 57–63, Cambridge, Massachusetts.

Hobbs, J. R. and Shieber, S. M. (1986). An Algorithm for Generating Quantifier Scoping. *Computational Linguistics*, 13:47–63.

Hodges, W. (1977). *Logic: An Introduction to Elementary Logic*. Penguin Books, 1991 edition.

Hodges, W. (1992). *Model Theory*. Number 42 in Encyclopedia of Mathematics and its Applications. Cambridge University Press.

Ide, N. and Véronis, J. (1990). Mapping Dictionaries: A Spreading Activation Approach. In *Proceedings of the 6th Annual Conference of the UW Centre for the New Oxford English Dictionary*, pages 52–64.

Ide, N. and Véronis, J. (1998). Word Sense Disambiguation: The State of the Art. *Computational Linguistics*, 24:1–40. An Introduction to the Special Issue on Word Sense Disambiguation.

Ioup, G. (1975). Some Universals for Quantifier Scope. In Kimball, J., editor, *Syntax and Semantics*, volume 4, pages 37–58. Academic Press, New York.

Jackendoff, R. (1972). *Semantic Interpretation in Generative Grammar*. MIT Press, Cambridge, Massachusetts.

Jaspars, J. (1997). Minimal Logics for Reasoning with Ambiguous Expressions. Technical Report 94, Universität des Saarlandes. CLAUS Report.

Jelinek, F. (1985). Markov Source Modelling of Text Generation. In Skwirzynski, J. K., editor, *The Impact of Processing Techniques on Communications*, volume E91 of *NATO ASI*, pages 569–598. Nijhoff, Dordrecht.

Jelinek, F., Lafferty, J., Magerman, D., Mercer, R., Patnaparkhi, A., and Roukos, S. (1994). Decision Tree Parsing using a Derivational Model. In *Proceedings of the Human Language Technology Workshop*, pages 272–277.

Jorgensen, J. (1990). The Psychological Reality of Word Senses. *Journal of Psycholinguistic Research*, 19:167–190.

Kamp, H. and Reyle, U. (1993). *From Discourse to Logic: Introduction to Model Theoretic Semantics of Natural Language, Formal Logic and Discourse Representation Theory*. Kluwer Academic Publishers, student edition.

Kaplan, A. (1950). An Experimental Study of Ambiguity and Context. *Mechanical Translation*, pages 36–46. Published, 1955.

Karlsson, F. (1990). Constraint Grammar as a Framework for Parsing Running Text. In *Proceedings of the 4th International Conference on Computational Linguistics (COLING'90)*, pages 168–173.

Kehler, A. (1993). A Discourse Copying Algorithm for Ellipsis and Anaphora Resolution. In *Proceedings of the 6th Conference of European Chapter of the Association for Computational Linguistics (EACL'93)*, pages 203–212, Utrecht.

Keller, W. (1986). Nested Cooper Storage: The Proper Treatment of Quantification in Ordinary Noun Phrases. In Reyle, U. and Rohrer, C., editors, *Natural Language Parsing and Linguistic Theory*, Studies in Linguistics and Philosophy, pages 432–437. Reidel.

Kennedy, C. and Boguraev, B. (1996). Anaphora for Everyone: Pronominal Anaphora Resolution without a Parser. In *Proceedings of the 10th International Conference on Computational Linguistics (COLING'96)*, pages 113–118, Copenhagen, Denmark.

Kilgarriff, A. (1993). Dictionary Word Sense Distinctions: An Enquiry into their Nature. *Computers and the Humanities*, 26:365–387.

Kilgarriff, A. (1997). What is Word Sense Disambiguation Good For. In *Proceedings of the NLP Pacific Rim Symposium'97*, Phuket, Thailand.

Kilgarriff, A. (1998). Gold Standard Datasets for Evaluating Word Sense Disambiguation Programs. *Computer Speech and Language*, 12(3).

Kilgarriff, A. and Palmer, M. (2000). Introduction to the Special Issue on SENSEVAL. *Computers and the Humanities*, 34(1–2):1–13.

Kleene, S. C. (1952). *Introduction to Metamathematics*. North–Holland, Amsterdam.

Koller, A. (1999). Constraint Languages for Semantic Underspecification. Master's thesis, University of Saarlandes.

Koller, A., Mehlhorn, K., and Niehren, J. (2000). A Polynomial–Time Fragment of Dominance Constraints. In *Proceedings of the 38th Annual Meeting of the Association for Computational Linguistics (ACL'2000)*, Hong Kong.

