CANADIAN THESES

THÈSES CANADIENNES

NOTICE

The quality of this microfiche is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Previously copyrighted materials (journal articles, published tests, etc.) are not filmed.

Reproduction in full or in part of this film is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30.

AVIS

La qualité de cette microfiche dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

Les documents qui font déjà l'objet d'un droit d'auteur (articles de revue, examens publiés, etc.) ne sont pas microfilmés.

La reproduction, même partielle, de ce microfilm est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30.

THIS DISSERTATION HAS BEEN MICROFILMED EXACTLY AS RECEIVED

LA THÈSE A ÉTÉ MICROFILMÉE TELLE QUE NOUS L'AVONS REÇUE

The University of Alberta

QUANTIFIER SCOPING IN INITIAL LOGICAL TRANSLATIONS OF ENGLISH SENTENCES

by

Sven O. Hurum

A thesis
submitted to the Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree
of Master of Science

Department of Computing Science

Edmonton, Alberta
Spring, 1987

ISBN 0-315-37757-7

THE UNIVERSITY OF ALBERTA

RELEASE FORM

NAME OF AUTHOR: Sven O. Hurum

TITLE OF THESIS: Quantifier Scoping in Initial Logical Translations of English Sentences

DEGREE FOR WHICH THESIS WAS PRESENTED: Master of Science

YEAR THIS DEGREE GRANTED: 1987

(Signed) Sven Hurum .....

Permanent Address:
10711 Saskatchewan Drive,
Apartment 906,
Edmonton, Alberta,
Canada T6E 4S4

Dated: March 16, 1987

THE UNIVERSITY OF ALBERTA

FACULTY OF GRADUATE STUDIES AND RESEARCH

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies and Research, for acceptance, a thesis entitled **Quantifier Scoping in Initial Logical Translations of English Sentences** submitted by **Sven O. Hurum** in partial fulfillment of the requirements for the degree of **Master of Science**.

L.A. Schubert
.....
Supervisor

Randy Goebel
.....

.....

Matthew S. Dyer
.....

Date: March 16, 1987

Abstract

The aim of this study has been to extend a general-purpose natural language understanding system currently being developed in this department to handle ambiguities of "quantifier scope". Quantifiers, such as "some" and "every", and other logical operators such as coordinators, negation and adverbs, interact with each other to give rise to various types of ambiguity. Such ambiguities are commonly represented in terms of the relative scopes of the operators.

Particular attention is given in this study to the problems associated with the logical representation and scoping of indefinite noun phrases. Evidence is presented that giving indefinites a "referential" interpretation cannot account for their immunity to certain constraints on scoping. Rather, it is argued that a major distinction needs to be made between the scoping of existential and distributive quantifiers.

A scoping algorithm is presented which generates the set of valid scoped readings of a sentence and which places these readings in an approximate order of preference. Logically redundant readings are removed as they are encountered during the process of scoping. The ordering procedure makes use of a set of domain-independent scoping heuristics which are based on information about the lexical type of each operator and on structural relations between pairs of operators. Some general problems regarding the design of scoping algorithms are discussed and some extensions to the current work are suggested.

Acknowledgements

First and foremost, I would like to thank Dr. Len Schubert for his encouragement and careful guidance of this work, and for the many Friday afternoons we spent discussing various aspects of the material covered in this thesis. It has been a most rewarding experience to have been able to spend the amount of time that I have with him, sharing his insights into problematic issues in computational linguistics, as well as in more general areas. The benefits I have received from this are only partly reflected in the production of this thesis.

I would also like to thank the other members of my thesis committee - Dr. Matthew Dryer (Linguistics), Dr. Renee Elio and Dr. Jeff Pelletier - for their valuable comments and suggestions, both during and after the exam. I am also indebted to the members of the Logical Grammar Study Group for their insightful comments on part of this work, as well as for introducing me to some of the current issues in grammatical theory and linguistic semantics.

Finally, I would like to thank the Department of Computing Science for supporting me for two semesters in the form of a Graduate Research Award and the Faculty of Graduate Studies and the National Science and Engineering Research Council of Canada for additional financial support.

Table of Contents

Page

Chapter 1: Quantifier Scope Ambiguities 1
1.1 Examples of Quantifier Scoping 2
1.2 Two Tests for Ambiguity 5
1.3 Model Theory and Ambiguity 6
1.4 Handling Quantifier Scope Ambiguities 9

Chapter 2: Some Theoretical Appoaches to Quantifier Scoping 13
2.1 May's Transformational Theory 13
2.2 Quantifier Scope and Bound Anaphora 16
2.3 Mapping Rules and Filters 18
2.4 Combinatoric Features 23
2.5 Branching Quantification 25
2.6 Generalized Scope Relations 27
2.7 Referential and Quantificational Indefinites 28

Chapter 3: The Scoping of Indefinite and Definite Noun Phrases 31
3.1 Non-Specific, Specific and Referential Indefinites 31
3.2 The Ability to "Escape" from Scope Islands 33
3.3 The Verb Phrase Deletion Constraint 38
3.4 Non-Restrictive Relative Clauses 41
3.5 *One* Anaphora 42
3.6 Type I and Type II Quantifiers 44
3.7 Bound and Unbound Pronouns 46
3.8 Referential and Attributive Definites 49
3.9 Specific Indefinites: A Lexical Class? 53
3.10 A Uniform Interpretation of Indefinites and Definites 57

Chapter 4: Scoping Heuristics 67
4.1 Structural Relations 68
4.2 Horizontal Scoping: The Hierarchies of Ioup and van Lehn 70
4.3 Scope Ordering Weights: Introduction 74
4.4 Quantifiers and Negation 76
4.5 Horizontal Scoping of Two Quantifiers 78
4.6 Vertical Scoping of Quantifiers 86
4.7 Coordinators and Negation 93
4.8 Coordinators and Quantifiers: Horizontal Scoping 95
4.9 Coordinators and Quantifiers: Vertical Scoping 97

Page

Chapter 5: Algorithm and Program 103
5.1 Initial Logical Translations 103
5.2 Comparing Operators 106
5.6 Combining Scoping Weights 109
5.4 Quantifier Scoping Algorithms: Background 111
5.5 The Scoping Program 117
5.6 Program Output 119
5.7 Conclusion 127

Bibliography 134

Appendix I : Examples of Scoping Weights 139
Appendix II : Tables of Scoping Weights 146
Appendix III : Output of Test Data 155
Appendix IV : Scoping Program 198

List of Tables

Page

Table 1.1: Some Logical Operators in English 1

Table 4.1: Horizontal Relations 79

Table 4.2: The Effect of Horizontal Relations on Scoping 81

Table 4.3: Vertical Relations 86

Table 5.1: Syntax of Initial Logical Translations 105

Table 5.2: Sentences used as Test Data 126

Table A2.1: Scoping of Quantifiers with Negation 146

Table A2.2: Horizontal Scoping of Two Quantifiers 149

Table A2.3: Scoping of Quantifiers Inside Prepositional Noun Complements 150

Table A2.4: Input File *Weights* 151

Table A2.5: Input File *Relations* 153

Table A2.6: Input File *Ratios* 154

Chapter 1

Quantifier Scope Ambiguities

Quantifier scoping is a method of dealing with a class of ambiguities created by the interaction of quantifiers with each other and with certain other logical operators such as coordinators, negation and adverbs. Some examples of the different types of logical operator which are frequently encountered in the English language are listed in Table 1.1. [1] The ambiguities which result from the interaction of such operators have traditionally been represented in terms of the relative scopes of the operators. In the section which follows, some examples will be given which illustrate the types of ambiguity which can occur and the usefulness of the scoping formalism in representing them.

The difference between "scoped" and "unscoped" operators should first be clarified. In the initial logical translations of English sentences used in this study, some operators, such as those corresponding to verbs and connectives, are already "scoped" and their position in the logical form is therefore fixed. Other operators, most notably quantifiers and coordinators, are "unscoped" and therefore need to be moved into their final positions. At the present time, the first four types of operator in table 1.1 are treated as unscoped. [2]

1. Quantifiers	:	*some, every, few, ...*
2. Coordinators	:	*and, or*
3. Negation	:	*not*
4. Temporal adverbs	:	*sometimes, always, often, ...*
5. Other Adverbs	:	*probably, quickly, ...*
6. Adjectives	:	*probable, possible, ...*
7. Tense	:	PRES, PAST, ...
8. Aspect	:	PROG, PERF, ...
9. Modals	:	*can, should, ...*
10. Verbs	:	*seek, believe, say, ...*
11. Connectives	:	*if-then, after, because, ...*

Table 1.1 : Some Logical Operators in English

1.1 Examples of Scope Ambiguities

A simple ambiguity involving two quantifiers is shown in (1) followed by the two standard scoped readings in (2) and (3). The readings are represented in an informal first order predicate logic (FOPL) with restrictions on quantifiers. [3]

(1) Everyone loves someone
(2) (Vx:person (Ey:person [x loves y]))
(3) (Ey:person (Vx:person [x loves y]))

These two readings are by no means uncontroversial. It has been argued by some people that (3) is logically redundant since it entails (2), and should therefore be accounted for by pragmatics (eg. Kempson & Cormack 1981), and by others that a third reading should be added in which *someone* is interpreted as a "referential" term (Fodor & Sag 1982). Referential terms are analogous to demonstratives, such as *that*.

The interaction between two numeral quantifiers introduces at least one or two further ambiguities. In addition to the two standard scoped readings of

(4) Two boys kissed three girls

in which the set of boys depends on the girl and vica versa, there is at least one distinct additional reading in which there is only one set of boys and one of girls. Kempson & Cormack describe two such readings: the "complete group" reading in which each of the boys kissed each of the girls and the "incomplete group" reading in which some combination of the boys kissed the girls. An empirical study by Gil (1982) suggests that the group readings are in fact favoured over the standard ones for mixed numeral sentences. Several of the proposals which have been made regarding the representation of such readings will be discussed in

chapter two. The logical representation of indefinites used in this study will be described in section 3.10.

The interaction of quantifiers with verb negation and with coordinators is shown in (5)-(7) and (8)-(10), respectively. Example (5) may mean that there is some book that John didn't buy (6) or that John bought no books (7). The two readings of (8) should be clear.

(5) John didn't buy a book
(6) (Ex:book ¬[John bought x])
(7) ¬(Ex:book [John bought x])

(8) Everyone likes Pam or Betty
(9) (Vx:person [[x likes Pam] v [x likes Betty]])
(10) [(Vx:person [x likes Pam]) v (Vx:person [x likes Betty])]

The interaction of quantifiers with opaque operators is more complex and there is evidence that the standard scope treatment may not adequately cover the different types of ambiguity (eg. Fodor 1970; Saarinen 1980; Enc 1981). [4] Opacity is usually defined by the failure of the substitutivity of expressions with identical denotations. [5] For example, the following argument will be invalid if the owner of the car is opaque to John:

(11) John wants to meet the owner of this car
(12) Bill is the owner of this car
(13) John wants to meet Bill

Indefinites show similar interactions with opaque operators. Sentence (14) contains an indefinite inside an opaque context created by the verb "want". The two scoped readings are shown in (15) and (16).

(14) John wants to marry a blonde
(15) (Ex:blonde [John wants [John marry x]])
(16) [John wants (Ex:blonde [John marry x])]

In the first reading we say that John holds an attitude towards a particular individual whereas

in the second he has only a description "in mind". Widening the scope of the quantifier makes it "specific" relative to the opaque context it scopes outside.

The scoping of quantifiers with tense is quite similar. Example (17) may refer to the girl who is prettiest at the time of utterance (18) or at some time in the future (19).

(17) John will kiss the prettiest girl
(18) (the x: prettiest-girl (will [John kiss x]))
(19) (will (the x: prettiest-girl [John kiss x]))

Quantifiers may scope relative to temporal adverbs (20) and to other adverbs such as manner adverbs (23).

(20) Someone always comes late
(21) (Ex:person (always [x comes late]))
(22) (always (Ex:person [x comes late]))

(23) John quickly lit all the lamps
(24) [John λy(Vx:lamp [y (quickly (lit x))])]
(25) [John (quickly λy(Vx:lamp [y (lit x)]))]

Example (20) may mean that the same person always comes late (21) or that there is always someone who comes late (22). Example (23) may mean that John lit each lamp quickly (24) or that the process of lighting all the lamps was fast (25).

There is no general consensus on how the ambiguities shown above should be handled. One somewhat controversial topic has been whether sentences containing mixed quantifiers are *semantically* ambiguous, meaning that the different scoped interpretations of such sentences express different propositions, or whether the different interpretations should be accounted for by *pragmatics*. This question will now be briefly considered first, in terms of two commonly used tests for ambiguity, and second, in relation to some recent work in linguistic model theory.

1.2 Two Tests for Ambiguity

Although there is no universally accepted test for determining whether or not a sentence is semantically ambiguous, a number of tests have been proposed and are at least of some use in helping to identify different types of ambiguity. Two such tests will be described here. The first test is simply that a sentence is semantically ambiguous if there exists a context (state of affairs) in which it may have both a true and a false interpretation (eg. Saarinen 1980). This may be considered to be a minimal criterion for ambiguity since what the test does is to determine whether or not a sentence *can* express two or more distinguishable propositions. Probably all sentences containing mixed quantifiers pass this test, even when one of the readings entails the others. For example,

(26) Someone loves everyone

could be interpreted as true by one person, who obtains the distributive reading, but false by a second person, who understands the sentence to require that the same, or perhaps a particular, person loves everyone.

A more stringent test is the *verb phrase anaphora test* introduced by Zwicky and Sadock and used by Kempson & Cormack (1981) to analyze mixed quantifier sentences. This test requires that an anaphoric expression, such as a VP pro-form, must be given the same interpretation as its antecedent if the antecedent is semantically ambiguous. This test works well for sentences containing syntactic ambiguities. If

(27) Mary likes entertaining guests

is followed by "and so does Bill" then the VP pro-form must have the same interpretation as its antecedent. At the other extreme, this test also works well for cases of clear vagueness such as

(28) The painter has done the sitting-room and the carpenter has too

where a "crossed" interpretation may easily be obtained (Kempson, 1977:130). What this test does is show whether the different possible understandings of an expression can share a common interpretation. For example, in (28) the common interpretation given to *do* is approximately that it means to accomplish a task.

Sentences having mixed quantifiers appear to be borderline cases. If one of the readings entails the other, as in

(29) John thinks that everyone loves someone and so does Bill

it may be argued that it is possible to obtain the crossed interpretation, in which John and Bill assign different scope orderings to the two quantifiers. Zwicky & Sadock consider this to be a drawback to their test, but Kempson & Cormack use this as evidence that mixed quantifier sentences fail the test. The latter also claim that sentences with two plural numerals fail this test even when different interpretations do not entail each other (1981:268).

1.3 Model Theory and Ambiguity

The difference between semantic and pragmatic ambiguity will now be considered in relation to model theory. A brief description will first be given of a modified version of Montague's Universal Grammar. The model consists of an interpretation function and an *index* which contains certain parameters which allow a sentence to be interpreted relative to the context of utterance. In addition, there is a *domain of discourse* which is independent of the model but which is required for the interpretation.

In Montague's original model, the interpretation of an expression involved a single phase, mapping the expression into its denotation at each index, the denotation being a truth

value in the case of a sentence. However, following the work of Kaplan, the index is now sometimes split into two parts, allowing a distinction to be made between *meaning* and *sense* (see below). A sentence is first interpreted relative to the "Context of Use" (C) and then relative to the "Circumstance of Evaluation" (I). The indexical parameters associated with C and I are:

Context of Use (C)	:	<s, a, wc, tc, ind>
Circumstance of Evaluation (I)	:	<w,t>

The first index contains five coordinates: the speaker, addressee, world and time of utterance and *ind*, an indicator of the speaker's intended referent, introduced by Kaplan to handle the gesture which often accompanies the use of demonstratives. The second index contains the *world* and *time* parameters which are recursively used in the evaluation of a proposition. The meaning of an expression is then defined to be a function from contexts to senses and the sense of an expression a function from the world and time of evaluation to its denotation. This may be summarized as:

(30) Given an expression α and an interpretation function f then

the meaning of α is	$f(\alpha)$
the sense of α is	$f(\alpha)$ (C)
the denotation of α is	$f(\alpha)$ (C) (I)

It is claimed that making this distinction between meaning and sense is necessary in order to allow propositions to be the objects of mental attitudes. For example, the following two sentences

(31) John thinks that I am happy
(32) John thinks that you are happy

may both express the fact that John thinks Bill is happy. In order for the proposition expressed by "Bill is happy" to be the object of John's attitude, the pronouns *I* and *you* must

be interpreted first and their denotations passed to the proposition. ' Expressions such as *I*, *you* and *here*, whose denotations are determined solely by C, are known as *indexicals*. Indexicals therefore have a "constant sense" although as Kaplan points out they are directly referential and it is misleading to think of them as having a sense. Following the above terminology, Enc defines an expression to be semantically ambiguous if its denotation depends on I but not on C and to be pragmatically ambiguous if the converse is true. ' However, it is hard to draw any firm line between indexical and non-indexical or between context and domain of discourse.

First, the difference between indexical and non-indexical will be considered. The most clearly indicated indexicals are the speaker, addressee, time and possibly place of discourse. Enc refers to these as *utterance sensitive indexicals* since they are solely dependent on the discourse setting. Enc also proposes a class of *freely referring indexicals* which require the contextual parameter *ind* to be used to specify their denotation. Kaplan originally proposed that *ind* be used as a parameter to allow demonstratives accompanied by a gesture to be included among the standard indexicals. However, Enc shows that deictic pronouns and also adverbs of time and place (eg. *then* and *there*) should also be treated as indexicals and likewise anaphoric occurrences of such expressions since it should not matter if the referents are introduced through perception or through language. [10] Some recent claims have also been made that definite and indefinite NPs should be given indexical interpretations (Wettstein 1981; Fodor & Sag 1982). The evidence for this proposal and some of the difficulties of making an indexical/non-indexical distinction for definites and indefinites will be discused in chapter three.

The difference between context and domain of discourse is also not that clear. The context, as well as the domain of discourse, may vary from word to word. Enc (p.83) shows that the two occurrences of *senior* in

(33) Every senior will chase every senior

may have different denotations in a case in which the context has made it clear that seniors from two different colleges are being referred to. [11] Therefore, it is questionable whether a useful distinction can be made between context and domain of discourse. [12] Consequently, it is not all that clear that a special case should be made for a class of "indexicals"; it could be argued instead that the context plus the domain of discourse should be applied to every word (or in some case to parts of words). An expression could then be defined to be semantically ambiguous, in line with the first test described in the previous section, if it can receive two distinct interpretations in the same context and domain of discourse. In other words, Enc's definition can now be equated with the criterion provided by the first test. According to this test, as we have seen, mixed quantifier sentences are semantically ambiguous. [13] For the purposes of this study, it will be assumed that different quantifier scope interpretations express distinct propositions.

1.4. The Handling of Quantifier Scope Ambiguities

In the next chapter, a number of different approaches to the handling of mixed quantifier sentences will be described. At one extreme, quantifier scope ambiguities may be treated as structural ambiguities, giving rise to distinct phrase markers (parse trees). This approach is taken by both Montague and May (see section 2.1). Another general approach is to have the parser generate "initial" logical translations which can then be used to give rise to the different scoped interpretations. This approach is advocated by Kempson & Cormack (section 2.3) and is also used, though in a different way, in this study. Webber also mentions using a general initial logical representation which can later be resolved into more specific, scoped interpretations (section 2.4). Some more radical proposals which have been made are that indefinites should not be scoped but be handled by pragmatics (see section 2.5), that indefinites should be given an indexical interpretation (section 2.7) and that scope relations

should be represented using explicit "general scope relation" operators, because linear and partially ordered representations are inadequate.

In chapter three, a discussion will be given of some of the problems involved in handling the scoping of indefinites and, to a lesser extent, of definites. In the final two chapters, the approach to quantifier scoping employed in this study will be described in detail. In chapter four, an attempt will be made to derive a set of domain-independent heuristics which may be used to help select preferred scope orderings. In chapter five, the algorithm currently being used will be described and some of the general problems confronting the design of scoping algorithms will be discussed. The algorithm is not yet completely implemented and so some possible improvements and extensions are discussed.

Footnotes

1. Several points should be noted here. (1) The term "logical operator" refers to the logical counterparts of the English words shown in the table. As a general rule, English words will be written in lower case and logical operators in upper case. Where this rule is not followed, the context should make it clear in which sense a word is being used. (2) In this study, the term "quantifier" will be taken to stand for "quantificational NP". Temporal adverbs are sometimes referred to as "quantificational adverbs" because of their close parallels with quantificational NPs, and therefore it might also be possible to use "quantifier" in a more general sense. Following the terminology of Barwise & Cooper (1981), whole NPs act as quantifiers and the term "determiner" is reserved for words such as *some* and *each*. While this distinction will be adhered to in principle, the term "quantifier" will sometimes be used loosely to refer to quantificational determiners where it should be understood that it is the whole NP prefixed by the determiner which acts as the quantifier. (3) Since operators such as coordinators and negation also require scoping, it would perhaps be more accurate to use the more general term "scoping", rather than "quantifier scoping". However, the latter is commonly used as a general term for the problem of scoping and in this study the predominant emphasis will be on the scoping of quantificational NPs.

2. The decision as to where to draw the line between scoped and unscoped operators in the initial logical translations is not well-defined and some of the difficulties regarding this will be discussed in chapters four and five. At the present time, the first four classes of operator listed in table 1.1 are treated as unscoped by the scoping program and the others as scoped. Clearly, it would be preferable to have a consistent treatment of adverbs and therefore this classification needs to be improved upon.

3. Sentential forms (which express propositions) are written in infix notation and functions in prefix notation. The syntax used for the initial logical translations is given in section 5.1.

4. One argument in favour of the scope treatment is the observation that the number of readings generally correlates with the number of opaque operators embedding a quantifier. For example, Fodor quotes Bach (Fodor 1970:72) on the specific/non-specific ambiguity relative to opaque contexts: "I do not believe it is possible to explain such ambiguities except by means of the notion of scope, because we find a systematic relationship between the number of interpretations and the number of embedded sentences ...".

5. Fodor (1970) also considers contexts which do not permit "existential generalization" to be opaque. By this criterion, verb negation and conditional operators create opaque contexts for indefinites.

6. The VP anaphora test is passed by sentences containing lexically ambiguous terms such as *John lives near the bank*, where *the bank* may have two lexical entries, one for a commercial bank and one for the side of a river. It might be argued in this case that sentences containing lexical ambiguities express different propositions and in fact it is possible to treat lexical ambiguities as "structural", that is, as giving rise to separate phrase markers. Therefore, this should not invalidate the use of the test for propositional, or semantic, ambiguity. However, it is generally very difficult to draw a line between lexical and non-lexical ambiguities. For example, *John likes the museum and so does Bill* could arguably be used to mean that John likes the building and Bill likes the exhibits, the term *museum* being given the broad interpretation of building plus exhibits.

7. The version described here is based on the discussion in chapter one of Enc (1981).

8. The meaning of the sentence *You are happy* is that, in any given context, the person who is being addressed is happy. Given a context in which Bill is being addressed, the proposition expressed by the sentence is *Bill is happy*.

9. Note that the term *pragmatic ambiguity* could also be used in a completely different way, to describe ambiguities related to the way in which a sentence is used and which are largely independent of the proposition expressed by the sentence. This term might be used to refer to sentences which are ambiguous with respect to implicatures, presuppositions, illocutionary force, intonation or stress etc. In some cases these sorts of ambiguities can be very clearcut, as opposed to "vague", for example the ambiguities related to which word receives constrastive stress.

10. Enc (1981) goes even further to propose that common nouns and tensed verbs should also be treated as indexicals. That is, their interpretation should depend on C but not on I. As evidence for this she shows that nouns must be interpreted independently of both each other and of the time and world of evaluation. Enc proposes that the denotation of common nouns should be determined by *ind* and be passed to the proposition in the same way as other indexicals. However, this proposal runs into problems in sentences containing nested tense or opaque operators and at the very least requires some modification.

11. Even the "purest" indexical, *I*, may change denotations within the same sentence. An example is *The book I am reading, which I bought (added Harry), would interest you*. The same is true of proper names. There are contexts in which it would be appropriate to say *John borrowed the book from John*.

12. Barwise & Perry (1983:chapter 2) also make use of Kaplan's two phased interpretation, but divide the "context" into three parts: the discourse situation (DS), speaker connections (SC) and resourse situations (RS). The first of these specifies the "indexicals", namely the speaker, addressee, time and place, and the latter two are used to specify "referential" terms, which include proper names, demonstratives, common nouns, and tensed verbs etc.

13. The close similarities between quantifier scope ambiguities and ambiguities of bound anaphora may perhaps be used to support the claim that the former are semantic and not pragmatic. As will be discussed in section 2.2, quantifier scope can be viewed as an abbreviated form of anaphora. However, ambiguities of anaphora are more clearcut and pronounced and sentences containing ambiguities of anaphora such as *John showed each person his room, and so did Bill* will be more likely to pass the verb phrase anaphora test.

Chapter 2

Some Theoretical Approaches to Quantifier Scoping

In this chapter some recent theoretical approaches to the logical representation and scoping of quantifiers will be described. Particular emphasis will be placed on two issues: (a) the way in which scope ambiguities are represented, and to what extent the representation method being used allows "gradual" disambiguation and (b) the computational feasibility of the approach, and in particular how well the approach allows heuristics to be used to select preferred readings. In chapter five some implementations of quantifier scoping algorithms will be described.

2.1. May's Transformational Theory

In his 1977 thesis, May attempted to extend the "trace theory" version of Universal Grammar (Chomsky 1976) to incorporate quantifier scoping. Very briefly, Universal Grammar is held to specify general principles which underlie the "core grammars" of particular languages. Core grammars consist of a combination of context-free "base rules" and transformational rules which handle movements, substitution and so on. Core grammars are held to be genetically determined and the structures which they generate are said to be "unmarked". Marked structures depend on rules specific to a given language for their formation. May claimed that the same types of rule determine unmarked readings in both syntax and logical form.

May's proposal is to scope quantifiers using a movement rule, Quantifier Raising (QR), which is subject to two constraints, Subjacency and the Condition on Proper Binding (CPB). These rules fit into the framework of trace theory. The QR rule is simply "Adjoin Q to S" where Q matches any quantificational NP and S any clause node. A new S node is created having the quantifier and the former S node as its left and right children; the

quantifier leaves behind a variable which is bound from the new position (analogous to a bound trace). Since the QR rule does not stipulate where the new node can go the above two constraints are needed to ensure correct binding.

The Subjacency rule states that a quantifier can only be moved to the most immediate S node. The CPB forces the quantifier to move to a higher rather than a subordinate S node. The rule states that "a variable is properly bound by a binding phrase ϕ iff it is c-commanded by ϕ". [1] The scope of a quantifier in the logical form is then everything it c-commands. The QR rule operates on a modified surface structure (SS) to give a scoped "logical" form which can be readily mapped onto a typed FOPL. The result is that scope ambiguities are treated as syntactic (structural) and the result of scoping is a linear ordering in FOPL.

In many ways May's approach is similar to the one used in this study; however, there are some computational difficulties with it which result from the strict use of syntactic rules to handle scoping. Some of these are discussed by van Lehn (1978). First of all, the Subjacency Rule, together with the CPB, forces all clauses to act as complete "scope islands"; that is a quantifier can only be applied to the immediate clause which contains it. In practice, this proves to be an overly restrictive constraint (see section 3.2 and chapter four). It might be possible to use the Subjacency Rule as a heuristic rather than as an absolute constraint and to allow quantifiers to widen scope over certain types of embedding clause. [2]

Secondly, the QR rule can only raise quantifiers to S nodes and therefore does not permit the scoping of quantifiers relative to VPs. Van Lehn points out the difficulty this poses for the scoping of quantifiers inside reduced relative clauses (VPs serving as noun complements). He notes that in May's system VPs must be either parsed as bare VPs or as clauses with null subjects. In the first case, quantifiers inside the VP must scope outside the head noun (since they cannot attach to the VP node) and in the second case the clause will be

trapped inside the complement. He mentions that it seems undesirable to require a syntactic ambiguity in order to get the two types of scope reading. Modifying the QR rule to allow attachment to VP nodes would solve this problem and would allow the use of heuristics to handle scoping with VPs (see chapter four).

Thirdly, the CPB constraint forces a quantifier inside a prepositional noun complement to scope outside the head noun, for the similar reason that quantifiers cannot attach to a PP node. However, there are many cases in which such quantifiers should scope inside the head noun ³ Finally, May's approach provides no rules for "horizontal scoping" (the scoping of clausemates relative to one another). Therefore a clause with *n* quantified arguments will have *n!* unmarked scope orderings. May suggests that such orderings merely differ in "preference" rather than in "markedness" but as van Lehn points out, this is a dubious distinction since in many cases the former type of effect may be just as pronounced as the latter. There is one exception to this, however. May's parser attaches *wh* quantifiers to a complementizer node outside the S node and this forces these to outscope all other quantifiers in the same clause. Although *wh* quantifiers often take wide scope it is easy to find cases in which they do not. An example is

(1) What do each of you want to drink?

It should be possible to add heuristics for the scoping of clausemates (other than *wh* quantifiers) to May's system. A discussion of the sorts of heuristics which might be added is given in chapter four.

These modifications would appear to remove many of the computational difficulties of May's system, but at the same time they would greatly reduce the extent to which the scoping is determined by syntactic rules. Van Lehn suggested that quantifier scoping is an "epiphenomen" in the sense that syntactic rules play a background role, merely influencing

our preferences for choosing scope readings (to the extent that we do select readings). This might be rephrased by saying that scoping is determined by the interaction of syntactic, semantic and pragmatic knowledge.

2.2. Quantifier Scope and Bound Anaphora

Quantifier scope dependencies may be thought of as abridged or "implicit" versions of anaphoric dependencies. For example, as the anaphoric pronoun in (2) is lost the scope ambiguity becomes increasingly difficult to resolve in (3) and (4).

(2) Each boy kissed a girl on his right
(3) Each boy kissed a nearby girl
(4) Each boy kissed a girl

Several attempts have been made to give a unified account of quantifier scope and anaphora by showing that they are both governed by the same set of constraints. van Lehn (1978) describes one such approach, based partly on the previous approaches of Keenan (1974) and Reinhart (1976).

Keenan uses a function-argument representation for sentences, analogous to the functional representation used by Montague. He proposes the following general principle which is intended to constrain both quantifier scope and anaphoric coindexing:

(5) Keenan's Functional Principle
The reference of the argument expression must be determinable independently of the meaning or reference of the function symbol.

This predicts that a quantifier in the subject position should generally outscope a quantifier inside the VP since the subject is usually taken to be the last argument to which the verb predicate (ie. function) is applied. However, it is possible to apply the function to the subject before an object, so that the subject may scope inside the object. Keenan also suggests that in

possessive NPs (eg. *every car's owner*) the possessive (*every car*) is the argument expression and should therefore take wide scope.

Van Lehn's approach is of interest in that scope dependencies are not represented by the linear ordering of quantifiers but by attaching argument variables to functionally dependent NPs as daughter nodes in the parse tree. This amounts to making "implicit" anaphoric dependencies explicit. The "scoped" tree may then be easily converted to a "skolem-like" representation, in the same way that May's scoped trees can be converted to a typed FOPL. In van Lehn's notation, the two scoped readings of (6) would be (7) and (8).

(6) Everyone loves someone
(7) (Vx:person() (Ey:person (x) [x loves y]))
(8) (Vx:person() (Ey:person() [x loves y]))

As this example shows, scope dependencies are expressed by "skolem-like" lists of arguments rather than by quantifier order. Although van Lehn refers to this as "typed skolem form" his notation is more general since univerally quantified NPs also have skolem-like attachments. The above representation language can express a partial ordering but this may be too powerful for representing scope dependencies in general, as these tend to be linearly ordered in the absence of indefinites or definites. One alternative would be to adopt a more genuine skolemized format in which only indefinites are skolem functions. Definites could be treated analogously by adding relations to their restrictions. Some potential advantages of this representation formalism are that (a) it can express "branching quantification" (see section 2.5), (b) it should permit the gradual resolution of scope ambiguities and (c) the scoped parse tree closely resembles the surface structure.

Van Lehn does not make full use of the expressive power of his representation formalism. He adapts Keenan's Functional Principle and also two constraints on bound anaphora taken from Reinhart (1976): "The Non-coreference Rule" and "The Non-definite

Anaphora Rule". The latter two rules attempt to use the c-command principle to constrain anaphoric coindexing and quantifier scoping. The resulting constraints on scope ordering turn out to be very similar to those of May. All clauses act as quantifier traps and quantifiers inside propositional noun complements must scope outside the head noun. The main advantage of the anaphoric approach over that of May is that the c-command constraint predicts the scope ordering of clausemates fairly well. However, as van Lehn shows this constraint proves to be no better than the effect of surface order in predicting the scoping of clausemates, so this is a dubious advantage.

2.3. Mapping Rules and Filters

Kempson & Cormack (1981) describe an approach to quantifier scoping which is designed to allow a gradual disambiguation. For their representation language, they extend FOPL to include quantification over sets of objects. This extension is especially useful for sentences with mixed numerals such as

(9) Two examiners marked six scripts.

The authors identify four interpretations of (9) shown as I to IV, where X_2 and S_6 are sets of two examiners and six scripts, respectively, and x and s are individual students and scripts. To simplify the notation it will be assumed here that $x \in X_2$ and $s \in S_6$.

(I) $EX_2\ Vx\ ES_6\ Vs\ Mxs$
(II) $ES_6\ Vs\ EX_2\ Vx\ Mxs$
(III) $EX_2\ ES_6\ (Vx\ Es\ Mxs\ \&\ Vs\ Ex\ Mxs)$
(IV) $EX_2\ ES_6\ Vx\ Vs\ Mxs$

The first two readings are the standard scoped readings. The "incomplete group" reading (III) asserts that a group of two examiners marked a group of six scripts, with each examiner and each script being involved in at least one marking relation. The "complete group" reading

(IV) is similar but asserts that each of the examiners marked each of the scripts.

The authors claim that these four readings are quite closely related and argue that they should not be represented as separate readings in the initial logical translation. In particular, the complete group interpretation (IV) entails the first three readings. It is also argued that I, II and II are quite closely related, although in a more general way. For example, they claim that crossed interpretations of (9) can be obtained when the VP anaphora test is used (see section 1.2) and argue that quantifier scope ambiguities, unlike syntactic ambiguities, are generally preserved across languages. [5]

The authors therefore propose that the initial semantic representation should be a general one, from which the more specific interpretations can later be derived by certain rules. They provide two possible initial representations for (9). The first, which they call the "general coordinate" form (V), is the most specific logical form which is entailed by each of I to IV.

(V) $EX_2\ Vx\ ES_6\ Es\ Mxs\ \&\ ES_6\ Vs\ EX_2\ Ex\ Mxs$

Interpretations I to IV can be derived from V by the application of two simple rules. They are (a) *the generalizing rule*: replace Ex with Vx and (b) *the uniformizing rule*: replace Vx Ey with Ey Vx. Applying the generalizing rule to the first and second terms of V gives readings I and II, respectively. The subsequent application of the uniformizing rule to either I or II will result in the complete group interpretation (IV). By starting with the uniformizing rule, and applying it to both terms of V, the logical form

(VI) $EX_2\ ES_6\ Vx\ Es\ Mxs\ \&\ ES_6\ EX_2\ Vs\ Ex\ Mxs$

is obtained. This can be made equivalent to the incomplete group interpretation III by using a constraint, or "filter", which ensures that a given NP can only have a single assignment of

reference (ie. the variable assignment function recognizes them as being identical). Thus the four readings I to IV can be derived from V using just two simple rules and a quite trivial filter. However, some additional filtering is needed to remove V and VI and also two additional intermediate logical forms which result when the uniformizing rule is applied to just one of the two terms of V instead of to both.

The authors note that V may not be sufficiently weak to represent all interpretations of sentences such as (9) in general. Sentences with mixed numerals may be true even though some members of the plural sets do not directly take part in the activity being predicated. As an example, they mention that

(10) Six students took five exams

could be true if only one of the students actually stole the exams but the others were co-conspirators in some way. In order to avoid having to add weakening (or approximating) rules to derive such interpretations from V, a more general initial semantic representation is proposed, which they call the "radical vagueness" representation. For sentence (9) this would be VII,

(VII) EX_2 ES_4 Ex Es Mxs

They compare this to the "general co-ordinate" account and come out in favour of the former, mainly because it does not require any weakening rules (although this will disputed below) and because it corresponds more closely to the syntactic structure.

There are a number of difficulties with both the general coordinate and the radical vagueness accounts. The need for filtering rules has already been mentioned for the former and this problem would be considerably worse for the latter. This problem would also become more serious as the number of quantifiers increased. One major problem with the first

account is that V is not structurally related to the surface form and therefore there could be no one-to-one mapping between syntax and an initial semantic representation of this type. The radical vagueness account is better in this respect, and Kempson & Cormack suggest that the representation shown in VII could be obtained by extracting the quantifiers in order of their surface appearance. However, there are two major problems with this.

First of all, since scope order does not necessary follow surface order it may later be necessary to reverse this ordering and therefore the initial representation would not be a general representation. This would be especially true for sentences containing distributive quantifiers. Secondly, once quantifiers have been extracted from their surface positions, information about their structural relations is lost. Therefore, it appears to be be necessary to at least partially determine scoping heuristics before extracting the quantifiers, rather than after. This criticism can be applied more generally to any method based on the application of interpretation rules and filters to an initially general semantic representation which does not preserve structural relations.

Another difficulty, which the authors address, is that neither V nor VII are proper semantic representations of (9) by themselves. For example, VII only asserts that there exist sets of two examiners and six scripts but the actual predications made are independent of the sizes of the sets. The authors try to remedy this problem by proposing that semantic interpretation involves two parts, an initial semantic representation and a set of semantic intepretation (mapping) rules and filters. Such an approach could be useful in that it might provide a means of gradually disambiguating quantifier scope relations, proceeding from general to gradually more detailed representations. However, it is not clear from the paper just where the authors draw the line between "semantic" and "pragmatic" disambiguation.

Finally, one of the main disadvantages of the general coordinate account is that it would be necessary to provide weakening rules to obtain readings in which not all the

members of some set are directly involved in the predication being made of the set. The main motivation for proposing the radical vagueness account was to avoid the need for such weakening rules. However, even the latter account does not appear to be general enough since there may be collective, or collaborative, readings in which the predication being made does not apply to any *individual* member of a set. Kempson & Cormack in fact mention an account of quantifier scoping (in footnote 12) which can represent collective predication and which is very similar in this respect to the representation method used for plural indefinites in this study (see section 3.10). In the account which they describe, a sentence has three primary representations: the two scoped readings and a "collective" reading in which the numeral set variables are the arguments of the predication. Kempson & Cormack argue that treating sets as primitives may be necessary for certain types of collective predicate such as *extinct* [7] but not for predicates such as *destroy*. They do concede that the sentence

(11) Five boys destroyed six flower-beds

may be true even if no boy destroyed a whole flower-bed by himself, but in this case they suggest that a weaker predicate should be substituted for *destroy*, for example *contribute to the destruction of*. However, if such a predicate-weakening rule were to be introduced there would be no need for the radical vagueness account in the first place since *all* members of a set must contribute to the predication being made in some way in order for a mixed numeral sentence to be true.

The approach used in this study is similar to that of Kempson & Cormack in that the initial logical representation is a "general" one from which the different scoped interpretations can be derived by "scoping rules". However, there are some major differences. The initial representations used in this study preserve the initial structural relations among quantifiers and other logical operators [8] and therefore are ideally suited to making use of the heuristic information contained in these structural relations. The initial representions are also

considered to be semantically "incomplete", that is they are not taken to express propositions until after scoping has been completed. In this study, a clear line is drawn between the disambiguation of scope relations and that of the type of predication being made. It is felt that the details of the predication being made in mixed numeral sentences is largely a property of the predicate and should be dealt with by meaning postulates and pragmatics. Therefore, both the complete and incomplete group interpretations, as well as a large number of possible collaborative types of predication, are all obtained after scoping from one of three initial scoped interpretations (see section 3.10).

2.4. Combinatoric Features

Webber (1983) argues that people do not generally perceive clear quantifier scope interpretations as such, but rather recognize certain "combinatoric features" such as iteration, dependency, cardinality, tense, negation and modality. *Iteration* refers to the iteration of a quantifier over a predication and *dependency* to the functional dependency of one quantifier on another. Her interest in quantifier scoping is only indirect, her main concern being to construct "initial descriptions" (IDs) from noun phrases which can then be used to support subsequent anaphoric reference. However, she notes that "something like" quantifier scoping is needed since anaphora is sensitive to different quantifier scope interpretations, and the logical notation she uses in the above paper is very similar to that used for standard scope interpretations. [10]

The representation language used in Webber (1983) is a modified version of FOPL. Plural NPs are represented using a "set" operator and a V operator is used to distribute over the members of the set. This is similar to the representation language used by Kempson & Cormack and to that used in this study. Definite and indefinite sets are represented using E! and E, respectively, and universal quantification is expressed by distribution over a definite set. For example, the distributive reading of (12) would be represented (after disambiguation)

as (13). [11]

(12) Each man ate a pizza
(13) (E!w:*set*(Man))(Vx∈w)(Ey:Pizza)Ate(x,y)

The collective and distributive interpretations of the noun phrase shown in (14) are represented in (15) and (16), respectively.

(14) Two men who together/each ate a pizza
(15) Ex:λ(v:*set*(Man))[(Ey:Pizza)Ate(v,y)&|v|=2]
(16) Ex:*set*(λ(v:Man)[(Ey:Pizza)Ate(v,y)])&|x|=2

Note that the cardinality of each set is represented explicitly and that the iteration inside the noun complement is in effect determined by the relative scopes of the *set* and λ operators.

Webber proposes three types of reading for a sentence containing two plural numerals: distributive, collective and conjunctive. She illustrates this with the sentence

(17) Three boys bought five roses

The distributive reading is the standard scoped reading in which the subject distributes over the object. [12] In the collective reading, only a "consortium" of the three boys can be considered to have bought anything. The conjunctive reading is similar to the incomplete group reading of Kempson & Cormack, but is weaker in that each boy may own only part of a rose. The complete group reading is not mentioned. This is quite similar to the interpretation used in this study, except that both the "collective" and "conjunctive" readings are derived after scoping from the same reading. The former may be considered to be a limiting case of the latter; furthermore, the difference between the collective and conjunctive readings may not always be that distinct.

In the next three sections, some recent proposals for the representation of mixed quantifier interpretations will be examined.

2.5. Branching Quantification

Hintikka (1974) claimed that certain English sentences could not be represented in FOPL but required a form of branching quantification known as finite partially ordered (FPO) quantification. One of the examples he used is

(18) Some relative of each villager and some relative of each townsman hate each other

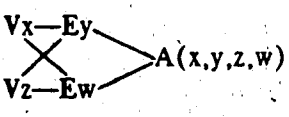
which he represented as

(19) $\begin{matrix} \forall x: \text{villager} & - & \exists y: \text{relative} \\ \forall z: \text{townsman} & - & \exists w: \text{relative} \end{matrix}$ [y hates w].

Hintikka claimed that the preferred reading of (18) could not be represented by the linear ordering of the quantifiers since the relative of the townsman must not depend on the villager and vica versa.

Hintikka's claim has proved to be controversial. Since his examples are restricted to universal and existential quantifiers, some suggestions have been made that sentences such as (18) could be handled using FOPL if indefinites were given a special treatment. Fauconnier (1975) found that sentences of this type were not considered to be false unless the weakest reading, ie. VVEE, was violated. He therefore proposed that the weak reading, which is entailed by the other readings, should be used as the semantic representation for such sentences. A similar proposal was made by Cooper (1979) who suggested that by giving indefinites narrow scope and obtaining the more specific readings by pragmatics, an account could also be given of their ability to escape from scope islands. [13]

The case for branching quantification has been taken up more recently by Barwise (1979). He first presents an explicit model of the situation described by (18) and shows that there can be a state of affairs in which the weakest reading (VVEE) is required. Therefore, (19) is not an adequate logical representation of (18), even though Hintikka's FPO langauge can always represent the weakest reading using the standard linear representation of FOPL. However, Barwise suggests that (18) can be represented in a more natural manner using a related branching quantification language, which appears to be able to represent a complete partial ordering. The modified representation of (18) is shown as (20).

(20) 

Although (20) has equivalent FOPL and FPO representations, this is not the case when other quantifiers are used. For example, it is argued that

(21) Most relatives of each villager and most relatives of each townsman hate each other

requires branching quantification. Barwise further suggests that branching quantification provides a more natural way of representing sentences with conjoined noun phrases even though this is not needed in this case. As an example, (22) would be represented as (23).

(22) Quite a few boys in my class and most girls in your class all passed the test

(23) 

2.6. Generalized Scope Relations

Gil (1982) bases his discussion on the results of a questionnaire in which subjects were asked to interpret simple sentences containing two numeral quantifiers. They were given a choice of four interpretations, corresponding to the four interpretations for mixed numeral sentences described by Kempson & Cormack (see section 2.3). Gil modifies their terminology somewhat, substituting *symmetric* for group and *strong* and *weak* for complete and incomplete. Questionnaires were given in several unrelated languages. The main findings were that (a) all four interpretations were obtained, and therefore need to be accounted for by the representation language and (b) the symmetric interpretations were preferred, with the strong symmetric interpretation being the favourite. [14]

Gil begins by comparing four existing proposals for representing quantifier scope interpretations: (a) standard FOPL, (b) branching quantification , (c) quantification over sets and (d) Montague and Keenan-Faltz logical forms. He notes that only (c) can represent the weak symmetric (incomplete group) reading and only (b) and (c) the strong symmetric (complete group) reading. [15] He also claims that the strong preference for symmetric interpretations is quite opposed to the scoped treatments given in (a) and (d) and is at odds with Kempson & Cormack's proposal for quantification over sets. [16] However, this preference is compatible with the branching quantification account.

Gil argues that the problems encountered by these four proposals stem from the linearity of logical notation which can be traced back to Frege. This linearity results in relations between quantifiers being represented primarily in terms of scope dependencies, rather than in some more general format. The use of branching quantification extends the format to a partial ordering but Gil claims this is not enough since, for example, it provides no means of representing the weak symmetric (incomplete group) reading. Instead he advocates introducing explicit "generalized scope relation" operators. The used of such

operators considerably simplifies the representation of "scope" relations among quantifiers. For example, the weak symmetric reading for (24) is simply (25).

(24) Two boys kissed three girls
(25) [R3(E3b, E2g)]S(b,g)

where R3 is the operator for the weak symmetric relation. [17]

Gil does not show how generalized scope relations could be used computationally, although he does claim that the set of generalized scope relations forms a Boolean algebra and suggests that a "unit generalized scope relation" could be formed from the disjunction of all relations. It would not be difficult to modify the scoping method described in this study to represent quantifier scope relations in the format recommended by Gil. However, it is not clear how Gil's method could be used to select the preferred readings of a sentence or how it could be extended to handle the interactions of quantifiers with other logical operators such as coordinators and negation.

2.7 Referential and Quantificational Indefinites

So far in this chapter a number of different proposals for representing indefinites have been described. These have included the skolem-like representation of van Lehn, the use of set-forming operators with universal partitives for plural indefinites (Kempson & Cormack and Webber) and the use of branching quantification and generalized relations. Recently, a number of people have proposed that indefinites are lexically ambiguous and should be given two interpretations: as standard quantifiers and as "referential terms" (ie. indexicals). Since indexicals are not scoped and quantifiers are largely clausebound this proposal would, if workable, considerably simplify the scoping of indefinites. The major part of the next chapter will be devoted to examining the evidence for and against this proposal.

Footnotes

1. C-command may be defined as follows: A phrase X c-commands a phrase Y iff every branching node that dominates X also dominates Y, and X does not dominate Y.

2. May did not attempt to deal with the ability of indefinites to readily escape from scope islands. Hornstein (1984) attempts to deal with this problem within the framework of transformational grammer (see section 3.6).

3. The scoping of quantifiers inside noun complements is discussed in section 5.4. In the system being used in this study, PPs are translated as clauses which have heuristic values associated with them which tend to favour the readings in which quantifiers inside the PP scope outside the head noun.

4. This is clearly an overly rigid rule although van Lehn does not seem to think so. He claims for example that the phrase *Every flight to an Eastern city* (p.86) cannot have a distributive reading (in which there is a different city per flight), although it does not seem very hard to get this reading. There is another problem introduced by the particular way in which van Lehn adapts the constraints of Keenan and Reinhart. He does not allow definite NPs to have Skolem modifiers (arguments) and therefore can apparently not get the distributive reading of *Each person brought his own car*. He also must introduce a very *ad hoc* condition into the formal semantics to get the distributive reading of a definite NP with a distributive NP inside a prepositional complement.

5. Although Gil (1982) shows that there are some differences in the types of quantifier scope interpretations which are made in different languages.

6. This latter route ends up with either I, II or IV, depending on which path is taken, so this presents the additional problem of redundancy.

7. However, *extinct* is a generic-kind predicate and is quite different from a non-generic predicate such as *form-a-circle* which can also only be used collectively.

8. It is not apparent how quantifiers could be scoped with other logical operators such as coordinators and negation using the method of Kempson & Cormack.

9. Webber uses four rules to derive IDs from indefinite and definite NPs. The two rules for definite NPs are straightforward since their uniqueness is already specified. IDs formed from indefinite NPs include a conjunction of (a) the restriction, (b) the main predication (the specification of which requires that VP ellipsis first be resolved) and (c) a predication stating which sentence "evoked" the ID.

10. In Webber (1983) it is assumed that disambiguation of the scope relations, or of the "combinatoric features", has already been performed and so no details are given about the process of disambiguation itself. She mentions that this process is carried out using the representation language KL-ONE which allows scope relations to be expressed as a partial ordering and which permits a gradual disambiguation of scope relations.

11. It is assumed that the unique referent will be specified by pragmatics. Kempson & Cormack use a similar format to express universal quantification but use an indefinite instead of a definite set determiner.

12. Webber apparently does not get the reading in which the object distributes over the subject, in contrast to Kempson & Cormack. In chapter four, it will be argued that this reading is usually not obtained although there are cases in which it can be obtained and therefore it is necessary to be able to derive it.

13. The ability of indefinites to escape from "scope islands" will be discussed in section 3.2. Fodor & Sag (1982:371) argue that Cooper's proposal cannot account for the island-escaping behaviour of indefinites. They give two examples, containing the quantifiers *exactly half* and *at least five*, in which the entailment relation does not hold. The second of these is *Mary dates at least five men who know a producer I know*. It should also be noted, as Saarinen (1980) points out, that when an indefinite is inside an opaque context neither the specific nor the opaque reading entails the other one.

14. Gil's empirical data is probably biased in favour of symmetric readings since sentences containing numerals have a strong tendency to receive symmetric interpretations. In contrast, quantifiers such as *each*, *most*, *few* and *no* interact to give predominantly linear scope interpretations. It would therefore be useful to make a companion study to Gil's using such quantifiers.

15. Gil mentions that Montague and Keenan-Faltz logical forms have been shown to force the same scope dependencies as FOPL (p. 422).

16. See footnote 12 in Gil (1982). It may also be noted that the mapping rules used by Kempson & Cormack tend to favour the scoped interpretations since the two group interpretations must always be derived from the former (and therefore are considered to be more "specific" interpretations).

17. There may be some syntactic basis for introducing explicit scope relation operators since adverbs such as *each, together, separately* and *independently* are explicit indicators of scope relations. Gil mentions that in Tagalog *bawat* is like the adverb *each* but entails that the dependent sets be disjoint. He proposes that this should be represented as a separate relation.

Chapter 3

The Scoping of Indefinite and Definite Noun Phrases

Indefinite and definite NPs differ in a number of ways from quantifiers such as *each*, *most*, *few* and *no*. For example, they are immune to the "scope island" and "VP deletion" constraints (see below), readily support non-restrictive relative clauses and may co-refer with pronouns across arbitrary stretches of discourse. Plural indefinites and definites commonly receive collective interpretations, in contrast to proper quantifiers. These special properties of indefinites and definites complicate their scoping behaviour and a number of attempts have been made to give simplified accounts of the scoping of indefinites. Some of these were mentioned the previous chapter. The main aim in this chapter will be to examine the recent proposal that indefinites and definites are lexically ambiguous between having referential and quantificational interpretations. This proposal is potentially very useful for scoping since referential terms are not scoped and distributive quantifiers are largely clausebound. However, there are a number of difficulties with the proposal which suggest that it at least requires some modification. Some of these difficulties will be discussed in sections 3.2 to 3.8. In the last two sections of this chapter two alternatives will be described. In section 3.9, a possible modification of referential indefinites will be described and in the last section a more uniform treatment of indefinites and definites will be given in which their special scoping properties are attributed to the E- and A- quantifiers. First, however, some terminology needs to be clarified.

3.1. Specific, Non-specific and Referential Indefinites

The terminology used to classify indefinites (and definites) can be quite confusing. To start with, a distinction is often made between *specific* and *non-specific* indefinites, usually with reference to the interpretation of indefinites inside opaque contexts. For example, in

(1) John wants to marry a blonde

the indefinite *a blonde* can be given a specific interpretation, if John has a particular blonde "in mind", or a non-specific interpretation, in which case the blonde is opaque to John. This terminology is used by, for example, Fodor (1970), Ioup (1977) and DeCarrico (1980). These two interpretations may be represented by scoping the indefinite relative to the opaque operator. The notion of "specificity" being used here is a relative one, since an indefinite may be transparent to one person and opaque to another. For example, there is a reading of

(2) Mary thinks that John wants to marry a blonde

in which John has a particular blonde in mind but the identity [2] of this blonde is opaque both to Mary and the speaker. This reading may be represented by scoping *a blonde* outside *wants* but inside *thinks*.

More recently, it has been proposed that indefinites in transparent contexts are also ambiguous, in relation to the speaker. If the speaker is making reference to a particular object, of which he knows the identity, the indefinite is interpreted as an indexical, or as a *referential indefinite*. For example, the referential interpretations of *a blonde* in (1) and (2) would correspond to the speaker having a particular blonde in mind. [3]

The difference between indexicals and non-indexicals was discussed in section 1.3. Indexicals are directly referential and their denotations are determined solely by the context. Among other things this means that the interpretation of indexicals is independent of quantifiers and other logical operators within whose scope they lie. Quantificational NPs are variable-binding operators and therefore display scope ambiguities relative to other operators. The indexical/quantifier distinction is sometimes known as the *referential/quantificational*

distinction for indefinites (eg. Fodor & Sag 1982) and the *referential/attributive* distinction for definites (eg. Wettstein 1981). In the next six sections, the evidence that indefinite and definites NPs should be given an indexical interpretation will be examined.

3.2. The Ability of Indefinites to "Escape" from Scope Islands

It is well known that clauses tend to act as strong "traps" for quantifiers, that is, quantifiers can seldom widen scope over an embedding clause. Some types of clause do occasionally permit a quantifier to "escape" from it, for example, clauses which serve as verb complements (3) or as relative clauses (4).

(3) A quick test showed that each drug was psychoactive
(4) Several tourists who visited each city bought a souvenir

Van Lehn (1978:8) empirically tested (3) and found that about half the time subjects chose the reading in which there was a different test per drug. It is possible to get the reading in (4) in which the set of tourists depends on the city.

Certain types of clause form what appear to be absolute traps for quantifiers and these are said to act as *scope islands*. Two examples are object complement and initial-*if* clauses. This can be seen by the implausibility of trying to force the distributive reading in (5) or (6).

(5) *Someone different heard a rumour that each student had arrived
(6) *If each student has arrived then a friend (of his) will go to meet him

However, indefinites and also definites, if they are treated as scoped elements, are immune to the scope island constraint. It is always easy to get a wide scope reading of an indefinite or definite no matter how deeply it is embedded. How can this be accounted for?

Fodor & Sag claim that "an indefinite that escapes from an island has maximally wide scope with respect to any quantifiers or logical operators outside the island" (1982:374). [4] The two key sentences they use to make the claim that indefinites inside scope islands cannot escape to intermediate positions are reproduced here as (7) and (11).

(7) Each teacher overheard the rumour that a student of mine had been called before the dean

(8) (Vx:teacher [x overheard the rumour that
(Ey:student of mine
[y had been called before the dean])])

(9) (Vx:teacher (Ey:student of mine
[x overheard the rumour that
[y had been called before the dean]]))

(10) (Ey:student of mine (Vx:teacher
[x overheard the rumour that
[y had been called before the dean]]))

(11) If a student in the syntax class cheats on the exam, every professor will be fired

For (7) they claim that the intermediate reading (9), in which each teacher has a specific student in mind, is not obtained. The authors admit that this may not be completely convincing, however, the absence of an intermediate reading in (11), in which there is a specific student per professor, is indisputable.

First, some counterexamples will be presented. The intermediate reading of (7) appears to be hard to get for pragmatic reasons: there is no apparent relation between the teacher and student. If such a relation is present, especially if an explicit anaphor is present, the intermediate reading is easy to get (12). [5]

(12) Each teacher overheard a rumour that a favourite student (of his) had been called before the dean

It is also frequently possible to obtain intermediate readings in sentences containing scope islands by embedding such sentences inside prepositional sentential adverbials (13).

(13) At each school, several students overheard a rumour that a certain teacher had quit

In this case, the "anaphoric relation" between *school* and *teacher* can easily be left implicit.

The absence of an intermediate reading in (11) is somewhat more difficult to explain and, in fact, raises some additional problems. However, it can again be seen that intermediate readings are possible by embedding variations of (11) inside a sentential adverbial (14) or an attitude clause (15).

(14) In each department, if a (certain) student cheats then every professor will be fired
(15) Each teacher knows that if a (certain) student (of his) cheats he will be reprimanded

Therefore, the ability of indefinites to "escape from" initial-*if* clauses cannot be accounted for by giving them an indexical interpretation, at least in the sense described by Fodor & Sag.

However, this would be possible if "referential" NPs were allowed to be functionally dependent. The context would then assign constant functions to such terms. In fact, it would appear to be necessary to allow for this possibility in the formal semantics defined by Fodor & Sag. In their semantics, indexicals are evaluated relative to the time and world of utterance (t_c,w_c), and so their denotations are independent of time and world (t,w). However, this method of evaluation does not make them immune from dependency on other quantifiers, and there appears to be no obvious way to modify their semantics to incorporate such an immunity. Therefore, one step towards reconciling their proposal with the data shown here would be to modify the definition of referentiality to allow for functional dependence.

However, this still does not appear to be a sufficient modification, since indefinites which escape from scope islands may still be embedded inside an opaque operator such as an attitude operator. In (16) there is a clear reading, perhaps the preferred one, in which there is

a particular student in each class who will be thrashed if he cheats but the students are opaque to John. That is, *a student* acts as though it scopes outside the *if*-clause but inside the opaque predicate *thinks*.

(16) John thinks that, in each class, if a (certain) student cheats then he will be thrashed

This seems to be incompatible with the interpretation of *a student* as a referential term. In section 3.9, the possibility of further modifying referential indefinites to allow them to be both within the "scope" of opaque operators and quantifiers and still retain their specificity (relative to negated contexts etc.) will be discussed.

Returning to example (11), an explanation is still required as to why there is no intermediate reading. Intuitively, we can see that the absence of an intermediate reading is a result of the surface order of the quantifiers: it is easy to get the intermediate reading if either the universal quantifier is first "moved" to the front of the sentence in the English version (17) or if the consequent clause is placed before the antecedent clause (18).

(17) For each professor, if a (certain) student cheats then he will be fired.
(18) Each professor may be fired if a (certain) student (he knows) cheats

One way of explaining this would be to postulate that the consequent clause also acts as a scope island when it follows the antecedent, thereby trapping the universal quantifier inside it. This explanation is suggested by the fact that it is not possible to substitute (17), in which the universal quantifier is placed outside the whole sentence in the English version, for (11) and by the fact that it does not appear to be possible for *every teacher* to bind an anaphoric pronoun associated with *a student* in

(19) *If a student of his cheats then every teacher will be fired

even when the indefinite is interpreted non-specifically.

This inability of distributive quantifiers inside a consequent clause to widen scope over a preceding antecedent clause appears to be a specific instance of a more general rule which can be applied to distributive quantifiers inside all clauses joined by connectives (see section 4.6). This gives the explanation some additional support. Although some alternative explanation may later prove to be more satisfactory, it seems clear that the absence of an intermediate reading in (11) is a result of the universal quantifier being placed after the *if* clause in surface order and not of the indefinite being unable to widen scope to intermediate positions.

Finally, perhaps the simplest and clearest evidence that indefinites inside scope islands can have non-indexical wide scope readings is to compare the wide-scope reading in a simple sentence such as (20) with those in sentences containing scope islands (21,22).

(20) John thinks that two friends of his are arriving
(21) John heard a rumour that two friends of his are arriving
(22) John thinks that if two friends of his arrive today they will drive to the country

In all three sentences it is equally easy to obtain the reading in which the two friends are specific to John but non-specific to the speaker, ie. the wide scope existential reading. Therefore, if the opaque/transparent distinction in (20) is obtained by scoping the indefinite, a parallel treatment should be given to (21) and (22). Perhaps a corollary of this is that if the wide scope readings of (21) and (22) are not to be obtained by quantifier movement then neither should that in (20), ie. the interaction of indefinites with opaque operators should not be handled by scoping. This possibility will be discussed further in section 3.9.

3.3. The VP Deletion Constraint

The second major argument used by Fodor & Sag to support the need for referential indefinites is the apparent immunity of indefinites to the following constraint on VP deletion, originally proposed by Sag (1976): [7]

(23) A verb phrase may be deleted only if its logical translation is an alphabetic variant of an expression in the logical translation of the surrounding discourse.

The effectiveness of this constraint depends on the representation of sentences in a particular form of intensional logic, in which verb phrases are represented as lambda expressions with quantifiers in the object position embedded inside the lambda operator. The stipulation that corresponding variables must be alphabetic variants applies to variables bound both by lambda operators and by quantifiers. [8] However, as we will see, the effectiveness of the constraint is most clearcut in the former case.

A simple example of the constraint involving two quantifiers can be seen when (24) is followed by (27). The logical forms are represented using the notation of Sag.

(24) Someone loves everyone
(25) (Ex) [x, λy((Vz) [y loves z])]
(26) (Vz) (Ex) [x, λy(y loves z)]

(27) Chris knows that someone does
(28) (Eu) [u, λv((Vw) [v loves w])]
(29) ?(Vw) (Eu) [u, λv(v loves w)]

Fodor & Sag claim that only the sequence of readings in which the universal quantifier scopes inside the lambda operator, (25) and (28), are permitted. The lambda expressions in (26) and (29) are not alphabetic variants of each other since *z* and *w* are free variables inside the expressions. Thus, it follows from (23) that a VP cannot be deleted if it contains a quantifier which widens scope outside the VP, unless the quantifier has scope over both the antecedent

and deleted VP.[9] However, it does seem *possible* to get the reading represented by (29), especially if *everyone* is stressed or replaced by *each person*. In this case it may not be possible to account for this by widening the scope of *everyone* since the latter should preferably not scope outside the attitude clause in (27).[10]

Fodor & Sag claim that indefinites are immune to the VP deletion constraint but that only referential indefinites, which entail maximally wide scope, have this immunity. However, counterexamples can be constructed by embedding sentences inside adverbials as in the previous section. In (30) it is easy to get a reading in which *a certain alderman* scopes outside both VPs but inside *both cities*. This reading does not violate the VP deletion constraint, but it shows that the ability of indefinites to scope outside a VP without constraining VP deletion cannot be explained by interpreting such indefinites as indexicals.

(30) In both cities, most of the men dislike a certain alderman and several of the women do, too

(31) In both wards, most people are expecting a certain candidate to win
(32) Yes, in both wards most people are
(33) Yes, most people in both wards are

There is also a reading of (30) in which the men and the women dislike separate aldermen, but this reading does violate the constraint and is much harder to get. This is also the case if (31) is followed by either (32) or (33), although here it does seem plausible to get the readings in which the constraint is violated by the indefinite.[11]

The claim of Fodor & Sag finds more support in sentences containing nested verb phrases. The VP deletion constraint predicts that when a lambda operator or quantifier has scope over two nested VPs, only the outermost VP will be deletable (Sag:119). The constraint works well with variables bound by lambda operators. A very elegant demonstration of this is provided by the inhibition of deletion in (36) but not in (34).

(34) John is ready to leave, and Mary is ready to, also
(35) (John, λx[x ready for (x, λz (z leave))])

(36) *Mary is fun to tease, and Betty is fun to, also
(37) (Mary, λx((Δ,λz[z tease x]),λy(y be fun)))

The logical representations for the first part of each sentence are shown in (35) and (37). [12] It can be seen that the unacceptability of (41) is attributable to the lambda variable *z* in (42) which will not be an alphabetic variant of the corresponding variable in the deleted verb phrase. [13]

Fodor & Sag argue that the above constraint also applies to quantifiers inside nested VPs. For example, they claim that the continuation shown in (38) is possible only if *everyone* has narrow scope but that the continuation "Chris does too" would be acceptable with both readings.

(38) Sandy thinks that someone loves everyone. Chris thinks that someone does too

Again there seems to be a certain degree of fuzziness here, and it does seem possible, even if unlikely, to get the forbidden readings (given the right context, stress etc.). This vagueness is not too surprising if quantifier scoping is not a structural but a secondary phenomenon, in contrast to gapping. [14] Fodor & Sag then use (39) as evidence that indexical indefinites are immune to the constraint.

(39) Sandy thinks that every student in our class plays chess better than a guy I beat this morning. Chris does thinks that every student does too

Here they claim that the continuation is possible if *a guy* is either completely non-specific or indexical, but not if it it scopes outside *every student* but inside the matrix verb (meaning that Sandy thinks such a person exists but has no one particular in mind). However, the intermediate reading as they describe it is a bit hard to grasp to start with. In sentences in

which intermediate readings are easier to grasp the constraint seems less convincing. [13] The authors in fact mention one such example (in footnote 12), namely

(40) Sandy thinks that everyone in our class sent a note to the teacher. Chris thinks that everyone did, too

but argue that in this case *everyone* should be given a collective interpretation. This is a reasonable suggestion but the deletion with the intermediate reading still seems possible in

(41) Sandy thinks that *each person* in our class signed his name on a card for the teacher. Chris thinks that each person did, too

where *each person* cannot have a collective interpretation.

Finally, it should be noted that there is a completely different argument which may be used against the need for referential indefinites, in the sense that referential terms, unlike attributive ones, directly denote. It is clear that VP deletion is not constrained by giving *a certain person* wide scope in

(42) Most people hope that a certain person will be apprehended, but some people do not

regardless of whether the indefinite is being used to directly refer to a known individual or is being used attributively, for example to refer to the hypothetical murderer of Smith (see section 3.8).

3.4. Non-restrictive Relative Clauses

Fodor & Sag claim that an indefinite which is modified by a non-restrictive relative clause as in (43) will tend to have (if not require) an indexical interpretation but that quantificational NPs cannot support non-restrictive relative clauses (44,45).

(43) A student in the syntax class, who has a Ph.D. in astrophysics, cheated on the exam
(44) *No/few students in the syntax class, who have Ph.D.'s in astrophysics, cheated on the exam
(45) *Each/every student in the syntax class, who has a Ph.D. in astrophysics, cheated on the exam

While examples (44) and (45), used by Fodor & Sag, are unacceptable, it does seem possible on occasion for a distributive quantifier to support a non-restrictive relative clause, as in (46).

(46) Each applicant, who must be a Canadian citizen, should first fill out these forms

However, such examples are probably rare and may never be possible with quantifiers such as *no* or *few*. [16] Even so, the ability to support non-restrictive relative clauses cannot be used as a test for indexicality, as the following examples make clear.

(47) Some dissident, who evidently wishes to remain anonymous, sent this note
(48) Mary wants several people, who have not been selected yet, to help her with the project
(49) Each student wants to get several people, who need not have any previous experience, to help with his project

The examples show that non-restrictive relative clauses can modify non-specific indefinites which refer to unknown individuals (47) or which scope inside opaque operators (48) or other quantifiers (49). Therefore, the above examples point to a difference between indefinites and distributive quantifiers, rather than between indexicals and quantifiers. [17]

3.5. *One* Anaphora

Indefinites can support two main types of anaphoric reference, which may be termed "specific" and "generic" and which are typified by the pronouns *it* and *one*. The difference can be illustrated by an example from Ioup (1977:233) in which (50) may be followed by

either (51) or (52):

(50) Melinda wants to buy a motorcycle
(51) She will buy it tomorrow
(52) She will buy one tomorrow

Ioup suggests that the specific and non-specific anaphors in (51) and (52) corefer with the *de re* and *de dicto* interpretations of *a motorcycle* in (50). [11]

Gillon (1986) claims that by forcing the indefinite to be specific, *one* anaphora can be prohibited and illustrates this with the unacceptability of

(53) *Rajan will buy a particular mattress; in fact, he will buy one today.

He argues that this provides some evidence that indefinites should be interpreted as either referential or quantificational. However, there are some difficulties with this argument. First of all, the indefinite in (53) need not be referential (in the sense of Fodor & Sag) but can be simply *de re*, that is specific relative to the opaque operator. Secondly, *one* anaphora may be prohibited for pragmatic reasons if it is clear that the same object, whether specific or non-specific, is being referred to. As (54) shows, this may occur with an indefinite embedded inside an opaque context.

(54) *John will order a pizza and he will eat one

Thirdly, in the right context, completely referential NPs can quite easily support *one* anaphora, although only in a generic sense. An example is

(55) Don't use this chair; get one from the living room

It might be argued, however, that the ability to use *one* anaphora in (55) results from a

secondary process. As Webber (1983) points out, inferencing is often required in order to obtain suitable "generic sets" which serve as antecedents for *one* anaphors. Some secondary processing is clearly required in

(56) John just bought this [pointing]; now everyone will want to get one

One possible difference between referential and quantificational indefinites is their ability to serve as antecedents to both *it* and *one* anaphors. Although (58) is acceptable, it is more awkward than (62). It seems that in the former we want Bill to buy a different copy of the same book.

(57) John bought a book yesterday and lent it to Mary. Bill bought one today.
(58) John bought this book yesterday and lent it to Mary. Bill bought one today.

However, the same difficulty in supporting *one* anaphora is also shown by functionally dependent specific indefinites (59).

(59) ?In both companies, John met a certain executive and Bill also met one.

3.6. Type I and Type II Quantifiers

Hornstein (1984) makes a distinction between "Type I" and "Type II" quantifiers which is quite similar to the referential/quantificational distinction. Type I quantifiers have a name-like logical syntax whereas Type II quantifiers act as variable-binding operators. Type I quantifiers are not scoped, but entail maximally wide scope. They include all definite NPs (apparently), *a certain* and *any*. Type II quantifiers are moved by May's QR rule (see section 2.1) and therefore have a clausebound scope. They include proper quantifiers and all indefinites other than *a certain*.[19]

Two major differences can be noted between the Type I/Type II distinction and the referential/quantificational distinction of Fodor & Sag. First, Type I quantifiers are not taken to be directly denoting expressions, as are referential terms, but are interpreted relative to a domain of entities. Hornstein reserves the directly referring interpretation for proper names. Secondly, Hornstein does not make a general Type I/Type II distinction for definite or indefinite NPs; the former are all considered to be Type I and all indefinites other than *a certain* are considered to be Type II.

Despite these differences, however, most of the arguments that Hornstein uses to support his Type I/Type II distinction apply equally well to the referential/quantificational distinction. He presents three main types of evidence, all within the framework of transformational grammar:

(i) Interpretative independence
(ii) Pronoun binding constraints
(iii) The empty category principle (ECP).

In the remainder of this section, the first of these will be briefly considered; the second will be treated in the following section. [20]

Hornstein claims that Type I quantifiers show "interpretative independence", that is their interpretation is independent of other logical operators such as quantifiers, negation and modals. There are some problems with this claim. We have already seen that both definite NPs and also *a certain* can be functionally dependent on other quantifiers. Also there seems to be no reason to single out *a certain* since all indefinites can be understood referentially. [21]

Hornstein claims that quantifiers do not scope with negation or modals. Type I quantifiers are independent of these operators and Type II quantifiers always scope inside them. This is clearly wrong in the case of negation: in particular, indefinites in the object position and universal quantifiers in both the the subject and object positions show clear scope ambiguities relative to verb negation (see chapter four). [22]

3.7. Bound and Unbound Pronouns

Pronouns are often classified as being either "bound" or "unbound". The difference between bound and unbound pronouns can be clearly illustrated with a sentence such as

(60) John showed each person his room

where the pronoun *his* may either be interpreted as a variable bound by the quantifier *each person*, as an unbound term coreferential with *John* or as an unbound term which refers to an individual previously mentioned or perceptually indicated. [23] If there is good evidence for classifying pronouns as either bound variables or as referential terms, then this could be used to determine whether indefinites should be interpreted as quantifiers or as both quantifiers and referential terms.

Hornstein claims that Type I and Type II quantifiers can be distinguished by certain syntactic constraints which affect pronouns which coindex with Type II but not with Type I quantifiers. [24] For example, he claims that Type II quantifiers cannot bind pronouns across sentential connectives or boundaries and that (61) is unacceptable. However, the problem with (61) appears to be related to *everyone* binding a singular pronoun and (62) is clearly acceptable.

(61) ?Everyone bought a beer. He drank it quickly
(62) Each person bought a beer. He drank it quickly

In fact, sentential boundaries do not appear to form strong scope islands and it is not difficult to find examples such as (62) in which both distributive and existential Type II quantifiers coindex pronouns across discourse. [25] Hornstein also claims that Type II quantifiers inside scope islands such as an initial-*if* clause cannot coindex pronouns outside the scope island. He uses (63) and (64) as examples.

(63) *If John owes every man money then Sam pays him
(64) ?If a/some large man loves every woman, then Sally loves him

However, the above constraint only applies to *distributive* Type II quantifiers such as *every* and the reading of (64) in which the same man loves every woman is acceptable. This can be seen more clearly by rephrasing (64) as (65). [26]

(65) If someone wins each race, Sally will admire him

Distributive quantifiers inside scope islands can coindex with pronouns outside the islands, but apparently only when they receive collective interpretations. In such cases, it is easy to show that the pronouns cannot be bound by the distributive quantifiers. For example,

(66) If few people register for the course they will learn more

does not mean that few people are such that if they register for the course they will learn more. Pronouns such as *they* in (66) are sometimes known as "E-type" pronouns (Evans 1980). The antecedents of E-type pronouns must presumably always be existential quantifiers. Therefore, the constraint on coindexing pronouns from within scope islands distinguishes existential from distributive quantifiers but not Type I from Type II quantifiers. [27]

We may then distinguish three types of pronoun antecedent: (a) referential terms and extra-linguistic objects, (b) distributive quantifiers and (c) existential and definite NPs. [28] Can this classification scheme be used as evidence either for or against making a referential/quantificational distinction for indefinites or definites?

Pronouns anaphoric to distributive quantifiers are generally treated as the bound variables of standard logic. One way of handling the binding of pronouns by distributive quantifiers across discourse is to use a process of "scope expansion" to bring pronouns within

the anaphoric scope of the quantifiers. This will give the correct scope relationships [29] in

(67) Every villager owns a donkey. However, he doesn't ride it

E-type pronouns have been interpreted in different ways. Lepore & Garson (1983) make a distinction between "semantic" and "anaphoric" scope. The former determines how the truth conditions of a sentence will be evaluated and the latter which variables are bound by a a quantifier. They are able to separate these two types of scoping by modifying the rules of interpretation. Their method handles the scoping of sentences containing a variety of logical connectives such as *and, or, unless* and *if-then*. In practice, their system consists of a set of prolog-like rules and the scoping of quantifiers and other logical operators is determined by the order of applying the rules. This method allows a quantifier inside an initial-*if* clause to anaphorically bind an external pronoun without the semantic scope of the quantifier being widened. This account does not appear to make any distinction between the handling of distributive and existential quantifiers. However, the account has some difficulty in interpreting pronouns whose antecedents are plural indefinites (as well as other plurals), where making this distinction may prove to be useful.

Most commonly, E-type pronouns are interpreted as standing for definite descriptions. A number of attempts have been made to describe how suitable definite descriptions may be constructed, usually by combining the descriptive content of the antecedent NP with contextual information (see discussions in Lepore & Garson and Schubert & Pelletier (1987b)). The interesting aspect about this work is that there is a generally felt need here to convert indefinite descriptions into definite descriptions, even though the discourse by itself provides no simple way of doing this (pronouns of "laziness", for example, can be shown to be inadequate). This is quite similar to the problem of how to interpret specific indefinites which appear to acting very much as definite descriptions, but with an invisible uniqueness condition. [30] This problem will be discussed further in section 3.9.

Although the interpretation of pronouns anaphoric to names, indexicals and extra-linguistic entities as referential terms is widely accepted, it is not uncontroversial. There is some evidence that pronouns anaphoric to names should be treated as bound variables since they are subject to certain syntactic constraints such as "disjoint reference" and "obligatory coreference", shown in (68) and (69), respectively. Similar examples are discussed in Enc (1981:9).

(68) Bill saw him
(69) Mary thinks of Sue that she likes Harry

In (68) *him* cannot refer to Bill (a reflexive must be used in this case) and in (69) *she* must refer to Sue. Enc notes that the above pronouns may be interpreted as indexicals, in which case a pragmatic explanation must be given of the restriction on coreference, or as variables subject to certain constraints on coreference (or "binding"). She mentions some evidence that the former account does not appear to be completely satisfactory. It could be argued further that there is no need for a separate class of referential terms since these can always represented as definite descriptions. Therefore, evidence from data on pronoun-binding points more towards a difference between existential and distributive quantifiers than between quantifiers and referential terms.

3.8. Referential and Attributive Definites

There are two standard ways of referring to objects in the world: [11] directly, by "pointing", and indirectly, via a description. In terms of the model theory described in chapter one, directly referential terms are interpreted as indexicals and decriptive NPs as variable binding operators. These are sometimes known as referential and attributive NPs, respectively. The former have their denotations passed directly to the proposition and the latter a description. There is a long standing debate on whether definite descriptions should be interpreted as pointing or descriptive. [12] More recently it has been proposed that both of these

viewpoints are correct and that definite NPs are lexically ambiguous between having a referential and an attributive interpretation. The latter view has been advocated in Donnellan (1966), Wettstein (1981), Fodor & Sag (1982) and Barwise & Perry (1984). The purpose of this section will be to examine the evidence for making a lexical distinction for definites. This is important since there can be no referential indefinites without referential definites; therefore, if there is little evidence for the latter this can be used as an argument against the former.

The first argument for treating definite NPs as semantically ambiguous was put forward by Donnellan (1966). He claimed that the descriptive content of a referential NP was used simply as a means of drawing attention to an object and played no part in the predication made of the object. He illustrated this with the sentence

(70) Smith's murderer is insane

which can be used in two ways. It may be used in the courtroom to comment on the crazy behaviour of Jones, who is on trial for the murder of Smith, or it may be used by someone at the scene of the crime as an opinion about "whoever" murdered Smith. Donnellan suggested that the difference between the two uses of *Smith's murderer* was shown most clearly in the case when it had no real referent. He claimed that in this case the referential use could result in a meaningful proposition, namely that Jones is insane, but that the attributive use would not.

Donnellan's proposal has been criticized because it blurs the distinction between what a speaker means to say and what he actually says. As one critic notes, Donnellan simply shows that "one can succeed in making a hearer think of something a by means of expressions that do not in reality as the language goes correspond with a".[33] However, Wettstein (1981,1983) argues that Donnellan's conclusion is essentially correct, but that it is misleading

to consider the case in which the definite description does not correctly refer. He first claims that most or all definite descriptions are "indefinite" or "incomplete", in the sense that the expression *the table* does not by itself describe a unique object, and that such descriptions cannot be completed by the context since there is in general no unique way to complete them. He argues that this invalidates both the Russellian and Fregean accounts of definite descriptions. He then addresses the question of how an indeterminate expression can be used in a determinate proposition. He claims that it is necessary to interpret some definite descriptions as directly referring terms, the context providing a demonstrative rather than completing the description. This would also allow the descriptive content of attributive definites to be completed by "anchoring" [34] some of the terms in the description to the real world.

Some criticisms of the referential/attributive distinction for definites will now be given. The two main features of referential definites which distinguish them from attributive definites, and which therefore need to be examined, are: (a) the objects denoted by referential terms are "recognized by", "identifiable by" or are "directly known to" the speaker and (b) the descriptive content of referential terms plays no semantic role but merely serves to identify the object being referred to.

First of all, what does it mean for the speaker (or for an individual described in the discourse) to recognize or to have direct acquaintance with an object? As may be expected, the line between a referential and an attributive use of a definite description can be quite fuzzy. Wettstein notes that the referential/attributive distinction depends on the "intentions" of the speaker (and may therefore presumably not be perceived by the addressee). This is not suggestive of a semantic ambiguity. Furthermore, the distinction may not be clear to the speaker either. The extent to which a person "recognizes" or is "acquainted with" an object forms a continuum; however, it is possible to make a distinction between reference made to individuals or objects which are encountered for the first time and "anaphoric" reference

made to those which have been previously encountered. If a distinction needs to be made, it should be made between new (attributive) and anaphoric descriptions rather than between attributive and referential descriptions. Whether or not the "anaphoric" referent is anchored to the real world is irrelevant to this distinction.

Once an entity is introduced into the discourse via a definite or indefinite description a "conceptual object" is formed which may then be referred to subsequently. In this respect, it acts no differently than if it were a real, directly perceptible object. Real, fictional and hypothetical entities may be referred to in the same way once they are introduced into discourse, and are just as "recognizable" or "identifiable". *Hamlet* and *Caesar* have the same linguistic properties even though the former is a fictional character. Schoorl (1980) criticizes the referential/attributive distinction. He claims that when a new object is brought into discourse a "dossier" is created for the object and that subsequent anaphoric reference is made to dossiers and only indirectly to "real objects". [35]

The descriptive content of all (non-generic) definite NPs plays two roles: it identifies a referent, whether previously encountered or not, and it plays a pragmatic role. For example, in

(71) The person who broke this window is crazy

the speaker may use the definite description to refer to a person he already knows, to someone he sees break the window but does not know, to a shadowy figure he sees throw a brick at the window but would not be able to identify or to an inferred person who he assumes exists after seeing a broken window. [36] There is no point at which we can say that the description is being used simply to identify an object rather than to make an assertion about an object. On the one hand, even if the speaker is using the description merely to identify a friend of his, the sentence will have to be judged false (or meaningless) if the description is inaccurate. On the other hand, even if the description is being used completely attributively,

to describe the inferred window-breaker, it need not be directly related to the predication being made. That is, the speaker may think the inferred window-breaker is crazy, not because he broke the window but because he didn't report it. The relation between the content of any description and the main predication being made is solely a matter of pragmatics.

3.9 Specific Indefinites: A Lexically Distinct Class?

In the previous six sections we have seen that the ability of indefinites to escape from scope islands, to widen scope while allowing VP deletion, to support non-restrictive relative clauses and to support extensive cross-discourse anaphora cannot be accounted for by giving indefinites an indexical interpretation. The evidence seems to point instead to a major distinction between the scoping of existential and distributive quantifiers.

How should the complex scoping behaviour of indefinites be accounted for? The two main questions which will be considered in this section are (a) whether a lexical distinction should be made between specific and non-specific indefinites and (b) whether existential quantifiers (specific or non-specific) should be treated as unscoped (moveable) operators or as unmoved elements whose scope dependencies are expressed in the restriction clause. In this section, we will use the term "specific indefinite" to refer to an indefinite which appears to have a "uniqueness" condition associated with it but which may be functionally dependent or be embedded inside an opaque context. It will assumed that the treatment given to definites (ie. as moved or an unmoved elements) will correspond to that given to specific indefinites.

The specific/non-specific distinction appears to be quite closely tied to the scope-widening of indefinites. In section 3.1, we saw the relation between specificity and the scoping of indefinites relative to opaque contexts. It also appears to be the case that indefinites are generally understood specifically when they widen scope relative to quantifiers and negation, even though it is sometimes possible to get non-specific readings. In (72) and (73) it seems (arguably) that we give *a book* a specific interpretation in the readings in which

it takes wide scope.

(72) Each person read a book
(73) John didn't read a book

To the extent that this is true, it seems that the process of "widening" the scope of an indefinite is closely related to giving it a specific interpretation. While this may not be too convincing in the two above examples, widening the scope of an indefinite relative to a scope island forces the specific interpretation. This can be seen by considering the examples of Fodor & Sag, repeated from section 3.2 as (74) and (75).

(74) Each teacher overheard the rumour that a student of mine had been called before the dean
(75) If a student in the syntax class cheats on the exam, every professor will be fired

Although it is possible to represent the specific reading of (74) in terms of an existential quantifier having wide scope, this will not work for (75) if the *if-then* construct is interpreted as a material conditional shown as (76). [37]

(76) (Ex:student [¬[x cheats] v [every professor will be fired]])

This will be true if there exists any student in the syntax class who does not cheat, which is clearly not the intended meaning. Even if the material conditional representation is not used, the problem remains that an indefinite which acts as though it widens scope outside an *if*-clause must be given a specific intepretation.

Three possible ways of accounting for this will be considered here: First, the proposal of Fodor & Sag might be modified so that a lexical distinction would be made between specific and non-specific indefinites, rather than between referential and quantificational indefinites. One advantage of this proposal is that there would be no need to posit a parallel lexical

ambiguity for definites. A second approach would be to represent the scope dependencies of all indefinites (and definites) by scope-widening, in conjunction with a pragmatic procedure which determines the degree of specificity of indefinites. A third approach would be to treat all indefinites and definites as unmoved elements and to represent their scope dependencies (relative to quantifiers, opaque operators and other operators if possible) by modifying their restriction clauses.

The third approach provides the simplest way of dealing with the problem of specificity but it creates the new problem of how the scope dependencies should be represented. To represent the scoping of indefinites relative to opaque operators it might be possible to represent the scope relations by constructing a definite description which includes the restriction clause "[x identifiable-by y]", where x is the variable bound by the indefinite and y is either the speaker or an individual referred to in the discourse. The three specific readings of

(77) Mary thinks that John wants to marry a (certain) blonde

could then be obtained by specifying the parameter y in the above restriction clause, where y can be either John, Mary or the speaker. The same method could be used to handle the scoping of definites relative to opaque operators. However, since the details of how to represent the functional dependencies of unmoved indefinites and definites are obscure at the moment, indefinites and definites will be treated as unscoped quantifiers in this study.

There is no question that indefinites such as *a book* in (72) can be *understood* in two ways: specifically or non-specifically. The question is whether this distinction should be accounted for by semantics (ie. by a lexical distinction) or by pragmatics. As Fodor & Sag mention, the latter is preferable on grounds of parsimony, in the absence of evidence to the contrary. However, if a pragmatic account is adopted, it will be necessary to explain why

indefinites are so often used in a unique sense. The specific interpretation which must be given to indefinites which escape from scope islands has already been described. Two other related problems will be briefly mentioned.

A second problem is how to explain why the presence of the modifier *certain* usually forces the indefinite to take wide scope over negation and *if* clauses as in (78) and (79).

(78) John doesn't love a certain girl
(79) If a certain girl comes to the party John will come too

The requirement of *a certain* to take wide scope here does not simply follow from its uniqueness since the indefinite *a symphony* in

(80) John did not hear a symphony which is twice as long as any other symphony

can scope inside the negation, even though it must be unique if it exists.

A third problem is how to explain the immunity of specific indefinites to the constraint on coreference proposed by DeCarrico (1980). She classifies modals and opaque verbs and adjectives into three categories, which in order of increasing "strength" are possibility, probability and necessity. She proposes a constraint on coreference which prevents a pronoun from being in a stronger context than its antecedent. Therefore,

(81) *It is possible that John will kiss a girl and certain that he will hug her

is unacceptable if *a girl* is non-specific but is acceptable if it is interpreted as a specific indefinite or is replaced by a referring term such as a name. Giving a special interpretation to specific indefinites might account for various other linguistic phenomena such as the possible immunity of specific, but not of non-specific, indefinites to certain syntactic constraints on coreference described by Hornstein (see footnote 27 in section 3.7). [31]

The decision as to whether a lexical distinction needs to be made between specific and non-specific indefinites must be postponed until the semantics of indefinites and anaphoric reference is more thoroughly worked out. Based on the evidence presented in this chapter, it is probably best to delegate the problem of reference, or specificity, to pragmatics. There is a close similarity between the problems of how to specify the uniqueness of specific indefinites and how to specify the uniqueness of E-type pronouns (E-type pronouns themselves may also be replaced, where appropriate, by specific indefinites especially when modified by *certain*). These problems have only very recently begun to be treated within the framework of formal semantics and at the moment it seems preferable to give a pragmatic account of the uniqueness of E-type pronouns (see Schubert & Pelletier, 1987b). In the next section a uniform interpretation of indefinites and definites will be given in which no distinction is made between non-specific and specific indefinites.

3.10. A Uniform Interpretation of Indefinite and Definite NPs

Plural indefinite and definite NPs differ from other quantifiers in the ease with which they can receive collective interpretations. Gil (1982) noted in his empirical study (see section 2.6) that for sentences which contained two plural indefinites people preferred "symmetric" readings, in which neither of the two indefinites was functionally dependent on the other. This stands in contrast to sentences with quantifiers such as *every* or *most* which only seldom have collective interpretations. [39]

It is therefore plausible to give a uniform interpretation to indefinites and definites by interpreting plural indefinites and definites as E- or ι-quantified "collections" with optional universal partitives. [40] If the initial logical translation of a definite or indefinite plural NP is t (which will denote a collection), the logical form after addition of an optional partitive will be $\langle V\ \lambda x[x \in t]\rangle$. This partitive is obtained by using the postprocessing rule

(82) $\quad ti \longrightarrow \langle V\ \lambda x[x \in ti]\rangle, \quad i=1,2, \ldots$

where ti is a variable over terms denoting level-i entities, level-0 entities are ordinary individuals and level-i entities (i > 0) are collections of level-(i-1) entities. This allows collective and distributive readings to be obtained without complication in the rules of translation for these NPs. [41]

The following are some examples of the translations for singular and plural indefinites. A plural noun such as *men* is translated as (PLUR man') where (PLUR P) is a predicate true of collections of entities of which *P* holds. The bare plural *men* (as an NP) is translated as (μ (PLUR man')) where μ forms a "kind" or "species" (see Pelletier & Schubert 1985; Schubert & Pelletier 1987a). The indefinite *a man* is translated as (a' man') where *a'* is lexically ambiguous between the existential quantifier E and an operator μ_1 which forms a "generic instance". The two translations are more accurately represented as $\lambda P<E\ P>$ and (μ_1 P) where the angled brackets in the former signify an unscoped operator (see section 5.1) and the round brackets a functional term. The translation of *two men* as a noun is (two' (PLUR man')), where *two'* modifies (PLUR man') so that it will be true of collections of size two. The syntactic-semantic rule pairs giving the translations for *a man* and *two men* are shown in (83)-(85).