Koller, A. and Niehren, J. (1999). Scope Underspecification and Processing. A Reader for the European Summer School on Logic, Linguistics and Information 1999 (ESSLLI'99).

Koller, A., Niehren, J., and Thater, S. (2003). Bridging the Gap between Underspecification Formalisms: Hole Semantics as Dominance Constraints. In *Proceedings of the 11th Meeting of the European Chapter of the Association for Computational Linguistics (EACL'2003)*, Budapest, Hungary.

Koller, A., Niehren, J., and Treinen, R. (1998). Dominance Constraints: Algorithms and Complexity. In *Proceedings of the 3rd conference on Logical Aspects of Computatioal Linguistics*, Grenoble.

König, E. and Reyle, U. (1996). A General Reasoning Scheme for Underspecified Representations. In Hans J. Ohlbach and Reyle, U., editors, *Logic and its Applications: Festchrift for Dov Gabbay.* Kluwer Academic Press. Part 1.

Kooij, J. G. (1971). *Ambiguity in Natural Language: An Investigation of Certain Problems in Linguistic Description.* North–Holland Linguistic Series. North–Holland, Amsterdam.

Kripe, S. (1975). Outline of a Theory of Truth. *Journal of Philosophy*, 74:690–716.

Krovetz, R. (1997). Homonymy and Polysemy in Information Retrieval. In *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics (ACL'97) and the 8th Meeting of the European Chapter of the Association for Computational Linguistics (EACL'97)*, pages 72–79, Madrid, Spain.

Krovetz, R. and Croft, W. B. (1989). Word Sense Disambiguation using Machine Readable Dictionaries. In *Proceedings of the 12th Annual International Association for Computing Machinery (ACM) and Conference and Research and Development in Information Retrieval, (SIGIR'89)*, pages 127–136, Cambridge, Massachusetts.

Krovetz, R. and Croft, W. B. (1992). Lexical Ambiguity and Information Retrieval. In *Association for Computing Machinery (ACM): Transactions on Information Systems*, volume 10, pages 115–141.

Kupiec, J. (1992). Robust Part–of–Speech Tagging using a Hidden Markov Model. *Computer Speech and Language*, 6:225–242.

Kurtzman, H. S. and MacDonald, M. C. (1993). Resolution of Quantifier Scope Ambiguities. *Cognition*, 48:243–279.

Lakoff, G. P. (1971). Semantic Interpretation in Generative Grammar. In Steinberg, D. A. and Jakobovits, L. A., editors, *Semantics: An Interdisciplinary Reader in Philosophy, Linguistics, Anthropology, and Psychology.* Cambridge University Press, United Kingdom.

Landman, F. (1991). *Structures in Linguistics and Philosophy.* Number 45 in Studies in Linguistics and Philosophy. Kluwer Academic Publishers.

Langholm, T. (1996). How Different is Partial Logic. In Doherty, P., editor, *Partiality, Modality and Nonmonotonicity*, Studies in Logic, Language and Information, pages 3–43. CSLI Publications.

Lappin, S. (1996). *The Handbook of Contemporary Semantic Theory.* Oxford, Blackwell.

Lesk, M. (1986). Automatic Sense Disambiguation: How to Tell a Pine Cone from an Ice Cream Cone. In *Proceedings of the 1986 SIGDOC Conference*, pages 24–26, New York. Association for Computing Machinery (ACM).

Levow, G. (1997). Corpus–Based Techniques for Word Sense Disambiguation. Technical Report. AIM–1637.

Liddy, E. D. and Paik, W. (1993). Statistically Guided Word Sense Disambiguation. In *Proceedings of the AAAI Fall Symposium Series*, pages 98–107.

Luk, A. K. (1995). Statistical Sense Disambiguation with Relatively Small Corpora using Dictionary Definitions. In *Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics (ACL'95)*, pages 181–188. Cambridge, Massachusetts.

Magerman, D. M. (1995). Statistical Decision–Tree Models for Parsing. In *Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics (ACL'95)*, pages 276–283. Cambridge, Massachusetts.

Magerman, D. M. and Mitchell, M. P. (1991). PEARL: A Probabilistic Chart Parser. In *Proceedings of the European Chapter of the Association for Computational Linguistics Conference (EACL'91)*, Berlin, Germany.

Malinowski, G. (1993). *Many–Valued Logics.* Number 25 in Oxford Logic Guides. Oxford Science Publications.