(83) NP —> DET N, (DET' N')
(84) NP[PLUR] —> N[PLUR,-NUM], (μ N')
(85) NP[PLUR] —> N[PLUR,NUM], (a' N')

Rule (84) applies only to non-numeral NPs such as *men*, while (85) applies to numeral NPs such as *two men*. The semantic output of rule (85) for *two men* would be (86), with the two possible lexical disambiguations shown in (87) and (88).

(86) (a' (two' (PLUR man')))
(87) <E (two' (PLUR man'))>
(88) (μ_1 (two' (PLUR man')))

The (87) and (88) translations account for the particular/generic ambiguity in

(89) Two men can lift this table

and the optional postprocessing rule (82) accounts for the further distributive/collective ambiguity of the particular reading. [42]

In constrast to indefinites and definites, the determiners *most*, *few* and *every* are treated as being lexically ambiguous between the collective and distributive readings, with a strong preference for the latter. This is done to account for the difficulty in obtaining collective readings with these determiners. As an example, the two translations for *every* are (a) the universal quantifier V and (b) $\lambda P<$the $\lambda x[V\ P> \epsilon\ x]>$. Note that it does not appear possible to prefix NPs such as *most* with partitives whereas this can usually be done with indefinites. [43]

In the next chapter some heuristics will be given for the scoping of both the existential and optional universal quantifiers associated with indefinites. It will be seen that the interpretation of plural indefinites as collections with optional partitives fits in well with intuitive judgements about scope ambiguities. However, one aspect of this approach should first be noted. By optionally associating partitives with the two indefinites in

(90) Two men lifted six boxes

and then scoping, a total of eight readings will be generated, shown schematically below. The symbols X and Y represent the sets of men and boxes, respectively, and x and y the individual

men and boxes.

1. EX EY L(X Y)

2. EX EY Vx L(x,Y)
3. EX Vx EY L(x,Y)

4. EX EY Vy L(X,y)
5. EY Vy EX L(X,y)

6. EX EY Vx Vy L(x,y)
7. EX Vx EY Vy L(x,y)
8. EY Vy EX Vx L(x,y)

The first reading will be obtained if the post-processing rule is not used, the next two readings if the rule is applied to the first indefinite, the next two if it is applied to the second indefinite and the last three if it is applied to both indefinites. This has been verified testing the above sentence on the scoping program and the results are discussed in section 5.5 (examples 16-18). There are actually fourteen valid combinations of the existential and universal quantifiers, however, six of these are redundant and are removed by the scoping program. (The redundant readings result from the switching of adjacent existential or universal quantifiers. In cases in which adjacent quantifiers are commutative, the readings in which the order of the quantifiers does not follow their surface order are removed by the program.)

Although all of the above readings are distinct and would eventually need to be represented, it seems preferable at this point to make a distinction between representing functional dependencies and types of predication. The problem with the post-processing rule as it stands is that the addition of a partitive forces individual predication, even in readings in which it does not add any new functional dependencies. Thus the first two readings differ only in that the latter makes a predication of individual men, rather than of a collection of men. (Note that the sixth reading, which differs from the first reading only in that a predication is made of individual men and boxes, is the *complete group* reading discussed in

chapter two.) By slightly modifying the program to remove readings in which optionally introduced partitives do not create any functional dependencies (such readings would be considered to be redundant at this stage of processing), only the first, third and fifth readings would remain.

1. EX EY L(X,Y)
3. EX Vx EY L(x,Y)
5. EY Vy EX L(X,y)

This gives the desired result. Now the effect of the post-processing rule is solely to create functional dependencies and the type of predication being made, which may be quite complex, can be disambiguated gradually, after scoping, using meaning postulates and pragmatics.

Footnotes

1. However, it is difficult, and probably not possible, to draw a clear line between indefinites and "proper" quantifiers in terms of criteria such as those just described.

2. The term "identity" will be used loosely for the moment but will be considered in more detail in section 3.8.

3. There is an alternative way of representing this ambiguity by scoping rather than by positing a lexical ambiguity. A performative clause such as "I hereby assert that ..." or "I say to you that ..." can be added to the sentence and the indefinite scoped relative to this (Cole 1975).

4. However they go on to interpret such indefinites as referential terms which "do not participate in the network of scope relations between true quantifiers, negation, higher predicates and the like" (p.375) so it is misleading to refer to such indefinites as "escaping".

5. In order to make the point clearer, *the rumour* has been replaced by *a rumour*.

6. The term *implicit anaphora* will be used to refer to expressions which are implicitly understood to contain anaphoric pronouns. This is commonly observed with definite descriptions, as in the sentence *Each person asked about the previous owner* in which *owner* is understood to mean *owner of it prior to him or her*. There is evidence that some nouns such as *mayor*, *driver* and *owner* should be treated as binary predicates at the syntactic level and at the level of logical form. In such cases, a "slot" for an implicit anaphor is automatically provided for the logical form translation.

7. This version of the constraint is from Fodor & Sag (1982:377). The original in Sag (1976:74) is: "With respect to a sentence S, VPD can delete any VP in S whose representation at the level of logical form is a λ-expression that is an alphabetic variant of another λ-expression present in the logical form of S or in the logical form of some other sentence S', which precedes S in discourse."

8. The notion of *alphabetic variance* used is described informally in Sag (1976) as follows: "For two λ-expressions, λx(A) and λy(B), to be alphabetic variants, every occurrence of x in (A) must have a corresponding occurrence of y in (B), and vice versa. Also, any Quantifier in A that binds variables (in A) must have a corresponding (identical) Quantifier in B that binds variables in all the corresponding positions (in B). However, if there are any variables in A that are bound by some quantifier outside of λx(A), then the corresponding variable in λy(B) must be bound by the same operator ..." (p.72).

9. An example in which a universal quantifier may scope over both VPs is *Sandy greeted everyone when Betsy did* which as Sag notes (p.41) has a reading in which Sandy and Betsy greeted each person at the same time.

10. It may be possible to find some more convincing examples of VP deletion in which universal quantifiers appear to disobey the constraint. Two possible examples are

(1) ?First a *boy* set each table and then John thinks a *girl* did

(2) ?At each restaurant John tried one of its house wines and at each pub Bill did

It seems possible to obtain a reading of (1) in which the boy and girl depend on the window. This reading cannot be obtained by giving *each window* wide scope because of the attitude

clause. The two VPs in (2) contain non-alphabetic variants since the pronouns are bound variables of different universal quantifiers outside the VPs. However, such examples seem very hard to get and so the VP deletion constraint seems to work quite well here.

11. Fodor & Sag state that a consequence of the VP deletion constraint is that "a verb phrase cannot be deleted if its antecedent contains a quantified phrase whose scope is wider than the verb phrase" (p.375-6). This should presumably be supplemented with "unless the scope of the quantified phrase is wider than both the VP being deleted and its antecedent". Otherwise, readings in which an indefinite is given maximally wide scope constitute exceptions to the VP deletion constraint.

12. Similar examples are given in McCawley (1981:399).

13. It has been pointed out to me that the VP deletion may be more difficult if a sentence such as (34) is passivized. However, it still seems to be acceptable, even though a bit more difficult to get. As an example, *John is ready to be taken home, and Mary is ready to (be), also* seems reasonably acceptable, although *?The steak is ready to be eaten by the guests, and the chicken is ready to be, also* does not. Adding *by Bill* to the first sentence also makes this much less acceptable.

14. There may also be some other problems in applying the VP deletion constraint to variables bound by quantifiers, as opposed to by lambda operators. One problem discussed by Sag (p.117) is how to account for the acceptability of the "transparent" reading (in Quine's sense) of *Bill wanted Betsy to read everything that Sam wanted her to read, and Peter did, too.* This requires some modification of the constraint, since the universally quantified variables occur outside the subjects and therefore are not alphabetic variants. The modification Sag proposes is to employ double lambda abstraction. An alternative would be to widen the scope of the universal quantifier, although this is generally not possible over conjoined clauses.

15. However, if *I* is replaced by *Joe* it seems possible to get the intermediate reading if, for example, both Sandy and Chris heard a rumour that Joe beat someone and Joe is a lousy player himself. This would violate the VP deletion constraint, but only if existential quantifiers are scoped with opaque operators in the way suggested by Fodor & Sag.

16.It is always possible for the noun expression of a quantificational NP to support a non-restrictive relative clause by a form of "generic" anaphora. An example is *No students in this course, all of whom were carefully selected, cheated on the exam.*

17. For some reason, it does not seem possible to equate two descriptions which both support non-restrictive relative clauses as in **The owner of this store, who lives nearby, is the secretary of the club, who organized the meeting* whereas this is possible when both descriptions support restrictive relative clauses as in *The person who owns this store is the person who organized the meeting*. It might be argued that this is indicative of a difference between referential and attributive definites: two referential definites, which can support non-restrictive relative clauses, cannot be equated whereas two attributve definites can. However, this is probably just a stylistic constraint, rather than a constraint on equating two referential terms since it is perfectly correct to say *Clark Kent is Superman.*

18. The *de re* interpretation means that Melinda has a particular motorcycle in mind. This interpretation is described in Fodor (1970).

19. Hornstein also describes a third category. Type III quantifiers share properties of Type I and Type II quantifiers, having maximally wide scope when originating in some syntactic

positions but being clausebound when originating in others. Examples include certain *wh* quantifiers and *ne-personne* in French.

20. The ECP is a constraint which prevents "long movement" of quantifiers in the subject position in tensed clauses. For example, *Who expects that John bought what?* is acceptable but *Who expects that what was bought by John?* is not (p. 68). However, such examples do not always seem very convincing.

21. For example, Hornstein says (p.21) that *?Everyone loves a certain woman* can only have the reading in which *a certain* takes wide scope; however, it is reasonable to state *Every man loves a certain woman, namely his girlfriend.*

22. It is claimed (p.51) that *Everyone didn't like the play* cannot have the reading "Not everyone liked the play", although it clearly can. It is also claimed (p.28) that *John didn't kiss every woman at the party* is "unambiguous, 'every' being mandatorily interpreted as lying within the scope of 'not'". Yet, it is later mentioned (p. 32) that *Someone doesn't like every woman* is "ambiguous, with either the universal or existential enjoying wide scope".

23. Some people (eg. Evans 1980) argue that proper names should be interpreted as variable-binding operators; however, it will be assumed here that pronouns coreferential with names are unbound.

24. Recall that although Type I NPs are interpreted relative to a domain of entitites, they have a name-like logical syntax and do not act as variable binding operators.

25. A further example given by Hornstein is *Take any number. If you multiply it by two, the result is even* which is used to show that the Type I NP *any* can support cross-discourse anaphora. However, the same example works well with both existential and universal Type II quantifiers: *Pick a number at random. If you multiply it by two, the result is even* and *Take each number in turn. If you multiply it by two, the result is even.*

26. This is really just a variation of the well known donkey sentence *If Pedro owns a donkey he beats it.*

27. Hornstein does present some examples which may show a more convincing difference between Type I and Type II quantifiers. Two examples of his are:

(1) *That some wooden house is highly inflammable makes it expensive to insure
(2) ?That he might be sent to the front doesn't bother some good soldier

However, the awkwardness or inacceptability of (1) and (2) if the indefinite is given a non-specific interpretation, as opposed to when it is replaced by *a certain*, may be partly stylistic. Sentence (2) is acceptable if *some* is relaced by *no* (3) or by a functionally dependent definite (4).

(3) That he might be sent to the front bothers no good soldier
(4) That he might be sent to the front doesn't bother the bravest soldier in each regiment

28. Almost all NPs may also support a form of generic anaphora, but this need not be considered here. The indexical *I* can never receive a generic interpretation, but *you* can as in *You don't have to be a skier to enjoy winter.* Pronouns can also be used non-specifically as in *If you get this message, notify the coast guard* [message in a bottle].

29. There may be some difficulty when a sentence containing a distributive quantifier is first asserted and then later negated or conditionally asserted. In such cases it seems that it is not sufficient to simply expand the scope of the quantifier but that the whole sentence which was previously asserted must now be placed inside the negation or conditional operator (this makes sense since the truth condition of the whole sentence is now being called into question). Two examples which illustrate this are:
(1) All of the villagers own a donkey. No they don't - I know one who doesn't
(2) All of the villagers own a donkey. If they do, then why can't I find one?
However, these sentences may be special cases and require special treatment since previous assertions are being modified.

30. The problem of the whether definite descriptions really are "unique" has been debated, for example by Wettstein (1981) and Wilson (1984). It is clearly easy to use a definite description to refer to a non-unique object previously introduced by a non-specific indefinite. An example is

(1) John owned a blue van. The van had a bent fender.

Suppose that John had two blue vans, one of which had a bent fender. Then the truth conditions of (1) will depend on how *the van* is interpreted. If it is interpreted as a bound variable, (1) will be true if it is satisfiable, which in this case it is. However, if *the van* is interpreted as a uniquely referring description then (1) can be either true or false, depending on which interpretation is given to *a van*. It seems quite difficult to decide which of these accounts is to be preferred. This problem is discussed in Schubert & Pelletier (1987b).

31. The same is true for time, place, nominalized states and events, imaginary objects etc.

32. Wettstein traces the "pointing" viewpoint back to John Stuart Mill and the descriptive one to Frege and Russell.

33. Comment of Castenada, quoted by Wettstein (1981:244).

34. The term "anchoring" will be used informally. It is taken from Barwise & Perry (1983:chapter 11) where it is used in connection with referential terms which are propagated through discourse via anaphors. Such co-referring referential terms are said to be "anchored" to one another. By analogy, we can say that such referential terms are also anchored to objects in the real world.

35. We could consider reference to involve two domains: the domain of real objects and the domain of abstract or conceptual objects. Equivalence predications made between symbols for real and conceptual objects could then be stored (when available) as part of the dossiers associated with conceptual objects. Such dossiers should contain propositions about attributes learned through perception, such as height, eye colour and voice-pattern, as well as those described in the discourse. In one sense, all descriptions are "anaphoric" since they must refer to conceptual objects which have already been formed in the mind of the speaker, for example through perception or through inference. Although we tend to think of objects as "units", infants must learn to recognize all objects gradually through sense impressions and as adults we use a similar method to initially distinguish and later recognize unfamiliar objects. Although any given object will presumably evoke different "dossiers" in different people, all that matters is that corresponding dossiers, or conceptual objects, in different people can be "anchored" or matched. The way in which this matching is done is presumably the same for real and abstract objects.

36. We frequently use definite descriptions to refer to inferred objects. On passing a shop, we may refer to *the owner* or *the cashier* or in a conversation about an apartment we may refer to *the kitchen* or *the fridge* even though these may not exist or may not be unique. However, once objects such as these have been introduced into discourse they may be referred to as if they were real objects.

37. This problem was pointed out to me by Jeff Pelletier.

38. The effect of the modifier *certain* is complex and it is wrong to suppose that indefinites modified by *certain* need to be very specific. *A certain* is used somewhat non-specifically in *Is there a certain time after which they lock the door?*. An even less specific use of *a certain* is in *In many small towns, there is a certain cafe where the locals like to meet. A certain* is also commonly used in a non-precise way with abstract nouns as in *Children need a certain amount of discipline.*

39. It is not possible to draw a line between NPs which can act collectively and those which cannot. The quantifiers *each* and possibly *every*, can probably not support collective predication, although *everyone* can in, for example, *Everyone lifted together*. *Most* also can on occasion, as in *Most of the people formed a circle* but usually sentences such as *Most boys like most girls* have only the two scoped, distributive readings. The same NP may be used in both collective and distributive predications. Two examples are: *Everyone stood up, put on their hats and formed a circle* and *Two men, who (each) own a car, share this house.*

40. This proposal is also described in Hurum & Schubert (1986).

41. A similar rule may be used to add partitives to plural names, generics and indexicals. An example of a sentence containing an indexical requiring a partitive is *We will go our separate ways.*

42. Note that plural indefinites do not have a distributive generic reading. This observation provided some motivation for the introduction of the μ_1, or *generic instance*, operator.

43. Note the ungrammaticality of *each of most people*. In some cases, it is awkward to place partitives in front of plural indefinites which are used non-specifically. It is ungrammatical to say **John wants to catch each of three fish* when the indefinite is being used non-specifically, but this is probably just a pragmatic constraint. It is possible to say *John wants to get each of the three volunteers he will pick to choose his own project.*

Chapter 4

Scoping Heuristics

The scoping of an operator is determined by five main factors: (a) its lexical type, (b) surface position and (c) "structural category" and also (d) world knowledge and (e) the context of discourse. [1] The ways in which these five factors interact can be quite complex, and it seems that an essential first step in determining a set of scoping heuristics is to try to determine the separate contributions of each factor. Some attempt must then be made to combine these.

The aim of this chapter will be to determine some domain-independent heuristics for the preferred scope orderings of pairs of operators, taking into account the first three of the above factors. As an example of (a), there are major differences among different classes of quantifer, such as universal and existential quantifiers, and also among different members of each class. For example, there is a clearly defined hierarchy of the ability of universal quantifiers to widen scope, *each* > *every* > *all*. As a general rule in English scope order tends to follow surface order, and so (b) needs to be taken into consideration. Two general rules based on "structural categories" (see next section) are (i) quantifiers inside prepositional adverbials tend to scope outside quantifiers in the main clause and (ii) different types of clause have characteristic tendencies to trap unscoped operators, and verb phrases in the same position always form substantially weaker traps.

It will be argued that heuristics such as these are largely independent of pragmatic factors and it is hoped that they can provide a base on top of which heuristics based on world knowledge and the context of discourse can then be added. The unscoped operators considered will be quantifiers, coordinators and negation. In the next chapter, the problem of how the heuristic values associated with pairs of operators should be combined to give the the preferred readings of a sentence will be examined.

4.1. Structural Relations

Scoping preferences are strongly influenced by what we shall call "structural relations", which is loosely speaking the relation between any two operators in the parse tree or in the corresponding initial logical translation. It makes no difference which of these two forms of representation of a sentence we use, since the structural relations are preserved (assuming that information about surface order is present in the form of suffixes), but it is perhaps more convenient for the present to use the terminology of syntax. Thus we can refer to general relations such as *subject/object* or *negation/adverbial* or to more detailed relations such as *preposed-adverbial/full-subject* [2] or *shifted-indirect-object/postposed-adverbial-op*. Note that the detailed, or *composite*, relations take into account surface position, so that two parameters of scoping have now been combined into one. In the discussion that follows, the context should make it clear whether we are referring to basic or composite structural relations. The term "structural category" will be used to refer to the categories of individual operators, such as *full-subject* or *direct-object*. The relation between syntactic and logical categories should be clear: *subject:first-term, direct-object:second-term* and *adverbial:function*.

A distinction is sometimes made between the scoping of "clausemates", such as subject and object, and the scoping of quantifiers at different levels of embedding (eg. Ioup 1975; van Lehn 1978). We will loosely refer to these two types of scoping as *horizontal* and *vertical*, respectively. This distinction proves to be a very useful one to use as a guide to the development of scoping heuristics, although in some cases the distinction is not that clear. [3] Using the terminology introduced above, we can classify structural relations as either horizontal or vertical. The most clearcut cases of vertical relations are those between a predicate having a clausal term and an operator inside the term and between a sentential connective such as *if-then* and an operator inside one of the clauses it connects. However,

there are some less clearcut cases.

Negation can be represented logically as a function (or predicate) applied to a clause, and therefore forming vertical relations with operators inside the clause. This appears to be the interpretation used in the scoping program of Hobbs & Shieber (1987). However, in this study, negation will be treated as an unscoped operator which is scoped horizontally with other operators in the same clause, such as subject and object. One reason for doing this is to allow the negation to widen scope over the subject and over operators inside post-posed adverbials (which are applied to the front of a clause in the logical translation). A similar treatment is given to quantificational adverbs. Quantifiers inside clausal adverbials must first be scoped vertically with the predicate of the clause modified by the adverbial, and then, if they widen scope, be scoped horizontally with subject and object quantifiers. However, prepositional adverbials are considered to form no trap for quantifiers, so that quantifiers inside them can be directly scoped horizontally with subject and object quantifiers. The above treatment simplifies the developement of heuristics for subject, object and adverbial operators and negation.

How much detail about structural relations needs to be incorporated into the scoping heuristics? For any pair of operators, the three basic factors to consider are (a) whether one of the operators embeds the other, (b) which operator appears first in surface order and (c) the structural relation between the two operators. Just in how much detail (c) needs to be specified remains to be determined, although some evidence will be presented which suggests that a considerable amount of detail is needed for accurate heuristics.

The simplest general rule is that an operator which embeds or precedes another operator tends to scope outside it. This tendency is strengthened if the operator which appears first in surface order has been "shifted" in front of the other operator, where shifting is used here in a general sense to include topicalization, the shifting of objects and, perhaps

by analogy, the preposing of adverbials (if the latter are thought of as being postposed by "default"). ' These two general prinicples, when augmented with lexical information, could by themselves be used to obtain a useful set of heuristics. The most general rule for vertical scoping is that subordinate clauses (with subject) form very strong scope traps, bare verb phrases intermediate traps and prepositional phrases, serving as noun complements or as adverbials, very weak traps (or no trap at all).

In the later sections of this chapter, we will examine some empirical data to test the usefulness of these general principles and to combine them with lexical information. First, some previous attempts to define some linguistic heuristics for horizontal scoping will be described.

4.2. Horizontal Scoping: The Hierarchies of Ioup and van Lehn

Some quite detailed studies of scoping preferences were made by Ioup (1975) and van Lehn (1978). ' The effects of lexical type, surface position and "grammatical function" (ie. structural category) on scoping were all considered. Ioup gives the following hierarchy :

(1) each > every > all > most > many > several > some > a few

in which the ordering indicates the tendency to take wide scope. This hierarchy is supposed to hold independently of the structural category of the quantifier; however, Ioup notes that *each* is the only quantifier which consistently has a preference for wide scope. Ioup mentions that the universal quantifiers appear first in the hierarchy and that the hierarchy correlates with the size of the set being selected. She also notes that it is very difficult to place the singular indefinite in the hierarchy, but that it probably should be quite high. Van Lehn gives a partial hierarchy which corresponds to the first part of (1):

(2) Each > every > all of the > all the > other plurals

He notes that the ordering in (2) is inversely related to the ability to support a collective predicate such as "meet".

Ioup also describes a hierarchy of "grammatical function" based on a questionnaire given in fourteen languages (p.57):

(3) topic >
full subject >
deep subject = surface subject >
indirect object >
prepositional object >
direct object

She includes topic as a separate category (rather than as an shifted object) since many languages have a grammatical category for topic. Ioup claims that surface order by itself is not a parameter of quantifier scope; however, this claim will be disputed here, at least with reference to the English language.[7] To start with, the above hierarchy partly follows surface order (in English); the two main exceptions are first, that the surface subject is not predicted to scope outside the deep subject and second, that the scope preferences of different objects do not depend on their relative surface positions. These two points will now be discussed in turn.

Ioup uses the following four sentences to support her claim regarding the scoping of deep and surface subject.

(4) Every girl took a chemistry course
(5) A chemistry course was taken by every girl
(6) Every chemistry course was taken by a girl
(7) A girl took every chemistry course

She claims that (4) has only the distributive ("individual") reading, that (5) and (6) have preferred distributive readings and that (7) has a preferred non-distributive ("collective") reading. This interpretation supports the hierarchy in (1) rather than the influence of surface

order since "every" is both deep and surface subject in (4), deep or surface subject in (5) and (6) and object in (7).

However, it is difficult to come to any firm conclusions based on these sentences alone. First of all, (4) seems to have a clear ambiguity, although as it stands the distributive reading is much preferred. It is not obvious to me that there is any real difference between the scope preferences of (4) and (6); both have preferred distributive readings. Similarly, the difference between (5) and (7) is not that great, although *every* is a bit more likely to take wide scope in (5) (this difference is less noticeable when *a* is replaced by *some*). Therefore, the deep and surface subject may have a slightly stronger preference for wide scope over the direct object than the surface subject over the deep subject (surface object). However, the effect of surface order seems actually more noticeable than that of grammatical relation; the indefinite is much more likely to take wide scope in (5) than in (6).

The influence of surface order can be seen more clearly by using examples with two plural quantifiers:

(8) Two boys kissed three girls
(9) Two boys were kissed by three girls
(10) Few people like everyone
(11) Few people are liked by everyone

In both (8) and (9) it is very hard to get the reading in which the set of boys depends on the girl, unless this is made explicit by specifying *each of three girls*. In both (10) and (11) *few* has a strong preference for wide scope. We might modify the first part of Ioup's hierarchy to get (12).

(12) topic >>
deep and surface subject >
surface subject >
deep subject >
deep and surface objects

The second claim made by Ioup is that the scoping of quantifiers serving as objects is determined solely by grammatical function and not by surface order. Ioup uses the following examples to support her claim:

(13) I told every child a story
(14) I told every story to a child

(15) I had many conversations with a friend
(16) I had a conversation with many friends

Her claim seems at least partly correct. In both (13) and (14), we prefer to scope the indirect object (the child) outside the direct object and in both (15) and (16) the prepositional object (the friend) tends to scope outside the direct object, although these are only preferences and could easily be swayed by context.

However, Ioup does not directly test for the *dative shift* effect, that is the effect of shifting the indirect object relative to the direct object. Van Lehn found in an empirical study that the preference for the reading in which the indirect object scopes outside the direct object was 30% for (17) and 100% for (18).

(17) John lent each book to a friend
(18) John lent a friend each of his books

Considerably more evidence for the dative shift effect will be given in section 4.5.

In summary, it seems clear that both "grammatical function" and surface order, as well as quantifier type, must be taken into account. Van Lehn describes two hierarchies based on grammatical function. The first is based on the c-command and is a modification of a hierachy previously proposed by Reinhart (van Lehn, p.37):

(19) preposed pp and topicalized np >
subject >
sentential pp and adverbial np >

verb phrase pp >
object

The second hierarchy he mentions is simply that determined by surface order and is the same as (19) with the order of the last three catagories reversed. Note that no distinction is made between direct and indirect object here, or between different types of subject. In empirical studies, van Lehn found that the two hierarchies fared about equally well as predictors of scope ordering.

4.3. Scope Ordering Weights : Introduction

In the remainder of this chapter, an attempt will be made to derive some detailed heuristics for the scoping of pairs of operators based on (a) their structural relations and (b) the lexical class of each operator. The only way to obtain such heuristics is empirically, by examining a large number of sentences containing different combinations of operator types in different structural relations to one another. The data presented in this chapter have been compiled from a large number of test sentences, a few of which are shown in Appendix I. An attempt has been made to keep the sentences as simple and pragmatically neutral as possible. Each sentence contains one or two unscoped operators, which may be quantifiers, coordinators or verb negation, and is followed by a value between 0 and 1 which will be referred to as a *scope ordering weight* or more simply as a *scoping weight*. The scoping weight indicates very approximately the preference for the reading in which the second operator (in order of appearance) widens scope outside the first. For example, the weight 0.7 in

(20) Some boy loves each girl 0.7

attaches a weight of 0.7 to the reading in which *each* scopes outside *some* (ie. there may be a different boy per girl) and correspondingly a weight of 0.3 to the alternative reading in which *some* scopes outside *each*. An example involving a vertical relation is

(21) Some tourist visiting every city took pictures 0.5

which assigns a weight of 0.5 to both possible scope orderings. The use of a 0 to 1 scale is quite arbitrary and some alternatives will be discussed in the next chapter. Very roughly, the values can be interpreted as follows, where the weight indicates the preference for giving the operator which appears second in surface order wide scope:

1.0 - unambiguous wide scope
0.99 - virtually unambiguous (there may possibly be exceptions)
0.95 - wide scope except for very rare cases
0.9 - very strong preference
0.8 - strong preference
0.7 - moderate preference
0.6 - slight preference
0.5 - no preference for either reading

Values between 0 and 0.5 have the same interpretations, but for the first quantifier having wide scope. Some of the data obtained from individual sentences has been compiled into a few tables, listed in Appendix II. These tables form the basis for the major portion of the scoping heuristics used by the scoping program, and will be discussed in some detail in the sections which follow. First, however, some comments should be made about the validity and reliability of the data.

Although a large number of sentences were examined, this was only barely sufficient to cover the many different *patterns* which were under consideration, a pattern being a combination of a structural relation and a pair of operators. Consequently, in most cases only about two sentences of each pattern were tested. Therefore, the data should only be thought of as rough approximations. In addition, the large number of sentences being tested precluded the use of a questionnaire, and so the data reflect the bias of one person (myself). For this reason, no attempt has been made to give any statistical measurements of the data, although the results of making independent assesments of some of the sentences at a later time suggest

that the standard deviation would be about ± 0.1 to 0.2. [10]

However, these drawbacks should not effect the usefulness of the data. The primary purpose in obtaining this data was to examine some general problems such as: how scoping weights should be represented, how they can be simplified without losing too much accuracy (ie. the heuristics should be based on a model which is accurate, economical and psychologically plausible), how many different types of pattern need to be considered in order to deal effectively with the problem of scoping in English, how linguistic heuristics should eventually be combined with domain-dependent heuristics and, finally, how the pairwise scoping weights given here should be combined in the scoping program when scoping sentences which contain more than two interacting operators. In addition, it may be mentioned that since the data is based on the observations of one person, at least a consistent treatment is obtained. This may be the most important criterion for the present study. [11]

4.4. Quantifiers and Negation

The scoping of a quantifier with the negation operator is the simplest type of scoping, involving only three parameters: the position, structural category and lexical type of the quantifier. The first two of these can be combined into a composite structural category, as was mentioned earlier, to give only two parameters, a composite structural category and the lexical type. As an example, nine composite categories are shown in Table A2.1 (in Appendix II). The reason for using composite categories will become apparent in the next section. Some examples of the sentences on which the data in the table are based are given in Part 1 of Appendix I. Note that the scoping weights shown indicate the preference for the quantifier scoping outside the negation.

Table A2.1 shows clearly the different contributions of each of the three parameters to the scoping weight. The position of a quantifier with respect to the negation operator appears to be the most important factor in determining its tendency to take wide scope.

Quantifiers inside preposed adverbials must scope outside the negation, non-universal quantifiers in the subject position generally scope outside the negation, and quantifiers in the object position generally scope inside the negation. There appears to be no noticeable difference between quantifiers acting as full or surface subjects or among quantifiers in different types of object position. However, quantifiers inside postposed adverbials have quite a high tendency to scope outside the negation, thereby breaking the direct correlation between position and scoping. Therefore, the basic category of the quantifier is a necessary parameter of scoping, in addition to position.

Universal quantifiers have a surprisingly weak tendency to scope outside the negation operator, both when in the subject and and in the object position. This contrasts with their tendency to take wide scope with respect to other quantifiers, noted by both Ioup and van Lehn. [12] The scoping of existential quantifiers depends very much on pragmatics, and so it is difficult to give accurate scoping weights here. However, if indefinites are used non-specifically, those in the subject position usually scope outside the negation [13] and those in the object-position inside. The hierarchy of universal quantifiers *each* > *every* > *all* > *both* is shown quite clearly in table A2.1, and there is some evidence for a hierarchy of indefinite quantifiers *some* > *three* > *several* > *many*. The first hierarchy correlates inversely with the ability to support collective predication, with the exception of the last quantifier, *both*, which cannot act collectively. The second hierarchy correlates directly with the ability to act collectively (and therefore to escape from scope islands), although all of the indefinites can support collective predication.

Although indefinites may introduce two quantifiers, an existential quantifier and a universal partitive (see section 3.10), it seems that the two quantifiers, if both are present, scope as a unit relative to the negation operator. As evidence for this, although (22) is ambiguous, (23) does not appear to be and similarly (24) appears to have only two readings and not three.

(22) John didn't kiss Sue and Mary
(23) John didn't kiss (each of) those two girls
(24) John didn't kiss two girls

That is, (23) is presumably false if John kissed one but not both girls, meaning that the implicit partitive, if present, must scope outside the negation. Perhaps this should only be treated as a preference. In contrast, (25) will be false if John kissed either of the girls.

(25) John didn't kiss both girls.

The quantifiers *few* and *no* have, if anything, the strongest tendency of all the quantifiers to scope outside the negation from the subject position. In contrast, *few* does not appear to be able to widen scope from the object position, if utterances such as

(26) John didn't kiss few girls. (He kissed many)

are acceptable. We will see later that in general, a distinction needs to be made between the inherent tendency of a quantifier to widen scope over a preceding operator and to maintain wide scope over, ie. to "trap", succeeding operators. *Few* and *no* have little ability to widen scope but create very strong scope traps when in the subject position. Therefore, at least two separate hierarchies are needed, if general hierarchies are at all possible.

4.5. Horizontal Scoping of Two Quantifiers

The scoping of two quantifiers depends on six parameters: the position, category and lexical type of each of the quantifiers. The first two parameters can be combined into the composite categories described in the previous section. Since there are two quantifiers, we need to consider the relations between the categories. Although the number of possible relations is large, probably only a relatively small number need to be considered. Some of the horizontal relations used by the scoping program are shown in Table 4.1. The abbreviations

used to refer to the relations are shown on the left followed by the syntactic categories of the two operators, with the category of the first operator (in surface order) on the left. [14]

(a)	pre-pre	:	two preposed adverbials
(b)	pre-post	:	preposed and postposed adverbials
(c)	pre-others	:	preposed adverbial & all other categories
(d)	topic-subj	:	topic & subject
(e)	topic-others	:	topic & all other categories
(f)	subj-obj	:	subject & object
(g)	subj-post	:	subject & postposed adverbial
(h)	dir-indir	:	direct & indirect object
(i)	indir-dir	:	shifted indirect & direct object
(j)	obj-post	:	object & postposed adverbials
(k)	post-post	:	two postposed adverbials

Table 4.1 : Horizontal Relations

First, the effect of the horizontal relations on scoping will be examined with disregard for the two quantifiers present. The scoping of quantifiers inside two preposed adverbials appears to depend mainly on pragmatics. It is difficult to assign domain-independent heuristics to sentences such as

(27) On several occasions, at several meetings, someone fell asleep

In such cases a neutral scoping weight of 0.5 is assigned. However, there is a tendency for scoping order to be inversely related to surface order for quantifiers inside postposed adverbials. It is more natural to say

(28) John fell asleep several times at each meeting

than

(29) John fell asleep at each meeting several times.

There is a very strong tendency for quantifiers inside preposed adverbials to scope outside those inside postposed adverbials. This accounts for the awkwardness of

(30) ?Several times, John fell asleep at each of the meetings

in which we want to scope *several* outside of *each*, meaning that there were several groups of meetings at which John fell asleep, even though this clashes with our pragmatic expectations. [15]

For the remainder of this section, reference will be made to the scoping weights summarized in table A2.2(a-e). The scoping weights are assigned to the readings in which the second quantifier scopes outside the first. Each part of the table corresponds to a different horizontal relation. Since no notable difference was found between the scoping of full and surface subjects or among different types of object (relative to the subject or to postposed adverbials) the general relations *subj-obj*, *subj-post* and *obj-post* will be used here in place of the more detailed relations used in the table. Existential quantifiers are assumed to be "non-specific", but an allowance is always made (in the form of a higher scoping weight) for the possibility that they may be given a specific interpretation by pragmatics. This may make quite a large difference to the weights when existential quantifiers are inside strong scope traps.

By averaging over the different combinations of quantifiers, we can get a rough estimate of the effect of structural relations on scoping. This is summarized in Table 4.2 [16] where the scoping weights are inversely correlated with the strength of the "trap" created by the horizontal relation. For example, the second weight means that a quantifier inside a postposed adverbial is strongly trapped by a quantifier inside a preposed adverbial, since the average scoping weight assigned to the reading in which the former takes wide scope is only 0.05. Some of these values would be subtantially lower if the ability of existential quantifiers

to widen scope from any position was not taken into consideration. For example, quantifiers inside prepositional preposed adverbials may form an absolute trap for distributive quantifiers.

pre-pre	0.50
pre-post	0.05
pre-others	0.05
topic-subj	0.20
topic-others	0.10
subj-obj	0.22
subj-post	0.40
dir-indir	0.40
indir-dir	0.15
obj-post	0.60
post-post	0.70

Table 4.2 : The Effect of Horizontal Relations on Scoping

Is there a simple way to account for the above data? The effect of surface order is again clear, with the exception of the *obj-post* relation. The effect of shifting is also pronounced, with preposed adverbials forming an apparently absolute trap for distributive quantifiers, and topicalized objects forming quite strong traps. The dative effect is very marked (compare *dir-indir* and *indir-dir*) and the "shifting" of an object to the subject position which occurs during passivization has a comparable effect. Some notion of "distance" must probably also be taken into account, for example, to explain why postposed adverbials, and possibly also topicalized objects, are more likely to scope outside the object than the subject. The "distance" factor could be used to modify scoping weights. It may also apply to subject-object combinations, so that an object may be more likely to widen scope over the subject the closer it is to the subject. As an example, the tendency of *each* to scope outside *some* decreases as the distance between subject and object is increased in (31) to (33).

(31) Some boy kissed each girl	.6
(32) Some boy gave flowers to each girl	.5
(33) Some boy gave flowers to Mary for each of her rooms	.2

However, the effect of distance is not shown by quantifiers inside postposed prepositional adverbials which have a strong tendency to widen scope (34).

(34) Some boy gave flowers to Mary at each dance	.5

The effect of the lexical types of the quantifiers will now be considered. The effect of quantifiers which appear in the first position (in surface order) can be seen by comparing the columns in Table A2.2. For each structural relation, six quantifiers were tested in the first position: *a, some, every, most, few* and *no*. On the whole, the effect of the first quantifier is less than that of the second quantifier, and certainly it is less than that of the structural relation. There is a small, but quite consistent, difference between *a* and *some*: the former generally forms a slightly weaker scope trap for other quantifiers. This difference seems to be related to the ability of the former to be used in a generic (ie. non-specific) sense and dissapears as soon as some detail is added to the indefinite description.

The second notable point is the strong scope traps created by the quantifiers *few* and *no*, especially in the *subj-obj* relation. This is in constrast to the near inability of these quantifiers to widen scope over the subject from the object position. The large differences between these two quantifiers and *a* can be shown clearly with some examples:

(35) A boy kissed every girl	0.7
(36) Few boys kissed every girl	0.1
(37) No boy kissed every girl	0.05

(38) A boy kissed Mary at more than three dances 0.7
(39) Few boys kissed Mary at more than three dances 0.1
(40) No boys kissed Mary at more than three dances 0.05

The strong traps created by *few* and *no* are presumably related to the fact that they create a type of negated context. This property seems to be shared by *not many*, although to a lesser extent:

(41) No boys kissed Mary at each dance 0.3
(42) Not many boys kissed Mary at each dance 0.5

The lexical type of the second quantifier will now be considered. The strong tendency of *each* to widen scope has already been noted and is evident throughout table A2.2. The universal quantifier hierarchy is consistently present, with the ratios of *each:every:all* being about 3:2:1 and 7:6:5 when the quantifiers are in the object position and inside postposed adverbials, respectively. This difference is indicative of the ease with which almost all quantifiers can take wide scope when inside prepositional adverbials. The indefinite hierarchy is also quite consistently present. Again, the differences are less pronounced when the quantifiers are inside prepositional adverbials. The ratios of *some:three:several:many* are about 10:9:7:5 and 10:9:8:7 when the indefinites are inside the object position and inside adverbials, respectively.

The quantifier *most* has a moderate tendency to widen scope, generally between that of *every* and *all*, while *few* and *no* have very little ability to widen scope from any position. This contrasts with their tendency to form strong scope traps, at least when in the subject position. Many of the scoping weights associated with sentences containing *few* or *no* are placed in brackets or are omitted. This signifies that sentences of that pattern may be ungrammatical. Often the reason for this is that the sentence should be paraphrased in a more acceptable, and usually less ambiguous, way. Some examples of such sentences, and their paraphrased versions, are:

(43) Some boys kissed no girls
(44) Some boys didn't kiss any girls

(45) Some boys kissed few girls
(46) Some boys didn't kiss many girls

(47) Mary read few children no stories
(48) There were few children that Mary read no stories to

(49) *At no dance, John didn't kiss Mary
(50) At no dance did John not kiss Mary

The heuristic information needed for the scoping program, which is largely based on the data just described, is stored in three files: *weights, relations* and *ratios*. These files are shown at the end of Appendix II. The file *relations* is used to find the structural relation between a pair quantifiers, given their categories and their relative surface order. For each structural relation, the weight associated with giving the second quantifier wide scope is listed in the file *weights*. Since certain classes of operator, such as the universal quantifiers, indefinites and optional indefinite partitives, appear to form quite consistent scoping hierarchies, it is only necessary to store heuristic information for one member of each class. This is referred to as the "standard" for that class; for universal and existential quantifiers, the standards are *each* and *some*. Other members of these classes are related to the standards by ratios, which may vary with the type of structural relation (as we have seen earlier), and which are stored in the file *ratios*. The purpose of using standards and ratios is to reduce the amount of heuristic information needed and to make it easier to add new operators of a given type. The effect of the lexical type of the first quantifier is handled separately by the function *first-op-factor* which in some cases makes an adjustment to the scoping weight.

Before going to vertical scoping two further points will be noted. The scoping weights for some sentences are tagged with a "+". This signifies that the quantifiers in these sentences are often used in a different sense which does not directly correspond to any straigtforward scope ordering of the quantifiers. This typically happens when both quantifiers are either *no, few* or *most*. The following sentences have varying degrees of acceptability, but

the meaning of the non-scoped interpretations are clear.

(51) *John didn't kiss no girl
(52) No boy kissed no girl
(53) Few boys kissed few girls
(54) John kissed few girls at few dances
(55) Mary didn't show many pictures to many children

In these sentences, the quantifier pairs are used to emphasize the total quantity of predications made, at least in what appears to be the preferred reading. The first example is only acceptable in certain dialects, but is quite similar to the other examples. It is difficult to find any straightforward way of representing such sentences, such as by interpreting the quantifiers as collections. To some extent, *many* and *most* can be used in the same way, to emphasize total quantity.

The quantifier *any* has not been included in this discussion because it has some unique scoping properties and has been the subject of much controversy (eg. see Hornstein 1984). Two differing viewpoints are that an *any* inside a negated context should be represented as a wide-scoping universal quantifier or as an existential quantifier with narrow scope. If the former account is adopted, then the universal quantifier must scope outside the negation but inside the subject in (56) and (57), both of which are unambiguous.[17]

(56) Most boys didn't kiss any girls
(57) Most boys didn't Mary at any of the dances

This is an awkward rule to use and it cannot be used when *any* is inside a negative context created by *few* or *no* as in (58) or (59).

?(58) Few boys kissed any girls
(59) None of the boys kissed any of the girls

Example (59) seems difficult to handle by both accounts, since there are two readings, one of

which requires a universal quantifier with wide scope and the other an existential quantifier with narrow scope. In contrast,

(60) Not many boys kissed any girls

is not ambiguous (for some reason). In a positive context, *any* behaves like a standard universal quantifier, showing the usual scope ambiguities. Both (61) and (62) are ambiguous.

(61) Some boy will kiss any girl who solves the riddle
(62) Any girl who solves the riddle will be kissed by some boy.

4.6. Vertical Scoping of Quantifiers

Vertical scoping will be defined here as the scoping of an operator with a predicate or function which embeds the phrase [19] in which the operator occurs. The types of vertical relation which will be considered here and which are handled by the scoping program are listed in Table 4.3.

quant-cl	:	a full relative clause
quant-vp	:	a verb phrase serving as a noun complement
quant-pp	:	a prepositional phrase serving as a noun complement
subj-cl	:	a clause serving as subject
subj-vp	:	a verb phrase serving as subject
obj-cl	:	a clause serving as object
obj-vp	:	a verb phrase serving as object
a_1	:	a clause serving as an adverbial (ie. function)
a_2	:	a verb phrase serving as an adverbial
a_3	:	a prepositional phrase serving as an adverbial
func	:	a phrase embedded inside a function
pre-if	:	a preposed antecedent clause
post-if	:	a postposed antecedent clause
pre-then	:	an initial consequent clause
post-then	:	a subsequent consequent clause

Table 4.3 - Vertical Relations

The first three relations are somewhat difficult to define. They may be considered to be between the predicate (verb) of a clause or verb phrase and an operator inside the complement of one of the terms to which the predicate is applied. However, it is simplest to conceive of the relation as being between the head determiner of the term and the operator in the complement, since this is a more meaningful relation and provides simpler heuristics. The next four relations are between a predicate having clausal terms and operators inside those terms. The next three relations are more difficult to define. They are relations between a predicate and operators inside clausal adverbials which are applied to the clause containing the predicate. Since prepositional clauses serving as adverbials do not appear to form any scope trap, quantifiers inside them can be scoped directly (horizontally) with quantifiers inside the main clause.

The *func* relation is a relation between a function, such as an adverb or a modal, and an operator inside the phrase to which the function is applied. The last four relations are between a connective, such as *if-then*, and operators inside one of the clauses joined by the connective. It is likely that different types of connective form different types of scope trap, in which case some further relations will need to be defined, if ratios cannot be used. A brief description will now be given of the different types of vertical scoping.

First, the scoping of quantifiers inside noun complements will be exam[illegible]n Lehn argues that there is an "embedding hierarchy" which determines the ability of a distributive quantifier inside a noun complement to widen scope over the head quantifier. The hierarchy,

(63) possessive > PP > gerund > infinitive > finite clause,

is partly illustrated by the following examples:

(64) At the conference yesterday, I managed to talk to a guy who is representing each raw rubber producer in Brazil.

(65) At the conference yesterday, I managed to talk to a guy representing each raw rubber producer in Brazil.
(66) At the conference yesterday, I managed to talk to a representative from each raw rubber producer in Brazil.

He notes that no people obtained the distributive ("different/per") reading in (64), whereas 50% did in (65) and 100% in (66).

How reliable is this hierarchy? Full relative clauses do form strong scopal traps, but it is possible to find exceptions. One exception, repeated from chapter one, is

(67) Two tourists who visited each city bought a souvenir

which may have a reading in which there is a different tourist per city, depending on the context and on the intonation and stress pattern. Another example is

(68) Most of the circles which were inside each square were too small

which presumably refers, given pragmatics, to different circles inside each square. However, although sentences such as (67) and (68) do occur in practice, they are presumably quite rare and it is preferable to use a prepositional or verb phrase when the wide scoped reading is intended. [20] Saint-Dizier (1985) also mentions some cases in which a quantifier inside a full relative clause scopes over the head determiner.

Verb phrases serving as noun complements generally form traps of intermediate strength which often result in quite ambiguous sentences. Sentence (65) is one such example and some further examples are given in Part 3 of Appendix I. Quantifiers inside prepositional noun complements generally scope outside the head noun; however, to some extent this is a result of world knowledge. Noun phrases such as *a museum in each city* and *a tourist from each country* have unambiguous distributive interpretations since it is generally not possible to

be in more than one place at once or to come from more than one country. The preposition *of* is a bit more ambivalent, and the phrase *the winner of every race* is more ambiguous. Prepositions such as *beyond* or *after* are still more pragmatically neutral, and the phrase *some valley beyond every hill* is completely ambiguous, depending entirely on the context for its interpretation. In some cases, the distributive reading can be made very unlikely, as in

(69) A three-week tour of all of these cities would be expensive

which is unlikely to refer to a different tour per city. Table A2.3 (in Appendix II) gives the scoping weights for sentences containing different combinations of head quantifiers and quantifiers inside prepositional noun complements.

Distributive quantifiers acting as possessive NPs may always scope outside the head noun. An example is

(70) Everyone's car is foreign

It is difficult to determine the preferred scope orderings of quantifiers inside adjacent noun complements. [11] In

(71) Several tourists from each country on most of the buses bought a souvenir

the preferred scope ordering is *most* > *each* > *several* > *a*. In this case, the quantifier in the second noun complement (*most*) widens scope over the quantifier in the first complement (*each*) in the manner of quantifiers inside postposed adverbials. However, van Lehn (1978:37) found that more informants (100% vs. 80%) obtained the distributive reading in (72), in which *each* precedes *a*, than in (73).

(72) The carving of each design from a block of wood is a requirement of the course

(73) The carving of a block of wood into each of these ten designs is a requirement of the course

One further point should be made about noun complements. Noun complements are represented in the initial logical translations as connective clauses, with adjacent complements joined by the connective "&". Generally, quantifiers inside clauses joined by a connective must be scoped relative to the connective, such as *if-then* or *because*. However, a special case is made for noun complements. This is done, first, because the special connective & is not a sentential word containing a suffix, and second, because the scoping appears to be unecessary. As an example, there appears to be no reading in

(74) Most tourists with a large camera taking each trip got lost

in which *each* scopes inside *most* but distributes over *a* (meaning that the same tourists went on each trip, but with a different camera per trip). Therefore, quantifiers inside noun complements are scoped vertically directly with the head quantifier.

Clauses serving as subjects or objects (verb complements) generally form very strong traps, while verb phrases form moderate traps. Van Lehn gives some evidence that clauses and verb phrases in the subject position obey his embedding hierarchy. His data include the following sentences:

(75)	The release of each demonstrator required a short hearing	(100%)
(76)	Freeing each demonstrator required a short hearing	(100%)
(77)	To free each demonstrator would require a short hearing	(71%)
(78)	For the court to free each demonstrator would require a short hearing	(50%)
(79)	?That each demonstrator can be released would require a short hearing	

The percentages indicate the preference for the distributive reading. Example (79) was considered to be somewhat ungrammatical, but the *that*-clause does appear to form a very

strong trap.

Generally, *that*-clauses serving as verb complements form strong traps. However, one exception, repeated from chapter one, is

(80) A quick test confirmed that each drug was psychoactive

in which it is easy to get the reading in which there is a different test per drug. Fodor & Sag (1982) mention that attitude clauses form quite weak scope islands; however, they appear to be referring to the scoping of quantifiers inside such clauses relative to the opaque attitude operator. [11] In contrast, it is very difficult, if at all possible, for a distributive quantifier inside a *that*-attitude clause to scope outside the subject. For example, there is no reading of

(81) Some boy thinks that each girl will show up

in which there is a different boy per girl. On the other hand, it is not hard to get such readings when verb phrases serve as verb complements. An example is

(82) At least two boys want to kiss each girl

The scoping of quantifiers inside prepositional adverbials was discussed earlier in connection with horizontal scoping. In contrast to prepositional adverbials, verb phrases serving as adverbials form quite strong traps. Two examples, with progressive and infinitive VPs, are

(83) Wishing to help someone, many people sent a donation to a charity
(84) In order to meet many girls, most of the boys came early

Although adverbial phrases are interpreted as functions and embed the clauses they modify in

the logical translations, it is questionable whether there are readings in which quantifiers in the main clause scope inside an adverbial phrase. At least such readings are hard to get and are omitted by the scoping program at the present time. [23] Distributive quantifiers inside adverbial verb phrases must be scoped twice: first, relative to the main predicate (even though this does not "embed" the adverbial in the logical translation), and then relative to each quantifier in the main clause. The first scope ordering indicates whether the adverbial quantifier iterates over the main predication. For example, we are more likely to expect a different meal per museum in (85) than in (86), even though *several* presumably scopes outside *many*.

(85) While visiting many museums, several people stopped to eat
(86) After visiting many museums, several people stopped to eat

The scope ordering of quantifiers inside clauses joined by connectives is strongly determined by the surface order of the clauses. [24] For example, the distributive reading, in which the girl depends on the boy, can be obtained in (87) but not in (88).

(87) Each boy bought flowers and (then) he gave them to a girl
(88) A girl wanted flowers and each boy gave her some

This correlates with the constraint on pronoun binding across conjoined clauses. A similar effect of surface order was seen earlier with *if-then* clauses in the sentences

(89) If a student in the syntax class cheats on the exam, every professor will be fired
(90) Every professor will be fired if a student in the syntax class cheats on the exam

The universal quantifier can only distribute over subsequent but not over preceding clauses. The *if*-clause, however, appears to always form a scope island for distributive quantifiers.

4.7. Coordinators and Negation

In the next three sections, some preliminary heuristics will be given for the scoping of coordinators. In the examples which follow, the values following each sentence indicate the approximate preference for scoping the coordinator outside the negation. Coordinators in preposed adverbials probably always scope outside negation:

(91) In (both) Paris and Madrid, John didn't meet Mary	.99
(92) In (either) Paris or Madrid, John didn't meet Mary	.99

The same is true of coordinators inside topicalized NPs. Coordinators in the subject position tend to scope outside the negation. This may be unambiguous with *or*.

(93) (Both) Sue and Mary didn't (both) dance with John	.7
(94) (Either) Sue or Mary didn't dance with John	.99

This is also true true of coordinators serving as surface subjects. Note that (93) may mean that it is not true that both girls danced with John. This reading might be favoured by placing stress on *both*. It may also be forced by placing *both* after the negation (95).

(95) Sue and Mary didn't *both* dance with John	0.0

However, it is misleading in this case to associate the scoping weight with the ordering of *and* and the negation, since it is the inability of *both* to widen scope over the negation (see also section 4.4) which accounts for the low value. Perhaps *both* should be treated as a partitive over the collection formed by *and*.

Coordinators in the object position tend to scope inside the negation, with *or*, perhaps surprisingly, appearing to be trapped to the greatest extent.

(96) John didn't kiss Sue and Mary	.4

(97) John didn't kiss (either) Sue or Mary .05

The wide scope reading of *or* in (97) occurs when the speaker is unsure about the correct reference. For example, he might follow (97) with *but I'm not sure which girl it was*. In this case he is not using the sentence to assert that both girls weren't kissed by John, rather that one of them wasn't but he is not sure which one. Since this reading is uncommon a default value of 0.05 is used. This "speaker uncertainty" reading is always present when *or* is used and allows *or* to escape from scope islands (see section 4.9).

Coordinators inside postposed adverbials are quite similar, having perhaps a slightly greater tendency to take wide scope.

(98) John didn't meet Mary in Paris and Madrid .6
(99) John didn't meet Mary in (either) Paris or Madrid .05

The scoping of verb and verb phrase coordinators parallels that of noun phrase coordinators in the object position. The following examples illustrate this.

(100) John didn't hug and kiss Mary .4
(101) John didn't hug Mary and kiss Sue .3
(102) John didn't hug or kiss Mary .05
(103) John didn't hug Mary or kiss Sue .05

Note that, as before, placing *both* after the negation in (100) or (101) will prevent the coordinator from taking wide scope. Also, the wide scope readings of *or* in (102) and (103) stem from speaker uncertainty. However, *or* may be given a wide scope reading by placing *either* before the negation, as in (104), although it is preferable to add a second negation (105).

(104) John either didn't kiss Sue or Mary
(105) John either didn't kiss Sue or didn't kiss Mary

4.8. Coordinators and Quantifiers: Horizontal Scoping

Since universal and existential quantifiers are commutative with *and* and *or*, respectively, there is no need to consider the scoping of these commutative pairs. However, plural indefinites may introduce a universal partitive which must then be scoped relative to *or*. In general, coordinators and quantifiers are affected in the same way by different types of structural relation. Both quantifiers and coordinators in preposed adverbials usually scope outside of quantifiers and coordinators in the subject or object position. The same is true of topicalized quantifiers or coordinators. Coordinators and quantifiers in the subject position tend to scope outside those in the object position. Most notably, implicit partitives and the coordinator *or* have very little ability to widen scope over the subject from the object position. Some examples will illustrate this:

(106)	John and Bill kissed some girl	.4
(107)	John or Bill kissed some girls	.05P
(108)	Some boy kissed Sue and Mary	.1
(109)	Every boy kissed Sue or Mary	.3

Note that in (109) the wide scope reading of *or*, resulting from speaker uncertainty, is quite easy to get. The strong scope trap created for *both* by negated contexts can again be seen by comparing (110) and (112) with (111) and (113). [25]

(110)	No boys kissed Sue and Mary	.5
(111)	No boys kissed both Sue and Mary	.05
(112)	John kissed no girls in Paris and Madrid	.7
(113)	John kissed no girls in both Paris and Madrid	.3

The dative shift effect also affects the scoping of coordinators. However, it is difficult to demonstrate the effect of the shift with a conjoined shifted indirect object, since conjoined noun phrases, such as *John and Bill* or *wine and cheese*, are often given a collective interpretation, and therefore act as though commutative with existential quantifiers. It was

difficult to find sentences in which this collective interpretation did not interfere with the assignment of scoping weights.

Some examples of the scoping of quantifiers with coordinated verbs and verb phrases follow. The first four examples show that *some* forms a strong trap for a conjoined verb or verb phrase whereas *every* forms a somewhat weaker trap for a disjoint verb or verb phrase.

(114)	Some boy hugged and kissed Mary	.01
(115)	Some boy hugged Mary and kissed Sue	.1
(116)	Every boy hugged or kissed Mary	.3
(117)	Every boy hugged Mary or kissed Sue	.3

Some examples with quantifiers in the object position follow. [26]

(118)	John painted and fixed a motorcycle	.8
(119)	John painted and fixed some motorcycle	.8
(120)	John painted and fixed some motorcycles	.5
(121)	John painted or fixed each motorcycle	.8
(122)	John painted or fixed every motorcycle	.6
(123)	John painted or fixed some motorcycles	.5P

For some reason, *few* and *no* are not trapped by this construct: [27]

(124)	John painted or fixed few motorcycles	.9

It is difficult to rule out the influence of pragmatics in sentences of this type. If the two predications cannot be made of the same object as in (125) and perhaps in (126)

(125) John won and lost two matches
(126) Mary lost and gained twenty pounds

then the coordinator must take wide scope.

4.9. Coordinators and Quantifiers: Vertical Scoping

A general rule appears to be that a distributive quantifier is always trapped by a coordinator which embeds it. This useful rule, which is presently being used by the scoping program, rules out the three "pointless" readings of

(127) Every man or every woman arrived late

leaving only the reading which asserts that either every man or every woman arrived late. [28] Similarly, it rules out the reading of

(128) Every man or some woman arrived late

in which the woman functionally depends on the man. However, some provision must be made (for example, by later using the process of "scope expansion") to allow distributive quantifiers to anaphorically bind pronouns in such cases. For example, if an anaphoric pronoun is added to (128) to give

(129) Every man or some friend of *his* arrived late

then *every* must scope outside the *or* in order to bind the pronoun. This is analogous to the problem of accounting for the ability of distributive quantifiers to bind pronouns across conjoined clauses (or sentences). An example is

(130) Every boy loves some girl and he often visits her

in which *every* must scope outside *and* in order to bind *he* and *her*. [29] Existential quantifiers and *or* can widen scope over embedding coordinators, although there is generally a preference for readings in which embedding relations are preserved. In the program, a scoping weight of

0.7 is assigned to readings in which embedding coordinators trap existential quantifiers.

Coordinators inside noun complements also tend to scope inside the complement. It is more natural to use a coordinated noun phrase to express the reading in which the coordinator has wide scope. For example, the *or* is more likely to have wide scope in (131) than in (132).

(131) Every boy or every girl will get a ride home
(132) Every boy or girl will get a ride home

The lexical type of the head determiner can influence the strength of the scope trap for coordinators, as for quantifiers. The strong traps created by *few* and *no* are may be seen by comparing (133-4) with (135-6).

(133)	All of the people from Banff or from Jasper are coming	.5
(134)	Most of the people from Banff or from Jasper are coming	.4
(135)	Few people from Banff or from Jasper are coming	.05
(136)	No one from Banff or from Jasper is coming	.05

The extent to which a coordinator is embedded inside a noun complement affects its ability to widen scope. If the coordinator in (134) is moved inside the prepositional phrase, giving

(137)	Most of the people from Banff or Jasper are coming	.3

the strength of the trap is increased.

Like universal quantifiers, *and* is trapped by scope islands. For example, in

(138) Someone heard the news that Sue and Mary were arriving

it is not possible to get a reading in which *and* scopes outside *someone*, that is, in which there may be a different person for Sue and Mary. However, *or* can widen scope outside of scope

islands, although this is not usually a preferred reading.[10] The reason *or* can always take wide scope is due to uncertainty on the part of the speaker or of someone mentioned in the discourse. Three examples will show this.

(139) Everyone who went swimming or sailing had a good time
(140) Everyone heard the news that Sue or Mary was arriving
(141) Each person heard the news that his aunt or his uncle was arriving

Example (139) may mean that either everyone who went swimming or everyone who went sailing had a good time. In (140) there is a reading, perhaps not too obvious at first, which means that the speaker is not sure whether it was Sue or Mary that everyone heard was arriving. In (141) there is clearly a reading, perhaps the preferred one, in which each person has either heard that his aunt or that his uncle is arriving. This shows that *or* has scoping properties similar to existential quantifiers: it may escape from scope islands, in contrast to *and*, and yet still be functionally dependent.

Footnotes

1. The term *lexical type* is being used here to refer to the lexical class of the operator being scoped, such as *each, most, and, or* and *not*, or in some cases to a more general class such as *numeral, existential quantifier* or *universal quantifier*. Nouns, verbs and prepositions can also be placed into lexical categories which can be used to disambiguate quantifier scope. For example, *Each person bought a book* is more likely to have a distributive reading than *Each person read a book* because the predicate *read* is more easily repeated with the same object. It is still possible that each person bought the same book in turn, but this is not possible with irreversible predicates such as *sacrifice*. *Each village sacrificed an ox* can only have the distributive reading. Therefore, some of the "world knowledge" mentioned above as the fourth category could be expressed in the form of quite simple rules related to lexical categories.

2. We will use the category *adverbial* to refer to an unscoped operator such as a quantifier inside a clausal adverbial and the category *adverbial-op* to refer to the adverbial itself, for example for an adverbial operator such as *often* or *quickly*. A *full-subject* is a subject which is both a surface and a deep subject.

3. Note that the difference between vertical and horizontal relations disappears in Montague-style intensional logic in which all operators are at different levels of embedding.

4. Note that a coordinator, and also a connective, can embed an operator which precedes it in surface order. In this case, the embbedding relation appears to predominate.

5. It does not matter here whether preposed or postposed adverbials are considered to be default; the effect of surface position is what is important. The same applies to the shifting of objects relative to one another.

6. These are the two most recent studies of scoping preferences that I am aware of, in addition to the study of Gil (1982). Some references to earlier studies of scoping are given by Ioup and an extensive discussion of various linguistic studies of scoping is given in Carden (1976) (the latter is based on a thesis completed in 1970). Carden provides evidence for the presence of dialects of scoping preferences, for example for the two readings of *All the boys didn't leave*. However, there is also evidence that by embedding such sentences in different contexts, any given person can be shown to perceive both readings. Newmeyer (1983:57) suggests that dialects reflect "speakers' differing contextualizations of possible readings for sentences that are ambiguous in their grammars." These references have been brought to my attention by Matthew Dryer.

7. The heuristics described in this chapter are based solely on data obtained from English sentences and might have to be modified to place more emphasis on "grammatical function" (ie. structural category) and less on surface order if data from other languages necessitates this. However, this modification would be quite simple to make since the heuristics are already organized in terms of structural categories and relations.

8. However, van Lehn made no attempt to filter out pragmatic information in his sentences which might override the preferences given by the hierarchies.

9. The decision to base the scoping weights on the tendency of the second quantifier to widen scope was adopted after some trial and error. Generally, the preferred scope ordering

correlates closely with the surface order of operators and it requires a certain amount of "effort" to reverse this order. This amount of effort corresponds to the strength of the "trap" created by the first operator (and by the structural relation). The scoping weight therefore gives a measure of the strength of the trap for a given second operator, with a strong trap keeping the weight down close to zero.

10. In a few cases, some quite large differences were noted, usually because one reading had been dismissed too quickly the first time. Some of the sentences were tested independently by a separate observer with comparable, and in some cases very close results. However, a few substantial differences were also encountered, apparently for the same reason as above. It is possible that by a careful specification of what is meant by the different scoped readings, and with some practice, a set of data with a low standard deviation could be obtained, both for one person and among different observers.

11. One other point which should be noted is that it is very difficult, if at all possible, to find completely pragmatically neutral sentences and therefore the only way to rule out the influence of pragmatics is to test a large number of sentences. One possible alternative is to substitute nonsense words in place of nouns and verbs, even though inevitably some attempt is made to guess at what the nonsense word might signify. Nonsense sentences may have quite marked scoping preferences, as can be seen in the sentences *Few gorms mirled every tok*, *Few gorms were mirled by every tok*, and *Mary mirled more than three toks at each warg*. In some cases, such sentences were tried and gave quite similar scoping preferences as ordinary sentences having the same pattern. However, such sentences quickly became tedious and therefore simple sentences were used in the survey.

12. It is not clear why this should be so. Perhaps it results from the usefulness of making non-universal statements, that is predications which do hold for all the members of some set. Another reason could be that universal quantifiers inside a negated context should be replaced by *any* when the wide scope reading is intended. For example, *John didn't kiss every girl* should be replaced by *John didn't kiss any girl* and *No boy kissed every girl* by *No boy kissed any girl*. However, this does not explain the ease with which universal subjects scope inside the negation.

13. It is possible for the negation operator to scope outside non-universal subjects in certain cases. An example is when the sentence follows a statement with the same or a similar pattern in the non-negated form. For example, *John thinks many girls worship him* may be followed by *Well, many girls don't (worship him)* meaning that it is not true that many girls worship John.

14. The two relations *dir-indir* and *indir-dir* are used for any pair of objects, with the latter relation being used if the first object has been shifted in front of the second.

15. In the logical translations, postposed adverbials are applied to the sentence in the reverse order of their appearance. This correlates with our intuitions about the correct order of embedding; however, the suffixes must then be ignored otherwise the innermost postposed adverbials will appear to be shifted.

16. The average values for five of the last six relations are taken from the lower right-hand corners of the different sections of Table A2.2. The other values are estimates made from data not shown in the tables. The values must only be treated as rough approximations.

17. When *any* is present it may have rule out certain orderings of other quantifiers. For example, it does not seem possible for the negation to scope outside *three* in *Three boys didn't*

kiss any girls even though this is (arguably) possible in *Three boys didn't kiss many girls*.

18. *Any* appears to have an additional interpretation which may be described as "just any". For example, *John won't date just any girl* does not mean that for all girls, John won't date them, which is the usual interpretation. This may require that a lexical ambiguity be made.

19. The term "phrase" means a clause, verb phrase or prepositional phrase, in terms of syntax; a more accurate definition is given in terms of initial logical form in section 5.1.

20. These examples may not be very convincing and at least one person has informed me that he cannot get the distributive reading of the latter sentence, even when pragmatics is taken into consideration. The question is whether it is ever possible to get readings of this type and, if so, what heuristic value to assign (clearly it should be very low).

21. In the scoping program, the horizontal relation *restr-restr* is used to scope quantifiers in adjacent noun complements. At the moment, a weight of 0.5 is used for all combinations of quantifiers. The relation *conn-conn* is used for all other types of connective clauses (ie. all those not serving as the restriction clause of a noun phrase).

22. For example, they claim that the preferred reading of *This producer believes that each actor in our company is too fat to appear in public* is the one in which *each* scopes outside *believes*, which they interpret as meaning that the producer holds separate beliefs about the individual actors. By comparison, they suggest that the opaque reading is preferred if *each* is replaced by *every*. However, this analysis is very dependent on the interpretation given by the authors to the scoping of *each* with opaque operators.

23. At the present time, quantifiers automatically widen scope outside all functions, including tense, modals and adverbs, with the exception of quantificational (temporal) adverbs which are treated as unscoped operators. However, this may need to be modified to allow quantifiers to scope relative to functions in general. It may be best to treat all such scoping as "vertical".

24. Clauses related by causal operators such as *if-then, therefore* or *because* may be parsed either as main and subordinate clauses or as clauses joined by a connective. The latter approach is taken here.

25. The scoping of plural definites such as *these* is similar to that of plural indefinites. It is difficult to get the reading in which the optional partitive scopes outside the coordinator in either *John or Bill kissed some girls* or *John or Bill kissed these girls*.

26. It is assumed here that the following sentences are parsed as having verb, rather than verb phrase, coordination.

27. Note that (124) would be better paraphrased as *John didn't paint or fix many motorcycles*.

28. The three extra readings are obtained by giving either of or both of the universal quantifiers wide scope over the coordinator.

29. In this case, widening the scope of *every* relative to *and* does not alter the truth condition.

30. Rooth & Partee (1982) note the wide-scoping property of *or*.

Chapter 5

Algorithm and Program

The quantifier scoping program described in this chapter is designed to be used as an extension to a general purpose natural language understanding system currently being developed (Schubert & Pelletier 1982). The parser uses a modified version of Generalized Phrase Structure Grammar with semantic rules that generate initial logical translations in a first order modal logic augmented with certain operators. The logical translations are in general ambiguous, leaving unscoped elements marked (indicated here by angled brackets) for subsequent disambiguation. The scoping algorithm operates on this preliminary logical form and generates scoped logical forms having the logical syntax described in chapter one. As an example, the initial logical form for (1) is shown in (2) and the two scoped readings in (3) and (4). [1]

(1) Most people on every committee know John

(2) [<most λx[[x person] & [x on <every committee>]]> (PRES {know John})]

(3) (most x [[x person] & (every y [y committee][x on y])] [x know John])
(4) (every y [y committee] (most x [[x person] & [x on y]] [x know John]))

A partial syntax for the initial logical translations will now be given.

5.1. Initial Logical Translations

A partial logical syntax for the initial logical translations is shown in Table 5.1. The notation is augmented with two types of operator, designated τ (for "term-forming") and α (for "adverbial-forming"): the operators τ_1 and τ_2 map infix and prefix formulas, respectively, into terms and α_1 and α_2 map infix and prefix formulas into functions (serving grammatically as adverbials). The operator α_3 is used to convert a special class of prefix expressions, those with prepositional predicates, into adverbial functions. For example, the

phrase "in each city", when serving as an adverbial, would be represented as "(a, (in <each city>))". [2] Some abbreviations are used in the table: *expr* for *expression*, *formula* for *sentential formula*, *pred* for *predicative*, *func* for *functional*, *coord* for *coordinator*, *conn* for *connective*, *nom-pred* for *nominal predicate*, *det* for *determiner* and *var* for *variable*. The symbol "++" means "two or more". The table is not complete, but shows the types of expression acccepted by the scoping program at the present time. [3]

The table shows that four types of expression are recognized. A distinction is made between sentential formulas, which are represented as infix expressions, and predicative and functional expressions. The use of infix representation allows a closer mapping to be obtained between surface structure and logical form. There is also some linguistic motivation for making this distinction, such as evidence that some adverbs should be applied to the verb phrase and others to complete sentences. The usefulness of representing sentences as infix expressions with verb phrases as predicates is also shown by, for example, work on VP deletion (Sag 1976). Making the above distinction allows for a more general treatment of initial logical translations. It would always be possible, for example with the use of a preprocessor, to convert all three types of expression into prefix form and use a prefix representation for sentential formulas. [4]

In table 5.1, a special case is made for negation and quantificational adverbs which are initially applied to prefix expressions (ie. to the verb phrase). These operators are handled as moveable (unscoped) operators and therefore placed inside angled brackets. Other functional expressions, such as tense, aspect, modals and other adverbs are not treated as scoped operators at the present; however, the program could be easily extended to allow this, so that, for example, modals and tense operators could widen scope over the subject. One alternative means of accomplishing this would be to apply such functions to the whole clause

expr	—>	formula | pred-expr | func-expr | term
formula	—>	<coord formula+ +> | (func-expr formula) | (det var formula formula) | [formula conn formula+] | [term pred-phrase] | [term prep-pred term+] | [var nom-pred]
coord	—>	*or, and*
conn	—>	&, v, *if-then, after, because, ...*
pred-expr	—>	pred-phrase | pred-const
pred-phrase	—>	<not pred-phrase> | <quant-adverb pred-phrase> | <coord pred-phrase+ +> | (func-expr pred-phrase) | lambda-expr | {verb-pred term*} | {prep-pred term+}
pred-const	—>	verb-pred | prep-pred | nom-pred
lambda-expr	—>	(λ var formula)
verb-pred	—>	*love, know, ...*
prep-pred	—>	*in, between, ...*
nom-pred	—>	*man, house, ...*
func-expr	—>	<coord func-expr+ +> | func-phrase | func-const
func-phrase	—>	(a_1 formula) | (a_2 pred-phrase) | (a_3 {prep-pred term+})
func-const	—>	tense | aspect | modal | adverb
tense	—>	PRES, PAST, FUT, GOING-TO
aspect	—>	PROG, PERF, PASV, INF
modal	—>	*may, can, should, ...*
adverb	—>	quant-adverb | *quickly, probably, ...*
quant-adverb	—>	*sometimes, always, often, ..*
term	—>	(τ_1 formula) | (τ_2 pred-phrase) | (μ lambda-expr) <coord term+ +> | quantifier | identifier
quantifier	—>	<det lambda-expr> | <det nom-pred>
det	—>	*some, many, every, ...*
identifier	—>	var | const
var	—>	*x1, x2, ...*
const	—>	*John*, (μ *water*), ...

Table 5.1 : Syntax of Initial Logical Translations

in the initial translations, in which case the functions could be treated as scoped elements. To convert the logical forms shown in table 5.1 into LISP S-expressions, different types of expression are prefixed with one of the following symbols

i - infix formula
p - prefix phrase
f - functional expresssion
q - quantifier
c - coordinated expression
l - lambda expression
n - negated expression

giving the following equivalences:

[term pred-expr]	<=>	(i term pred-expr)
{verb term*}	<=>	(p verb term*)
(func-expr formula)	<=>	(f func-expr formula)
<coord expr++>	<=>	(c coord expr++)
<det nom-pred>	<=>	(q det nom-pred).

Before describing the scoping program itself, some of the general problems which must be taken into consideration in the design of a scoping algorithm will be discussed. The first problem that needs to be considered is how the scoping weights associated with individual pairs of operators should be represented and combined in order to arrive at the preferred reading(s) of a sentence. This leads to two questions: (a) which operators should be compared in a given reading and (b) how should the weights associated with each comparison be combined. These questions will be discussed, in turn, in the following two sections. Some further problems will be discussed in section 5.4.

5.2. Comparing Operators

Depending of the method used to scope quantifiers, different pairs of operators may be compared during the scoping of a given reading. This is certainly a drawback if one wants to find some theoretical basis for the design of a quantifier scoping algorithm; and the problem is compounded by the difficulty in deciding, on intuitive or on other grounds, which of the options is more correct. We will illustrate the nature of the problem with three

examples.

The first example involves the simple case of a sentence having three clausemates, such as a subject and two objects. An example is

(5) Most of the boys gave a flower to each girl

which can be represented as "[a b c]" using a simple notation in which infix expressions are designated by square brackets, functions by round brackets, unscoped quantifiers by angled brackets and variables by the letters *x*-*z*. The standard approach to scoping (5) is to first select the quantifier which is to have innermost scope and then apply this quantifier to the clause, substituting a variable in its place. An example of one possible scoping sequence of (5) is shown in (6).

(6) [<a> <b> <c>] —> (b [<a> y <c>]) —>
(a (b [x y <c>])) —> (c (a (b [x y z]))

However, there is more than one way of comparing pairs of quantifiers. In the *complete* method, comparisons are made before each quantifier is raised, between that quantifier and every other quantifier, so that the quantifier having the strongest tendency to take narrow scope can be raised first. This will result in n(n-1)/2 comparisons being made for *n* quantifiers, that is, all pairs of quantifiers will be compared. An alternative method would be to raise the quantifiers at random and make comparisons while the quantifiers are being raised relative to one another. For example, in the partially scoped expression "(a (b [x y <c>]))", as *c* is raised it will first be compared to *b* and then to *a*, but only if *c* scopes outside *b*. This will result in no comparison being made between *a* and *c* in the *abc* reading. The *a*/*c* comparison will also not be made in the *abc* reading if an algorithm is used which only makes a minimum number of comparisons in selecting the operator which is to be raised first. Using

this method, if the first two comparisons made are *a/b* and *b/c* and if it is found that *b* should scope inside *a* but outside *c*, then there is no need to make the *a/c* comparison at this point. It is very difficult to decide on intuitive grounds which of these methods is more realistic. [5] The *complete* method is used in this study.

The second example involves the scoping of a noun phrase with a quantifier in its complement. An example is sentence (1) in the previous section which can be represented as "[a-b c]" where the hyphen signifies the complement relation. If *b* scopes inside *a* then clearly no comparison should be made between *b* and *c*. However, if *b* scopes outside *a* then it may or may not be compared with *c* depending on which method is used. If a top-down, or *head-first*, approach is used for the scoping of clausemates, then the *a/c* comparison will be made before the *a/b* comparison, and *c* will only be compared with *b* if it scopes outside of *a*. Again, it is very difficult to make a judgement about which method is preferable. In this study a bottom-up approach is used and both *a* and *b* will be compared with *c* provided that the former scopes outside of *a*. [6]

The third example involves the scoping of two quantifiers both with each other and relative to an embedding operator. In

(7) John thinks that each boy loves some girl

the quantifiers may be first compared either with each other (*horizontal-first*) or with the embedding operator *think* (*vertical-first*). This will make a difference if one but not both of the quantifiers ends up scoping outside the embedding operator, since in this case the two quantifiers will only be compared with each other if the horizontal comparison is made first. It is very difficult to decide which method corresponds more closely to how people scope. The method used in this study is horizontal-first (although it could be modified to be vertical-first). [7]

5.3. Combining Scoping Weights

Given a set of comparisons between pairs of operators, how should the scoping weights obtained be combined to give the overall scoping weight for a reading? The main criterion is that it should be possible to select the preferred reading of a sentence, but it would also be useful to be able to arrange all the readings of the sentence in an approximate order of preference. However, once again, there are many possible ways of combining scoping weights and it seems that some fairly arbitrary choice of method must be made.

A probabilistic treatment does not appear to be feasible. One reason for this is that different readings may involve the comparison of different numbers of operators. A simple example of this is a sentence of the form "[a-b c]" which has five readings: two which involve two comparisons *(abc,cab)*, two which involve three comparisons *(cba,bca)* and one which may involve either two or three comparisons, depending on whether or not the *head-first* approach is used *(bca)*.

The method used in this study is simply to find the average scoping weight. This method has the advantage of simplicity and appears to provide quite good results. The major problem with this method is it tends to smooth out the effect of low pairwise scoping weights. For example, a reading with the two weights 0.01 and 0.99 will have the same average weight as one with two weights of 0.5. This is clearly undesirable, since it should be almost impossible to obtain a reading which contains an individual weight of 0.01. However, this problem can be minimized by using a cutoff point to eliminate readings which contain an unacceptably poor scoping weight (ie. ordering of a pair of operators). This is done in the scoping program by using a parameter *$min-weight* which is set to the minimum acceptable value for any individual scoping weight (eg. 0.05).

Can we guarantee that the best reading is always found? In the example "[a-b c]", suppose the individual weights are *ab*: 0.6, *ac*: 0.6 and *bc*: 0.9. Then the average weights for

the readings *abc* and *bac* will be 0.6 and 0.63, respectively. However, the first reading should probably be favoured since there is a preference for scoping *b* inside *a* (the *ab* ordering has a weight of 0.6). The reason it is not is that the high value associated with the *bc* comparison is not available to the first reading. [8]

The simplest way of dealing with this problem would be to tag readings which involve one or more non-preferred orderings. [9] This would solve the problem in the above example, but would not work in general since optimal overall readings sometimes involve non-preferred pairwise orderings. There appears to be a way of guaranteeing that the best reading is found which requires that a distinction be made between horizontal and vertical scoping. There is no problem with vertical scoping, since the best overall reading is always composed of the preferred pairwise orderings (since the operators being scoped only interact with one embedding operator at a time.) For horizontal scoping, it would be necessary that best combination of orderings of the operators at a given clausal level (prefix or infix) be tagged when the scoping at that level is completed. This could be done by adding a simple procedure which finds the reading which has the best overall ordering of operators at that clausal level, and then either tags the other readings as sub-optimal or scales all the averaged scoping weights at that level so that the best one is given a perfect weight. In the latter case, the best reading(s) would have an perfect weight at all times. In either case, this should guarantee that the "best" reading, according to the averaging criterion, is obtained.

Finally, there is the question of how the scoping weights should be represented. In chapter five, and in the current version of the program, weights are placed on a 0 to 1 scale, with the latter being the optimal value and with preferred orderings having weights between 0.5 and 1. An alternative method would be to scale the values so that the preferred ordering is always given an optimal value. This could be used in conjunction with the second method just described for guaranteeing the best reading of a sentence (according to the averaging method). Another possibility would be to reverse the scale, so that the optimal value is 0 and higher

values are interpreted as penalties. These three different types of representation can all be obtained simply by changing the values in the input file *pref-values*. In each case, the program will determine the average value.

5.4. Quantifier Scoping Algorithms: Background

Until recently, the problem of quantifier scoping has been relatively neglected in the design of natural language understanding programs. Two of the earliest attempts to deal with this problem were the LUNAR program (Woods, 1978) and the work of Dahl (1979) and Colmerauer (1979). Woods uses a modified version of FOPL to represent quantification. The general form of a quantified expression is

(8) (FOR <quant> X / <class> : (p X) ; (q X))

where <quant> stands for a determiner such as *some* or *every*, X is the variable of quantification, <class> the set over which the quantification ranges, (p X) a restriction on this range and (q X) the expression being quantified. Scoping is possible among quantifiers, between quantifiers and negation, and to a limited extent between quantifiers and opaque operators. [10]

Working in the Prolog programming language, Dahl, Colmerauer and others use an analogous modification of FOPL to represent quantification. Sentences are represented as trees having quantifiers, conjunctions and negations as nodes and variables and constants, and in some cases predicates, as leaves. Quantifier nodes have three branches: the variable, the restriction and the expression being quantified (this is sometimes referred to as "three-branched quantification"). A summary of the work on quantifier scoping in Prolog is given in Saint-Dizier (1985) and will only be briefly described here. In the early work, scope ordering is determined by a small set of general rules which are closely tied to syntax and which are applied during the parsing. An example of a rule from Dahl (1979) is that when a

verb has two quantifiers serving as complements, the order of scoping is the reverse of surface order. Similar heuristics are used by McCord (1982) and Warren & Pereira (1983).

Saint-Dizier (1985) describes a different approach to scoping in Prolog based on the use of rewriting rules. The initial logical form he uses is a scoped reading with the determiners scoped in the order in which they appear in the sentence. Each determiner is then converted to a "range indicator", which represents a class of determiner. The quantificational determiners are divided into four classes, represented by the range indicators *D*, *E*, *Q* and *U* (see below). As part of this process, suffixes are attached to each determiner in the same way as is done in this study. A set of rewrite rules, each of which interchanges a pair of range indicators, is then applied to the initial logical form to generate the set of valid scoped readings. The rewrite rules have the general form

(9) Ri : Xn Ym —> Ym Xn (or Xm Yn)

where X, Y belong to {E,D,Q,U} and *n,m* are integers with $n < m$. Eight rules are given for the interchange of two quantifiers and three for the interchange of a quantifier with negation. A method is briefly described for the scoping of quantifiers with the conjunction operator and some preliminary work on the incorporation of pragmatic information is shown.

Another recent scoping algorithm, and one which is the most similar to the one used in this study, is described in a preliminary form by Hobbs (1983) and is presented in detail by Hobbs & Shieber (1987). The algorithm generates all valid scoped readings of a sentence, starting with an initial logical representation which is very similar to the one described by Schubert & Pelletier (1982) and which is used in this study. The main differences are that all phrases are written in a uniform prefix notation and the only unscoped elements are quantifiers. The quantifiers are selected one at a time for scoping. The algorithm works partly bottom-up, scoping nested clauses first. However, noun phrases are scoped top-down, the

whole noun phrase first being scoped relative to its clausemates and then any quantifiers inside its complement being scoped recursively.

The algorithm used in this study differs from the above two in being completely bottom-up and left-to-right, in parallel with the parser, and in being able to both compute scoping weights and generate the set of scoped readings in one pass. In addition, the scoping program handles a wider range of initial logical representations, allowing distinctions to be made between, for example, verb phrases and clauses serving as complements. The algorithm will be described in the next section. First, however, the different algorithms described here will be compared in terms of two general problematic issues in quantifier scoping: (a) the use of heuristic information and (b) the scoping of coordinated expressions.

Since the determination of scoping weights requires quite extensive knowledge of the structural relations between pairs of operators, these weights must be at least partially determined *before* the unscoped operators are moved since most of this information is lost after movement. This consideration was a major factor in the design of the algorithm described here, in which the scoping algorithm parallels the parser. If the rewriting method is used, it will presumably be necessary to store some of the information about structural relations between pairs of operators before the scoping starts. The algorithm of Hobbs & Shieber is potentially capable of making use of structural relations; however, as the authors mention the highly recursive nature of their algorithm appears to make the inclusion of such heuristic information difficult.

Neither Saint-Dizier nor Hobbs & Shieber give much information about how they would incorporate heuristic information into their programs. The latter do mention some places where such information could be introduced, but note that the recursive nature of their program makes the use of such information quite difficult and that it cannot be guaranteed that the best reading will be found. Saint-Dizier provides little information about how scoping

heuristics would be used in his program, although it would be natural to associate scoping weights of the kind described here with the rewriting rules. However, it is not clear how useful it would be to associate heuristics directly with the four classes of "range indicator". Class *D* consists of *each* and *every*, and yet these have quite different scoping properties. Class *U* contains *all* and *no* and these are also very different. Class *E*, which is taken to represent the "first-order logic existential quantifier", includes the determiners *most of the* and *few* [11] in addition to some of the standard indefinites.

The scoping of coordinated expressions poses several problems. One problem is illustrated by the sentence

(10) John met a blonde or a brunette

which contains a coordinated verb complement. It is not possible to scope either of the indefinites before the coordinator, otherwise the subsequent application of the coordinator will result in vacuous quantification. For example, if the existential quantifier corresponding to *a blonde* is scoped first, followed by *or*, the following partially scoped logical form would result:

(11) [(Ex: blonde [John {met x}]) or (Ex: blonde [John {met <a brunette>}])]

The second clause contains vacuous quantification. [12] In this case, the problem could be solved by stating that coordinators must be scoped first (rather than the operator which is to take innermost scope). However, there are two problems with this. First, the *or* may scope outside the outermost clause in

(12) John wants to marry a blonde or a brunette

and this means that the inner phrase (the infinitive phrase serving as a verb complement)

cannot be scoped until the whole sentence is scoped, which violates the bottom-up principle. Second, if there are two nested coordinators as in

(13) John met (Sue or_1 (Ellen or_2 Mary))

the same problem ("vacuous coordination") occurs if the embedded coordinator or_2 is scoped before or_1, giving (14), and then or_1 is given wide scope (15).

(14) [[John {met <or_1 Sue Ellen>}] or_2 [John {met <or_1 Sue Mary>}]]

(15) [[[John {met Sue}] or_2 [John {met Sue}]] or_1
[[John {met Ellen}] or_2 [John {met Mary}]]]

These two problems could be solved by scoping in several passes, in each pass scoping only operators which are not embedded inside a coordinator. However, this considerably complicates the scoping algorithm and also violates the principle of applying the innermost operator first.

A second problem is the creation of multiple copies of the same operator which can occur when coordinators are present. This problem is unavoidable when it results from the parsing of sentences such as (16), which will be parsed into the initial logical translation shown in (17), and (18) which may be translated into (19).

(16) John did or didn't meet $some_1$ girl
(17) [John <or (PAST {meet <$some_1$ girl>})
(PAST <not {meet <$some_1$ girl>}>)>]

(18) John wants and hopes to meet $some_1$ girl
(19) [John <and (PRES {want (r_2 (INF {meet <$some_1$ girl>}))})
(PRES {hope (r_2 (INF {meet <$some_1$ girl>}))})>]

Three constraints on a duplicated operator such as $some_1$ are (a) it must scope consistently with respect to all other operators, (b) it must only be compared once to each operator (in

terms of computing the scoping weights) and (c) if it scopes outside an coordinator which initially embeds it, only one copy of the operator can be carried up. This poses a problem for bottom-up approaches to scoping since some global knowledge is evidently needed to ensure the consistency of the scoping of *some*$_1$ in the two separate expressions in which it occurs in both (17) and (19). It therefore seems necessary to use some overhead to keep track of the scope relations of operators which are present in multiple copies, and to store this information separately for each reading. [13]

In the scoping program shown in Appendix IV the problem of duplication is handled by labelling readings, if multiple copies are generated by the parser, [14] and by storing on the property list of each duplicated operator [15] a list of the operators having been scoped inside and outside the operator. In addition, the notion of "equivalent predicate" needs to be mentioned. In (19) the predicates *want* and *hope* are "equivalent" in the sense that they occur at the same level of embedding by the coordinator *and*. Therefore, the two copies of *some*$_1$ must scope consistently with the two equivalent predicates. [16]

Duplication of operators can also result from the scoping process. For example, this would occur when one of the coordinators in

(20) John and Bill love Betty or Sue

is scoped, creating a duplicate copy of the other coordinator. However, at present the scoping program avoids this problem, as well as the first problem desribed above, by using a "branch-trimming" function which removes incorrectly embedded operators from the different branches of an coordinator at the time of embedding the coordinator. The function is simple to use, but it does involve some extra overhead and temporarily results in the generation of redundant readings which must later be removed. [17]

Neither of these problems is addressed by Saint-Dizier or Hobbs & Shieber. The former does give an example of the scoping of quantifiers with the conjunction operator, but it is difficult to tell how the problems described here would be handled. The algorithm described by Hobbs & Shieber does not deal with coordination at present and it seems that the bottom-up and highly recursive approach used would make the handling of the above problems quite difficult and would at least require a considerable amount of overhead.