Manning, C. D. and Schütze, H. (1999). *Foundations of Statistical Natural Language Processing.* The MIT Press, Cambridge, Massachusets, London, England.

Markov, A. A. (1913). An Example of Statistical Investigation in the Text of 'Eugene Onyegin' illustrating Coupling of 'Tests' in Chains. In *Proceedings of the Academy of Sciences*, volume 7, pages 153–162, St. Petersburg.

Martin, W. A., Church, K. W., and Patil, R. S. (1987). Preliminary Analysis of a Bredth–First Parsing Algorithm: Theoretical and Experimental Results. In Bolc, L., editor, *Natural Language Parsing Systems*. Sringer–Verlag, Berlin. Also published as an MIT LCS Technical Report TR–261.

May, R. (1977). *The Grammar of Quantification*. PhD thesis, Massachusetts Institute of Technology (MIT).

Mitamuran, T., Nyberg, E., Torrejon, E., Svoboda, D., Brunner, A., and Baker, K. (2002). Pronominal Anaphora Resolution in the KANTOO Multilingual Machine Translation System. In *Proceedings of the 2002 International Conference on Theoretical and Methodological Issues in Machine Translation (TMI'2002)*.

Mitkov, R. (1999). Multilingual Anaphora Resolution. *Computational Linguistics*, 14. Special issue on Anaphora Resolution in Machine Translation and Multilingual NLP.

Mohanty, A. K. (1983). Perceptual Complexity of Lexical, Surface Structure, and Deep Structure types of Ambiguous Sentences and Change in Heart Rate. *The Journal of Psycolinguistic Research*, 12(3):339–352.

Monz, C. (1999). Underspecified Theorem Proving with Different Kinds of Ambiguity. Master's thesis, Institute for Computational Linguistics (IMS) University of Stuttgart, Germany.

Monz, C. and de Rijke, M. (1998). A Tableaux Calculus for Ambiguous Quantification. In de Swart, H., editor, *Automated Reasoning with Analytic Tableaux and Related Methods, TABLEAX'98, LNAI 1397*, pages 232–246. Springer–Verlag.

Moran, D. B. (1988). Quantifier Scoping in the SRI Core Language Engine. In *Proceedings of the 26th Annual Meeting of the Association for Computational Linguistics (ACL'88)*, Buffalo, New York.

Muskens, R. (1989). Going Partial in Montague Grammar. In Bartsch, R., van Benthem, J., and van Boas, P., editors, *Semantics and Contextual Expression*, number 11 in Groningen–Amsterdam Studies in Semantics, pages 175–220. Foris Publications.

Muskens, R. (1996). Combining Montague Semantics and Discourse Representation. *Linguistics and Philosophy*, 19(1):143–186. Kluwer Academic Publishers.

Muskens, R. (1999). Underspecified Semantics. In *Reference and Anaphoric Relations*, volume 72 of *Studies in Linguistics and Philosophy*, pages 311–338. Kluwer Academic Publishers.

Nederhof, M. J. and Satta, G. (2002). Probabilistic Parsing Strategies. In Dassow, J., Hoeberechts, M., Jürgensen, H., and Wotschke, D., editors, *Descriptional Complexity of Formal Systems (DCFS): Pre–Proceedings of a Workshop*, pages 216–230, London, Canada. University of Western Ontario. Report No. 586.

Niehren, J., Pinkal, M., and Ruhrberg, P. (1997). On Equality up to Constraints over Finite Trees, Context Unification, and One–Step Rewriting. In *Proceedings of the 14th CADE*. Springer–Verlag.

Niehren, J. and Thater, S. (2003). Bridging the Gap between Underspecification Formalisms: Minimal Recursion Semantics as Dominance Constraints. In *Proceedings of 41st Meeting of the Association of Computational Linguistics (ACL'2003)*, pages 367–374.

Origgi, G. and Sperber, D. (2000). Evolution, Communication and the Proper Function of Language: A Discussion of Millikan in the light of Pragmatics and the Psychology of Misreading. In Curruthers, P. and Chamberlain, A., editors, *Evolution and the Human Mind: Language, Modularity and Social Cognition*. Cambridge University Press.

Pereira, F. and Schabes, Y. (1992). Inside–Outside Reestimation from Partially Bracketed Corpora. In *Proceedings of the 13th Annual Meeting of the Association for Computational Linguistics (ACL'92)*, pages 128–135.