5.5 The Scoping Program

This section contains a brief outline of the scoping program. More details will be found in the program listing in Appendix IV. Procedures (or functions) referred to will be written in italics. The program reads in a list of sentential formulas and for each formula prints a list of scoped readings in approximate order of preference. Each formula is passed to *scope-phrase* which calls the appropriate scoping procedure, such as *scope-infix* or *scope-func*, depending on the type of phrase. Terms are scoped by *scope-term* and by *scope-quant* when appropriate. One procedure, *scope-coord*, is used to scope all types of coordinated expression, whether phrase, term or constant.

Each procedure returns a list of partially or fully scoped readings. A "reading" is a list containing an average scoping weight, the logical form of the expression being scoped, a list of raised operators and a label for the reading if the reading contains multiple copies of one or more unscoped operators. Each procedure for scoping phrases has an input parameter *construct* which describes the type of embedding construct (ie. vertical relation), and also lists the embedding operator where appropriate. The construct type provides heuristic information necessary for vertical scoping which is performed by procedure *expand*. Examples of phrases which require vertical scoping are phrases serving as terms or as functions, phrases joined by connectives such as *if-then* and phrases inside noun complements. The outermost phrase is given the construct *top* which forces all remaining unscoped operators to be embedded.

Horizontal scoping is performed by procedure *combine* which generates and evaluates all permutations of the unscoped operators present in corresponding pairs of readings. Some additional overhead is required when coordinated expressions are present.

The structural category of each unscoped operator is determined by *get-categories* and updated where necessary by *update-category*. The category of each operator is stored on its property list under the attribute *category*. The scoping weight for a pair of operators is calculated by *scoping-weight* which is called when readings are combined or expanded (ie. scoped horizontally or vertically). This procedure makes use of heuristic information stored in the three files *pref-values*, *srelations* and *ratios*. The way in which these files are used is described in chapter five.

Reflexives in the subject or object positions are handled when two readings are combined by substituting the logical form of the scoped operator (or the variable for quantifiers) for the unscoped second occurrence of the operator. Some overhead is required for this. If multiple copies of the same operator are present as a result of the parser, these operators are given special treatment to ensure that they are uniformly scoped. At present, this has only been partially implemented. [11]

Redundant readings are found using the functions *redundant?* and *commutative?* and are removed as soon as they are detected. A reading is redundant if two commutative operators are embedded consecutively and the suffix of the outer operator is greater than that of the inner one. For convenience a different criterion is used when one of the operators is a coordinator: in this case, the quantifier should scope inside the coordinator. Readings will also be removed if they contain an ordering of a pair of operators which has a scoping weight less than the parameter *$min-weight*. When such an ordering is detected, either during horizontal or vertical scoping, the reading containing it is removed. No scoping weights are attached to the relative ordering of commutative operators.

At the present time, the scoping program has not been completed. The function for retrieving the list of "equivalent predicates" has not yet been written and therefore vertical scoping is not yet possible for operators which have been duplicated by the parser. Some other possible extensions are described in the conclusion.

5.6 Program Output

Some sentences used as test data are shown in table 5.2 and the output in Appendix III. The output for each sentence consists of first, the initial logical translation of the sentence which is used as input to the scoping program, followed by a list of readings in descending order of preference (ordered according to the average scoping weight). Input sentences containing coordinated expressions which can be parsed in more than one way may have brackets inserted to indicate which translation is being scoped.

The first example gives an introduction to the logical notation being used. Note that for readings which have not required the comparison of any operators, the average weight is set to the optimal value (1.0) at the time of printing. The next nine examples give an idea of how passivization, shifting and topicalization affect the scoping weights. The weights shown are based on the tables discussed in chapter four. All of the examples have two scoped readings and require one comparison. Sentences (11) to (15) provide some further examples of scoping weights with different quantifiers. Note the strong influence of surface order in these sentences.

In chapter four a distinction was made between the scoping of the existential and universal quantifiers introduced by indefinites. For convenience, we will interpret the indefinite quantifiers to stand for existential quantifiers (ie. *two boys* will be represented in the initial translations as <two boy> but will be interpreted as <E (two boy)> and the universal partitives, which must be introduced by a post-processing rule (see section 3.10), are represented as *u-some, u-num, u-several* and *u-many*. Note that it would have been possible

simply to represent all the partitives as a standard universal quantifier such as *each*. However, for the time being a separate notation is used to allow a separate set of scoping weights to be associated with the partitives. If this later proves to be unecessary it would be a simple matter to change.

Examples (16) to (18) show how all the eight distinct readings of sentence (16) (discussed in section 3.10) can be generated. Since no partitive is present in the first example, only one reading is obtained (the second one being redundant since the two existential quantifiers are commutative). In (17), it is assumed that the post-processing rule has been applied to the first quantifier resulting in two readings. The two new readings both have a predication made of the individual members of the collection of two boys, and in one of the readings the girl functionally depends on the boy. The presence of both partitives in example (18) results in three distinct readings, all involving individual predications and two of them involving functional dependencies. The two additional readings of the sentence would be obtained by applying the post-processing rule to only the second indefinite.

The next examples involve the scoping of negation, quantificational adverbs and unmoved adverbs. Sentence (19) has six readings, the preferred reading being one in which the negation has wide scope. This appears to be intuitively correct assuming that the indefinite is most likely non-specific (the specificity of indefinites being a matter for pragmatics to determine). The scope ordering of negation and adverbs usually, or always, follows surface order. Sentence (22) is used to show that the scope of negation appears to be restricted to the clause in which it occur. This is true of adverbs as well. Sentence (23) simply shows that non-quantificational adverbs are not treated as unscoped operators at the present time and so are not moved.

The next examples show the scoping of quantifiers inside different types of noun complement. Sentences (24) to (26) contain the quantifier *each* inside prepositional, verb

phrase and relative clause complements and it can be seen that the tendency of *each* to scope over the head quantifier decreases correspondingly. Example (27) shows the five readings obtained when three quantifiers forming the pattern "[a-b c]" (discussed in section 5.2) are scoped with each other. The first reading appears to be the one which would be preferred intuitively in a pragmatically neutral context. Sentence (28) contains quantifiers inside nested noun complements and also has five readings, with the preference for the embedded quantifiers taking wide scope (since both complements are prepositional). Sentence (29) contains two quantifiers in adjacent (non-nested) complements and in this example the preference appears to be for the scope ordering to be the reverse of surface order (as is the case with postposed prepositional clauses, but in contrast to quantifiers in formulas joined by a connective).

The next five examples contain quantifiers inside phrases serving as subjects or objects (verb complements). Such phrases are converted into terms using the τ operator. Examples (30) and (31) contain a verb phrase and a full clause respectively, serving as the subjects, and examples (32) and (33) have the corresponding phrases serving as verb complements. The heuristics used, discussed in chapter four, treat the clausal terms as forming fairly complete scope islands. The verb phrase serving as a subject appears to also form a strong scope island; however, this is not the case for a verb phrase in the object position. It is not unreasonable to obtain the reading for (32) in which there is a different tourist per museum. However, this may also depend on the absence of a preceding verb complement. In (34) the presence of the direct object *John* should substantially reduce the ability of *each* to widen scope over the subject; however, the heuristics will need to be modified to take into account the presence of the direct object. [19]

Example (35) contains a *that*-clause serving as an object noun complement. According to Fodor & Sag (1982) such a construct forms a strong scope island and some examples of this were discussed in section 3.1. To represent such constructs we introduce a new predicate,

has-content [10]. Some other method of representation may be used instead; however, a τ operator is needed and this creates a fairly strong trap. This trap could be made stronger by making a distinction between relative clauses introduced by *what* and *that*, if the latter are found to form stronger scope traps. Only one reading is shown since the scoping weight associated with widening the scope of *every* is 0.005 which is below the current minimum allowable weight (0.01). If the value of *$min-weight* were reduced to zero, two additional readings of (35) would be obtained.

Examples (36) and (37) contain quantifiers inside preposed and postposed prepositional adverbials, repsectively. Prepositional phrases serving as adverbials are converted into functions by the operator α_3 (really a special case of α_1 which is used for non-prepositional predicative phrases) and this is taken to offer no resistance to the widening of scope. Therefore, in (36) there is a strong preference for *most* taking wide scope; however, this tendency is markedly reduced when the adverbial is postposed (37). Noun phrases serving as adverbials, such as *every week* as treated as elliptical for prepositional phrases, such as *during each week*, by the parser and therefore are scoped as though they were inside a prepositional phrase (38). Examples (39) and (40) show that *after* is also treated as a preposition when embedding a verb phrase (39) or full clause (40) in an adverbial position (*after* may in other cases be treated as a connective). Sentence (41) shows that a verb phrase converted into a function by the operator α_2 forms a fairly strong scope trap. Finally, sentence (42) gives an example of the need for a lambda expression in the adverbial position.

Examples (43) to (46) show four variations of an *if-then* sentence containing a universal and an existential quantifier. An initial-*if* clause is an absolute scope island for a universal quantifier (43) but not for an existential quantifier (45). The consequent clause appears to be a strong scope island for the universal quantifier when it follows the antecedent (44) but not when it precedes the antecedent (46).

Most of the remaining examples deal with different types of coordination. Examples (47) to (49) contain verb coordination, which is parsed as such (not as verb phrase coordination, as is the case in examples (56) and (64)). The preferred readings of (47) and (48) follow the surface order of the quantifiers and coordinators, which appears to be intuitively correct. The scoping of *and* is quite similar to that of universal quantifiers and therefore there is a preference is for it to be trapped inside a verb phrase serving as a a subject (although it is very hard to decide on intuitive grounds which of the two readings is intended here).

Example (50) has two universal quantifiers embedded inside a VP coordinator and, as discussed in chapter five, universal quantifiers cannot widen scope over an embedding coordinator. [21] Examples (51) to (54) contain NP coordination. The first example has three NPs coordinated by the same *or* and which must therefore be scoped together, giving only two readings. Examples (53) and (54) are taken from Schubert & Pelletier (1982) and the derivations of the different readings are described in that paper.

Examples (55) to (59) contain coordinators inside noun complements. Some suggested heuristics for the scoping of such coordinators are described in section 4.9. Example (55) has a coordinated NP inside a prepositional noun complement and the heuristic used is that coordinators inside a noun complement tend to remain inside the complement. [22] Examples (56) to (58) have coordinated noun complements. The complements are constants in (56), simple phrases in (57) and contain distributive quantifiers in (58). The preferred reading of the latter is the one in which the coordinator takes wide scope allowing *each* to scope inside *most*. This sentence is interesting in that there is a tradeoff here between the tendency of *or* to scope inside *every* and for the two *most*'s to scope outside *every*, since *most* cannot scope outside *or*.

The second and third readings show a problem which is not dealt with by the program at the present. Since the two *most* quantifiers are different (ie. they have different suffixes) they must be allowed to scope separately with *every*. This results in two asymmetric readings (the second and third) in which *most* scopes outside *every* on one side of the coordinator but not on the other. These readings are given higher scoping weights than the symmetric reading in which both *most's* scope inside *each* (since a higher scoping weight is assigned to the *most-each* ordering.) However, in practice such asymmetric readings are very unlikely, if they are ever obtained. There seems to be no simple way of removing such readings, other than by adding a procedure which tests for asymmetric orderings inside readings with scoped coordinators; furthermore, it could be quite tricky to define what is meant by "symmetry" in the general case. The same problem of asymmetric readings occurs with (59).

The next examples contain more than one copy of the same operator in the initial logical translation. Examples (61) to (63) contain reflexives in the object position, which are handled by substitution. Three types of reflexives are shown: a simple noun phrase (60) (note that only two readings are obtained due to commutativity), a noun phrase with a coordinator (61) and a noun phrase containing a quantifier, so that the reflexive may become functionally dependent (62).

Examples (63) to (65) will have two copies of each quantifier in the verb phrase in the initial logical translations as a result of the parser interpreting the coordination as a verb phrase coordination (indicated by the bracketing). It can be seen, most clearly for (64) and (65), that in all of the readings the two copies of each operator scope consistently with respect to other operators and that a comparison between any two operators is only made once.

1. John loves Mary
2. Some boy kissed every girl
3. Some girl was kissed by every boy
4. Every girl was kissed by some boy
5. Some girl was given flowers by every boy
6. Mary read every story to most (of the) children
7. Mary read most (of the) children every story
8. To most (of the) children, Mary read every story
9. To some girl, flowers were given by every boy
10. To some girl, every boy gave flowers

11. Few boys kissed every girl
12. Few boys were kissed by every girl
13. No boy kissed every girl
14. Some boys kissed no girls
15. Some boys kissed few girls

16. Two boys kissed three girls
17. Same - with partitive on "two boys"
18. Same - with partitives on both quantifiers

19. Every boy didn't kiss two girls
20. John doesn't often kiss every girl
21. John often doesn't kiss every girl
22. John thinks that Mary hasn't arrived
23. Probably John will come

24. Most people in each city bought souvenirs
25. Most people visiting each city bought souvenirs
26. Most people who visited each city bought souvenirs
27. Most people on each committee attended several meetings
28. Most people on each (bus on most of the trips) arrived
29. Most people (from each country) (on most buses) arrived

30. Visiting most (of the) museums pleased many tourists
31. That John visited most (of the) museums surprised many people
32. Some tourist wants to visit each museum
33. Some person thinks that John will visit each museum
34. Some person wants John to visit each museum
35. John received several messages that every flight was on time

36. In most cities, several people bought tickets
37. Several people bought tickets in most (of the) cities
38. Every month, many people visit Europe
39. After having visited several cities, John wrote each postcard
40. After John visited several cities, (he) wrote each postcard
41. Having visited each city, most people returned
42. Having been briefed by Jones, each volunteer left

43. If each girl arrives then some boy will be happy
44. Some boy will be happy if each girl arrives
45. If some girl arrives then each boy will be happy
46. Each boy will be happy if some girl arrives

47. Every boy wants to (kiss or hug) Mary
48. John (wants and hopes) to (kiss or hug) Mary
49. To kiss and hug Mary is nice

50. Two boys kissed every girl or hugged every girl
51. Mary read some story to each child or told some story to each child
52. John, Bill and Sam love Sue or Mary
53. John loves and admires Fido or Kim
54. All men want to marry Peggy or Sue
55. Every boy or every girl went to some movie

56. Every person in Zermatt or Mufren skis
57. Every (boy or girl) in Nice swims
58. Every boy in Nice or girl in Monaco swims
59. Every (man on most buses) or (woman on most trains) arrived

60. Some boy wants to buy two ties for himself
61. Some boy or man shaved himself
62. Some man on each trip shaved himself

63. Some boy (kissed or hugged each girl)
64. Mary (read or told some story to each child)
65. Few teachers (read or told some story to each child)

Table 5.2 : Sentences Used as Test Data

5.7 Conclusion

Some critics of approaches such as this may argue that people don't "quantifier scope" (van Lehn, 1978) or that determining functional dependencies is mainly just a pragmatic process. While these arguments carry some weight and should be kept in mind, it has been argued in this thesis that people often do give strong, and sometimes unambiguous, preferences to certain "scope orderings" and that domain independent information such as structural relations among operators and the intrinsic properties of the operators themselves can play a major role in disambiguating sentences containing such operators. One of the main objectives of this work has been to try to separate out, in as far as this is possible, the contributions made to scoping by the above types of domain independent information. It is hoped that this will simplify the later task of adding heuristics based on the effects of domain- and discourse- dependent information.

In chapter four some examples were given of sentences in which an unambiguous interpretation is given to a certain scope ordering. Such sentences include those containing strong scope islands, such as if-then sentences, and also sentences with certain quantifiers such as *any*, *few*, *no* or *more than three* which often have unambiguous or very nearly unambiguous scoped interpretations. An example of one such sentence is

(21) At no dance did John not kiss any girls [13]

which contains three operators requiring scoping but which has an unambiguous interpretation. Examples (11) to (14) in table 5.2 have strongly preferred scope orderings, though sentences having the patterns found in these sentences need not in general be unambiguous. At the other extreme, there are sentences which have almost completely ambiguous, scoped interpretations in the absence of pragmatic information. Sentences

containing the indefinites *some* and *a* such as

(22) Some boy was kissed by every girl

provide a good example. Therefore, the scoping program as it stands is primarily useful in ruling out invalid and redundant readings and readings which contain certain orderings of operators which are so unlikely as to fall below a preset level of minimum acceptability (specified by the parameter *$min-weight*). Otherwise, the scoping weights associated with each reading should only be considered as rough approximations which can serve as a guide to the use of pragmatic information.

How should the present program be extended to incorporate heuristics based on pragmatic information? A distinction might be made here between two ways in which pragmatic information may be used: first, to disambiguate parsing and semantic interpretation in the initial stages of sentence processing, and second, to operate on fully specified logical translations of a sentence to determine inferences etc. The advantage of applying pragmatic information in parallel with the parser, translator and possibly with the scoping program is that this would make it possible to trim the potentially rapid growth of readings at an early stage.

The obvious place to apply pragmatic knowledge in the scoping program is at the stage of determining the preferred orderings of pairs of operators (ie. in the function *scoping-weight*). General world knowledge that could be used would include information about expected relations among objects, such as the fact that there is presumably a different mayor per city in *the mayor of each city*, and about properties of verbs and prepositions such as the fact that it is generally not possible to be *in* more than one place at once as in *a man in each room*. Discourse-dependent knowledge would include a set of possible referents which could be used to scope "specific" indefinites and definites.

The heuristic information used by the program is in need of considerable refinement. For one thing, it is clearly psychologically implausible to suppose that people store their knowledge about domain independent scoping preferences in the form of quite large tables. However, at the moment some tradeoff seems unavoidable between economy and accuracy. As was mentioned in chapter four, it is possible to simplify the structural relations into only three features: surface order, shifting (or topicalization) and embedding. However, this can only be done at considerable expense of accuracy. What will eventually be required is a better motivated model of how people "scope" from which the empirical data shown in these tables will emerge naturally. In the meantime, however, there is a need for more empirical data. One positive aspect of this work is that the number of patterns [24] which need to be considered for scoping in the English language is probably quite limited, and hopefully a considerable fraction of these (including many of the major categories) have already been dealt with here. Therefore, it does seem unreasonable to hope that an approach such as this could be extended to provide domain independent scoping heuristics of sufficient breadth to deal with the range of ordinary discourse.

A more problematic question is how feasible it is to define heuristics solely for independent pairs of operators and to then somehow combine these weights to obtain an overall weight. In sections 5.2 and 5.3 it was argued that the choice of a method of combining scoping weights must be somewhat arbitrary. Some of the difficulties of using the averaging method have already been described. In addition, it may turn out to be quite difficult to apply pragmatic information independently to pairs of operators, even though this appears to be quite feasible for structural and lexical information.

The scoping program could be extended in several ways:

(a) The accuracy and range of the heuristic information needs to be improved. One major weakness at the present is the treatment of quantifiers which have already widened

scope over an embedding operator and which should probably be tagged so that their tendency to subsequently widen scope is reduced. [15] The heuristics, and also the algorithm, for scoping operators in a coordinated noun complement with the head quantifier need to be improved. Some problems with this are discussed in section 5.4 and also in connection with examples (55) to (59). It remains to be determined whether the reduced entrapment of operators by verb phrases, as opposed to clauses, depends on the absence of a tensed verb or on the absence of a subject. The latter is suggested by example (34) and a partial solution was suggested in connection with this example.

(b) The range of operators handled by the program needs to be extended. By having largely separated the heuristic information from the program this will be quite simple to do. Also, by defining "standard" operators, such as *some* and *each*, which are used as standards for all indefinites and universal quantifiers, respectively, it is simple to add new operators of the same class (such operators need only be added to the file *ratios*). This would be especially useful for the classes of adverbs and connectives which contain many members, but is also useful for adding indefinites such as *a finite number of*. The recognition of such classes probably also offers a more psychologically plausible basis for the use of scoping heuristics.

(c) It would be very simple to modify the program to treat all functional expressions, such as tense, aspect, modals and non-quantificational adverbs as unscoped, moveable operators. This has not been done at present because there are some theoretical difficulties, such as the problem of "scope clashes" (eg. Enc 1981) which may result from scoping opaque operators and quantifiers together (although this problem is already present to some extent) and it is very difficult to find any domain independent heuristics for the scoping of such operators. The program does not scope generics and gives only a superficial treatment of quantificational adverbs and this is also largely due to the need for a better theoretical understanding of how to represent and scope such operators.

(d) The English language contains many words spread throughout the discourse which provide significant domain independent information about scoping preferences. Such words include *both* and *each*, when used as adverbs, *together*, *separately*, *same* and *different* etc. Some means must eventually be found to take account of the information provided by such key words (whose function is primarily to disambiguate) by the scoping program.

(e) The program should be extended to handle some more types of input expressions, including those with plural nouns (ie. *(PLUR nom-pred)*, generics *(μ nom-pred)* and complex functions *(func-expr func-expr)*.

(f) When pragmatic information is added, specific indefinites and definites with anaphoric or real-world referents should be tagged with their referents and be forced to scope to their intended position(s). Other readings can be removed using the methods already present for removing readings having an unacceptable ordering.

(g) The handling of reflexives should be expanded to include more than subject-object and object-object combinations. A more difficult, and theoretically unresolved, problem is how to handle the "anaphoric scoping" of quantifiers so that they may bind anaphoric pronouns in non-embedded clauses, such as in donkey sentences. At present the program does not test for the presence of anaphoric pronouns, but assumes that this problem can be dealt with at a later time by scope expansion or by some other process. The latter is needed in any case to deal with cross-discourse anaphora, unless an entire discourse is given to the scoping program and sentential boundaries represented by special connectives. [16]

Footnotes

1. The two readings have been simplified by treating PRES as an identity transformation. In addition, the suffixes added to each word to mark its surface position in the input sentence have been omitted.

2. The corresponding LISP expression would be "(a, (p in, (q each, city,)))".

3. Some missing expressions are (func-expr func-expr) and (PLUR nom-pred). In addition, the handling of quantifiers scoped by the parser, written as "(det var formula formula)", is only partially implemented at present.

4. One difficulty with using a preprocessor is that some means would be needed to indicate if a term corresponding to the subject was present, since this is important to the determination of scoping heuristics. This might be done by reserving the first "slot" for the subject term.

5. However, regardless of what method is used to compare unscoped operators it seems clear that the determination of the pairwise scoping weights must be at least partially made before the operators are raised, to allow use to be made of structural relations.

6. Hobbs & Shieber (1987) would appear to use a *head-first* method, although, as they point out, the highly recursive nature of their program makes such comparisons difficult. For example, it appears to be difficult to make the *b-c* comparison using their method, even when both *b* and *c* scope outside *a*, since the two operators are scoped somewhat independently. The rewriting method of Saint-Dizier would appear to result in the same comparisons being made as in this study, although it is difficult to be sure how scoping heuristics would be used in his system.

7. The algorithm of Hobbs & Shieber would most naturally support a *vertical-first* method of comparison since quantifiers in nested clauses are first scoped relative to the embedding operator. Saint-Dizier does not give rewriting rules for the interchange of quantifiers with opaque predicates such as *think*.

8. Probably examples such as this in which the averaging method does not find the optimal reading do not often occur, but a possible example would be *Most people on several/all of the committees attended few meetings* in which *several* or *all* are slightly favoured to scope inside *most* but strongly favoured to scope outside *few* in the cases in which they do scope outside *most*.

9. "Non-preferred ordering" means an ordering of a pair of operators *ab* which has a lower scoping weight than the alternative ordering *ba*.

10. The LUNAR system does not deal with belief contexts, but does in certain cases make a distinction between opaque (roughly the same as "value-free") and non-opaque ("value-loaded") interpretations of a noun phrase.

11. Perhaps this should be *a few*, which acts like an indefinite, rather than *few*?

12. This problem will not arise in the readings in which both indefinites scope outside the coordinator. However, the point being made here is that in the general case this sequence of application will not work.

13. It might be possible to reduce or eliminate the need for overhead if the operators were scoped one at a time and all copies of the same operator were raised "together". However, raising all copies of an operator in parallel might prove to be equally awkward to implement.

14. As the program stands, there is no need to label readings which contain no copies of any operators.

15. The information is stored on the attribute Ri, where Ri is a given reading.

16. Therefore, equivalent predicates must be stored on the lists of scoped operators mentioned above. This will require each predicate to be tagged with its equivalent predicates (at the start of the program). This has only been partially implemented at present, and so the program will not currently accept input in which copied operators require vertical scoping.

17. The branch-trimming function is only needed to allow all of the operators to be scoped in one pass. It would not be needed if, as suggested earlier, only operators not embedded by an unscoped coordinator were scoped in a given pass.

18. Until the procedure for determining "equivalent predicates" (ie. predicates at the same levels of embedding by coordinators) has been written, operators present in multiple copies may only be scoped horizontally. Some examples of this are described in the next section.

19. This reduction is not made by the program at present but it could be handled by using a separate heuristic for verb phrase complements in different positions. Instead of setting the value of the "construct" parameter to *object* when scoping the embedded phrase, separate constructs *object1* and *object2* could be used. The latter would then create a stronger scope trap.

20. Unfortunately this must be given a suffix for the program to handle it at the present.

21. This heuristic is very useful in practice since, for example, it rules out the three pointless readings of *Every man or every woman left*. However, the program could be modified to obtain these readings by using function *permute* instead of *simple-permute* to scope the operators in *outer-ops* in the function *combine-readings*.

22. This heuristic is handled by the function *coord-scope-weight*.

23. The lack of ambiguity is a result of the *pattern* of operators (as defined earlier) and does not depend in any way on pragmatics. The same lack of ambiguity is present in nonsense sentences having the same pattern such as *At no warg did John not mirl any togs*.

24. Recall that a *pattern* was defined as a combination of pair of operators, of specified types, and in a particular relation to one another.

25. This would be quite simple to implement, but first some empirical data would be required to determine how this should be done.

26. Using the heuristics for connective sentences, it would be possible for a distributive quantifier to widen scope over subsequent sentences but not easily, if at all, over preceding ones, which appears to give the desired results.

Bibliography

Barwise, J. (1979). "On Branching Quantifiers in English", *Journal of Philosophical Logic* 8, 47-80.

Barwise, J. & R. Cooper (1981). "Generalized Quantifiers and Natural Language", *Linguistics and Philosophy* 4, 159-219.

Barwise, J. & J. Perry (1984). *Situations and Attitudes* (Cambridge: MIT Press), 1-352.

Carden, G. (1976). *English Quantifiers: Logical Structure and Linguistic Variation*, (New York: Academic Press), 1-108.

Chomsky, N. (1976). "Condition on Rules of Grammar", *Linguistic Analysis* 2.4.

Cole, P. (1975). "Referential Opacity, Attributiveness, and the Performative Hypothesis", *Proceedings of the Eleventh Regional Meeting of the Chicago Linguistic Society*, 672-686.

Colmerauer, A. (1979) "An Interesting Subset of Natural Language" in Clark and Tarnlund (eds.), *Logic Programming* (London: Academic Press), 45-66.

Cooper, R. (1979) "Variable Binding and Relative Clauses", in F. Guenthner & S.J. Schmidt (eds.), *Formal Semantics and Pragmatics for Natural Languages* (Dordrecht: Reidel), 131-169.

Dahl, V. (1979). "Quantification in a Three-Valued Logic for Natural Language Question-Answering Systems", *Proceedings of the Sixth International Joint Conference for Artificial Intelligence*, 182-187.

DeCarrico, J.S. (1980). "A Counterproposal for Opaque Contexts", *Linguistic Analysis* 6, 1-20.

Donnellan, K. (1966). "Reference and Definite Descriptions", *Philosophical Review* 75, 281-304.

Enc, M. (1981). *Tense Without Scope: An Analysis of Nouns as Indexicals*, unpublished Ph.D. Dissertation (University of Wisconsin, Madison).

Evans, G. (1980), "Pronouns", *Linguistic Inquiry* 11, 337-362.

Fauconnier, G. (1975), "Do Quantifiers Branch?", *Linguistic Inquiry* 6, 555-578.

Fodor, J.D. (1976), *The Linguistic Description of Opaque Contexts*, Ph.D. Dissertation, available from Indiana University Linguistics Club.

Fodor, J.D. & I.A. Sag (1982), "Referential and Quantificational Indefinites", *Linguistics and Philosophy* 5, 355-398.

Gil, D. (1982), "Quantifier Scope, Linguistic Variation, and Natural Language Semantics", *Linguistics and Philosophy* 5, 421-472.

Gillon, B. (1986), "Bare Plurals as Plural Indefinite Noun Phrases", (ms., Department of Philosophy, University of Alberta), 1-60.

Hintikka, J. (1974), "Quantifiers vs. Quantification Theory", *Linguistic Inquiry* 5, 153-177.

Hobbs, J.R. (1983), "An Improper Treatment of Quantification in Ordinary English", *Proceedings of the Twenty-First Annual Meeting of the Association for Computaional Linguistics*, 57-63.

Hobbs, J.R. & S.M. Shieber (1987), "An Algorithm for Generating Quantifier Scopings", *American Journal of Computational Linguistics*.

Hornstein, N. (1984), *Logic as Grammar*, (Cambridge: MIT Press), 1-176.

Hurum, S. & L.K. Schubert (1986), "Two Types of Quantifier Scoping", *Proceedings of the Sixth Canadian Conference on Artificial Intelligence*", 39-43.

Ioup, G. (1975), "Some Universals for Quantifier Scope", in J.P. Kimball (ed.), *Syntax and Semantics*, Vol. 4, (New York: Academic Press), 37-58.

Ioup, G. (1977), "Specificity and the Interpretation of Quantifiers", *Linguistics and Philosophy* 1, 233-245.

Keenan, E. (1974), "The Functional Principle: A generalization of the notion 'subject of'", Chicago Linguistic Society 10.

Kempson, R.M. (1977), *Semantic Theory*, (Cambridge: U.P.), 1-216.

Kempson, R.M. & A. Cormack (1981), "Ambiguity and Quantification", *Linguistics and Philosophy* 4, 259-309.

Lepore, E. & J. Garson (1983), "Pronouns and Quantifier-Scope in English", *Journal of Philosophical Logic* 12, 327-358.

May, R. (1977), *The Grammar of Quantification*, Ph.D. Dissertation, available from Indiana University Linguistics Club.

McCawley, J.D. (1981), *Everything that Linguists have Always Wanted to Know about Logic*, (Chicago: U.P.).

McCord, M.C. (1981), "Focalizers, the Scoping Problem and Semantic Interpretation Rules in Logic Grammars", *Proceedings of the International Workshop on Logic Programming for Expert Systems*, Logicon, (Woodland Hills).

Newmeyer, F.J. (1983), *Grammatical Theory: Its Limits and Possibilities*, (Chicago: U.P.), 1-193.

Pelletier, F.J. & L.K. Schubert (1985), "Mass Expressions: Some Philosophical Problems", in D. Gabbay & F. Guenther (eds.), *Handbook of Philosophical Logic* (Dordrecht: Reidel).

Reinhart, T. (1976), *The Syntactic Domain of Anaphora*, Ph.D. Dissertation, available from Indiana Universtity Linguistics Club.

Rooth, M. & B. Partee (1982), "Conjunction, Type Ambiguity, and Wide Scope 'Or'", *Proceedings of the First West Coast Conference on Formal Linguistics*, 353-362.

Saarinen, E. (1980), "Quantifier Phrases are (at Least) Five Ways Ambiguous in Intensional Contexts", in F. Heny (ed.), *Ambiguities in Intensional Contexts*, (Dordrecht: Reidel), 1-45.

Sag, I.A. (1976), *Deletion and Logical Form*, Ph.D. Dissertation, available from Indiana University Linguistics Club.

Saint-Dizier, P. (1985), "Handling Quantifier Scope Ambiguities in a Semantic Representation of Natural Language Sentences", in V. Dahl & P. Saint-Dizier (eds.), *Natural Language Understanding and Logic Programming*, (North-holland), 49-63.

Schoorl, S. (1980), "Opacity and Transparency: A Pragmatic View", in J. van der Auwera (ed.), *The Semantics of Determiners*, Croom Helm, (Baltimore: London and University Park Press), 156-165.

Schubert, L.K. & F.J. Pelletier (1982), "From English to Logic: Context-Free Computation of 'Conventional' Logical Translation", *American Journal of Computational Linguistics* 8, 26-44. Reprinted (with corrections) in B.J. Grosz, K. Sparck-Jones & B.L. Webber (eds.), *Readings in Natural Language Processing*, (Los Altos: Morgan Kaufman), 1986.

Schubert, L.K. & F.J. Pelletier (1987a), "Problems in the Representation of the Logical Form of Generics, Plurals and Mass Nouns", in E. Lepore (ed.), *New Directions in Semantics*, (Academic Press).

Schubert, L.K. & F.J. Pelletier (1987b), "Generically Speaking, With Remarks on the Interpretation of Pronouns and Tenses". To appear in G. Chierchia, B. Partee & R. Turner (eds.), *Property Theory, Type Theory, and Semantics*, (Dordrecht: Reidel).

van Lehn, K. (1978), *Determining the Scope of English Quantifiers*, MIT Artificial Intelligence Laboratory Technical Report AI-TR-483, 1-123.

Warren, D.H.D. & F.C.N. Pereira (1982), "An Efficient Easily Adaptable System for Interpreting Natural Language Queries", *American Journal of Computational Linguistics* 8, 110-119.

Webber, B.L. (1983), "So What Can We Talk About Now?", in M. Brady & R. Berwick (eds.), *Computational Models of Discourse*, (Cambridge: MIT Press), 331-371. Reprinted in B.J. Grosz, K. Sparck-Jones & B.L. Webber (eds.), *Readings in Natural Language Processing*, (Los Altos: Morgan Kaufman), 1986, 395-414.

Wettstein, H.K. (1981), "Demonstrative Reference and Definite Descriptions", *Philosophical Studies* 40, 241-257.

Wettstein, H.K. (1983), "The Semantic Significance of the Referential-Attributive Distinction", *Philosophical Studies* 44, 187-196.

Wilson, G.M. (1984), "Pronouns and Pronomial Descriptions: A New Semantical Category", *Philosophical Studies* 45, 1-30.

Woods, W.A. (1978), "Semantics and Quantification in Natural Language Question Answering", *Advances in Computers*, vol. 17, (New York: Academic Press), 1-87.

Appendix I

Examples of Scoping Weights

This appendix contains a sample of the sentences used to determine the scoping heuristics. Each sentence is followed by an approximate scoping weight, which is attached to the reading in which the second operator (quantifier or coordinator) scopes outside the first. An exception is made in Part 1 in which the weights are attached to the reading in which the quantifier scopes outside the negation (regardless of the surface order).

Sentences containing plural indefinites may be followed by two scoping weights; the one followed by a "P" is the weight associated with the scoping of the optional partitive introduced by the indefinite. Some weights are placed in brackets or are tagged with the symbol "+". The former indicates that the sentence may not be grammatical and the latter that the preferred reading does not directly correspond to a scope ordering of the quantifiers (this is explained in section 5.5).

Part 1. Quantifiers and Negation

(a) Category: *Full clause*

Some boy didn't kiss Mary	.8
Three boys didn't kiss Mary	.8
Several boys didn't kiss Mary	.8
Many boys didn't kiss Mary	.8
Each boy didn't kiss Mary	.7
Every boy didn't kiss Mary	.5
All of the boys didn't kiss Mary	.4
Both boys didn't kiss Mary	.4
Most of the boys didn't kiss Mary	.9
Not many boys didn't kiss Mary	1.0
Few boys didn't kiss Mary	.9
No boys didn't kiss Mary	(.99)

(b) Category: *Surface subject*

Some boy wasn't kissed by Mary	.9
Each boy wasn't kissed by Mary	.6
Every boy wasn't kissed by Mary	.5
All of the boys weren't kissed by Mary	.5
Both boys weren't kissed by Mary	.3
Most of the boys weren't kissed by Mary	.8
More than three boys weren't kissed by Mary	.7
Few boys weren't kissed by Mary	.99
No boys weren't kissed by Mary	.99

(c) Category: *Deep subject (surface object)*

Mary wasn't kissed by some boy	.4
Mary wasn't kissed by three boys	.4
Mary wasn't kissed by several boys	.3
Mary wasn't kissed by many boys	.2
Mary wasn't kissed by each boy	.2
Mary wasn't kissed by every boy	.2
Mary wasn't kissed by all of the boys	.1
Mary wasn't kissed by both boys	.1
Mary wasn't kissed by most of the boys	.3
?Mary wasn't kissed by few boys	.01
*Mary wasn't kissed by no boys	()

(d) Category: *Direct object*

Mary didn't kiss some boy	.4
Mary didn't kiss three boys	.4
Mary didn't kiss several boys	.3
Mary didn't kiss many boys	.1
Mary didn't kiss each boy	.2
Mary didn't kiss every boy	.1
Mary didn't kiss all of the boys	.02
Mary didn't kiss both boys	.1
Mary didn't kiss most of the boys	.4

(e) Other categories

Preposed Adverbial:	
At most of the dances, John didn't kiss Mary	.99
At some dance, John didn't kiss Mary	.99
At no dances did John not kiss Mary	1.0
Topic:	
To most of the girls, John didn't give flowers	.9
To some girl, John didn't give flowers	.99
To no girls did John not give flowers	1.0
Shifted indirect object:	
John didn't give most of the girls flowers	.3
John didn't give some girl flowers	.5
Indirect object:	
John didn't bring Mary to most of the dances	.3
John didn't bring Mary to some dance	.5

Prepositional object:

Mary doesn't wear rings on most of her fingers	.5
Mary doesn't wear a ring on some finger	.3

Postposed adverbial:

John didn't kiss Mary at most of the dances	.5
John didn't kiss Mary at some dances	.7

Part 2. Horizontal Scoping of Two Quantifiers

(a) Relation: *Full subject - direct object*

Some boy kissed each girl	.8
Some boy kissed every girl	.6
Some boy kissed all of the girls	.3
Some boy kissed several girls	.1P
Some boy kissed most of the girls	.4
Some boy kissed few girls	.1
Some boy kissed no girls	0.0
Every boy kissed a girl	.1
Every boy kissed some girl	.3
Every boy kissed three girls	.2
Every boy kissed several girls	.1
Every boy kissed many girls	.02
Every boy kissed most of the girls	.2
Every boy kissed few girls	.2
Every boy kissed no girls	.05
Most of the boys kissed a girl	.2
Most of the boys kissed some girl	.3
Most of the boys kissed three girls	.2
Most of the boys kissed several girls	.1
Most of the boys kissed many girls	.05
Most of the boys kissed each girl	.6
Most of the boys kissed every girl	.4
Most of the boys kissed all of the girls	.2
Most of the boys kissed most of the girls	.4
Most of the boys kissed few girls	.05
Most of the boys kissed no girls	.02
Few boys kissed some girl	.3
Few boys kissed each girl	.3
Few boys kissed every girl	.1
Few boys kissed all of the girls	.02
Few boys kissed most of the girls	.2
Few boys kissed few girls	.1+
Few boys kissed no girls	.01

No boys kissed some girl	.4
No boys kissed each girl	.2
No boys kissed every girl	.1
No boys kissed all of the girls	.02
No boys kissed most of the girls	.05
No boys kissed few girls	.01
No boys kissed no girls	.01+

(b) Relation: *Full subject - postposed adverbial*

Some boy kissed Mary at each dance	.8
Some boy kissed Mary at every dance	.7
Some boy kissed Mary at all of the dances	.6
Some boy kissed Mary at most of the dances	.7
Some boy kissed Mary at few dances	.4
Some boy kissed Mary at no dances	.4
Every boy kissed Mary at a dance	.3
Every boy kissed Mary at some dance	.5
Every boy kissed Mary at many dances	.2
Most of the boys kissed Mary at some dance	.5
Most of the boys kissed Mary at three dances	.4
Most of the boys kissed Mary at several dances	.3
Most of the boys kissed Mary at many dances	.2
Most of the boys kissed Mary at each dance	.5
Most of the boys kissed Mary at every dance	.5
Most of the boys kissed Mary at all of the dances	.3
Most of the boys kissed Mary at most of the dances	.5
Most of the boys kissed Mary at few dances	.4
Most of the boys kissed Mary at no dances	(.6)
Few boys kissed Mary at a dance	.3
Few boys kissed Mary at some dance	.5
Few boys kissed Mary at each dance	.3
Few boys kissed Mary at every dance	.2
Few boys kissed Mary at all of the dances	.1
Few boys kissed Mary at most of the dances	.3
?Few boys kissed Mary at few dances	(.4)
?Few boys kissed Mary at no dances	(.2)
No boys kissed Mary at some dances	.5
No boys kissed Mary at each dance	.3
No boys kissed Mary at every dance	.2
No boys kissed Mary at all of the dances	.05
No boys kissed Mary at most of the dances	.5

(c) Relation: *Surface subject - direct object*

Some girl was kissed by each boy	.8
Some girl was kissed by every boy	.6
Some girl was kissed by all of the boys	.4
Some girl was kissed by few boys	.05
Most of the girls were kissed by some boy	.3
Most of the girls were kissed by each boy	.5
Most of the girls were kissed by every boy	.3
Most of the girls were kissed by all of the boys	.1
Most of the girls were kissed by most of the boys	.2
Few girls were kissed by some boy	.3
Few girls were kissed by each boy	.4
Few girls were kissed by every boy	.2
Few girls were kissed by all of the boys	.01
Few girls were kissed by most of the boys	.1

(d) Relation: *Direct object - indirect object*

Mary showed most of the pictures to some child	.5
Mary showed most of the pictures to three children	.4
Mary showed most of the pictures to several children	.3
Mary read most of the pictures to many children	.3
Mary showed most of the pictures to each child	.5
Mary showed most of the pictures to every child	.4
Mary showed most of the pictures to all of the children	.3
Mary showed most of the pictures to most of the children	.4
Mary showed most of the pictures to few children	.3
Mary showed most of the pictures to no children	(.4)

(e) Relation: *Shifted indirect object - direct object*

Mary showed most of the children some picture	.4,.02P
Mary showed most of the children three pictures	.3,.01P
Mary showed most of the children several pictures	.2,.01P
Mary showed most of the children many pictures	.1,.01P
Mary showed most of the children each picture	.2
Mary showed most of the children every picture	.1
Mary showed most of the children all of the pictures	.02
Mary showed most of the children most of the pictures	.05
Mary showed most of the children few pictures	.01
Mary showed most of the children no pictures	.01

(f) Relation: *Direct object - postposed adverbial*

John kissed most of the girls at some dance	.5

John kissed most of the girls at each dance	.8
John kissed most of the girls at every dance	.6
John kissed most of the girls at all of the dances	.6
John kissed most of the girls at most of the dances	.6
John kissed most of the girls at few dances	.6
John kissed most of the girls at no dances	(.6)

Part 3. Quantifiers Inside Noun Complements

(a) Relation: *Full relative clause*

Most of the lines which were drawn through each point	.5
Most of the lines which were drawn through every point	.3
Most of the lines which were drawn through all of the points	.1
Most of the lines which were drawn through most of the points	.3
Most of the lines which were drawn through some points	.3
Most of the lines which were drawn through three points	.2
Most of the lines which were drawn through many points	.1
Most of the lines which were drawn through few points	.02
?Most of the lines which were drawn through no points	.00

(b) Relation: *Reduced relative clause*

Most of the lines drawn through each point	.8
Most of the lines drawn through every point	.5
Most of the lines drawn through all of the points	.3
Most of the lines drawn through most of the points	.5
Most of the lines drawn through some point	.5
Most of the lines drawn through several points	.3
Most of the lines drawn through many points	.1
Most of the lines drawn through few points	.02
Most of the lines drawn through no points	.01

(c) Relation: *Prepositional noun complement*

Most of the circles under each square	.9
Most of the circles under every square	.7
Most of the circles under all of the squares	.4
Most of the circles under most of the squares	.7
Most of the circles under some square	.5
Most of the circles under three squares	.3
Most of the circles under several squares	.3
Most of the circles under many squares	.3
Most of the circles under few squares	.1
Most of the circles under no squares	.02

Few circles under each square	.9
Few circles under every sqaure	.6
Few circles under all of the squares	.3

Few circles under most of the squares	.7
Few circles under some squares	.5
Few circles under three squares	.4
Few circles under several squares	.3
Few circles under many squares	.2
?Few circles under few squares	(.1)+
Few circles under no squares	.02

Appendix II

Tables of Scoping Weights

	Category of Quantifier								
Quant	pre	topic	full	surface	deep	direct	shifted	indirect	post
each	.99	.9	.7	.6	.2	.2	.1	.1	.3
every	.99	.5	.4	.5	.2	.1	.1	.05	.2
all	.99	.5	.4	.5	.1	.02	.01	.01	.1
both	.99	.5	.3	.3	.1	.02	.01	.01	.1
most	.99	.9	.9	.8	.3	.4	.3	.3	.5
a	.99	.9	.8	.8	.2	.2	.2	.2	.4
some	.99	.99	.8	.9	.4	.4	.5	.5	.6
three	.99	.9	.8	.9	.4	.4	.4	.4	.5
several	.99	.9	.8	.9	.3	.3	.3	.3	.4
many	.99	.9	.8	.9	.2	.1	.3	.2	.3
few	.99	.99	.9	.99	(.01)	(.01)	(.01)	(.01)	(.4)
no	.99	.99	.99	.99	x	x	x	x	x

Table A2.1 : Scoping of Quantifiers with Negation

The scoping weights are associated with the readings in which a quantifier (shown in the left column) in a given category (shown in the top row) scopes outside the negation operator. Quantifiers in the first four categories precede the negation in surface order. The "x's" signify grammatically unacceptable readings. The categories are:

pre	:	preposed adverbial
topic	:	topic
full	:	full subject
surface	:	surface subject
deep	:	deep subject
direct	:	direct object
shifted	:	shifted indirect object
indirect	:	indirect object
post	:	postposed adverbial

2nd\1st	a	some	every	most	few	no	ave.
a	-	-	.1	.2	.2	.3	.20
some	-	-	.3	.3	.3	.4	.33
three	-	-	.2	.2	.3	.3	.23
several	-	-	.1	.1	.1	.	.10
many	-	-	.02	.05	.02	.02	.10
each	.8	.8	-	.6	.3	.2	.54
every	.7	.6	-	.4	.1	.05	.37
all	.3	.3	-	.2	.02	.02	.17
most	.5	.4	.2	.4	.2	.05	.29
few	(.1)	.1	.2	.05	.1+	.01	.09
no	(.0)	.0	.05	.02	.01	.01+	.02
ave.	.40	.37	.15	.23	.14	.13	.22

(a) Relation: *full subject-direct object*

2nd\1st	a	some	every	most	few	no	ave.
a	-	-	.3	.3	.3	.3	.30
some	-	-	.5	.5	.5	.5	.50
three	-	-	.4	.4	.4	.4	.40
several	-	-	.3	.3	.4	.4	.35
many	-	-	.2	.2	.2	.2	.20
each	.8	.8	-	.5	.3	.3	.54
every	.7	.7	-	.5	.2	.2	.46
all	.6	.6	-	.3	.1	.05	.33
most	.7	.7	.5	.5	.3	.5	.53
few	(.6)	.4	(.6)	.4	(.4)+	(.4)	.47
no	(.6)	.4	(.6)	(.6)	(.2)	(.2)	.43
ave.	.67	.60	.43	.41	.34	.35	.41

(b) Relation: *full subject-postposed adverbial*

2nd\1st	a	some	every	most	few	no	ave.
a	-	-	.5	.5	.4	.4	.33
some	-	-	.5	.5	.6	.6	.55
three	-	-	.4	.4	.5	.5	.45
several	-	-	.3	.3	.3	.3	.30
many	-	-	.2	.3	.2	.2	.28
each	.7	.6	-	.5	.6	.5	.58
every	.6	.5	-	.4	.3	.3	.42
all	.4	.3	-	.3	.2	.3	.30
most	.4	.4	.5	.4	.5	.5	.45
few	.5	(.2)	.6	.3	(.2)+	.4	.36
no	.5	(.2)	.6	(.4)	(.1)	(.4)+	.37
ave.	.52	.37	.45	.39	.39	.40	.40

(c) Relation: *direct-indirect object*

2nd\1st	a	some	every	most	few	no	ave.
a	-	-	.2	.4	.3	.3	.30
some	-	-	.3	.4	.4	.3	.35
three	-	-	.2	.3	.3	.3	.28
several	-	-	.1	.2	.2	.2	.18
many	-	-	.05	.1	.1	.1	.09
each	.3	.3	-	.2	.2	.05	.21
every	.1	.1	-	.1	.1	.02	.08
all	.05	.05	-	.02	.02	.01	.03
most	.1	.1	.1	.05	.02	.02	.07
few	(.02)	(.01)	.01	.01	(.01)+	(.01)	.01
no	(.02)	(.01)	(.01)	.01	.01	(.01)+	.01
ave.	.10	.10	.12	.16	.15	.13	.15

(d) Relation: *shifted indirect-direct object*

2nd\1st	a	some	every	most	few	no	ave.
a	-	-	.5	.5	.5	.5	.50
some	-	-	.5	.5	.6	.8	.60
three	-	-	.5	.5	.5	.5	.50
several	-	-	.5	.5	.5	.6	.53
many	-	-	.5	.5	.5	.5	.50
each	.8	.8	-	.8	.8	.6	.76
every	.7	.7	-	.6	.6	.5	.62
all	.7	.6	-	.6	.6	.4	.58
most	.6	.6	.6	.6	.6	.6	.60
few	.6	.3	.5	.6	(.5)+	(.6)	.52
no	.6	(.3)	.6	(.6)	(.5)	(.5)+	.52
ave.	.67	.55	.53	.57	.56	.55	.57

(e) Relation: *direct object-postposed adverbial*

Table A2.2 : Horizontal Scoping of Two Quantifiers

The scoping weights are associated with the readings in which the quantifiers which appear second in surface order (shown in the left columns) scope outside those which appear first (shown in the top rows). The horizontal relations are shown below each section of the table.

2nd\1st	a	some	every	most	few	no	ave.
a	-	-	.2	.3	.3	.3	.28
some	-	-	.3	.3	.5	.5	.40
three	-	-	.3	.3	.4	.4	.35
several	-	-	.2	.2	.3	.3	.25
many	-	-	.2	.2	.2	.2	.20
each	.9	.9	-	.9	.9	(.6)	.90
every	.8	.8	-	.7	.6	(.4)	.66
all	.6	.5	-	.4	.3	.2	.40
most	.7	.7	.7	.7	.7	.5	.63
few	.1	.05	.1	.1	(.1)+	.02	.08
no	.02	.02	.02	.02	.02	(.01)+	.02
ave.	.52	.50	.25	.37	.39	.31	.38

Table A2.3 : Scoping of Quantifiers in Prepositional Noun Complements

The scoping weights are associated with the readings in which the quantifier inside the prepositional noun complement (shown in the left column) scope outside the head quantifier (shown in the top row).

```
pre-pre       all   .50
pre-post      all   .10
pre-others    list  .20 .01 .01 .01 .01 .01 .20 .01
topic-subj    all   .20
topic-neg     all   .01
topic-post    list  .20 .01 .01 .01 .01 .01 .20 .01
topic-others  list  .20 .01 .01 .01 .01 .01 .20 .01

subj-neg      list  .15 .30 .35 .15 .05 .01 .10 .50
subj-obj      list  .30 .10 .60 .30 .10 .10 .30 .10
subj-post     list  .55 .30 .60 .50 .20 .20 .30 .10
dir-indir     list  .35 .30 .65 .50 .00 .00 .50 .50
indir-dir     list  .01 .05 .16 .05 .00 .00 .50 .50
neg-obj       list  .45 .50 .15 .40 .01 .01 .50 .50
neg-post      list  .60 .50 .30 .50 .40 .10 .50 .50
obj-post      list  .65 .50 .80 .70 .00 .00 .50 .50
post-post     all   .70

conn-conn     all   .20
restr-restr   all   .20
subj-func     all   .50
neg-func      all   .01
func-neg      all   .00
func-func     all   .01
func-obj      all   .50
func-post     all   .50

subj-vp       list  .35 .10 .10 .10 .00 .00 .50 .10
subj-cl       list  .30 .00 .00 .01 .00 .00 .20 .00
obj-vp        list  .35 .10 .20 .10 .00 .00 .30 .10
obj-cl        list  .30 .01 .01 .01 .00 .00 .20 .00
ALPHA1        all   .10
ALPHA2        all   .20
ALPHA3        all   .99

restr         list  .00 .00 .00 .00 .00 .00 .30 .30
quant-pp      list  .40 .50 .70 .50 .05 .01 .30 .10
quant-vp      list  .10 .30 .50 .30 .01 .01 .20 .05
quant-cl      list  .01 .10 .20 .10 .01 .01 .10 .00

pre-if        list  .30 .00 .00 .00 .00 .00 .05 .00
post-if       list  .30 .10 .10 .01 .01 .01 .30 .01
pre-then      list  .50 .10 .30 .30 .20 .10 .50 .50
post-then     list  .30 .01 .01 .01 .00 .00 .40 .40
```

Table A2.4 : Input File "Weights"

```
* pre-advl
  pre-advl      pre-pre
  post-advl     pre-post
  others        pre-others

* topic
  full-subj     topic-subj
  surface-subj  topic-subj
  post-advl     topic-post
  neg           topic-neg
  others        topic-others

* full-subj
  pred          subj-pred
  dir-obj       subj-obj
  indir-obj     subj-obj
  prep-obj      subj-obj
  post-advl     subj-post
  neg           subj-neg
  func          subj-func

* surface-subj
  pred          subj-pred
  dir-obj       subj-obj
  indir-obj     subj-obj
  prep-obj      subj-obj
  post-advl     subj-post
  neg           subj-neg
  func          subj-func

* dir-obj
  indir-obj     dir-indir
  prep-obj      dir-indir
  post-advl     obj-post

* indir-obj
  dir-obj       indir-dir
  prep-obj      indir-dir
  post-advl     obj-post

* func
  func          func-func
  neg           func-neg
  dir-obj       func-obj
  indir-obj     func-obj
  prep-obj      func-obj
  post-advl     func-post
```

```
* pred
 pred          pred-pred
 dir-obj       pred-obj
 indir-obj     pred-obj
 prep-obj      pred-obj
 post-advl     pred-post

* post-advl
 post-advl     post-post

* neg
 dir-obj       neg-obj
 indir-obj     neg-obj
 prep-obj      neg-obj
 post-advl     neg-post
 func          neg-func

* subject
 vp            subj-vp
 cl            subj-cl

* object
 vp            obj-vp
 cl            obj-cl

* restr
 vp            quant-vp
 cl            quant-cl

* restr
 restr         restr-restr

* conn
 conn          conn-conn
```

Table A2.5 : Input File "Relations"

```
* every each 0.7
subj-post 0.8
neg-post  0.8
obj-post  0.8

* all each 0.33
subj-post 0.7
neg-post  0.7
obj-post  0.7

* num some 0.9

* several some 0.75
subj-post  0.8
obj-post   0.8

* many some 0.65
subj-post   0.7
obj-post    0.7

* u-num u-some 0.7

* u-several u-some 0.5

* u-many u-some 0.3
```

Table A2.6 : Input File "Ratios"

Appendix III

Output of Test Data

Sentence 1

(i John1 (f PRES (p love2 Mary3)))

1. The average weight is 1.0 based on 0 comparisons

(i John1 (f PRES (p love2 Mary3)))

time used = 106 msecs.

Sentence 2

(i (q some1 boy2) (f PAST (p kiss3 (q every4 girl5))))

1. The average weight is 0.58 based on 1 comparison

```
(q some1
  y8
  (i y8 boy2)
  (q every4 y11 (i y11 girl5) (i y8 (f PAST (p kiss3 y11)))))
```

2. The average weight is 0.42 based on 1 comparison

```
(q every4
  y11
  (i y11 girl5)
  (q some1 y8 (i y8 boy2) (i y8 (f PAST (p kiss3 y11)))))
```

time used = 81 msecs.

Sentence 3

(i (q every4 boy5) (f PAST (p kiss3 (q some1 girl2))))

1. The average weight is 0.58 based on 1 comparison

```
(q some1
  y16
  (i y16 girl2)
  (q every4 y13 (i y13 boy5) (i y13 (f PAST (p kiss3 y16)))))
```

2. The average weight is 0.42 based on 1 comparison

```
(q every4
```

```
  y13
  (i y13 boy5)
  (q some1 y16 (i y16 girl2) (i y13 (f PAST (p kiss3 y16)))))
```

time used = 151 msecs.

Sentence 4

(i (q some4 boy5) (f PAST (p kiss3 (q every1 girl2))))

1. The average weight is 0.7 based on 1 comparison

```
(q every1
  y21
  (i y21 girl2)
  (q some4 y18 (i y18 boy5) (i y18 (f PAST (p kiss3 y21)))))
```

2. The average weight is 0.3 based on 1 comparison

```
(q some4
  y18
  (i y18 boy5)
  (q every1 y21 (i y21 girl2) (i y18 (f PAST (p kiss3 y21)))))
```

time used = 80 msecs.

Sentence 5

(i (q every5 boy6) (f PAST (p give3 (MU flowers4) (q some1 girl2))))

1. The average weight is 0.58 based on 1 comparison

```
(q some1
  y26
  (i y26 girl2)
  (q every5 y23 (i y23 boy6) (i y23 (f PAST (p give3 (MU flowers4) y26)))))
```

2. The average weight is 0.42 based on 1 comparison

```
(q every5
  y23
  (i y23 boy6)
  (q some1 y26 (i y26 girl2) (i y23 (f PAST (p give3 (MU flowers4) y26)))))
```

time used = 92 msecs.

Sentence 6

(i Mary1 (f PAST (p read2 (q every3 story4) (q most5 child6))))

1. The average weight is 0.5 based on 1 comparison

```
(q every3
  y30
  (i y30 story4)
  (q most5 y31 (i y31 child6) (i Mary1 (f PAST (p read2 y30 y31)))))
```

2. The average weight is 0.5 based on 1 comparison

```
(q most5
  y31
  (i y31 child6)
  (q every3 y30 (i y30 story4) (i Mary1 (f PAST (p read2 y30 y31)))))
```

time used = 92 msecs.

Sentence 7

```
(i Mary1 (f PAST (p read2 (q every5 story6) (q most3 child4))))
```

1. The average weight is 0.88 based on 1 comparison

```
(q most3
  y36
  (i y36 child4)
  (q every5 y35 (i y35 story6) (i Mary1 (f PAST (p read2 y35 y36)))))
```

2. The average weight is 0.11 based on 1 comparison

```
(q every5
  y35
  (i y35 story6)
  (q most3 y36 (i y36 child4) (i Mary1 (f PAST (p read2 y35 y36)))))
```

time used = 90 msecs.

Sentence 8

```
(i Mary3 (f PAST (p read4 (q every5 story6) (q most1 child2))))
```

1. The average weight is 0.99 based on 1 comparison

```
(q most1
  y41
  (i y41 child2)
  (q every5 y40 (i y40 story6) (i Mary3 (f PAST (p read4 y40 y41)))))
```

time used = 74 msecs.

Sentence 9

(i (q every5 boy6) (f PAST (p give4 (MU flowers3) (q some1 girl2))))

1. The average weight is 0.99 based on 1 comparison

(q some1
 y46
 (i y46 girl2)
 (q every5 y43 (i y43 boy6) (i y43 (f PAST (p give4 (MU flowers3) y46)))))

time used = 76 msecs.

Sentence 10

(i (q every3 boy4) (f PAST (p give5 (MU flowers6) (q some1 girl2))))

1. The average weight is 0.8 based on 1 comparison

(q some1
 y51
 (i y51 girl2)
 (q every3 y48 (i y48 boy4) (i y48 (f PAST (p give5 (MU flowers6) y51)))))

2. The average weight is 0.2 based on 1 comparison

(q every3
 y48
 (i y48 boy4)
 (q some1 y51 (i y51 girl2) (i y48 (f PAST (p give5 (MU flowers6) y51)))))

time used = 168 msecs.

Sentence 11

(i (q few1 boy2) (f PAST (p kiss3 (q every4 girl5))))

1. The average weight is 0.66 based on 1 comparison

(q few1
 y53
 (i y53 boy2)
 (q every4 y56 (i y56 girl5) (i y53 (f PAST (p kiss3 y56)))))

2. The average weight is 0.33 based on 1 comparison

(q every4
 y56
 (i y56 girl5)
 (q few1 y53 (i y53 boy2) (i y53 (f PAST (p kiss3 y56)))))

time used = 83 msecs.

Sentence 12

(i (q every4 girl5) (f PAST (p kiss3 (q few1 boy2))))

1. The average weight is 0.66 based on 1 comparison

```
(q few1
  y61
  (i y61 boy2)
  (q every4 y58 (i y58 girl5) (i y58 (f PAST (p kiss3 y61)))))
```

2. The average weight is 0.33 based on 1 comparison

```
(q every4
  y58
  (i y58 girl5)
  (q few1 y61 (i y61 boy2) (i y58 (f PAST (p kiss3 y61)))))
```

time used = 83 msecs.

Sentence 13

(i (q no1 boy2) (f PAST (p kiss3 (q every4 girl5))))

1. The average weight is 0.79 based on 1 comparison

```
(q no1
  y63
  (i y63 boy2)
  (q every4 y66 (i y66 girl5) (i y63 (f PAST (p kiss3 y66)))))
```

2. The average weight is 0.21 based on 1 comparison

```
(q every4
  y66
  (i y66 girl5)
  (q no1 y63 (i y63 boy2) (i y63 (f PAST (p kiss3 y66)))))
```

time used = 82 msecs.

Sentence 14

(i (q some1 boy2) (f PAST (p kiss3 (q no4 girl5))))

1. The average weight is 0.9 based on 1 comparison

```
(q some1
  y68
  (i y68 boy2)
  (q no4 y71 (i y71 girl5) (i y68 (f PAST (p kiss3 y71)))))
```

2. The average weight is 0.1 based on 1 comparison

```
(q no4
  y71
  (i y71 girl5)
  (q some1 y68 (i y68 boy2) (i y68 (f PAST (p kiss3 y71)))))
```

time used = 81 msecs.

Sentence 15

```
(i (q some1 boy2) (f PAST (p kiss3 (q few4 girl5))))
```

1. The average weight is 0.9 based on 1 comparison

```
(q some1
  y73
  (i y73 boy2)
  (q few4 y76 (i y76 girl5) (i y73 (f PAST (p kiss3 y76)))))
```

2. The average weight is 0.1 based on 1 comparison

```
(q few4
  y76
  (i y76 girl5)
  (q some1 y73 (i y73 boy2) (i y73 (f PAST (p kiss3 y76)))))
```

time used = 81 msecs.

Sentence 16

```
(i (q two1 boy2) (f PAST (p kiss3 (q three4 girl5))))
```

1. The average weight is 1.0 based on 0 comparisons

```
(q two1
  y78
  (i y78 boy2)
  (q three4 y81 (i y81 girl5) (i y78 (f PAST (p kiss3 y81)))))
```

time used = 126 msecs.

Sentence 17

```
(i (q u-two1 (l x6 (i x6 element-of7 (q two1 boy2))))
  (f PAST (p kiss3 (q three4 girl5))))
```

1. The average weight is 0.73 based on 1 comparison

```
(q two1
  y85
  (i y85 boy2)
  (q u-two1
    x6
    (i x6 element-of7 y85)
    (q three4 y88 (i y88 girl5) (i x6 (f PAST (p kiss3 y88))))))
```

2. The average weight is 0.27 based on 1 comparison

```
(q two1
  y85
  (i y85 boy2)
  (q three4
    y88
    (i y88 girl5)
    (q u-two1 x6 (i x6 element-of7 y85) (i x6 (f PAST (p kiss3 y88))))))
```

time used = 130 msecs.

Sentence 18

```
(i (q u-two1 (l x6 (i x6 element-of7 (q two1 boy2))))
  (f PAST (p kiss3 (q u-three4 (l x8 (i x8 element-of9 (q three4 girl5)))))))
```

1. The average weight is 0.83 based on 2 comparisons

```
(q two1
  y92
  (i y92 boy2)
  (q u-two1
    x6
    (i x6 element-of7 y92)
    (q three4
      y97
      (i y97 girl5)
      (q u-three4 x8 (i x8 element-of9 y97) (i x6 (f PAST (p kiss3 x8)))))))
```

2. The average weight is 0.6 based on 2 comparisons

```
(q two1
  y92
  (i y92 boy2)
  (q three4
    y97
    (i y97 girl5)
    (q u-two1
      x6
```

```
    (i x6 element-of7 y92)
    (q u-three4 x8 (i x8 element-of9 y97) (i x6 (f PAST (p kiss3 x8)))))))
```

3. The average weight is 0.17 based on 2 comparisons

```
(q three4
  y97
  (i y97 girl5)
  (q u-three4
    x8
    (i x8 element-of9 y97)
    (q two1
      y92
      (i y92 boy2)
      (q u-two1 x6 (i x6 element-of7 y92) (i x6 (f PAST (p kiss3 x8)))))))
```

time used = 221 msecs.

Sentence 19

```
(i (q every1 boy2) (f PAST (f not3 (p kiss4 (q two5 girl6)))))
```

1. The average weight is 0.69 based on 3 comparisons

```
(q every1
  y99
  (i y99 boy2)
  (f not3 (q two5 y103-(i y103 girl6) (i y99 (f PAST (p kiss4 y103))))))
```

2. The average weight is 0.63 based on 3 comparisons

```
(q every1
  y99
  (i y99 boy2)
  (q two5 y103 (i y103 girl6) (f not3 (i y99 (f PAST (p kiss4 y103))))))
```

3. The average weight is 0.52 based on 3 comparisons

```
(f not3
  (q every1
    y99
    (i y99 boy2)
    (q two5 y103 (i y103 girl6) (i y99 (f PAST (p kiss4 y103))))))
```

4. The average weight is 0.47 based on 3 comparisons

```
(q two5
  y103
  (i y103 girl6)
  (q every1 y99 (i y99 boy2) (f not3 (i y99 (f PAST (p kiss4 y103))))))
```

5. The average weight is 0.37 based on 3 comparisons

```
(f not3
  (q two5
    y103
    (i y103 girl6)
    (q every1 y99 (i y99 boy2) (i y99 (f PAST (p kiss4 y103))))))
```

6. The average weight is 0.3 based on 3 comparisons

```
(q two5
  y103
  (i y103 girl6)
  (f not3 (q every1 y99 (i y99 boy2) (i y99 (f PAST (p kiss4 y103))))))
```

time used = 233 msecs.

Sentence 20

```
(i John1 (f PRES (f not2 (f often3 (p kiss4 (q every5 girl6))))))
```

1. The average weight is 0.79 based on 3 comparisons

```
(f not2
  (f often3 (q every5 y109 (i y109 girl6) (i John1 (f PRES (p kiss4 y109))))))
```

2. The average weight is 0.79 based on 3 comparisons

```
(f not2
  (q every5 y109 (i y109 girl6) (f often3 (i John1 (f PRES (p kiss4 y109))))))
```

3. The average weight is 0.53 based on 3 comparisons

```
(q every5
  y109
  (i y109 girl6)
  (f not2 (f often3 (i John1 (f PRES (p kiss4 y109))))))
```

4. The average weight is 0.46 based on 3 comparisons

```
(f often3
  (f not2 (q every5 y109 (i y109 girl6) (i John1 (f PRES (p kiss4 y109))))))
```

5. The average weight is 0.2 based on 3 comparisons

```
(f often3
  (q every5 y109 (i y109 girl6) (f not2 (i John1 (f PRES (p kiss4 y109))))))
```

6. The average weight is 0.2 based on 3 comparisons

```
(q every5
  y109
  (i y109 girl6)
  (f often3 (f not2 (i John1 (f PRES (p kiss4 y109))))))
```

time used = 254 msecs.

Sentence 21

(i John1 (f often2 (f PRES (f not3 (p kiss4 (q every5 girl6))))))

1. The average weight is 0.79 based on 3 comparisons

(f often2
  (f not3 (q every5 y115 (i y115 girl6) (i John1 (f PRES (p kiss4 y115))))))

2. The average weight is 0.53 based on 3 comparisons

(f often2
  (q every5 y115 (i y115 girl6) (f not3 (i John1 (f PRES (p kiss4 y115))))))

3. The average weight is 0.53 based on 3 comparisons

(q every5
  y115
  (i y115 girl6)
  (f often2 (f not3 (i John1 (f PRES (p kiss4 y115))))))

time used = 166 msecs.

Sentence 22

(i John1
  (f PRES (p think2 (TAU1 (i Mary3 (f PRES (f PERF (f not4 (p arrive5)))))))))

1. The average weight is 1.0 based on 1 comparison

(i John1
  (f PRES (p think2 (TAU1 (f not4 (i Mary3 (f PRES (f PERF (p arrive5)))))))))

time used = 166 msecs.

Sentence 23

(f probably1 (i John2 (f FUT (p come3))))

1. The average weight is 1.0 based on 0 comparisons

(f probably1 (i John2 (f FUT (p come3))))

time used = 41 msecs.

Sentence 24

```
(i (q most1 (l x2 (i (i x2 person3) & (i x2 in4 (q each5 city6)))))
  (f PAST (p buy7 (MU souvenirs8))))
```

1. The average weight is 0.7 based on 1 comparison

```
(q each5
  y133
  (i y133 city6)
  (q most1
    x2
    (i (i x2 person3) & (i x2 in4 y133))
    (i x2 (f PAST (p buy7 (MU souvenirs8))))))
```

2. The average weight is 0.3 based on 1 comparison

```
(q most1
  x2
  (i (i x2 person3) & (q each5 y133 (i y133 city6) (i x2 in4 y133)))
  (i x2 (f PAST (p buy7 (MU souvenirs8)))))
```

time used = 136 msecs.

Sentence 25

```
(i (q most1
     (l x2 (i (i x2 person3) & (i x2 (f PROG (p visit4 (q each5 city6)))))))
  (f PAST (p buy7 (MU souvenirs8))))
```

1. The average weight is 0.5 based on 1 comparison

```
(q most1
  x2
  (i (i x2 person3)
    &
    (q each5 y143 (i y143 city6) (i x2 (f PROG (p visit4 y143)))))
  (i x2 (f PAST (p buy7 (MU souvenirs8)))))
```

2. The average weight is 0.5 based on 1 comparison

```
(q each5
  y143
  (i y143 city6)
  (q most1
    x2
    (i (i x2 person3) & (i x2 (f PROG (p visit4 y143))))
    (i x2 (f PAST (p buy7 (MU souvenirs8))))))
```

time used = 235 msecs.

Sentence 26

```
(i (q most1
     (l x2 (i (i x2 person3) & (i x2 (f PAST (p visit4 (q each5 city6)))))))
   (f PAST (p buy7 (MU souvenirs8))))
```

1. The average weight is 0.8 based on 1 comparison

```
(q most1
  x2
  (i (i x2 person3)
    &
    (q each5 y153 (i y153 city6) (i x2 (f PAST (p visit4 y153)))))
  (i x2 (f PAST (p buy7 (MU souvenirs8)))))
```

2. The average weight is 0.19 based on 1 comparison

```
(q each5
  y153
  (i y153 city6)
  (q most1
    x2
    (i (i x2 person3) & (i x2 (f PAST (p visit4 y153))))
    (i x2 (f PAST (p buy7 (MU souvenirs8))))))
```

time used = 158 msecs.

Sentence 27

```
(i (q most1 (l x2 (i (i x2 person3) & (i x2 on3 (q each4 committee5)))))
   (f PAST (p attend6 (q several7 meeting8))))
```

1. The average weight is 0.75 based on 3 comparisons

```
(q each4
  y161
  (i y161 committee5)
  (q most1
    x2
    (i (i x2 person3) & (i x2 on3 y161))
    (q several7 y164 (i y164 meeting8) (i x2 (f PAST (p attend6 y164))))))
```

2. The average weight is 0.56 based on 3 comparisons

```
(q each4
  y161
  (i y161 committee5)
  (q several7
    y164
    (i y164 meeting8)
    (q most1
      x2
      (i (i x2 person3) & (i x2 on3 y161))
```

(i x2 (f PAST (p attend6 y164))))))

3. The average weight is 0.53 based on 2 comparisons

```
(q most1
  x2
  (i (i x2 person3) & (q each4 y161 (i y161 committee5) (i x2 on3 y161)))
  (q several7 y164 (i y164 meeting8) (i x2 (f PAST (p attend6 y164)))))
```

4. The average weight is 0.38 based on 3 comparisons

```
(q several7
  y164
  (i y164 meeting8)
  (q each4
    y161
    (i y161 committee5)
    (q most1
      x2
      (i (i x2 person3) & (i x2 on3 y161))
      (i x2 (f PAST (p attend6 y164))))))
```

5. The average weight is 0.26 based on 2 comparisons

```
(q several7
  y164
  (i y164 meeting8)
  (q most1
    x2
    (i (i x2 person3) & (q each4 y161 (i y161 committee5) (i x2 on3 y161)))
    (i x2 (f PAST (p attend6 y164)))))
```

time used = 324 msecs.

Sentence 28

```
(i (q most1
     (l x2
       (i (i x2 person3)
          &
          (i x2
            on4
            (q each5
              (l x6 (i (i x6 bus7) & (i x6 on8 (q most9 trip10))))))))))
  (f PAST (p arrive11)))
```

1. The average weight is 0.6 based on 2 comparisons

```
(q each5
  x6
  (i (i x6 bus7) & (q most9 y174 (i y174 trip10) (i x6 on8 y174)))
  (q most1
    x2
```

```
    (i (i x2 person3) & (i x2 on4 x6))
    (i x2 (f PAST (p arrive11))))))
```

2. The average weight is 0.56 based on 3 comparisons.

```
(q most9
  y174
  (i y174 trip10)
  (q each5
    x6
    (i (i x6 bus7) & (i x6 on8 y174))
    (q most1
      x2
      (i (i x2 person3) & (i x2 on4 x6))
      (i x2 (f PAST (p arrive11))))))
```

3. The average weight is 0.43 based on 3 comparisons

```
(q most1
  x2
  (i (i x2 person3)
    &
    (q most9
      y174
      (i y174 trip10)
      (q each5 x6 (i (i x6 bus7) & (i x6 on8 y174)) (i x2 on4 x6))))
  (i x2 (f PAST (p arrive11))))
```

4. The average weight is 0.43 based on 3 comparisons

```
(q most9
  y174
  (i y174 trip10)
  (q most1
    x2
    (i (i x2 person3)
      &
      (q each5 x6 (i (i x6 bus7) & (i x6 on8 y174)) (i x2 on4 x6)))
    (i x2 (f PAST (p arrive11)))))
```

5. The average weight is 0.4 based on 2 comparisons

```
(q most1
  x2
  (i (i x2 person3)
    &
    (q each5
      x6
      (i (i x6 bus7) & (q most9 y174 (i y174 trip10) (i x6 on8 y174)))
      (i x2 on4 x6)))
  (i x2 (f PAST (p arrive11))))
```

time used = 287 msecs.

Sentence 29

```
(i (q most1
     (l x2
       (i (i x2 person3)
         &
         (i x2 from4 (q each5 country6))
         (i x2 on5 (q most6 bus7)))))
   (f PAST (p arrive8)))
```

1. The average weight is 0.66 based on 3 comparisons

```
(q each5
  y182
  (i y182 country6)
  (q most6
    y184
    (i y184 bus7)
    (q most1
      x2
      (i (i x2 person3) & (i x2 from4 y182) (i x2 on5 y184))
      (i x2 (f PAST (p arrive8))))))
```

2. The average weight is 0.6 based on 2 comparisons

```
(q each5
  y182
  (i y182 country6)
  (q most1
    x2
    (i (i x2 person3)
      &
      (i x2 from4 y182)
      (q most6 y184 (i y184 bus7) (i x2 on5 y184)))
    (i x2 (f PAST (p arrive8)))))
```

3. The average weight is 0.46 based on 3 comparisons

```
(q most6
  y184
  (i y184 bus7)
  (q each5
    y182
    (i y182 country6)
    (q most1
      x2
      (i (i x2 person3) & (i x2 from4 y182) (i x2 on5 y184))
      (i x2 (f PAST (p arrive8))))))
```

4. The average weight is 0.4 based on 2 comparisons

```
(q most1
  x2
  (i (i x2 person3)
```

```
    &
    (q each5 y182 (i y182 country6) (i x2 from4 y182))
    (q most6 y184 (i y184 bus7) (i x2 on5 y184)))
  (i x2 (f PAST (p arrive8))))
```

5. The average weight is 0.4 based on 2 comparisons

```
(q most6
  y184
  (i y184 bus7)
  (q most1
    x2
    (i (i x2 person3)
      &
      (q each5 y182 (i y182 country6) (i x2 from4 y182))
      (i x2 on5 y184))
    (i x2 (f PAST (p arrive8)))))
```

time used = 341 msecs.

Sentence 30

```
(i (TAU2 (f PROG (p visit1 (q most2 museum3))))
  (f PAST (p please4 (q many5 tourist6))))
```

1. The average weight is 0.9 based on 1 comparison

```
(q many5
  y193
  (i y193 tourist6)
  (i (TAU2 (q most2 y190 (i y190 museum3) (f PROG (p visit1 y190))))
    (f PAST (p please4 y193))))
```

2. The average weight is 0.45 based on 2 comparisons

```
(q most2
  y190
  (i y190 museum3)
  (q many5
    y193
    (i y193 tourist6)
    (i (TAU2 (f PROG (p visit1 y190))) (f PAST (p please4 y193)))))
```

3. The average weight is 0.14 based on 2 comparisons

```
(q many5
  y193
  (i y193 tourist6)
  (q most2
    y190
    (i y190 museum3)
    (i (TAU2 (f PROG (p visit1 y190))) (f PAST (p please4 y193)))))
```

time used = 136 msecs.

Sentence 31

(i (TAU1 (i John1 (f PAST (p visit2 (q most3 museum4)))))
(f PAST (p surprise5 (q many6 person7))))

1. The average weight is 0.99 based on 1 comparison

(q many6
y201
(i y201 person7)
(i (TAU1
(q most3 y198 (i y198 museum4) (i John1 (f PAST (p visit2 y198)))))
(f PAST (p surprise5 y201))))

2. The average weight is 0.4 based on 2 comparisons

(q most3
y198
(i y198 museum4)
(q many6
y201
(i y201 person7)
(i (TAU1 (i John1 (f PAST (p visit2 y198))))
(f PAST (p surprise5 y201)))))

3. The average weight is 0.1 based on 2 comparisons

(q many6
y201
(i y201 person7)
(q most3
y198
(i y198 museum4)
(i (TAU1 (i John1 (f PAST (p visit2 y198))))
(f PAST (p surprise5 y201)))))

time used = 151 msecs.

Sentence 32

(i (q some1 tourist2)
(f PRES (p want3 (TAU2 (f INF (p visit3 (q each4 museum5)))))))

1. The average weight is 0.8 based on 1 comparison

(q some1
y203
(i y203 tourist2)
(i y203

```
    (f PRES
      (p want3
        (TAU2 (q each4 y208 (i y208 museum5) (f INF (p visit3 y208))))))))
```

2. The average weight is 0.4 based on 2 comparisons

```
(q each4
  y208
  (i y208 museum5)
  (q some1
    y203
    (i y203 tourist2)
    (i y203 (f PRES (p want3 (TAU2 (f INF (p visit3 y208)))))))))
```

3. The average weight is 0.29 based on 2 comparisons

```
(q some1
  y203
  (i y203 tourist2)
  (q each4
    y208
    (i y208 museum5)
    (i y203 (f PRES (p want3 (TAU2 (f INF (p visit3 y208)))))))))
```

time used = 144 msecs.

Sentence 33

```
(i (q some1 person2)
  (f PRES (p think3 (TAU1 (i John4 (f FUT (p visit5 (q each6 museum7))))))))
```

1. The average weight is 0.99 based on 1 comparison

```
(q some1
  y210
  (i y210 person2)
  (i y210
    (f PRES
      (p think3
        (TAU1
          (q each6
            y216
            (i y216 museum7)
            (i John4 (f FUT (p visit5 y216))))))))))
```

2. The average weight is 0.3 based on 2 comparisons

```
(q each6
  y216
  (i y216 museum7)
  (q some1
    y210
    (i y210 person2)
```

(i y210 (f PRES (p think3 (TAU1 (i John4 (f FUT (p visit5 y216)))))))))

3. The average weight is 0.2 based on 2 comparisons

```
(q some1
  y210
  (i y210 person2)
  (q each6
    y216
    (i y216 museum7)
    (i y210 (f PRES (p think3 (TAU1 (i John4 (f FUT (p visit5 y216)))))))))
```

time used = 163 msecs.

Sentence 34

```
(i (q some1 person2)
  (f PRES (p want3 John4 (TAU2 (f INF (p visit5 (q each6 museum7)))))))
```

1. The average weight is 0.8 based on 1 comparison

```
(q some1
  y218
  (i y218 person2)
  (i y218
    (f PRES
      (p want3
        John4
        (TAU2 (q each6 y223 (i y223 museum7) (f INF (p visit5 y223))))))))
```

2. The average weight is 0.4 based on 2 comparisons

```
(q each6
  y223
  (i y223 museum7)
  (q some1
    y218
    (i y218 person2)
    (i y218 (f PRES (p want3 John4 (TAU2 (f INF (p visit5 y223)))))))))
```

3. The average weight is 0.29 based on 2 comparisons

```
(q some1
  y218
  (i y218 person2)
  (q each6
    y223
    (i y223 museum7)
    (i y218 (f PRES (p want3 John4 (TAU2 (f INF (p visit5 y223))))))))
```

time used = 234 msecs.

Sentence 35

```
(i John1
  (f PAST
    (p receive2
      (q several3
        (l x4
          (i (i x4 message5)
            &
            (i x4
              (f PRES
                (p has-content6
                  (TAU1
                    (i (q every7 flight8)
                      (f PAST (p on-time9)))))))))))))
```

1. The average weight is 0.99 based on 1 comparison

```
(q several3
  x4
  (i (i x4 message5)
    &
    (i x4
      (f PRES
        (p has-content6
          (TAU1
            (q every7
              y234
              (i y234 flight8)
              (i y234 (f PAST (p on-time9)))))))))
  (i John1 (f PAST (p receive2 x4))))
```

time used = 155 msecs.

Sentence 36

```
(f (ALPHA3 (p in1 (q most2 city3)))
  (i (q several4 person5) (f PAST (p buy6 (MU tickets7)))))
```

1. The average weight is 0.92 based on 2 comparisons

```
(q most2
  y239
  (i y239 city3)
  (q several4
    y241
    (i y241 person5)
    (f (ALPHA3 (p in1 y239)) (i y241 (f PAST (p buy6 (MU tickets7))))))))
```

2. The average weight is 0.57 based on 2 comparisons

```
(q several4
```

```
  y241
  (i y241 person5)
  (q most2
    y239
    (i y239 city3)
    (f (ALPHA3 (p in1 y239)) (i y241 (f PAST (p buy6 (MU tickets7)))))))
```

time used = 120 msecs.

Sentence 37

```
(f (ALPHA3 (p in5 (q most6 city7)))
  (i (q several1 person2) (f PAST (p buy3 (MU tickets4)))))
```

1. The average weight is 0.74 based on 2 comparisons

```
(q most6
  y246
  (i y246 city7)
  (q several1
    y248
    (i y248 person2)
    (f (ALPHA3 (p in5 y246)) (i y248 (f PAST (p buy3 (MU tickets4)))))))
```

2. The average weight is 0.74 based on 2 comparisons

```
(q several1
  y248
  (i y248 person2)
  (q most6
    y246
    (i y246 city7)
    (f (ALPHA3 (p in5 y246)) (i y248 (f PAST (p buy3 (MU tickets4)))))))
```

time used = 118 msecs.

Sentence 38

```
(f (ALPHA3 (p during1 (q every2 month3)))
  (i (q many4 person5) (f PRES (p visit6 Europe7))))
```

1. The average weight is 0.93 based on 2 comparisons

```
(q every2
  y253
  (i y253 month3)
  (q many4
    y255
    (i y255 person5)
    (f (ALPHA3 (p during1 y253)) (i y255 (f PRES (p visit6 Europe7)))))))
```

2. The average weight is 0.56 based on 2 comparisons

```
(q many4
  y255
  (i y255 person5)
  (q every2
    y253
    (i y253 month3)
    (f (ALPHA3 (p during1 y253)) (i y255 (f PRES (p visit6 Europe7))))))
```

time used = 117 msecs.

Sentence 39

```
(f (ALPHA3 (p after1 (TAU2 (f PROG (f PERF (p visit2 (q several3 city4)))))))
  (i John1 (f PAST (p write5 (q each6 postcard7)))))
```

1. The average weight is 0.74 based on 3 comparisons

```
(q several3
  y263
  (i y263 city4)
  (q each6
    y267
    (i y267 postcard7)
    (f (ALPHA3 (p after1 (TAU2 (f PROG (f PERF (p visit2 y263))))))
      (i John1 (f PAST (p write5 y267))))))
```

2. The average weight is 0.73 based on 1 comparison

```
(q each6
  y267
  (i y267 postcard7)
  (f (ALPHA3
    (p after1
      (TAU2
        (q several3
          y263
          (i y263 city4)
          (f PROG (f PERF (p visit2 y263))))))
    (i John1 (f PAST (p write5 y267))))))
```

3. The average weight is 0.42 based on 3 comparisons

```
(q each6
  y267
  (i y267 postcard7)
  (q several3
    y263
    (i y263 city4)
    (f (ALPHA3 (p after1 (TAU2 (f PROG (f PERF (p visit2 y263))))))
      (i John1 (f PAST (p write5 y267)))))))
```

time used = 184 msecs.

Sentence 40

```
(f (ALPHA3 (p after1 (TAU1 (i John2 (f PAST (p visit3 (q several4 city5)))))))
  (i John1 (f PAST (p write6 (q each7 postcard8)))))
```

1. The average weight is 0.77 based on 1 comparison

```
(q each7
  y277
  (i y277 postcard8)
  (f (ALPHA3
    (p after1
      (TAU1
        (q several4
          y273
          (i y273 city5)
          (i John2 (f PAST (p visit3 y273)))))))
    (i John1 (f PAST (p write6 y277)))))
```

2. The average weight is 0.73 based on 3 comparisons

```
(q several4
  y273
  (i y273 city5)
  (q each7
    y277
    (i y277 postcard8)
    (f (ALPHA3 (p after1 (TAU1 (i John2 (f PAST (p visit3 y273))))))
      (i John1 (f PAST (p write6 y277)))))))
```

3. The average weight is 0.4 based on 3 comparisons

```
(q each7
  y277
  (i y277 postcard8)
  (q several4
    y273
    (i y273 city5)
    (f (ALPHA3 (p after1 (TAU1 (i John2 (f PAST (p visit3 y273))))))
      (i John1 (f PAST (p write6 y277)))))))
```

time used = 193 msecs.

Sentence 41.

```
(f (ALPHA2 (f PROG (f PERF (p visit1 (q each2 city3)))))
  (i (q most4 person5) (f PAST (p return6))))
```

1. The average weight is 0.8 based on 1 comparison

```
(q most4
  y284
  (i y284 person5)
  (f (ALPHA2 (q each2 y282 (i y282 city3) (f PROG (f PERF (p visit1 y282)))))
    (i y284 (f PAST (p return6)))))
```

2. The average weight is 0.59 based on 2 comparisons

```
(q each2
  y282
  (i y282 city3)
  (q most4
    y284
    (i y284 person5)
    (f (ALPHA2 (f PROG (f PERF (p visit1 y282))))
      (i y284 (f PAST (p return6))))))
```

3. The average weight is 0.1 based on 2 comparisons

```
(q most4
  y284
  (i y284 person5)
  (q each2
    y282
    (i y282 city3)
    (f (ALPHA2 (f PROG (f PERF (p visit1 y282))))
      (i y284 (f PAST (p return6))))))
```

time used = 149 msecs.

Sentence 42

```
(f (ALPHA2 (l x1 (i Jones2 (f PROG (f PERF (f PASV (p brief3 x1)))))))
  (i (q each4 volunteer5) (f PAST (p leave6))))
```

1. The average weight is 1.0 based on 0 comparisons

```
(q each4
  y295
  (i y295 volunteer5)
  (f (ALPHA2 (l x1 (i Jones2 (f PROG (f PERF (f PASV (p brief3 x1)))))))
    (i y295 (f PAST (p leave6)))))
```

time used = 107 msecs.

Sentence 43

```
(i (i (q each1 girl2) (f FUT (p arrive3)))
  if-then4
  (i (q some5 boy6) (f FUT (p happy7))))
```

1. The average weight is 0.85 based on 2 comparisons

```
(i (q each1 y300 (i y300 girl2) (i y300 (f FUT (p arrive3))))
  if-then4
  (q some5 y304 (i y304 boy6) (i y304 (f FUT (p happy7)))))
```

2. The average weight is 0.65 based on 2 comparisons

```
(q some5
  y304
  (i y304 boy6)
  (i (q each1 y300 (i y300 girl2) (i y300 (f FUT (p arrive3))))
    if-then4
    (i y304 (f FUT (p happy7)))))
```

time used = 193 msecs.

Sentence 44

```
(i (i (q each5 girl6) (f FUT (p arrive7)))
  if-then4
  (i (q some1 boy2) (f FUT (p happy3))))
```

1. The average weight is 0.7 based on 2 comparisons

```
(i (q each5 y309 (i y309 girl6) (i y309 (f FUT (p arrive7))))
  if-then4
  (q some1 y313 (i y313 boy2) (i y313 (f FUT (p happy3)))))
```

2. The average weight is 0.7 based on 2 comparisons

```
(q some1
  y313
  (i y313 boy2)
  (i (q each5 y309 (i y309 girl6) (i y309 (f FUT (p arrive7))))
    if-then4
    (i y313 (f FUT (p happy3)))))
```

3. The average weight is 0.46 based on 3 comparisons

```
(q some1
  y313
  (i y313 boy2)
  (q each5
    y309
    (i y309 girl6)
    (i (i y309 (f FUT (p arrive7))) if-then4 (i y313 (f FUT (p happy3))))))
```

4. The average weight is 0.29 based on 2 comparisons

```
(q each5
  y309
```

(i y309 girl6)
(i (i y309 (f FUT (p arrive7)))
if-then4
(q some1 y313 (i y313 boy2) (i y313 (f FUT (p happy3))))))

5. The average weight is 0.26 based on 3 comparisons

(q each5
y309
(i y309 girl6)
(q some1
y313
(i y313 boy2)
(i (i y309 (f FUT (p arrive7))) if-then4 (i y313 (f FUT (p happy3))))))

time used = 180 msecs.