Pereira, F., Tishby, N., and Lilian, L. (1993). Distributional Clustering of English. In *Proceedings of the 31st Annual Meeting of the Association for Computational Linguistics (ACL'93)*, pages 183–190, Ohio State University, Columbus, Ohio.

Pinkal, M. (1985). *Logic and the Lexicon: The Semantics of the Indefinite*. Studies in Linguistics. Kluwer Academic Publishers.

Poesio, M. (1994). Ambiguity, Underspecification and Discourse Interpretation. In *Proceedings of the 1st International Workshop on Computational Semantics (IWCS'94)*.

Poesio, M. (1996). Semantic Ambiguity and Perceived Ambiguity. In Peters, S. and van Deemter, K., editors, *Semantic Ambiguity and Underspecification*, number 15 in CSLI lecture notes. CSLI, Stanford, CA.

Pook, S. L. and Catlett, J. (1988). Making Sense of Searching. In *The Online Information Conference*. Published, 1998.

Quillian, M. R. (1969). The Teachable Language Comprehender: A Simulation Program and the Theory of Language. In *Communications of the Association for Computing Machinery (ACM)*, volume 12(8), pages 459–476.

Radford, A. (1997). *Syntax: A Minimalist Introduction*. Cambridge University Press. An abridged version of Syntactic Theory and the Structure of English.

Ratnaparkhi, A. (1996). A Maximum Entropy Model for Part–of–Speech Tagging. In *EMNLP 1*, pages 133–142.

Reinhart, T. (1983). *Anaphora and Semantic Interpretation*. Croom Helm, London.

Resnik, P. and Yarowsky, D. (1997). A Perspective on Word Sense Disambiguation Methods and their Evaluation. In *Proceedings of SIGLEX 97*, pages 76–86, Washington DC.

Reyle, U. (1993). Dealing with Ambiguities by Underspecification: Construction, Representation and Deduction. *The Journal of Semantics*, 10(2):123–179.

Reyle, U. (1995). On Reasoning with Ambiguities. In *Proceedings of the European Chapter of the 6th Meeting of the Association for Computational Linguistics (EACL'95)*, pages 1–8, Dublin.

Reyle, U. (1996). Co–indexing Labelled DRSs to Represent and Reason with Ambiguities. In Peters, S. and van Deemter, K., editors, *Semantic Ambiguity and Underspecification*, number 15 in CSLI lecture notes, pages 145–158. CSLI, Stanford, CA.

Riezler, S., King, T. H., Kaplan, R. M., Crouch, R., Maxwell, J. T., and Johnson, M. (2000a). Parsing the Wall Street Journal using a Lexical–Functional Grammar and Discriminative Estimation Techniques. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL'2002)*, Philadelphia, PA.

Riezler, S., Kuhn, J., Prescher, D., and Johnson, M. (2000b). Lexicalized Stochastic Modelling of Constraint–Based Grammars using Log–Linear Measures and EM Training. In *Proceedings of the 38th Annual Meeting of the Association for Computational Linguistics (ACL'2000)*, Hong Kong.

Samuelsson, C. and Voutilainen, A. (1997). Comparing a Linguistic and a Stochastic Tagger. In *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics (ACL'97)*, pages 246–253.

Schabes, Y., Roth, M., and Osborne, R. (1993). Parsing the Wall Street Journal with the Inside–Outside Algorithm. In *Proceedings of the 6th Conference of the European Chapter of the Association for Computational Linguistics (EACL'93)*, pages 341–347.

Schiehlen, M. (1997). Disambiguation of Underspecified Discourse Representation Structures under Anaphoric Constraints. Technical Report 188, The University of Stuttgart, Germany. Verbmoboil Project.

Schmid, H. (1994). Probabilistic Part–of–Speech Tagging using Decision Trees. In *International Conference on New Methods in Language Processing*, pages 44–49, Manchester, England.

Sima'an, K. (1996). Computational Complexity of Probabilistic Disambiguation by means of Tree–Grammars. In *Proceedings of the 16th International Conference on Computational Linguistics (COLING'96)*, pages 1175–1180.

Sima'an, K., Bod, R., Krauwer, S., and Scha, R. (1994). Efficient Disambiguation by means of Stochastic Tree Substitution Grammars. In *Proceedings of the International Conference on New Methods in Langauge Processing.*

Simons, P. (1999). Maccoll and Many–Valued Logic: An Exclusive Conjunction. *Nordic Journal of Philosophical Logic*, 1:85–90.