Sentence 45

(i (i (q some1 girl2) (f FUT (p arrive3)))
if-then4
(i (q each5 boy6) (f FUT (p happy7))))

1. The average weight is 0.84 based on 2 comparisons

(i (q some1 y318 (i y318 girl2) (i y318 (f FUT (p arrive3))))
if-then4
(q each5 y322 (i y322 boy6) (i y322 (f FUT (p happy7)))))

2. The average weight is 0.64 based on 2 comparisons

(q some1
y318
(i y318 girl2)
(i (i y318 (f FUT (p arrive3)))
if-then4
(q each5 y322 (i y322 boy6) (i y322 (f FUT (p happy7))))))

3. The average weight is 0.37 based on 3 comparisons

(q some1
y318
(i y318 girl2)
(q each5
y322
(i y322 boy6)
(i (i y318 (f FUT (p arrive3))) if-then4 (i y322 (f FUT (p happy7))))))

4. The average weight is 0.35 based on 2 comparisons

(q each5
y322
(i y322 boy6)

```
(i (q some1 y318 (i y318 girl2) (i y318 (f FUT (p arrive3))))
  if-then4
  (i y322 (f FUT (p happy7))))))
```

5. The average weight is 0.17 based on 3 comparisons

```
(q each5
  y322
  (i y322 boy6)
  (q some1
    y318
    (i y318 girl2)
    (i (i y318 (f FUT (p arrive3))) if-then4 (i y322 (f FUT (p happy7))))))
```

time used = 186 msecs.

Sentence 46

```
(i (i (q some5 girl6) (f FUT (p arrive7)))
  if-then4
  (i (q each1 boy2) (f FUT (p happy3))))
```

1. The average weight is 0.7 based on 2 comparisons

```
(i (q some5 y327 (i y327 girl6) (i y327 (f FUT (p arrive7))))
  if-then4
  (q each1 y331 (i y331 boy2) (i y331 (f FUT (p happy3)))))
```

2. The average weight is 0.5 based on 2 comparisons

```
(q each1
  y331
  (i y331 boy2)
  (i (q some5 y327 (i y327 girl6) (i y327 (f FUT (p arrive7))))
    if-then4
    (i y331 (f FUT (p happy3)))))
```

3. The average weight is 0.5 based on 2 comparisons

```
(q some5
  y327
  (i y327 girl6)
  (i (i y327 (f FUT (p arrive7)))
    if-then4
    (q each1 y331 (i y331 boy2) (i y331 (f FUT (p happy3))))))
```

4. The average weight is 0.46 based on 3 comparisons

```
(q each1
  y331
  (i y331 boy2)
  (q some5
    y327
```

(i y327 girl6)
(i (i y327 (f FUT (p arrive7))) if-then4 (i y331 (f FUT (p happy3))))))

5. The average weight is 0.26 based on 3 comparisons

(q some5
y327
(i y327 girl6)
(q each1
y331
(i y331 [illegible]2)
(i (i y[illegible] FUT (p arrive7))) if-then4 (i y331 (f FUT (p happy3))))))

time used = 179 msecs.

Sentence 47.

(i (q every1 boy2)
(f PRES (p want3 (TAU2 (f INF (p (c or5 kiss4 hug6) Mary7))))))

1. The average weight is 0.7 based on 4 comparison

(q every1
y335
(i y335 boy2)
(i y335
(f PRES
(p want3
(TAU2
(i y340
(i (i y340 (f INF (p kiss4 Mary7)))
or5
(i y340 (f INF (p hug6 Mary7))))))))))

2. The average weight is 0.5 based on 2 comparisons

(q every1
y335
(i y335 boy2)
(i (i y335 (f PRES (p want3 (TAU2 (f INF (p kiss4 Mary7)))))))
or5
(i y335 (f PRES (p want3 (TAU2 (f INF (p hug6 Mary7))))))))

3. The average weight is 0.3 based on 2 comparisons

(i (q every1
y335
(i y335 boy2)
(i y335 (f PRES (p want3 (TAU2 (f INF (p kiss4 Mary7))))))))
or5
(q every1
y335
(i y335 boy2)

```
(i y335 (f PRES (p want3 (TAU2 (f INF (p hug6 Mary7))))))))
```

time used = 318 msecs.

Sentence 48

```
(i John1
  (f PRES
    (p (c and3 want2 hope4) (TAU2 (f INF (p (c or6 kiss5 hug7) Mary8)))))))
```

1. The average weight is 0.7 based on 1 comparison

```
(i (i John1
     (f PRES
       (p want2
         (TAU2
           (l y346
             (i (i y346 (f INF (p kiss5 Mary8)))
               or6
               (i y346 (f INF (p hug7 Mary8))))))))))
   and3
   (i John1
     (f PRES
       (p hope4
         (TAU2
           (l y346
             (i (i y346 (f INF (p kiss5 Mary8)))
               or6
               (i y346 (f INF (p hug7 Mary8))))))))))
```

2. The average weight is 0.5 based on 2 comparisons

```
(i (i (i John1 (f PRES (p want2 (TAU2 (f INF (p kiss5 Mary8))))))
     or6
     (i John1 (f PRES (p want2 (TAU2 (f INF (p hug7 Mary8)))))))
   and3
   (i (i John1 (f PRES (p hope4 (TAU2 (f INF (p kiss5 Mary8))))))
     or6
     (i John1 (f PRES (p hope4 (TAU2 (f INF (p hug7 Mary8)))))))))
```

3. The average weight is 0.3 based on 2 comparisons

```
(i (i (i John1 (f PRES (p want2 (TAU2 (f INF (p kiss5 Mary8))))))
     and3
     (i John1 (f PRES (p hope4 (TAU2 (f INF (p kiss5 Mary8)))))))
   or6
   (i (i John1 (f PRES (p want2 (TAU2 (f INF (p hug7 Mary8))))))
     and3
     (i John1 (f PRES (p hope4 (TAU2 (f INF (p hug7 Mary8)))))))))
```

time used = 365 msecs.

Sentence 49

(i (TAU2 (f INF (p (c and2 kiss1 hug3) Mary4))) (f PRES (p nice5)))

1. The average weight is 0.9 based on 1 comparison

```
(i (TAU2
   (l y350
      (i (i y350 (f INF (p kiss1 Mary4)))
         and2
         (i y350 (f INF (p hug3 Mary4))))))
   (f PRES (p nice5)))
```

2. The average weight is 0.09 based on 1 comparison

```
(i (i (TAU2 (f INF (p kiss1 Mary4))) (f PRES (p nice5)))
   and2
   (i (TAU2 (f INF (p hug3 Mary4))) (f PRES (p nice5))))
```

time used = 133 msecs

Sentence 50

```
(i (q two1 boy2)
   (c or6
      (f PAST (p kiss3 (q every4 girl5)))
      (f PAST (p hug7 (q every8 girl9)))))
```

1. The average weight is 0.79 based on 4 comparisons

```
(i (q two1
      y354
      (i y354 boy2)
      (q every4 y358 (i y358 girl5) (i y354 (f PAST (p kiss3 y358)))))
   or6
   (q two1
      y354
      (i y354 boy2)
      (q every8 y361 (i y361 girl9) (i y354 (f PAST (p hug7 y361)))))))
```

2. The average weight is 0.75 based on 4 comparisons

```
(i (q every4
      y358
      (i y358 girl5)
      (q two1 y354 (i y354 boy2) (i y354 (f PAST (p kiss3 y358)))))
   or6
   (q two1
      y354
      (i y354 boy2)
      (q every8 y361 (i y361 girl9) (i y354 (f PAST (p hug7 y361)))))))
```

3. The average weight is 0.71 based on 4 comparisons

```
(i (q every4
     y358
     (i y358 girl5)
     (q two1 y354 (i y354 boy2) (i y354 (f PAST (p kiss3 y358)))))
   or6
   (q every8
     y361
     (i y361 girl9)
     (q two1 y354 (i y354 boy2) (i y354 (f PAST (p hug7 y361))))))
```

time used = 391 msecs.

Sentence 51

```
(i Mary1
  (c or7
    (f PAST (p read2 (q some3 story4) (q each6 child6)))
    (f PAST (p tell8 (q some9 story10) (q each11 child12)))))
```

1. The average weight is 0.82 based on 4 comparisons

```
(i (q each5
     y367
     (i y367 child6)
     (q some3 y366 (i y366 story4) (i Mary1 (f PAST (p read2 y366 y367)))))
   or7
   (q each11
     y371
     (i y371 child12)
     (q some9 y370 (i y370 story10) (i Mary1 (f PAST (p tell8 y370 y371))))))
```

2. The average weight is 0.75 based on 4 comparisons

```
(i (q some3
     y366
     (i y366 story4)
     (q each5 y367 (i y367 child6) (i Mary1 (f PAST (p read2 y366 y367)))))
   or7
   (q each11
     y371
     (i y371 child12)
     (q some9 y370 (i y370 story10) (i Mary1 (f PAST (p tell8 y370 y371))))))
```

3. The average weight is 0.75 based on 4 comparisons

```
(i (q each5
     y367
     (i y367 child6)
     (q some3 y366 (i y366 story4) (i Mary1 (f PAST (p read2 y366 y367)))))
   or7
   (q some9
```

```
    y370
    (i y370 story10)
    (q each11 y371 (i y371 child12) (i Mary1 (f PAST (p tell8 y370 y371))))))
```

1. The average weight is 0.67 based on 4 comparisons

```
(i (q some3
     y366
     (i y366 story4)
     (q each5 y367 (i y367 child6) (i Mary1 (f PAST (p read2 y366 y367)))))
   or7
   (q some9
     y370
     (i y870 story10)
     (q each11 y371 (i y371 child12) (i Mary1 (f PAST (p tell8 y370 y371))))))
```

time used = 814 msecs.

Sentence 52

```
(i (c and3 John1 Bill2 Sam4) (f PRES (p love5 (c or7 Sue6 Mary7))))
```

1. The average weight is 0.7 based on 1 comparison

```
(i (i (i John1 (f PRES (p love5 Sue6))) or7 (i John1 (f PRES (p love5 Mary7))))
   and3
   (i (i Bill2 (f PRES (p love5 Sue6))) or7 (i Bill2 (f PRES (p love5 Mary7))))
   (i (i Sam4 (f PRES (p love5 Sue6))) or7 (i Sam4 (f PRES (p love5 Mary7)))))
```

2. The average weight is 0.3 based on 1 comparison

```
(i (i (i John1 (f PRES (p love5 Sue6)))
     and3
     (i Bill2 (f PRES (p love5 Sue6)))
     (i Sam4 (f PRES (p love5 Sue6))))
   or7
   (i (i John1 (f PRES (p love5 Mary7)))
     and3
     (i Bill2 (f PRES (p love5 Mary7)))
     (i Sam4 (f PRES (p love5 Mary7)))))
```

time used = 241 msecs.

Sentence 53

```
(i John1 (f PRES (p (c and3 love2 admire4) (c or6 Fido5 Kim7))))
```

1. The average weight is 0.7 based on 1 comparison

```
(i (i (i John1 (f PRES (p love2 Fido5))) or6 (i John1 (f PRES (p love2 Kim7))))
   and3
```

(i (i John1 (f PRES (p admire4 Fido5)))
or6
(i John1 (f PRES (p admire4 Kim7)))))

2. The average weight is 0.3 based on 1 comparison

(i (i (i John1 (f PRES (p love2 Fido5)))
and3
(i John1 (f PRES (p admire4 Fido5))))
or6
(i (i John1 (f PRES (p love2 Kim7)))
and3
(i John1 (f PRES (p admire4 Kim7)))))

time used = 204 msecs.

Sentence 54

(i (q all1 man2)
(f PRES (p want3 (TAU2 (f INF (p marry4 (c or6 Peggy5 Sue7)))))))

1. The average weight is 0.7 based on 1 comparison

(q all1
y379
(i y379 man2)
(*y379
(f PRES
(p want3
(TAU2
(l y384
(i (i y384 (f INF (p marry4 Peggy5)))
or6
(i y384 (f INF (p marry4 Sue7))))))))))

2. The average weight is 0.5 based on 2 comparisons

(q all1
y379
(i y379 man2)
(i (i y379 (f PRES (p want3 (TAU2 (f INF (p marry4 Peggy5))))))
or6
(i y379 (f PRES (p want3 (TAU2 (f INF (p marry4 Sue7)))))))))

3. The average weight is 0.3 based on 2 comparisons

(i (q all1
y379
(i y379 man2)
(i y379 (f PRES (p want3 (TAU2 (f INF (p marry4 Peggy5)))))))
or6
(q all1
y379

(i y379 man2)
(i y379 (f PRES (p want3 (TAU2 (f INF (p marry4 Sue7))))))))

time used = 240 msecs.

Sentence 55

(i (c or3 (q every1 boy2) (q every4 girl5)) (f PAST (p go6 (q some7 movie8))))

1. The average weight is 0.85 based on 4 comparisons

(i (q every1
y386
(i y386 boy2)
(q some7 y390 (i y390 movie8) (i y386 (f PAST (p go6 y390)))))
or3
(q every4
y387
(i y387 girl5)
(q some7 y390 (i y390 movie8) (i y387 (f PAST (p go6 y890))))))

2. The average weight is 0.75 based on 4 comparisons.

(i (q every1
y386
(i y386 boy2)
(q some7 y390 (i y390 movie8) (i y386 (f PAST (p go6 y390)))))
or3
(q some7
y390
(i y390 movie8)
(q every4 y387 (i y387 girl5) (i y387 (f PAST (p go6 y390))))))

3. The average weight is 0.65 based on 4 comparisons

(i (q some7
y390
(i y390 movie8)
(q every1 y386 (i y386 boy2) (i y386 (f PAST (p go6 y390)))))
or3
(q some7
y390
(i y390 movie8)
(q every4 y387 (i y387 girl5) (i y387 (f PAST (p go6 y390))))))

time used = 440 msecs.

Sentence 56

(i (q every1 (l x2 (i (i x2 person3) & (i x2 in4 (c or6 Zermatt5 Murren7)))))
(f PRES (p ski8)))

1. The average weight is 0.7 based on 1 comparison

```
(q every1
  x2
  (i (i x2 person3) & (i (i x2 in4 Zermatt5) or6 (i x2 in4 Murren7)))
  (i x2 (f PRES (p ski8))))
```

2. The average weight is 0.3 based on 1 comparison

```
(i (q every1
     x2
     (i (i x2 person3) & (i x2 in4 Zermatt5))
     (i x2 (f PRES (p ski8))))
   or6
   (q every1
     x2
     (i (i x2 person3) & (i x2 in4 Murren7))
     (i x2 (f PRES (p ski8)))))
```

time used = 167 msecs.

Sentence 57

```
(i (q every1 (l x2 (i (i x2 (c or4 boy3 girl5)) & (i x2 in6 Nice7))))
  (f PRES (p swim8)))
```

1. The average weight is 0.8 based on 1 comparison

```
(q every1
  x2
  (i (i (i x2 boy3) or4 (i x2 girl5)) & (i x2 in6 Nice7))
  (f x2 (f PRES (p swim8))))
```

2. The average weight is 0.19 based on 1 comparison

```
(i (q every1 x2 (i (i x2 boy3) & (i x2 in6 Nice7)) (i x2 (f PRES (p swim8))))
  or4
  (q every1 x2 (i (i x2 girl5) & (i x2 in6 Nice7)) (i x2 (f PRES (p swim8)))))
```

time used = 172 msecs.

Sentence 58

```
(i (q every1
     (l x2
       (c or6
         (i (i x2 boy3) & (i x2 in4 Nice5))
         (i (i x2 girl7) & (i x2 in8 Monaco9)))))
  (f PRES (p swim10)))
```

1. The average weight is 0.7 based on 1 comparison

```
(q every1
  x2
  (i (i (i x2 boy3) & (i x2 in4 Nice5))
    or6
    (i (i x2 girl7) & (i x2 in8 Monaco9)))
  (i x2 (f PRES (p swim10))))
```

2. The average weight is 0.3 based on 1 comparison

```
(i (q every1 x2 (i (i x2 boy3) & (i x2 in4 Nice5)) (i x2 (f PRES (p swim10))))
  or6
  (q every1
    x2
    (i (i x2 girl7) & (i x2 in8 Monaco9))
    (i x2 (f PRES (p swim10)))))
```

time used = 191 msecs.

Sentence 69

```
(i (q every1
    (c or7
      (i x2 (i (i x2 man3) & (i x2 on4 (q most5 bus6))))
      (i x2 (i (i x2 woman8) & (i x2 on9 (q most10 train11)))))))
  (f PAST (p arrive12)))
```

1. The average weight is 0.66 based on 5 comparisons

```
(i (q most5
    y422
    (i y422 bus6)
    (q every1
      x2
      (i (i x2 man3) & (i x2 on4 y422))
      (i x2 (f PAST (p arrive12)))))
  or7
  (q most10
    y427
    (i y427 train11)
    (q every1
      x2
      (i (i x2 woman8) & (i x2 on9 y427))
      (i x2 (f PAST (p arrive12))))))
```

2. The average weight is 0.57 based on 4 comparisons

```
(i (q every1
    x2
    (i (i x2 man3) & (q most5 y422 (i y422 bus6) (i x2 on4 y422)))
    (i x2 (f PAST (p arrive12))))
  or7
```

```
(q most10
  y427
  (i y427 train11)
  (q every1
    x2
    (i (i x2 woman8) & (i x2 on9 y427))
    (i x2 (f PAST (p arrive12))))))
```

3. The average weight is 0.57 based on 4 comparisons

```
(i (q most5
    y422
    (i y422 bus6)
    (q every1
      x2
      (i (i x2 man3) & (i x2 on4 y422))
      (i x2 (f PAST (p arrive12)))))
  or7
  (q every1
    x2
    (i (i x2 woman8) & (q most10 y427 (i y427 train11) (i x2 on9 y427)))
    (i x2 (f PAST (p arrive12)))))
```

4. The average weight is 0.56 based on 3 comparisons

```
(q every1
  x2
  (i (i (i x2 man3) & (q most5 y422 (i y422 bus6) (i x2 on4 y422)))
    or7
    (i (i x2 woman8) & (q most10 y427 (i y427 train11) (i x2 on9 y427))))
  (i x2 (f PAST (p arrive12))))
```

5. The average weight is 0.43 based on 3 comparisons

```
(i (q every1
    x2
    (i (i x2 man3) & (q most5 y422 (i y422 bus6) (i x2 on4 y422)))
    (i x2 (f PAST (p arrive12
  or7
  (q every1
    x2
    (i (i x2 woman8) & (q most10 y427 (i y427 train11) (i x2 on9 y427)))
    (i x2 (f PAST (p arrive12)))))
```

time used = 633 msecs.

Sentence 60

```
(i (q some1 boy2)
  (f PRES (p want3 (TAU2 (f INF (p buy4 (q two5 tie6) (q some1 boy2)))))))
```

1. The average weight is 0.68 based on 1 comparison

```
(q some1
  y431
  (i y431 boy2)
  (i y431
    (f PRES
      (p want3
        (TAU2 (q two5 y436 (i y436 tie6) (f INF (p buy4 y436 y431))))))))
```

2. The average weight is 0.31 based on 1 comparison

```
(q some1
  y431
  (i y431 boy2)
  (q two5
    y436
    (i y436 tie6)
    (i y431 (f PRES (p want3 (TAU2 (f INF (p buy4 y436 y431))))))))
```

time used = 134 msecs.

Sentence 61

```
(i (q some1 (c or3 boy2 man4)) (f PAST (p shave5 (q some1 (c or3 boy2 man4)))))
```

1. The average weight is 0.7 based on 1 comparison

```
(q some1
  y441
  (i (i y441 boy2) or3 (i y441 man4))
  (i y441 (f PAST (p shave5 y441))))
```

2. The average weight is 0.3 based on 1 comparison

```
(i (q some1 y441 (i y441 boy2) (i y441 (f PAST (p shave5 y441))))
  or3
  (q some1 y441 (i y441 man4) (i y441 (f PAST (p shave5 y441)))))
```

time used = 135 msecs.

Sentence 62

```
(i (q some1 (l x2 (i (i x2 man3) & (i x2 on4 (q each5 trip6)))))
  (f PAST
    (p shave7
      (q some1 (l x2 (i (i x2 man3) & (i x2 on4 (q each5 trip6))))))))
```

1. The average weight is 0.7 based on 1 comparison

```
(q each5
  y455
  (i y455 trip6)
```

```
(q some1
  x2
  (i (i x2 man3) & (i x2 on4 y455))
  (i x2 (f PAST (p shave7 x2)))))
```

2. The average weight is 0.3 based on 1 comparison

```
(q some1
  x2
  (i (i x2 man3) & (q each5 y455 (i y455 trip6) (i x2 on4 y455)))
  (i x2 (f PAST (p shave7 x2))))
```

time used = 255 msecs.

Sentence 63

```
(i (q some1 boy2)
  (c or4
    (f PAST (p kiss3 (q each6 girl7)))
    (f PAST (p hug5 (q each6 girl7)))))
```

1. The average weight is 0.8 based on 2 comparisons

```
(i (q each6
     y468
     (i y468 girl7)
     (q some1 y464 (i y464 boy2) (i y464 (f PAST (p kiss3 y468)))))
   or4
   (q each6
     y468
     (i y468 girl7)
     (q some1 y464 (i y464 boy2) (i y464 (f PAST (p hug5 y468))))))
```

2. The average weight is 0.7 based on 2 comparisons

```
(i (q some1
     y464
     (i y464 boy2)
     (q each6 y468 (i y468 girl7) (i y464 (f PAST (p kiss3 y468)))))
   or4
   (q some1
     y464
     (i y464 boy2)
     (q each6 y468 (i y468 girl7) (i y464 (f PAST (p hug5 y468))))))
```

time used = 264 msecs.

Sentence 64

```
(i Mary1
  (c or4
```

```
    (f PAST (p read3 (q some6 story7) (q each8 child9)))
    (f PAST (p tell5 (q some6 story7) (q each8 child9)))))
```

1. The average weight is 0.82 based on 2 comparisons

```
(i (q each8
     y488
     (i y488 child9)
     (q some6 y486 (i y486 story7) (i Mary1 (f PAST (p read3 y486 y488)))))
   or4
   (q each8
     y488
     (i y488 child9)
     (q some6 y486 (i y486 story7) (i Mary1 (f PAST (p tell5 y486 y488))))))
```

2. The average weight is 0.67 based on 2 comparisons

```
(i (q some6
     y486
     (i y486 story7)
     (q each8 y488 (i y488 child9) (i Mary1 (f PAST (p read3 y486 y488)))))
   or4
   (q some6
     y486
     (i y486 story7)
     (q each8 y488 (i y488 child9) (i Mary1 (f PAST (p tell5 y486 y488))))))
```

time used = 436 msecs.

Sentence 65

```
(i (q few1 girl2)
  (c or4
    (f PAST (p read3 (q some6 story7) (q each8 child9)))
    (f PAST (p tell5 (q some6 story7) (q each8 child9)))))
```

1. The average weight is 0.78 based on 3 comparisons

```
(q few1
  y516
  (i y516 girl2)
  (i (q each8
       y522
       (i y522 child9)
       (q some6 y520 (i y520 story7) (i y516 (f PAST (p read3 y520 y522)))))
     or4
     (q each8
       y522
       (i y522 child9)
       (q some6
         y520
         (i y520 story7)
         (i y516 (f PAST (p tell5 y520 y522)))))))
```

2. The average weight is 0.68 based on 3 comparisons

```
(q some6
  y520
  (i y520 story7)
  (q few1
    y516
    (i y516 girl2)
    (i (q each8 y522 (i y522 child9) (i y516 (f PAST (p read3 y520 y522))))
      or4
      (q each8
        y522
        (i y522 child9)
        (i y516 (f PAST (p tell5 y520 y522)))))))
```

3. The average weight is 0.66 based on 5 comparisons

```
(q few1
  y516
  (i y516 girl2)
  (i (q some6
       y520
       (i y520 story7)
       (q each8 y522 (i y522 child9) (i y516 (f PAST (p read3 y520 y522)))))
     or4
     (q some6
       y520
       (i y520 story7)
       (q each8
         y522
         (i y522 child9)
         (i y516 (f PAST (p tell5 y520 y522)))))))
```

4. The average weight is 0.65 based on 3 comparisons

```
(i (q few1
     y516
     (i y516 girl2)
     (q each8
       y522
       (i y522 child9)
       (q some6 y520 (i y520 story7) (i y516 (f PAST (p read3 y520 y522))))))
   or4
   (q few1
     y516
     (i y516 girl2)
     (q each8
       y522
       (i y522 child9)
       (q some6
         y520
         (i y520 story7)
         (i y516 (f PAST (p tell5 y520 y522)))))))
```

5. The average weight is 0.65 based on 3 comparisons

```
(i (q each8
     y522
     (i y522 child9)
     (q few1
       y516
       (i y516 girl2)
       (q some6 y520 (i y520 story7) (i y516 (f PAST (p read3 y520 y522))))))
  or4
  (q each8
    y522
    (i y522 child9)
    (q few1
      y516
      (i y516 girl2)
      (q some6
        y520
        (i y520 story7)
        (i y516 (f PAST (p tell5 y520 y522)))))))
```

6. The average weight is 0.65 based on 3 comparisons

```
(i (q each8
     y522
     (i y522 child9)
     (q some6
       y520
       (i y520 story7)
       (q few1 y516 (i y516 girl2) (i y516 (f PAST (p read3 y520 y522))))))
  or4
  (q each8
    y522
    (i y522 child9)
    (q some6
      y520
      (i y520 story7)
      (q few1 y516 (i y516 girl2) (i y516 (f PAST (p tell5 y520 y522)))))))
```

7. The average weight is 0.58 based on 5 comparisons

```
(i (q few1
     y516
     (i y516 girl2)
     (q some6
       y520
       (i y520 story7)
       (q each8 y522 (i y522 child9) (i y516 (f PAST (p read3 y520 y522))))))
  or4
  (q few1
    y516
    (i y516 girl2)
    (q some6
      y520
      (i y520 story7)
```

```
      (q each8
        y522
        (i y522 child9)
        (i y516 (f PAST (p tell5 y520 y522)))))))
```

8. The average weight is 0.48 based on 5 comparisons

```
(i (q some6
     y520
     (i y520 story7)
     (q few1
       y516
       (i y516 girl2)
       (q each8 y522 (i y522 child9) (i y516 (f PAST (p read3 y520 y522))))))
   or4
   (q some6
     y520
     (i y520 story7)
     (q few1
       y516
       (i y516 girl2)
       (q each8
         y522
         (i y522 child9)
         (i y516 (f PAST (p tell5 y520 y522)))))))
```

9. The average weight is 0.47 based on 5 comparisons

```
(i (q some6
     y520
     (i y520 story7)
     (q each8
       y522
       (i y522 child9)
       (q few1 y516 (i y516 girl2) (i y516 (f PAST (p read3 y520 y522))))))
   or4
   (q some6
     y520
     (i y520 story7)
     (q each8
       y522
       (i y522 child9)
       (q few1 y516 (i y516 girl2) (i y516 (f PAST (p tell5 y520 y522)))))))
```

time used = 1186 msecs.