Spector, C. C. (1997). *Saying One Thing, Meaning Another*. Thinking Publications.

Stevenson, M. and Wilks, Y. (1999). Combining Weak Knowledge Sources for Sense Disambiguation. In *Proceedings IJCAI'99*.

Strzalkowski, T. (1995). Information Retrieval using Robust Language Processing. In *AAAI Spring Symposium on Representation and Acquisition of Lexical Information*, pages 104–111, Stanford.

Thomason, R. H. (1974). *Formal Philosophy: Selected Papers of Richard Montague*. Yale University Press, New Haven and London.

Thorsten, B. (2000). TnT: A Statistical Part–of–Speech Tagger. In *Proceedings of the 6th Applied Natural Language Processing Conference*, Seattle, WA.

Van Dalen, D. (1932). *Logic and Structure*. Universitext, Springer–Verlag, third edition.

Van Deemter, K. (1996). Towards a Logic of Ambiguous Expressions. In Peters, S. and van Deemter, K., editors, *Semantic Ambiguity and Underspecification*, number 55 in CSLI lecture notes. CSLI, Stanford, CA.

Van Deemter, K. (1998). Ambiguity and the Principle of Idosyncratic Interpretation. *The Journal of Semantics*, 15(1):5–36.

Van Eijck, D. J. N. (1998). The Logic of Syntax Trees.

Van Eijck, D. J. N. and Jaspars, J. (1996). Ambiguity and Reasoning. Technical Report ISSN 0169–118X,CSR9616, Centrum voor Wiskunde en Informatica (CWI).

Van Fraassen, B. C. (1966). Singular Terms, Truth–Value Gaps and Free Logic. *The Journal of Philosophy*, 63:481–495.

Van Genabith, J. and Crouch, R. (1996). F–Structures, QLFs and UDRSs. Technical Report CA–2196, Dublin City University, School of Computer Applications.

Van Lehn, K. A. (1978). Determining the Scope of English Quantifiers. Technical Report AI–TR–483, Artificial Intelligence Laboratory, MIT, Cambridge, Massachusetts.

Van Orman Quine, W. (1954). Quantification and the Empty Domain. *The Journal of Symbolic Logic*, 19(3).

Viterbi, A. (1967). Error Bounds for Convultional Codes an Asymptotically Optimum Decoding Algorithm. In *IEEE Transactions on Information Theory*, volume IT13, page 260.

Voutilainen, A. (1995). A Syntax Based Part–of–Speech Analyser. In *Proceedings of the European Chapter of the Association for Computational Linguistics (EACL'95)*.

Walker, D. (1987). Knowledge Resource Tools for Accessing Large Text Files. In Nirenberg, S., editor, *Machine Translation: Theoretical and Methodological Issues*. Cambridge University Press.

Weaver, W. (1949). Translation. In Locke, W. N. and Booth, A. D., editors, *Machine Translation of Languages*, pages 15–23. John Wiley & Sons, New York. Published, 1955.

Weischedel, R., Meteer, M., Schwartz, R., Ramshaw, L., and Palmucci, J. (1993). Coping with Ambiguity and Unknown Words through Probabilistic Models. *Computational Linguistics*, 19:359–382.

Wilks, Y. A. and Fass, D. (1990). Preference Semantics: A Family History. Technical Report MCCS–90–194, Computing Research Laboratory, New Mexico State University, Las Cruces, New Mexico.

Willis, A. and Manandhar, S. (1999). The Availability of Partial Scopings in Underspecified Semantic Representation. In *Proceedings of the 3rd International Workshop of Computational Semantics*, Tilburg, The Netherlands.

Yarowsky, D. (1992). Word–Sense Disambiguation using Statistical Models of Roget's Categories Trained on Large Corpora. In *Proceedings of the International Conference on Computational Linguistics (COLING'92)*, pages 454–460.

Yarowsky, D. (1993). One Sense per Collocation. In *Proceedings of ARPL Human Language Technology Workshop*, pages 266–271, Princeton, New Jersey.

Yarowsky, D. (1995). Unsupervised Word Sense Disambiguation Rivaling Supervised Methods. In *Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics (ACL'95)*, pages 189–196, Cambridge, Massachusetts.

Zernik, U. (1991). Train1 vs Train2: Tagging Word Senses in a Corpus. In *Proceedings of Intelligent Text and Image Handling (RIAO'91)*, pages 567–585, Barcelona, Spain.