Appendix IV

Scoping Program

March, 1987

```
; Introduction:
;   The program reads in a sequence of initial logical translations of
; English sentences and for each generates the set of valid scoped
; readings, arranged in an approximate order of preference.
; The program is written in Franz Lisp.
;
; Input:
;   The input formulas are read in from the file "in". Heuristic
; information needed for scoping is stored in the files "weights",
; "relations" and "ratios".
;
; Output:
;   The output goes to the file "out". For each input formula the program
; echoes the input then prints a list of scoped readings preceeded by the
; average scoping weight for the reading and the number of comparisons on
; which this is based.
;
; Summary:
;   The program is divided into ten parts:
;
;   1. Input and output
;   2. Definitions and auxiliary functions
;   3. Modification of logical form
;   4. Redundancy and scoping weights
;   5. Horizontal scoping
;   6. Vertical scoping
;   7. Determination of structural categories
;   8. Scoping of terms
;   9. Scoping of phrases
;  10. Main section
```

```
;-----------------------------------------------------------------------
;                PART 1: INPUT AND OUTPUT
;
; The first three functions read in heuristic data from the files "weights",
; "relations" and "ratios". The last two functions are used for output.
;-----------------------------------------------------------------------

(defun store-weights ()
 (prog (srelation optype)
  (setq inport (infile 'weights))
  read-loop
   (setq srelation (read inport))
   (cond ((null srelation) (return nil)) (t nil))
   (setq optype (read inport))
   (cond ((equal optype 'all) (put srelation 'all (read inport)))
         (t (mapc
              '(lambda (op)
                 (put srelation op (read inport)))
             '(some u-some each most few no or and))))
   (go read-loop)))

(defun store-relations ()
 (prog (category1 category2)
  (setq inport (infile 'relations))
  (read inport)
  category1-loop
   (setq category1 (read inport))
   category2-loop
      (setq category2 (read inport))
      (cond ((null category2) (return nil))
            ((equal category2 '*) (go category1-loop))
            (t (progn (put category1 category2 (read inport))
                      (go category2-loop))) )))

(defun store-ratios ()
 (prog (op srelation)
  (setq inport (infile 'ratios))
  (read inport)
  op-loop
   (setq op (read inport))
   (put op 'standard (read inport))
   (put op 'ratio (read inport))
   relation-loop
      (setq srelation (read inport))
      (cond ((null srelation) (return nil))
            ((equal srelation '*) (go op-loop))
            (t (progn (put op srelation (read inport))
                      (go relation-loop)))) ))

; "Printr" prints a scoped reading preceeded by the average
; scoping weight and the number of comparisons on which the
; average is based.

(defun printr (r)
 (progn (terpri outport)
```

```
        (patom '||outport)
           (print (setq $n (add1 $n)) outport)
           (patom '|. |outport)
        (patom '|The average weight is |outport)
        (print (truncate (caar r)) outport)
        (patom '|based on |outport)
        (print (cadar r) outport)
        (patom '|comparison|outport)
        (cond ((equal (cadar r) 1) (patom '||outport))
              (t (patom '|s |outport)))
        (terpri outport) (terpri outport)
           (pp-form (cadr r) outport)
           (terpri outport)))

(defun print-line (n outport)
  (cond ((equal n 0) (terpri outport))
        (t (progn (patom '|_|outport)
                  (print-line (sub1 n) outport))) ))
```

```
;-----------------------------------------------------------------------
;                PART 2: DEFINITIONS AND AUXILIARY FUNCTIONS
;-----------------------------------------------------------------------
; Some definitions follow. Numerals and numeral partitives are given
; the same scoping heuristics. The parameter "$min-weight" is the minimal
; acceptable scoping weight for the comparison of a pair of operators.
; "Remove-op" is used to remove an operator from a list of operators.
; "Suffix" returns the suffix of an expression; if the expression is a phrase,
; the suffix of the main predicate is returned. "Store-digits" is needed to
; convert suffix characters into numbers. "Store-words" splits a word into
; the suffix and the remaining word and stores both on the property list of
; the word. "Sort is called by "main" to sort the list of readings of a
; sentence by the average scoping weight (which is obtained by dividing the
; total weight by the number of comparisons made).
;-----------------------------------------------------------------------
(setq $numerals '(two three four five six seven eight nine ten))
(setq $num-partitives '(u-two u-three u-four u-five u-six
                u-seven u-eight u-nine u-ten))
(setq $tensed-mods '(PRES PAST FUT))
(setq $quant-advls '(sometimes always often usually seldom never))
(setq $existential-quants '(some num several many))
(setq $universal-quants '(each every all u-some u-num u-several u-many))
(setq $optimal-weight 1.0)
(setq $min-weight 0.01)

(defun put (node attrib value)
 (putprop node value attrib))

(defun ave (a b) (quotient (add a b) 2))

(defun exchange (a b)
 (setq $temp a a b b $temp))

(defun remove-op (op oplist newlist)
 (cond ((null oplist) newlist)
       ((equal op (car oplist)) (remove-op op (cdr oplist) newlist))
       (t (remove-op op (cdr oplist) (append1 newlist (car oplist))))))

(defun remove-ops (inner-ops ops)
 (cond ((null inner-ops) ops)
       (t (remove-ops (cdr inner-ops)
                (remove-op (car inner-ops) ops nil))) ))


(defun trim-zeroes (l)
 (cond ((equal (car l) '0) (trim-zeroes (cdr l))) (t l)))

(defun formsym (char)
 (implode
  (cons char (trim-zeroes (cdr (explode (gensym char))))) ))

(defun preposed? (phrase1 phrase2)
 (lessp (suffix phrase1) (suffix phrase2)))

(defun untensed? (phrase)
```

```
  (cond ((atom phrase) t)
        ((equal (car phrase) 'f)
          (cond ((member (cadr phrase) Stensed-mods) nil)
                (t (untensed? (caddr phrase))) ))
        (t t)))

(defun suffix (expr)
 (cond ((atom expr) (get expr 'suffix))
       ((member (car expr) '(q c p))
         (cond ((atom (cadr expr)) (get (cadr expr) 'suffix))
               (t (suffix (cadr expr))) ))
       (t (suffix (car (last expr)))) ))


(defun word (op)
 (cond ((null op) nil)
       ((atom op) (get op 'word))
       ((atom (cadr op)) (get (cadr op) 'word))
       (t nil)))

(defun get-pred (phrase)
 (cond ((atom expr) expr)
       ((equal (car expr) 'p)
         (cond ((atom (cadr expr)) (cadr expr))
               (t (get-pred (cadr expr))) ))
       (t (get-pred (car (last expr)))) ))

(defun digit (char)
 (get char 'digit))

(defun truncate (num)
 (quotient (float (fix (times num 100.0))) 100.0))

(defun store-digits ()
 (mapc
  '(lambda (pair)
    (put (car pair) 'digit (cadr pair)))
  '((|1| 1) (|2| 2) (|3| 3) (|4| 4) (|5| 5)
    (|6| 6) (|7| 7) (|8| 8) (|9| 9) (|0| 0)) ))


(defun number (revl)
 (cond ((null revl) 0)
       (t (add (digit (car revl)) (times 10 (number (cdr revl)))) )))

(defun split-word (wordlist word)
 (cond ((null wordlist) nil)
       ((digit (car wordlist))
         (list (implode word) (number (reverse wordlist))))
       (t (split-word (cdr wordlist) (append1 word (car wordlist)))) ))

(defun store-words (phrase)
 (progn
  (cond ((not (atom (car phrase))) (store-words (car phrase)))
        ((member (car phrase) '(q c l p i f)) nil)
```

```
        (t (progn
            (setq spl (split-word (explode (car phrase)) nil))
            (cond ((null spl) nil)
              (t (progn
                  (put (car phrase) 'word
                    (cond
                      ((member (car spl) $numerals) 'num)
                      ((member (car spl) $num-partitives) 'u-num)
                      (t (car spl)) ))
                  (put (car phrase) 'suffix (cadr spl))))) )))
   (cond ((null (cdr phrase)) nil)
         (t (store-words (cdr phrase)))) ))

(defun insert-sort (r slist)
  (cond ((null slist) (list r))
        ((greaterp (caar r) (caaar slist)) (cons r slist))
        (t (cons (car slist) (insert-sort r (cdr slist)))) ))

(defun sort (slist rlist)
  (cond ((null rlist) slist)
        (t (sort
            (insert-sort
              (rplaca (car rlist)
                (list
                  (cond ((equal (caaar rlist) 0) 1.0)
                        (t (quotient (caaar rlist) (cadaar rlist))))
                  (cadaar rlist)))
              slist)
            (cdr rlist))) ))
```

```
;-----------------------------------------------------------------------
;        PART 3: MODIFICATION OF LOGICAL FORMS
;
;  The following functions are used to modify logical forms.
; "Embed" is called by "expand" to apply trapped operators
; to the phrase which they will embed. "Embed-op" performs the
; application for each operator; if the operator is a
; coordinator "add-coord" is called to perform the application.
; "Embed-quant" is called by "scope-quant" to embed the scoped
; restriction clause inside the head determiner and variable.
; "Embed-subj" is called by "scope-infix".
; "Form-list" is used throughout the program to initialize a
; list of readings.
; The remaining functions are used to apply a coordinator to a
; phrase during vertical scoping.
;-----------------------------------------------------------------------

(defun embed (r)
 (list (car r) (embed-ops (cadr r) (reverse (cdddr r))) (caddr r) ))

(defun embed-ops (log-form oplist)
 (cond ((null oplist) log-form)
       (t (embed-ops
            (embed-op log-form (car oplist)) (cdr oplist))) ))

(defun embed-op (log-form op)
 (cond ((member (car op) '(q f)) (append1 op log-form))
       (t (add-coord (cadr op) log-form))) )


(defun add-label (op rlist)
 (mapc
   '(lambda (r)
     (dsubst (list op (cadr r)) (cadr r) r))
    rlist))

(defun embed-quant (l hq hv)
 (mapcar
   '(lambda (r)
     (append1 (cons (car r) (cons hv (cddr r)))
           (cond ((null (cdadr r)) (list 'q hq hv (caadr r)))
                 ((member (caadr r) '(c C)) (list 'q hq hv (cadr r)))
                       (t (list 'q hq hv (cadr r))) )))
   l))

(defun embed-subj (slist)
 (mapcar
   '(lambda (r)
     (cons (car r)
       (cons (list 'i (cadr r))
         (cddr r)))) slist))

(defun form-list (log-form index ops)
 (progn
  (cond ((null index) nil)
```

```
      (t (setq index (formsym 'r))))
   (cond ((null ops) (list (list (list 0 0) log-form index)))
         (t (list (list (list 0 0) log-form index ops)))) ))


; "Form-branches" is called by "add-coord" to form n copies
; of a phrase to be used as the branches of the applied
; coordinator.

(defun form-branches (phrase-list n)
 (cond ((equal n 0) phrase-list)
       (t (form-branches (cons (car phrase-list) phrase-list)
                         (sub1 n))) ))


; "Add-coord" first calls "form-branches" to create copies of
; the logical form to which it is applied. An iteration through
; these branches is then made in "branch-loop". For each branch,
; the set of operators which may have been wrongly applied to
; that branch are stored in "wrong-ops" and the wrong operators
; (if any) are removed by "trim-branch". Then all occurrences
; of the unscoped coordinated expression inside the branch are
; replaced with the appropriate expression (the nth expression
; for the nth branch). The finished branches are stored in
; "newbranches". When all branches have been processed, the
; coordinator is applied to the list of branches.

(defun add-coord (coord phrase)
 (prog (branches branch branch-num pattern oplist all-ops
        wrong-ops newbranches var)
   (setq branches (form-branches (list phrase)
                  (length (cdddr (get-pattern coord phrase))) ))
   (setq all-ops (get coord 'ops))
   (setq branch-num 0)
   branch-loop
    (cond ((null branches)
           (return
            (cond
             ((pred? (car newbranches))
              (progn
               (setq var (formsym 'y))
               (setq newbranches
                (mapcar '(lambda (pred) (list 'i var pred))
                        newbranches))
               (list 'l var
                     (append (list 'i (car newbranches) coord)
                             (cdr newbranches))) ))
             (t (append (list 'i (car newbranches) coord)
                        (cdr newbranches)) )) ))
          (t (setq branch (car branches) branches (cdr branches))))
    (setq branch-num (add1 branch-num))
    (setq wrong-ops (wrong-oplist all-ops (get coord branch-num) nil))
    (cond ((null wrong-ops) nil)
          (t (setq branch (trim-branch wrong-ops branch))))
    subst-loop
```

```
        (setq pattern (get-pattern coord branch))
        (cond ((null pattern)
               (progn (setq newbranches (append1 newbranches branch))
                      (go branch-loop)))
              (t (progn (setq branch
                              (subst (nth branch-num (cdr pattern)) pattern branch))
                        (go subst-loop)))) ))

  ; "Wrong-oplist" returns a list of the operators which are
  ; initially embedded inside the coordinator but not inside the
  ; current branch (the parameter "right-ops" contains a list of
  ; the operators originally present in the current branch).
  ; Coordinators are placed before quantifiers in "wrong-ops" to
  ; improve the efficiency of any subsequent branch-trimming.

 (defun wrong-oplist (all-ops right-ops wrong-ops)
  (cond ((null all-ops) wrong-ops)
        ((member (car all-ops) right-ops)
          (wrong-oplist (cdr all-ops) right-ops wrong-ops))
        (t (wrong-oplist (cdr all-ops) right-ops
            (cond ((coord? (car all-ops)) (cons (car all-ops) wrong-ops))
                  (t (append1 wrong-ops (car all-ops)))) )) ))

  ; "Trim-branch" iterates through the list of wrong operators in
  ; "op-loop" and for each operator removes all instances of it
  ; (if any) from the branch (in "pattern-loop"). For vacuous
  ; coordination, the first coordinand (any will do) is substituted
  ; for the coordinated expression.

 (defun trim-branch (wrong-ops branch)
  (prog (op pattern)
    op-loop
    (cond ((null wrong-ops) (return branch))
          (t (setq op (car wrong-ops) wrong-ops (cdr wrong-ops))))
    pattern-loop
     (setq pattern (get-pattern op branch))
     (cond ((null pattern) (go op-loop))
           ((quant? pattern)
             (setq branch (subst (car (cddddr pattern)) pattern branch)))
           (t (setq branch (subst (cadr pattern) pattern branch))))
     (go pattern-loop)))


  ; "Get-pattern" returns the first occurrence of the expression
  ; embedded inside the operator "op" in "phrase". The patterns
  ; (if any) are recognized by the prefix symbol (eg. "q") and
  ; the operator (eg. "some2"). If no pattern is present, nil is
  ; returned.

 (defun get-pattern (op phrase)
  (prog (pattern)
    (cond ((atom phrase) (return nil))
          ((equal (cadr phrase) op) (return phrase))
          ((and (greaterp (length phrase) 2)
                (equal (caddr phrase) op))
```

```
      (return phrase))
     (t (setq phrase (cdr phrase))))
phrase-loop
 (cond ((null phrase) (return nil)) (t nil))
 (setq pattern (get-pattern op (car phrase)))
 (cond ((null pattern) (progn (setq phrase (cdr phrase))
                              (go phrase-loop)))
       (t (return pattern))) ))
```

```
;-----------------------------------------------------------------------
;          PART 4: REDUNDANCY AND SCOPING WEIGHTS
;
;   The following functions are used to detect redundant
; readings and to compute scoping weights.
;-----------------------------------------------------------------------

(defun parallel? (op1 op2)
  (or (member op1 (get op2 'parallel))
      (member op2 (get op1 'parallel))))


(defun existential-op? (op)
  (or (equal (word op) 'or)
      (member (word op) $existential-quants)))

(defun universal-op? (op)
  (or (equal (word op) 'and)
      (member (word op) $universal-quants)))

(defun commutative? (op1 op2)
  (cond ((and (existential-op? op1) (existential-op? op2)) t)
        ((and (universal-op? op1) (universal-op? op2)) t)
        (t nil)))


; "Redundant?" determines whether a list of operators will
; create a redundant reading when applied to a phrase in
; left-to-right order. A reading is redundant if:
; - two adjacent quantifiers are commutative and the suffixes
;   are in the wrong-order
; - a quantifier is directly before a commututative coordinator*
; - two adjacent coordinators are commutative and the first one
;   neither embeds nor has a lower suffix than the second
;   one
; - two adjacent operators are parallel (that is, they occur in
;   separate branches of a coordinated expression) and are in the
; wrong order.

(defun redundant? (oplist)
  (prog (op1 op2)
    (cond ((lessp (length oplist) 2) (return nil))
          (t (setq op1 (cadar oplist) op2 (caadr oplist))))
    (cond ((member (word op1) $existential-quants)
           (cond ((and (member (word op2) $existential-quants)
                       (lessp (suffix op2) (suffix op1)))
                  (return t))
                 ((equal (word op2) 'or) (return t))
                 (t (return (redundant? (cdr oplist))))))
          ((member (word op1) $universal-quants)
           (cond ((and (member (word op2) $universal-quants)
                       (lessp (suffix op2) (suffix op1)))
                  (return t))
                 ((equal (word op2) 'and) (return t))
                 (t (return (redundant? (cdr oplist)))))))
```

```
        ((commutative? op1 op2)
          (cond ((and (not (embeds? op1 op2))
                      (member (word op2) '(or and))
                      (lessp (suffix op2) (suffix op1))) (return t))
                (t (return (redundant? (cdr oplist))))) )
        ((and (parallel? op1 op2)
              (lessp (suffix op2) (suffix op1))) (return t))
        (t (return (redundant? (cdr oplist)))) )))


(defun standard-ratio (word s-relation)
  (cond ((null (get word s-relation)) (get word 'ratio))
        (t (get word s-relation)) ))

; "First-op-factor" is called by "scoping-weight" to add the
; effect of the first operator to the scoping weight (when
; two unscoped operators are being compared).
; It increases the weight if the first operator is "a" and
; reduces the weight if it is "few" or "no".

(defun first-op-factor (first-op initial-weight)
  (cond ((equal first-op 'a)
         (min (ave initial-weight $optimal-weight)
              (times 2.0 initial-weight)))
        ((equal first-op 'few) (times initial-weight 0.8))
        ((equal first-op 'no) (times initial-weight 0.5))
        (t initial-weight)))

(defun embeds? (op1 op2)
  (cond ((member op2 (get op1 'ops)) t)
        (t (apply 'or (mapcar '(lambda (op)
                                 (embeds? op op2))
                              (get op1 'ops)))) ))

(defun coord? (op)
  (cond ((atom op) (member op '(and or)))
        (t (member (car op) '(c C))) ))

; "Coord-weight" uses some simplified heuristics to obtain a
; scoping weight for coordinators. If a coordinator or
; existential quantifier is embedded inside a coordinator
; a weight of 0.4 is returned; for a distributive quantifier
; a weight of 0.0. If a coordinator is preceeded by but not
; embedded by another coordinator, a weight of 0.3 is returned.

(defun coord-weight (op1 op2)
  (prog (word1 word2)
    (setq word1 (word op1) word2 (word op2))
    (return
      (cond
        ((embeds? (cadr op1) (cadr op2))
          (cond
            ((coord? word1)
              (cond ((equal word2 'or) 0.4)
                    ((member word2 $existential-quants) 0.4)
```

```
            (t 0.0)))
        (t 1.0)))
    ((embeds? (cadr op2) (cadr op1))
      (diff 1.0 (coord-weight op2 op1)))
    ((lessp (suffix op1) (suffix op2)) 0.3)
    (t 0.7))) ))

; "Scoping-weight" computes the scoping weight for a pair
; of operators. It is used for both horizontal and vertical
; scoping (being called by "combine" and "expand").
; For horizontal scoping, the parameter "srelation" will be
; nil and will be determined from the suffixes and categories
; of the two operators by calling "structural-relation".
; For vertical scoping, "srelation" will describe the type
; of vertical relation and only one operator will be present
; (the parameter op1=nil).

(defun scoping-weight (srelation op1 op2)
 (prog (word1 word2 standard initial-weight reversed?)
  (cond ((atom srelation) nil)
      ((equal (car srelation) 'if-then) nil)
      (t (setq srelation (car srelation))))
  (cond ((null op1)
          (cond ((equal (word op2) 'not) (return 0.0))
            ((member (word op2) $quant-advls) (return 0.0))
            ((member srelation '(quant-pp quant-vp quant-cl))
              (setq op1 $headop))
            (t nil)))
      ((or (coord? op1) (coord? op2))
        (return (diff 1.0 (coord-weight op2 op1))))
        ((lessp (suffix op2) (suffix op1))
        (progn (setq reversed? t)
            (setq $temp op1 op1 op2 op2 $temp)))
      (t nil))
  (setq word1 (word op1) word2 (word op2))
  (setq srelation
   (cond ((null srelation) (structural-relation op1 op2))
       ((atom srelation) srelation)
       ((equal (caddr srelation) 1)
         (cond ((cadr srelation) 'pre-if) (t 'post-if)))
       ((equal (caddr srelation) 2)
         (cond ((cadr srelation) 'post-then) (t 'pre-then)))
       (t 'post-then)))
  (cond ((equal word2 'not)
       (setq $temp word1 word1 word2 word2 $temp)) (t nil))
  (setq standard (get word2 'standard))
  (setq initial-weight
   (cond ((get srelation 'all) (get srelation 'all))
         ((null standard) (get srelation word2))
         (t (times (get srelation standard)
                 (standard-ratio word2 srelation))) ))
  (setq initial-weight
   (cond ((null word1) initial-weight)
         (t (first-op-factor word1 initial-weight)) ))
```

```
    (return
      (cond (reversed? (diff 1.0 initial-weight))
            (t initial-weight))) ))

(defun structural-relation (op1 op2)
  (prog (cat1 cat2 srelation)
    (setq cat1 (category op1) cat2 (category op2))
    (setq srelation (get cat1 cat2))
    (return
      (cond ((null srelation) (get cat1 'others))
            (t srelation))) ))
```

```
;-----------------------------------------------------------------------
;                    PART 5: HORIZONTAL SCOPING
;
;  The horizontal scoping of operators in two lists of
;  readings is performed by "combine" which calls on
;  "combine-readings" to scope each pair of readings.
;  The actual scoping is performed by "permute" or "simple-
;  permute" which generate the set of scoped readings
;  corresponding to different permutations of the two sets
;  of operators. Some additional functions are required to
;  handle the scoping of coordinators and of duplicated
;  operators.
;-----------------------------------------------------------------------

; "Outer" and "inner" retrieve the lists of operators which
; scope outside and inside, respectively, a duplicated
; operator in a given reading. The two lists are stored on
; the property list of the operator under the attribute Ri,
; where Ri is the label for the reading. "Outer" and "inner"
; will combine the lists for two readings if the parameters
; "r1" and "r2" are both non-null.

(defun outer (op r1 r2)
 (cond ((null r1)
        (cond ((null r2) nil)
              (t (car (get op r2)))))
       ((null r2) (car (get op r1)))
       (t (append (car (get op r1)) (car (get op r2))) )))

(defun inner (op r1 r2)
 (cond ((null r1)
        (cond ((null r2) nil)
              (t (cadr (get op r2)))))
       ((null r2) (cadr (get op r1)))
       (t (append (cadr (get op r1)) (cadr (get op r2))) )))

; "Create-r" is called by "combine-readings" to create a label
; for a new reading. The input parameters "r1" and "r2" are
; the labels (possibly nil) for the old readings which were
; combined into the new one. If either is non-null a new label
; is created and each duplicted operator in "oplist" is
; updated with its scope relations in the new reading.
; These relations include those inherited from the previous
; reading(s) and any new ones found by "update-new-r".
; The latter function determines the scope relations of
; any duplicated operators in "oplist" with other operators
; in the list and puts this information on the attribute
; "new-r", the newly created reading.

(defun update-new-r (new-r outer-ops oplist)
 (cond ((null oplist) nil)
       ((member (car oplist) Scopied-ops)
        (progn
```

```
        (put (car oplist) new-r
          (list (append outer-ops (outer (car oplist) new-r nil))
                (append (inner (car oplist) new-r nil) (cdr oplist))))
        (update-new-r new-r (cons (car oplist) outer-ops)
                      (cdr oplist))))
    (t (update-new-r new-r (cons (car oplist) outer-ops) (cdr oplist))) ))

(defun create-r (r1 r2 oplist)
 (prog (inner outer new-r)
  (cond ((or r1 r2 (intersect? $copied-ops
                               (mapcar 'cadr oplist)))
         (setq new-r (formsym 'r)))
        (t (return nil)))
  (mapc
   '(lambda (op)
     (cond ((member (cadr op) $copied-ops)
            (put (cadr op) new-r
              (list (outer (cadr op) r1 r2)
                    (inner (cadr op) r1 r2))))
           (t nil)))
    oplist)
  (update-new-r new-r nil (mapcar 'cadr oplist))
  (return new-r)))


; "Inner-ops" returns a list of the operators which scope
; inside the coordinator ("coord") in "oplist",
; and "outer-ops" those which scope outside it. Both functions
; are called by "combine-readings".

(defun inner-ops (coord oplist)
 (cond ((equal (car oplist) coord) (cdr oplist))
       (t (inner-ops coord (cdr oplist))) ))

(defun outer-ops (coord oplist)
 (cond ((equal (car oplist) coord) nil)
       (t (cons (car oplist) (outer-ops coord (cdr oplist))) )))


; "Valid" returns nil if the two readings labelled by "r1"
; and "r2" contain any inconsistent scope orderings of any
; duplicated operators (listed in $copied-ops).
; "Intersect" returns the intersection of two lists.

(defun valid? (r1 r2 copied-ops)
 (cond ((null copied-ops) t)
       ((or (intersect? (car (get (car copied-ops) r1))
                        (cadr (get (car copied-ops) r2)))
            (intersect? (car (get (car copied-ops) r2))
                        (cadr (get (car copied-ops) r1))))
        nil)
       (t (valid? r1 r2 (cdr copied-ops))) ))

(defun intersect? (list1 list2)
 (cond ((null list1) nil)
```

```
        ((member (car list1) list2) t)
        (t (intersect? (cdr list1) list2))))

(defun update-compared-ops (ops1 ops2)
  (cond ((null ops1) nil)
        ((member (cadar ops1) $copied-ops)
          (progn
            (add-compared-ops (cadar ops1) (append (cdr ops1) ops2))
            (update-compared-ops (cdr ops1) (cons (car ops1) ops2))))
        (t (update-compared-ops (cdr ops1) (cons (car ops1) ops2))) ))

(defun add-compared-ops (op oplist)
  (prog (compared)
    (setq compared (get op 'compared))
    op-loop
    (cond ((null oplist)
           (progn (put op 'compared compared) (return nil)))
          ((member (cadar oplist) compared) nil)
          (t (setq compared (cons (cadar oplist) compared))))
    (setq oplist (cdr oplist))
    (go op-loop)))

; "Add-weights" adds the scoping weights obtained by giving
; an operator ("op") wide scope over each of the operators
; in a list ("oplist"). If giving the operator wide scope
; results in an invalid reading (if duplicated operators are
; present), nil is returned. Otherwise the total scoping
; weight ("sum") is increased by "add-sums".

(defun add-sums (sum op1 op2)
  (list (add (car sum) (scoping-weight nil op1 op2))
        (add1 (cadr sum))))

(defun add-weights (op oplist sum)
  (cond ((null sum) nil)
        ((null oplist) sum)
        (t (add-weights op (cdr oplist)
             (cond ((commutative? op (car oplist)) sum)
                   ((greaterp $min-weight
                      (scoping-weight nil (car oplist) op)) nil)
                   ((and (atom coord) (not (null coord))
                         (member coord (list (cadr op) (cadar oplist))))
                    (cond
                      ((and (member (cadr op) $copied-ops)
                            (equal (cadar oplist) coord))
                       (cond
                         ((member (cadr op) $old-comp) sum)
                         (t (progn (setq $new-comp (cons (cadr op) $new-comp))
                                   (add-sums sum (car oplist) op)))))
                      ((and (member (cadar oplist) $copied-ops)
                            (equal (cadr op) coord))
                       (cond
                         ((member (cadar oplist) $old-comp) sum)
                         (t (progn (setq $new-comp (cons (cadar oplist) $new-comp))
```

```
                    (add-sums sum (car oplist) op)))) )
            (t (add-sums sum (car oplist) op))))
        ((member (cadr op) Scopied-ops)
          (cond
            ((member (cadar oplist) (outer (cadr op) Sr1 Sr2)) nil)
            ((member (cadar oplist) (get (cadr op) 'compared)) sum)
            (t (add-sums sum (car oplist) op)) ))
        ((member (cadar oplist) Scopied-ops)
          (cond
            ((member (cadr op) (inner (cadar oplist) Sr1 Sr2)) nil)
            ((member (cadr op) (get (cadar oplist) 'compared)) sum)
            (t (add-sums sum (car oplist) op)) ))
        (t (add-sums sum (car oplist) op)) )))

(defun simple-permute (l0 l1 l2)
  (cond ((null l1) (list (append l0 l2)))
        ((null l2) (list (append l0 l1)))
        (t (append
             (simple-permute (append1 l0 (car l1)) (cdr l1) l2)
             (simple-permute (append1 l0 (car l2)) l1 (cdr l2)))) ))

(defun permute (l0 l1 l2 sum)
  (cond ((null sum) nil)
        ((null l1) (list (cons sum (append l0 l2))))
        ((null l2) (list (cons sum (append l0 l1))))
        (t (append
             (permute (append1 l0 (car l1)) (cdr l1) l2
                      (add-weights (car l1) l2 sum))
             (permute (append1 l0 (car l2)) l1 (cdr l2)
                      (add-weights (car l2) l1 sum)) )) ))

; "Combine-readings" scopes operators in a pair of readings ("leftr" and
; "rightr"). First, the labels for the readings ("r1" and "r2") are checked
; for compatability by "valid?"; if they contain an inconsistent scoping of
; the same pair of operators, the function returns nil (the labels will only
; be present if the readings contain duplicated operators).
; The "format" parameter is used to specify how the two logical forms are to
; be combined. If the format is "subj" or "obj" a check is first made for
; reflexives; if present, the correct "pattern" is substituted for the
; reflexive. The "coord" parameter may be nil, "t" or the coordinator
; (prefixed with "c"). The latter two cases only occur when "combine" is
; called by "scope-coord".
; Coordinators are scoped with each coordinated expression before the
; operators in different expressions are combined. Therefore, in general
; the list of operators in a reading in "rl-list" (see "scope-coord")
; contains operators scoping both outside and inside the coordinator.
; These are retrieved by "outer-ops" and "inner-ops", respectively.
; If a duplicated operator is present in both "leftops" and "rightops",
; it is removed at this point from the latter. The scoping itself is
; performed by "permute" which generates a set of readings containing the
; different permutations of the operators (both "leftops" and "rightops"
; are already internally ordered).
; If "coord" is not an atom (meaning that the two readings from "rl-list"
; are being combined) then "simple-permute" can be called instead. This
; simplification is possible because it is assumed that only existential
```

```
; quantifiers can widen scope over a coordinator and therefore no new
; scoping weights need to be calculated for the operators in the "outer-ops"
; and the operators in the two "inner-ops" are embedded inside different
; branches of the same coordinator and therefore will not be scoped with
; each other.
; "Update-compared-ops" is called to update the scope relations of any
; present duplicated operators. Finally, the list of readings ("newlist")
; is returned. If "coord" is an atom, the readings must be updated with
; the new scoping weights (a list of the sum and number of comparisons)
; and with a new label if needed (formed by "create-r").

(defun combine-readings (leftr rightr format coord)
 (prog (leftops rightops log-form oldsum oldnum newlist $r1 $r2 $coord-ops
       lf1 lf2 pattern)
  (setq $r1 (caddr leftr) $r2 (caddr rightr))
  (cond ((or (null $r1) (null $r2)) nil)
      ((not (valid? $r1 $r2 $copied-ops)) (return nil))
      (t nil))
  (setq lf1 (cadr leftr) lf2 (cadr rightr))
  (setq leftops (cdddr leftr) rightops (cdddr rightr))
  (cond ((member format '(subj obj))
        (progn
        (setq pattern
          (cond ((atom (car (last lf1)))
                 (get (cadar (last leftr)) 'pattern))
                ((equal (caar (last lf1)) 'c)
                  (get (cadar (last lf1)) 'pattern))
                (t nil)))
        (cond ((null pattern) nil)
              (t (setq lf2 (subst (car (last lf1)) pattern lf2)))))
      (t nil))
  (setq log-form
   (caseq format
     ('func  (list 'f (cadr leftr) (cadr rightr)))
     ('append  (append (cadr leftr) (cadr rightr)))
     ('list    (list (cadr leftr) (cadr rightr)))
       ('append1   (append1 (cadr leftr) (cadr rightr)))
     ('left  (cadr leftr))
     ('right  (cadr rightr))
     ('subj  (append1 (cadr leftr)
               (cond ((null pattern) (cadr rightr))
                     (t (subst (cadadr leftr) pattern (cadr rightr))))))
     ('obj  (append1 (cadr leftr) (cadr rightr)))
     (t nil)))
  (setq leftops (cdddr leftr) rightops (cdddr rightr))
  (cond ((atom coord) nil)
      (t (setq rightops
          (append
           (remove-ops (outer-ops coord leftops)
                   (outer-ops coord rightops))
           (list coord)
           (remove-ops (inner-ops coord leftops)
                   (inner-ops coord rightops)))) ))
  (setq oldsum (add (caar leftr) (caar rightr))
        oldnum (add (cadar leftr) (cadar rightr)))
```

```
  (setq newlist
   (delete nil
    (cond ((atom coord) (permute nil leftops rightops (list 0 0)))
          (t (mapcar
              '(lambda (left-ops)
                   (cons (list oldsum oldnum)
                    (cons log-form
                     (cons (create-r Sr1 Sr2 left-ops)
                      (append left-ops (list coord)
                       (append (inner-ops coord leftops)
                               (inner-ops coord rightops))) ))))
             (simple-permute nil (outer-ops coord leftops)
                             (outer-ops coord rightops)) )) )))
   (update-compared-ops
    (cond ((not (atom coord)) (cdddar newlist)) (t (cdar newlist))) nil)
   (return
    (cond ((not (atom coord)) newlist)
          (t (mapcar
              '(lambda (r)
               (cons (list (add (caar r) oldsum) (add (cadar r) oldnum))
                (cons log-form
                 (cons (create-r Sr1 Sr2 (cdr r))
                  (cdr r)))))
           newlist)))) ))

(defun combine (leftlist rightlist format coord?)
 (delete nil
 (apply 'append
  (mapcar
   '(lambda (leftr)
       (apply 'append
        (mapcar
         '(lambda (rightr)
             (combine-readings leftr rightr format coord?))
       rightlist))) leftlist))))
```

```
;-----------------------------------------------------------------------
;        PART 6: VERTICAL SCOPING

; Function "expand" may be called following the scoping of
; any phrase to scope the list of readings ("rlist") relative
; to the embedding operator (the information needed is present
; in the parameter "construct"). Each reading is vertically
; scoped by "raise-ops". This may call on four other functions,
; described below.
;-----------------------------------------------------------------------

; "Base-weight" computes the combined scoping weight of scoping
; all the operators in a list ("ops") inside an embedding
; operator (the vertical relation is given by "construct").
; This gives the weight associated with the "base" reading,
; in which all the operators are trapped. The individual
; scoping weights are determined by "scoping-weight" and if
; any weight is below "$min-value", then "invalid?" is set to
; true. The base reading will not be included in this case.
; (The scoping of duplicated operators with equivalent
; predicates is partially implemented).

(defun base-weight (construct ops)
 (prog (sum num)
  (setq sum 0 num 0)
  (mapc
   '(lambda (op)
     (prog (weight)
      (cond ((member (cadr op) $copied-ops)
             (cond
              ((intersect? $equiv-preds (cadr (get (cadr op) $old-r)))
               (progn (setq $invalid? t) (return nil)))
              ((intersect? $equiv-preds (car (get (cadr op) $old-r)))
               (return nil))
              (t nil))) (t nil))
      (setq weight
       (diff 1.0 (scoping-weight construct nil op)))
      (cond ((greaterp $min-weight weight) (setq $invalid? t))
            (t nil))
      (setq sum (add sum weight) num (add1 num))
      (return nil)))
    ops)
  (return (list sum num))))

; The next three functions are called by "raise-ops".
; "Form-r" forms a reading by calling "embed-ops" to apply
; the trapped operators to the logical form (the "raised"
; operators will be carried up).
; ("Create-rd" will be used to handle the scoping of
; operators with equivalent predicates).
; "Increase" computes the increase in the total scoping weight
; which is obtained by raising an operator (which is trapped
; in the base reading).

(defun create-rd (r preds trapped-ops raised-ops)
```

```
 nil)

(defun form-r (newval log-form raised-ops trapped-ops)
 (append
  (list newval (embed-ops log-form (reverse trapped-ops))
      (create-rd Sold-r Sequiv-preds trapped-ops raised-ops))
    raised-ops))

(defun increase (weight)
 (diff (times 2 weight) 1.0))

; "Raise-ops" performs the vertical scoping for a reading.
; First the base reading in which all operators are trapped
; is formed (by "form-r") and its scoping weight is
; computed by "base-weight". This list of vertically scoped
; readings is stored in "rlist"; a reading is omitted if it
; contains an individual scoping weight below "Smin-value" or
; if it contains a redundant ordering of trapped operators.
; The operators are raised in left-to-right order in "op-loop".
; If a redundant reading is detected the next operator is
; raised; if a scoping weight below Smin-weight is found,
; partially-formed list of readings is returned.

(defun raise-ops (construct oldval log-form Sold-r trapped-ops)
 (prog (newsum total Sequiv-preds raised-ops bval op Sinvalid?
       rlist weight)
  (setq Sequiv-preds (get (cadr construct) 'equiv-preds))
  (setq bval (base-weight construct trapped-ops)
      newsum (add (car oldval) (car bval))
      total (add (cadr oldval) (cadr bval)))
  (setq rlist
   (cond ((or Sinvalid?
           (redundant? (append1 trapped-ops (outer-op log-form))))
       nil)
      (t (list (form-r (list newsum total) log-form nil trapped-ops)))))
 op-loop
  (cond ((null trapped-ops) (return rlist))
      (t (setq op (car trapped-ops) trapped-ops (cdr trapped-ops))))
  (cond ((redundant? trapped-ops) (go op-loop))
      ((and (not (null Sequiv-preds))
          (intersect? Sequiv-preds (car (get (cadr op) Sold-r))))
       (return rlist))
      ((member (cadr construct) (get (cadr op) 'compared)) nil)
      (t (progn
          (setq weight (scoping-weight construct nil op))
          (cond ((greaterp Smin-weight weight) (return rlist))
              (t (setq newsum (add newsum (increase weight))) ))) ))
  (setq raised-ops (append1 raised-ops op))
  (setq rlist
   (append1 rlist
    (form-r (list newsum total) log-form
        raised-ops trapped-ops)))
  (go op-loop)))

(defun expand (construct rlist)
```

```
(apply 'append
 (mapcar
     '(lambda (r)
        (cond ((null (cdddr r)) (list r))
              (t (raise-ops construct (car r) (cadr r) (caddr r) (cdddr r)))))
     rlist) ))
```

```
;-------------------------------------------------------------------------
;    PART 7: DETERMINATION OF STRUCTURAL CATEGORIES
;
;  The structural category of each unscoped operator is stored
; on the property list under the attribute "category". This
; category is used to determine structural relations and may
; change as the operator moves upward during vertical scoping
; (eg, a quantifier in the subject position of an adverbial
; verb phrase will be given the category "func" after it
; widens scope outside the verb phrase). Such changes are made
; by calling "update-category" after an embedded phrase has
; been scoped.
; "Classify-terms" is called by "scope-prefix" and is used to
; determine detailed categories such as "topic", "surface
; subject", "indir-obj" or "prep-obj". This is done by
; comparing suffixes with position in the logical form.
;-------------------------------------------------------------------------

(defun category (op)
 (get (cadr op) 'category))

(defun put-category (op category)
 (cond ((atom op) nil)
       ((atom (cadr op))
            (put (cadr op) 'category category)) (t nil)))

(defun classify-terms (terms subj-suffix pred-suffix)
 (prog (term1 term2 count)
  (cond ((null terms) (return nil))
        (t (setq term1 (car terms) terms (cdr terms))))
  (cond ((null terms) nil)
        (t (setq term2 (car terms) terms (cdr terms))))
  (setq count 1)
  (cond ((atom term1) (setq $category (append1 $category t)))
        ((lessp (suffix term1) pred-suffix)
          (cond ((lessp subj-suffix pred-suffix)
                 (setq $category (append1 $category 'topic)))
                ((null term2)
                 (setq $category (append1 $category 'surface-subj)))
                   ((lessp (suffix term2) pred-suffix)
                    (setq $category (append1 $category 'topic)))
                --(t (setq $category (append1 $category 'surface-subj))) ))
       (t (setq $category (append1 $category 'dir-obj))))
  (setq count 2)
  (cond ((null term2) (return nil))
       ((atom term2) (setq $category (append1 $category t)))
          ((lessp (suffix term2) pred-suffix)
            (cond ((or (lessp subj-suffix pred-suffix)
                       (lessp (suffix term1) pred-suffix))
                   (setq $category (append1 $category 'topic)))
                  (t (setq $category (append1 $category 'surface-subj))) ))
       (t (setq $category (append1 $category 'indir-obj))))
  prep-obj-loop
   (setq count (add1 count))
   (cond ((null terms) (return nil))
```

```
        (t (progn
             (cond ((lessp (suffix (car terms)) pred-suffix)
                    (setq $category (append1 $category 'topic)))
                   (t (setq $category (append1 $category 'prep-obj)))))
             (setq terms (cdr terms))
             (go prep-obj-loop)))))

(defun update-category (newlist category)
 (mapc
  '(lambda (r)
    (mapc
       '(lambda (op)
          (put-category op category))
       (cdr r)))
  newlist))
```

```
;-------------------------------------------------------------------------
;                PART 8: SCOPING OF TERMS
;
;  The following functions handle the scoping of non-atomic
; terms. "Scope-term" determines the type of the term by
; checking the label. For quantifiers (label = "q") a variable
; is obtained from "getvar" and "scope-quant" is called.
; Different procedure calls are made for coordinated terms,
; generics and terms formed using "tau1" or "tau2".
; "Scope-quant" scopes the restriction (unless the restriction
; is a nominal-predicate such as "man") and embeds this inside
; the head quantifier and variable.
; "Getvar" returns a variable of quantification. At present,
; a new variable is created (using "newsym") unless there is
; already a lambda variable present or the quantifier already
; has been assigned a variable (in the case of duplicated
; quantifiers - this is stored on the attribute "var").
; For terms with coordinated restrictions, an iteration is
; made through the restrictions to find a lambda expression.
; (This needs to be improved).
;-------------------------------------------------------------------------

(defun newvar (det var)
 (put det 'var
  (cond ((null var) (formsym 'y))
        (t var))))

(defun getvar (term)
 (prog (pred-list pred)
  (cond ((member (car term) '(TAU1 TAU2)) (return nil))
        ((equal (car term) 'c) (return nil))
        ((not (null (get (cadr term) 'var))) (return (get (cadr term) 'var)))
        ((atom (caddr term)) (return (newvar (cadr term) nil)))
        ((equal (caaddr term) 'l)
         (return (newvar (cadr term) (cadr (caddr term)))))
        (t (setq pred-list (cddr (caddr term))) ))
  coord-loop
   (cond ((null pred-list) (return (newvar (cadr term) nil)))
         (t (setq pred (car pred-list)
                  pred-list (cdr pred-list))))
   (cond ((atom pred) (go coord-loop))
         ((equal (car pred) 'l) (return (newvar (cadr term) (cadr pred))))
         ((member (car pred) '(TAU1 TAU2)) (go coord-loop))
         (t (progn (setq pred-list (append pred-list (cddr pred)))
                   (go coord-loop))) )))


(defun term? (expr)
 (not (phrase? expr)))

(defun phrase? (expr)
 (cond ((atom expr) nil)
       ((member (car expr) '(q TAU1 TAU2)) nil)
       ((equal (car expr) 'c) (phrase? (caddr expr)))
```

```
      (t t)))

(defun pred? (phrase)
 (cond ((member (car phrase) '(l p)) t)
       ((equal (car phrase) 'f) (pred? (caddr phrase)))
       (t nil)))

(defun quant? (term)
 (cond ((atom term) t)
       ((equal (car term) 'q) t) (t nil)))


(defun scope-term (construct term)
 (cond ((equal (car term) 'q)
        (scope-quant (getvar term) (cdr term)))
       ((equal (car term) 'c) (scope-coord construct nil term))
       ((equal (car term) 'MU)
        (cond ((atom (cadr term))
               (form-list
                (cond ((equal (car construct) 'subject) (list 'i term))
                      (t term))
                nil nil))
              (t (add-label 'MU (scope-lambda 'top nil
                                  (cadadr term) (caddadr term)))) ))
       ((equal (car term) 'TAU1)
        (add-label 'TAU1
         (scope-phrase (list (get (car construct) 'cl) 'z)
                       nil (cadr term))))
       (t (add-label 'TAU2
           (scope-phrase (list (get (car construct) 'vp) 'z)
                         nil (cadr term)))) ))


(defun scope-quant ($headvar quant)
 (prog ($headop restr phrase rlist newlist)
  (setq $headop (car quant) restr (cadr quant))
  (cond ((atom restr)
         (return (form-list $headvar nil
                            (list 'q $headop $headvar (list 'i $headvar restr)))))
        (t (return
            (embed-quant
             (cond ((equal (car restr) 'c)
                    (scope-coord (list 'restr) nil restr))
                   (t (scope-phrase (list 'restr) nil restr)))
             $headop $headvar))) )))
```

```
;-----------------------------------------------------------------
;                PART 9: SCOPING OF PHRASES
;
;   This section contains functions for scoping the
; different types of phrase and for all types of coordinated
; expression. The main function is "scope-phrase" which
; determines the type of phrase and calls the appropriate
; scoping function. The list of readings for each phrase is
; usually stored in "rlist". Vertical scoping is preformed
; as the last step in each function; depending on the type
; of vertical relation (the parameter "construct") and on the
; type of phrase, the list of readings may be returned as it
; stands or be passed to "expand". A special case is made for
; coordinated expressions inside noun complements.
;-----------------------------------------------------------------


; "Scope-prefix" scopes prefix phrases (corresponding to verb
; phrases and subject-less prepositional phrases). First, the
; structural categories of the subject (if there is one) and
; the terms in the prefix phrase are determined by "classify-
; terms". The subject is required as a parameter for this.
; Next, "rlist" is initialized with the predicate, which must
; be scoped if it is coordinated. Next, the terms are scoped
; in turn and combined with "rlist" in the "term-loop". Terms
; which are atoms or reflexives (if the term has previously
; been encountered) are simply added to "rlist", otherwise
; they are passed to "scope-term" and the position of the
; term is stored on the attribute "reflex" so that later
; occurences of the term can be recognized as reflexives.
;
(defun scope-prefix (construct subj  prefix)
 (prog (pred term terms rlist newlist pred-suffix
       subj-suffix count $category)
  (setq pred (car prefix) terms (cdr prefix))
  (setq pred-suffix (suffix pred) subj-suffix (suffix subj))
  (cond ((null subj) nil)
        ((lessp subj-suffix pred-suffix)
          (setq $subj-cat 'full-subj))
        (t (setq $subj-cat 'prep-obj)))
  (classify-terms terms subj-suffix pred-suffix)
  (setq rlist
   (cond ((equal (car construct) 'quant-pp) (form-list (list pred) nil nil))
         ((atom pred) (form-list (list 'p pred) nil nil))
         (t (combine (form-list '(p) nil nil)
                     (scope-coord (list 'func) nil pred) 'append1 nil))))
  (update-category rlist 'pred)
  (setq count 0)
 term-loop
  (setq count (add1 count))

  (cond ((null terms)
         (cond ((member (car construct) '(infix func quant-pp restr top))
                (return rlist))
```

```
            (t (return (expand construct rlist)))))
          (t (progn
              (setq term (car terms) terms (cdr terms))
             (cond ((atom term)
                    (setq newlist (form-list term nil nil)))
                   ((and (member (car term) '(e q))
                         (not (null (get (cadr term) 'reflex)))
                         (lessp (get (cadr term) 'reflex) count))
                    (setq newlist (form-list term nil nil)))
                   (t (progn
                    (cond ((member (car term) '(e q))
                           (progn
                            (put (cadr term) 'pattern term)
                            (cond ((null (get (cadr term) 'reflex))
                                   (put (cadr term) 'reflex count))
                                  (t nil)) ))
                          (t nil))
                           (setq newlist (scope-term (list 'object) term))
                    (update-category newlist
                                     (nth (sub1 count) $category)))))
             (setq rlist (combine rlist newlist 'obj nil))
             (go term-loop))) ) ))
; "Scope-infix" scopes all infix expressions except connective
; clauses. A test is made for nominal-predicative clauses such
; as "[x man]" which need not be scoped. The term corresponding
; to the subject is scoped first; if it is not an atom, the
; position and pattern are stored to enable later reflexives
; to be detected. Next, if the infix expression is a noun
; complement (construct = "restr") the type of expression is
; determined (PP, VP or clause) and "construct" is updated.
; (The test for a VP complement is that is is untensed).
; The scoped subject ("rlist") and predicate ("predlist") are
; scoped horizontally (using "combine").

(defun scope-infix (construct infix)
 (prog (subj pred rlist predlist $subj-cat $untensed? constr)
  (setq subj (car infix) pred (cadr infix))
  (cond ((and (atom pred) (equal (length infix) 2))
         (return (form-list (list 'i subj pred) nil nil)))
        ((atom subj)
         (setq rlist (form-list (list 'i subj) nil nil)))
        (t (progn
         (cond ((member (car subj) '(e q))
                (progn
                 (put (cadr subj) 'reflex 0)
                 (put (cadr subj) 'pattern subj)))
               (t nil))
          (setq rlist
                (embed-subj
                 (scope-term (list 'subject) subj))) )))
  (setq constr (copy construct))
  (cond ((atom pred)
         (setq constr (list 'quant-pp 'z)
               predlist (scope-prefix constr subj (cdr infix))))
        (t (setq predlist (scope-phrase (list 'infix) subj pred))))
```

```
    (cond ((not (atom subj)) (update-category rlist $subj-cat)) (t nil))
    (setq $untensed? (untensed? pred))
    (cond ((equal (car constr) 'restr)
           (cond ($untensed?
                  (rplaca constr (get (car constr) 'vp)))
                 (t (rplaca constr (get (car constr) 'cl)))) )
          (t nil))
    (setq rlist (combine rlist predlist
                 (cond ((atom pred) 'append) (t 'subj)) nil))
    '(cond ((member (car constr) '(func top)) (return rlist))
           ((and (atom pred) (equal (word pred) 'element-of)) (return rlist))
           (t (return (expand constr rlist)))) ))

;
; "Scope-func" separately scopes the function (into "flist")
; and the phrase to which the function is applied ("rlist")
; and then scopes these horizontally (using "combine").
; Phrases serving as functions are prefixed with the "alpha"
; operator and are passed to "scope-phrase"; constant
; functions are not scoped but quantificational adverbs are
; treated as unscoped operators and are later scoped with
; "rlist". The negation operator is treated as unscoped
; function (to simplify the algorithm). The category
; of the function may depend on whether or not the function
; is preposed or postposed (determined by "postposed?").

(defun scope-func (construct subj func phrase)
 (prog (fvar flist rlist func-op)
   (setq func-op (list 'f func))
   (setq flist
    (cond ((equal (word func) 'not) (form-list nil nil func-op))
          ((member (word func) $quant-advls)
           (form-list nil nil func-op))
          ((atom func) (form-list func nil nil))
          ((member (car func) '(ALPHA1 ALPHA2 ALPHA3))
           (add-label (car func)
            (scope-phrase (list (car func)) nil (cadr func))))
          (t (form-list func nil nil))))
   (cond ((equal (word func) 'not) (put func 'category 'neg))
         ((member (word func) $quant-advls) (put func 'category 'func))
         (t (update-category flist
             (cond ((preposed? func phrase) 'pre-advl)
                   (t 'post-advl))) ))
   (setq rlist (scope-phrase (list 'func) subj phrase))
   (setq rlist (combine flist rlist
                (cond ((or (equal (word func) 'not)
                           (member (word func) $quant-advls)) 'right)
                      (t 'func))
                nil))
   (cond ((member (car construct) '(func infix top)) (return rlist))
         (t (return (expand construct rlist)))) ))


; "Scope-quant" is designed to scope expressions which have
```

```
; have scoped quantifiers applied to them in the initial
; logical translations. This function cannot be used at the
; present since it is not clear how the structural category
; of the quantifier should be determined.

(defun scope-quant-phrase (construct subj quant-phrase)
 (prog ($headop var restr phrase qlist rlist)
  (setq $headop (car quant-phrase) var (cadr quant-phrase)
      restr (caddr quant-phrase) phrase (cadddr quant-phrase))
  (setq qlist (form-list nil nil (list 'q $headop var restr)))
  (setq rlist (scope-phrase (list 'func) subj phrase))
  (setq rlist (combine qlist rlist 'right nil))
  (return
   (cond ((member (car construct) '(top func infix))
          (expand (list 'q $headop) rlist))
         (t (expand construct rlist)))) ))

(defun scope-lambda-phrase (construct subj var phrase)
 (mapcar
  '(lambda (r)
   (cond ((equal (car construct) 'restr) r)
         (t (cons (car r)
                  (cons (list 'l var (cadr r)) (cddr r)))) ))
  (scope-phrase construct subj phrase)))

; "Scope-conn-phrase" scopes connective infix expressions.
; The connected phrases are scoped in turn in "phrase-loop"
; and are combined using the horizontal relation "conn" or
; "restr" (if the connective clause is a restriction clause).
; It is necessary to determine whether the first two clauses
; are shifted for the scoping heuristics.

(defun scope-conn-phrase (construct conn-phrase)
 (prog (phrase conn rlist newlist count preposed?)
  (setq phrase (car conn-phrase) conn (cadr conn-phrase)
      conn-phrase (cddr conn-phrase))
  (setq preposed? (preposed? phrase (car conn-phrase)))
  (setq rlist (form-list '(i) nil nil))
  (setq count 1)
  phrase-loop
  (setq count (add1 count))
   (setq newlist
    (scope-phrase
     (cond ((equal (car construct) 'restr) construct)
           (t (list (word conn) preposed? (sub1 count))))
     nil phrase))
   (update-category newlist 'conn)
   (setq rlist (combine rlist newlist 'append1 nil))
   (cond ((equal count 2)
          (setq rlist
           (combine rlist (form-list conn nil nil) 'append1 nil))) (t nil))
   (cond ((null conn-phrase)
          (cond ((member (car construct) '(func restr top))
                 (return rlist))
                (t (return (expand construct rlist)))))
```

```
        (t (progn (setq phrase (car conn-phrase)
                        conn-phrase (cdr conn-phrase))
                  (go phrase-loop)))) ))

; "Update-coord-list" is used to scope coordinated expressions
; inside noun complements relative to the head quantifier.
; It is called by "scope-coord" (below) after the coordinated
; expression has been scoped.

(defun update-coord-list (coord-op rlist outer-val)
 (prog (inner-val outer-r)
  (setq inner-val (diff $optimal-weight outer-val))
  (return
   (apply 'append
    (mapcar
     '(lambda (r)
       (progn
        (setq outer-r
         (cons (list (add (caar r) outer-val) (add1 (cadar r)))
          (cdr r)))
        (cond ((equal coord-op (car (last r)))
               (list outer-r
                (cons (list (add (caar r) inner-val)
                            (add1 (cadar r)))
                 (cons (add-coord (cadr coord-op) (cadr r))
                  (cons (caddr r)
                   (remove-op coord-op (cdddr r) nil)))))))
              (t (list outer-r))) ))
     rlist))) ))

; "Get-ops" returns a list of the unscoped operators in an
; expression. It is called by "scope-coord" (to obtain the
; operators in each branch of the expression) and by "main"
; (to scan for duplicated operators).
; "Update-coord-ops" is called by "scope-coord".
;
(defun get-ops (expr all?)
 (cond ((atom expr) nil)
       ((equal (car expr) 'c)
          (progn
           (setq $oplist (cons (cadr expr) $oplist))
           (cond (all?
                  (mapc '(lambda (expr) (get-ops expr t)) (cddr expr)))
                 (t nil)) ))
       ((member (car expr) '(q n t))
         (progn (setq $oplist (cons (cadr expr) $oplist))
                (get-ops (cddr expr) all?)))
       (t (progn (get-ops (car expr) all?) (get-ops (cdr expr) all?))) ))


(defun get-copied-ops (copies oplist)
 (cond ((null oplist) copies)
       ((member (car oplist) (cdr oplist))
         (get-copied-ops (cons (car oplist) copies)
                         (remove-op (car oplist) (cdr oplist) nil)))
```

```
        (t (get-copied-ops copies (cdr oplist))) ))

(defun update-coord-ops (coord oplist)
  (prog (upper-coord branch-num)
    (setq upper-coord (get coord 'coord))
    (cond ((null upper-coord) (return nil))
          (t (setq branch-num (get coord 'branch))))
    (put upper-coord 'ops
      (append (get upper-coord 'ops) oplist))
    (put upper-coord branch-num
      (append (get upper-coord branch-num) oplist))
    (update-coord-ops upper-coord oplist)
    (return nil)))

(defun outer-op (log-form)
  (cond ((atom log-form) nil)
        ((member (car log-form) '(q C))
         (list (car log-form) (cadr log-form)))
        (t nil)))
; "Scope-coord" scopes all types of coordinated expression.
; Each expression is first scoped separately and the
; resulting lists of readings stored in "rl-list". The
; scoping of each expression involves some overhead: the
; list of unscoped operators is obtained by "get-ops"; the
; branch number ("expr-num") of each unscoped operator is
; recorded and the list of operators on each branch of the
; coordinator is stored on the its property list (this
; information is needed for the branch-trimming function).
; "Parallel" operators (those occurring in different branches
; of the same coordinator) are tagged as such, so that they
; will not be scoped relative to one another (ie. they are
; treated as commutative). The lists of readings are then
; scoped horizontally in "combine-loop". If the coordinated
; expression is inside a noun complement, a simplified
; vertical scoping is preformed by "update-coord-list".
;
(defun scope-coord (construct subj coord-expr)
  (prog (coord rlist rl-list $oplist coord-op expr-num $new-comp $old-comp)
    (setq coord (cadr coord-expr) coord-op (list 'c coord))
    (setq expr-num 0)
    (setq rl-list
      (mapcar
        '(lambda (expr)
          (progn
            (setq expr-num (add1 expr-num))
            (setq $old-comp (append $old-comp $new-comp)
                  $new-comp nil)
            (setq $oplist nil)
            (get-ops expr nil)
            (update-coord-ops coord $oplist)
            (put coord expr-num $oplist)
            (mapc '(lambda (op) (put op 'coord coord)) $oplist)
            (mapc '(lambda (op) (put op 'branch expr-num)) $oplist)
            (mapc '(lambda (op)
                     (put op 'parallel
```

```
                (append (get op 'parallel)
                 (remove-ops $oplist (get coord 'ops)))))
          $oplist)
         (put coord 'ops (append (get coord 'ops) $oplist))
        (combine
         (form-list nil nil coord-op)
         (cond ((atom expr)
                (cond ((equal (car construct) 'restr)
                       (form-list (list 'i $headvar expr) nil nil))
                      (t (form-list expr nil nil))))
               ((member (car construct) '(subject object))
                (scope-term construct expr))
               ((equal (car construct) 'restr)
                (scope-phrase (list 'restr) nil expr))
               (t (scope-phrase (list 'func) subj expr)))
         'right (cadr coord-op))))
       (cddr coord-expr)))
   (setq rlist
    (combine (form-list (list 'c coord) nil nil)
             (car rl-list) 'append1 nil)
    rl-list (cdr rl-list))
   combine-loop
    (setq rlist (combine rlist (car rl-list) 'append1 coord-op)
          rl-list (cdr rl-list))
    (cond ((null rl-list) nil) (t (go combine-loop)))
   (return
    (cond ((equal (car construct) 'restr)
           (update-coord-list coord-op rlist (get 'restr (word coord))))
          ((member (car construct) '(top func infix subject
                                     object restr)) rlist)
          (t (expand construct rlist)))) ))

; "Scope-phrase" checks the prefix symbol to determine the
; type of phrase and calls the appropriate scoping function.
; A special test is required for connective phrases which
; are scoped separately from other infix phrases.

(defun scope-phrase (construct subj phrase)
   (caseq (car phrase)
     ('f (scope-func construct subj (cadr phrase) (caddr phrase)))
     ('q (scope-quant-phrase construct subj (cdr phrase)))
     ('l (scope-lambda-phrase construct subj (cadr phrase) (caddr phrase)))
     ('c (scope-coord construct subj phrase))
     ('i (cond ((and (atom (caddr phrase)) (phrase? (cadr phrase)))
                (scope-conn-phrase construct (cdr phrase)))
               (t (scope-infix construct (cdr phrase)))) )
     (t (scope-prefix construct subj (cdr phrase))) ))
```

```
;-----------------------------------------------------------------------
;                PART 10: MAIN SECTION
;
;    The program reads in the scoping heuristics from three
; files and loops through the input formulas. For each formula
; words with suffixes are first updated using "store-words"
; and duplicated operators are stored in "$copied-ops".
; Each formula is passed to "scope-phrase" which returns a
; list of scoped readings; the list is sorted and printed
; after any remaining operators are applied to the logical
; forms.
;-----------------------------------------------------------------------
(defun main ()
 (prog ()
  (store-weights)
  (store-relations)
  (store-ratios)
  (store-digits)
  (setq inport (infile 'in) outport (outfile 'out))
  (setq formula (read inport))
  (setq formula-num 0)
  read-loop
   (setq $time (ptime))
   (setq $n 0)
   (setq formula-num (add1 formula-num))
   (patom '|Sentence | outport)
   (print formula-num outport) (terpri outport)
   (patom '|__________| outport) (terpri outport) (terpri outport)
   (pp-form formula outport) (terpri outport)
   (store-words formula)
   (setq $oplist nil)
   (get-ops formula t)
   (setq $copied-ops (get-copied-ops nil $oplist))
   (mapc 'printr
    (sort nil
     (delete nil
      (mapcar
       '(lambda (r)
         (cond ((redundant?
                 (append1 (cdddr r) (outer-op (cadr r)))) nil)
               (t (embed r))))
       (scope-phrase (list 'top) nil formula)))))
   (setq $time-used (diff (car (ptime)) (car $time)))
   (terpri outport)
   (patom '|time used = | outport)
   (print $time-used outport)
   (patom '| msecs.| outport) (terpri outport)
   (patom '|| outport)
   (print-line 70 outport) (terpri outport)
   (mapc '(lambda (op) (remob op)) $oplist)
   (setq formula (read inport))
   (cond ((null formula) (return nil)) (t (go read-loop))) ))

(main)
```